Ordonnancement semi-online sur machine unitaire pour l'industrie du futur

Hajar Nouinou

Thèse
de doctorat
de l'UTT

Hajar NOUINOU

Ordonnancement *semi-online* sur machine unitaire pour l'industrie du futur

Champ disciplinaire :
Sciences pour l'Ingénieur

2021TROY0028 **Année 2021**

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

THESE

pour l'obtention du grade de

DOCTEUR
de l'UNIVERSITE DE TECHNOLOGIE DE TROYES

en SCIENCES POUR L'INGENIEUR

Spécialité : OPTIMISATION ET SURETE DES SYSTEMES

présentée et soutenue par

Hajar NOUINOU

le 27 septembre 2021

Ordonnancement *semi-online* sur machine unitaire pour l'industrie du futur

JURY

M. Benoît IUNG	PROFESSEUR DES UNIVERSITES	Président
M. Stéphane DAUZÈRE-PÉRÈS	PROFESSEUR ECOLE DES MINES ST-ETIENNE - HDR	Rapporteur
Mme Nathalie SAUER	PROFESSEURE DES UNIVERSITES	Rapporteure
Mme Gülgün ALPAN-GAUJAL	PROFESSEURE DES UNIVERSITES	Examinatrice
M. Farouk YALAOUI	PROFESSEUR DES UNIVERSITES	Examinateur
M. Taha ARBAOUI	ENSEIGNANT CHERCHEUR UTT	Directeur de thèse
Mme Alice YALAOUI	MAITRE DE CONFERENCES - HDR	Directrice de thèse

À la mémoire de mon père

Remerciements

J'adresse ma vif gratitude à mes directeurs de thèse Madame Alice YALAOUI et Monsieur Taha ARBAOUI pour la confiance qu'ils m'ont accordé en acceptant d'encadrer mes travaux, pour m'avoir encouragé, conseillé et soutenu avec patience tout au long de cette thèse. Je suis reconnaissante pour leur disponibilité et la qualité de leur supervision.

J'exprime toute ma gratitude aux membres du jury qui me font l'honneur d'examiner cette thèse : Madame Nathalie SAUER, professeur des universités à l'université de Lorraine et Monsieur Stéphane DAUZÈRE-PÉRÈS, professeur à l'école des mines de Saint-Étienne en leur qualité de rapporteurs. Madame Gülgün ALPAN-GAUJAL, professeur des universités à l'université Grenoble Alpes, Monsieur Farouk YALAOUI professeur des universités à l'université de technologie de Troyes et Monsieur Benoît IUNG, professeur des universités à l'université de Lorraine en leur qualité d'examinateurs.

Je tiens à remercier également les membres du Laboratoire Informatique et Société Numérique (LIST3N). Je remercie Véronique BANSE et Bernadette ANDRE pour l'excellent service et la gentillesse au secrétariat du pôle ROSAS ainsi qu'aux membres de l'École doctorale, et particulièrement Pascale DENIS et Isabelle LECLERCQ pour l'amabilité et l'aide prêtée dans les procédures administratives.

Enfin, je ne saurais comment remercier mes parents Noureddine NOUINOU et Malika ZIANE pour leur soutien indéfectible et leur amour inconditionnel. Je remercie mes frères et ma soeur pour leurs soutiens et attentions. Une pensée particulière à mon mari Bilal pour sa patience, sa compréhension et l'énergie qu'il m'a donnée pour poursuivre mes études. Je remercie également mes beaux-parents pour leurs encouragements. C'est à ma famille que je dois tout mon succès.

Résumé

Nous étudions la valeur de l'information dans des problèmes d'ordonnancement *semi-online* sur machine unitaire. Nous proposons des algorithmes *semi-online* pour résoudre ces problèmes et nous évaluons leurs performances. Contrairement aux problèmes d'ordonnancement classiques *offline* où le décideur connaît toutes les caractéristiques de l'instance à ordonnancer, dans les problèmes d'ordonnancement *online* ou *semi-online* la prise de décision est effectuée sans aucune information ou uniquement avec des informations partielles sur l'instance. Notre travail consiste à distinguer les informations qui peuvent améliorer la prise de décision dans un contexte d'ordonnancement *semi-online* des informations qui, même disponibles, n'apportent aucune amélioration. Nous traitons principalement les problèmes *semi-online* dont la fonction objective est la minimisation de la somme des dates de fin des tâches sur machine unitaire. Nous commençons par étudier plusieurs modèles *semi-online* avec des informations partielles sur les temps de traitement des tâches. Ensuite, nous considérons le problème avec une information sur les dates d'arrivée et finalement la combinaison de l'information sur les temps de traitements et les dates d'arrivée des tâches. Pour chaque problème étudié où l'information partielle est identifiée comme utile, nous proposons un algorithme *semi-online* intégrant l'information dans la prise de décision. Ensuite, nous évaluons sa performance à l'aide d'une analyse de compétitivité ou par étude expérimentale comparative.

Abstract

We study the value of information in semi-online single machine scheduling problems. We propose semi-online algorithms to solve these problems and evaluate their performances. Unlike the classical offline scheduling problems where the decision maker knows all characteristics of the instance to be scheduled, in online scheduling problems the decision making is performed without previous information or only with partial information about the instance. Our work consists in distinguishing information that can improve the decision making in a semi-online scheduling context from information that, even if available, does not bring any improvement. We mainly deal with semi-online problems for minimising the total completion time on a single machine. We start by studying several semi-online models with partial information on processing times of jobs. Then, we consider the problem with information on jobs release dates and finally the combination of information on processing times and jobs release dates. For each studied problem where partial information is identified as useful, we propose a semi-online algorithm integrating the information into the decision making. We then evaluate its performance using a competitive analysis or a comparative experimental study.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Contexte générale

Au fil du temps et grâce au développement des technologies, trois révolutions sont apparues dans le monde industriel. La première révolution industrielle a utilisé l'énergie hydraulique et la vapeur pour mécaniser la production. La deuxième a utilisé l'énergie électrique pour créer une production de masse. La troisième a utilisé l'électronique et les technologies de l'information pour automatiser la production.
Aujourd'hui, la révolution numérique qui se produit depuis le milieu de la dernière décennie donne naissance à ce que l'on appelle « l'industrie 4.0 ». Elle se caractérise par une fusion des technologies qui fait disparaître les frontières entre les sphères physique, numérique et biologique. Selon une étude menée par Asadollahi-Yazdi et al. [8], le terme « Industrie 4.0 » qui est consacré à la quatrième révolution arrive sans le recul temporel suffisant pour la proclamer. Ainsi, cette révolution n'est qu'une évolution qui se dessine à travers le développement de plusieurs technologies et le besoin d'adaptation de plusieurs industries afin de continuer à exister dans ce monde industriel compétitif.

Du côté de l'offre, de nombreux secteurs voient l'introduction de nouvelles technologies qui créent des moyens nouveaux afin de répondre aux besoins existants et perturbent considérablement les chaînes de valeur industrielles existantes. Du côté de la demande, la transparence croissante, l'engagement des consommateurs et les nouveaux modèles de comportement des consommateurs (qui reposent de plus en plus sur l'accès à l'internet et aux données) obligent les entreprises à adapter la manière dont elles conçoivent, commercialisent et fournissent leurs produits et services. Pour une vision plus claire et complète sur les défis auxquels sont confrontées les industries dans la quatrième révolution industrielle, nous référons le lecteur vers un article récent de Panetto et al. [67].

La hiérarchie traditionnelle dans la gestion de la production qui a depuis longtemps été reconnue et acceptée dans la pratique est celle constituée de trois niveaux de décision (stratégique, tactique et opérationnelle) à laquelle le niveau de décision en temps réel vient s'ajouter. Les décisions prises dans un niveau deviennent des contraintes à satisfaire dans les niveaux inférieurs et chaque niveau de décision possède ses propres modèles de décision et approches de résolution. Dans la Planification Hiérarchique de la Production (HPP), l'approche hiérarchique courante est celle qui se compose de deux niveaux : la planification et l'ordonnancement. Le niveau de planification (ou de *lot sizing*) détermine les flux de produits, tandis que le niveau d'ordonnancement détermine les séquences de tâches sur les machines [24].

L'ordonnancement est un processus de prise de décision qui est utilisé régulièrement dans de nombreuses industries de fabrication et de services. Dans le contexte actuel, un système d'ordonnancement et de séquencement efficace est devenu une nécessité pour continuer à exister dans le marché concurrentiel actuel. Les entreprises doivent répondre avec efficacité et réactivité aux besoins du marché, de sorte que les délais d'expédition soient respectés tout en assurant la qualité des prestations fournies. De même, les entreprises doivent ordonnancer les tâches tout en optimisant l'utilisation des ressources disponibles.

L'ordonnancement consiste à organiser dans le temps la réalisation des tâches, compte tenu des contraintes temporelles et des contraintes de disponibilité des ressources requises. Son but est d'optimiser un ou plusieurs objectifs afin de maximiser l'efficacité de l'opération et réduire les coûts.

Les ressources et les tâches peuvent prendre plusieurs formes dans une organisation. Les ressources peuvent être les employés dans un site de construction, les machines dans un atelier, les serveurs de bases de données dans un réseau, les pistes dans un aéroport, CPU dans un ordinateur, etc. Les tâches peuvent être les étapes dans un projet de construction, les opérations dans un processus de production, les données dans un réseau, les décollages et les atterrissages dans un aéroport, les exécutions des programmes informatiques, etc. Les tâches peuvent avoir un ou plusieurs paramètres, comme le temps de traitement requis, la date d'arrivée, la date d'échéance souhaitée, un certain niveau de priorité, etc. Les fonctions objectifs peuvent également prendre plusieurs formes. Pour les problèmes de minimisation par exemple, nous identifions la date de fin moyenne des tâches, la date de fin de la dernière tâche ordonnancée, appelée *makespan*, le retard total ou moyen, etc.

Contrairement aux problèmes d'ordonnancement classiques « *offline* », où l'on connaît toutes les informations relatives à une instance donnée, dans les problèmes d'ordonnan-

cement « *online* » nous n'avons aucune information sur le nombre de tâches qui vont arriver et quand elles vont arriver. De même, toutes les caractéristiques des tâches à ordonnancer comme les temps de traitement et les dates d'échéance sont indisponibles au décideur. Les premiers algorithmes *online* se trouvent dans la thèse de doctorat [45] et dans l'article du journal [46] de Johnson. Johnson suggère que l'origine des mots « *offline* » et « *online* » se trouve dans les systèmes de cryptographie, dans lesquels le décryptage était effectué dans le cadre du système de communication (communication en ligne) ou après le fait en utilisant d'autres moyens (communication hors ligne) [29].

Entre ces deux modèles d'ordonnancement *online* et *offline*, nous y trouvons l'ordonnancement *semi-online* qui permet une relaxation de la contrainte *online* où aucune information sur la séquence des tâches n'est connue à l'avance. Un algorithme *semi-online* ressemble plus à un algorithme *online* du fait que les décisions concernant les tâches disponibles doivent être immédiates et irrévocables. Cependant, l'algorithme *semi-online* a l'avantage de connaître une ou plusieurs informations partielles sur la séquence des tâches à ordonnancer. En effet, dans la plupart des scénarios pratiques actuels, toutes les données ne sont pas disponibles dès le départ. De plus, les données ne se présentent pas forcément en *online*, mais peuvent apparaître avec des informations supplémentaires. Par exemple, dans le monde réel, le scénario *semi-online* suivant peut se produire dans la gestion des données distribuées [10] : les systèmes distribués et parallèles sont souvent confrontés au stockage de fichiers de taille variable sur des serveurs à capacité limitée. Il est évident que les fichiers sont transmis de manière aléatoire à partir d'une source connue et que chaque fichier reçu doit être attribué immédiatement à l'un des serveurs. L'ordonnanceur central du système est limité par les transmissions successives avant de procéder à une affectation irrévocable. Cependant, on sait par exemple que la source de transmission a stocké les fichiers dans k serveurs de capacité unitaire, ce qui donne une idée de la taille totale des fichiers à recevoir. Le défi consiste à stocker les fichiers sur les serveurs avec une exigence de mémoire minimale.

Plusieurs informations partielles ont été considérées dans la littérature, comme la somme des temps de traitement des tâches énoncée dans l'exemple précédent, le temps de traitement maximal ou minimal ou même la combinaison de plusieurs informations. Plus des informations sont combinées, plus on s'approche du modèle *offline* classique. Dans ce contexte, le défi consiste à identifier les informations partielles qui peuvent apporter un avantage en améliorant la prise de décision.

La majorité des recherches en ordonnancement *semi-online* ont considéré les problèmes dont la fonction objective est la minimisation du *makespan*, en particulier sur

l'environnement à machines parallèles, tandis que la recherche relative à des fonctions objectives comme la somme des dates de fin des tâches n'est rien qu'à ses prémices.

C'est dans cette perspective que nous nous intéressons, dans cette thèse, à l'étude des problèmes d'ordonnancement *semi-online* de minimisation de la somme des dates de fin des tâches sur machine unitaire. L'environnement à machine unitaire constitue un cas particulier de tous les autres environnements et les résultats obtenus constituent une base pour aborder des environnements plus complexes.

Dans cette thèse, nous traitons plusieurs problèmes d'ordonnancement *semi-online* en étudiant l'influence de plusieurs informations partielles sur la prise de décision. Ensuite, nous développons des algorithmes *semi-online* qui vont exploiter l'information ajoutée et nous analysons leurs performances. En raison de l'absence de l'information globale sur l'instance, lorsque l'on cherche à résoudre un problème d'ordonnancement *online* ou *semi-online*, on ne peut pas évaluer les performances d'un algorithme par rapport à ses capacités d'obtention de la solution optimale. La qualité d'un algorithme *online*/*semi-online* est généralement évaluée soit par la performance moyenne, soit par la performance dans le pire des cas. Dans les deux cas, l'évaluation est faite en comparant la solution obtenue avec la solution optimale en *offline*. Pour analyser la performance dans les pires des cas, la méthode qui a connue le plus de popularité est la méthode d'analyse de compétitivité. En identifiant la pire instance que peut rencontrer un algorithme *online* ou *semi-online*, l'analyse de compétitivité apporte une garantie de performance quelque soit la séquence des tâches à ordonnancer.

En ce qui concerne l'analyse de performance moyenne, nous avons opté pour l'étude expérimentale qui permet d'évaluer la performance d'un algorithme *semi-online* dans la pratique en le comparant à d'autres algorithmes existants. En particulier, cette comparaison est effectuée par rapport à l'algorithme *online* existant qui n'intègre aucune information partielle dans sa prise de décision.

Une question importante dans ce domaine est la suivante : quel est l'intérêt de connaître l'avenir ? Nous pouvons répondre à cette question en donnant des bornes inférieures générales sur le ratio de compétitivité. Une borne inférieure sur le ratio de compétitivité d'un problème implique qu'aucun algorithme *online* ou *semi-online* ne peut avoir un ratio de compétitivité inférieur à cette borne, et donne donc une indication de ce que l'on gagnera en ayant une information partielle à l'avance. Nous utilisons ensuite le principe de *information value* introduit par Tan et Zhang [85] qui consiste à comparer la borne inférieure générale sur le ratio de compétitivité du modèle *semi-online* à la borne inférieure du modèle *online*. Si l'ajout d'une information contribue

à diminuer la borne inférieure du modèle *online*, alors l'information est dite *valuable*. Sinon, elle est dite *valueless*.

Structure de la thèse

Dans le premier chapitre, nous commençons par présenter des notions de base nécessaires à la description du sujet d'étude de cette thèse ainsi qu'à la compréhension de la contribution scientifique des travaux réalisés. Ensuite, nous fournissons une description des méthodes d'évaluation de la performance des algorithmes *online* et *semi-online*. Finalement, nous présentons un état de l'art des problèmes *online* et *semi-online* identifiés dans la littérature en insistant sur les problèmes d'ordonnancement sur machine unitaire.

Dans le chapitre 2, nous étudions plusieurs informations sur les temps de traitement des tâches et leur influence sur la performance d'un algorithme *online*. Le problème étudié est la minimisation de la somme des dates de fin des tâches sur machine unitaire. Les tâches sont de type *over-time*. Nous étudions alors les bornes inférieures pour plusieurs problèmes *semi-online*, en présentant pour chaque cas l'exemple pour lequel aucun algorithme *semi-online* ne peut garantir une meilleure performance. Nous nous focalisons dans ce chapitre sur l'information sur les temps de traitement des tâches et leur influence sur la compétitivité d'un algorithme *semi-online*. À la fin de cette étude, nous présentons une classification des problèmes *semi-online* selon la notion de *information value* telle que définie par Tan et Zhang [85].

Dans le chapitre 3, nous étudions le problème *semi-online* de minimisation de la somme des dates de fin des tâches sur machine unitaire avec des tâches qui arrivent en ordre décroissant des temps de traitement. Nous présentons un nouvel algorithme, nommé ϕD-SPT, qui prend en compte cette information et qui assure une meilleure performance que l'algorithme classique *online*. Nous analysons le ratio de compétitivité de cet algorithme en utilisant la méthode de transformation d'instance introduite par Tao et al. [87]. Le ratio de compétitivité correspond à la borne inférieure du problème *semi-online*, ce qui veut dire que ϕD-SPT est le meilleur algorithme pour le problème.

Dans le chapitre 4, nous étudions dans un premier temps le modèle *semi-online* du problème de minimisation de la somme des dates de fin des tâches sur machine unitaire. Dans cette étude, nous considérons le cas où les dates d'arrivée des tâches sont connues au départ, alors que les temps de traitement demeurent inconnus. Nous

développons d'abord un algorithme *semi-online*, nommé VD-SPT (*Variable Delayed Shortest Processing Time*) qui obtient de très bons résultats sur des instances de petite taille selon notre étude expérimentale. Ensuite, nous présentons un algorithme nommé CSPTn pour le problème qui, en le comparant aux algorithmes *online* et *semi-online* existants, obtient de très bons résultats sur des instances de grande taille et plus précisément sur des instances présentant une grande variabilité en termes de temps de traitement. Finalement, nous présentons la borne inférieure sur le ratio de compétitivité qui permet de vérifier l'utilité de l'information partielle pour le problème étudié. Dans un second temps, nous étudions le problème de minimisation de la somme des dates de fin pondérées des tâches sur machine unitaire. Nous démontrons d'abord que l'information sur les dates d'arrivée des tâches n'apporte aucune amélioration. Ensuite, nous étudions la combinaison de l'information sur les dates d'arrivée potentielles et une borne sur le plus petit temps de traitement. Pour ce problème, nous développons un algorithme *semi-online*, nommé MCSWPT, et nous évaluons sa performance par une étude expérimentale comparative.

Enfin, une conclusion générale clôture ce manuscrit en résumant les différents travaux réalisés ainsi que des perspectives pour des travaux futurs.

Chapitre 1

État de l'art

Résumé :
Dans ce chapitre, nous présentons une revue de la littérature des problèmes d'ordonnancement qui peuvent être distingués en trois catégories : *offline* classique, *online* et *semi-online*. Dans les modèles d'ordonnancement *offline*, les caractéristiques des tâches sont supposées connues à l'avance lors de l'établissement de l'ordonnancement. Cependant, dans les problèmes d'ordonnancement *online*, ces caractéristiques ne sont pas disponibles et des algorithmes spécifiques qui prennent les décisions en temps réel doivent être développés. Entre ces deux catégories, le modèle *semi-online* offre une relaxation de la contrainte *online* en rendant connue une information partielle sur les caractéristiques des tâches à ordonnancer. Tout d'abord, nous présentons dans ce chapitre des notions de base nécessaires à la description du sujet d'étude de cette thèse ainsi qu'à la compréhension de la contribution scientifique des travaux réalisés. Ensuite, nous fournissons une description des méthodes d'évaluation de la performance des algorithmes *online* et *semi-online* et des approches de résolutions adaptées à ces différents problèmes. Finalement, nous présentons un état de l'art des problèmes *online* et *semi-online* identifiés dans la littérature en insistant sur les problèmes d'ordonnancement sur machine unitaire.

1.1 Problèmes d'ordonnancement

Au cours des soixante dernières années, un effort de recherche considérable a été consacré à l'ordonnancement déterministe. Le nombre et la variété des modèles envisagés est important. Les modèles dans le domaine de l'ordonnancement sur machines sont devenus très standardisés et une notation s'est développée permettant de décrire la structure de beaucoup de modèles déterministes qui ont été considérés dans la littérature.

1.1.1 Notations

Afin d'introduire les différents problèmes, considérons un système composé de m machines dans lequel nous souhaitons ordonnancer n tâches. L'indice j désigne une tâche tandis que l'indice i désigne une machine, ainsi J_j désigne une tâche et m_i une machine. Si une tâche nécessite un certain nombre d'opérations, l'étape de traitement ou l'opération de la tâche J_j sur la machine m_i est désignée par $J_{j,i}$. A chaque tâche J_j sont associées les données suivantes.

Release date **(r_j)** : désigne la date d'arrivée de la tâche dans le système et correspond à l'instant où la tâche J_j devient disponible pour le traitement. En d'autres mots, aucune tâche J_j ne peut commencer l'ordonnancement avant sa date d'arrivée r_j.

Processing time **($p_{i,j}$)** : représente le temps de traitement de la tâche J_j sur la machine m_i. L'indice i est supprimé si le temps de traitement de la tâche J_j ne dépend pas de la machine ou si la tâche J_j ne sera traitée que sur une seule machine. Le temps de traitement de la tâche J_j est alors noté p_j.

Weight **(w_j)** : désigne le poids de la tâche J_j. Ce poids reflète l'importance d'une tâche par rapport aux autres tâches de l'instance. A titre d'exemple, Il peut être traduit comme le coût de maintien ou d'inventaire de la tâche dans le système.

Due date **(d_j)** : désigne la date d'échéance de la tâche J_j et représente la date à laquelle la tâche est promise au client. La tâche peut être accomplie après la date d'échéance définie, mais un coût est encouru dans ce cas. Cependant, s'il n'y a pas de tolérance de dépassement de la date d'échéance, on parle alors de date limite.

Il convient de noter ici que les données mentionnées ci-dessus peuvent être soit de nature déterministe soit de nature stochastique.

1.1.2 Environnements machine

Un problème d'ordonnancement est décrit selon la notation $\alpha|\beta|\gamma$ [35]. Le champ α désigne les ressources ou l'environnement machine. β désigne les caractéristiques et les contraintes liées aux tâches. γ désigne le critère à optimiser.

Certains environnements de machines possibles spécifiés dans le champ α sont les suivants :

Machine unitaire (1) : l'environnement de la machine unitaire est très simple et constitue un cas particulier de tous les autres environnements. En outre, les résultats qui peuvent être obtenus pour les modèles de machine unitaire peuvent servir de base à des heuristiques applicables à des environnements de machines plus complexes. En effet, ceux-ci sont souvent décomposés en des sous-problèmes portant sur des machines unitaires.

Machines parallèles identiques (P_m) : il y a m machines identiques en parallèle. La tâche J_j nécessite une seule opération et peut être traitée sur n'importe laquelle des m machines. En effet, toutes les machines peuvent faire toutes les tâches et on dispose de plusieurs machines simplement pour être plus efficace.

Machines parallèles uniformes (Q_m) : il y a m machines parallèles avec des vitesses différentes. La vitesse de la machine m_i est généralement notée par v_i. Le temps p_{ij} que la tâche J_j passe sur la machine m_i est égale à $\frac{p_j}{v_i}$. Pour des vitesses identiques, l'environnement devient identique au précédent.

Machines parallèles non-reliées (R_m) : cet environnement est une généralisation supplémentaire du précédent. Les machines sont également en parallèle et chaque machine m_i peut traiter une tâche J_j à une vitesse v_{ij}. Le temps p_{ij} que la tâche J_j passe sur la machine m_i est égale à $\frac{p_j}{v_{ij}}$. Si les vitesses des machines sont indépendantes de la tâche, l'environnement devient identique au précédent.

***Flow shop* (F_m) :** il y a m machines en série. Chaque tâche doit être traitée sur chacune des m machines. Toutes les tâches doivent suivre le même chemin, c'est-à-dire qu'elles doivent être traitées d'abord sur la machine m_1, puis sur la machine m_2, et ainsi de suite. Une fois le traitement sur une machine terminé, une tâche rejoint la file d'attente de la machine suivante.

***Job shop* (J_m) :** contrairement à l'environnement précédent, dans un *job shop* à m machines, chaque tâche a sa propre route prédéterminée à suivre.

***Open shop* (O_m) :** il y a m machines. Chaque tâche doit être traitée à nouveau sur chacune des m machines. Cependant, certains de ces temps de traitement peuvent être nuls. Il n'existe aucune restriction quant au cheminement de chaque tâche dans l'environnement des machines. Le planificateur est autorisé à déterminer une route pour chaque tâche.

1.1.3 Contraintes de traitement

Les contraintes et restrictions de traitement spécifiées dans le champ β peuvent inclure des entrées multiples. Nous citons les entrées possibles dans le champ β les plus utilisées :

Dates d'arrivée (r_j) : si ce symbole apparaît dans le champ β, alors la tâche J_j ne peut pas commencer son traitement avant sa date d'arrivée r_j. En revanche, si ce symbole n'apparaît pas dans le champ β, cette tâche peut commencer son traitement à tout moment, sans restriction.

Dates d'échéances (d_j) : ce symbole signifie qu'une tâche J_j devrait se terminer avant une date limite d_j, appelée également *due date*. Le *due date* peut être dépassé mais son apparition dans le champ β signifie que tout dépassement sera traduit par un coût supplémentaire dans la fonction objective.

Préemptions ($prmp$) : ce symbole signifie que le planificateur est autorisé à interrompre le traitement d'une tâche à tout moment afin de planifier une autre tâche. Dans le cas où la tâche est préemptée, la partie traitée de la tâche n'est pas perdue, mais lorsqu'elle est traitée ultérieurement, la tâche préemptée reprend là où elle s'était arrêtée.

***Restarts* (res) :** ce symbole signifie que la tâche peut être interrompue à tout moment afin de planifier une autre tâche. Cependant, contrairement à la contrainte précédente, quand la tâche est planifiée après avoir été interrompue, elle doit recommencer son traitement à zéro.

Contraintes de précédence ($prec$) : les contraintes de précédence peuvent apparaître sur une seule machine ou dans un environnement à machines en parallèles,

exigeant qu'une ou plusieurs tâches soient terminées avant qu'une autre tâche soit autorisée à commencer son traitement.

Traitement par lots (*batch*(*b*)) : une machine peut être capable de traiter un certain nombre de tâches, disons b, simultanément ; c'est-à-dire qu'elle peut traiter un lot d'au plus b tâches en même temps. Les tâches en cours peuvent avoir des temps de traitement différents et un lot n'est terminé que lorsque la dernière tâche du lot est terminée.

Setup time **(s_{ji}) :** s_{ji} représente le temps de préparation dépendant de la séquence entre le traitement des tâches J_j et J_i.

Online **(*online*) :** cette contrainte signifie que la décision doit être prise sans connaître les informations sur les tâches (ou avec des informations partielles sur les tâches, dans ce cas la contrainte devient "*semi-online*"). Lorsque le symbole des dates d'arrivée r_j est également présent dans le champ β, alors les tâches arrivent en *over-time*. Alors que dans le cas contraire, les tâches arrivent une par une sous forme d'une liste. Nous fournissons plus de détails sur les contraintes *online* et *semi-online* dans les sections 1.2 et 1.3.

1.1.4 Critères à optimiser

L'objectif à minimiser est toujours une fonction des dates de fin des tâches, qui dépendent bien sûr du séquencemment. La date de fin (*completion time*) de la tâche J_j sur la machine m_i est noté C_{ij}. La date de fin de la tâche J_j dans le système est noté C_j.

L'objectif peut également être une fonction des dates d'échéance. Le retard algébrique (*lateness*) de la tâche J_j est noté :

$$L_j = C_j - d_j \tag{1.1}$$

Cette valeur peut être positive ou négative. Lorsque L_j est une valeur positive, alors la tâche J_j est terminée en retard, tandis que dans le cas où $L_j < 0$ la tâche est en avance. Le retard (*tardiness*) de la tâche J_j est défini comme suit :

$$T_j = \max(L_j, 0) \tag{1.2}$$

Contrairement au retard algébrique, la valeur de T_j n'est jamais négative. L'indicatrice

de retard de la tâche J_j est définie comme suit :

$$U_j = \begin{cases} 1 \ si \ C_j > d_j \\ 0 \ sinon \end{cases} \tag{1.3}$$

Des exemples de fonctions objectives possibles à minimiser sont :

***Makespan* (C_{max}) :** signifie la durée totale de l'ordonnancement et est définie comme $C_{max} = \max\{C_1, ..., C_n\}$. Elle représente la date de fin de la dernière tâche ayant quitté le système. Un *makespan* minimum implique généralement une bonne utilisation de la machine (maximisation de la productivité).

***Lateness* maximum (L_{max}) :** signifie le plus grand retard algébrique et est défini comme $L_{max} = \max\{L_1, ..., L_n\}$. Il mesure la pire infraction du *due date*.

La somme des dates de fin des tâches ($\sum C_j$) : cette fonction objectif donne une indication du coût total de possession ou d'inventaire engendré par l'ordonnancement. Une autre variante est la somme des dates de fin pondérées des tâches $\sum w_j C_j$ quand les tâches ont des poids différents.

Retard total ($\sum T_j$) : la minimisation de cette fonction objective permet la minimisation des encours. Quand les tâches ont des poids différents, nous parlons alors du retard total pondéré $\sum w_j T_j$.

Nombre de tâches en retard ($\sum U_j$) : le nombre de tâches en retard n'est pas seulement une mesure d'intérêt académique, c'est souvent un objectif dans la pratique car c'est une mesure qui peut être enregistrée très facilement. Quand les tâches ont des poids différents, nous parlons alors du nombre pondéré total des tâches en retard $\sum w_j U_j$.

Toutes les fonctions objectives ci-dessus sont des mesures de performance dites régulières, ce qui signifie qu'elles sont non décroissantes en fonction de $C_1, ..., C_n$. La liste des environnements machines, des contraintes et des critères que nous avons évoquer n'est pas exhaustive. Pour plus de détails nous référons le lecteur aux ouvrages [70, 16].

1.1.5 Théorie de la complexité

De nombreux problèmes d'ordonnancement sont de nature combinatoire : il s'agit de problèmes où l'on recherche l'optimum à partir d'un nombre très important mais fini de

solutions. Parfois, ces problèmes peuvent être résolus rapidement et efficacement, mais souvent les meilleures approches de résolution disponibles sont lentes et fastidieuses. Il devient donc important d'évaluer la performance d'une approche proposée.

La théorie de la complexité a été développée afin d'étudier la difficulté des problèmes et s'est avéré très utile pour l'optimisation combinatoire [47]. L'efficacité d'un algorithme pour un problème d'optimisation est mesurée par le nombre maximum d'étapes de calcul nécessaires pour obtenir une solution optimale en fonction de la taille de l'instance. En ordonnancement, la taille d'une instance est généralement désignée par le nombre de tâches dans l'instance. De plus, une étape de calcul peut être définie comme une comparaison, une multiplication ou toute autre étape de manipulation de données concernant une tâche. L'efficacité d'un algorithme est alors mesurée par le nombre maximum d'étapes de calcul nécessaires pour obtenir une solution optimale. Le nombre d'étapes de calcul peut souvent correspondre au nombre maximal d'itérations que l'algorithme doit effectuer. Ce nombre d'itérations est généralement approximé. Par exemple, un algorithme de $O(n^3)$ est généralement appelé un algorithme en temps polynomial : le nombre d'itérations est polynomial à la taille (n) du problème.

La classe de tous les problèmes qui peuvent être résolus en temps polynomial est appelée classe P. Une autre classe de problèmes d'optimisation est connue sous le nom de problèmes NP-*hard*, pour de tels problèmes, aucun algorithme en temps polynomial n'est connu et on pense généralement que ces problèmes ne peuvent être résolus en temps polynomial. La règle suivante est largement acceptée : si un problème est NP-*hard*, il est peu probable qu'il admette un algorithme en temps polynomial, et doit être traité par d'autres méthodes [70, 47].

Une fois que nous constatons qu'un problème est NP-*hard*, nous devons déterminer si nous voulons sa solution exacte ou si nous pouvons nous contenter d'une solution approximative. Une solution exacte peut être trouvée par diverses méthodes d'énumération réduite, généralement par un algorithme de *branch* & *bound*, des méthodes de programmation dynamique ou programmation linéaire [23] qui ont des limites en matière de taille du problème à cause du temps de calcul. Dans tous les cas, pour les problèmes d'intérêt pratique, seules les instances de petite taille peuvent être traitées par des méthodes exactes.
Afin de trouver une «bonne» solution dans un temps acceptable, nous pouvons utiliser des méthodes approchées comme les heuristiques qui utilisent des propriétés d'optimalité ou de dominance pour définir des règles de priorité.

1.2 Ordonnancement *online*

Dans la section précédente, les hypothèses de base reposaient sur le fait que toutes les données du problème (temps de traitement, dates d'arrivée, nombre de tâches, *due-dates*, poids, etc.) sont connues à l'avance. Le décideur peut déterminer à l'instant zéro l'ensemble de l'ordonnancement en disposant de toutes les informations. Ce paradigme le plus courant est généralement appelé "ordonnancement *offline*".

En pratique, souvent nous ne connaissons pas toutes les tâches à l'avance et une décision doit être prise en ne se basant que sur les informations disponibles à l'instant présent. Ce genre de problème est appelé un problème d'ordonnancement *online*. Le monde de la recherche s'intéresse de plus en plus aux modèles d'ordonnancement *online*. Ces modèles sont distingués selon la manière dont les informations sont révélées au décideur. Nous présentons dans la suite les deux principaux modèles : l'arrivée des tâches en *over-list* et en *over-time*.

L'arrivée des tâches en *over-list* : Dans ce modèle, les tâches arrivent une par une selon une liste. Dès qu'une tâche J_j est révélée, toutes ses caractéristiques deviennent disponibles. Par ailleurs, un algorithme *online* doit prendre une décision immédiate et irrévocable concernant cette tâche avant de voir la prochaine tâche dans la liste. Une décision consiste à affecter cette tâche à une machine disponible pendant un intervalle de temps tout en respectant les contraintes liées à la tâche, comme le *due date*. Ce modèle représente le modèle *online* standard pour les séquences des requêtes. Il est utilisé pour des problèmes *online* où la notion du temps n'est pas employée, comme le *bin packing*, coloration de graphes, *load balancing* et *paging* (gestion des ressources dans les systèmes d'exploitation) [2].

L'arrivée des tâches en *over-time* : Dans ce modèle, les tâches arrivent séparément dans le temps. En effet, à chaque tâche J_j est assignée une date d'arrivée r_j à laquelle elle devient disponible pour exécution. Contrairement au modèle précèdent, une tâche ne doit pas nécessairement être ordonnancée pour voir la prochaine tâche. L'algorithme peut prendre la décision d'exécuter immédiatement une tâche disponible comme il peut faire attendre la tâche pour vérifier si d'autres tâches vont être révélées dans le futur. Si l'algorithme *online* est non-clairvoyant, alors même quand une tâche devient disponible, le temps de traitement ne devient connu que lorsque la tâche a fini l'exécution. Cependant, pour un algorithme *online* clairvoyant, dés qu'une tâche devient disponible, son temps de traitement devient connu.

Les modèles *offline* et *online* peuvent être considérés comme les deux cas extrêmes des cas que nous pouvons rencontré dans la réalité, c'est-à-dire le modèle où nous connaissons tout sur le futur et le modèle où nous ne connaissons absolument rien sur le futur. Cependant, souvent une ou plusieurs informations sur les tâches à venir est disponible au préalable, et le décideur souhaiterait probablement savoir si cette information peut l'aider à prendre de meilleurs décisions. Le type de problèmes que l'on rencontre dans ce contexte est dit *semi-online*. Ainsi, les algorithmes *semi-online* sont des algorithmes qui fonctionnent en *online* tout en exploitant une information supplémentaire afin d'améliorer la prise de décision [4].

1.3 Ordonnancement *semi-online*

Le modèle *semi-online* est un cadre intermédiaire pour aborder la praticabilité et les limites des deux modèles *offline* et *online*. Plusieurs recherches ont étudié les scénarios où la contrainte *online* est « détendue » ou relaxée, c'est-à-dire que l'algorithme *online* est fourni avec quelques informations sur la séquence de travail : l'algorithme *semi-online* connaît par exemple la somme des temps d'exécution des tâches, la valeur optimale en *offline*, des informations sur les dates d'arrivée des tâches ou encore une stratégie *online* qui peut réorganiser les tâches. Le défi consiste donc dans ce cas à démontrer quelle information est utile pour quel problème. Voici un exemple de trois applications où le modèle *semi-online* est présent :

— Les commandes clients peuvent être imprévues. Ainsi, les décisions concernant les commandes de chaque client doivent être prises immédiatement en affectant les tâches aux ressources disponibles sans connaître les commandes futures. Cependant, pour une usine qui fabrique un certain nombre de types de produits connus, les temps de traitement maximal et minimal sont logiquement des informations disponibles à l'avance.

— Les processus de commande et de planification périodiques qui sont largement utilisés dans l'industrie. Les commandes sont acceptées, par exemple, chaque semaine. Entre deux commandes, aucune nouvelle tâche n'est disponible. Jusqu'à ce qu'une commande arrive, les détails de la commande, comme la durée de traitement, ne sont pas connus. En attendant, il est nécessaire de prendre des décisions de planification concernant les tâches actuellement disponibles, en tenant compte de la possibilité que de nouvelles commandes arrivent au début des semaines suivantes.

— Dans un système de serveurs parallèles, il existe généralement des estimations assez précises de la charge de travail qui arrive sur un horizon de temps donné. Dans un atelier, un planificateur accepte généralement les commandes (tâches) d'un volume déterminé pour une période de temps donnée, par exemple un jour ou une semaine. Ceci explique la motivation derrière la connaissance de l'information sur la somme des temps de traitement des tâches à ordonnancer.

Les algorithmes *semi-online* ont un fonctionnement similaire aux algorithmes *online*, c'est-à-dire que les décisions sont prises en se basant uniquement sur les informations disponibles à l'instant de décision. En effet, l'information partielle fait partie de ces informations disponibles pour l'algorithme *semi-online*. Logiquement, dans la construction d'un algorithme *semi-online*, nous devons exploiter l'information partielle de manière à améliorer la prise de décision. A défaut, l'algorithme serait *online* en principe. De plus, la manière dont cette information est utilisée ne devrait pas être aléatoire. La majorité des algorithmes *semi-online* existants ont exploité l'information partielle de manière à minimiser l'impacte de la pire instance dans l'environnement *semi-online* considéré [89, 42, 6].

1.4 Mesures de performance des algorithmes *online* et *semi-online*

Plusieurs méthodes d'évaluation de la performance des algorithmes *online* et *semi-online* peuvent être identifiées dans la littérature. Dans cette section, nous présentons une description des deux méthodes que nous avons utilisées et qui sont : l'étude de compétitivité et l'étude expérimentale. Les autres méthodes qui peuvent être rencontrées dans la littérature sont : l'augmentation des ressources et la compétitivité locale amortie.

1.4.1 Étude de la compétitivité

Un algorithme *online* ne peut généralement pas obtenir la solution optimale en raison de l'absence des informations sur les tâches. Les décisions prises par ce type d'algorithmes ne dépendent que des informations disponibles à l'instant de décision. De ce fait, nous cherchons des algorithmes *online* qui garantissent des solutions proches de l'optimal. La qualité d'un algorithme *online* est souvent évaluée en utilisant le ratio de

compétitivité. L'idée de la compétitivité est de comparer le résultat d'un algorithme *online* au résultat produit par un algorithme *offline* optimal. Un algorithme *offline* optimal connaît toutes les informations relatives à une instance en avance.

Modèle de l'adversaire

La mesure de la performance d'un algorithme *online* s'effectue en considérant que l'algorithme *online* joue contre un adversaire. L'adversaire construit une séquence de tâches qui doivent être ordonnancées par l'algorithme *online*. Il se charge de son côté d'ordonnancer ces tâches de manière optimale en exploitant toutes les informations relatives à cette instance (*offline*). L'adversaire tente donc de proposer une instance mettant en difficulté l'algorithme *online* en augmentant ses coûts (valeur de la fonction objectif) par rapport à ses propres coûts.

Différents types d'adversaires peuvent être identifiés [14]. Les différences concernent les informations sur l'algorithme *online* dont dispose l'adversaire et la manière dont il traite ses requêtes. Le premier type est généralement utilisé pour évaluer la performance des algorithmes déterministes, tandis que le deuxième et le troisième type sont spécifiques à des algorithmes *online* randomisés qui utilisent l'aléatoire dans leur prise de décision.

Adversaire *oblivious* : celui qui doit construire la séquence des tâches à l'avance, ainsi que la solution optimale correspondante, en se basant uniquement sur la description de l'algorithme *online* (c'est-à-dire, une fois la séquence des tâche est définie au départ, l'adversaire ne peut plus changer sa décision en fonction des mouvements de l'algorithme *online*).

Adversaire adaptatif *online* : un adversaire qui lance la prochaine requête en se basant sur les réponses de l'algorithme aux précédentes, mais qui la traite immédiatement.

Il est évident que pour les algorithmes déterministes, cet adversaire est équivalent à l'adversaire *oblivious*, puisque les réponses de l'algorithme sont complètement prévisibles. Pour comprendre à quel point la randomisation est utile contre lui, nous introduisons un adversaire encore plus fort (voir aussi Raghavan et Snir [73]).

Adversaire adaptatif *offline* : celui qui fait la prochaine séquence des tâches en fonction des réponses de l'algorithme *online* aux précédentes, mais les sert de manière

optimale à la fin. Comme on peut le supposer, cet adversaire est si fort que la randomisation n'apporte aucun avantage à l'algorithme *online*. Ceci est dû au fait que cet adversaire ne connaît pas seulement la description de l'algorithme en question, mais il connaît également son mouvement exacte et est donc capable de répondre avec une séquence des tâches qui va augmenter les coûts de cet algorithme. De plus, contrairement à l'adversaire adaptatif *online*, cet adversaire construit l'ordonnancement optimal à la fin, ce qui lui donne un grand avantage en testant plusieurs combinaison afin de trouver la meilleure.

Ratio de compétitivité

Le ratio de compétitivité d'un algorithme *online* est défini comme suit. Notons par $ALG(I)$ la solution obtenue par l'algorithme *online* A pour une instance donnée I. De plus, la solution obtenue par un algorithme *offline* optimal pour l'instance I est notée $OPT(I)$.

Définition 1. Un algorithme *online* A est dit ρ-compétitif si

$$\frac{ALG(I)}{OPT(I)} \leq \rho, \quad \forall I \tag{1.4}$$

En d'autres termes, si, pour chaque instance, la valeur objective (attendue) de l'ordonnancement produit par l'algorithme *online* est au plus égale à ρ fois la valeur de l'ordonnancement optimal, cet algorithme déterministe a un coefficient de compétitivité ρ (également appelé borne supérieure), ce qui correspond au coefficient de compétitivité avec un adversaire *oblivious*.

Borne inférieure sur le ratio de compétitivité

Pour un algorithme spécifique, nous pouvons donner une instance spéciale pour que l'algorithme ait la pire performance possible (c'est-à-dire le pire-cas). Ainsi, nous obtenons une borne inférieure (du ratio de compétitivité) de cet algorithme. De plus, si son coefficient de compétitivité atteint cette borne inférieure, cette borne est dite serrée.

En outre, Il ne faut pas commettre l'erreur de confondre la borne inférieure sur le ratio de compétitivité d'un algorithme *online* ou *semi-online* avec la borne inférieure générale du problème d'ordonnancement *online* ou *semi-online* considéré. Pour

un problème de minimisation, une borne inférieure générale sur le ratio de compétitivité implique qu'il n'existe aucun algorithme *online* avec un ratio de compétitivité inférieur à cette borne. Par conséquent, la borne inférieure générale du ratio de compétitivité doit être vérifiée par rapport au meilleur algorithme *online* possible du problème.

Définition 2. En notant I_1 la pire instance que peut rencontrer un algorithme *online*. b est une borne inférieure sur le ratio de compétitivité si :

$$\exists I_1, \forall ALG, \quad \frac{ALG(I_1)}{OPT(I_1)} \geq b \tag{1.5}$$

Cette borne inférieure indique que n'importe quel algorithme *online* (donc même le meilleur algorithme possible) ne peut avoir un ratio de compétitivité meilleur que b. Ainsi, si b est égal à ρ pour un algorithme A, l'algorithme est dit optimal.

La difficulté dans la démonstration du ratio de compétitivité réside dans deux aspects : le premier est dans la recherche de l'instance qui représente le pire cas dans l'espace des instances. Le deuxième est qu'une borne supérieure sur la valeur objective optimale doit être construite afin de calculer le ratio de compétitivité. Ces deux difficultés rendent la plupart des méthodes d'analyse de compétitivité dépendante du problème.

Nous avons choisi l'analyse de compétitivité comme l'une des méthodes d'analyse des algorithmes *online* et *semi-online*. En mettant l'accent sur l'analyse de compétitivité, nous ne faisons aucune déclaration sur sa praticabilité ultime, que ce soit en elle-même ou par rapport aux méthodes d'analyse du cas moyen comme l'étude numérique. Nous pensons que l'analyse de compétitivité est un cadre intéressant pour l'analyse mathématique des algorithmes. Plus important encore, l'analyse de compétitivité conduit à la dérivation et à l'étude d'algorithmes particuliers qui ne seraient pas naturellement envisagés en utilisant un modèle conceptuellement différent. La question est de savoir si les algorithmes suggérés par l'analyse de compétitivité peuvent être utilisés comme point de départ pour des algorithmes dont les performances sont réellement compétitives ou supérieures aux algorithmes basés sur d'autres approches. En outre, dans certaines applications telles que la planification financière, il peut y avoir des situations dans lesquelles des garanties de performance dans le pire des cas sont nécessaires ; dans ce cas, l'analyse de compétitivité est essentielle. Tout modèle conceptuel présente des inconvénients et des avantages. L'analyse de compétitivité a l'inconvénient d'être trop

pessimiste, en supposant un adversaire malveillant qui choisit la pire instance pour évaluer la performance d'un algorithme, ce qui constitue la faiblesse de toute analyse du pire cas.

1.4.2 Étude expérimentale

L'analyse de compétitivité a été le recours de plusieurs études qui ont visé à évaluer la performance d'un algorithme *online* [26]. Cependant, puisque l'analyse de compétitivité ne peut capturer que le comportement le plus mauvais d'un algorithme *online*, une question intéressante est de savoir si un algorithme *online* est uniquement orientée vers le pire cas pathologique ou si il conduit également à de meilleurs résultats dans la pratique ? C'est le but de l'étude expérimentale qui a connue un grand intérêt parmi les chercheurs [5, 90, 15].

Hall et Posner [38] discutent plusieurs schémas de génération de données largement utilisés, et démontrent que ces schémas ne sont souvent pas représentatifs de la façon dont les données sont générées dans des situations pratiques. Les auteurs remédient à ces déficiences en décrivant plusieurs principes de génération de données et plusieurs propriétés souhaitables dans un schéma de génération, ce qui leur permet de fournir des propositions spécifiques pour la génération des données pour une variété de problèmes d'ordonnancement. Hall et Posner [38] ont présenté des principes et des propriétés pour la génération des instances pour différents problèmes d'ordonnancement de manière à satisfaire certains critères qui assurent le cohérence des instances générées. Un principe de génération de données est un axiome ou une perspective à partir duquel on peut aborder la conception d'un schéma de génération. Les principes que nous décrivons sont généraux en ce sens qu'ils ne dépendent pas des objectifs ou des caractéristiques d'une expérience particulière. Voici quelques principes à prendre en compte lors de la génération des instances [38] :

1. **Objectif :** générer des données pour satisfaire les objectifs de l'expérience.
2. **Comparabilité :** rendre les tests informatiques comparables.
3. **Impartialité :** éviter d'introduire des biais involontaires dans les données.
4. **Reproductibilité :** rendre le schéma de génération reproductible.

De plus, les propriétés qui sont importantes pour l'évaluation numérique d'un algorithme incluent [38] :

1. **Variété :** crée un large échantillon d'instances du problème.
2. **Pertinence pratique :** génère des données qui modélisent des scénarios du monde réel.
3. **Invariance d'échelle :** lorsque l'échelle des données d'entrée change, les autres caractéristiques ne changent pas.
4. **Invariance de la taille :** lorsque la taille des données d'entrée change, les autres caractéristiques ne changent pas.
5. **Régularité :** des types d'entrée identiques sont traités de manière similaire.
6. **Descriptibilité :** facile à décrire.
7. **Efficacité :** facile et efficace à mettre en œuvre, à utiliser et à reproduire.
8. **Parcimonie :** on ne fait varier que les paramètres qui peuvent affecter l'analyse.

Le tableau (1.1) montre quels principes conduisent généralement à quelles propriétés. Un X indique qu'un principe spécifique induit une propriété spécifique si les caractéristiques du problème et de l'expérience le rendent applicable.

Table 1.1 – Les relations entre les principes et les propriétés [38]

	Principes			
Propriétés	Objectif	Comparabilité	Impartialité	Reproductibilité
Variété	X	X		
Pertinence pratique	X			
Invariance d'échelle		X	X	
Invariance de la taille		X	X	
Régularité			X	
Descriptibilité				X
Efficacité				X
Parcimonie				X

Albers et Schröder [5] ont mené une étude expérimentale pour comparer des algorithmes *online* pour le problème de minimisation du *Makespan* sur machines parallèles avec des tâches qui arrivent en *over-list*. Ils ont analysé les algorithmes existants sur

différentes séquences de tâches générées selon des lois de probabilités ainsi que des instances réelles. Ils concluent que la performance des algorithmes d'ordonnancement dépend des caractéristiques de l'instance. Pour les instances à faible variabilité, l'algorithme simple de Graham [35] a la meilleure performance. Cependant, pour les instances réelles à haute variabilité, les nouveaux algorithmes sont souvent plus performants que l'algorithme de Graham. Ces résultats montrent également l'importance de choisir les bonnes instances lors de l'évaluation expérimentale des algorithmes d'ordonnancement.

1.4.3 Autres méthodes d'évaluation de performance

Deux autres méthodes d'évaluation de performance des algorithmes *online* et *semi-online* rencontrées dans la littérature sont :

L'augmentation des ressources [69] : dans ce modèle, un algorithme *online* est augmenté de ressources supplémentaires sous la forme de processeurs plus rapides ou de processeurs supplémentaires. Soit A_s un algorithme *online* qui fonctionne avec des processeurs de vitesse s où $s \geq 1$. Nous appelons un algorithme *online* A_s, un algorithme ρ-compétitif à vitesse s, avec ρ le ratio de compétitivité, si $f(A_s, I) \leq \rho f(O_1, I)$ pour toutes les instances d'entrée I, avec O_1 l'algorithme *offline* optimal à vitesse 1. En effet, les résultats qui décrivent les performances d'un algorithme sur des machines plus rapides fournissent une indication au concepteur d'un système de combien les processeurs doivent être plus rapides pour garantir le niveau de performance souhaité.

La compétitivité locale amortie [12] : cette méthode consiste à utiliser des fonctions potentielles pour analyser les algorithmes d'ordonnancement *online*. Ces fonctions potentielles sont utilisées pour montrer qu'un algorithme *online* donné est localement compétitif dans un sens amorti. Les analyses d'algorithmes utilisant des fonctions potentielles sont parfois critiquées comme semblant relever de la magie noire, car les preuves formelles ne contiennent aucune discussion sur l'intuition qui est derrière la conception de la fonction potentielle. Cela peut s'expliquer par le fait que les auteurs trouvent généralement la fonction potentielle par des essais sans qu'une intuition ne guide le développement. Pour un tutoriel complet sur cette méthode, nous référons les lecteurs à un article de Im et al. [43] présentant une fonction potentielle "standard" qui semble être applicable à un large éventail de problèmes.

1.5 État de l'art des problèmes d'ordonnancement *semi-online*

Dans la pratique, la plupart des problèmes ne sont ni purement *offline* ni *online*, mais se situent quelque part entre les deux. Cette observation fait de l'ordonnancement *semi-online* un domaine de recherche actif. Le *semi-online* peut être considéré comme une relaxation du *online* ou un état intermédiaire entre le *offline* et le *online* [49]. Le principal intérêt de l'étude des problèmes l'ordonnancement *semi-online* réside dans le fait que, dans la vie réelle, un décideur peut disposer d'un certain type d'informations sur les tâches à venir. Cependant, dans les problèmes d'ordonnancement, certaines informations ne permettent pas de prendre de meilleures décisions. Par conséquent, la question la plus importante à laquelle l'étude de l'ordonnancement *semi-online* répond est la suivante : Quelle information est utile pour quel problème ? Tan et Zhang [85] ont fourni une définition des informations utiles et non-utiles. La valeur d'un modèle *semi-online*, désigné par V_s, est évaluée en comparant sa borne supérieure (ou inférieure) à celle d'un modèle *online* correspondant fonctionnant dans le même environnement machine et avec la même fonction objective, représenté par V_o. Le modèle *semi-online* est dit utile s'il existe un algorithme *semi-online* pour V_s dont le ratio de compétitivité est plus petit que la borne optimale (ou la borne inférieure) pour V_o. Par contre, si la borne inférieure de V_s est supérieure au ratio de compétitivité d'un algorithme pour V_o, il est dit non-utile.

La majorité des études en *semi-online* se sont focalisées sur des problèmes de minimisation du *makespan* avec des tâches qui arrivent en *over-list*. Les quatre types d'informations partielles suivantes sont les plus utilisées :

sum **:** La somme des temps de traitement des tâches $\sum_{j=1}^{n} p_j$ dans l'instance est connue au début de la prise de décision.
opt **:** Le makespan optimal offline de l'instance est connu avant l'arrivée de la première tâche.
max **:** Le temps de traitement maximal de toutes les tâches $p_{max} = \max_{j=1,\ldots,n} p_j$ est connu avant l'arrivée de la première tâche.

En effet, la connaissance de l'information sur la somme des temps de traitement "sum" ainsi que sur le temps de traitement maximal "p_{max}" est réaliste. Si nous considérons à titre d'exemple un usine qui fabrique un certain nombre de types de produits, le temps de traitement maximal qu'une tâche peut prendre est une information généra-

lement connue à l'avance. De plus, le planificateur accepte généralement les commandes avec un volume déterminé pour une période de temps donnée.
La motivation de *sum* et *max* est évidente, alors qu'il peut sembler étrange à première vue que le *makespan* optimal *offline* puisse être connu à l'avance. En fait, cela peut être interprété à partir du fait que les problèmes *online* où le *makespan* est connu à l'avance ont déjà été étudiés avant que le *semi-online* ne commence à attirer l'attention, puisque le lemme suivant est utile dans l'analyse de compétitivité de problèmes *online* difficiles.

Lemme 1. [9] Pour un problème d'ordonnancement du modèle *online over-list* dont l'objectif est de minimiser le *makespan*, s'il existe un algorithme pour la variante *opt* avec un ratio de compétitivité ρ, alors il existe un algorithme pour le modèle *online* pur avec un ratio de compétitivité au plus égal à 4ρ.

D'autre informations moins courantes qui peuvent être identifiées dans la littérature sont :

min **:** Le temps de traitement minimal de toutes les tâches $p_{min} = \min_{j=1,...,n} p_j$ est connu avant l'arrivée de la première tâche.
num **:** Le nombre totale des tâches n est connu avant l'arrivée de la première tâche.
UB **:** La borne supérieure sur les temps de traitement des tâches dans l'instance ($p_{UB} \geq p_j$ pour $j = \{1, ..., n\}$) est connue avant l'arrivée de la première tâche.
LB **:** La borne inférieure sur les temps de traitement des tâches dans l'instance ($p_{LB} \leq p_j$ pour $j = \{1, ..., n\}$) est connue avant l'arrivée de la première tâche.
incr **:** Les tâches arrivent dans l'ordre croissant de leurs temps de traitement.
decr **:** Les tâches arrivent dans l'ordre décroissant de leurs temps de traitement.
lookahead **:** Lors de l'affectation de la tâche en cours, les informations relatives aux k tâches suivantes sont connues. k étant un nombre constant.
buffer **:** il existe un *buffer* de taille K qui peut stocker au maximum K tâches. La tâche qui arrive peut soit être planifiée immédiatement, soit être stockée dans le *buffer*, ce qui lui permet d'être planifiée plus tard.
reassignement **:** Il est permis de réaffecter certaines des tâches pendant ou après le traitement.
$\boldsymbol{T_k}$ **:** Les tâches ne peuvent arriver qu'à des instants connus dans le temps notés T_k, avec $k = \{1, ..., m\}$ et m le nombre des dates d'arrivée potentielles.
deter **:** Les temps de traitement des tâches se détériorent dans le temps et sont exprimés en fonction d'un taux de détérioration et de la date de début de traitement des tâches
$\boldsymbol{\gamma}$ **:** Le ratio du plus grand temps de traitement sur le plus petit temps de traitement est borné par une valeur connue γ

La majorité des informations citées ci-dessus ont été considérées pour les problèmes de minimisation du *makespan*. Nous présentons dans le suite un état de l'art des problèmes *semi-online* existants catégorisés selon les fonctions objectives les plus étudiées. Afin d'identifier les travaux les plus pertinents, nous avons effectué une première sélection en se basent sur les mots-clés, ensuite nous avons analysé les titres puis les résumés des articles sélectionnés. Des exemples d'articles utilisant des techniques de sélection des travaux pertinents sont [23, 25, 82].

Minimisation du *makespan* :
La majorité des travaux dans le modèle *semi-online* se sont focalisés sur la minimisation du *makespan*. Considérons d'abord le problème $P_m|sum|C_{max}$. Kellerer et al. [49] ont étudié le problème à deux machines, $P_2|sum|C_{max}$. Ils ont présenté un algorithme *semi-online* pour le problème avec un ratio de compétitivité égale à 1.333. En effet, ils ont également prouvé qu'aucun algorithme *semi-online* ne peut garantir une meilleure performance pour ce problème. Pour $m = 3$, Lee et Lim [55] ont présenté un algorithme *semi-online* 1.4-compétitive, tandis que la borne inférieure est égale à 1.393 tel que prouvé par Angelellie et al. [7].
Pour une valeur de m arbitraire, la borne inférieure sur le ratio de compétitivité passe à 1.585 alors que la meilleure borne supérieure est de l'ordre de 1.6. En ce qui concerne le problème $P_m|max|C_{max}$, la variante à deux machines a été résolue par He et Zhang en 1999 [41] qui ont présenté un algorithme *semi-online* avec un ratio optimal égal à 1.333. Il convient de noter que pour la variante *max*, non seulement les temps de traitement de toutes les tâches ne dépassent pas p_{max}, mais également au moins une tâche dont le temps de traitement est est égale à p_{max} arrivera tôt ou tard.
Pour $m = 3$, Cai [78] a prouvé que la borne inférieure sur le ratio de compétitivité devient égale à 1.414. Wu et al. [95] ont présenté ensuite un algorithme *semi-online* de compétitivité égale à 1.5. En outre, pour un nombre de machines arbitraire la borne inférieure devient égale à 1.457. L'écart de cette borne par rapport à la borne supérieure reste considérable, avec 1.920 le ratio de compétitivité du meilleur algorithme *semi-online* présenté à ce jour [1, 78, 32]. Le problème avec information sur la valeur optimale *opt* a également fait l'objet d'une enquête approfondie, le meilleur algorithme *semi-online* garantit une performance de 1.6 [11, 21, 55]. Cette borne diminue pour les problèmes avec moins de machines. Cependant, la borne inférieure sur le ratio de compétitivité demeure toujours égale à 1.333 tel que prouvé par Kellerer et al. en 1997 [49]. Le tableau 1.2 présente les travaux dans des problèmes *semi-online* pour machine parallèles identiques qui ont considéré les trois types d'informations les plus populaires : *sum*, *max* et *opt*.

TABLE 1.2 – Les meilleures bornes inférieures et supérieures existantes pour le problème $P_m|semi-online|C_{max}$

	sum		*max*		*opt*	
Nombre de machines (m)	BI	BS	BI	BS	BI	BS
2	1.333 [49]	1.333 [49]	1.333 [41]	1.333 [41]	1.333[49]	1.333[49]
3	1.393 [7, 55]	1.400 [55]	1.414 [78]	1.500 [95]	1.333 [49]	1.400 [11, 55]
4	1.442 [55]	1.462 [55]	1.457 [55]	1.618 [55]	1.333 [49]	1.462 [11, 55]
5	1.482 [55]	1.500 [55]	1.457 [55]	1.667 [55]	1.333 [49]	1.500 [11, 55]
Arbitraire	1.585 [21, 4, 55]	1.600 [21]	1.457 [95, 55]	1.920 [1, 78, 32]	1.333 [49]	1.600 [21, 11, 55]

TABLE 1.3 – Problèmes *semi-online* avec une seule information et des informations combinées pour tous types d'environnement machine

		Informations partielles										Fonction objectif		
Références	Année	sum	opt	max	$decr$	bf	γ	UB	LB	T_k	$deter$	$reas$	C_{max}	$\sum w_j C_j$	$\sum C_j$
[49, 55, 21, 4]	[1997, 2013]	X											X		
[49]	1997					X							X		
[49]	1997											X	X		
[55, 21, 11]	[2001, 2013]		X										X		
[21]	2005	X											X		
[41, 78, 95, 32, 1, 55]	[1999, 2008]			X									X		
[76]	2000				X								X		
[22]	2012				X								X		
[75]	2009											X	X		
[84]	2008											X	X		
[20]	2013											X	X		
[17]	2012		X			X							X		
[17]	2012			X		X							X		
[17]	2012				X	X							X		
[83]	2002	X		X									X		
[83]	2002	X			X								X		
[40, 18]	2007-2011							X	X				X		
[39]	2009									X				X	
[89]	2010						X							X	
[60]	2016										X			X	
[87]	2009						X								X
[58]	2009										X				X

Minimisation de la somme des dates de fin des tâches et la somme des dates de fin pondérées :
Pour le problème de minimisation de la somme des dates de fin des tâches, très peu de travaux peuvent être identifiés dans la littérature. Ceci est également le cas pour le problème de minimisation de la somme des dates de fin pondérées, où chaque tâche a un poids différent. En effet, la recherche dans ce contexte en est encore à ses débuts. Les quelques contributions sont limitées à l'environnement de machine unitaire que nous détaillons dans la section suivante. Nous présentons dans le Tableau 1.3 une vue d'ensemble des travaux qui ont considéré les informations partielles pour la minimisation du *makespan* et la somme des dates de fin pondérées. Nous pouvons clairement remarquer l'énorme nombre de contributions pour la première fonction objectif par rapport à la deuxième.

1.6 Ordonnancement sur machine unitaire

Les modèles d'ordonnancement sur machine unitaire sont importants pour diverses raisons. L'environnement à machine unique est simple et constitue un cas particulier de tous les autres environnements. Les modèles à machine unique ont souvent des propriétés que n'ont ni les machines en parallèle ni les machines en série. Les résultats qui peuvent être obtenus pour les modèles à une seule machine ne fournissent pas uniquement un aperçu de l'environnement à une seule machine, ils fournissent également une base pour les heuristiques qui sont applicables à des environnements de machines plus complexes. En pratique, les problèmes d'ordonnancement dans des environnements de machines plus complexes sont souvent décomposés en sous-problèmes qui traitent des machines uniques.
Nous présentons dans cette section un bref état de l'art des principaux résultats, dans le contexte *offline* classique, classifiés selon deux types de critère d'optimisation : la minimisation du coût total qui est exprimé sous forme d'une somme comme $\sum C_j$, $\sum w_j C_j$ et $\sum T_j$, et la minimisation du coût maximal qui représente une valeur maximale comme C_{max}, T_{max} et L_{max}. Ensuite, nous nous focalisons sur les contextes *online* et *semi-online* pour lesquelles nous présentons un état de l'art plus exhaustif.

1.6.1 Le contexte *offline*

Minimisation du coût total :
Le problème $1||\sum w_j C_j$ peut être résolu en $O(n\ log(n)))$ de temps en utilisant la règle de Smith : Ordonnancer une tâche selon l'ordre décroissant de $\frac{w_j}{p_j}$ [81]. Si les poids sont

égaux, la règle SPT (Shortest Processing Time) est utilisée, celle-ci consiste à ordonnancer la tâche en ordre croissant de leurs temps de traitement. Cependant, l'ajout des contraintes de précédence résulte en NP-difficulté, même pour des temps de traitement et des poids unitaires [51, 56]. En outre, si les dates d'arrivée sont introduites, $1|r_j|\sum C_j$ devient NP-difficile selon [56]. Cependant, dans le cas où les préemptions sont autorisées, le problème peut être facilement résolu en utilisant la règle SRPT (Shortest Remaining Processing Time) qui consiste à ordonnancer les tâches en ordre croissant des temps de traitement restants. En ce qui concerne le problème $1|pmtn, r_j|\sum w_jC_j$, Labetoulle et al. [50] ont prouvé que le problème est NP-difficile. De plus, une borne inférieure sur la solution optimale pour ce problème peut être obtenue en utilisant une relaxation simple de fractionnement des tâches, également appelée *mean busy date relaxation*, développée par Posner [71] et Belouadah et al. [13], et généralisée par Goemans [33] et Uma et al. [92]. La relaxation consiste à découper une tâche en plusieurs morceaux et à attribuer à chaque morceau de la tâche un poids égale à $\rho_j = \frac{w_j}{p_j}$ fois son temps de traitement. Ainsi, un coût est enregistré à la fin de traitement de chaque morceau de la tâche.

D'autre part, le problème $1||\sum w_jT_j$ est NP-difficile [53, 56]. Plusieurs méthodes de résolution ont été proposées pour ce problème. De plus, des bornes inférieures ont été développées pour le problème comprennant une relaxation d'assignation linéaire utilisant une sous-estimation du coût d'affectation de la tâche J_k à la position k [74] et une relaxation de l'exigence selon laquelle la machine ne peut traiter qu'une seule tâche à la fois [31]. Cependant, quand les dates d'arrivée sont introduites, $1|r_j|\sum T_j$ est NP-difficile [37].

En ce qui concerne le problème $1||\sum U_j$, un algorithme introduit par Moore [62] permet de trouver une solution dans $O(n\ log(n))$ de temps : les tâches sont affectées à l'ordonnancement dans un ordre décroissant de leurs dates d'échéance et si l'affectation de J_j résulte que cette tâche est terminée après d_j, la tâche planifiée avec le plus long temps de traitement est marquée et celle-ci est retirée. Le problème $1||\sum w_jU_j$ est NP-difficile selon Karp [48], mais peut être résolu en utilisant la programmation dynamique en $O(n\sum p_j)$ de temps [54]. Évidemment, $1|r_j|\sum U_j$ est également NP-difficile ainsi que la version avec pondération des tâches $1|r_j|\sum w_jU_j$. Afin de résoudre ce problème, la première métaheuristique a été proposée par Sevaux et Dauzère-Pérès [77]. Enfin, des procédures de *branch & bound* ont été développées pour résoudre le cas où toutes les dates d'arrivée sont égales [72] $(1||\sum w_jC_j)$.

Minimisation du coût maximal :
Lawler [52] a présenté un algorithme de complexité $O(n^2)$ pour résoudre le problème $1|prec|f_{max}$ pour des fonctions de coût arbitraires non décroissantes. Son algorithme consiste à définir à chaque étape un ensemble S d'indices des tâches disponibles, $p(S) = \sum_{j \in S} p_j$ et $S' \subset S$ qui indique les tâches dont les successeurs ont été planifiés. L'algorithme choisit une tâches J_k à mettre à la dernière position parmi les tâches $\{J_j | j \in S\}$ tout en exigeant que $f_k(p(S)) \leq f_j(p(S))$ pour tout $j \in S'$.
Pour le problème $1|r_j|C_{max}$, il suffit d'ordonnancer chaque tâche dès son arrivée sans insérer un temps d'inactivité dans n'importe quel ordre pour trouver la solution optimale. Cependant, quand des environnements machine plus complexes sont considérés comme par exemple le *flow time*, le problème devient NP-difficile. Afin de résoudre ce problème, plusieurs heuristiques et métaheuristiques ont été proposés comme [44, 91].
Le problème $1|r_j|L_{max}$ est NP-difficile selon Lenstra [56] alors que la version sans les dates d'arrivée r_j peut être résolue en utilisant la règle de Jackson (EDD) : ordonnancer les tâches en fonction de dates d'échéance non décroissantes.
Les deux problèmes $1|prec, r_j, p_j = 1|L_{max}$ et $1|pmtn, prec, r_j|L_{max}$ peuvent être résolus en temps polynomial en mettant à jour les dates d'arrivée ainsi que les dates d'échéance des tâches afin qu'elles reflètent bien les contraintes de précédence et ensuite appliquer la règle de Jackson sur les tâches disponibles [51].

1.6.2 Le contexte *online*

Dans le modèle *online*, les problèmes de minimisation de la somme des dates de fin des tâches sur une seule machine sont largement étudiés, notés $1|online, r_j|\sum C_j$. Philips et al. [68] ont présenté un algorithme de compétitivité 2 en se basant sur l'ordonnancement préemptif optimal, celui-ci est obtenu en utilisant la règle SRPT (Shortest Remaining Processing Time) [79]. Hoogeveen et Vestjens [42] ont présenté un autre algorithme pour le problème, nommé D-SPT (Delayed Shortest Processing Time).

L'algorithme D-SPT
A chaque instant t quand la machine est inactive et des tâches sont disponibles, choisir la tâche avec le plus petit temps de traitement, noté J_j. Si plusieurs tâches avec le même temps de traitement existent, choisir celle arrivée en premier. Si $p_j \leq t$, alors planifier la tâche J_j ; Sinon, attendre jusqu'à ce que la condition soit vérifiée ou jusqu'à ce qu'une nouvelle tâche arrive, selon l'événement arrivant en premier.

Pour le problème $1|online, r_j|\sum C_j$, Stougie (cité dans [94]) a repris l'idée de D-SPT qui consiste à décaler les dates d'arrivée des tâches dans le temps avant de les

ordonnancer. Lu et al. [59] ont ensuite généralisé l'idée et ont proposé un ensemble d'algorithmes avec le même ratio de compétitivité de 2.

Les algorithmes cités ci-dessus sont les meilleurs possibles puisque Hoogeveen et Vestjens [42] ont montré qu'aucun algorithme *online* ne peut avoir un ratio de compétitivité meilleur que 2. Pour le problème $1|online, r_j, restart\ |\sum C_j$ où l'on peut interrompre une tâche et la redémarrer ultérieurement, Vestjens [94] a prouvé que la borne inférieure sur le ratio de compétitivité est 1.112. Epstein et Van Stee [28] ont ensuite amélioré cette borne vers 1.2108. Ensuite, Stee et Poutré [93] ont présenté un algorithme *online* intégrant le redémarrage des tâches avec un ratio de compétitivité de $\frac{3}{2}$.

En ce qui concerne le problème $1|online, r_j|\sum w_j C_j$, Anderson et Potts [6] ont démontré qu'aucun algorithme *online* ne peut garantir un ratio de compétitivité meilleur que 2. Ainsi, ils ont présenté un algorithme, nommé D-SWPT (*Delayed Shortest Weighted Processing Time*), une extension directe de l'algorithme D-SPT. En utilisant une nouvelle technique de démonstration, ils ont prouvé que D-SWPT est 2-compétitif. Cependant, quand le redémarrage des tâches est autorisé, la borne inférieure du problème passe à 1.2232 [28], tandis que la borne supérieure est toujours égale à 2 [6].
Contrairement à la version non-pondérée, même en autorisant les préemptions Vestjens [94] a montré que la borne inférieure du problème $1|online, r_j, pmtn|\sum w_j C_j$ est 1.0333. Epstein et Van Stee [28] ont amélioré ensuite cette borne à 1.0730. Il a été ensuite prouvé que de nombreux algorithmes sont 2-compétitif, à savoir D-SWPT, P-SWPT et SWRPT [61]. L'algorithme *online* nommé P-SWPT (*Preemptive SWPT*) consiste à ordonnancer à chaque instant la tâche avec le plus grand ratio $\frac{w_j}{p_j}$ alors que l'algorithme SWRPT (*Shortest Weighted Remaining Processing Time)* ordonnance à chaque instant la tâche avec le plus petit ratio du poids sur le temps de traitement restant. Une instance présentée par Xiong et Chung [96] a montré que le ratio de compétitivité de SWRPT ne peut pas être plus petit que 1.215. Bien qu'il y ait un grand écart entre les bornes inférieure et supérieure, le développement d'un algorithme *online* avec un ratio de compétitivité meilleur que 2 n'est pas une tâche triviale. Plus tard, Sitters [80] a présenté un algorithme *online* pour le problème, nommé *ONLINE(c)*, avec un ratio de compétitivité égale à $C \geq \psi$, où $\psi \approx 1.57$ est la racine réelle de $2\psi^3 - 4\psi^2 + 2\psi - 1 = 0$. Par conséquent, il existe un algorithme ψ-compétitif pour le problème avec préemptions. L'algorithme *ONLINE(c)* utilise le paramètre $c \geq 1$ et applique la règle P-SWPT avec la restriction qu'une tâche ne peut être interrompue à un instant t si elle peut finir avant ct.

Vue la compétitivité relativement élevée de plusieurs algorithmes *online*, des recherches récentes ont étudié des scénarios dans lesquels la contrainte *online* est relaxée. Nous présentons dans ce qui suit les développements sur les problèmes d'ordonnancement *semi-online*.

1.6.3 Le contexte *semi-online*

Pendant près de 20 ans, seuls les algorithmes purement *online* et *offline* ont été analysés, et aucune attention n'a été accordée aux algorithmes se situant entre ces deux classes. Dans un cadre plus général, nous pouvons envisager de fournir à l'algorithme plus d'informations sur la liste et/ou plus de liberté par rapport au modèle *online* pur. Dans cette section, nous présentons les quelques études traitants le modèle d'ordonnancement *semi-online* sur machine unitaire.

Tan et Zhang [83] ont présenté un état de l'art des études existantes, tout en précisant les bornes inférieures et supérieures obtenues pour chaque problème. Tao et al. [87] ont étudié le problème $1|online, r_j, \frac{p_{max}}{p_{min}} \leq \gamma| \sum C_j$ où le rapport entre le temps de traitement le plus long p_{max} et le plus court p_{min} n'est pas supérieur à une valeur γ. Ils ont présenté un algorithme *semi-online*, nommé αD-SPT, avec un ratio de compétitivité égale à $1 + \frac{1}{\alpha}$, où $\alpha = \frac{1+\sqrt{1+\gamma(\gamma-1)}}{\gamma-1}$. L'algorithme αD-SPT est décrit comme suit.

αD-SPT : à chaque fois que la machine est inactive et que certaines tâches sont disponibles, choisir une tâche avec le temps de traitement le plus court parmi toutes les tâches arrivées et non ordonnancées, disons J_j. Si $p_j \leq \alpha t$, planifier J_j ; sinon, attendre que l'inégalité ci-dessus soit satisfaite ou qu'une nouvelle tâche arrive.

Liu et al. [58] ont ensuite étudié le problème de détérioration des tâches $1|online, r_j \geq t_0, p_j = a_j t| \sum C_j$, avec $a_j \geq 0$ et t sont le taux de détérioration et la date de début de traitement des tâches, respectivement. Ils ont proposé un algorithme appelé D-SGR (*Delayed Smallest Growth Rate*) avec un ratio de compétitivité de $1 + a_{max}$, avec $a_{max} = \max_{J_j \in l}\{a_j\}$.
De même, pour le problème $1|online, r_j| \sum w_j C_j$, peu de variantes *semi-online* ont été étudiées. Ma et al. [60] ont étudié le problème de la détérioration linéaire des tâches avec $p_j = \alpha_j(A + BS_j)$, où A et B sont des valeurs non-négatives, $A + B > 0$ et $\alpha_j \geq 0$ est le taux de détérioration de la tâche J_j. Ils ont proposé le meilleur algorithme *semi-online* possible pour le problème, avec un ratio de compétitivité égale à $1 + \lambda(A) + \alpha_{max} B$, où $\alpha_{max} = \max_{1 \leq j \leq n} \alpha_j$ et $\lambda(A) = 0$ ou $\lambda(A) = 1$ en fonction de si $A = 0$ ou $A > 0$. En outre, Tao et al. [89] ont présenté un algorithme *semi-online* nommé βD-SWPT

(β *Delayed Shortest Weighted Processing Time*), où $\beta = \frac{\sqrt{4\gamma^2+1}+1}{2\gamma}$, avec un ratio de compétitivité égale à $1 + \frac{1}{\beta}$, pour le problème *semi-online* où le rapport entre le temps de traitement le plus long et le plus court n'est pas supérieur à une valeur γ.

Tous les travaux cités ci-dessus portaient sur les informations relatives aux temps de traitement des tâches. Hall et al. [39] ont travaillé sur l'information sur les dates d'arrivée des tâches en considérant le problème d'ordonnancement *online* avec périodes de planification. L'environnement de planification modélise les processus périodiques de commande et d'ordonnancement. Les commandes sont acceptées, par exemple, chaque semaine. Entre deux commandes, aucune nouvelle tâche n'est disponible. Jusqu'à ce qu'une commande arrive, les détails de la commande, par exemple, la quantité de traitement qu'elle nécessite et sa valeur, ne sont pas connus. Entre-temps, il est nécessaire de prendre des décisions d'ordonnancement concernant les tâches actuellement disponibles, en tenant compte de la possibilité que de nouvelles tâches arrivent au début des semaines suivantes. Notons par T_k une date d'arrivée potentielle, avec $k \in \{1, ..., m\}$ et m désigne le nombre total des dates d'arrivée potentielles. Hall et al. [39] ont présenté un algorithme *semi-online* pour le problème, nommé CSWPT (*Critical Shortest Weighted Processing Time*), avec un ratio de compétitivité égal à $\frac{1+\sqrt{(5)}}{2} \leq R^* < 2$. De plus, ils ont prouvé que la borne inférieure sur le ratio de compétitivité du problème est R^*, ce qui signifie que CSWPT est le meilleur algorithme possible pour le problème considéré (voir le tableau 1.4).

Les travaux cités ci-dessus ont pu fournir des algorithmes plus compétitifs en utilisant une information partielle sur l'instance à ordonnancer (tableau 1.4). Cependant, le modèle *semi-online* du problème de minimisation de la somme des dates de fin des tâches ou la somme des dates de fin pondérées sur machine unitaire n'a pas été suffisamment étudié dans la littérature. C'est pourquoi, dans cette thèse, nous avons choisis de nous focaliser sur ce problème en identifiant les informations partielles qui contribuent à l'amélioration de la prise de décision d'ordonnancement.

Table 1.4 – Les meilleurs résultats connus sur le modèle *over-time* pour l'ordonnancement sur machine unitaire en *online* et en *semi-online*

Problème	Information	Notations	Travaux	Algorithmes	BI	BS
$1\|r_j\|\sum C_j$	Aucune	Aucune	[42]	D-SPT	2	2
	Temps de traitement bornés	$\frac{p_{max}}{p_{min}} \leq \gamma$	[87]	αD-SPT	$1+\frac{1}{\alpha}$	$1+\frac{1}{\alpha}$
	Détérioration des tâches	$p_j = a_j t$	[58]	D-SGR	$1+a_{max}$	$1+a_{max}$
$1\|r_j\|\sum w_j C_j$	Aucune	Aucune	[6]	D-SWPT	2	2
	Temps de traitement bornés	$\frac{p_{max}}{p_{min}} \leq \gamma$	[89]	βD-SWPT	$1+\frac{1}{\beta}$	$1+\frac{1}{\beta}$
	Détérioration des tâches	$p_j = \alpha_j(A + BS_j)$	[60]	DSWGR	$1+\lambda(A)+\alpha_{max}B$	$1+\lambda(A)+\alpha_{max}B$
	Dates d'arrivée potentielles	T_k	[39]	CSWPT	R^*	R^*

1.7 Conclusion

Dans ce chapitre, nous avons introduit quelques définitions relatives à l'ordonnancement *offline* classique, *online* et *semi-online*. Ces définitions sont nécessaires pour permettre au lecteur de bien comprendre les particularités de chaque modèle et leurs différences, notamment en termes de méthodes d'évaluation de performance. Ensuite, nous avons présenté un état de l'art des algorithmes *online* et *semi-online* rencontrés dans la littérature.

Cette revue de la littéraire nous a permis d'apercevoir le grand manque des travaux sur les problèmes de minimisation de la somme des dates de fin des tâches par rapport aux autres fonctions objectives. Étant donné que le problème à machine unitaire est d'un intérêt fondamental avant qu'il soit d'intérêt pratique, nous avons choisit de nous focaliser sur cet environnement qui est un cas particulier de tous les autres environnements. Ainsi, les résultats obtenus pour les modèles de machine unitaire pourront servir de base à des heuristiques applicables à des environnements plus complexes.

Dans le chapitre suivant, nous étudions plusieurs variantes *semi-online* où nous considérons des informations partielles relatives aux temps de traitement des tâches pour la minimisation de la somme des dates de fin des tâches sur machine unitaire. L'étude permettra une classification des informations selon une analyse de valeur.

Chapitre 2

Utilité des informations sur les temps de traitement pour le problème $1|semi-online, r_j|\sum C_j$

Résumé :

Dans ce chapitre, nous étudions plusieurs informations sur les temps de traitement des tâches et leur influence sur la performance d'un algorithme *online*. Le problème étudié est la minimisation de la somme des dates de fin des tâches sur machine unitaire. Les tâches arrivent en *over-time*. Nous étudions alors les bornes inférieures pour plusieurs problèmes *semi-online*, en présentant pour chaque cas l'exemple pour lequel aucun algorithme *semi-online* ne peut garantir une meilleure performance. Nous nous focalisons dans ce chapitre sur l'information sur les temps de traitement des tâches et leur influence sur la compétitivité d'un algorithme *online*. À la fin de cette étude, nous présentons une classification des problèmes *semi-online* selon la notion de *valuableness analysis* telle que définie par Tan et Zhang [85].

2.1 Introduction

L'ordonnancement *online* a été intrinsèquement et largement utilisé dans les contextes scientifiques et techniques modernes. Cependant, vu la compétitivité relativement élevée des algorithmes *online*, plusieurs études ont été réalisées où la contrainte *online* est relâchée partiellement. En effet, dans le monde réel, plusieurs informations sur l'instance à ordonnancer peuvent être disponibles au préalable. C'est pourquoi, un décideur doit savoir si les informations disponibles sont utiles ou si elles n'apportent rien en termes d'amélioration de la prise de décision.
La valeur d'un modèle *semi-online* telle qu'introduite par Tan et Zhang [85] est définie en comparant la borne inférieure sur le ratio de compétitivité du modèle *semi-online* à celle du modèle *online*. Si l'ajout d'une information contribue à diminuer la limite inférieure du modèle *online*, alors l'information est considérée comme *valuable*, sinon, elle est *valueless*.

Dans le contexte *semi-online*, la majorité des études se sont focalisées sur l'objectif de minimisation du *makespan*. Nous citons quelques travaux qui ont étudié l'intérêt de la connaissance de l'information sur les temps de traitement des tâches :

— [49] Kellerer *et al.* ont étudié le problème $P_2|online, max|C_{max}$. Ils ont présenté une heuristique, nommée $H3$, pour une arrivée des tâches *over-list*. Ils ont prouvé que le ratio de compétitivité de $H3$ est $\frac{4}{3}$, qui s'avère être le meilleur ratio possible. En d'autres termes, la borne inférieure sur le ratio de compétitivité est également égale à $\frac{4}{3}$. Ce résultat bat la meilleure borne connue de $\frac{3}{2}$ en *online* [35].

— [22] Cheng *et al.* ont considéré le problème $P_m|online, decr|C_{max}$ pour un nombre de machines m arbitraire. La version *online* de ce problème a été étudiée par Gormley *et al.* [34], qui ont prouvé une borne inférieure du problème égale à 1.853. Cependant, en ajoutant l'information que les temps de traitement des tâche arrivent en ordre décroissant, Cheng *et al.* [22] ont pu améliorer cette borne en présentant un algorithme 1.25-compétitif pour $m \geq 3$ et un algorithme qui est le meilleur possible pour $m = 3$.

En ce qui concerne l'objectif de minimisation de la somme des dates de fin des tâches, peu de travaux étudiants la variante *semi-online* peuvent être identifiés dans la littératures. Nous citons les deux travaux qui ont étudié une information sur les temps de traitement des tâches :

— [87] Tao *et al.* ont étudié le cas où le rapport du temps de traitement le plus long sur le plus court n'est pas supérieur à une valeur $\gamma > 0$. Ils ont pu améliorer la compétitivité *online* en introduisant un algorithme *semi-online* appelé αD-SPT, une version modifiée de l'algorithme D-SPT présenté par Hoogeveen et Vestjens [42]. L'idée de αD-SPT est que moins de temps d'attente est nécessaire puisque les temps de traitement des tâches ne peuvent varier que dans un intervalle limité. αD-SPT est $1 + \frac{1}{\alpha}$ -compétitive, avec $\alpha = \frac{1+\sqrt{1+\gamma(\gamma-1)}}{\gamma-1}$. Ce résultat bat la borne connue de 2 [42] puisque $1 + \frac{1}{\alpha} < 2$. Il est à noter que le ratio de compétitivité de αD-SPT converge vers 2 lorsque γ tends vers l'infini.

— [58] Liu *et al.* ont étudié le problème de la détérioration des temps de traitement des tâches, le problème est noté $1|online, r_j, p_j = a_j t|\sum C_j$, où $a_j \geq 0$ et t sont respectivement le taux de détérioration et la date de décision. Ils présentent un algorithme appelé D-SGR (*Delayed Smallest Growth Rate*) avec un ratio compétitivité de $1 + a_{max}$, où $a_{max} = max_{J_j \in l}\{a_j\}$. Encore une fois, ce résultat étant meilleur que la borne connue de 2, les auteurs ont ainsi prouvé l'avantage de l'information partielle qu'ils ont considérée.

Dans les deux algorithmes *semi-online* αD-SPT et D-SGR et même dans l'algorithme *online* D-SPT, quand plusieurs tâches sont disponibles à un instant t, la tâche avec le plus petit temps de traitement est choisie dans un premier temps. Ensuite, chacun de ces algorithmes applique une règle différente pour déterminer si cette tâche doit être ordonnancée immédiatement. Ceci vient du fait que pour le problème $1||\sum C_j$, où les dates d'arrivée des tâches sont égales, un ordonnancement optimal peut être construit en utilisant la règle SPT. Celle-ci consiste à ordonnancer les tâches en ordre croissant de leurs temps de traitement. Nous rappelons également que pour le problème *offline* $1|r_j, pmtn|\sum C_j$, où les préemptions sont autorisées, la règle SRPT, qui consiste à ordonnancer les tâches en ordre croissant des temps de traitement restants, est optimale.

Comme indiqué dans le tableau 2.1 ci-dessous, aucune étude (NW) n'a été faite pour les problèmes d'ordonnancement *semi-online* sur machine unitaire avec minimisation de la somme des dates de fin des tâches, où l'ordre des temps de traitement (décroissant ou croissant), le plus long temps de traitement en dernier, le temps de traitement maximal ou la somme des temps de traitement, est connu à l'avance.
Dans ce chapitre, nous étudions chacun de ces problèmes et nous présentons une analyse

dans laquelle nous distinguons les informations dites *valuable* et *valueless* en fonction de la borne inférieure sur le ratio de compétitivité, notée BI. Nous montrons que certains des problèmes étudiés ont la même borne inférieure sur le ratio de compétitivité que le problème *online* et donc des algorithmes *online* existants avec une borne supérieure sur le ratio de compétitivité équivalents, noté BS, peuvent être appliqués.

Table 2.1 – Récapitulatif des travaux sur le problème $1|semi-online, r_j|\sum C_j$ avec une information sur les temps de traitement des tâches

Information	Notation	Référence	BI	BS
Temps de traitement bornés	$\frac{p_{max}}{p_{min}} \leq \gamma$	[87]	$1+\frac{1}{\alpha}$	$1+\frac{1}{\alpha}$
Détérioration des tâches	$p_j = a_j t$	[58]	$1+a_{max}$	$1+a_{max}$
Ordre décroissant	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0<\beta<1$	NW		
	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0<\beta<1$ et $p_j \geq p_L$	NW		
Ordre croissant	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $\beta \geq 1$	NW		
Ordre arbitraire	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $0<\beta<1$	NW		
	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $\beta \geq 1$	NW		
le plus long temps de traitement en dernier	LL	NW		
Temps de traitement maximal	max	NW		
Somme des temps de traitement	sum	NW		

2.2 Modèle de l'adversaire

Vu que le comportement d'un algorithme *online* ou *semi-online* dépend de l'instance des tâches considérées, il est nécessaire d'avoir un modèle de ce qui produit ces instances,

afin de pouvoir analyser l'algorithme. Normalement, nous considérons les entrées comme dérivant d'un agent (une personne ou un programme, ou une personne exécutant un programme) appelé un adversaire. Le mot adversaire suggère que l'agent travaille contre l'algorithme *online* qui essaie de bien fonctionner dans toutes les situations y compris la pire situation possible. Dans le chapitre précédent, nous avons introduit plusieurs types d'adversaires qui peuvent être utilisés afin de construire la pire instance. Dans notre étude, nous avons choisi l'adversaire le plus fort qui est «l'adversaire adaptatif *offline*» afin de prendre en compte tous les types de prise de décision possibles (aléatoires et déterministes). Nous rappelons ci-dessous la définition de la borne inférieure générale sur le ratio de compétitivité .

Définition 3. b est une borne inférieure sur le ratio de compétitivité si :

$$\exists I_1, \forall ALG, \quad \frac{ALG(I_1)}{OPT(I_1)} \geq b \tag{2.1}$$

Avec I_1 la pire instance que peut rencontrer un algorithme *online*.

2.3 Les modèles *semi-online*

Nous étudions dans cette section le problème *semi-online* où les temps de traitement des tâches consécutives doivent respecter une certaine contrainte ou écart. Nous concluons alors sur l'utilité de l'information ajoutée sur la compétitivité d'un algorithme *online*.

Le premier problème considéré dans cette section est $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $0 < \beta < 1$. Nous démontrons que le ratio de compétitivité reste inchangé quand le nombre de tâches, noté n, tend vers l'infini.
Ensuite, pour le même problème avec $\beta \geq 1$, nous montrons que la règle SPT (*Shortest Processing Time*) assure la même performance qu'un algorithme *offline* optimal. En d'autres termes, la borne inférieure est la borne supérieure sur le ratio de compétitivité sont égales à 1 (Tableau 2.2).
En plus, pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta|\sum C_j$, où β est une valeur positive, nous prouvons qu'aucun algorithme *semi-online* ne peut avoir un ratio de compétitivité meilleur que 2.
Nous ajoutons ensuite une information sur le temps de traitement de la dernière tâche, noté p_L, au dernier problème étudié avec $0 < \beta < 1$. Ainsi, en considérant l'information sur le temps de traitement de la première tâche, noté p_F, nous montrons que la borne inférieure devient égale à $1 + \frac{1}{\phi'}$, où $\phi' > 1$.

Table 2.2 – Résumé des travaux existants et apportés pour le problème $1|semi-online, r_j|\sum C_j$

Information	Notation	Référence	BI	BS
Temps de traitement bornés	$\frac{p_{max}}{p_{min}} \leq \gamma$	[87]	$1+\frac{1}{\alpha}$	$1+\frac{1}{\alpha}$
Détérioration des tâches	$p_j = a_j t$	[58]	$1+a_{max}$	$1+a_{max}$
Ordre décroissants	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0<\beta<1$	Théorème 3.	2	2
	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0<\beta<1$ et $p_j \geq p_L$	Théorème 4.	$1+\frac{1}{\phi'}$	
Ordre croissants	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $\beta \geq 1$	Théorème 2.	1	1
Ordre arbitraire	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $0<\beta<1$	Théorème 1.	2	2
	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $\beta \geq 1$	Théorème 3.	2	2
Le plus long temps de traitement en dernier	LL	Théorème 6.	2	2
Temps de traitement maximal	MAX	Théorème 5.	2	2
Somme des temps de traitement	SUM	Théorème 5.	2	2

2.3.1 Ordre arbitraire des temps de traitement

Théorème 1. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $0<\beta<1$, aucun algorithme semi-online ne peut garantir un ratio de compétitivité meilleur que 2.

Démonstration. Nous montrons ce résultat en décrivant un ensemble d'instances pour lesquelles aucun algorithme semi-online ne peut garantir un ratio de compétitivité strictement inférieur à 2.

Soit $ALG(I)$ la valeur objective de l'algorithme *semi-online*, et soit $OPT(I)$ la

valeur objective optimale de l'algorithme *offline* pour une instance I. Considérons la situation suivante :
Une première tâche J_1 arrive à $r_1 = 0$ avec un temps de traitement $p_1 = p$. L'algorithme *semi-online* décide d'ordonnancer J_1 à un instant S. En effet, en fonction des données du problème, un algorithme *semi-online* doit forcément ordonnancer la tâche à un certain moment que nous désignons ici par S. Selon la valeur de S, l'adversaire a deux choix pour vaincre l'algorithme *semi-online* :

— Soit il ne lance plus aucune tâche et donc l'algorithme *semi-online* aurait attendu pour rien (cas de l'instance I_1).

— Soit il lance $(n-1)$ tâches à l'instant $S+\epsilon$ avec $p_j = \beta^j p$ pour $j = \{2, ..., n\}$ où ϵ est un nombre positif infiniment petit. Ainsi, l'algorithme *semi-online* n'aurait pas attendu suffisamment (cas de l'instance I_2).

Instance I_1 : L'algorithme *semi-online* ordonnance J_1 à l'instant S alors que l'algorithme *offline* l'ordonnance immédiatement à son arrivée (Figure 2.1),

$$\frac{ALG\,(I_1)}{OPT\,(I_1)} = \frac{S+p}{p} \tag{2.2}$$

Instance I_2 : L'ordonnancement optimal ne serait pas pire que celui qui ordonnance d'abord les $(n-1)$ petites tâches suivies de la tâches J_1, (Figure 2.1)

$$\frac{ALG\,(I_2)}{OPT\,(I_2)} \geq \frac{n\,(S+p) + p\sum_{j=1}^{n-1} j\beta^j}{n\,(S+\epsilon) + p\sum_{j=1}^{n} j\beta^{j-1}} \tag{2.3}$$

En combinant les deux équations (2.2) et (2.3), nous obtenons le résultat suivant :

$$\rho \geq max\left\{\frac{S+p}{p},\ \frac{n\,(S+p) + p\sum_{j=1}^{n-1} j\beta^j}{n\,(S+\epsilon) + p\sum_{j=1}^{n} j\beta^{j-1}}\right\} \tag{2.4}$$

Le choix de S **:** Le meilleur algorithme *semi-online* choisira S de telle sorte à minimiser la valeur de l'expression (2.4). L'expression à gauche est croissante en fonction de S, alors que l'expression à droite est décroissante en fonction de S. Par conséquent, la valeur de S est obtenue en étudiant l'égalité. Ainsi, quand ϵ tend vers 0, nous obtenons

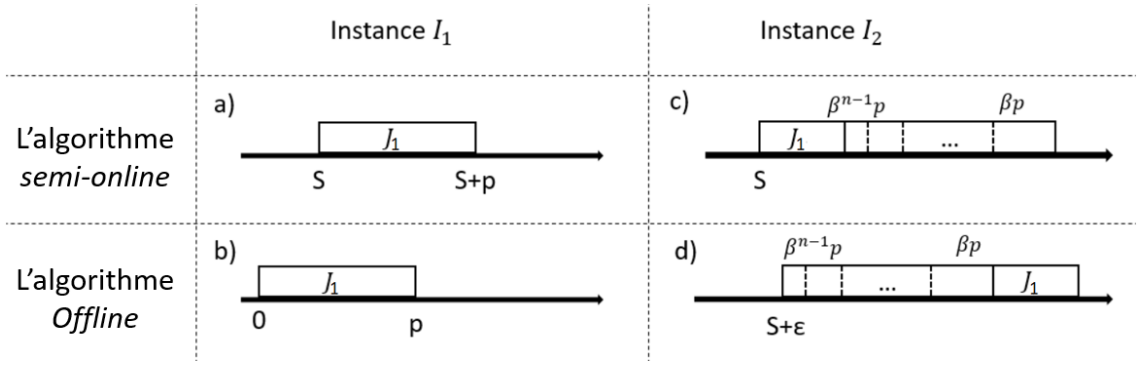

Figure 2.1 – Ordonnancements par l'algorithme *offline* optimal et l'algorithme *semi-online* (Théorème 1)

$$S = \frac{\sqrt{\left(\sum_{j=1}^{n} j\beta^{j-1}\right)^2 - 4n\left(\sum_{j=0}^{n-1}\beta^j - n\right)} - \sum_{j=1}^{n} j\beta^{j-1}}{2n} p \tag{2.5}$$

En remplacent l'expression de S dans (2.2), nous avons

$$\rho \geq 1 + \frac{\sqrt{\left(\sum_{j=1}^{n} j\beta^{j-1}\right)^2 - 4n\left(\sum_{j=0}^{n-1}\beta^j - n\right)} - \sum_{j=1}^{n} j\beta^{j-1}}{2n} \tag{2.6}$$

Afin de calculer la limite de l'expression à droite de l'inégalité (2.6) quand n tend vers l'infinie, nous simplifions l'expression ci-dessus en utilisant l'égalité suivante,

$$\sum_{j=1}^{n} j\beta^{j-1} = n\frac{\beta^n - 1}{\beta - 1} - \frac{\beta^{n-1} - 1}{(\beta - 1)^2} + \frac{n-1}{\beta - 1} \tag{2.7}$$

Nous obtenons alors l'expression suivante,

$$\rho \geq 1 + \frac{\sqrt{\left(\frac{\beta^n-1}{\beta-1} - \frac{\beta^{n-1}-1}{n(\beta-1)^2} + \frac{1-\frac{1}{n}}{\beta-1}\right)^2 - 4\left(\frac{\beta^n-1}{n(\beta-1)} - 1\right)} - \frac{\beta^n-1}{\beta-1} + \frac{\beta^{n-1}-1}{n(\beta-1)^2} - \frac{1-\frac{1}{n}}{\beta-1}}{2} \tag{2.8}$$

Quand n tend vers l'infini, la borne inférieure sur le ratio de compétitivité tend vers 2. □

Avec l'algorithme D-SPT proposé par Hoogeveen et Vestjens [42], nous annonçons l'observation ci-dessous.

Observation 1. L'algorithme D-SPT est optimale pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $0 < \beta < 1$.

2.3.2 Ordre croissant des temps de traitement

Théorème 2. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $\beta \geq 1$, SPT est optimale.

Démonstration. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $\beta \geq 1$, la règle SRPT est optimale quand la préemption est autorisée. En effet, SRPT est optimale pour le problème *online* avec préemptions alors elle est également optimale pour le problème *semi-online*.

De plus, puisque les tâches arrivent en ordre croissant des temps de traitement, SRPT n'aurait pas besoin de préempter des tâches puisque le temps de traitement restant de chaque tâche en cours de traitement sera forcément plus petit que le temps de traitement d'une tâche arrivant dans le futur. Ainsi, SRPT devient équivalent à SPT pour ce type d'instances. Ceci implique que la règle SPT est optimale pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $\beta \geq 1$. □

2.3.3 Ordre décroissant des temps de traitement

Théorème 3. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta|\sum C_j$, où β une valeur positive, aucun algorithme semi-online ne peut garantir un ratio de compétitivité meilleur que 2.

Démonstration. Nous pouvons utiliser les mêmes instances que celles employées par Hoogeveen et Vestjens [42] afin de prouver la borne inférieure sur le ratio de compétitivité de ce problème. Nous rappelons ainsi ces instances et leur application dans ce cas : Considérons la situation suivante. La première tâche arrive à l'instant 0 avec un temps de traitement p. L'algorithme *online* décide d'ordonnancer cette tâche à un instant S. Selon cette valeur de S, soit plus aucune tâche n'arrive, soit $(n-1)$ tâches avec un temps de traitement 0 arrivent à $S+1$. Dans le premier cas, le ratio serait $\frac{S+p}{p}$, alors

que dans le second cas le ratio n'est pas pire que $\frac{n(S+p)}{n(S+1)+p}$. Ceci résulte en une borne inférieure sur le ratio de compétitivité égale à $2-\frac{1}{n}-\frac{1}{p}$. Quand $n \to \infty$ et $p \to \infty$, nous retrouvons la borne de 2.
Ainsi, dans le cas étudié où les temps de traitement des tâches vérifient $\frac{p_{j+1}}{p_j} \leq \beta$, avec β une valeur positive. L'adversaire peut directement lancer une infinité des tâches avec des temps de traitement 0 et donc de retrouver avec la borne inférieure de 2. □

Théorème 4. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$, où $0 < \beta < 1$ et $p_L > 0$, aucun algorithme *semi-online* ne peut garantir un ratio de compétitivité meilleur que $1+\frac{1}{\phi'}$, où $\phi' = \frac{2\bar{n}}{-1-\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}+\sqrt{\left(1+\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2-4\bar{n}\left(1-\bar{n}+\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}\right)}}$ et $\phi' > 1$.

Démonstration. La construction de la pire instance est similaire à celle du théorème 1. La première tâche J_1 arrive à $r_1 = 0$ avec un temps de traitement $p_1 = p_F$. Dans le premier cas, l'adversaire ne libère plus aucune tâche (Instance I_1). De plus, puisque les tâches vérifient $\frac{p_{j+1}}{p_j} \leq \beta$ et $p_n \geq p_L$ alors les plus petits temps de traitement possibles peuvent être déduits en prenant la borne inférieure pour chaque temps de traitement à partir de celui de la dernière tâche. Par conséquent, les temps de traitement des $(n-1)$ petites tâches de l'instance I_2 sont égales à $p_j = \frac{p_L}{\beta^{n-j}}$ pour $j = \{2, ..., n\}$. De plus, le nombre total des tâches n est borné par $\bar{n} = \left\lfloor \frac{ln(\gamma)}{ln(\beta)} + 1 \right\rfloor$, où $\gamma = \frac{p_L}{p_F}$, selon l'inégalité suivante,

$$\begin{aligned} & p_{j+1} \leq \beta p_j \quad pour \quad j \in \{1, ..., n\} \\ \iff & n \leq \frac{ln(\gamma)}{ln(\beta)} + 1 \end{aligned} \tag{2.9}$$

Instance I_1 : L'algorithme *offline* ordonnance la tâche J_1 à l'instant 0 (Figure 2.2).

$$\frac{ALG(I_1)}{OPT(I_1)} = 1 + \frac{S}{p_F} \tag{2.10}$$

Instance I_2 : L'algorithme *offline* optimal n'aura pas une performance pire que celle où les $(n-1)$ petites tâches sont ordonnancées en premier, suivies de J_1 (Figure 2.2). Donc,

$$\frac{ALG\,(I_2)}{OPT\,(I_2)} \geq \frac{n\,(S+p_F)+p_L\sum_{j=2}^{n}(j-1)\,\beta^{j-n}}{n(S+\epsilon)+p_F+p_L\sum_{j=2}^{n}j\beta^{j-n}}$$
$$= 1+\frac{n-1-\gamma\sum_{j=2}^{n}\beta^{j-n}}{n\frac{(S+\epsilon)}{p_F}+1+\gamma\sum_{j=2}^{n}j\beta^{j-n}} \tag{2.11}$$

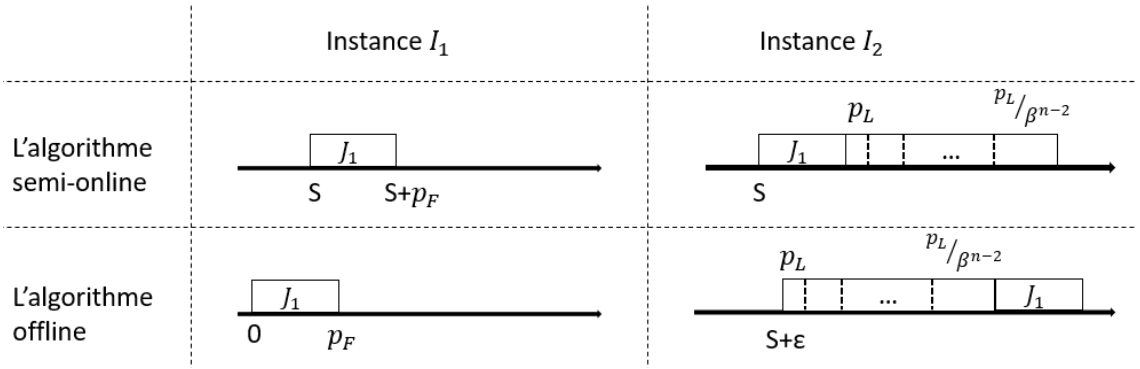

Figure 2.2 – Ordonnancements par l'algorithme *offline* optimale et l'algorithme *semi-online* (Théorème 4)

En combinant les deux équations (2.10) et (2.11), nous obtenons

$$\rho \geq max\left\{\frac{S+p_F}{p_F}, \frac{n\,(S+p_F)+p_L\sum_{j=2}^{n}(j-1)\,\beta^{j-n}}{nS+p_F+p_L\sum_{j=2}^{n}j\beta^{j-n}}\right\} \tag{2.12}$$

L'algorithme *semi-online* peut choisir S de manière à minimiser les deux expressions de l'inégalité (2.12). Cela peut être obtenu en étudiant l'égalité. Par conséquent, quand $\epsilon \to 0$, nous avons

$$S = \frac{-1-\gamma\sum_{j=2}^{n}j\beta^{j-n}+\sqrt{\left(1+\gamma\sum_{j=2}^{n}j\beta^{j-n}\right)^2-4n\left(1-n+\gamma\sum_{j=2}^{n}\beta^{j-n}\right)}}{2n}p_F \tag{2.13}$$

Cela implique,

$$\rho \geq 1+\frac{-1-\gamma\sum_{j=2}^{n}j\beta^{j-n}+\sqrt{\left(1+\gamma\sum_{j=2}^{n}j\beta^{j-n}\right)^2-4n\left(1-n+\gamma\sum_{j=2}^{n}\beta^{j-n}\right)}}{2n} \tag{2.14}$$

Quand n tend vers sa valeur maximale $\bar{n} = \left\lfloor \frac{ln(\gamma)}{ln(\beta)}+1 \right\rfloor$, nous obtenons la borne inférieure

sur le ratio de compétitivité,

$$\rho \geq 1 + \frac{-1-\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(1+\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\bar{n}\left(1-\bar{n}+\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}\right)}}{2\bar{n}} \tag{2.15}$$

$$\rho \geq 1 + \frac{1}{\phi'} \tag{2.16}$$

où $\phi' = \frac{2\bar{n}}{-1-\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(1+\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\bar{n}\left(1-\bar{n}+\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}\right)}}$ et $\phi' > 1$.

□

Remarque. Le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L|\sum C_j$, où $0 < \beta < 1$ et $\gamma = \frac{p_L}{p_F} = 0$ ou $\gamma \to \infty$, devient équivalent au problème énoncé par le théorème 3.

2.3.4 Information sur *max*, *sum* et *LL*

Avec le même raisonnement que dans les sections précédentes nous trouvons que les informations sur le temps de traitement maximal, la somme des temps de traitement ou lorsque le temps de traitement de la dernière tâche est le plus long, ne permettent pas d'améliorer la performance d'un algorithme *online*. Nous énonçons cela par les deux théorèmes suivantes.

Théorème 5. Pour les problèmes $1|online, r_j, max|\sum C_j$ et $1|online, r_j, sum\ |\sum C_j$, aucun algorithme semi-online ne peut garantir un ratio de compétitivité meilleur que 2.

Démonstration. L'instance utilisée par Hoogeveen et Vestjens [42] que nous avons décrite dans la démonstration du théorème 3 reste valide même si l'une des informations suivantes est connue : le temps de traitement maximal ou la somme des temps de traitement. Malgré cette connaissance préalable, l'algorithme *semi-online* est incapable de changer ses décisions car des tâches ayant des temps de traitements égaux à 0 peuvent être libérées au moment où l'algorithme semi-online choisit d'ordonnancer une longue tâche. Ainsi, l'adversaire garde le même avantage sur l'algorithme *semi-online*. □

Théorème 6. Pour le problème $1|online, r_j, LL|\sum C_j$, aucun algorithme *semi-online* ne peut garantir un ratio de compétitivité meilleur que 2.

Démonstration. Considérons les situations suivantes : une première tâche J_1 arrive à $r_1 = 0$ avec un temps de traitement $p_1 = 1 - \epsilon$. L'algorithme *semi-online* ordonnance cette tâche à un instant S. Si plus aucune tâche n'arrive dans le futur, le ratio sera égal à $\frac{S+1-\epsilon}{1-\epsilon}$ (Figure 2.3). Considérons maintenant le cas où $(n-2)$ tâches arrivent à $S + \epsilon$ avec des temps de traitement égaux à 0 et où une dernière tâche J_n arrive à $S + 2\epsilon$ avec un temps de traitement égal à 1. L'algorithme *offline* optimal ordonnance les $(n-2)$ petites tâches en premier suivies des autres tâches en ordre croissant des temps de traitement (Figure 2.3). Par conséquent, le ratio sera alors $\frac{n(s+1-\epsilon)+1}{nS+3+\epsilon(n-2)}$. De plus,

$$\rho \geq max\left\{\frac{S+1-\epsilon}{1-\epsilon}, \frac{n\left(s+1-\epsilon\right)+1}{nS+3+\epsilon\left(n-2\right)}\right\} \tag{2.17}$$

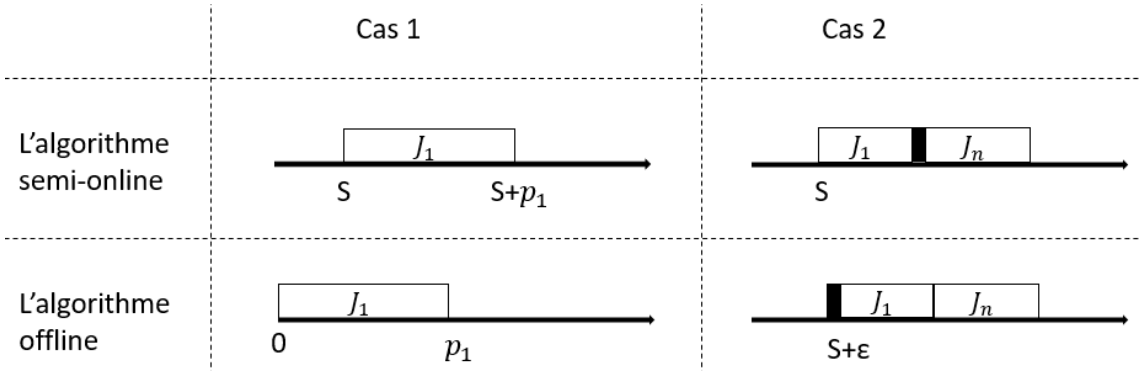

Figure 2.3 – Ordonnancements par l'algorithme *offline* optimale et l'algorithme *semi-online* (Théorème 6)

Quand ϵ tend vers 0, nous obtenons

$$\rho \geq max\left\{S+1, \frac{n\left(s+1\right)+1}{nS+3}\right\} \tag{2.18}$$

En étudiant l'égalité des deux expressions ci-dessus nous trouvons la valeur de S qui minimise la borne.

$$S = \frac{-3+\sqrt{9-8n+4n^2}}{2n} \tag{2.19}$$

Quand n tend vers l'infini, S tend vers 1 et la borne inférieure tend vers 2. □

2.4 Analyse de valeur

La valeur d'un modèle *semi-online* tel qu'introduite par Tan et Zhang [85] est définie en comparant la borne inférieure du modèle *semi-online* à la borne inférieure du modèle *online*. Si l'ajout d'une information contribue à diminuer la borne inférieure du modèle *online*, alors l'information est dite *valuable*. Sinon, elle est dite *valueless*.

Après avoir analysé différents sous-problèmes *semi-online* du problème $1|online, r_j|\sum C_j$ et en essayant de trouver des informations qui pourraient aider à améliorer la borne inférieure, nous pouvons observer que le seul moyen prometteur est de pouvoir limiter le nombre de tâches libérées ou d'avoir une information sur le temps de traitement de la plus petite tâche. Dans le tableau 2.3, nous présentons une classification de certains modèles *semi-online* trouvés dans la littérature que nous étendons avec des modèles que nous avons étudiés.

L'instance présentée par Hoogeveen et Vestjens [42] ne peut pas être appliquée dans l'avant-dernier cas du tableau 2.3 car nous avons des informations sur la diminution des temps de traitement des tâches et sur le temps de traitement de la dernière tâche. Ces deux informations permettent de prédire le nombre maximum des tâches. En conséquence, l'adversaire devient limité dans la valeur du plus petit temps de traitement et dans le nombre de tâches à libérer, qui sont deux techniques fortes et percutantes utilisées pour diminuer les performances d'un algorithme *online* jusqu'à 2. Par conséquent, un algorithme *semi-online* pour ce problème ne doit pas être aussi prudent dans sa stratégie d'attente qu'un algorithme *online*.

Les résultats numériques de la figure 2.4 et de la figure 2.5 ci-dessous permettent d'illustrer la variation des bornes inférieures (BI). La figure 2.4 illustre la variation de la borne inférieure par rapport au nombre de tâches n pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $0 < \beta < 1$. Pour une petite valeur de β, la borne inférieure converge vers la valeur connue de 2, même pour peu de tâches. Cette augmentation est due au fait que les prochaines tâches peuvent être aussi petites que β fois le temps de traitement de la tâche en cours. Par conséquent, en augmentant le nombre de tâches, les temps de traitement des tâches suivantes peuvent diminuer jusqu'à ce qu'ils atteignent la valeur 0. Cependant, pour une grande valeur de β, nous remarquons une petite diminution de la borne inférieure pour un petit nombre de tâches, alors qu'elle finit par converger vers 2.

La figure 2.5 illustre la variation de la borne inférieure par rapport à $\gamma = \frac{p_L}{p_F}$ pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L|\sum C_j$, où $0 < \beta < 1$. Nous remarquons que

Table 2.3 – Résumé des informations dites *valuable* et *valueless* pour le problème $1|online, r_j|\sum C_j$

Ordre des temps de traitement	Informations connues	*Valuable*	*Valueless*
Arbitraire [87]	$\frac{p_{max}}{p_{min}} \leq \gamma$	X	
Décroissant [58]	$p_j = a_j t$	X	
Arbitraire	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $0 < \beta < 1$		X
Croissant	$\frac{p_{j+1}}{p_j} \geq \beta$ pour $\beta \geq 1$	X	
Arbitraire	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $\beta \geq 1$		X
Décroissant	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0 < \beta < 1$		X
Décroissant	$\frac{p_{j+1}}{p_j} \leq \beta$ pour $0 < \beta < 1$ et $p_j \geq p_L$	X	
Arbitraire	max, sum or LL		X

la borne inférieure est plus petite pour une petite valeur de β, car moins de tâches sont susceptibles d'être libérées selon le nombre maximum de tâches $\bar{n} = \left\lfloor \frac{ln(\gamma)}{ln(\beta)} + 1 \right\rfloor$. Cependant, pour une valeur élevée de β, plus de tâches peuvent être libérées à l'avenir, c'est pourquoi la borne inférieure est plus importante.

Pour conclure, les deux paramètres γ et β influencent le ratio de compétitivité de la pire instance en agissant sur le nombre maximum des tâches libérés. Par conséquent, les performances d'un algorithme *semi-online* peuvent varier en fonction de ces paramètres.

2.5 Conclusion

Nous avons étudié, dans ce chapitre, plusieurs sous-problèmes *semi-online* du problème de minimisation de la somme des dates de fin des tâches sur machine unitaire avec des tâches qui arrivent *over-time*. En dérivant des bornes inférieures sur le ratio de compétitivité de chaque problème étudié, nous avons présenté une classification de ceux-ci selon une analyse de valeur qui permet de distinguer les informations utiles de celles non utiles. Ainsi, pour les problèmes d'ordonnancement *semi-online* où l'information partielle est non-utile, le ratio de compétitivité ne peut pas s'améliorer et donc des algorithmes *online* existants peuvent être appliqués. En ce qui concerne les problèmes *semi-online* où l'information est utile comme pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta|\sum C_j$ où $\beta \geq 1$, nous avons démontré que SPT est optimal.

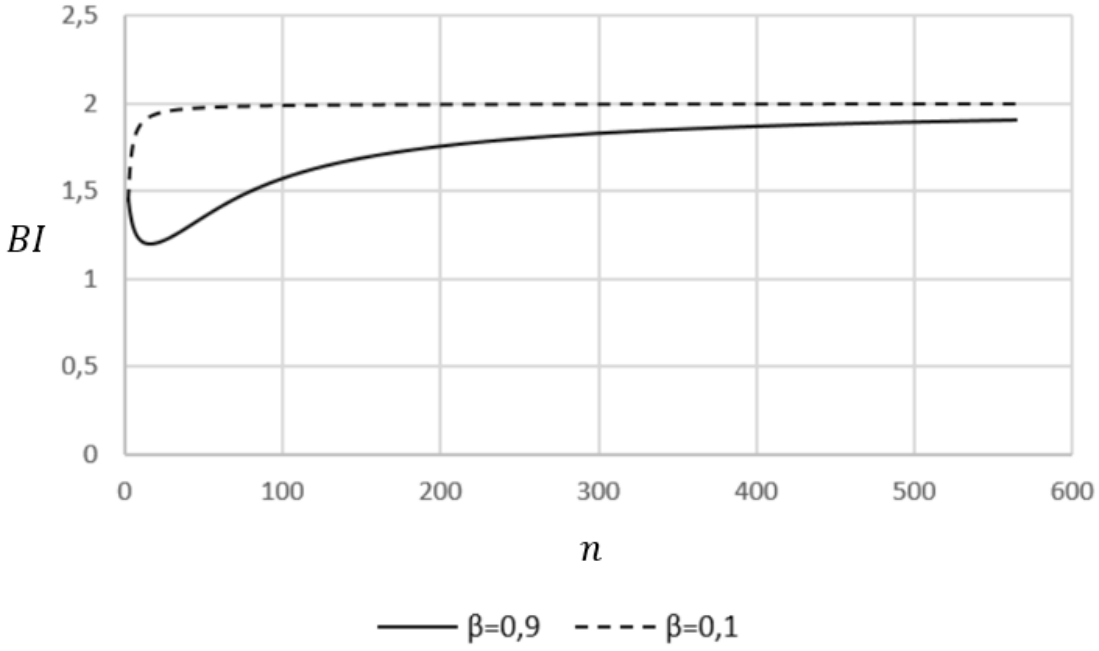

Figure 2.4 – Variation de la borne inférieure sur le ratio de compétitivité en fonction de n pour $\beta = 0,9$ et $\beta = 0,1$ | Problème $1|online, r_j, \frac{p_{j+1}}{p_j} \geq \beta| \sum C_j$ où $0 < \beta < 1$.

En outre, pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$, où $0 < \beta < 1$, nous avons démontré que la borne inférieure générale sur le ratio de compétitivité est égale à $1 + \frac{1}{\phi'}$, où $\phi' > 1$. Pour ce problème, aucun algorithme existant permettant d'atteindre un meilleur ratio de compétitivité ne peut être appliqué. Dans le chapitre suivant, nous présentons un nouvel algorithme *semi-online* pour ce problème permettant d'atteindre une meilleure compétitivité.

Les travaux présentés dans ce chapitre ont fait l'objet des publications suivantes :

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. Semi-online Scheduling For Minimizing The Total Completion Time : Information Value. International Conference on Optimization and Learning, Spain, 2020. [64]

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. Minimising The Total Completion Time for a Class of Semi-online Single Machine Scheduling Problems. Theoretical Computer Science, 2021, en révision. [65]

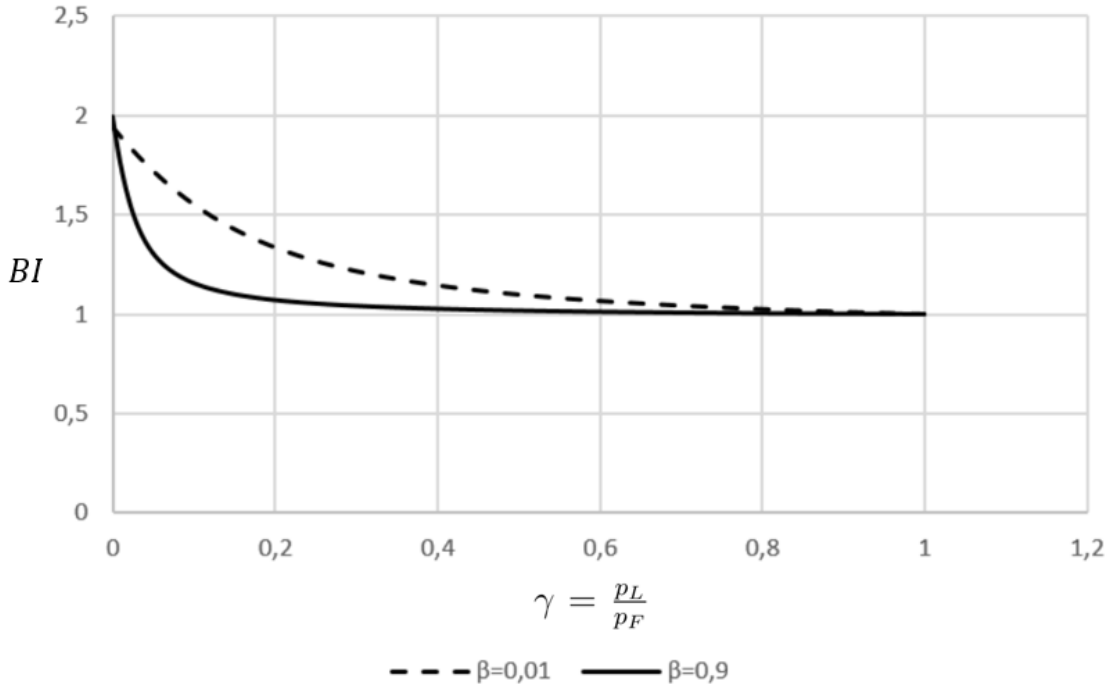

Figure 2.5 – Variation de la borne inférieure sur le ratio de compétitivité en fonction de γ pour $\beta = 0,9$ et $\beta = 0,01$ | problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$ où $0 < \beta < 1$.

Chapitre 3

Algorithme *semi-online* avec informations sur les temps de traitement pour $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$

Résumé :
Dans ce chapitre, nous étudions le problème *semi-online* de minimisation de la somme des dates de fin des tâches sur machine unitaire avec des tâches qui arrivent en ordre décroissant des temps de traitement. Nous présentons un nouvel algorithme, nommé ϕD-SPT, qui prend en compte cette information et qui assure une meilleure performance que l'algorithme classique *online* D-SPT. Nous analysons le ratio de compétitivité de cet algorithme en utilisant la méthode de transformation d'instance introduite par Tao et al. [87]. Le ratio de compétitivité correspond à la borne inférieure du problème *semi-online* obtenue au chapitre précédent, ce qui veut dire que ϕD-SPT est le meilleur algorithme pour le problème.

3.1 Introduction

Comme évoqué au chapitre 1, à la section 1.4, l'ordonnancement *online* ou *semi-online* est un domaine important dans la théorie de l'ordonnancement et fait l'objet d'une attention particulière. En raison de l'absence d'information globale sur l'instance, un ordonnancement *online* ne peut être optimal dans la plupart des cas. La qualité d'un algorithme *online* est souvent évaluée en fonction de ses performances compétitives. Un algorithme est appelé ρ-compétitif si, pour une instance donnée, la valeur de la fonction objective de l'ordonnancement obtenu par cet algorithme n'est pas pire que ρ fois celui de l'ordonnancement *offline* optimal [30].

Dans ce chapitre, nous étudions le problème *semi-online* de minimisation de la somme des dates de fin des tâches, où le décideur a l'avantage d'avoir une information sur les temps de traitement des tâches : l'ordre d'arrivée des tâches est décroissant par rapport aux temps de traitement et ceux-ci sont bornés.
Nous citons quelques travaux qui ont étudié le problème *semi-online* avec des tâches qui arrivent en ordre décroissant des temps de traitement :

— Seiden et al. [76] ont étudié le problème d'ordonnancement *semi-online* pour la minimisation du *makespan* sur machines parallèles identiques où les tâches arrivent dans un ordre décroissant des temps de traitement. Ils ont présenté des bornes inférieures et supérieures sur le ratio de compétitivité pour tout nombre de machines m. Ces bornes tendent vers $\frac{1+\sqrt{3}}{2}$ quand le nombre de machines tend vers l'infini. En ce qui concerne la variante non-préemptive du problème *semi-online* avec des temps de traitement des tâches décroissants, l'algorithme de Graham [36] est $\frac{4}{3} - \frac{1}{3m}$-compétitif. Pour 2 machines, Seiden et al. ont prouvé que cet algorithme est le meilleur possible, avec un ratio de compétitivité de $\frac{7}{6}$. Finalement, ils ont démontré que pour $m = 3$, la borne inférieure est $\frac{1+\sqrt{37}}{6}$.

— Epstein et Favrholdt [27] ont considéré le problème d'ordonnancement *semi-online* non-préemptif où les tâches arrivent en *over-list* dans un ordre décroissant des temps de traitement des tâches. L'environnement d'ordonnancement est composé de deux machines uniformes avec comme fonction objectif la minimisation du *makespan*. Ils ont analysé à la fois le ratio de compétitivité global et le ratio de compétitivité en fonction du rapport de vitesse ($q \geq 1$) entre les deux machines. Ils ont montré que l'algorithme LPT (*Longest Processing Time*) a un ratio de compétitivité optimal de $1,28$ en général, mais pas pour toutes les valeurs de q. Ainsi, ils conçoivent différents algorithmes avec les meilleures compétitivités pour les intervalles où LPT n'est pas le meilleur algorithme.

— Cheng et al. [22] ont considéré le problème d'ordonnancement multiprocesseur *semi-online* pour minimiser le *makespan*, où les tâches arrivent dans un ordre décroissant des temps de traitement. Ils ont construit un algorithme *semi-online* meilleur que LPT avec un ratio de compétitivité de $\frac{5}{4}$ pour $m \geq 3$ et un algorithme optimal pour $m = 3$, c'est-à-dire, un algorithme $\frac{1+\sqrt{37}}{6}$-compétitif.

— Cao et Wan [19] se sont intéressés au problème d'ordonnancement *semi-online* avec information combinée sur deux machines parallèles identiques pour minimiser le *makespan*, où les tâches ont des temps de traitement dans l'intervalle $[1, t]$, avec $t \geq 1$, et elles arrivent dans un ordre décroissant de leurs temps de traitement. Pour $t \geq 1$, ils ont construit un algorithme LS optimal avec un ratio de compétitivité qui correspond à la borne inférieure du problème.

Nous pouvons conclure de ce bref état de l'art que l'information sur les temps de traitement des tâches décroissants a permis d'améliorer la prise de décision pour plusieurs problèmes d'ordonnancement. Les chercheurs se sont particulièrement intéressés aux problèmes de minimisation du *makespan*. Cependant, en ce qui concerne le problème de minimisation de la somme des dates de fin des tâches, aucune étude ne peut être identifiée dans la littérature considérant l'arrivée des tâches en ordre décroissant des temps de traitement. Dans ce chapitre nous présentons un nouvel algorithme pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$ pour $0 < \beta < 1$ et $p_L > 0$ suivi d'une analyse de compétitivité.

3.2 Notations et définition du problème

Nous considérons le problème *semi-online* avec une information combinée pour la minimisation de la somme des dates de fin des tâches sur machine unitaire avec des tâches qui arrivent en *over-time*. Les tâches ont des temps de traitement dans l'intervalle $[p_F, p_L]$, avec $p_F > p_L > 0$, et elles arrivent dans un ordre décroissant de leurs temps de traitement selon le critère $p_{j+1} \leq \beta p_j$, avec $0 < \beta < 1$ et p_j le temps de traitement de la tâche J_j pour $j = \{1, ..., n-1\}$ et n le nombre de tâches dans une instance.
En ce qui suit, nous allons utiliser les notations suivants :

— r_j : Date d'arrivée d'une tâche J_j.

— C_j : La date de fin de la tâche J_j dans l'ordonnancement σ construit par l'algorithme *semi-online*.

— C_j^* : La date de fin de la tâche J_j dans l'ordonnancement σ^* construit par l'algorithme *offline* optimale.

— C_j^{**} : La date de fin de la tâche J_j dans l'ordonnancement σ^{**} construit par l'algorithme *offline* optimale SRPT (*Shortest Remaining Processing Time*) où les préemptions sont autorisées.

— $ALG(I) = \sum C_j(I)$: La fonction objectif obtenue de l'ordonnancement σ pour une instance I.

— $OPT(I) = \sum C_j^*(I)$: La fonction objectif obtenue de l'ordonnancement σ^* pour une instance I.

— $LB(I) = \sum C_j^{**}(I)$: La fonction objectif obtenue de l'ordonnancement σ^{**} pour une instance I.

— $\gamma = \frac{p_L}{p_F} < 1$: Le ratio de la borne inférieure du plus petit temps de traitement sur la borne supérieure du plus grand temps de traitement.

Considérons une instance pour le problème le problème semi-online étudié $1|online, r_j, p_{j+1} \leq \beta p_j, p_j \geq p_L| \sum C_j$. Nous pouvons trouver une borne supérieure sur le nombre de tâches dans cette instance en utilisant les deux informations connues. En effet, nous avons

$$p_2 \leq \beta p_F \tag{3.1}$$
$$p_3 \leq \beta p_2 \tag{3.2}$$
$$...$$
$$p_L \leq \beta p_{n-1} \tag{3.3}$$

Ce qui implique,

$$p_L \leq \beta^{n-1} p_F \tag{3.4}$$
$$\iff \beta^{n-1} \geq \gamma \tag{3.5}$$
$$\iff n \leq \frac{ln\,(\gamma)}{ln\,(\beta)} + 1 \tag{3.6}$$

En conséquence, le nombre maximale $\bar{n}$ de tâches dans une instance I est noté

$$\bar{n} = \left\lfloor \frac{ln\,(\gamma)}{ln\,(\beta)} + 1 \right\rfloor \tag{3.7}$$

3.3 L'algorithme ϕD-SPT

L'algorithme *semi-online* proposé, nommé ϕD-SPT (ϕ-*Delayed Shortest Processing Time*), est inspiré de l'algorithme *online* D-SPT introduit par Hoogeveen et Vestjens [42] où une tâche J_j avec un temps de traitement p_j ne peut être ordonnancée que si elle vérifie la condition $p_j \leq t$ avec t le temps de décision actuel. Dans le cas où cette condition n'est pas vérifiée, la tâche est retardée jusqu'à ce qu'elle vérifie la condition ou jusqu'à l'arrivée d'une autre tâche. Le nouvel algorithme ϕD-SPT permet d'insérer moins de temps d'attente par rapport à D-SPT, cela peut être justifié par la quantité d'informations connues sur l'instance à ordonnancer. En effet, l'algorithme *semi-online* n'a pas besoin d'être aussi prudent que l'algorithme *online* qui se trouve face à une incertitude totale. Les informations sur l'ordre d'arrivée des tâches et la valeur maximum et minimum des temps de traitement dans l'instance sont utilisées pour trouver l'expression idéale de ϕ, ce dernier est un paramètre de notre problème qui sera défini par la suite.

L'algorithme ϕD-SPT s'énonce comme suit :
Lorsque la machine est inactive et que certaines tâches sont disponibles, choisir la tâche J_j avec le plus petit temps de traitement p_j parmi les tâches disponibles. Ensuite, si

$$p_j \leq \phi t \tag{3.8}$$

Alors ordonnancer J_j ; sinon, attendre que l'inégalité soit vérifiée ou qu'une nouvelle tâche arrive.

3.4 Analyse de compétitivité de ϕD-SPT

Afin de prouver le ratio de compétitivité de ϕD-SPT, il est important de fournir la preuve de certaines propriétés dans la section suivante.

3.4.1 Réduction des temps de traitement

Lemme 2. Lorsqu'une tâche débute son traitement à un instant S selon ϕD-SPT suivi par $(n-1)$ tâches libérées à $S+\epsilon$, alors la diminution des temps de traitement des $(n-1)$

tâches ne peut qu'augmenter le rapport de la valeur objective obtenue par ϕD-SPT sur la valeur objective obtenue par l'algorithme *offline* optimal.

Démonstration. Considérons une instance I avec une tâche J_1 ayant un temps de traitement p_1 et une date d'arrivée r_1 dont le traitement commence à un instant S, selon ϕD-SPT, suivi de $(n-1)$ tâches, arrivées à $S+\epsilon$ avec des temps de traitement $p_j < p_1$ avec $j \in \{2, ..., n\}$ (Figure 3.1).

La valeur objective de l'ordonnancement obtenue par ϕD-SPT est :

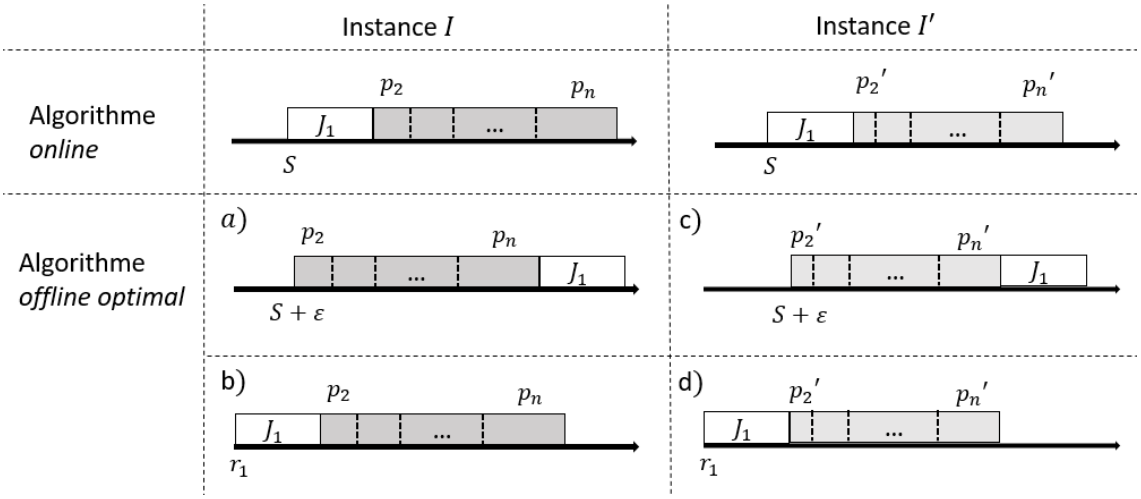

Figure 3.1 – Les ordonnancements par l'algorithme *semi-online* et l'algorithme *offline* optimal

$$ALG(I) = (S + p_1) + (S + p_1 + p_2) + ... + (S + p_1 + p_2 + ... + p_n) \tag{3.9}$$
$$= nS + np_1 + (n-1)p_2 + ... + p_n \tag{3.10}$$
$$= nS + np_1 + \sum_{j=1}^{n-1}(n-j)p_{j+1} \tag{3.11}$$

Nous avons $\sum_{j=1}^{n-1}(n-j)p_{j+1} = \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n} p_j$. Ainsi,

$$ALG(I) = nS + np_1 + \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n} p_j \tag{3.12}$$

La valeur objective de l'ordonnancement *offline* optimal peut être soit $OPT_a(I)$ soit $OPT_b(I)$, où $OPT_a(I)$ représente le cas où l'algorithme *offline* optimal décide de démarrer les $(n-1)$ tâches en premier suivi de J_1 (Figure 3.1 cas a), tandis que $OPT_b(I)$ représente le cas où il décide de programmer d'abord J_1 à r_1 puis les $(n-1)$ tâches (Figure 3.1 cas b). D'où,

$$\begin{aligned} OPT_a(I) &= (S+\epsilon+p_2)+...+(S+\epsilon+p_2+...+p_n) \\ &\quad +(S+\epsilon+p_2+...+p_n+p_1) && (3.13)\\ &= n(S+\epsilon)+p_1+np_2+...+2p_n && (3.14)\\ &= n(S+\epsilon)+p_1+\sum_{j=2}^{n}(n+2-j)p_j && (3.15) \end{aligned}$$

Et,

$$\begin{aligned} OPT_b(I) &= (r_1+p_1)+...+(r_1+p_1+...+p_n) && (3.16)\\ &= nr_1+np_1+(n-1)p_2+...+p_n && (3.17)\\ &= nr_1+np_1+\sum_{j=1}^{n-1}(n-j)p_{j+1} && (3.18)\\ &= nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p_j-\sum_{j=2}^{n}p_j && (3.19) \end{aligned}$$

Ainsi, le ratio peut être soit ρ_a soit ρ_b. Quand $\epsilon \to 0$, nous avons

$$\lim_{\epsilon\to 0}\rho_a = \frac{ALG(I)}{OPT_a(I)} = \frac{nS+np_1+\sum_{j=2}^{n}(n+2-j)p_j-\sum_{j=2}^{n}p_j}{nS+p_1+\sum_{j=2}^{n}(n+2-j)p_j} \quad (3.20)$$

Et,

$$\rho_b = \frac{ALG(I)}{OPT_b(I)} = \frac{nS+np_1+\sum_{j=2}^{n}(n+2-j)p_j-\sum_{j=2}^{n}p_j}{nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p_j-\sum_{j=2}^{n}p_j} \quad (3.21)$$

Dans la suite, les temps de traitement p_j pour $j=\{2,...,n\}$ sont réduits pour être ramenés à p'_j de sorte que $p'_j < p_j$ pour $j=\{2,...,n\}$. La nouvelle instance est désignée par I' (Figure 3.1) et les ratios résultants sont désignés par ρ_c et ρ_d. Les ordonnancements *offline* optimaux sont représentés dans la figure 3.1 cas c et d.

Il convient de noter qu'en diminuant les temps de traitement p_j pour $j = \{2, ..., n\}$, la solution optimale peut rester la même ou changer pour une autre solution avec une valeur objective plus petite. La condition pour laquelle $OPT_a(I) < OPT_b(I)$ peut être obtenue comme suit :

$$OPT_a(I) < OPT_b(I) \tag{3.22}$$

$$\iff n(S+\epsilon) + p_1 + \sum_{j=2}^{n}(n+2-j)p_j$$

$$< nr_1 + np_1 + \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n} p_j \tag{3.23}$$

$$\iff n(S+\epsilon) + p_1 < nr_1 + np_1 - \sum_{j=2}^{n} p_j \tag{3.24}$$

$$\iff (n-1)p_1 - n(S+\epsilon-r_1) > \sum_{j=2}^{n} p_j \tag{3.25}$$

Ainsi, $OPT_a(I)$ est meilleur que $OPT_b(I)$ lorsque $\sum_{j=2}^{n} p_j < (n-1)p_1 - n(S+\epsilon-r_1)$, ce qui signifie que dans le cas où $OPT_a(I)$ est la valeur objective de la solution optimale, la réduction des temps de traitement des $(n-1)$ tâches ne changera pas la solution optimale. Cependant, dans le cas où $OPT_b(I)$ est la valeur objective de la solution optimale, c'est-à-dire $\sum_{j=2}^{n} p_j \geq (n-1)p_1 - n(S+\epsilon-r_1)$, alors la réduction des temps de traitement pourrait changer la solution optimale.

En conséquence, nous devons comparer ρ_a à ρ_c et ρ_b à ρ_d pour lesquels l'ordre d'ordonnancement par l'algorithme *offline* optimal ne change pas de l'instance I à l'instance I'. De plus, lorsque la solution optimale est OPT_b, alors la solution optimale peut changer lorsque $\sum_{j=2}^{n} p'_j \leq (n-1)p_1 - n(S+\epsilon-r_1)$. Par conséquent, nous devons également comparer ρ_b à ρ_c et prouver que le ratio de compétitivité augmente.

Signe de $\rho_a - \rho_c$:

$$\rho_a - \rho_c = \frac{nS + np_1 + \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n} p_j}{nS + p_1 + \sum_{j=2}^{n}(n+2-j)p_j} - \frac{nS + np_1 + \sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n} p'_j}{nS + p_1 + \sum_{j=2}^{n}(n+2-j)p'_j} \tag{3.26}$$

$$= \frac{\left[\begin{array}{l}(nS+np_1)\left(\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}(n+2-j)p_j\right)\\ \quad +(nS+p_1)\left(\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}(n+2-j)p'_j\right)\\ \qquad\qquad +(nS+p_1)\left(\sum_{j=2}^{n}p'_j - \sum_{j=2}^{n}p_j\right)\end{array}\right]}{\left[\begin{array}{l}\left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p_j\right)\\ \quad \left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p'_j\right)\end{array}\right]} \tag{3.27}$$

Désignons par C une valeur positive, $C = \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}(n+2-j)p'_j$ puisque $p_j > p'_j$ pour $j = \{2, ..., n\}$. D'où,

$$\rho_a - \rho_c = \frac{\left[\begin{array}{l}-C\,(nS+np_1) + C\,(nS+p_1)\\ \quad +(nS+p_1)\left(\sum_{j=2}^{n}p'_j - \sum_{j=2}^{n}p_j\right)\end{array}\right]}{\left[\begin{array}{l}\left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p_j\right)\\ \quad \left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p'_j\right)\end{array}\right]} \tag{3.28}$$

$$= \frac{Cp_1(1-n) + (nS+p_1)\left(\sum_{j=2}^{n}p'_j - \sum_{j=2}^{n}p_j\right)}{\left[\begin{array}{l}\left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p_j\right)\\ \quad \left(nS+p_1+\sum_{j=2}^{n}(n+2-j)p'_j\right)\end{array}\right]} \tag{3.29}$$

Puisque $\sum_{j=2}^{n}p'_j - \sum_{j=2}^{n}p_j < 0$ et $n \geq 2$, nous avons le résultat souhaité suivant,

$$\rho_a - \rho_c < 0 \Longleftrightarrow \rho_a < \rho_c \tag{3.30}$$

Signe de $\rho_b - \rho_d$:

$$\rho_b - \rho_d = \frac{nS + np_1 + \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j}{nr_1 + np_1 + \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j} - \frac{nS + np_1 + \sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j}{nr_1 + np_1 + \sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j} \tag{3.31}$$

$$= \frac{\left[\begin{array}{l}(nS+np_1)\left(\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)\\ \quad -(nS+np_1)\left(\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j\right)\\ \quad +(nr_1+np_1)\left(\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j\right)\\ \quad\quad -(nr_1+np_1)\left(\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)\end{array}\right]}{\left[\begin{array}{l}\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j\right)\\ \quad\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)\end{array}\right]} \tag{3.32}$$

Désignons par $C' = \sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j - \left(\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)$ avec C' une valeur positive, puisque $p_j > p'_j$ pour $j = \{2, ..., n\}$. D'où,

$$\rho_b - \rho_d = \frac{-C'(nS+np_1) + C'(nr_1+np_1)}{\left[\begin{array}{l}\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j\right)\\ \quad\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)\end{array}\right]} \tag{3.33}$$

$$= \frac{C'n(r_1-S)}{\left[\begin{array}{l}\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p_j - \sum_{j=2}^{n}p_j\right)\\ \quad\left(nr_1+np_1+\sum_{j=2}^{n}(n+2-j)p'_j - \sum_{j=2}^{n}p'_j\right)\end{array}\right]} \tag{3.34}$$

Puisque $S \geq r_1$, nous avons le résultat souhaité suivant,

$$\rho_b \leq \rho_d \tag{3.35}$$

Signe de $\rho_b - \rho_c$:

Nous avons prouvé que $\rho_b \leq \rho_d$. Ainsi, $\frac{ALG(I)}{OPT_b(I)} \leq \frac{ALG(I')}{OPT_d(I')}$ pour tout p_j et p'_j, tel que $p'_j \leq p_j$ et $j \in \{2, ..., n\}$. De plus, quand la solution optimale pour l'instance I' est $OPT_c(I')$, nous avons :

$$OPT_c(I') \leq OPT_d(I') \Longleftrightarrow \frac{ALG(I')}{OPT_d(I')} \leq \frac{ALG(I')}{OPT_c(I')} \tag{3.36}$$

De (3.35) et (3.36), nous pouvons conclure que

$$\frac{ALG(I)}{OPT_b(I)} \leq \frac{ALG(I')}{OPT_c(I')} \Longleftrightarrow \rho_b \leq \rho_c \tag{3.37}$$

De (3.30), (3.35) et (3.37), nous pouvons conclure qu'en diminuant les temps de traitement p_j pour $j = \{2, ..., n\}$, le rapport de la valeur objective obtenue par l'algorithme *online* sur celle obtenue par l'algorithme *offline* optimal augmente.

□

3.4.2 La structure en blocs de l'ordonnancement par ϕD-SPT

L'ordonnancement de ϕD-SPT est composé de blocs l pour $l = \{1, ..., b\}$ avec b le nombre total de blocs (Figure 3.2). Dans chaque bloc, les tâches sont ordonnancées en continu, tandis que des temps d'inactivité séparent les blocs. Les temps d'inactivité sont soit dus à la stratégie d'attente implémentée par ϕD-SPT, soit parce que toutes les tâches arrivées ont déjà été terminées avant l'arrivée d'une nouvelle tâche. De plus, les tâches planifiées avant une période d'inactivité n'influencent pas la prise de décision concernant les tâches planifiées après une période d'inactivité, et vice versa. Par conséquent, nous pouvons diviser l'instance I en plusieurs instances plus petites I_l pour $l = \{1, ..., b\}$. Chacune se compose de tâches dans un bloc de $\sigma(I)$. Pour chacune de ces petites instances, ϕD-SPT garde les mêmes dates de début des tâches que dans $\sigma(I)$. Par conséquent, pour l'une de ces petites instances, le ratio de compétitivité n'est pas inférieur à celui de I [42].

$$\rho(I) \leq \max_{l \in \{1,...,b\}} \rho(I_l) \tag{3.38}$$

En conséquence, dans le pire des cas, nous pouvons considérer uniquement une de ces petites instances, noté I_0, pour laquelle l'ordonnancement par ϕD-SPT n'est composé que d'un seul bloc.

Notons la première date d'arrivée dans l'instance I_0 par r_0 et notons par S la date de début du bloc choisi par ϕD-SPT. Les tâches peuvent être davantage partitionnées en des sous-blocs de manière à avoir des tâches ordonnancées par SPT (*Shortest Processing Times*) dans chaque sous-bloc. Aussi, la dernière tâche dans un sous-bloc a un temps de traitement plus long que la première tâche du sous-bloc suivant. Notons ces sous-blocs par B_i, pour $i = \{1, ..., m\}$ et m le nombre total des sous-blocs. De plus, la dernière tâche d'un sous-bloc i est notée J_{B_i}, son temps de traitement est noté p_{B_i} et sa date de départ dans $\sigma(I_0)$ est notée S_{B_i} (Figure 3.3).

Trois ensembles de tâches de I_0 peuvent être définis selon leurs dates d'arrivées :

— Q^0_{BS} contient les tâches arrivant entre r_0 et S. Ces tâches ont manifestement des temps de traitement qui vérifient $p_j \geq \phi S$ car sinon elles auraient été ordonnancées avant S selon ϕD-SPT.

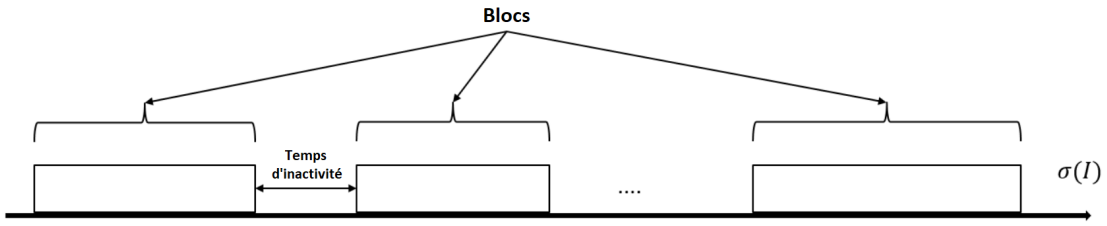

Figure 3.2 – Structure en blocs de $\sigma(I)$ par ϕD-SPT pour une instance I

— Q^0_S contient les tâches arrivant à S avec des temps de traitement qui vérifient $p_j < \phi S$, puisque toutes les tâches qui vérifient $p_j \geq \phi S$ sont inclues dans Q^0_{BS}.

— Q^0_{AS} contient le reste des tâches qui arrivent après S.

Il est à noter qu'à l'instant S, quand une tâche vérifie la condition de ϕD-SPT, alors toutes les tâches qui arrivent ensuite et qui sont inclues dans Q^0_{AS}, vérifieront la condition, puisque les tâches arrivent en ordre décroissant des temps de traitement.

$$I_0 \begin{cases} Q^0_{BS} = \{J_j | r_0 \leq r_j \leq S, \quad p_j \geq \phi S\} \\ Q^0_S = \{J_j | r_j = S, \quad p_j < \phi S\} \\ Q^0_{AS} = \{J_j | r_j > S\} \end{cases} \tag{3.39}$$

Selon les trois sous-ensembles définis dans l'équation (3.39) et la structure de $\sigma(I_0)$ représentée dans la Figure 3.3, il est à noter que la première tâche ordonnancée par ϕD-SPT appartient à l'un des deux sous-ensembles Q^0_{BS} ou Q^0_S.
Dans le cas où $Q^0_S = \emptyset$, notons cette instance par $I_{0,1}$. Les trois sous-ensembles seront définis ainsi :

$$I_{0,1} \begin{cases} Q^{0,1}_{BS} = Q^0_{BS} \\ Q^{0,1}_S = \emptyset \\ Q^{0,1}_{AS} = Q^0_{AS} \end{cases} \tag{3.40}$$

Pour cette instance, la première tâche ordonnancée par ϕD-SPT serait la dernière tâche arrivant parmi les tâches dans $Q_{BS}^{0,1}$.

Dans le cas où $Q_S^0 \neq \emptyset$, notons cette instance par $I_{0,2}$. Les trois sous-ensembles seront définis ainsi :

$$I_{0,2} \begin{cases} Q_{BS}^{0,2} = Q_{BS}^0 \\ Q_S^{0,2} = Q_S^0 \\ Q_{AS}^{0,2} = Q_{AS}^0 \end{cases} \tag{3.41}$$

Pour cette instance $I_{0,2}$, la première tâche ordonnancée par ϕD-SPT serait la dernière tâche arrivant parmi les tâches dans $Q_S^{0,2}$, puisque les tâches dans ce sous-ensemble ont des temps de traitement plus petits que ceux dans $Q_{BS}^{0,2}$.

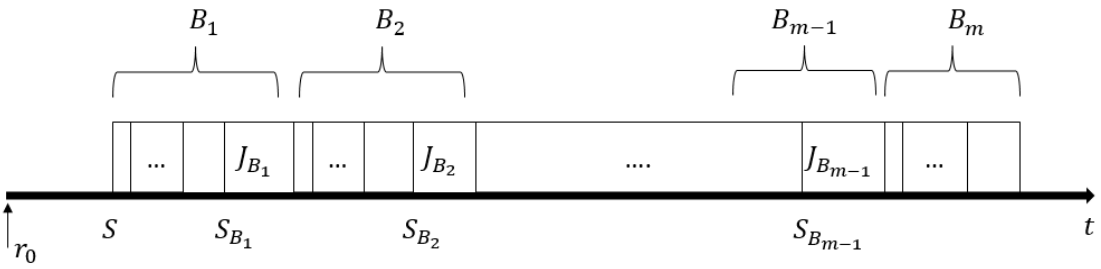

Figure 3.3 – Ordonnancement $\sigma(I_0)$ de l'instance I_0 par l'algorithme $\phi D - SPT$

Tout au long de l'analyse, le ratio de compétitivité est calculé par rapport à l'ordonnancement préemptif optimal puisque sa valeur objective est une borne inférieure sur la valeur objective pour l'ordonnancement non préemptif optimal. De plus, il peut être facilement obtenue par la règle SRPT (*Shortest Remaining Processing Time*). Pour une instance I, l'ordonnancement obtenu par SRPT est noté par $\sigma^{**}(I)$ et sa valeur objective par $LB(I)$. Nous avons :

$$\rho = \max_I \frac{ALG(I)}{OPT(I)} \leq \max_I \frac{ALG(I)}{LB(I)} \tag{3.42}$$

3.4.3 La méthode de transformation d'instances

L'analyse de compétitivité d'un algorithme est une analyse qui dépend du problème considéré. Ainsi, aucun cadre général n'existe dans la plupart des cas. Puisque le ratio de compétitivité est le ratio de compétitivité de la pire instance, pouvons-nous trouver la pire instance dans l'espace d'instances d'une manière appropriée ? Nous utilisons une méthode basée sur l'idée de transformation d'instance introduite par Tao et al. [87]. Elle commence par une instance arbitraire I du problème considéré, puis construit une nouvelle instance $I^{'}$ en modifiant I tel que la nouvelle instance a non seulement une structure plus particulière par rapport à I, mais satisfait également la relation :

$$\frac{ALG(I)}{LB(I)} \leq \max\left\{\frac{ALG(I^{'})}{LB(I^{'})}, F(\phi)\right\} \tag{3.43}$$

Avec $F(\phi)$ une fonction de ϕ.

La transformation que nous allons opérer ici consiste à modifier les dates d'arrivée des tâches et les temps de traitement de telle sorte que le ratio de compétitivité ne va qu'augmenter après la transformation. Après exécution de la transformation, étape par étape, l'instance obtenue va avoir une structure plus simple, ce qui va permettre d'analyser son ratio de compétitivité. De cette manière, nous allons obtenir une borne supérieure sur le ratio de compétitivité (CR). Les étapes de cette transformation sont illustrées à la Figure 3.4.

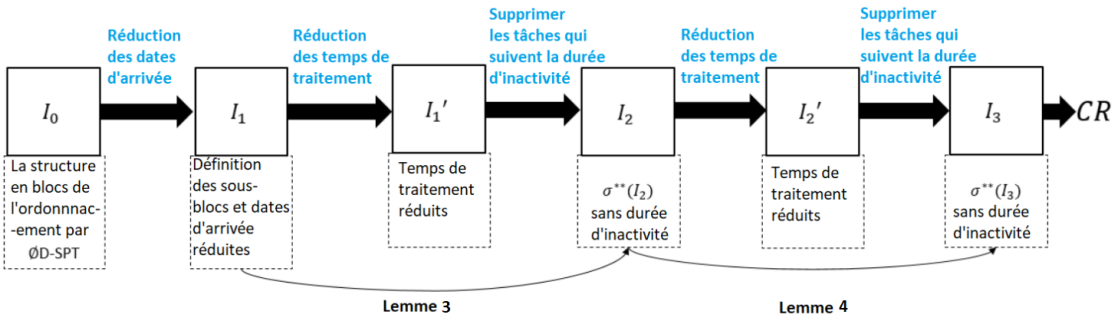

Figure 3.4 – Étapes de la transformation d'instance pour l'obtention du ratio de compétitivité de l'algorithme ϕD-SPT

Réduction des dates d'arrivée des tâches ($I_0 \rightarrow I_1$)

Dans cette section, nous modifions les dates d'arrivée de quelques tâches J_j. Notons I_1 la nouvelle instance et r_j^1 les nouvelles dates d'arrivée pour $j = \{1, ..., n\}$.

Première réduction : Pour les tâches dans Q_{BS}^0 de I_0, nous pouvons réduire leurs dates d'arrivée à r_0 puisqu'elles ne commencent pas avant S selon ϕD-SPT, c'est-à-dire que toutes ces tâches vérifient $p_j \geq \phi S$. En conséquent, cette réduction ne changera pas l'ordre d'ordonnancement des tâches dans $\sigma(I_0)$.

Seconde réduction : Pour les tâches dans Q_{AS}^0, nous pouvons réduire leurs dates d'arrivée de manière à ce que des tâches dans un sous-bloc B_i arrivent à $S_{B_{i-1}} + \epsilon$ pour $i = \{2, ..., m\}$ sans influencer l'ordre d'ordonnancement, où ϵ est une valeur positive infiniment petite. Par conséquence, il ne reste que $m - 1$ dates d'arrivée après S. De plus, nous désignons par A_i l'ensemble des tâches libérées à $S_{B_i} + \epsilon$.

Finalement et suite aux deux étapes de réduction appliquées, nous pouvons définir trois sous-ensembles de tâches en terme de leurs dates d'arrivée.

- Q_{BS}^1 contient les tâches qui arrivent à r_0 avec des temps de traitement qui vérifient $p_j \geq \phi S$.
- Q_S^1 contient les tâches qui arrivent à S vérifiant $p_j < \phi S$.
- Les tâches qui arrivent après S à $S_{B_i} + \epsilon$ pour $i = \{1, ..., m-1\}$ sont contenues dans le sous-ensemble $Q_{AS}^1 = \underset{i}{\cup} A_i$ avec $A_i = \{J_j^1 \quad avec \quad r_j^1 = S_{B_i} + \epsilon\}$.

Il faut noter que même après la procédure de réduction des dates d'arrivée des tâches, l'ordonnancement de I_1 présenté dans la figure 3.5 est le même que l'ordonnancement de I_0 présenté dans la figure 3.3, c'est-à-dire $\sigma(I_1) = \sigma(I_0)$.

$$I_1 \begin{cases} Q_{BS}^1 = \{J_j^1 \quad avec \quad r_j^1 = r_0 | r_0 \leq r_j \leq S, \qquad p_j \geq \phi S\} \\ Q_S^1 = Q_S^0 \\ Q_{AS}^1 = \{J_j^1 \quad avec \quad r_j^1 = S_{B_i} + \epsilon, i = \{1, ..., m-1\} | r_j > S\} \end{cases} \tag{3.44}$$

D'après les trois sous-ensembles de l'équation (3.44) et la structure de l'ordonnancement de I_1 représentée sur la figure 3.5, nous remarquons que $\sigma(I_1)$ débute avec des tâches dans Q^1_S si ce dernier n'est pas vide, sinon il commence par la dernière tâche libérée dans Q^1_{BS}. En outre, le premier sous-bloc contient certaines tâches de Q^1_{BS} et des tâches de Q^1_S. De plus, chaque sous-bloc B_i pour $i = \{2, ..., m-1\}$ contient également des tâches de Q^1_{AS} libérées à $S_{B_{i-1}} + \epsilon$ suivies d'une ou plusieurs tâches de Q^1_{BS}. Enfin, le dernier sous-bloc B_m contient les tâches de Q^1_{AS} libérées à $S_{B_{m-1}} + \epsilon$ suivies des tâches restantes de Q^1_{BS}.

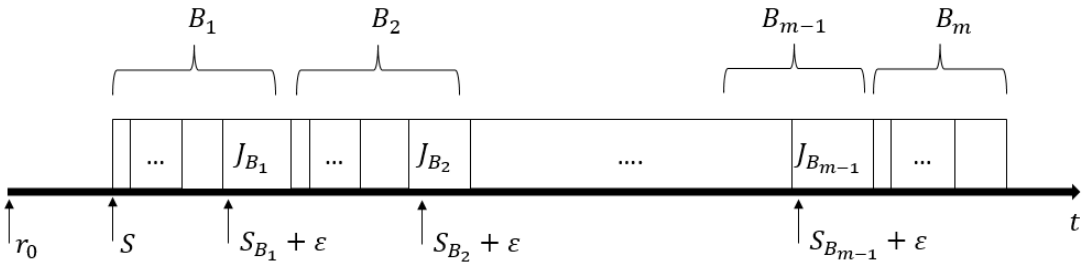

Figure 3.5 – L'ordonnancement $\sigma(I_1)$ de l'instance I_1 par l'algorithme $\phi D - SPT$

Par la suite, nous montrerons que l'instance I_1 peut être modifiée étape par étape pour former une nouvelle instance I_2, telle que I_2 a une structure plus spéciale avec un ratio de compétitivité supérieur ou égal au ratio de compétitivité pour I_1.

Réduction des temps de traitement des tâches dans Q^1_{AS} ($I_1 \rightarrow I'_1 \rightarrow I_2$)

Lemme 3. Une nouvelle instance, dénommée I_2, peut être construite en modifiant I_1, de telle sorte que Q^2_{AS} de I_2 ne contient que des tâches dont le temps de traitement est le plus court possible ou est vide. De plus,

$$\frac{ALG(I_1)}{LB(I_1)} \leq max\left\{\frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{1}{\phi}\right\} \tag{3.45}$$

Démonstration. Cette preuve est composée de deux étapes. La première étape est une procédure de réduction des temps de traitement des tâches où l'instance obtenue dénommée I'_1 garantit un ratio de compétitivité plus élevé que celui de l'instance I_1.

En outre, dans la deuxième étape, nous modifions l'instance I'_1 en se basant sur des caractéristiques de l'ordonnancement par SRPT, noté $\sigma^{**}(I'_1)$. L'instance obtenue est dénommée I_2 et vérifie l'inégalité (3.45).

Étape 1 : Réduction des temps de traitement ($I_1 \rightarrow I'_1$)

Nous appliquons d'abord la procédure de réduction des temps de traitement sur les tâches de l'ensemble A_{m-1} qui arrivent à $S_{B_{m-1}} + \epsilon$. Supposons que le nombre de tâches dans l'ensemble A_{m-1} est n_{m-1}. Nous désignons ces tâches dans un ordre décroissant de leurs temps de traitement par $J_{m-1,k}$ et des temps de traitement $p_{m-1,k}$ avec $k \in \{1, ..., n_{m-1}\}$, de sorte que $p_{m-1,1} > p_{m-1,2} > ... > p_{m-1,n_{m-1}}$. Notons que ces tâches sont ordonnées dans leur ordre décroissant en bloc B_m suivi des tâches restants de Q^1_{BS} (figure 3.6).
Pour ces tâches de A_{m-1} libérées à $S_{B_{m-1}} + \epsilon$, nous pouvons réduire leurs temps de traitement d'une valeur $\delta_{m-1,k}$ sans modifier leur ordre de traitement et sans diminuer le ratio de compétitivité. La procédure de réduction des temps de traitement est justifiée par le lemme 1 de la section 4.

Le choix de $\delta_{m-1,k}$:
Selon le lemme 1, le rapport de compétitivité est à son maximum lorsque les temps de traitement des tâches dans A_{m-1} sont à leur minimum tout en respectant les inégalités suivantes :

$$p_{m-1,2} \leq \beta p_{m-1,1} \tag{3.46}$$

$$...$$

$$p_{m-1,n_{m-1}} \leq \beta p_{m-1,n_{m-1}-1} \tag{3.47}$$

Nous pouvons modifier les temps de traitement des tâches dans A_{m-1} en retenant la borne inférieure pour chaque temps de traitement à partir de celui de la dernière tâche.

Par conséquent, pour appliquer cette transformation, nous ajoutons à chaque temps de traitement $p_{m-1,k}$ pour $k = \{1, ..., n_{m-1}\}$ la valeur suivante :

$$\delta_{m-1,k} = \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} - p_{m-1,k} \tag{3.48}$$

Pour chaque tâche J^1_j appartenant à l'ensemble A_{m-1}, le nouveau temps de traitement est $p^{1'}_j = \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}}$ avec k son rang dans l'ordre décroissant.

Les figures 3.6 et 3.7 montrent cette première transformation par réduction des temps de traitement des tâches dans A_{m-1}. Il convient de noter que ces tâches pourraient être suivies par un nombre de tâches, noté n', appartenant au sous-ensemble Q^1_{BS}. Ces tâches étaient disponibles mais ne pouvaient pas être programmées avant dans l'ordonnancement $\sigma(I_1)$.

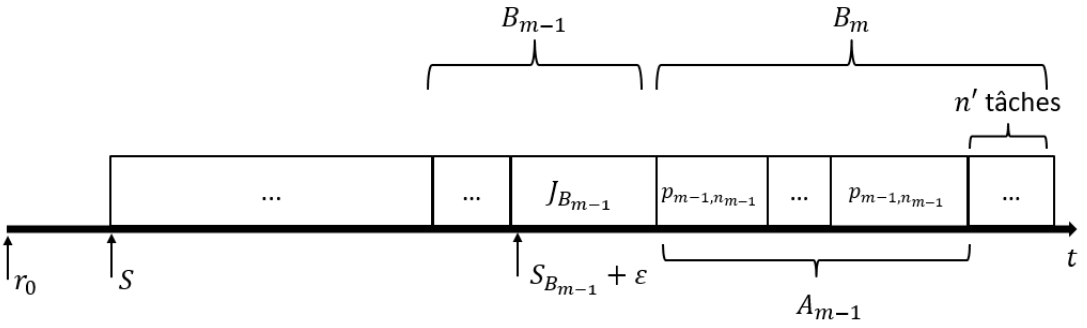

Figure 3.6 – Ordonnancement $\sigma(I_1)$ de l'instance I_1 avant réduction des temps de traitement des tâches lancées à $S_{B_{m-1}} + \epsilon$

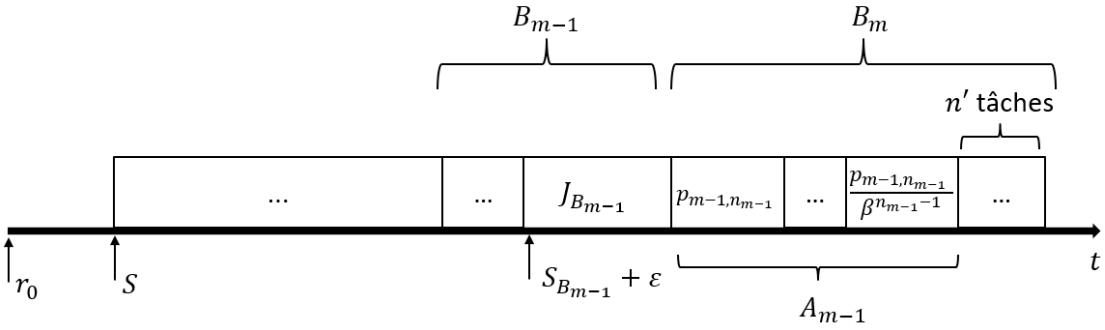

Figure 3.7 – Ordonnancement $\sigma(I_1)$ de l'instance I_1 suite à la réduction des temps de traitement des tâches lancées à $S_{B_{m-1}} + \epsilon$

Nous appliquons la même transformation aux tâches relevant des autres ensembles A_i pour $i = \{1, ..., m-2\}$ en diminuant les temps de traitement $p_{i,k}$, pour $k = \{1, ..., n_i\}$ et $i = \{1, ..., m-2\}$, avec une valeur $\delta_{i,k} = \frac{p_{i,n_i}}{\beta^{n_i-k}} - p_{i,k}$.

Les nouveaux sous-ensembles sont notés comme suit :

$$I_1' \begin{cases} Q_{BS}^{1'} = Q_{BS}^1 \\ Q_S^{1'} = Q_S^1 \\ Q_{AS}^{1'} = \left\{ \begin{array}{r} J_j^{1'} \ avec \ r_j^{1'} = r_j^1 \ et \ p_j^{1'} = \frac{p_{i,n_i}}{\beta^{n_i-k}}, i = \{1, ..., m-1\}, \\ k = \{1, ..., n_i\} | r_j^1 = S_{B_i} + \epsilon \quad p_j^1 = p_{i,k} \end{array} \right\} \end{cases} \tag{3.49}$$

Avant de passer à l'étape 2, nous devons définir certaines propriétés de l'ordonnancement $\sigma^{**}(I_1')$ générées par SRPT pour l'instance I_1'.

A) Les tâches sont ordonnancées à partir de r_0 et l'ordonnancement commence avec des tâches qui appartiennent à $Q_{BS}^{1'}$.

B) Des périodes d'inactivité peuvent se produire et ne sont dues qu'au fait que toutes les tâches disponibles sont terminées avant que de nouvelles tâches puissent arriver.

C) Après l'instant S, si une période d'inactivité existe, elle se terminera avec l'arrivée des tâches de $Q_{AS}^{1'}$. Celles-ci sont programmées dans le même ordre que dans $\sigma(I_1')$.

D) Désignons par S_{FI} la date de fin de la première période d'inactivité. Compte tenu de la structure de l'instance, nous pouvons conclure que des périodes d'inactivité se terminent à chaque $S_{B_k} + \epsilon$ pour $k > i$, avec i l'indice d'arrivée des tâches à la fin de la première période d'inactivité.

La dernière propriété peut être démontrée par induction. Supposons que la machine est inactive à $S_{B_k} + \epsilon$ pour $k > i$ dans $\sigma^{**}(I_1')$.
Notons par C_k et C_k^{**} la date de fin de la dernière tâche ordonnancée parmi les tâches libérées à $S_{B_k} + \epsilon$ en $\sigma(I_1')$ et en $\sigma^{**}(I_1')$, respectivement (Figure 3.8). Nous avons :

$$C_k = S_{B_{k+1}} \tag{3.50}$$

Ainsi,

$$C_k^{**} = C_k - p_{B_k} + \epsilon \tag{3.51}$$

En conséquent,

$$C_k^{**} < S_{B_{k+1}} + \epsilon \tag{3.52}$$

L'équation (3.52) implique qu'il y a également une période d'inactivité d'une durée $p_{B_k} - \epsilon$ précédant $S_{B_{k+1}} + \epsilon$. Ainsi, par induction, la machine se trouve inactive à la dernière date d'arrivée $S_{B_{m-1}} + \epsilon$.

Ensuite dans l'étape 2, nous vérifierons si, après les transformations ci-dessus, l'ordonnancement $\sigma^{**}(I_1')$ contient une ou plusieurs périodes d'inactivité. Si tel est le cas, nous considérons S_{FI} comme le moment où la première période d'inactivité se termine. De plus, nous appliquons l'étape suivante afin de supprimer les tâches planifiées après cette période d'inactivité. L'instance obtenue pour laquelle l'ordonnancement par SRPT ne contient aucune période d'inactivité est alors notée I_2.

Étape 2 : Si une période d'inactivité se termine à S_{FI} en $\sigma^{**}(I_1')$, nous supprimons toutes les tâches libérées à partir de cette période d'inactivité, c'est-à-dire des tâches arrivant à $S_{B_i} + \epsilon \geq S_{FI}$ pour $i = \{1, ..., m-1\}$ $(I_1' \rightarrow I_2)$

Afin de supprimer toutes les tâches libérées après la période d'inactivité, nous commençons par la dernière date d'arrivée $S_{B_{m-1}} + \epsilon$ qui est également précédée d'une période d'inactivité selon la propriété D. La figure 3.9 illustre le cas où une période d'inactivité existe juste avant la dernière date d'arrivée. Les n' tâches représentées dans la figure 3.9 sont toutes les tâches restantes du sous-ensemble $Q_{BS}^{1'}$ qui n'ont pas pu être ordonnancées avant.

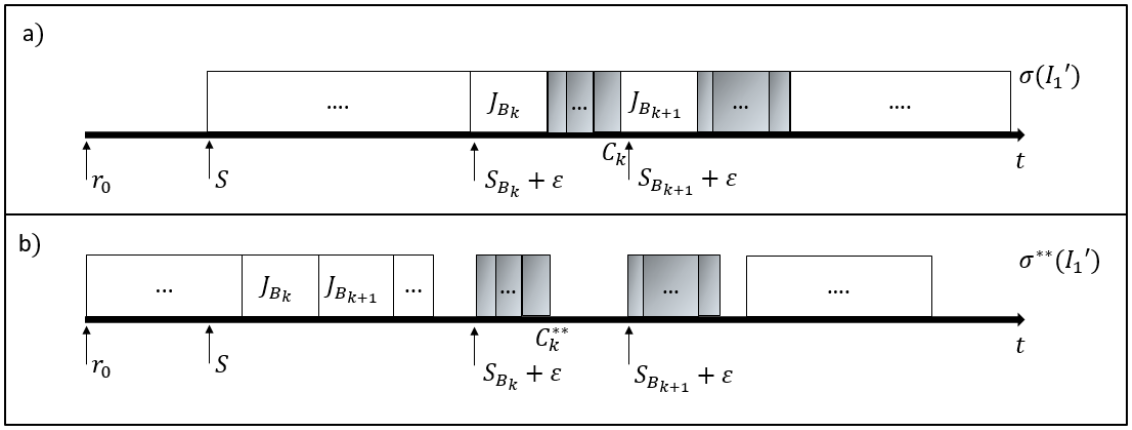

Figure 3.8 – Ordonnancements $\sigma(I_1')$ (a) et $\sigma^{**}(I_1')$ (b) avec des périodes d'inactivité dans $\sigma^{**}(I_1')$

La tâche $J_{B_{m-1}}$ dont le temps de traitement est $p_{B_{m-1}}$ ainsi que les n' tâches sont toutes ordonnancées avant $S_{B_{m-1}} + \epsilon$ dans l'ordonnancement $\sigma^{**}(I_1')$, ainsi les n_{m-1} tâches commenceront à être ordonnancées à partir de leur date d'arrivée. De plus, les

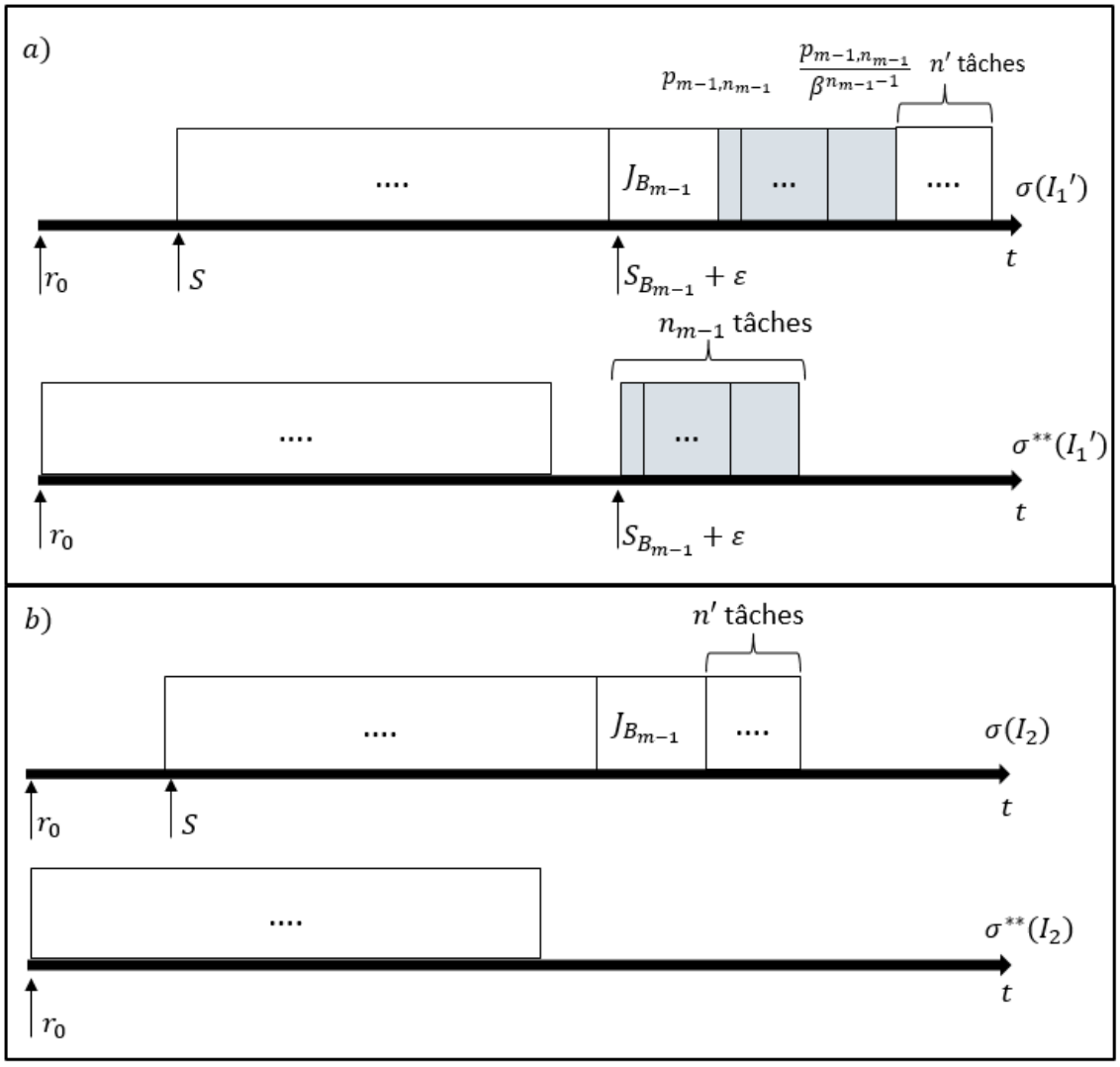

FIGURE 3.9 – Transformation de I_1' (a) vers I_2 (b) quand $\sigma^{**}(I_1')$ contient une période d'inactivité précédant la dernière date d'arrivée

n' tâches ont évidemment des temps de traitement plus longs que celle de $J_{B_{m-1}}$ car, sinon, elles auraient été choisies pour être ordonnancées en premier. Il convient donc de vérifier l'inégalité suivante

$$n' p_{B_{m-1}} + p_{B_{m-1}} \leq S - r_0 \tag{3.53}$$

Les tâches de $Q_{BS}^{1'}$ ont toutes un temps de traitement non inférieur à ϕS et doivent être

terminées avant $S_{B_{m-1}} + \epsilon$ dans $\sigma^{**}(I_{1'})$, donc cela implique

$$S_{B_{m-1}} - r_0 \geq \phi S \tag{3.54}$$

Ensuite, nous supprimons ces n_{m-1} tâches planifiées après la période d'inactivité dans $\sigma^{**}(I_{1'})$ et nous désignons l'instance intermédiaire par I_2. En combinant (3.53) et (3.54) avec quelques opérations algébriques de base, nous avons

$$\frac{ALG(I_1')}{LB(I_1')} = \frac{\left[\begin{array}{l} ALG(I_2) + n_{m-1}\left(S_{B_{m-1}} + p_{B_{m-1}}\right) \\ \quad + \sum_{k=1}^{n_{m-1}} k\frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} + n' \sum_{k=1}^{n_{m-1}} \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} \end{array}\right]}{LB(I_2) + n_{m-1}S_{B_{m-1}} + \sum_{k=1}^{n_{m-1}} k\frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}}} \tag{3.55}$$

$$\leq max\left\{ \frac{ALG(I_2)}{LB(I_2)}, \frac{\left[\begin{array}{l} n_{m-1}S_{B_{m-1}} + \sum_{k=1}^{n_{m-1}} k\frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} \\ \quad + n_{m-1}p_{B_{m-1}} + n' \sum_{k=1}^{n_{m-1}} \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} \end{array}\right]}{n_{m-1}S_{B_{m-1}} + \sum_{k=1}^{n_{m-1}} k\frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}}} \right\} \tag{3.56}$$

$$= max\left\{ \frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{n_{m-1}p_{B_{m-1}} + n' \sum_{k=1}^{n_{m-1}} \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}}}{n_{m-1}S_{B_{m-1}} + \sum_{k=1}^{n_{m-1}} k\frac{p_L}{\beta^{n_{m-1}-k}}} \right\} \tag{3.57}$$

$$\leq max\left\{ \frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{n_{m-1}p_{B_{m-1}} + n'n_{m-1}p_{B_{m-1}}}{n_{m-1}S_{B_{m-1}} + \sum_{k=1}^{n_{m-1}} k\frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}}} \right\} \tag{3.58}$$

$$\leq max\left\{ \frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{p_{B_{m-1}} + n'p_{B_{m-1}}}{S_{B_{m-1}}} \right\} \tag{3.59}$$

$$\leq max\left\{ \frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{S - r_0}{\phi S + r_0} \right\} \tag{3.60}$$

$$\leq max\left\{ \frac{ALG(I_2)}{LB(I_2)}, 1 + \frac{1}{\phi} \right\} \tag{3.61}$$

Où l'inégalité (3.58) est obtenue à partir de $\sum_{k=1}^{n_{m-1}} \frac{p_{m-1,n_{m-1}}}{\beta^{n_{m-1}-k}} \leq n_{m-1}p_{B_{m-1}}$. Dans le cas où il n'y a aucune période d'inactivité, la procédure ci-dessus n'est pas appliquée.

Après cette modification, A_{m-1} devient vide et $S_{B_{m-1}} + \epsilon$ n'est plus une date d'arrivée. De plus, nous appliquons la même modification à la nouvelle dernière date d'arrivée $S_{B_{m-2}} + \epsilon$. Cette procédure est répétée jusqu'à ce qu'il n'y ait plus de période d'inactivités dans l'ordonnancement par SRPT.

Enfin, et après ces étapes définies, Q^2_{AS} devient soit vide, soit ne contient que des tâches dont les temps de traitement sont les plus courts possibles, alors que l'inégalité (3.45) demeure valable dans les deux cas. La dernière instance pour laquelle l'ordonnancement par SRPT ne présente pas des périodes d'inactivité est dénommée I_2. La figure 3.10 illustre la structure de $\sigma(I_2)$ qui a la même structure que $\sigma(I_1)$ sauf que $\sigma(I_2)$ contient éventuellement moins de sous-blocs s'il existe une première période d'inactivité qui se termine à S_{FI} dans l'ordonnancement par SRPT.

$$I_2 \begin{cases} Q^2_{BS} = Q^{1'}_{BS} \\ Q^2_S = Q^{1'}_S \\ Q^2_{AS} = Q^{1'}_{AS} \setminus \{J_j^{1'} \in Q^{1'}_{AS} \quad avec \quad r_j^{1'} \geq S_{FI}\} \end{cases} \tag{3.62}$$

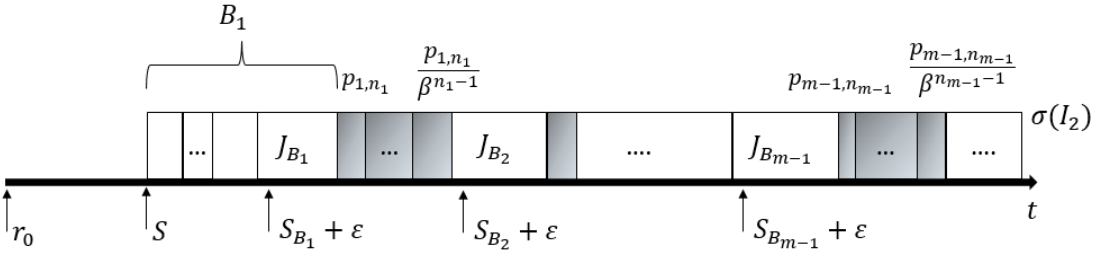

Figure 3.10 – Ordonnancement $\sigma(I_2)$ pour l'instance I_2

□

Réduction des temps de traitement des tâches dans Q^2_S ($I_2 \rightarrow I'_2 \rightarrow I_3$)

Lemme 4. Une nouvelle instance, dénommée I_3, peut être construite en modifiant I_2, de telle sorte que Q^3_S de I_3 ne contient que des tâches dont le temps de traitement est le plus court possible ou est vide. De plus,

$$\frac{ALG(I_2)}{LB(I_2)} \leq max\left\{\frac{ALG(I_3)}{LB(I_3)}, 1+\frac{1}{\phi}\right\} \tag{3.63}$$

Démonstration. La preuve est la même que pour le lemme 3. Pour les n_S tâches libérées à S, nous réduisons leurs temps de traitement $p_{S,k}$ par une valeur $\delta_{S,k}$; pour $k = \{1, ..., n_S\}$, l'instance obtenue est dénommée I_2'.

Le choix de $\delta_{S,k}$:
Selon le lemme 2, le ratio de compétitivité est à son maximum lorsque les temps de traitement des n_S tâches libérées à S sont au minimum tout en respectant les inégalités suivantes :

$$p_{S,2} \leq \beta p_{S,1} \tag{3.64}$$

$$...$$

$$p_{S,n_S} \leq \beta p_{S,n_S-1} \tag{3.65}$$

Nous pouvons modifier les temps de traitement des n_S tâches en retenant la borne inférieure de chaque temps de traitement à partir de la dernière tâche.

Par conséquence, nous ajoutons à chaque temps de traitement $p_{S,k}$ pour $k = \{1, ..., n_S\}$ la valeur suivante :

$$\delta_{S,k} = \frac{p_{S,n_S}}{\beta^{n_S-k}} - p_{S,k} \tag{3.66}$$

Pour chaque tâche J_j^2 appartenant à l'ensemble Q_S^2, le nouveau temps de traitement est $p_j^{2'} = \frac{p_{S,n_S}}{\beta^{n_S-k}}$ avec k son rang dans l'ordre décroissant.

Les nouveaux sous-ensembles relatifs à I_2' sont désignés comme suit :

$$I_2^{'} \begin{cases} Q_{BS}^{2'} = Q_{BS}^2 \\ Q_S^{2'} = \left\{ \begin{array}{cc} J_j^{2'} \quad avec \quad r_j^{2'} = r_j^2 \quad p_j^{2'} = \frac{p_{S,n_S}}{\beta^{n_S-k}}, \\ k = \{1, ..., n_S\} | r_j^2 = S \quad p_j^2 = p_{S,k} \end{array} \right\} \\ Q_{AS}^{2'} = Q_{AS}^2 \end{cases} \tag{3.67}$$

Comme pour l'étape 2 du lemme 3, nous vérifions si certaines périodes d'inactivité apparaissent dans l'ordonnancement par SRPT de l'instance I_2'. Si c'est le cas, nous supprimons les tâches de $Q_{AS}^{2'}$ libérées après la première période d'inactivité qui se termine à une date d'arrivée dénommée S_{FI}' dans $\sigma^{**}(I_2')$. L'instance obtenue pour laquelle les tâches sont planifiées en continu par SRPT est dénommée I_3. Les sous-ensembles de

I_3 sont les mêmes que ceux de I'_2, sauf qu'il est possible de se retrouver avec à la fois Q^3_S et Q^3_{AS} vides s'il y a une période d'inactivité qui se termine à S dans l'ordonnancement par SRPT, ou moins de sous-blocs dans Q^3_{AS} s'il y a une période d'inactivité qui se termine à S'_{FI} dans $\sigma^{**}(I'_2)$. Nous avons

$$I_3 \begin{cases} Q^3_{BS} = Q^{3'}_{BS} \\ Q^3_S = Q^{3'}_S \\ Q^3_{AS} = Q^{3'}_{AS} \setminus \{J^{1'}_j \in Q^{3'}_{AS} \quad avec \quad r^{3'}_j \geq S'_{FI}\} \end{cases} \tag{3.68}$$

La figure 3.11 illustre la structure de $\sigma(I_3)$. Elle débute avec des tâches de Q^3_S à l'instant S, si ce dernier n'est pas vide, puis alterne entre des tâches de Q^3_{BS} et Q^3_{AS}. Dans $\sigma^{**}(I_3)$, les tâches sont exécutées en continu à partir de r_0. En outre, les tâches de Q^3_{BS} conservent le même ordre de traitement que dans $\sigma(I_3)$, tandis que certaines d'entre elles peuvent être préemptées par des tâches de Q^3_S et Q^3_{AS}.

□

Remarque. Dans les démonstrations précédentes, si certaines périodes d'inactivité apparaissent dans l'ordonnancement par ϕD-SPT suite à la réduction des temps de traitement, nous pouvons obtenir une instance plus petite avec un ratio de compétitivité plus élevé. Par conséquent, nous supposons que les tâches sont toujours programmées en continu pendant les ajustements de l'ordonnancement par ϕD-SPT [85].

3.4.4 Le ratio de compétitivité de ϕD-SPT

Les deux ordonnancements $\sigma(I_3)$ et $\sigma^{**}(I_3)$ sont illustrés dans la figure 3.11. Le ratio de compétitivité peut être obtenu en comparant les dates de fin des tâches dans les deux ordonnancements.

Lemme 5. Pour l'instance I_3,

$$\frac{ALG(I_3)}{LB(I_3)} \leq max \left\{ 1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}} \right\} \tag{3.69}$$

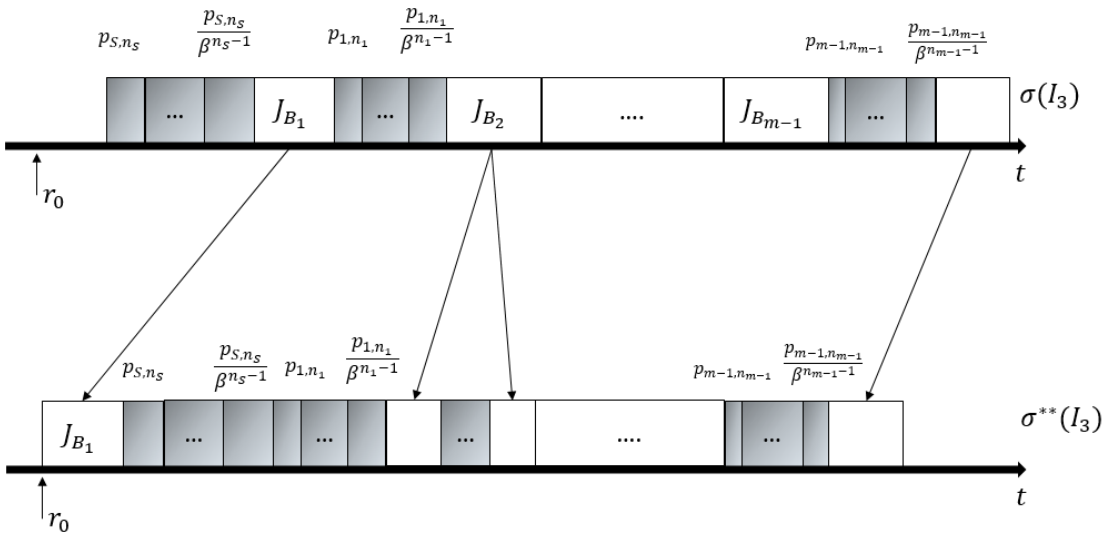

Figure 3.11 – Ordonnancement $\sigma(I_3)$ de l'instance I_3 par l'algorithme $\phi D - SPT$

Démonstration. Désignons l'ensemble des tâches de I_3 par Q^3, et désignons un de ses sous-ensembles par F. Pour chaque décomposition D de l'ensemble Q^3 en sous-ensembles disjonctifs [87], nous avons

$$\frac{ALG(I_3)}{LB(I_3)} \leq \max_{F \subset D} \left\{ \frac{\sum_{J_j \in F} C_j}{\sum_{J_j \in F} C_j^{**}} \right\} \tag{3.70}$$

Selon l'équation 3.70, nous pouvons définir trois sous-ensembles de l'ensemble Q^3 contenant des tâches de I_3. L'ensemble Q^3 contient toutes les tâches de l'instance I_3, c'est-à-dire $Q^3 = Q^3_{BS} \cup Q^3_S \cup Q^3_{AS}$. Toutefois, dans ce qui suit, nous définirons de nouveaux sous-ensembles disjonctifs de Q^3.

Nous allons désormais décomposer Q^3 en plusieurs sous-ensembles disjonctifs et comparer la somme des dates de fin des tâches dans chaque sous-ensemble. Cette décomposition prend en considération toutes les possibilités d'ordonnancement pour l'instance I_3 par SRPT. En d'autres termes, chaque structure de planification des tâches dans $\sigma^{**}(I_3)$ peut être identifiée dans l'un des trois sous-ensembles. Le premier sous-ensemble, désigné par F', contient J_{B_1}, les n_S tâches restantes de Q^3_S et les n_1 tâches de A_1. Pour ce sous-ensemble, SRPT devrait produire l'une des possibilités suivantes :

A) J_{B_1} n'est pas préemptée par les n_S tâches de Q_S^3 (figure 3.12-A). Cela signifie que le temps de traitement restant de J_{B_1} à l'instant S est inférieur ou égal au temps de traitement p_{S,n_S} de la dernière tâche libérée dans Q_S^3.

B) J_{B_1} n'est pas préemptée par les n_S tâches de Q_S^3 mais la dernière tâche ordonnancée de Q_S^3 est préemptée par les n_1 tâches de A_1 (figure 3.12-B). La non-préemption de J_{B_1} signifie que son temps de traitement restant à S, qui peut être défini comme $p_{B_1} - S$, est inférieur ou égal à p_{S,n_S}. En outre, étant donné que les tâches de Q_S^3 sont retardées d'une période $p_{B_1} - S$ dans l'ordonnancement par SRPT, alors au maximum la première tâche libérée de Q_S^3 pourrait être préemptée par les tâches de A_1 puisque le temps de retard après $S_{B_1} + \epsilon$ est inférieur ou égal à p_{S,n_S}.

C) J_{B_1} n'est pas préemptée par les n_S tâches de Q_S^3 mais la dernière tâche de Q_S^3 est préemptée par les $n_1 - n_{B'}$ tâches de A_1 (figure 3.12-C). Cela signifie que les $n_{B'}$ tâches de A_1 ont un temps de traitement supérieur au temps de traitement restant de la dernière tâche de Q_S^3.

D) J_{B_1} est préemptée par les n_S tâches de Q_S^3 et donc elle est également préemptée par les n_1 tâches de A_1 puisque les tâches arrivent dans un ordre décroissant des temps de traitement (figure 3.13-D).

E) J_{B_1} est préemptée et certaines n_B tâches de Q_S^3 ont des temps de traitement supérieurs au temps de traitement restant de J_{B_1}. Ainsi, ces n_B tâches contenues dans un ensemble, noté B, seront ordonnancées par SRPT après avoir terminé J_{B_1} (figure 3.13-E). Cela représente le cas où le temps de traitement restant de J_{B_1} est supérieur aux temps de traitement de certaines tâches de Q_S^3 alors qu'il est inférieur ou égal aux temps de traitement des n_B tâches restantes de Q_S^3 qui font partie de l'ensemble B.

F) J_{B_1} est préemptée et certaines n_B tâches de Q_S^3 ont un temps de traitement supérieur au temps de traitement restant de J_{B_1}. Ainsi, ces n_B tâches qui sont contenues dans un ensemble, noté B, seront ordonnancées par SRPT après avoir terminé J_{B_1}, tandis que la dernière tâche parmi les tâches de B est préemptée par les tâches de A_1 et est ordonnancée en dernier (figure 3.13-F). Cela représente le même cas que E, sauf que le temps de traitement restant de la dernière tâche en B à l'instant $S_{B_1} + \epsilon$ est supérieur aux temps de traitement des tâches en A_1.

G) Ce cas, noté G, représente la même situation que le cas F, sauf que la dernière tâche de Q_S^3 qui est préemptée par les n_1 tâches de A_1 est maintenant préemptée seulement par $n_1 - n_{B'}$ tâches de A_1 (figure 3.13-G).

Les scénarios décrits plus haut représentent tous les cas possibles de planification des tâches en F' par SRPT. Les autres combinaisons ne sont pas prises en compte car elles sont impossibles. Nous pouvons annoncer les combinaisons irréalisables suivantes :

— Cas B, C, F et G : Nous avons considéré dans ces cas qu'une seule tâche de Q_S^3, qui est la dernière ordonnancée, est préemptée par des tâches de A_1. En effet, lorsque J_{B_1} n'est pas préemptée, les tâches de Q_S^3 sont retardées d'au plus p_{S,n_S} puisque le temps de traitement restant de J_{B_1} à l'instant S est inférieur à celui de la première tâche ordonnancée parmi les tâches de Q_S^3. Ainsi, une seule tâche peut être préemptée parmi les tâches de Q_S^3. En outre, lorsque J_{B_1} est préemptée par certaines tâches de Q_S^3, le temps de retard est inférieur au temps de traitement de la dernière tâche ordonnancée de Q_S^3, ce qui signifie également que seule cette tâche peut être préemptée par des tâches de A_1.

— Cas D : nous avons considéré ici que lorsque J_{B_1} est préemptée par toutes les tâches de Q_S^3 alors elle est également préemptée par toutes les tâches de A_1. En fait, étant donné que les tâches de A_1 ont des temps de traitement plus courts que celles de Q_S^3, si le temps de traitement restant de J_{B_1} est supérieur aux temps de traitement des tâches de Q_S^3, il est alors clairement supérieur aux temps de traitement des tâches de A_1. Par conséquent, le cas où J_{B_1} finit l'exécution immédiatement après les tâches dans Q_S^3 n'est pas possible.

Si une tâche J_{B_i} du sous-bloc B_i (à l'exclusion de J_{B_1}) est préemptée par des tâches de A_i dans $\sigma^{**}(I_3)$, nous construisons un sous-ensemble, noté F_i, qui contient J_{B_i} et toutes les tâches de A_i. Pour toutes les tâches restantes dans I_3, nous les mettions dans un sous-ensemble, noté F''. Cet ensemble contient alors les tâches J_{B_i} qui ne sont pas préemptées et les tâches de A_i qui ne préemptent pas d'autres tâches. Les figures 3.14-A$'$ et 3.14-B$'$ illustrent les cas où J_{B_2} et A_2 sont inclus dans l'ensemble F_2, tandis que la figure 3.14-C$'$ illustre le cas où J_{B_2} et A_2 sont inclus dans l'ensemble F''.

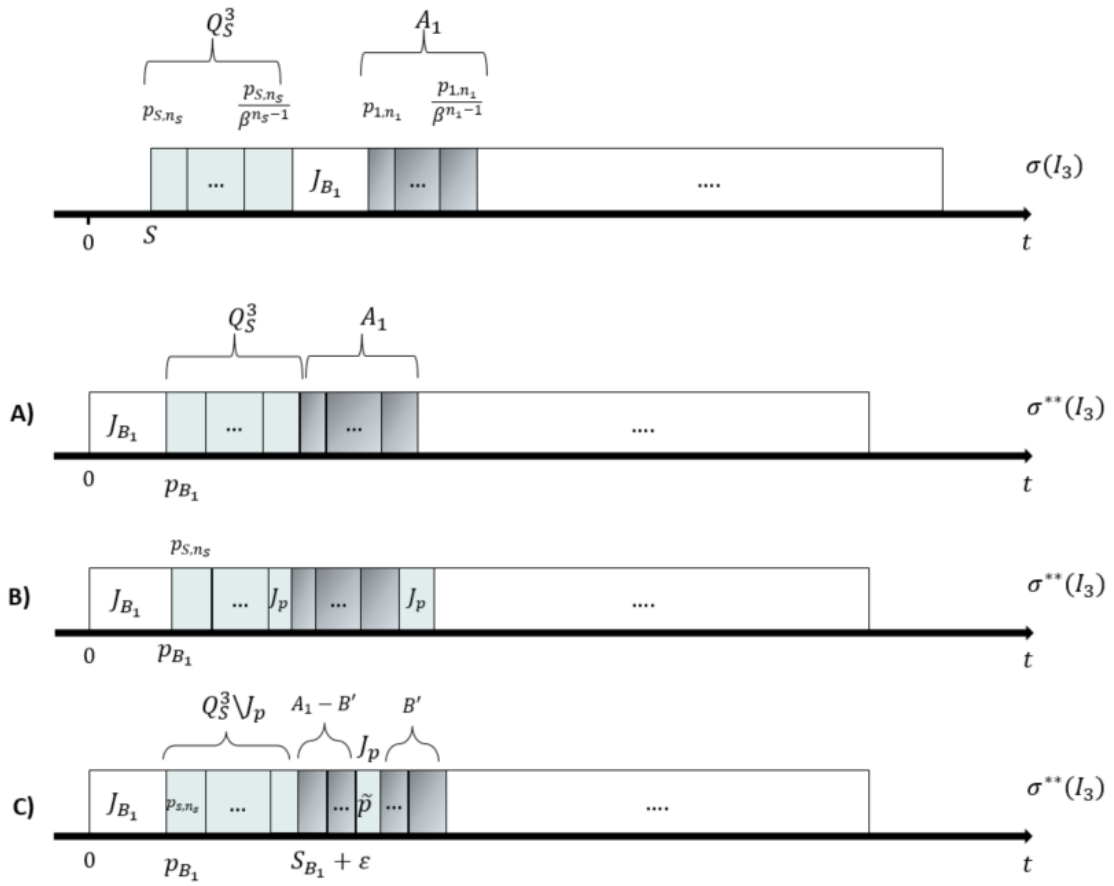

Figure 3.12 – Les ordonnancements $\sigma(I_3)$ et $\sigma^{**}(I_3)$ avec trois possibilités d'ordonnancement (A), (B) et (C) des tâches de F' par SRPT lorsque J_{B_1} n'est pas préemptée

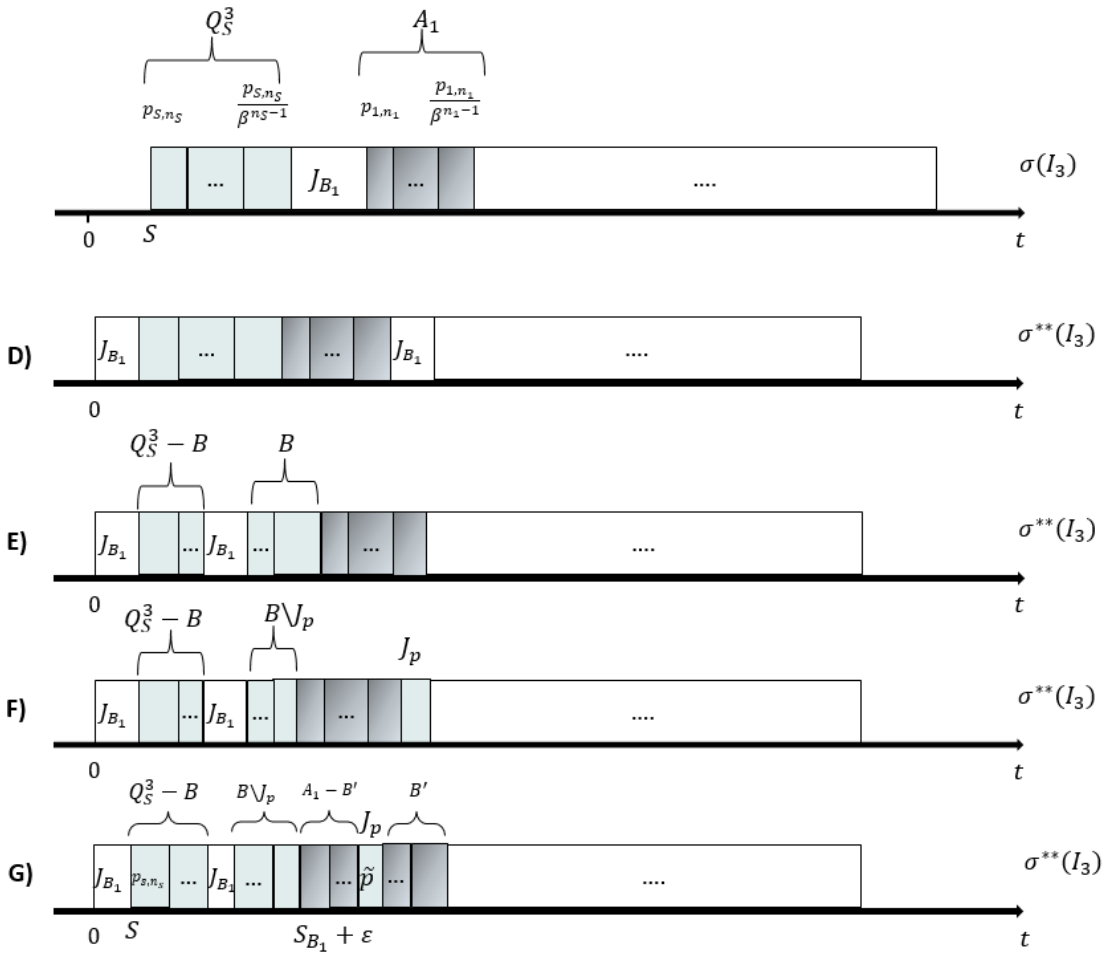

Figure 3.13 – Les ordonnancements $\sigma(I_3)$ et $\sigma^{**}(I_3)$ avec quatre possibilités d'ordonnancement (D), (E), (F) et (G) des tâches de F' par SRPT lorsque J_{B_1} est préemptée

Le sous-ensemble $F' = \{J_{B_1}\} \cup Q_S^3 \cup A_1$

A) J_{B_1} n'est pas préemptée par les tâches de Q_S^3 dans l'ordonnancement $\sigma^{**}(I_3)$ (Figure 3.12-A).

Nous avons

$$C_1^{**} = p_{B_1} \tag{3.71}$$

$$C_1 = C_1^{**} + S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} \tag{3.72}$$

$$C_j = C_j^{**} + S - p_{B_1} \quad \forall J_j \in Q_S^3 \tag{3.73}$$

$$C_j = C_j^{**} + S \quad \forall J_j \in A_1 \tag{3.74}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} n'est pas préemptée.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{\left[\begin{array}{c} C_1^{**} + S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} \\ + \sum_{J_j \in Q_S^3} \left(C_j^{**} + S - p_{B_1}\right) + \sum_{J_j \in A_1} \left(C_j^{**} + S\right) \end{array}\right]}{C_1^{**} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.75}$$

$$= 1 + \frac{S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} + n_S \left(S - p_{B_1}\right) + \sum_{J_j \in A_1} S}{p_{B_1} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.76}$$

$$\leq 1 + \max\left\{ \frac{S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} + n_S \left(S - p_{B_1}\right)}{p_{B_1} + \sum_{J_j \in Q_S^3} C_j^{**}}, \max_{J_j \in A_1} \frac{S}{C_j^{**}} \right\} \tag{3.77}$$

$$\leq 1 + \max\left\{ \frac{S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} + n_S \left(S - p_{B_1}\right)}{p_{B_1} + n_S p_{B_1} + \sum_{k=1}^{n_S} k \frac{p_{S,n_S}}{\beta^{n_S-k}}}, \frac{S}{p_{B_1}} \right\} \tag{3.78}$$

$$\leq 1 + \max\left\{ \frac{S + n_S S}{p_{B_1} + n_S p_{B_1}}, \frac{S}{p_{B_1}} \right\} \tag{3.79}$$

$$\leq 1 + \frac{1}{\phi} \tag{3.80}$$

Où l'inégalité (3.78) est due au fait que les tâches de A_1 ne commencent pas avant p_{B_1} en $\sigma^{**}(I_3)$, et l'inégalité (3.79) est due à $n_S p_{B_1} > \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}}$, alors que l'inégalité (3.80) est due à $p_{B_1} \geq \phi S$, d'où la valeur maximale que S peut atteindre est $\frac{p_{B_1}}{\phi}$.

B) J_{B_1} n'est pas préemptée par les tâches de Q_S^3, mais la dernière tâche de Q_S^3, dénommée J_p, est préemptée par les n_1 tâches de A_1 (Figure 3.12-B).

Nous avons

$$C_1 = C_1^{**} + S + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} \tag{3.81}$$

$$C_j = C_j^{**} - (p_{B_1} - S) \quad \forall J_j \in Q_S^3 \backslash J_p \tag{3.82}$$

$$C_p = C_p^{**} - (p_{B_1} - S) - \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \quad J_p \tag{3.83}$$

$$C_j = C_j^{**} + p_{B_1} \quad \forall J_j \in A_1 \tag{3.84}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} n'est pas préemptée mais que la dernière tâche de Q_S^3 est préemptée par toutes les n_1 tâches de A_1.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{\left[\begin{array}{r} C_1^{**} + S + \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} \\ + \sum_{J_j \in Q_S^3 \backslash J_p} \left(C_j^{**} - (p_{B_1} - S)\right) + C_p^{**} \\ - (p_{B_1} - S) - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ + \sum_{J_j \in A_1} \left(C_j^{**} + p_{B_1}\right) \end{array}\right]}{C_1^{**} + \sum_{J_j \in Q_S^3 \backslash J_p} C_j^{**} + C_p^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.85}$$

$$= 1 + \frac{\left[\begin{array}{r} S + \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} - \sum_{J_j \in Q_S^3 \backslash J_p} (p_{B_1} - S) \\ - (p_{B_1} - S) - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 p_{B_1} \end{array}\right]}{C_1^{**} + \sum_{J_j \in Q_S^3 \backslash J_p} C_j^{**} + C_p^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.86}$$

$$= 1 + \frac{\left[\begin{array}{r} S + n_S S + n_1 p_{B_1} + \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} \\ - n_S p_{B_1} - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \end{array}\right]}{\left[\begin{array}{r} p_{B_1} + n_S p_{B_1} + \sum_{J_j \in Q_S^3 \backslash J_p} k \frac{p_{S,n_S}}{\beta^{n_S-k}} + \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} \\ + \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 S_{B_1} + \sum_{J_j \in A_1} k \frac{p_{1,n_1}}{\beta^{n_1-k}} \end{array}\right]} \tag{3.87}$$

$$\leq \max\left\{1+\frac{S}{p_{B_1}}, 1+\frac{\left[\begin{array}{r} n_1 p_{B_1}+\sum_{J_j\in Q_S^3}\frac{p_{S,n_S}}{\beta^{n_S-k}} \\ -n_S p_{B_1}-\sum_{J_j\in A_1}\frac{p_{1,n_1}}{\beta^{n_1-k}}\end{array}\right]}{\left[\begin{array}{r}\sum_{J_j\in Q_S^3\setminus J_p} k\frac{p_{S,n_S}}{\beta^{n_S-k}} \\ +\sum_{J_j\in Q_S^3}\frac{p_{S,n_S}}{\beta^{n_S-k}}+\sum_{J_j\in A_1}\frac{p_{1,n_1}}{\beta^{n_1-k}} \\ +n_1 S_{B_1}+\sum_{J_j\in A_1} k\frac{p_{1,n_1}}{\beta^{n_1-k}}\end{array}\right]}\right\} \tag{3.88}$$

$$\leq \max\left\{1+\frac{1}{\phi}, 1+\frac{n_1-\frac{p_{1,n_1}}{p_{B_1}}\sum_{J_j\in A_1}\beta^{k-n_1}}{\left[\begin{array}{r} n_1\frac{S_{B_1}}{p_{B_1}}+\frac{p_{1,n_1}}{p_{B_1}}\sum_{J_j\in A_1}\beta^{k-n_1} \\ +\frac{p_{1,n_1}}{p_{B_1}}\sum_{J_j\in A_1} k\beta^{n_1-k}\end{array}\right]}\right\} \tag{3.89}$$

$$\leq \max\left\{1+\frac{1}{\phi}, 1+\frac{\bar{n}-1-\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi}+\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}\right\} \tag{3.90}$$

Où l'inégalité (3.89) est due au fait que $\sum_{J_j\in Q_S^3}\frac{p_{S,n_S}}{\beta^{n_S-k}} - n_S p_{B_1} \leq 0$ et $p_{B_1} \geq \phi S$, et l'inégalité (3.90) est due à $p_{B_1} \geq \phi S_{B_1}$.

C) J_{B_1} n'est pas préemptée par les tâches de Q_S^3, mais la dernière tâche de Q_S^3, dénommée J_p, est préemptée par les $n_1 - n_{B'}$ tâches de A_1 (Figure 3.12-C).

Nous avons,

$$C_1 = C_1^{**} + S + \sum_{J_j\in Q_S^3}\frac{p_{S,n_S}}{\beta^{n_S-k}} \tag{3.91}$$

$$C_j = C_j^{**} - (p_{B_1} - S) \quad \forall J_j \in Q_S^3\setminus J_p \tag{3.92}$$

$$C_p = C_p^{**} - (p_{B_1} - S) - \sum_{J_j\in A_1 - B'}\frac{p_{1,n_1}}{\beta^{n_1-k}} \quad J_p \tag{3.93}$$

$$C_j = C_j^{**} + p_{B_1} \quad \forall J_j \in A_1 - B' \tag{3.94}$$

$$C_j = C_j^{**} + p_{B_1} - (p_{B_1} - S) = C_j^{**} + S \quad \forall J_j \in B' \tag{3.95}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} n'est pas préemptée mais que la dernière tâche de Q_S^3 est préemptée par $n_1 - n_{B'}$ tâches de A_1.

$$\frac{\sum_{J_j\in F'} C_j}{\sum_{J_j\in F'} C_j^{**}} = \frac{\begin{bmatrix} C_1^{**} + S + \sum_{J_j\in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} + \sum_{J_j\in Q_S^3\setminus J_p} \left(C_j^{**} - (p_{B_1} - S)\right) \\ + C_p^{**} - (p_{B_1} - S) - \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ + \sum_{J_j\in A_1-B'} \left(C_j^{**} + p_{B_1}\right) + \sum_{J_j\in B'} \left(C_j^{**} + S\right) \end{bmatrix}}{C_1^{**} + \sum_{J_j\in Q_S^3\setminus J_p} C_j^{**} + C_p^{**} + \sum_{J_j\in A_1-B'} C_j^{**} + \sum_{J_j\in B'} C_j^{**}} \tag{3.96}$$

$$= 1 + \frac{\begin{bmatrix} S + \sum_{J_j\in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} - \sum_{J_j\in Q_S^3\setminus J_p} (p_{B_1} - S) \\ - (p_{B_1} - S) - \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ + (n_1 - n_{B'})\, p_{B_1} + \sum_{J_j\in B'} S \end{bmatrix}}{\begin{bmatrix} C_1^{**} + \sum_{J_j\in Q_S^3\setminus J_p} C_j^{**} + C_p^{**} \\ + \sum_{J_j\in A_1-B'} C_j^{**} + \sum_{J_j\in B'} C_j^{**} \end{bmatrix}} \tag{3.97}$$

$$= 1 + \max\left\{ \frac{\begin{bmatrix} S + n_S S + \sum_{J_j\in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} - n_S p_{B_1} \\ - \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} + (n_1 - n_{B'}) p_{B_1} \end{bmatrix}}{\begin{bmatrix} p_{B_1} + (n_S - 1) p_{B_1} + \sum_{J_j\in Q_S^3\setminus J_p} k \frac{p_{S,n_S}}{\beta^{n_S-k}} + S_{B_1} \\ + \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} + p_{B_1} - S \\ + (n_1 - n_{B'}) S_{B_1} + \sum_{J_j\in A_1-B'} k \frac{p_{1,n_1}}{\beta^{n_1-k}} \end{bmatrix}}, \max_{J_j\in B'} \frac{S}{C_j^{**}} \right\} \tag{3.98}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{S + n_S S}{p_{B_1} + n_S p_{B_1}}, \frac{\begin{bmatrix} \sum_{J_j\in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} - n_S p_{B_1} \\ - \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ + (n_1 - n_{B'}) p_{B_1} \end{bmatrix}}{\begin{bmatrix} \sum_{J_j\in Q_S^3\setminus J_p} k \frac{p_{S,n_S}}{\beta^{n_S-k}} + S_{B_1} \\ + \sum_{J_j\in A_1-B'} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ -S + (n_1 - n_{B'}) S_{B_1} \\ + \sum_{J_j\in A_1-B'} k \frac{p_{1,n_1}}{\beta^{n_1-k}} \end{bmatrix}} \right\}, \max_{J_j\in B'} \frac{S}{C_j^{**}} \right\} \tag{3.99}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{S}{p_{B_1}}, \frac{\left[\begin{array}{c}(n_1 - n_{B'})p_{B_1} \\ -\sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1-k}}\end{array}\right]}{\left[\begin{array}{c}(n_1 - n_{B'})S_{B_1} \\ +\sum_{J_j \in A_1 - B'} k\frac{p_{1,n_1}}{\beta^{n_1-k}}\end{array}\right]} \right\}, \max_{J_j \in B'} \frac{S}{C_j^{**}} \right\} \tag{3.100}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{S}{p_{B_1}}, \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}} \right\}, \frac{S}{p_{B_1}} \right\} \tag{3.101}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{1}{\phi}, \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}} \right\}, \frac{1}{\phi} \right\} \tag{3.102}$$

$$= \max\left\{ 1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}} \right\} \tag{3.103}$$

L'équation (3.100) est due au fait que $\sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} < n_S p_{B_1}$ et $S_{B_1} > S$. De plus, l'équation (3.101) vient du fait que les dates de fin des tâches de B' sont au moins égales à p_{B_1} tandis que (3.102) vient du fait que $p_{B_1} \geq \phi S$.

D) J_{B_1} est préemptée par les n_S tâches de Q_S^3 puis elle est également préemptée par les n_1 tâches de A_1 (Figure 3.13-D).

Nous avons,

$$C_1 = C_1^{**} + S - \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \tag{3.104}$$

$$C_j = C_j^{**} \quad \forall J_j \in Q_S^3 \tag{3.105}$$

$$C_j = C_j^{**} + p_{B_1} \quad \forall J_j \in A_1 \tag{3.106}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} est préemptée.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{C_1^{**} + S - \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} \left(C_j^{**} + p_{B_1}\right)}{C_1^{**} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.107}$$

$$\leq 1 + \frac{S - \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 p_{B_1}}{p_{B_1} + \sum_{k=1}^{n_S} \frac{p_{S,n_S}}{\beta^{n_S-k}} + \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 S_{B_1} + \sum_{k=1}^{n_1} k \frac{p_{1,n_1}}{\beta^{n_1-k}}} \tag{3.108}$$

$$\leq \max\left\{1 + \frac{S}{p_{B_1}}, 1 + \frac{n_1 p_{B_1} - \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}}}{n_1 S_{B_1} + \sum_{k=1}^{n_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + \sum_{k=1}^{n_1} k \frac{p_{1,n_1}}{\beta^{n_1-k}}}\right\} \tag{3.109}$$

$$= \max\left\{1 + \frac{S}{p_{B_1}}, 1 + \frac{n_1 - \frac{p_{1,n_1}}{p_{B_1}} \sum_{k=1}^{n_1} \beta^{k-n_1}}{\left[\begin{array}{c} n_1 \frac{S_{B_1}}{p_{B_1}} + \frac{p_{1,n_1}}{p_{B_1}} \sum_{k=1}^{n_1} \beta^{k-n_1} \\ + \frac{p_{1,n_1}}{p_{B_1}} \sum_{k=1}^{n_1} k\beta^{k-n_1} \end{array}\right]}\right\} \tag{3.110}$$

$$\leq \max\left\{1 + \frac{1}{\phi}, 1 + \frac{n_1 - \gamma \sum_{k=1}^{n_1} \beta^{k-n_1}}{\frac{n_1}{\phi} + \gamma \sum_{k=1}^{n_1} \beta^{k-n_1} + \gamma \sum_{k=1}^{n_1} k\beta^{k-n_1}}\right\} \tag{3.111}$$

$$\leq \max\left\{1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}\right\} \tag{3.112}$$

Où l'inégalité (3.111) est due à $p_{B_1} \geq \phi S$, $\frac{p_{1,n_1}}{p_{B_1}} > \gamma$, et $p_{B_1} \leq \phi S_{B_1}$ vient de la règle ϕD-SPT.

E) J_{B_1} n'est préemptée que par $n_S - n_B$ tâches de Q_S^3 (figure 3.13-E).

Nous avons,

$$C_1 = C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S-k}} \tag{3.113}$$

$$C_j = C_j^{**} \quad \forall J_j \in Q_S^3 - B \tag{3.114}$$

$$C_j = C_j^{**} - (p_{B_1} - S) \quad \forall J_j \in B \tag{3.115}$$

$$C_j = C_j^{**} + S \quad \forall J_j \in A_1 \tag{3.116}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} est préemptée par $n_S - n_B$ tâches de Q_S^3.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{\left[\begin{array}{c} C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in Q_S^3} C_j^{**} \\ + \sum_{J_j \in A_1} \left(C_j^{**} + S\right) - \sum_{J_j \in B} \left(p_{B_1} - S\right) \end{array}\right]}{C_1^{**} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.117}$$

$$= 1 + \frac{S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in A_1} S - \sum_{J_j \in B} \left(p_{B_1} - S\right)}{C_1^{**} + \sum_{J_j \in Q_S^3} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.118}$$

$$= 1 + \frac{S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} \sum_{J_j \in A_1} S - \sum_{J_j \in B} \left(p_{B_1} - S\right)}{\left[\begin{array}{r} p_{B_1} + \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + (n_S - n_B)S + \sum_{J_j \in Q_S^3 - B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ + n_B p_{B_1} + n_B \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} + n_1 p_{B_1} \\ + n_1 \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in A_1} k \frac{p_{1,n_1}}{\beta^{n_1 - k}} \end{array}\right]} \tag{3.119}$$

$$= 1 + \frac{\left(1 + n_1 + n_B\right) S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - n_B p_{B_1}}{\left[\begin{array}{r} \left(1 + n_1 + n_B\right) p_{B_1} + \left(n_S - n_B\right) S + \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ + \sum_{J_j \in Q_S^3 - B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} + n_B \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ + \sum_{J_j \in B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} + n_1 \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in A_1} k \frac{p_{1,n_1}}{\beta^{n_1 - k}} \end{array}\right]} \tag{3.120}$$

$$\leq 1 + \frac{S}{p_{B_1}} \tag{3.121}$$

$$\leq 1 + \frac{1}{\phi} \tag{3.122}$$

Où l'inégalité (3.121) est due à $\sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - n_B p_{B_1} < 0$, alors que l'inégalité (3.122) est parce que $p_{B_1} \geq \phi S$.

F) J_{B_1} n'est préemptée que par $n_S - n_B$ tâches de Q_S^3 et J_p est préemptée par les tâches de A_1 (figure 3.13-F).

Nous avons,

$$C_1 = C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S-k}} \tag{3.123}$$

$$C_j = C_j^{**} \quad \forall J_j \in Q_S^3 - B \tag{3.124}$$

$$C_j = C_j^{**} - (p_{B_1} - S) \quad \forall J_j \in B \backslash J_p \tag{3.125}$$

$$C_p = C_p^{**} - (p_{B_1} - S) - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \quad J_p \tag{3.126}$$

$$C_j = C_j^{**} + p_{B_1} \quad \forall J_j \in A_1 \tag{3.127}$$

Nous comparons donc la somme des dates de fin des tâches de F' lorsque J_{B_1} est préemptée par $n_S - n_B$ tâches de Q_S^3 et que J_p est préemptée par les tâches de A_1.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{\left[\begin{array}{r} C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S-k}} + \sum_{J_j \in Q_S^3 - B} C_j^{**} \\ + \sum_{J_j \in B \backslash J_p} \left(C_j^{**} - p_{B_1} + S\right) + C_p^{**} - p_{B_1} + S \\ - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + \sum_{J_j \in A_1} \left(C_j^{**} + p_{B_1}\right) \end{array}\right]}{\left[\begin{array}{r} C_1^{**} + \sum_{J_j \in Q_S^3 - B} C_j^{**} + C_p^{**} \\ + \sum_{J_j \in B \backslash J_p} C_j^{**} + \sum_{J_j \in A_1} C_j^{**} \end{array}\right]} \tag{3.128}$$

$$= 1 + \frac{\left[\begin{array}{r} S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S-k}} + (n_B - 1)(S - p_{B_1}) \\ - p_{B_1} + S - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 p_{B_1} \end{array}\right]}{C_1^{**} + \sum_{J_j \in Q_S^3 - B} C_j^{**} + C_p^{**} + \sum_{J_j \in B \backslash J_p} C_j^{**} + \sum_{J_j \in A_1} C_j^{**}} \tag{3.129}$$

$$= 1 + \frac{S + n_B S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S-k}} - n_B p_{B_1} - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} + n_1 p_{B_1}}{\left[\begin{array}{r} p_{B_1} + \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S-k}} + (n_S - n_B) S \\ + \sum_{J_j \in Q_S^3 - B} k \frac{p_{S,n_S}}{\beta^{n_S-k}} + p_{B_1} + \sum_{J_j \in Q_S^3} \frac{p_{S,n_S}}{\beta^{n_S-k}} + \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1-k}} \\ + (n_B - 1) p_{B_1} + (n_B - 1) \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S-k}} \\ + \sum_{J_j \in B \backslash J_p} k \frac{p_{S,n_S}}{\beta^{n_S-k}} + n_1 S_{B_1} + \sum_{J_j \in A_1} k \frac{p_{1,n_1}}{\beta^{n_1-k}} \end{array}\right]} \tag{3.130}$$

$$\leq \max\{1 + \frac{S + n_B S}{p_{B_1} + n_B p_{B_1}}, 1 + \frac{n_1 p_{B_1} - \sum_{J_j \in A_1} \frac{p_{1,n_1}}{\beta^{n_1 - k}}}{n_1 S_{B_1} + \sum_{J_j \in A_1} k \frac{p_{1,n_1}}{\beta^{n_1 - k}}}\} \tag{3.131}$$

$$\leq \max\left\{1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}\right\} \tag{3.132}$$

Où l'inégalité (3.131) est due à $\sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} < n_B p_{B_1}$, tandis que l'inégalité (3.132) est due à $p_{B_1} \geq \phi S$ et $p_{B_1} \leq \phi S_{B_1}$.

G) J_{B_1} n'est préemptée que par $n_S - n_B$ tâches de Q_S^3 et J_p n'est préemptée que par $n_1 - n_{B'}$ tâches de A_1 (figure 3.13-G).

Nous avons

$$C_1 = C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} \tag{3.133}$$

$$C_j = C_j^{**} \quad \forall J_j \in Q_S^3 - B \tag{3.134}$$

$$C_j = C_j^{**} - (p_{B_1} - S) \quad \forall J_j \in B \backslash J_p \tag{3.135}$$

$$C_p = C_p^{**} - (p_{B_1} - S) - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} \quad J_p \tag{3.136}$$

$$C_j = C_j^{**} + p_{B_1} \quad \forall J_j \in A_1 - B' \tag{3.137}$$

$$C_j = C_j^{**} + S \quad \forall J_j \in B' \tag{3.138}$$

Ainsi, nous comparons la somme des dates de fin des tâches de F' lorsque J_{B_1} est préemptée par $n_S - n_B$ tâches de Q_S^3 et J_p est préemptée par $n_S - n_{B'}$ tâches de A_1.

$$\frac{\sum_{J_j \in F'} C_j}{\sum_{J_j \in F'} C_j^{**}} = \frac{\left[\begin{array}{r} C_1^{**} + S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in Q_S^3 - B} C_j^{**} \\ + \sum_{J_j \in B \backslash J_p} \left(C_j^{**} - p_{B_1} + S\right) + C_p^{**} - p_{B_1} + S \\ - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + \sum_{J_j \in A_1 - B'} \left(C_j^{**} + p_{B_1}\right) \\ + \sum_{J_j \in B'} \left(C_j^{**} + S\right) \end{array}\right]}{\left[\begin{array}{r} C_1^{**} + \sum_{J_j \in Q_S^3 - B} C_j^{**} + C_p^{**} + \sum_{J_j \in B \backslash J_p} C_j^{**} \\ + \sum_{J_j \in A_1 - B'} C_j^{**} + \sum_{J_j \in B'} C_j^{**} \end{array}\right]} \tag{3.139}$$

$$= 1 + \frac{\left[\begin{array}{l} S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - \sum_{J_j \in B \setminus J_p} (p_{B_1} - S) - p_{B_1} + S \\ \quad - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + \sum_{J_j \in A_1 - B'} p_{B_1} + \sum_{J_j \in B'} S \end{array}\right]}{\left[\begin{array}{l} C_1^{**} + \sum_{J_j \in Q_S^3 - B} C_j^{**} + C_p^{**} \\ \quad + \sum_{J_j \in B \setminus J_p} C_j^{**} + \sum_{J_j \in A_1 - B'} C_j^{**} + \sum_{J_j \in B'} C_j^{**} \end{array}\right]} \tag{3.140}$$

$$\leq 1 + \max \left\{ \frac{\left[\begin{array}{l} S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - (n_B - 1)(p_{B_1} - S) \\ \quad S - p_{B_1} - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + \sum_{J_j \in A_1 - B'} p_{B_1} \end{array}\right]}{\left[\begin{array}{l} C_1^{**} + \sum_{J_j \in Q_S^3 - B} C_j^{**} + C_p^{**} \\ \quad + \sum_{J_j \in B \setminus J_p} C_j^{**} + \sum_{J_j \in A_1 - B'} C_j^{**} \end{array}\right]}, \max_{J_j \in B'} \frac{S}{C_j^{**}} \right\} \tag{3.141}$$

$$\leq 1 + \max \left\{ \frac{\left[\begin{array}{l} S + \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - (n_B - 1)(p_{B_1} - S) - p_{B_1} + S \\ \qquad - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + \sum_{J_j \in A_1 - B'} p_{B_1} \end{array}\right]}{\left[\begin{array}{l} p_{B_1} + \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + (n_S - n_B) S \\ \quad + \sum_{J_j \in Q_S^3 - B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} + (n_{B_1} - 1) p_{B_1} \\ \quad + (n_{B_1} - 1) \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + \sum_{J_j \in B \setminus J_p} k \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ \qquad + S_{B_1} + \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + p_{B_1} - S \\ \qquad + (n_1 - n_{B'}) S_{B_1} + \sum_{J_j \in A_1 - B'} k \frac{p_{1,n_1}}{\beta^{n_1 - k}} \\ \\ \end{array}\right]}, \frac{S}{p_{B_1}} \right\} \tag{3.142}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{S + n_B S}{p_{B_1} + n_B p_{B_1}}, \frac{\left[\begin{array}{l} \sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} - n_B p_{B_1} \\ \quad - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + (n_1 - n_{B'}) p_{B_1} \end{array}\right]}{\left[\begin{array}{l} \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} + (n_S - n_B) S \\ \quad + \sum_{J_j \in Q_S^3 - B} k \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ \quad\quad + (n_{B_1} - 1) \sum_{J_j \in Q_S^3 - B} \frac{p_{S,n_S}}{\beta^{n_S - k}} \\ \quad + \sum_{J_j \in B \setminus J_p} k \frac{p_{S,n_S}}{\beta^{n_S - k}} + S_{B_1} - S \\ \quad\quad + \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}} + (n_1 - n_{B'}) S_{B_1} \\ \quad\quad\quad + \sum_{J_j \in A_1 - B'} k \frac{p_{1,n_1}}{\beta^{n_1 - k}} \end{array}\right]} \right\}, \frac{S}{p_{B_1}} \right\}$$

$$\leq 1 + \max\left\{ \max\left\{ \frac{S}{p_{B_1}}, \frac{(n_1 - n_{B'}) p_{B_1} - \sum_{J_j \in A_1 - B'} \frac{p_{1,n_1}}{\beta^{n_1 - k}}}{(n_1 - n_{B'}) S_{B_1} + \sum_{J_j \in A_1 - B'} k \frac{p_{1,n_1}}{\beta^{n_1 - k}}} \right\}, \frac{S}{p_{B_1}} \right\} \tag{3.143}$$

$$\leq \max\left\{ 1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j - \bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j \beta^{j - \bar{n}}} \right\} \tag{3.144}$$

L'équation (3.143) vient du fait que $\sum_{J_j \in B} \frac{p_{S,n_S}}{\beta^{n_S - k}} < n_B p_{B_1}$ et $S_{B_1} > S$, alors que l'équation (3.144) est obtenue en considérant que $p_{B_1} \geq \phi S$.

Le sous-ensemble $F_i = \{J_{B_i}\} \cup A_i$ lorsque J_{B_i} est préemptée par les tâches de A_i

Pour ce sous-ensemble, SRPT devrait produire l'une des deux possibilités A$'$ ou B$'$ représentées dans la figure 3.14.

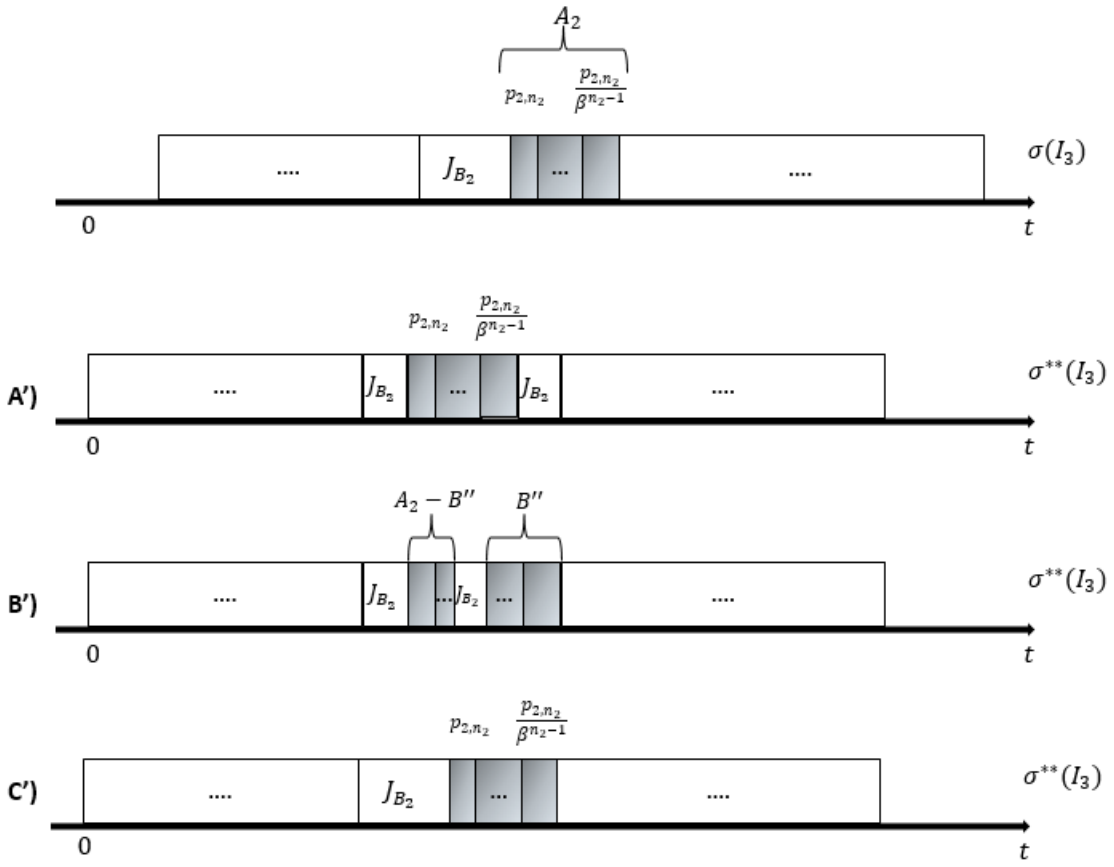

Figure 3.14 – Les ordonnancements $\sigma(I_3)$ et $\sigma^{**}(I_3)$ avec les possibilités d'ordonnancement (A$'$), (B$'$) et (C$'$) des tâches de F_i et F'' par SRPT

A$'$) La tâche J_{B_i} est préemptée par les tâches de A_i (figure 3.14-A$'$).

$$C_{B_i} = C^{**}_{B_i} + S - \sum_{k=1}^{n_i} \frac{p_{i,n_i}}{\beta^{n_i-k}} \tag{3.145}$$

$$C_j = C^{**}_j + p_{B_i} \quad \forall J_j \in A_i \tag{3.146}$$

La figure 3.14-A$'$ illustre le cas où J_{B_2} est préemptée par des tâches de A_2.

Ainsi, nous comparons la somme des dates de fin des tâches de F_i.

$$\frac{\sum_{J_j\in F_i} C_j}{\sum_{J_j\in F_i} C^{**}_j} = \frac{C^{**}_{B_i} + S - \sum_{k=1}^{n_i} \frac{p_{i,n_i}}{\beta^{n_i-k}} + \sum_{J_j\in A_i} \left(C^{**}_j + p_{B_i}\right)}{C^{**}_{B_i} + \sum_{J_j\in A_i} C^{**}_j} \tag{3.147}$$

$$\leq 1 + \frac{S - \sum_{k=1}^{n_1} \frac{p_{i,n_i}}{\beta^{n_1-k}} + n_i p_{B_i}}{p_{B_i} + \sum_{k=1}^{n_i} \frac{p_{i,n_i}}{\beta^{n_i-k}} + n_i S_{B_i} + \sum_{k=1}^{n_i} k \frac{p_{i,n_i}}{\beta^{n_i-k}}} \tag{3.148}$$

$$\leq \max\left\{1 + \frac{S}{p_{B_i}}, 1 + \frac{n_i p_{B_i} - \sum_{k=1}^{n_i} \frac{p_{i,n_i}}{\beta^{n_i-k}}}{n_i S_{B_i} + \sum_{k=1}^{n_i} \frac{p_{i,n_i}}{\beta^{n_i-k}} + \sum_{k=1}^{n_i} k \frac{p_{i,n_i}}{\beta^{n_i-k}}}\right\} \tag{3.149}$$

$$\leq \max\left\{1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}\right\} \tag{3.150}$$

B$'$) La tâche J_{B_i} est préemptée par quelques tâches de A_i (figure 3.14-B$'$).

Désignons par B'' l'ensemble des tâches parmi les tâches de A_i planifiées après la date de fin de J_{B_i} dans l'ordonnancement par SRPT et $n_{B''}$ le nombre de tâches dans cet ensemble.

$$C_{B_i} = C^{**}_{B_i} + S - \sum_{J_j\in A_i - B''} \frac{p_{i,n_i}}{\beta^{n_i-k}} \tag{3.151}$$

$$C_j = C^{**}_j + p_{B_i} \quad \forall J_j \in A_i - B'' \tag{3.152}$$

$$C_j = C^{**}_j + p_{B_i} - (p_{B_i} - S_{B_i}) = C^{**}_j + S_{B_i} \quad \forall J_j \in B'' \tag{3.153}$$

La figure 3.14-B$'$ illustre le cas où J_{B_2} est préemptée par $(n_2 - n_{B''})$ tâches de A_2.

Ainsi, nous comparons la somme des dates de fin des tâches de F_i.

$$\frac{\sum_{J_j\in F_i} C_j}{\sum_{J_j\in F_i} C_j^{**}} = \frac{\left[\begin{array}{c} C_{B_i}^{**} + S - \sum_{J_j\in A_i - B''} \frac{p_{i,n_i}}{\beta^{n_i-k}} \\ + \sum_{J_j\in A_i-B''}\left(C_j^{**}+p_{B_i}\right) + \sum_{J_j\in B''}\left(C_j^{**}+S_{B_i}\right)\end{array}\right]}{C_{B_i}^{**} + \sum_{J_j\in A_i-B''} C_j^{**} + \sum_{J_j\in B''} C_j^{**}} \tag{3.154}$$

$$= 1 + \frac{S - \sum_{J_j\in A_i-B''} \frac{p_{i,n_i}}{\beta^{n_i-k}} + (n_i - n_{B''})\, p_{B_i} + n_{B''} S_{B_i}}{\left[\begin{array}{r} p_{B_i} + \sum_{J_j\in A_i-B''} \frac{p_{i,n_i}}{\beta^{n_i-k}} + (n_i - n_{B''})\, S_{B_i} \\ + \sum_{J_j\in A_i-B''} k\frac{p_{i,n_i}}{\beta^{n_i-k}} + n_{B''} S_{B_i} + n_{B''}\sum_{J_j\in A_i-B''} \frac{p_{i,n_i}}{\beta^{n_i-k}} \\ + n_{B''}\left(p_{B_i} - S_{B_i}\right) + \sum_{J_j\in B''} k\frac{p_{i,n_i}}{\beta^{n_i-k}}\end{array}\right]} \tag{3.155}$$

$$\leq \max\left\{1 + \frac{S_{B_i} + n_{B''} S_{B_i}}{p_{B_i} + n_{B''} p_{B_i}}, 1 + \frac{(n_i - n_{B''})\, p_{B_i} - \sum_{J_j\in A_i-B''} \frac{p_{i,n_i}}{\beta^{n_i-k}}}{(n_i - n_{B''})\, S_{B_i} + \sum_{J_j\in A_i-B''} k\frac{p_{i,n_i}}{\beta^{n_i-k}}}\right\} \tag{3.156}$$

$$\leq \max\left\{1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma\sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}\right\} \tag{3.157}$$

Alors que l'inégalité (3.156) est due à $S < S_{B_i}$, tandis que l'inégalité (3.157) est due à $p_{B_i} \leq \phi S_{B_i}$.

Le sous-ensemble $F'' = \{J_{B_i}\} \cup A_i$ lorsque J_{B_i} n'est pas préemptée par les tâches de A_i

F'' contient toutes les tâches de Q_{BS}^3 qui ne sont pas préemptées dans $\sigma^{**}(I_3)$, et les autres tâches dans Q_{AS}^3 qui ne préemptent aucune tâche dans $\sigma^{**}(I_3)$. Nous pouvons conclure que chaque tâche de F'' débutera avec un retard de S dans $\sigma(I_3)$ par rapport à $\sigma^{**}(I_3)$. D'où,

$$\frac{\sum_{J_j\in F''} C_j}{\sum_{J_j\in F''} C_j^{**}} \leq \max_{J_j\in F''}\left\{\frac{C_j^{**} + S}{C_j^{**}}\right\} \leq 1 + \frac{S}{p_{B_1}} \leq 1 + \frac{1}{\phi} \tag{3.158}$$

La figure 3.14-C′ illustre le cas où J_{B_2} n'est pas préemptée.

Puisque chaque tâche dans Q^3 est contenue dans un des sous-ensembles F', F_i ou F'', alors selon les équations (3.70), (3.80), (3.90), (3.103), (3.112), (3.122), (3.132), (3.144), (3.150), (3.157), et (3.158), nous obtenons le résultat proposé. □

Théorème 7. Pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$ pour $0 < \beta < 1$ et $p_L > 0$, l'algorithme ϕD-SPT a un ratio de compétitivité de $1 + \frac{1}{\phi}$, avec $\phi = \frac{2(\bar{n}-1)}{-\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4(\bar{n}-1)\left(1-\bar{n}+\gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)}}$.

Démonstration. A partir des Lemmes 3, 4 et 5, nous obtenons le ratio de compétitivité ρ de ϕD-SPT comme suit :

$$\rho = \max_{I_1} \frac{ALG(I_1)}{OPT(I_1)} \leq \max_{I_1} \frac{ALG(I_1)}{LB(I_1)} \tag{3.159}$$

$$\leq \max \left\{ 1 + \frac{1}{\phi}, 1 + \frac{\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}} \right\} \tag{3.160}$$

Nous avons $1 + \frac{1}{\phi} = 1 + \frac{\bar{n}-1-\gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}}{\frac{\bar{n}-1}{\phi} + \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}}$, alors,

$$\left(\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right) \phi^2 - \gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} \phi + 1 - \bar{n} = 0 \tag{3.161}$$

$$\Delta = \left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\left(\bar{n} - 1\right)\left(1 - \bar{n} + \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right) \tag{3.162}$$

Ainsi,

$$\phi = \frac{\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\left(\bar{n} - 1\right)\left(1 - \bar{n} + \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)}}{2\left(\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)} \tag{3.163}$$

$$= \frac{\left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - \left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 + 4\left(\bar{n} - 1\right)\left(1 - \bar{n} + \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)}{\left[\begin{array}{l} 2\left(\bar{n} - 1 - \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right) \\ \quad \left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} - \sqrt{\left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\left(\bar{n} - 1\right)\left(1 - \bar{n} + \gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)}\right) \end{array}\right]} \tag{3.164}$$

$$= \frac{2\left(\bar{n}-1\right)}{-\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\left(\bar{n}-1\right)\left(1-\bar{n}+\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}\right)}} \tag{3.165}$$

Nous avons

$$\rho \leq 1 + \frac{1}{\phi} \tag{3.166}$$

$$\rho \leq 1 + \frac{\left[\begin{array}{l} -\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} \\ \quad + \sqrt{\left(\gamma\sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4\left(\bar{n}-1\right)\left(1-\bar{n}+\gamma\sum_{j=2}^{\bar{n}}\beta^{j-\bar{n}}\right)} \end{array}\right]}{2\left(\bar{n}-1\right)} \tag{3.167}$$

□

La figure 3.15 ci-dessous illustre l'évolution du ratio de compétitivité et la borne inférieure par rapport à $\frac{1}{\gamma} = \frac{P_F}{P_L}$. Le petit écart entre le ratio de compétitivité de ϕD-SPT et la borne inférieure du problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L|\sum C_j$, où $0 < \beta < 1$ et $p_L > 0$, est due à l'utilisation de l'ordonnancement préemptif dans l'analyse de compétitivité alors que l'ordonnancement non-préemptif est utilisé dans la preuve de la borne inférieure.

3.5 ϕD-SPT et αD-SPT

Tao et al [87] ont étudié le problème $1|online, r_j, \frac{p_{max}}{p_{min}} \leq \gamma'|\sum C_j$, ils ont présenté un algorithme *semi-online* appelé αD-SPT avec un ratio de compétitivité égal à $1 + \frac{1}{\alpha}$, où $\alpha = \frac{1+\sqrt{1+\gamma'(\gamma'-1)}}{\gamma'-1}$.

Dans notre problème, nous avons l'information sur l'ordre décroissant des temps de traitement avec un facteur de β et l'information que le temps de traitement de la dernière tâche est limité par une valeur p_L. En outre, étant donné que la première tâche libérée dans notre problème a le temps de traitement maximum, tandis que la dernière tâche libérée a le temps de traitement minimum, l'information peut être exprimée comme suit :

$$\frac{p_{max}}{p_{min}} \leq \frac{1}{\gamma} \tag{3.168}$$

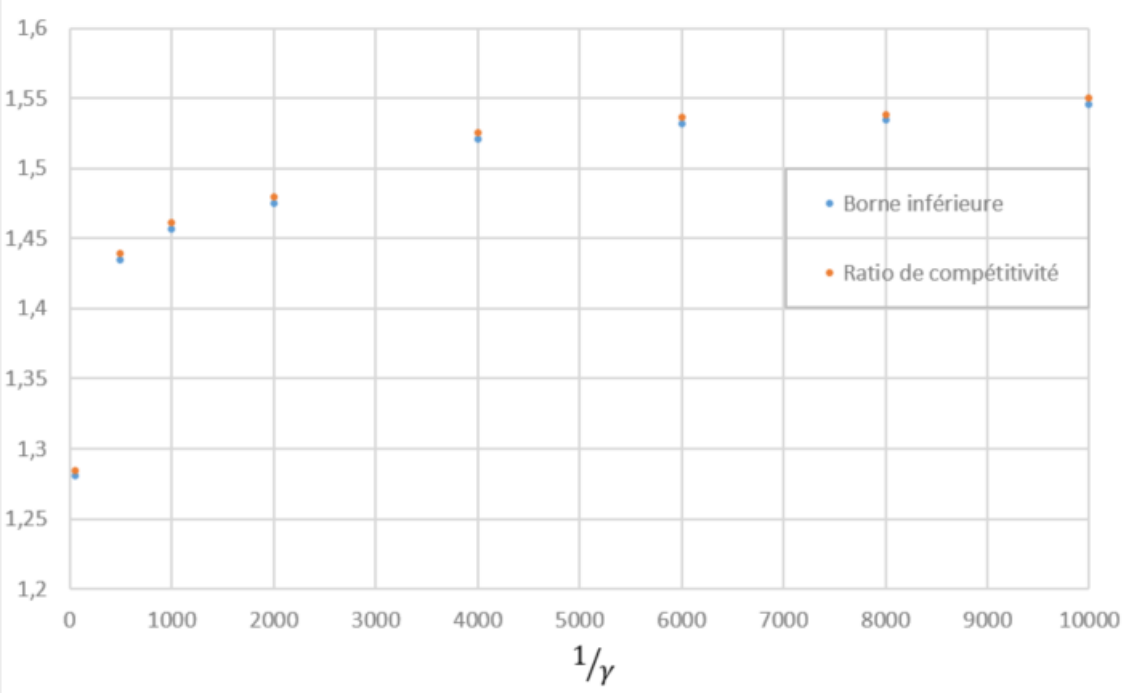

FIGURE 3.15 – Évolution de la borne inférieure et du ratio de compétitivité pour le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$, avec $0 < \beta < 1$, par rapport à $\frac{1}{\gamma}$

Considérons maintenant le ratio de compétitivité de ϕD-SPT, noté CR_ϕ, et le ratio de compétitivité de αD-SPT, noté CR_α, nous avons :

$$CR_\phi = 1 + \frac{-\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}} + \sqrt{\left(\gamma \sum_{j=2}^{\bar{n}} j\beta^{j-\bar{n}}\right)^2 - 4(\bar{n}-1)\left(1-\bar{n}+\gamma \sum_{j=2}^{\bar{n}} \beta^{j-\bar{n}}\right)}}{2(\bar{n}-1)} \tag{3.169}$$

Pour $\bar{n} = 2$, nous avons

$$CR_\phi(\bar{n}=2) = 1 + \frac{-2\gamma + \sqrt{4\gamma^2 - 4(\gamma-1)}}{2} \tag{3.170}$$

$$= 1 + \frac{\left(-2\gamma + \sqrt{4\gamma^2 - 4(\gamma-1)}\right)\left(2\gamma + \sqrt{4\gamma^2 - 4(\gamma-1)}\right)}{2\left(2\gamma + \sqrt{4\gamma^2 - 4(\gamma-1)}\right)} \tag{3.171}$$

$$= 1 + \frac{-4\gamma^2 + 4\gamma^2 - 4(\gamma - 1)}{2\left(2\gamma + \sqrt{4\gamma^2 - 4(\gamma - 1)}\right)} \tag{3.172}$$

$$= 1 + \frac{(1 - \gamma)}{\gamma + \sqrt{1 + \gamma(\gamma - 1)}} \tag{3.173}$$

Pour $\gamma = \frac{1}{\gamma'}$

$$CR_\phi(\bar{n} = 2) = 1 + \frac{(\gamma' - 1)}{1 + \sqrt{1 + \gamma'(\gamma' - 1)}} \tag{3.174}$$

Ainsi,

$$CR_\phi(\bar{n} = 2) = CR_\alpha \tag{3.175}$$

Selon l'équation suivante :

$$\bar{n} = \frac{ln(\gamma)}{ln(\beta)} + 1 \tag{3.176}$$

Pour $\bar{n} = 2$, nous avons $\gamma = \beta$.

$$\gamma = \beta \Rightarrow CR_\phi = CR_\alpha \tag{3.177}$$

Par conséquence, si nous considérons une instance avec au plus deux tâches, le ratio de compétitivité de ϕD-SPT devient équivalent à celui de αD-SPT. En effet, pour $\bar{n} = 2$, la pire instance pour les deux algorithmes αD-SPT et ϕD-SPT est la même, c'est-à-dire lancer une longue tâche avec un temps de traitement $p_{max} = p_F$ suivit d'une petite tâche avec un temps de traitement $p_{min} = p_L$.

3.6 Étude numérique

Dans ce qui suit, une analyse numérique est présentée en implémentant quatre algorithmes différents : D-SPT, αD-SPT, ϕD-SPT et SRPT. La méthode SRPT est utilisée comme borne inférieure de la valeur objective d'un algorithme *offline* optimal du problème, ce qui permet de calculer une borne supérieure du *ratio de* compétitivité pour chaque algorithme. De plus, D-SPT est l'algorithme *online* proposé par Hoogeveen et Vestjens [42] où aucune information sur les tâches à venir n'est connue, tandis que αD-SPT est l'algorithme *semi-online* présenté par Tao et al. [87], où le ratio du plus grand temps de traitement sur le plus petit est borné par une valeur noté $\gamma^{'}$.
De nombreuses études expérimentales peuvent être identifiées dans la littérature. Un travail récent proposé par Albers [3] s'est concentré sur les algorithmes d'ordonnancement *online* pour minimiser le *makespan* sur les machines parallèles. Elle a analysé des

algorithmes sur diverses séquences de tâches. Certaines d'entre-elles étaient générées par des distributions de probabilités, tandis que d'autres étaient des données réelles. Hall et al. [38] ont fourni des propositions spécifiques pour le schéma de génération d'une variété de problèmes d'ordonnancement, y compris les problèmes avec des dates d'arrivée r_j.

D-SPT, αD-SPT, ϕD-SPT, et SRPT ont été testés sur des problèmes avec $\frac{1}{\gamma} = 1000, 5000, 10000$. Quand aux temps de traitement, nous générons pour chaque tâche un temps de traitement entier p_j à partir de la distribution uniforme $U[1, \beta p_{j-1}]$, pour $\beta = 0,8$, afin de satisfaire la condition de notre problème $\frac{p_j}{p_{j-1}} \leq \beta$. En outre, comme la variation des dates d'arrivée des tâches est susceptible d'influencer l'efficacité des algorithmes, nous les générons comme suit [38] :

$$r_1 = 0 \text{ et } r_j = r_{j-1} + X_j,\ j = 2, ..., n$$

Où X_j est une variable aléatoire générée à partir d'une distribution uniforme $U[0, 100]$. Pour chaque valeur de γ, 1000 instances ont été générées.

Les algorithmes ont été codés en C++ sur Microsoft Visual Studio et fonctionnent sur un processeur Intel i5-8350U à 1,7 GHz.

Afin de représenter au mieux la performance des algorithmes testés, nous avons choisi de présenter trois indicateurs de performance en se basant sur les ratios de compétitivité des 1000 instances générées pour les trois valeurs de γ : le ratio moyen, le nombre de fois où un algorithme est meilleur et le nombre de fois où un algorithme est en avance. Le premier indicateur représente la valeur moyenne du ratio de compétitivité sur les 1000 instances testées, le deuxième indicateur représente le nombre de fois que l'algorithme en question fournit la meilleure performance parmi les performances obtenues. Finalement, le troisième indicateur indique le nombre de fois où un algorithme a été en avance de performance sur les autres algorithmes. Les figures 3.16, 3.17 et 3.18 illustrent les performances des trois algorithmes testés. Nous pouvons constater à partir de la figure 3.17 que ϕD-SPT obtient les meilleurs résultats en moyenne par rapport à D-SPT et αD-SPT. La figure 3.16 montre la grandeur de l'amélioration de l'algorithme proposé. Finalement, nous pouvons conclure à partir de la figure 3.18 que dans la plupart du temps ϕD-SPT est en avance sur les autres algorithmes testés.

3.7 Conclusion

Dans ce chapitre, nous avons étudié le problème $1|online, r_j, \frac{p_{j+1}}{p_j} \leq \beta, p_j \geq p_L| \sum C_j$ pour $0 < \beta < 1$ et $p_L > 0$. Dans ce problème, les tâches arrivent en ordre décroissant des temps de traitement et une borne sur le temps de traitement de la dernière tâche est connue à l'avance. Nous avons construit un nouvel algorithme pour le problème, nommé ϕD-SPT, avec ϕ un paramètre du problème intégrant les informations partielles considérées. De plus, nous avons établie une analyse de compétitivité de l'algorithme *semi-online* proposé en utilisant la méthode de la réduction des instances introduite par Tao et al. [87]. Nous avons donc prouvé que ϕD-SPT est $1 + \frac{1}{\phi}$-compétitive, avec $\phi > 1$. Une très légère différence par rapport à la borne inférieure du problème est due au fait que le ratio de compétitivité de ϕD-SPT est obtenue en utilisant l'ordonnancement avec préemptions comme ordonnancement *offline* optimal. Finalement, une étude numérique permet également de montrer l'avantage de l'algorithme *semi-online* proposé en pratique.

Dans le chapitre suivant, nous nous intéressons au problème $1|semi{-}online, r_j, | \sum C_j$ avec des dates d'arrivée des tâches connues à l'avance tandis qu'aucune information n'est disponible sur les temps de traitement des tâches.

Les travaux présentés dans ce chapitre ont fait l'objet des publications suivantes :

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. Semi-online Scheduling For Minimizing The Total Completion Time : Information Value. International Conference on Optimization and Learning, Spain, 2020. [64]

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. Minimising The Total Completion Time for a Class of Semi-online Single Machine Scheduling Problems. Theoretical Computer Science, 2021, en révision. [65]

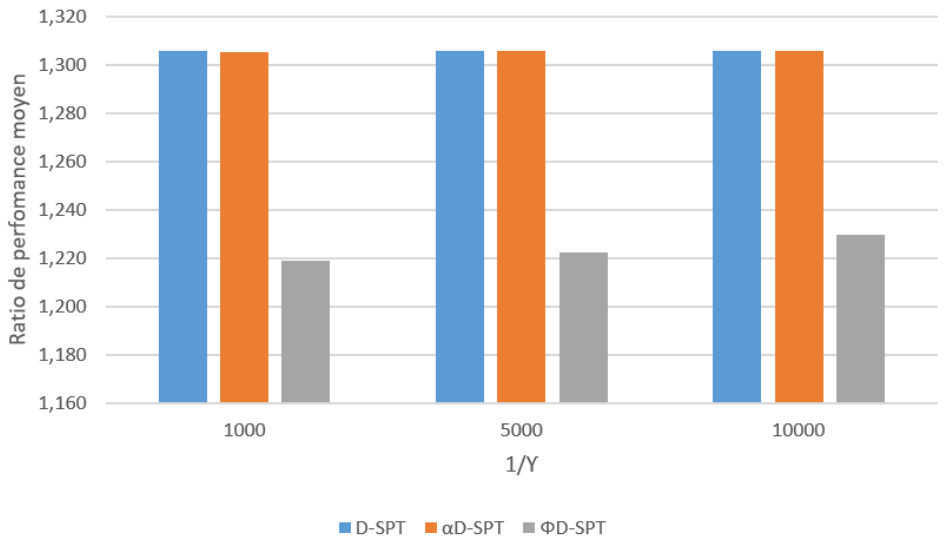

Figure 3.16 – Ratios de compétitivité moyen pour D-SPT, αD-SPT et ϕD-SPT sur les instances choisies

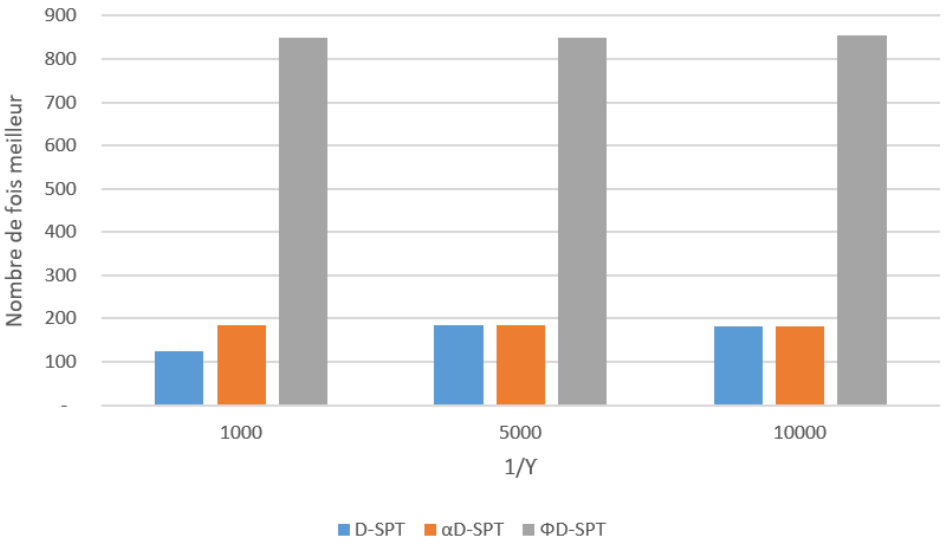

Figure 3.17 – Le Nombre de fois où un algorithme est meilleur pour D-SPT, αD-SPT et ϕD-SPT

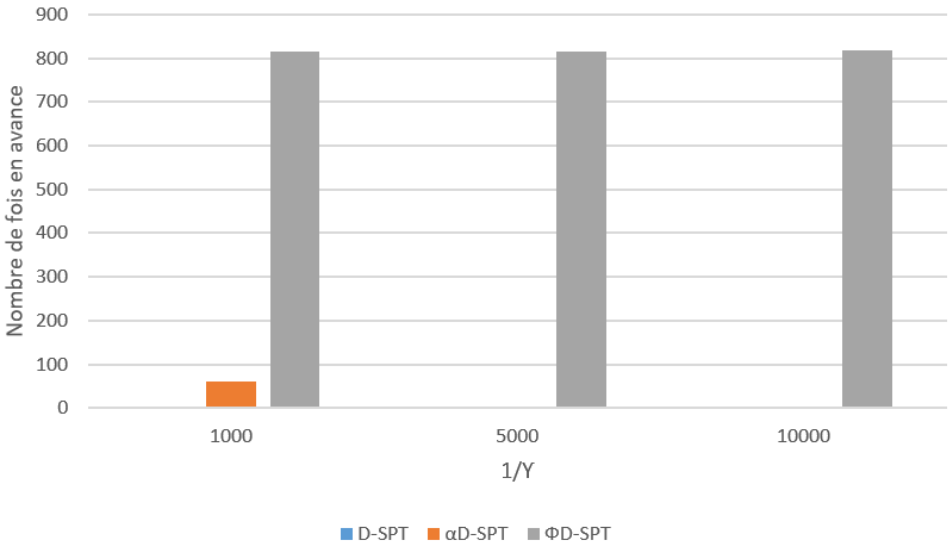

Figure 3.18 – Le Nombre de fois où un algorithme est en avance pour D-SPT, αD-SPT et ϕD-SPT

Chapitre 4

Algorithmes *semi-online* avec intégration de l'information sur les dates d'arrivée des tâches

Résumé :
Nous étudions dans un premier temps le modèle *semi-online* du problème de minimisation de la somme des dates de fin des tâches sur machine unitaire. Dans cette étude, nous considérons le cas où les dates d'arrivée des tâches sont connues au départ, alors que les temps de traitement demeurent inconnus. Nous développons d'abord un algorithme *semi-online*, nommé VD-SPT (*Variable Delayed Shortest Processing Time*) qui obtient de très bons résultats sur des instances de petite taille selon notre étude expérimentale. Ensuite, nous présentons un algorithme nommé CSPTn pour le problème qui, en le comparant aux algorithmes *online* et *semi-online* existants, obtient de très bons résultats sur des instances de grande taille et plus précisément sur des instances présentant une grande variabilité en termes de temps de traitement. Finalement, nous présentons la borne inférieure sur le ratio de compétitivité qui permet de vérifier l'utilité de l'information partielle pour le problème étudié. Dans un second temps, nous étudions le problème de minimisation de la somme des dates de fin pondérées des tâches sur machine unitaire. Nous démontrons d'abord que l'information exacte sur les dates d'arrivée des tâches n'apporte aucune amélioration. Ensuite, nous étudions la combinaison de l'information sur les dates d'arrivée potentielles et une borne sur le plus petit temps de traitement. Pour ce problème, nous développons un algorithme *semi-online*, nommé MCSWPT, et nous évaluons sa performance par une étude expérimentale comparative.

4.1 Introduction

Dans les problèmes d'ordonnancement *online* ou *semi-online*, lorsque les modèles d'ordonnancement prennent en compte la restriction de l'arrivée des tâches dans le temps, les tâches ne doivent pas obligatoirement être ordonnancées dès leurs arrivées. En effet, l'algorithme *online* ou *semi-online* doit décider, à chaque fois que la machine est inactive, soit de planifier une des tâches disponibles, soit de garder la machine inactive pour que, si une tâche importante arrive dans le futur, elle puisse être traitée immédiatement. Nous avons étudié dans le chapitre précédent le modèle *semi-online* avec des informations sur les temps de traitement des tâches. Dans ce chapitre, nous étudions le problème *semi-online* avec intégration de l'information sur les dates d'arrivée des tâches. Cependant, les temps de traitement des tâches demeurent inconnus. En fait, la variante *semi-online* avec une information sur les dates d'arrivée des tâches n'a pas été suffisamment étudiée dans la littérature. Nous citons les deux travaux qui ont étudié le problème *semi-online* :

— Pour le problème d'ordonnancement sur machine unitaire, qui consiste à minimiser la somme des dates de fin des tâches, Tao [88] montre que l'information sur la prochaine date d'arrivée à l'instant de décision présent n'apporte aucune amélioration. En effet, il a prouvé que la borne inférieure sur le ratio de compétitivité de ce problème est égale à la meilleure borne connue de 2 même en ajoutant cette information.

— Hall et al. [39] ont étudié le problème où les tâches ne peuvent arriver qu'à des instants connus dans le temps. Cet environnement se situe entre les environnements classiques d'ordonnancement *offline* et *online*. L'objectif est de minimiser la somme des dates de fin pondérées des tâches. En considérant un environnement non-préemptif à machine unitaire, ils montrent que la borne inférieure du problème étudié est la solution d'un modèle mathématique, noté $P(v)$. Cette borne inférieure est égale à $R^* < 2$. La valeur exacte dépend des dates d'arrivée potentielles des tâches. Ils présentent également un algorithme d'ordonnancement *semi-online*, nommé CSWPT (*Critical Shortest Weighted Processing Time*), et montrent que son ratio de compétitivité correspond à cette borne inférieure.

Dans la section suivante, nous étudions le problème $1|online, known\ r_j|\sum C_j$ pour lequel nous présentons deux algorithmes *semi-online* utilisant l'information sur les dates d'arrivée des tâches. Une étude numérique est menée pour évaluer la performance de chaque algorithme.

4.2 Problème $1|online, known\ r_j|\sum C_j$

L'algorithme D-SPT prend des décisions en *online*, ce qui signifie qu'aucune information concernant les tâches à venir n'est disponible au départ. Ceci justifie la nécessité de définir un temps d'attente fixe qui ne dépend que du temps de traitement d'une tâche disponible et de la valeur de l'instant de décision présent. Cependant, dans cette section, nous étudions la variante *semi-online* du problème où les dates d'arrivée des tâches sont connues à l'avance. Le premier avantage de l'ajout de cette information est que l'algorithme *semi-online* ne doit pas toujours attendre. Par exemple, si des tâches sont disponibles et que nous savons qu'aucune autre tâche ne sera libérée dans le futur, un algorithme *semi-online* n'insérera plus de période d'inactivité et les tâches seront planifiées par ordre SPT. La réponse à la question : est-ce que d'autres tâches vont être libérées dans le futur ? devient cruciale dans ce cas. Un autre exemple montrant l'importance d'avoir l'information sur les dates d'arrivée est celui où certaines tâches sont disponibles et que la prochaine date d'arrivée est supérieure à la somme des temps de traitement des tâches disponibles et de l'instant de décision présent. Dans ce cas, les tâches peuvent également être planifiées en mode *offline* sans insérer des périodes d'inactivité puisque nous savons que la prochaine date d'arrivée n'interférera pas avec la planification des tâches disponibles.

4.2.1 L'algorithme VD-SPT

La conception d'un algorithme *semi-online* ou *online* repose sur la même logique que la conception d'un algorithme *offline*. La principale différence réside dans la quantité des informations utilisées dans l'exécution de l'algorithme. Un algorithme *online* ne peut utiliser que les informations disponibles à l'instant présent de décision, tandis qu'un algorithme *offline* dispose de toutes les informations nécessaires pour construire un bon ordonnancement. Cependant, dans la conception de VD-SPT, nous utilisons une information existante, à savoir les dates d'arrivée des tâches, ce qui signifie que nous connaissons également le nombre n de tâches qui vont arriver.

Définition de t_{max}

Pour le problème *semi-online* de la minimisation de la somme des dates de fin des tâches, un algorithme *semi-online* doit répondre aux questions suivantes : Si une tâche est disponible, est-il préférable d'attendre ou de la planifier immédiatement ? et s'il choisit d'attendre, pour combien de temps ? Puisque nous avons une information sur les dates d'arrivée, nous utilisons cette information dans la construction de notre algorithme.

Hoogeveen et Vestjens [42] ont prouvé que D-SPT (Delayed Shortest Processing Time) est le meilleur algorithme possible pour le problème $1|r_j, online|\sum C_j$ en dérivant une borne inférieure sur le ratio de compétitivité de tout algorithme *online*. Cette borne inférieure a été obtenue en considérant un ensemble d'instances qui représentent les pires scénarios pour tout algorithme *online* et pour lesquelles aucun algorithme *online* ne peut garantir un résultat strictement inférieur au double de l'optimum. Pour trouver ces instances, la méthode de l'adversaire est utilisée. Dans cette méthode, le pire cas est obtenu en jouant le rôle d'un adversaire qui essaie de rendre la performance de l'algorithme *online* aussi mauvaise que possible par rapport à sa propre performance lorsqu'il traite la même instance en *offline*. Les instances peuvent être décrites comme suit : une première tâche arrive avec un temps de traitement p. Le premier scénario est celui où l'algorithme *online* décide d'ordonnancer la première tâche à un instant S alors que l'adversaire ne libère aucune autre tâche dans le futur. Le second scénario est celui où $n-1$ tâches avec des temps de traitement infiniment petits sont lancées immédiatement après le début de traitement de la première tâche.
Par intuition et en se basant sur la même logique de construction de l'instance qui représente le pire cas pour le modèle *online*, nous pouvons imaginer deux cas possibles qui représentent les pires scénarios pour un algorithme *semi-online* qui a l'avantage de connaître les dates d'arrivée des tâches. Tout d'abord, parmi les tâches disponibles, nous choisissons une tâche ayant le plus petit temps de traitement, noté p. De plus, deux pires cas possibles peuvent se produire. Le premier cas est celui où l'adversaire libère un grand nombre de tâches avec de petits temps de traitement suite à la décision de l'algorithme *semi-online* de programmer une longue tâche. Le second cas est celui où l'algorithme décide d'attendre les prochaines tâches et que celles-ci s'avèrent avoir le même temps de traitement que la première. Ces deux cas sont étudiés afin de déterminer la valeur de t_{max}, qui représente le temps maximum qu'un algorithme *semi-online* peut se permettre d'attendre afin de garantir un compromis entre les deux pires cas décrits (Figures 4.1 et 4.2).

Nous tenons à noter que dans ces deux cas, nous nous concentrons uniquement sur les tâches arrivant dans l'intervalle de temps $]t, t+p[$, avec t étant l'instant de décision présent, puisque les tâches arrivant après $t+p$ n'interfèrent pas avec l'ordonnancement de la tâche actuelle. Nous désignons par t' la date d'arrivée de n' tâches, telles que $t' \in]t, t+p[$ et $1 \leq n' \leq n-1$ où n désigne le nombre total de tâches. L'étude des deux pires cas suivants permet de déterminer la valeur de t_{max}. Dans la suite, C_j^* désigne la date de fin de la tâche J_j dans l'ordonnancement *offline*.

Cas 1 : L'algorithme *semi-online* décide d'ordonnancer immédiatement la tâche disponible avec un temps de traitement p à l'instant t, tandis que l'adversaire libère n' tâches avec des temps de traitement égaux à ϵ à l'instant t', avec ϵ une valeur positive infiniment petite (Figure 4.1). Par conséquent, quand $\epsilon \to 0$ la valeur objective obtenue par l'algorithme *semi-online* est la suivante :

$$\sum C_j = t + p + n'(t + p) \tag{4.1}$$

$$\Leftrightarrow \sum C_j = (n' + 1)(t + p) \tag{4.2}$$

Puisque nous étudions le pire cas, qui est une mauvaise décision prise par un algorithme *semi-online* par rapport à l'algorithme *offline*, nous considérons que $t' < \frac{n'}{n'+1}p + t$, qui est l'instant à partir duquel l'algorithme *offline* décide d'attendre les petites tâches puisque son coût sera meilleur que celui de l'algorithme *semi-online* (Figure 4.1), alors que dans le cas contraire, l'adversaire prendra les mêmes décisions que l'algorithme *semi-online*. Par conséquent, les équations (4.1) et (4.2) ne sont valables que si t' vérifie cette inégalité.

$$\sum C_j^* = n't' + t' + p \tag{4.3}$$

$$\Leftrightarrow \sum C_j^* = (n' + 1)t' + p \tag{4.4}$$

Le rapport des deux valeurs de la fonction objectif est donc le suivant :

$$\frac{\sum C_j}{\sum C_j^*} = \frac{(n' + 1)(t + p)}{(n' + 1)t' + p} \tag{4.5}$$

Cas 2 : Pour ce deuxième cas, l'algorithme *semi-online* décide d'attendre l'arrivée des n' tâches à l'instant t', tandis que l'adversaire lance ces tâches avec un temps de traitement p égal au temps de traitement de la première tâche (Figure 4.2). Par conséquent, la somme des dates de fin des tâches obtenue par l'algorithme *semi-online* est la suivante :

$$\sum C_j = t' + p + \frac{n'(n' + 1)}{2}p + n'(t' + p) \tag{4.6}$$

$$\Leftrightarrow \sum C_j = (n' + 1)(t' + p) + \frac{n'(n' + 1)}{2}p \tag{4.7}$$

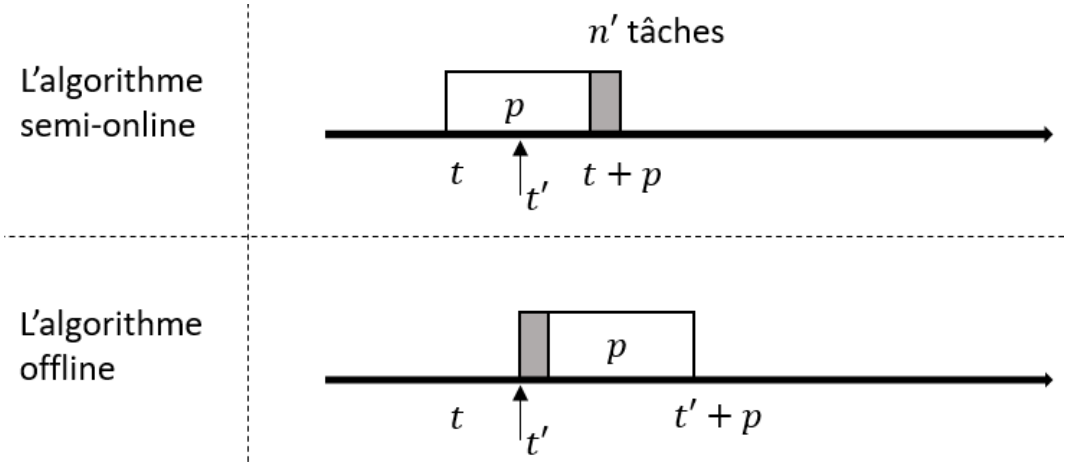

Figure 4.1 – Cas 1 : ordonnancements obtenus par l'algorithme *semi-online* et l'algorithme *offline*

L'algorithme *offline* optimal planifiera évidemment la première tâche immédiatement puisque tout attente serait une perte de temps (Figure 4.2).

$$\sum C_j^* = t + p + \frac{n'(n'+1)}{2}p + n'(t+p) \tag{4.8}$$

$$\Leftrightarrow \sum C_j^* = (n'+1)(t+p) + \frac{n'(n'+1)}{2}p \tag{4.9}$$

Le rapport des deux valeurs objectives est donc le suivant

$$\frac{\sum C_j}{\sum C_j^*} = \frac{2t' + (n'+2)p}{2t + (n'+2)p} \tag{4.10}$$

L'équation (4.5) est une fonction décroissante par rapport à t' alors que l'équation (4.10) est une fonction croissante par rapport à t'. Afin de trouver le valeur de t_{max} qui représente le compromis entre ces deux cas, nous devons étudier l'égalité de ces deux fonctions. Dans la suite, nous désignons par t_{max} la valeur de t' qui résulte de l'étude de l'égalité.

$$\frac{(n'+1)(p+t)}{(n'+1)t' + p} = \frac{2t' + (n'+2)p}{2t + (n'+2)p} \tag{4.11}$$

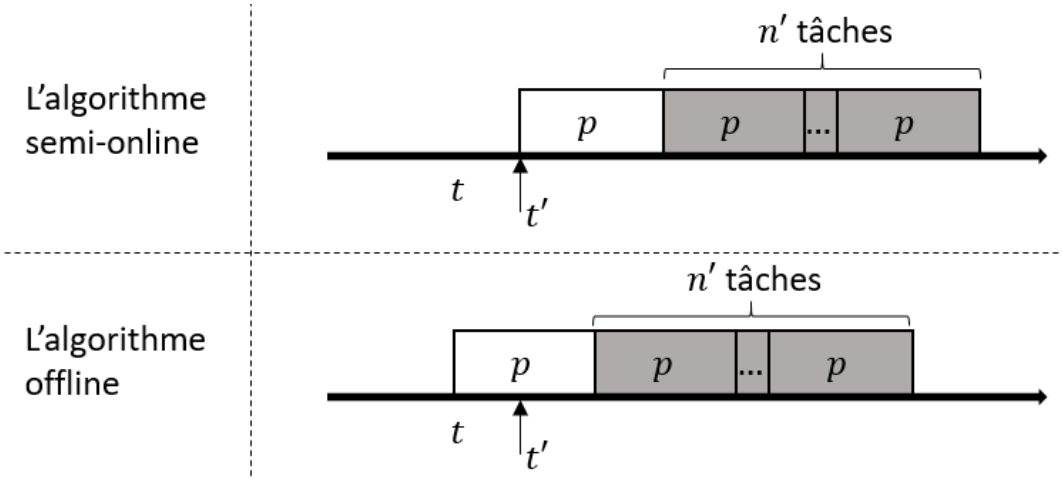

Figure 4.2 – Cas 2 : ordonnancements obtenus par l'algorithme *semi-online* et l'algorithme *offline*

(4.11) est équivalente à :

$$\begin{aligned} &2\left(n'+1\right)t_{max}^2+\left(\left(n'+1\right)\left(n'+2\right)+2\right)pt_{max}-\left(n'^2+5n'+4\right)tp \\ &-\left(2n'+2\right)t^2-n'\left(n'+2\right)p^2=0 \end{aligned} \tag{4.12}$$

Ce qui donne :

$$t_{max}=\frac{\left[\begin{array}{l}-\left(2+\left(n'+1\right)\left(n'+2\right)\right)p \\ \quad+\sqrt{\left(2+\left(n'+1\right)\left(n'+2\right)\right)^2p^2+8\left(n'+1\right)\left[\left(n'^2+5n'+4\right)tp\right.} \\ \qquad\qquad\left.+\left(2n'+2\right)t^2+n'\left(n'+2\right)p^2\right]\end{array}\right]}{4\left(n'+1\right)} \tag{4.13}$$

Nous pouvons observer à partir de l'équation (4.13) que la valeur de t_{max} dépend de trois paramètres : n' qui est le nombre de tâches libérées, t qui correspond à l'instant de décision actuel et p qui est le temps de traitement de la plus petite tâche parmi les tâches disponibles. Dans ce qui suit, un algorithme *semi-online* est présenté en utilisant l'expression de t_{max}.

VD-SPT

Précédemment, nous avons supposé que n' tâches pouvaient arriver à t'. Cependant, dans la suite nous utilisons l'expression de t_{max} en considérant que n'' tâches sont libérées dans l'intervalle $]t, t+p[$. Par conséquent, dans la suite, l'équation (4.13) sera utilisée en utilisant n'' au lieu de n' et pour $t = 0$, alors que le temps maximum que VD-SPT peut se permettre d'attendre est $t+t_{max}(n'', p)$. De plus, $t+t_{max}$ sera comparé à la dernière date d'arrivée dans l'intervalle $]t, t+p[$, noté $r_{n''}$, de manière itérative, de telle sorte que si la condition n'est pas vérifiée, la dernière tâche est supprimée et est remplacée par celle en amont si celle-ci existe.

Dans l'étape 1 de l'algorithme VD-SPT, une tâche disponible avec le plus petit temps de traitement, noté p_m, est sélectionnée. Ensuite, à l'étape 2, l'ensemble A est rempli avec les n'' tâches qui arrivent dans l'intervalle $]t, t+p_m[$. Nous trions et désignons les dates d'arrivée de ces tâches par $A = \{R_1, ..., R_{n''}\}$. Si aucune tâche n'arrive dans l'intervalle considéré, nous planifions immédiatement la tâche sélectionnée et mettons à jour la valeur de l'instant de décision actuel ($t = t + p_m$). À l'étape 3, nous calculons la valeur de $t_{max}(n'', p_m)$. Si le temps que nous nous autorisons à attendre la dernière tâche dans l'ensemble A est supérieur ou égal à la date d'arrivée de cette dernière, nous attendons, sinon nous mettons à jour l'ensemble A en supprimant la dernière tâche. La procédure ci-dessus est répétée jusqu'à ce que l'ensemble A devienne vide ou que la décision d'attendre soit prise. Ainsi, la valeur de t est mise à jour.

Étape 0 : Fixer t à la première date d'arrivée, N les tâches disponibles à l'instant t et N' les tâches avec des dates d'arrivées supérieures strictement à t.
Étape 1 : Fixer $A = \emptyset$. À l'instant de décision t et parmi les tâches dans N, sélectionner la tâche, notée J_M, avec le plus petit temps de traitement, noté p_m.
Étape 2 : Si des tâches arrivent dans l'intervalle $]t, t+p_m[$, noter leurs dates d'arrivées R_1,..., $R_{n''}$ triées en ordre croissant. Remplir A avec ces dates d'arrivées et passer à l'étape 3. Sinon, passer à l'étape 6.
Étape 3 : Calculer la valeur de t_{max} pour n'' et p_m et passer à l'étape 4.
Étape 4 : Si $R_{n''} \leq t+t_{max}$, alors ajouter les tâches dans A à l'ensemble N, supprimer les tâches dans A de l'ensemble N' et passer à l'étape 5. Sinon supprimer $R_{n''}$ de A et réinitialiser l'ensemble avec la même notation pour $n'' = n'' - 1$. si $A = \emptyset$ alors passer à l'étape 6, sinon revenir à l'étape 3.
Étape 5 : Si la première tâche dans A peut finir avant $R_{n''}$ alors retirer cette tâche de l'ensemble N, sinon passer à la tâche suivante et répéter la procédure jusqu'à la dernière tâche dans A. Finalement, mettre $t = R_{n''}$ et revenir à l'étape 1.
Étape 6 : Ordonnancer J_M à l'instant t et actualiser l'instant de décision en mettant

Table 4.1 – Exemple d'instance

j	1	2	3	4
r_j	0	0	1	2
p_j	8	6	ϵ	4

$t = t + p_m$. Si $N = \emptyset$ et $N' \neq \emptyset$ alors fixer t à la prochaine date d'arrivée et revenir à l'étape 1. Si $N \neq \emptyset$ alors revenir à l'étape 1. Si $N = \emptyset$ et $N' = \emptyset$ alors stop.

Considérons l'instance du tableau 4.1 à titre d'exemple pour appliquer l'algorithme VD-SPT. Les dates d'arrivée sont connues mais le temps de traitement d'une tâche ne devient connu que lorsque l'instant de décision actuel t est supérieur ou égal à la date d'arrivée de cette tâche. À l'étape 0, nous mettons $t = r_1 = 0$, $N = \{J_1, J_2\}$, et $N' = \{J_3, J_4\}$. Ensuite, à l'étape 1, la plus petite tâche J_2 est choisie avec un temps de traitement $p_2 = p_m = 6$. À l'étape 2, nous constatons que dans l'intervalle $]t, t + p_m[=]0, 6[$ il y a $n'' = 2$ tâches libérées, nous désignons leurs dates de libération $A = \{R_1, R_2\} = \{r_3, r_4\} = \{1, 2\}$. Dans l'étape 3, nous obtenons la valeur de $t_{max} = 2.8$ pour $n'' = 2$ et $p_m = 6$. À l'étape 4, nous comparons $R_{n''} = 2$ à $t + t_{max} = 2.8$, et puisque le second est supérieur au premier, nous mettons à jour les ensembles avec $N = \{J_1, J_2, J_3, J_4\}$ et $N' = \emptyset$ et passons à l'étape 5. Puisque la première tâche dans A avec la date d'arrivée $R_1 = 1$ peut finir avant $R_{n''}$, nous la planifions et mettons à jour l'ensemble $N = \{J_1, J_2, J_4\}$. De plus, nous mettons à jour l'instant de décision avec $t = R_{n''}$ et passons à l'étape 1. La même procédure est répétée et comme résultat, la tâche J_3 est programmée en premier à $t = 1$ suivie de J_4, J_2 et J_1 à partir de $t = 2$.

4.2.2 Étude expérimentale de VD-SPT

Dans cette section, une étude expérimentale comparative est présentée par la mise en œuvre de trois algorithmes différents D-SPT, VD-SPT et SRPT. Premièrement, une étude de cas moyen est présentée où nous testons des instances générées selon des lois de probabilité. Deuxièmement, une analyse du pire cas est présentée en étudiant une instance où des longues tâches arrivent suivies de petites tâches.

Average case study

Dans un premier temps, VD-SPT, D-SPT et SRPT ont été testés sur des instances de 10, 15 et 20 tâches. Pour les temps de traitement, nous générons pour chaque tâche,

un temps de traitement entier p_j à partir de la distribution uniforme $U[1, 10]$. De plus, puisque la plage des dates d'arrivée est susceptible d'influencer l'efficacité des algorithmes, les dates d'arrivée sont des entiers générés comme suit [38] :

$$r_1 = 0 \text{ et } r_j = r_{j-1} + X_j, \text{ pour } j = 2, ..., n$$

où X_j est une variable aléatoire générée selon une loi uniforme de distribution $U[0, 10]$. Pour chaque valeur de n, 1000 instances sont générées.

Puisque la configuration *offline* du problème étudié est NP-difficile, la règle SRPT (*Shortest Remaining Processing Time*) est utilisée pour avoir une borne inférieure. Celle-ci est optimale lorsque les préemptions sont autorisées. La valeur de la fonction objectif par SRPT est une borne inférieure de la valeur optimale de la fonction objectif en *offline* et peut donc être utilisée pour trouver le ratio de compétitivité d'un algorithme *online* sur les instances testées. Par conséquent, le ratio de compétitivité d'un algorithme *online* ou *semi-online* sur les instances testées est le ratio de sa valeur objective obtenue par celle obtenue par SRPT.

Les figures 4.3, 4.4 et 4.5 montrent les ratios de compétitivité résultants de l'exécution de 10 instances par les algorithmes D-SPT et VD-SPT. La figure 4.3 montre les résultats pour les problèmes à 10 tâches tandis que les figures 4.4 et 4.5 montrent les résultats pour les problèmes à 15 et 20 tâches, respectivement. Pour le problème à 10 tâches, nous constatons que l'algorithme *semi-online* VD-SPT garantit une meilleure performance que l'algorithme *online* D-SPT, tandis que dans certains cas, comme l'instance 9 (figure 4.3), les deux algorithmes ont approximativement les mêmes performances. Cependant, pour le problème à 15 tâches (figure 4.4), nous remarquons que pour certaines instances, D-SPT garantit une performance légèrement meilleure que VD-SPT. La figure 4.5 montre que les performances des deux algorithmes se maintiennent relativement au même niveau, tandis que pour certaines instances telles que les instances 4 et 7, nous remarquons que VD-SPT garantit de meilleures performances que D-SPT.

Globalement, d'après le tableau 4.2 qui représente la moyenne sur 1000 instances testées, nous pouvons remarquer que VD-SPT offre une meilleure performance que D-SPT pour les problèmes de 10, 15 et 20 tâches. De plus, pour des problèmes de petite taille la différence entre les performances de VD-SPT et D-SPT est plus remarquable que lorsqu'ils sont testés sur des problèmes de plus grande taille.

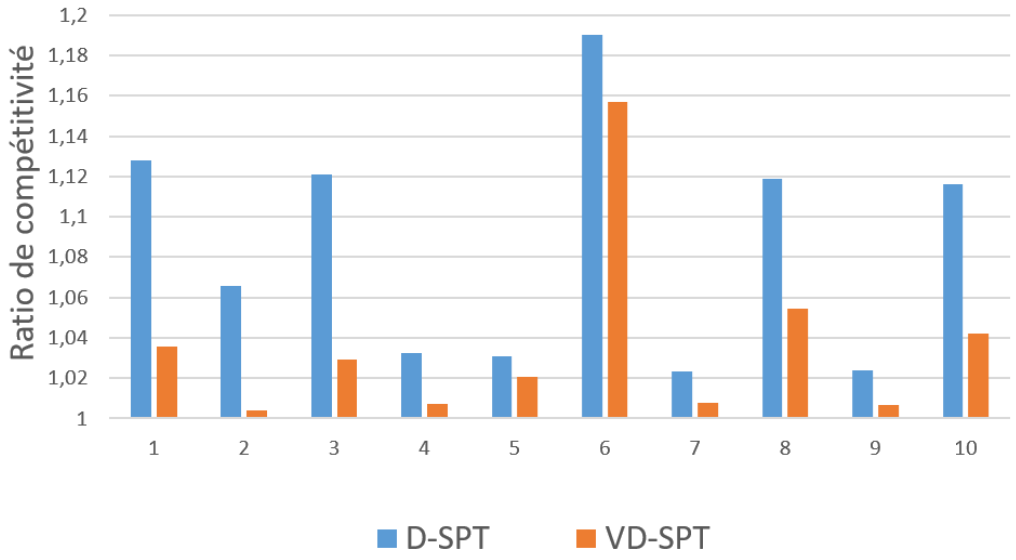

Figure 4.3 – Les ratios de compétitivité pour le problème à 10 tâches

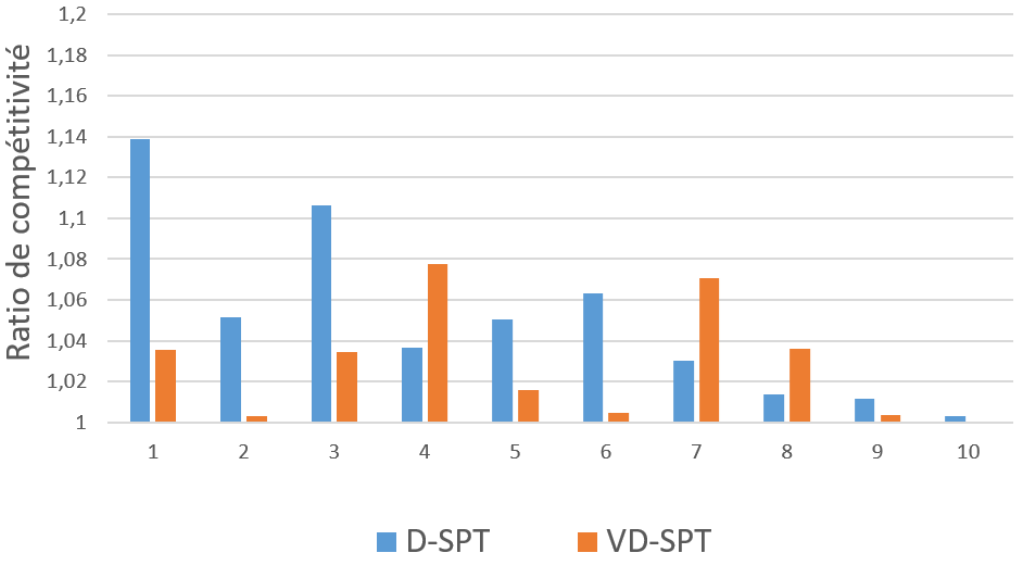

Figure 4.4 – Les ratios de compétitivité pour le problème à 15 tâches

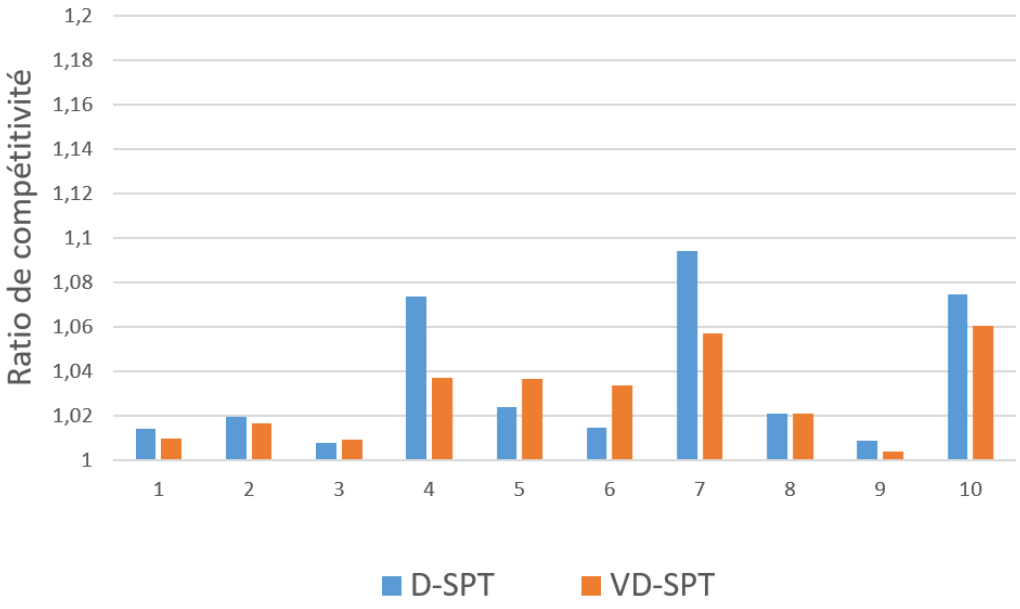

Figure 4.5 – Les ratios de compétitivité pour le problème à 20 tâches

Table 4.2 – Ratios de compétitivité moyens sur 1000 instances testées

	10 tâches	15 tâches	20 tâches
VD-SPT	1.034	1.033	1.037
D-SPT	1.082	1.050	1.039

Table 4.3 – Caractéristiques de l'instance pire-cas

Nombre de tâches	Dates d'arrivée	Temps de traitement
1	22	22
11	23	ϵ
1	45	45
7	46	ϵ
1	91	91
9	92	ϵ

Worst case study

Afin de révéler l'avantage de l'algorithme VD-SPT proposé, une instance spéciale est testée pour laquelle l'algorithme VD-SPT présente un clair avantage. L'instance est caractérisée par des tâches nécessitant un traitement long, qui sont lancées et suivies par des tâches nécessitant un traitement réduit (tableau 4.3). Si nous considérons que la condition de D-SPT est vérifiée pour les tâches longues, c'est-à-dire que $p_j \leq t$, l'algorithme *online* décide de planifier immédiatement la tâche dès son arrivée. Cela signifie que les petites tâches devront attendre que la tâche longue soit terminée puisque la préemption n'est pas autorisée (figure 4.6). D'un autre côté, comme le montre la figure 4.6, VD-SPT choisira d'attendre l'arrivée des petites tâches puisque la condition de t_{max} est vérifiée, ce qui lui donnera l'avantage sur son homologue *online*.

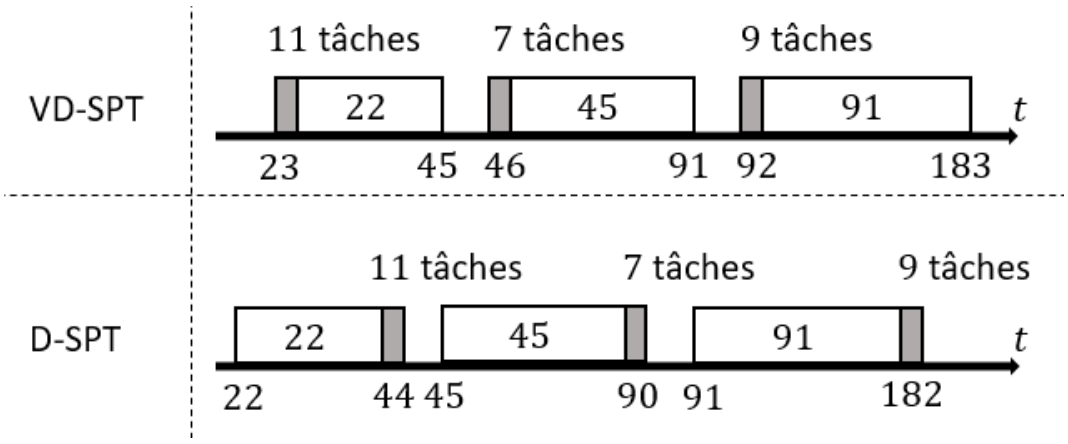

Figure 4.6 – Ordonnancement par VD-SPT et D-SPT pour l'instance pire-cas

La figure 4.7 ci-dessous montre la variation des valeurs de la fonction objective lorsque l'instance est testée sur les algorithmes *online* et *semi-online* et en considérant que $\epsilon \to 0$. Nous pouvons constater que D-SPT fournit une très mauvaise performance par rapport à VD-SPT. De plus, la figure 4.6 montre l'ordonnancement construit par VD-SPT pour l'instance considérée, qui représente également l'ordonnancement optimal dans un environnement *offline*. Par conséquent, pour ce type d'instances, VD-SPT présente une meilleure performance que D-SPT en construisant un ordonnancement proche de l'ordonnancement optimal.

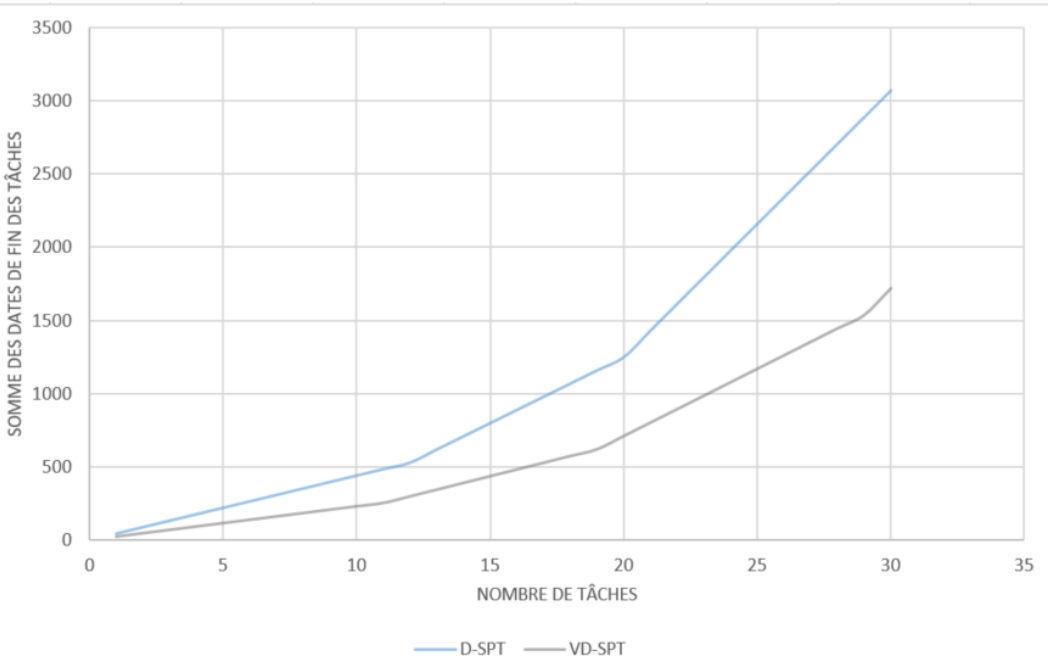

Figure 4.7 – La somme des dates de fin des tâches pour l'instance pire-cas

4.2.3 L'algorithme CSPTn

Nous présentons un nouvel algorithme *semi-online*, appelé *Critical Shortest Processing Time with known number of jobs n* (CSPTn), inspiré de la construction de CSWPT [39], pour le problème de la minimisation de la somme des dates de fin pondérées sur

machine unitaire avec des dates d'arrivée potentielles des tâches connues. Dans notre étude, nous considérons la version avec des poids égaux de CSWPT que nous appelons CSPT (*Critical Shortest Processing Time*). Nous montrons que le nouvel algorithme proposé surpasse VD-SPT sur des instances de grande taille et est plus performant que CSPT pour certaines caractéristiques d'instances. Nous présentons quelques notations qui seront utilisées par la suite :

— Les dates d'arrivée des tâches sont notées $\{t_0, t_1, ..., t_T\}$. Où $0 = t_0 < t_1 < ... < t_T$ et T est l'indice de la date d'arrivée finale, tel que $T \leq n$, et n désigne le nombre total de tâches.

— À chaque date d'arrivée t_k, un ensemble de n_k tâches, noté N_k, sont libérées. tel que $n = \sum_{k=0,..,T} n_k$ et $N = \cup_{k=0}^{T} N_k$ est l'ensemble contenant les n tâches.

— Chaque tâche J_j dans N_k a une date d'arrivée $r_j = t_k$ et un temps de traitement p_j.

— Soit σ et σ^* les ordonnancements obtenus par CSPTn et l'algorithme optimal *offline*, respectivement.

Au début du processus de décision, nous supposons que le décideur connaît les dates d'arrivée des tâches r_j pour $j = \{1, ..., n\}$, ce qui signifie qu'il connaît les dates d'arrivée t_k pour $k = \{0, ..., T\}$ ainsi que le nombre de tâches n_k libérées. Cependant, les temps de traitement des tâches demeurent inconnus.

La différence entre l'algorithme CSPTn proposé et CSPT réside dans la quantité d'informations auxquelles le décideur aura accès. À savoir, CSPTn sait exactement combien de tâches vont arriver à chaque date d'arrivée, ce qui signifie également que le nombre total de tâches n est connu. D'autre part, CSPT connaît les dates d'arrivée potentielles, ce qui signifie que les tâches peuvent ou non arriver et que l'algorithme ne sait pas combien de tâches vont arriver.

La valeur de R^{**}

Considérons le problème suivant $P^M(v)$ qui caractérise une instance avec une tâche J_j avec un temps de traitement p et l'arrivée de petites tâches ou de longues tâches. La tâche J_j arrive à $t_0 = 0$. Les variables du problème sont p et R, où R représente le ratio du pire-cas.

Comme énoncé dans la section précédente, pour un ordonnanceur qui connaît les dates d'arrivée des tâches à l'avance, il peut faire face à deux pires cas, le premier est d'attendre les tâches qui arrivent et il s'avère qu'elles ont le même temps de traitement que J_j, tandis que le second cas est de prendre le risque d'ordonnancer J_j et les tâches qui arrivent s'avèrent avoir des temps de traitement très petits. Le problème peut être formulé comme suit :

$P^M(v)$

$$
\begin{aligned}
&Maximiser \quad R \\
&Sous \quad contraintes \\
&\frac{(n_{s+1}+1)\,(t_s+p)}{(n_{s+1}+1)\,t_{s+1}+p} \geq R, \quad s=0,...,v-1 && (4.14)\\
&\frac{2t_v+(n_v+2)\,p}{2t_0+(n_v+2)\,p} \geq R && (4.15)\\
&p, R \geq 0 && (4.16)
\end{aligned}
$$

L'inégalité (4.14) représente le cas où la tâche J_j commence à l'instant t_s et se termine à l'instant $t_s+p > t_{s+1}$ tandis que n_{s+1} tâches avec des petits temps de traitement égaux à ϵ, avec ϵ une valeur positive infiniment petite, arrivent à l'instant t_{s+1}. Nous faisons tendre ϵ vers 0 dans (4.14). Si la tâche J_j n'était pas en cours de traitement à l'instant t_{s+1}, alors les petites tâches pourraient se terminer à t_{s+1} (Figure 4.8-cas 1).
L'inégalité (4.15) correspond au cas où la tâche J_j commence à t_v alors que n_v tâches avec le même temps de traitement p arrivent à t_v (Figure 4.8-cas 2).
Soit $R^M(v)$ la valeur maximale de R pour le problème $P^M(v)$ et $p^M(v)$ la valeur correspondante de p. La solution est notée $(R^M(v), p^M(v))$. En outre, soit $R^{**} = \max_{1\leq v\leq T}\left\{R^M(v)\right\}$.

Lemme 6. Si (R,p) est une solution réalisable du problème $P^M(v)$ dans lequel soit chaque contrainte de l'équation (4.14) est satisfaite en inégalité stricte, soit la contrainte (4.15) est satisfaite en inégalité stricte, alors $R < R^M(v)$.

Démonstration. Considérons d'abord que chaque contrainte (4.14) est satisfaite en inégalité stricte pour la solution (R,p). Considérons une nouvelle solution :

$$
(R', p') = \left(R + \frac{2t_v\epsilon}{(n_v+2)p(p-\epsilon)}, p-\epsilon\right) \tag{4.17}
$$

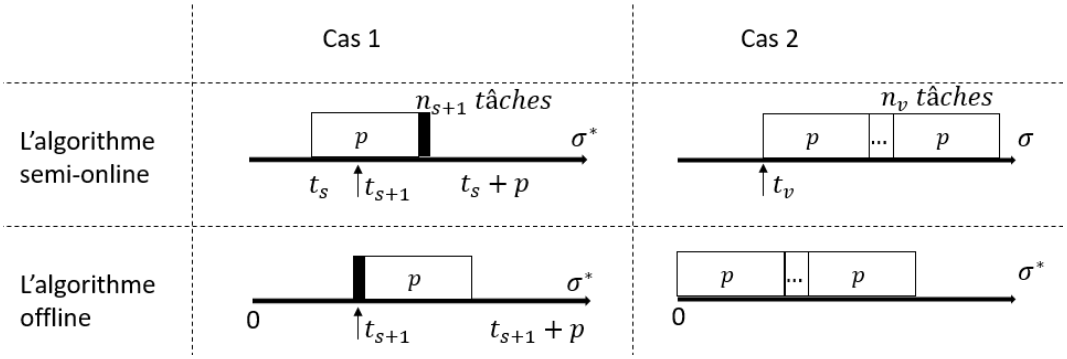

Figure 4.8 – Les deux cas du problème $P^M(v)$

Pour une valeur de $\epsilon > 0$ suffisamment petite. Les contraintes (4.14) restent satisfaites. En outre,

$$\begin{aligned}\frac{2t_v + (n_v+2)(p-\epsilon)}{(n_v+2)(p-\epsilon)} &= \tfrac{2t_v+(n_v+2)p}{(n_v+2)p} + \tfrac{2t_v\epsilon}{(n_v+2)p(p-\epsilon)} \\ &\geq R + \frac{2t_v\epsilon}{(n_v+2)p(p-\epsilon)} \qquad (4.18)\end{aligned}$$

Où l'inégalité (4.18) est obtenue à partir de la faisabilité de (R,p) pour la contrainte (4.15). Ainsi, la solution (R',p') est faisable pour le problème $P^M(v)$.

Alternativement, supposons que la contrainte (4.15) soit satisfaite en inégalité stricte pour la solution (R,p). Considérons une solution :

$$(R',p') = \left(R + \tfrac{(n_{s+1}+1)((n_{s+1}+1)t_{s+1}-t_s)\epsilon}{((n_{s+1}+1)t_{s+1}+p+\epsilon)((n_{s+1}+1)t_{s+1}+p)}, p+\epsilon\right) \qquad (4.19)$$

Pour une valeur de ϵ suffisamment petite, les contraintes (4.14) restent satisfaites car :

$$\frac{(n_{s+1}+1)(t_s+p+\epsilon)}{(n_{s+1}+1)t_{s+1}+p+\epsilon} \qquad (4.20)$$

$$= \frac{(n_{s+1}+1)(t_s+p)}{(n_{s+1}+1)t_{s+1}+p} + \tfrac{(n_{s+1}+1)((n_{s+1}+1)t_{s+1}-t_s)\epsilon}{((n_{s+1}+1)t_{s+1}+p+\epsilon)((n_{s+1}+1)t_{s+1}+p)} \qquad (4.21)$$

$$\geq R + \frac{(n_{s+1}+1)\left((n_{s+1}+1)t_{s+1}-t_s\right)\epsilon}{\left((n_{s+1}+1)t_{s+1}+p+\epsilon\right)\left((n_{s+1}+1)t_{s+1}+p\right)} \qquad (4.22)$$

Où l'inégalité (4.22) découle de la faisabilité de (R, p) pour la contrainte (4.14). De plus, puisque ϵ est choisi suffisamment petit, la contrainte (4.15) reste satisfaite. Ainsi, la solution (R', p') est réalisable pour le problème $P^M(v)$.

Dans tous les cas, nous avons construit une solution réalisable (R', p') au problème $P^M(v)$ dans lequel $R' > R$. Ceci implique que (R, p) n'est pas une solution optimale. Par conséquent, $R^M(v) > R$. □

Corollaire 1. Dans une solution optimale au problème $P^M(v)$, la contrainte (4.15) et au moins une des contraintes (4.14) sont satisfaites à égalité.

Démonstration. Le lemme 6 montre que si les conditions du corollaire ne sont pas satisfaites, il existe une meilleure solution. □

Le corollaire 1 implique que dans une solution optimale de $P^M(v)$, certaines contraintes s de l'équation (4.14) et la contrainte (4.15) sont satisfaites à égalité.

En étudiant $\frac{(n_{s+1}+1)(t_s+p)}{(n_{s+1}+1)t_{s+1}+p} = R_s^M(v)$ et $\frac{2t_v+(n_v+2)p}{2t_0+(n_v+2)p} = R_s^M(v)$, nous obtenons :

$$R_s^M(v) = \frac{\left[\begin{array}{c}(n_{s+1}+1)(n_v+2)(t_{s+1}+t_s) - 2t_v + \\ \sqrt{\left((n_{s+1}+1)(n_v+2)(t_{s+1}-t_s)+2t_v\right)^2 \\ +8n_{s+1}(n_{s+1}+1)(n_v+2)t_v t_{s+1}}\end{array}\right]}{2(n_{s+1}+1)(n_v+2)t_{s+1}} \tag{4.23}$$

Théorème 8.

$$R^{**} = \max_{v=1,\ldots,T}\left\{\min_{s=0,\ldots,v-1}\left\{R_s^M(v)\right\}\right\} \tag{4.24}$$

Démonstration. Il suffit de démontrer que $R^M(v) = \min_{s=0,\ldots,v-1}\left\{R_s^M(v)\right\}$.
Considérons deux solutions faisables $(R_{s_1}^M(v), p_{s_1}^M(v))$ et $(R_{s_2}^M(v), p_{s_2}^M(v))$ qui valident les conditions du Corollaire 1 pour le problème $P^M(v)$. Pour ces deux solutions, nous avons :

$$R_s^M(v) = \frac{(n_{s+1}+1)\left(t_s + p_s^M(v)\right)}{(n_{s+1}+1)\,t_{s+1} + p_s^M(v)} \tag{4.25}$$

Et

$$R_s^M(v) = \frac{2t_v + (n_v+2)\,p_s^M(v)}{(n_v+2)\,p_s^M(v)} \quad \textit{pour } s = s_1, s_2 \tag{4.26}$$

Supposons que $R_{s_1}^M(v) < R_{s_2}^M(v)$ pour $s = s_1$. Ceci implique que $p_{s_1}^M(v) > p_{s_2}^M(v)$. Nous montrons que la solution $(R_{s_2}^M(v), p_{s_2}^M(v))$ n'est pas faisable pour le problème $P^M(v)$.

La faisabilité de $(R_{s_1}^M(v), p_{s_1}^M(v))$ implique que :

$$t_{s_1} = \frac{\left((n_{s+1}+1)t_{s_1+1} + p_{s_1}^M(v)\right) R_{s_1}^M(v) - (n_{s+1}+1)p_{s_1}^M(v)}{n_{s+1}+1} \tag{4.27}$$

Alors,

$$\begin{aligned}
&\tfrac{(n_{s+1}+1)\left(t_{s_1}+p_{s_2}^M(v)\right)}{(n_{s+1}+1)t_{s_1+1}+p_{s_2}^M(v)} \\
&= \tfrac{\left((n_{s+1}+1)t_{s_1+1}+p_{s_1}^M(v)\right)R_{s_1}^M(v)+(n_{s+1}+1)(p_{s_2}^M(v)-p_{s_1}^M(v))}{(n_{s+1}+1)t_{s_1+1}+p_{s_2}^M(v)} && (4.28)\\
&= R_{s_1}^M(v) + \tfrac{(p_{s_2}^M(v)-p_{s_1}^M(v))(n_{s+1}+1-R_{s_1}^M(v))}{(n_{s+1}+1)t_{s_1+1}+p_{s_2}^M(v)} && (4.29)\\
&< R_{s_1}^M(v) < R_{s_2}^M(v) && (4.30)
\end{aligned}$$

Par conséquent, la solution $(R_{s_2}^M(v), p_{s_2}^M(v))$ est infaisable pour le problème $P^M(v)$ pour $s = s_1$, ce qui est une contradiction. □

CSPTn

Tout d'abord, la valeur de R^{**} est calculée selon le Théorème 8. Ensuite, à un instant t, lorsque la machine est inactive et que certaines tâches sont disponibles, choisir la tâche J_j, avec le plus petit temps de traitement p_j, parmi toutes les tâches disponibles. En

cas d'égalité, choisir la tâche avec le plus petit indice. De plus, notons par t_k la date d'arrivée suivante. Ensuite, si

$$t + p_j \leq R^{**}t_k \tag{4.31}$$

ordonnancer J_j à l'instant t. Sinon, attendre la date d'arrivée suivante et répéter la procédure.

L'expression $R^{**}t_k$ représente ici la date de fin critique de la tâche J_j. Si, en ordonnançant J_j, la date de fin critique est dépassée, alors CSPTn attend la prochaine arrivée. Toutefois, si J_j peut être accomplie avant la date de fin critique, elle est alors ordonnancée immédiatement.

4.2.4 Étude expérimentale de CSPTn

Dans cette section, la performance de CSPTn sur différentes types d'instances est évaluée et une comparaison avec les meilleurs algorithmes *online* et *semi-online* existants est établie : D-SPT, CSPT, et VD-SPT.

Les instances sont générées en utilisant le schéma de génération proposé par Tao et al. [86]. Les dates d'arrivée des tâches sont générées selon un processus de Poisson avec un paramètre $\lambda = \{0.5, 1, 3\}$. Les temps de traitement des tâches sont générés selon une distribution uniforme. Cependant, afin de capturer au mieux les charges de travail réelles et de mettre en évidence les avantages de chaque algorithme, des instances présentant une forte variabilité en termes de temps de traitement sont générées. Trois catégories sont considérées :

— Type 1 tâches : Petites tâches avec $p_j \sim U[1, \frac{n}{\alpha_1}]$.

— Type 2 tâches : Tâches longues avec $p_j \sim U[\alpha_2 n, \alpha_3 n]$.

— Type 3 tâches : Tâches uniformes avec $p_j \sim U[1, \alpha_4 n]$.

Les valeurs $\alpha_i \geq 1$, pour $i = \{1, 2, 3, 4\}$ et $\alpha_2 < \alpha_3$, sont des paramètres de simulation à définir. Des instances de différentes tailles sont considérées avec $n \in \{10, 30, 50, 100\}$. Pour chaque taille d'instance, 1000 instances sont générées aléatoirement et le ratio de compétitivité moyen sur les instances testées est calculé.

Les algorithmes ont été codés en C++ sur Microsoft Visual Studio et exécutés sur un processeur Intel i5-8350U à $1,7$ GHz.

Tout d'abord, les algorithmes sont testés sur des tâches uniformes de type 3 avec $\alpha_4 \in \{1, 2\}$ pour les trois catégories d'arrivée des tâches. La figure 4.9 montre les résultats expérimentaux pour $\alpha_4 = 1$. Nous remarquons que pour le problème à 10 tâches, VD-SPT montre de bonnes performances, alors que pour les instances de plus grande taille, les performances se dégradent avec une différence significative par rapport aux autres algorithmes. De plus, D-SPT, CSPT, et CSPTn montrent une performance rapprochée avec un avantage pour CSPTn sur le problème à 10 tâches pour $\lambda = 3$ alors que CSPT prend l'avantage sur les autres catégories d'arrivée des tâches. Il faut noter que pour les problèmes de plus grande taille, les performances des algorithmes deviennent proche de 1. Les mêmes conclusions peuvent être faites pour $\lambda_4 = 2$.
Pour la deuxième configuration, l'idée était d'illustrer la variabilité de la charge de travail en générant 70% de tâches de type 1 avec 30% de tâches de type 2. Cette configuration permet de comparer au mieux les stratégies d'attente employées par les différents algorithmes et montre ainsi l'avantage des nouveaux algorithmes *semi-online* par rapport à D-SPT.

Les figures 4.10 et 4.11 montrent les résultats expérimentaux pour $\alpha_1 \in \{5, 10\}$, $\alpha_2 = 2$, et $\alpha_3 = 3$. Nous pouvons clairement remarquer l'avantage de l'algorithme CSTPn proposé par rapport aux algorithmes existants. De plus, D-SPT n'est pas très performant pour ce type de configuration. En outre, nous pouvons constater que pour une valeur plus grande de α_1, c'est-à-dire que les temps de traitement des tâches de type 1 sont plus petits, la performance de CSPTn s'améliore significativement par rapport aux autres algorithmes, en particulier pour la catégorie d'arrivée des tâches $\lambda = 3$. Le tableau 4.4 présente les améliorations de CSPTn par rapport aux autres algorithmes existants pour les instances présentant une grande variabilité en termes de temps de traitement et pour la catégorie d'arrivée des tâches $\lambda = 3$. Les pourcentages représentés dans le tableau montrent l'écart entre la solution de l'algorithme considéré et la meilleure solution obtenue par les algorithmes simulés.

Table 4.4 – Écart par rapport à la meilleure solution pour les instances à forte variabilité et pour $\lambda = 3$

n	CSPTn	CSPT	D-SPT	VD-SPT
10	0.0%	15.9%	16.0%	11.7%
30	0.1%	7.3%	7.1%	5.0%
50	0.0%	6.4%	6.3%	8.9%
100	0.0%	0.1%	0.1%	5.7%

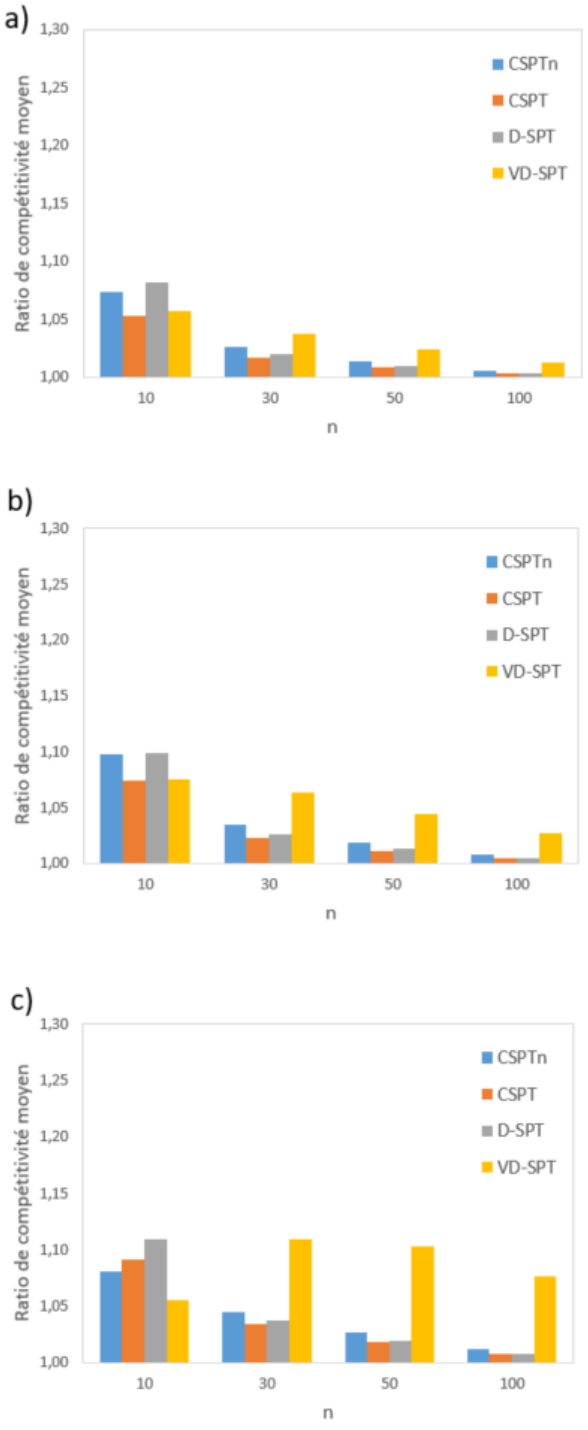

Figure 4.9 – Ratios de compétitivité moyens pour différentes tailles d'instances, des tâches de type 3 pour $\alpha_4 = 1$, et trois catégories d'arrivée, $\lambda = 0.5$ (a), $\lambda = 1$ (b), $\lambda = 3$ (c)

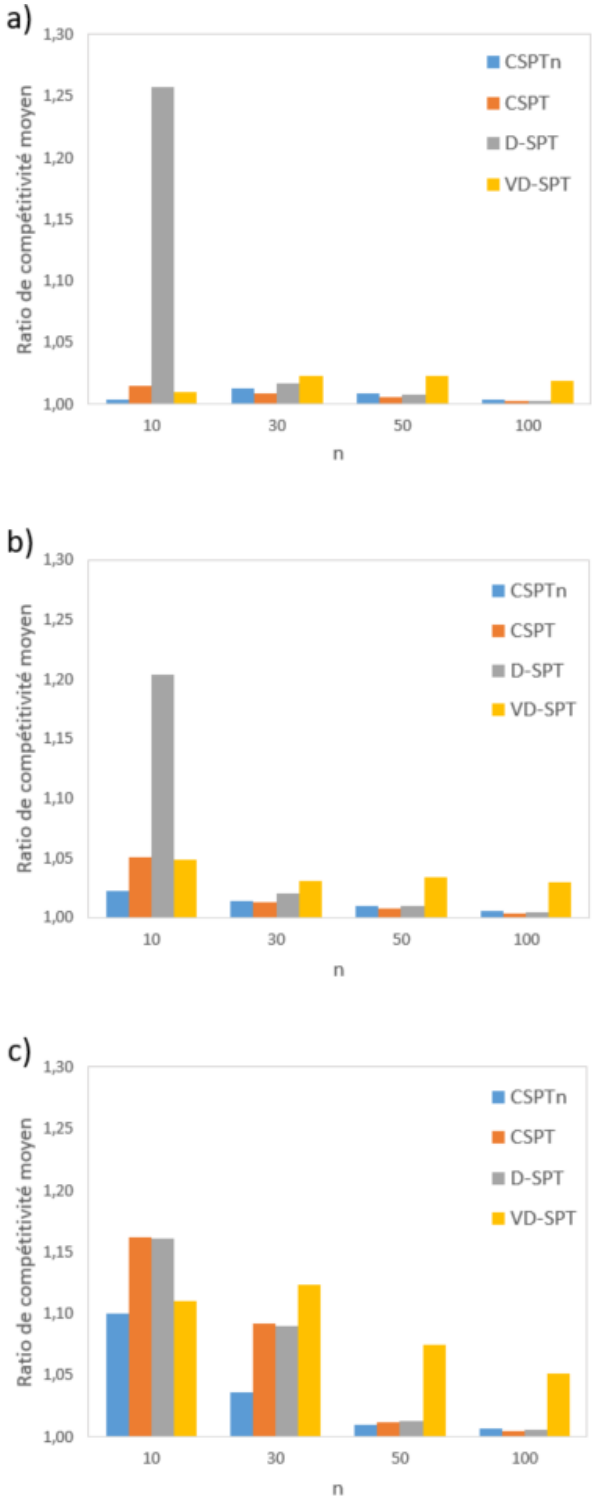

Figure 4.10 – Ratios de compétitivité moyens pour différentes tailles d'instances, 70% de tâches de type 1 et 30% de tâches de type 2 pour $\alpha_1 = 5$, et trois catégories d'arrivée, $\lambda = 0.5$ (a), $\lambda = 1$ (b), $\lambda = 3$ (c)

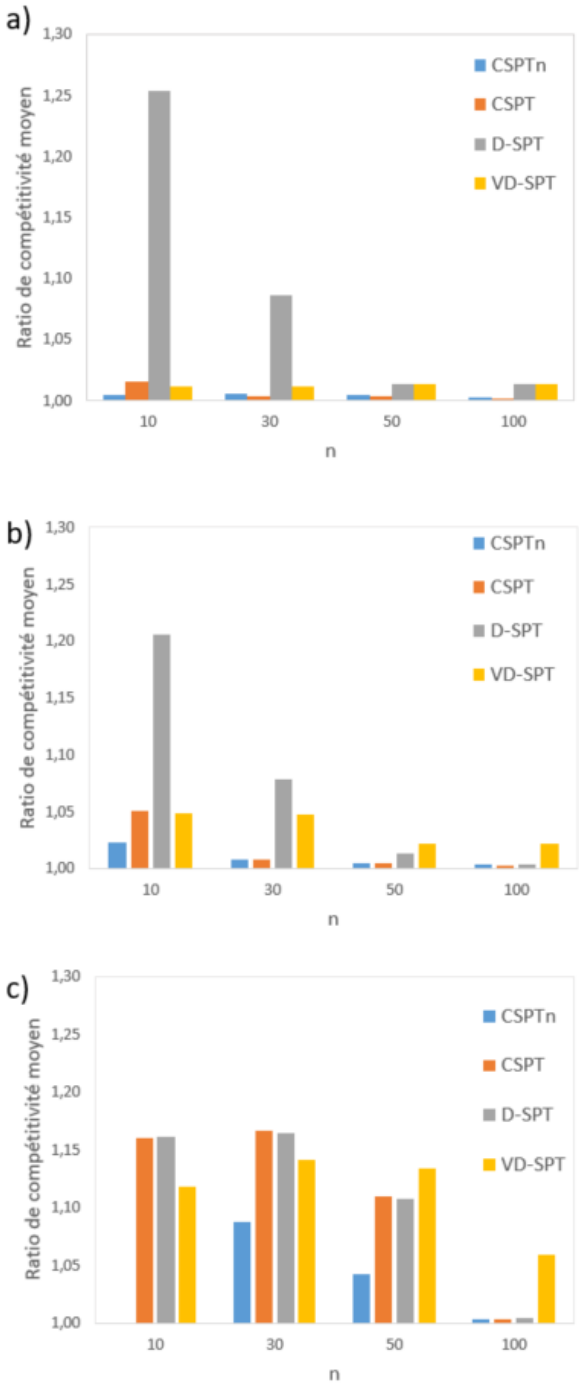

Figure 4.11 – Ratios de compétitivité moyens pour différentes tailles d'instances, 70% de tâches de type 1 et 30% de tâches de type 2 pour $\alpha_1 = 10$, et $\lambda = 0.5$ (a), $\lambda = 1$ (b), $\lambda = 3$ (c)

4.2.5 Borne inférieure pour le problème $1|online, known\ r_j|\sum C_j$

Après deux résultats préliminaires, Nous démontrons la borne inférieure sur le ratio de compétitivité pour le problème $1|online, known\ r_j|\sum C_j$, où les dates d'arrivée des tâches sont connues à l'avance.

Lemme 7. Il existe un ordonnancement optimal dans lequel chaque tâche commence soit à la date de fin d'une autre tâche, soit à l'une des dates d'arrivée $t_0, ..., t_T$ [39].

Démonstration. Considérons une certaine tâche qui commence son exécution à l'instant t, où $t_k < t < t_{k+1}$ pour $0 \leq k \leq T$, et que la machine soit inactive immédiatement avant l'instant t. Si cette tâche commence plus tôt dans la période d'inactivité précédant immédiatement t mais pas avant t_k, alors la somme des dates de fin des tâches n'augmente pas. De plus, toute planification des tâches après l'instant t dans l'ordonnancement original reste faisable lorsque la tâche sélectionnée commence plus tôt. □

Lemme 8. $R^M(v) > 1$ et $p^M(v) > t_{s+1} - t_s$, pour $0 \leq s \leq v-1$

Démonstration. La valeur maximale de R est égale au minimum des parties gauches des contraintes (4.14) et (4.15). Si p est choisi de telle sorte que $p > t_{s+1} - t_s$ pour $0 \leq s \leq v-1$, alors chacune de ces parties gauches dépasse 1. Cependant, si $p \leq t_{s+1} - t_s$ pour une certaine valeur de s où $0 \leq s \leq v-1$, alors R n'est pas maximisée. □

Théorème 9. Pour le problème $1|online, known\ r_j|\sum C_j$ où les dates d'arrivée des tâches sont connues à l'avance, aucun algorithme *semi-online* ne peut garantir un ratio de compétitivité meilleur que R^{**}.

Démonstration. Considérons une première tâche J_1 qui arrive à $r_1 = t_0 = 0$ avec un temps de traitement $p_1 = p$. Un algorithme *semi-online* doit impérativement commencer l'ordonnancement à une date d'arrivée selon le lemme 7. Supposons que l'algorithme *semi-online* ordonnance J_1 à une date d'arrivée t_s avec $0 \leq s \leq T$.
Premièrement, supposons que $s < v$. Dans ce cas, l'adversaire lance n_{s+1} tâches à l'instant t_{s+1} avec des temps de traitement les plus petits possible $p_j = \epsilon$, avec ϵ une

valeur positive infiniment petite. Selon le lemme 8, $p > t_{s+1} - t_s$, ce qui implique que la tâche J_1 n'a pas encore fini quand les n_{s+1} tâches arrivent.

Quand $\epsilon \to 0$, les n_{s+1} tâches finissent le traitement à $t_s + p$ dans l'ordonnancement par l'algorithme *semi-online*, alors que l'algorithme *offline* optimal planifie les n_{s+1} tâches à t_{s+1} suivis de J_1. Ainsi,

$$\frac{\sum C_j}{\sum C_j^*} = \frac{(t_s + p)(n_{s+1} + 1)}{(n_{s+1} + 1)t_{s+1} + p} \geq R^{**} \tag{4.32}$$

Où la dernière inégalité est obtenue de la contrainte (4.14) du problème $P^M(v)$.

Alternativement, considérons le cas où $s \geq v$. L'adversaire lance n_v tâches avec des temps de traitements égaux au temps de traitement de J_1 ($p_j = p$). L'algorithme *semi-online* commence l'ordonnancement à partir de $t_s \geq t_v$, alors que l'algorithme *offline* optimal commence dans les meilleurs cas à $t_0 = 0$. Nous aurons alors,

$$\frac{\sum C_j}{\sum C_j^*} \geq \frac{(t_v + p)(n_v + 1) + \frac{n_v(n_v+1)}{2}p}{(t_0 + p)(n_v + 1) + \frac{n_v(n_v+1)}{2}p} = \frac{2t_v + (n_v + 2)p}{(n_v + 2)p} \geq R^{**} \tag{4.33}$$

De même, la dernière inégalité est obtenue de la contrainte (4.15) du problème $P^M(v)$. Ainsi, quel que soit le choix retenu par l'algorithme *semi-online*, $\frac{\sum C_j}{\sum C_j^*}$ peut être rendu arbitrairement proche d'une valeur supérieure ou égale à R^{**}. Ainsi, aucun algorithme d'ordonnancement *semi-online* ne peut avoir un ratio de compétitivité inférieur à R^{**}. □

Nous pouvons ainsi conclure que l'information partielle sur les dates d'arrivée des tâches est une information qui peut être utile pour le problème de minimisation de la somme des dates de fin des tâches sur machine unitaire.

Dans les sections suivantes, nous étudions des variantes *semi-online* du problème de minimisation de la somme des dates de fin pondérées des tâches, où chaque tâche J_j a un poids w_j. Ce poids reflète l'importance d'une tâche par rapport aux autres tâches de l'instance. À titre d'exemple, Il peut être traduit comme le coût de maintien ou d'inventaire de la tâche dans le système.

4.3 Problème $1|online, known\ r_j|\sum w_jC_j$

Dans cette section nous étudions le problème $1|online, known\ r_j|\sum w_jC_j$. Nous vérifions pour ce problème si l'information sur les dates d'arrivée des tâches reste toujours utile si chaque tâche a un poids différent.

Comme nous l'avons précisé dans l'introduction de ce chapitre. Hall et al. [39] ont étudié le problème $1|online, r_j, t_k|\sum w_jC_j$ où les dates d'arrivée potentielles des tâches sont connues à l'avance. Ils ont prouvé que la borne inférieure du problème est égale à R^* où $R^* < 2$ est une valeur qui dépend des paramètres de l'information partielle du problème. Ainsi, la présente étude nous permet de vérifier l'utilité de l'information sur les dates d'arrivée exactes des tâches en formulant le problème $P'(v)$ qui reprend les pires scénarios que peut rencontrer un algorithme *semi-online* pour le problème.

Formulation du problème $P'(v)$

Afin de formuler le problème $P'(v)$ pour $v = \{1, ..., T\}$, nous étudions des situations avec des instances composées de deux dates d'arrivée, puis trois dates d'arrivée, pour construire finalement une généralisation des scénarios qui représentent les pires-cas que peut rencontrer un algorithme *semi-online* pour le problème étudié. La construction des instances se base sur la méthode de l'adversaire. Comme défini dans le chapitre 1, cette méthode permet de maximiser le ratio de compétitivité d'un algorithme *semi-online* en le confrontant à des instances pire-cas.

a) Instance avec deux dates d'arrivée t_0 et t_1

Considérons une instance avec deux dates d'arrivée t_0 et t_1. À $t_0 = 0$, une première tâche J_1 arrive avec $(p_1, w_1) = (p, W)$ et $\rho_1 = \rho = \frac{W}{p}$. À l'instant $t_1 < t_0 + p$, l'adversaire doit faire un choix qui va permettre de diminuer la performance d'un algorithme *semi-online* par rapport à sa performance en servant l'instance en mode *offline*. Ce choix va dépendre de la décision prise par l'algorithme *semi-online* quant à l'ordonnancement de la tâche J_1. En ce qui suit, nous étudions deux instance I_1^1 et I_2^1 qui représentent les deux pires-cas possibles.

Instance I_1^1 : L'algorithme *semi-online* ordonnance la tâche J_1 à l'instant t_0 (Figure 4.12)

Dans ce cas, l'adversaire lance $n_1 \leq n-1$ tâches à l'instant t_1 avec $(p_j, w_j) = (\epsilon, W')$

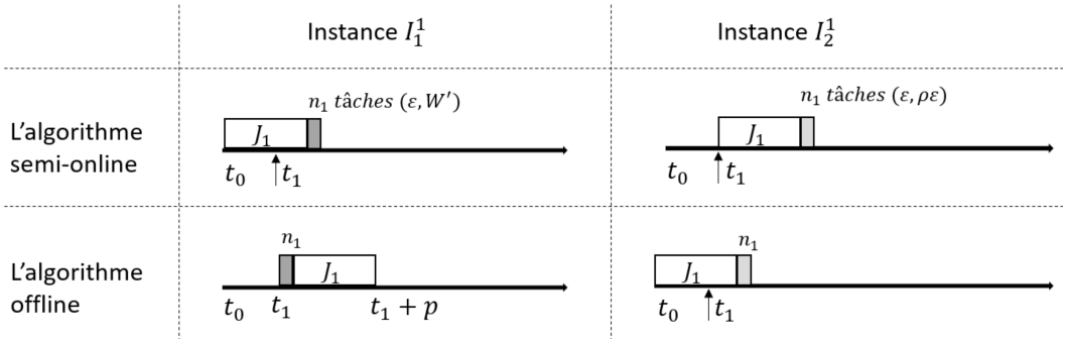

Figure 4.12 – Ordonnancements *semi-online* et *offline* pour les instances I_1^1 et I_2^1

pour $j \in \{2, ..., n_1\}$, $W' >> W$ et ϵ une valeur positive infiniment petite.
Pour $\epsilon \to 0$, nous avons

$$\sum w_jC_j(I_1^1) = Wp + n_1W'p \tag{4.34}$$

$$\sum w_jC_j^*(I_1^1) = n_1W't_1 + (t_1 + p)W \tag{4.35}$$

Ainsi,

$$\frac{\sum w_jC_j(I_1^1)}{\sum w_jC_j^*(I_1^1)} = \frac{Wp + n_1W'p}{n_1W't_1 + (t_1 + p)W} \tag{4.36}$$

Quand $W' \to \infty$,

$$\frac{\sum w_jC_j(I_1^1)}{\sum w_jC_j^*(I_1^1)} \leq \frac{p}{t_1} \tag{4.37}$$

Instance I_2^1 : l'algorithme *semi-online* attend jusqu'à l'instant t_1 (Figure 4.12)

Dans ce cas, l'adversaire lance n_1 tâches à l'instant t_1 qui ont la même importance que la tâche J_1, c'est-à-dire $\rho_j = \rho$. Afin de maximiser le ratio de compétitivité, les temps de traitement de ces n_1 tâches sont choisis de manière à ce qu'ils soient les plus petits possibles $(p_j, w_j) = (\epsilon, \rho\epsilon)$. Puisque il n'y a plus de dates d'arrivée, l'algorithme *semi-online* ordonnance J_1 à t_1 suivie des n_1 tâches, tandis qu'en *offline* la tâche J_1 est ordonnancée dès son arrivée suivie des n_1 tâches.

$$\sum w_jC_j(I_2^1) = (t_1 + p)W + n_1(t_1 + p)\rho\epsilon + \frac{n_1(n_1 + 1)}{2}\rho\epsilon^2 \tag{4.38}$$

$$\sum w_jC_j^*(I_2^1) = pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 \tag{4.39}$$

Ainsi,

$$\frac{\sum w_jC_j(I_2^1)}{\sum w_jC_j^*(I_2^1)} = \frac{(t_1+p)W + n_1(t_1+p)\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2}{pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2} \tag{4.40}$$

Quand $\epsilon \to 0$, nous aurons

$$\frac{\sum w_jC_j(I_2^1)}{\sum w_jC_j^*(I_2^1)} \leq 1 + \frac{t_1}{p} \tag{4.41}$$

b) Instance avec trois dates d'arrivée t_0, t_1 et t_2

Quatre nouveaux cas peuvent être dérivés des deux cas étudiés précédemment. Deux de ces cas suivent le premier cas où l'algorithme *semi-online* décide d'ordonnancer J_1 à t_0 et n_1 tâches avec $(p_j, w_j) = (\epsilon, W')$ arrivent à t_1. Ici l'adversaire peut libérer n_2 tâches à t_2 avec soit $(p_j, w_j) = (\epsilon, W')$ soit $(p_j, w_j) = (\epsilon, \epsilon\rho)$. Quel que soit le cas, la décision prise par l'algorithme optimal *offline* serait similaire à celle de l'algorithme *semi-online*. Par conséquent, il est inutile d'étudier ces deux cas. Ainsi, nous n'étudions que les deux cas qui peuvent être dérivés du deuxième cas où l'algorithme *semi-online* décide d'attendre jusqu'à l'instant t_1 où n_1 tâches avec $(p_j, w_j) = (\epsilon, \epsilon\rho)$ arrivent. Nous étudions alors les deux instances pire-cas $I_{2.1}^1$ et $I_{2.2}^1$.

Instance $I_{2.1}^1$: l'algorithme *semi-online* ordonnance la tâche J_1 à l'instant t_1 (Figure 4.13)

Les n_1 tâches qui arrivent à l'instant t_1 sont ordonnancées dès leur arrivée puisqu'elles finissent le traitement à t_1 quand $\epsilon \to 0$. Ainsi, dans le cas où l'algorithme *semi-online* décide d'ordonnancer la tâche J_1 à t_1, l'adversaire lance $n_2 < n - 1 - n_1$ tâches à t_2 avec $(p_j, w_j) = (\epsilon, W')$ pour $j \in \{n_1 + 1, ..., n_2\}$, $W' >> W$ et ϵ une valeur positive infiniment petite. De plus, t_2 est choisie de manière à ce que sa valeur soit supérieure à $t_0 + p$ ($t_2 > t_0 + p$), puisque dans le cas où $t_1 < t_2 \leq t_0 + p$ les performances de l'algorithme *semi-online* peuvent être dégradées si l'adversaire considère $t_2 = t_1$ ce qui revient à étudier l'instance I_1^1.
L'algorithme *offline* optimal ordonnance la tâche J_1 à t_0 suivie des n_1 tâches qui arrivent à t_1 suivies des n_2 tâches à t_2.

$$\sum w_jC_j(I^1_{2.1}) = n_1t_1\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2(t_1+n_1\epsilon+p)W' + \frac{n_2(n_2+1)}{2}\epsilon W' \quad (4.42)$$

$$\sum w_jC^*_j(I^1_{2.1}) = pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2t_2W' + \frac{n_2(n_2+1)}{2}\epsilon W' \quad (4.43)$$

Ainsi,

$$\frac{\sum w_jC_j(I^1_{2.1})}{\sum w_jC^*_j(I^1_{2.1})} = \frac{n_1t_1\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2(t_1+n_1\epsilon+p)W' + \frac{n_2(n_2+1)}{2}\epsilon W'}{pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2t_2W' + \frac{n_2(n_2+1)}{2}\epsilon W'} \quad (4.44)$$

Quand $W' \to \infty$ et $\epsilon \to 0$, nous aurons

$$\frac{\sum w_jC_j(I^1_{2.1})}{\sum w_jC^*_j(I^1_{2.1})} \leq \frac{t_1+p}{t_2} \quad (4.45)$$

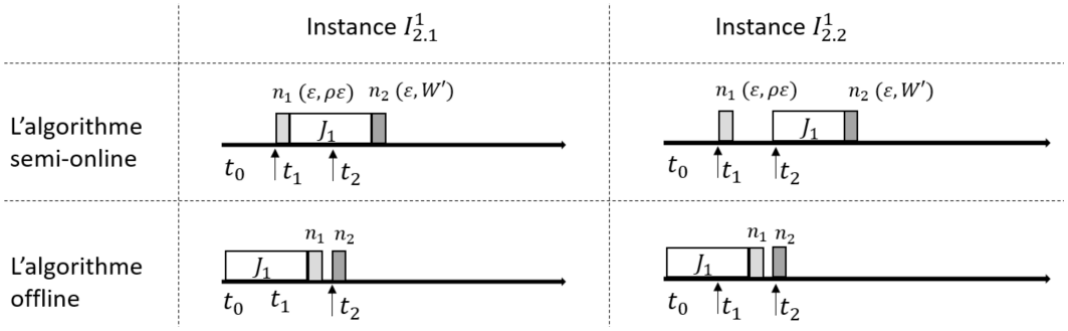

Figure 4.13 – Ordonnancements *semi-online* et *offline* pour les instances $I^1_{2.1}$ et $I^1_{2.2}$

Instance $I^1_{2.2}$: l'algorithme *semi-online* attend jusqu'à t_2 (Figure 4.13)

Dans ce cas, l'adversaire lance n_2 tâches à l'instant t_2 avec la même importance que J_2 mais avec $(p_j, w_j) = (\epsilon, \rho\epsilon)$, pour $j \in \{n_1+1, ..., n_2\}$. L'algorithme *semi-online* ordonnance ainsi les n_1 tâches à t_1 et la tâche J_1 à t_2 suivie des n_2 tâches, tandis que l'algorithme *offline* optimal ordonnance J_1 à son arrivée, ensuite les n_1 tâches à t_1, puis les n_2 tâches à t_2.

$$\sum w_jC_j(I^1_{2.2}) = n_1t_1\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + (t_2+p)W + n_2(t_2+p)\rho\epsilon + \frac{n_2(n_2+1)}{2}\rho\epsilon^2 \quad (4.46)$$

$$\sum w_jC_j^*(I_{2.2}^1) = pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2t_2\rho\epsilon + \frac{n_2(n_2+1)}{2}\rho\epsilon^2 \tag{4.47}$$

Ainsi,

$$\frac{\sum w_jC_j(I_{2.2}^1)}{\sum w_jC_j^*(I_{2.2}^1)} = \frac{n_1t_1\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + (t_2+p)W + n_2(t_2+p)\rho\epsilon + \frac{n_2(n_2+1)}{2}\rho\epsilon^2}{pW + n_1p\rho\epsilon + \frac{n_1(n_1+1)}{2}\rho\epsilon^2 + n_2t_2\rho\epsilon + \frac{n_2(n_2+1)}{2}\rho\epsilon^2} \tag{4.48}$$

Quand $\epsilon \to 0$, nous aurons

$$\frac{\sum w_jC_j(I_{2.2}^1)}{\sum w_jC_j^*(I_{2.2}^1)} \leq 1 + \frac{t_2}{p} \tag{4.49}$$

Généralisation du problème $P'(v)$:
À partir des équations (4.37), (4.41), (4.45) et (4.49), nous présentons la généralisation du problème qui maximise le ratio de compétitivité R, noté $P'(v)$ pour $v = \{1, ..., T\}$.

$P'(v)$

$$\begin{aligned}
&Maximiser \quad R \\
&Sous \quad contraintes \\
&\frac{t_s + p}{t_{s+1}} \geq R \quad for \quad s = \{0, ..., v-1\} && (4.50)\\
&\frac{t_v + p}{p} \geq R && (4.51)\\
&p, R \geq 0 && (4.52)
\end{aligned}$$

La contrainte (4.50) représente le cas où une tâche J_j, avec $(p_j, w_j) = (p, W)$ et $\rho_j = \rho = \frac{W}{p}$, est ordonnancée à l'instant t_s et se termine à $t_s + p > t_{s+1}$, puis des tâches avec $(p_j, w_j) = (\epsilon, W')$ et $W' >> W$ arrivent à t_{s+1}. Puisque ces tâches ont des poids dominants, le résultat de la contrainte (4.50) est obtenu en considérant $W' \to \infty$. La contrainte (4.52) correspond à une situation où la tâche J_j est ordonnancée à t_v alors que des tâches avec $(p_j, w_j) = (\epsilon, \epsilon\rho)$ arrivent.

Nous pouvons conclure à ce niveau que l'information sur les dates d'arrivée exacte des tâches pour le problème *semi-online* de minimisation de la somme des dates de fin pondérées sur machine unitaire n'apporte aucun avantage dans la prise de décision. En effet, la borne inférieure sur le ratio de compétitivité du problème $1|online, known\ r_j|\sum w_jC_j$ ne serait pas différente de R^* obtenue par Hall et al. [39], puisque les contraintes du problème $P'(v)$ sont les mêmes que les contraintes du problème $P(v)$.

4.4 Problème $1|online, r_j, t_k, p_{min}|\sum w_j C_j$

Dans cette section, nous étudions le problème *semi-online* de minimisation de la somme des dates de fin pondérées sur machine unitaire avec des dates d'arrivée potentielles des tâches connues à l'avance ainsi qu'une borne inférieure p_{min} sur le plus petit temps de traitement dans l'instance. Contrairement au problème précédant où le décideur connaît exactement combien de tâches vont arriver à chaque instant t_k pour $k = \{0, ..., T\}$, dans le problème considéré dans cette section, les instants t_k sont des dates d'arrivée potentielles, c'est-à-dire que le nombre de tâches qui vont arriver à ces instants n'est pas connu et il est tout à fait possible qu'aucune tâche n'arrive à un instant t_k. En ce qui suit, nous étudions l'utilité de ces informations, à savoir les dates d'arrivée potentielles et la borne p_{min}, en formulant le problème $P''(v)$.

4.4.1 Formulation du problème $P''(v)$

Nous étudions des situations avec deux dates d'arrivée, puis trois dates d'arrivée pour construire finalement une formulation du problème $P''(v)$ qui représente une généralisation des pires scénarios que peut rencontrer un algorithme *semi-online* pour le problème.

a) Instance avec deux dates d'arrivée t_0 et t_1

Considérons une instance avec deux dates d'arrivée t_0 et t_1, où à l'instant $t_0 = 0$ une première tâche arrive avec $(p_1, w_1) = (p, W)$ et $\rho_1 = \frac{W}{p}$. À l'instant $t_1 < t_0 + p$, l'adversaire prend une décision selon le choix pris par l'algorithme *semi-online* concernant la tâche J_1.

Instance I_1^2 : L'algorithme *semi-online* ordonnance la tâche J_1 à t_0 (Figure 4.14)

Dans ce cas, l'adversaire lance une seconde tâches J_2 à l'instant t_1 avec $(p_2, w_2) = (p_{min}, W')$, où $W' >> W$. L'algorithme *offline* optimal ordonnance ainsi la tâche J_2 à son arrivée suivie de la tâche J_1 pour une valeur de W' suffisamment grande.

$$\sum w_j C_j(I_1^2) = (t_0 + p)W + (t_0 + p + p_{min})W' \tag{4.53}$$

$$\sum w_j C_j^*(I_1^2) = (t_1 + p_{min})W' + (t_1 + p_{min} + p)W \tag{4.54}$$

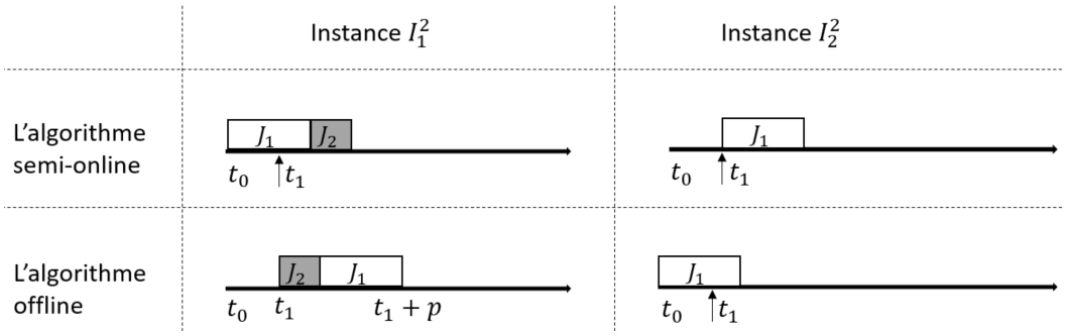

Figure 4.14 – Ordonnancements *semi-online* et *offline* pour les instances I_1^2 et I_2^2

Ainsi,

$$\frac{\sum w_j C_j(I_1^2)}{\sum w_j C_j^*(I_1^2)} = \frac{(t_0+p)W + (t_0+p+p_{min})W'}{(t_1+p_{min})W' + (t_1+p_{min}+p)W} \tag{4.55}$$

Quand $W' \to \infty$,

$$\frac{\sum w_j C_j(I_1^2)}{\sum w_j C_j^*(I_1^2)} \leq \frac{t_0+p+p_{min}}{t_1+p_{min}} \tag{4.56}$$

Instance I_2^2 : L'algorithme *semi-online* attend jusqu'à t_1 (Figure 4.14)

Dans ce cas, l'adversaire ne lance plus aucune tâche. Ainsi, l'ordonnancement *offline* optimale serait d'ordonnancer J_1 dés son arrivée.

$$\sum w_j C_j(I_2^2) = (t_1+p)W \tag{4.57}$$

$$\sum w_j C_j^*(I_2^2) = (t_0+p)W \tag{4.58}$$

Ainsi,

$$\frac{\sum w_j C_j(I_2^2)}{\sum w_j C_j^*(I_2^2)} = \frac{t_1+p}{p} \tag{4.59}$$

b) Instance avec trois dates d'arrivée t_0, t_1 et t_2

Nous n'avons besoin d'étudier que les deux cas qui peuvent être dérivés du cas 2 lorsque l'algorithme *semi-online* décide d'attendre jusqu'à l'instant t_1 alors qu'aucune autre tâche n'arrive à t_1.

Instance $I^2_{2.1}$: L'algorithme *semi-online* ordonnance la tâche J_1 à l'instant t_1 (Figure 4.15)

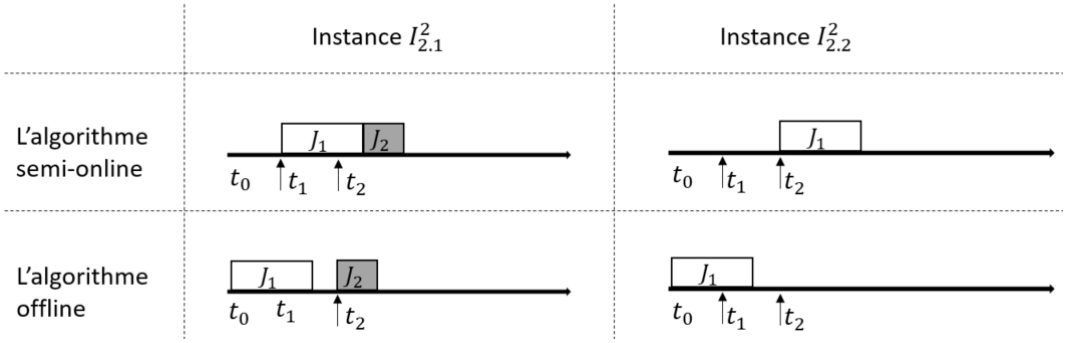

Figure 4.15 – Ordonnancements *semi-online* et *offline* pour les instances $I_2^{2.1}$ et $I_2^{2.2}$

Dans ce cas, l'adversaire lance une tâche J_3 avec $(p_3, w_3) = (p_{min}, W')$ à t_2. De plus, la valeur de t_2 est choisie de manière à ce qu'elle soit supérieure à $t_0 + p$ ($t_2 > t_0 + p$), puisque dans le cas où $t_1 < t_2 \leq t_0 + p$, les performances de l'algorithme *semi-online* peuvent être dégradées davantage si l'adversaire considère $t_2 = t_1$. Ceci revient à étudier le cas de l'instance I_1^2. Ainsi, l'algorithme *offline* optimal ordonnance la tâche J_1 dès son arrivée puis ordonnance la tâche J_2 à t_2.

$$\sum w_j C_j(I^2_{2.1}) = (t_1 + p)W + (t_1 + p + p_{min})W' \tag{4.60}$$

$$\sum w_j C^*_j(I^2_{2.1}) = pW + (t_2 + p_{min})W' \tag{4.61}$$

Ainsi,

$$\frac{\sum w_j C_j(I^2_{2.1})}{\sum w_j C^*_j(I^2_{2.1})} = \frac{(t_1 + p)W + (t_1 + p + p_{min})W'}{pW + (t_2 + p_{min})W'} \tag{4.62}$$

Quand $W' \to \infty$,

$$\frac{\sum w_j C_j(I^2_{2.1})}{\sum w_j C^*_j(I^2_{2.1})} \leq \frac{t_1 + p + p_{min}}{t_2 + p_{min}} \tag{4.63}$$

Instance $I^2_{2.2}$: l'algorithme *semi-online* attend jusqu'à l'instant t_2 (Figure 4.15)

Dans ce cas, l'adversaire ne lance plus aucune tâche.

$$\sum w_j C_j(I^2_{2.2}) = (t_2 + p)W \tag{4.64}$$

$$\sum w_j C^*_j(I^2_{2.2}) = (t_0 + p)W \tag{4.65}$$

Ainsi,

$$\frac{\sum w_j C_j(I^2_{2.2})}{\sum w_j C^*_j(I^2_{2.2})} = \frac{t_2 + p}{p} \tag{4.66}$$

Généralisation du problème $P''(v)$:

À partir des équations (4.56), (4.59), (4.63) et (4.66), nous présentons la généralisation du problème qui maximise le ratio de compétitivité R, noté $P''(v)$ pour $v = \{1, ..., T\}$.

$P''(v)$

$$\begin{aligned}
&Maximiser \quad R \\
&Sous \quad contraintes \\
&\frac{t_s + p + p_{min}}{t_{s+1} + p_{min}} \geq R \quad for \quad s = \{0, ..., v-1\} && (4.67)\\
&\frac{t_v + p}{p} \geq R && (4.68)\\
&p, R \geq 0 && (4.69)
\end{aligned}$$

La contrainte (4.67) représente le cas où la tâche J_j, avec $(p_j, w_j) = (p, W)$, est ordonnancée à l'instant t_s et se termine à l'instant $t_s + p > t_{s+1}$, tandis qu'une autre tâche arrive à t_{s+1}, avec $(p_j, w_j) = (p_{min}, W')$ et $W' >> W$. Puisque cette dernière a un poids dominant, le résultat de la contrainte (4.67) peut être atteint en considérant $W' \to \infty$.
La contrainte (4.68) correspond à la situation où la tâche J_j est ordonnancée à un instant t_v, tandis que plus aucune tâche n'arrive.

4.4.2 Borne inférieure du problème $1|online, r_j, t_k, p_{min}|\sum w_j C_j$

Soit $R''(v)$ la valeur maximale de R pour le problème $P''(v)$, et soit $p''(v)$ la valeur correspondante de p. Nous appelons une telle solution $(p''(v), R''(v))$. De plus, soit $R^{*''} = \max_{1 \leq v \leq T} R''(v)$. Nous définissons deux lemmes avant de passer à l'étude de la borne inférieure sur le ratio de compétitivité du problème considéré. Les démonstrations sont similaires à celles des deux lemmes 7 et 8 de la sous-section 4.2.5.

Lemme 9. [39] Il existe un ordonnancement optimal dans lequel chaque tâche commence soit à la date de fin d'une autre tâche, soit à l'une des dates d'arrivée $t_0, ..., t_T$.

Lemme 10. $R''(v) > 1$ et $p''(v) > t_{s+1} - t_s$, pour $0 \leq s \leq v-1$

Nous présentons ainsi la borne inférieure sur le ratio de compétitivité du problème étudié.

Théorème 10. Le ratio de compétitivité de n'importe quel algorithme *semi-online* pour le problème $1|online, r_j, t_k, p_{min}|\sum w_j C_j$ est au moins égale à $R^{*''}$.

Démonstration. Notons par v l'indice pour lequel $R^{*''} = R''(v)$ et $p^{*''} = p''(v)$. Considérons une instance avec une première tâche qui arrive à $t_0 = 0$, avec $(p_1, w_1) = (p, W)$. Nous prouvons, sans pour autant perdre la généralité des choses, qu'aucun algorithme *semi-online* ne peut garantir un ratio de compétitivité meilleur que $R^{*''}$.
Compte tenu du lemme 9, nous considérons que l'algorithme *semi-online* décide d'ordonnancer J_1 à un instant t_s où $0 \leq s \leq T$.

Premièrement, supposons que $s < v$. Dans ce cas, l'adversaire lance une seconde tâche J_2 à l'instant t_{s+1}, avec $(p_2, w_2) = (p_{min}, W')$ où $W' >> W$. Selon le lemme 10, $p > t_{s+1} - t_s$, ce qui implique que la tâche J_1 n'a pas encore fini quand la tâche J_2 arrive. Ainsi, l'algorithme *semi-online* ordonnance la tâche J_1 à t_s suivie de J_2, tandis que l'algorithme *offline* optimal ordonnance J_2 en premier à l'instant t_{s+1} pour une valeur de W' suffisamment large. Ainsi,

$$\frac{\sum w_j C_j}{\sum w_j C_j^*} = \frac{(t_s + p)W + (t_s + p + p_{min})W'}{(t_{s+1} + p_{min})W' + (t_{s+1} + p_{min} + p)W} \tag{4.70}$$

Quand $W^{'} \to \infty$ et $\epsilon \to 0$, la valeur de $\frac{\sum w_j C_j}{\sum w_j C_j^*}$ s'approche de $\frac{t_s+p+p_{min}}{t_{s+1}+p_{min}} \geq R^{*''}$. Cette inégalité est obtenue de la contrainte (4.67) du problème $P''(v)$.

Alternativement, supposons que $s \geq v$. Dans ce cas, l'adversaire ne lance plus aucune tâche. Ainsi,

$$\frac{\sum w_j C_j}{\sum w_j C_j^*} = \frac{t_v + p}{p} \geq R^{*''} \tag{4.71}$$

La dernière inégalité est obtenue de la contrainte (4.68) du problème $P''(v)$.

Ainsi, quel que soit le choix de l'algorithme *semi-online*, $\frac{\sum w_j C_j}{\sum w_j C_j^*}$ peut être rendue arbitrairement proche d'une valeur supérieure ou égale à $R^{*''}$. Par conséquence, aucun algorithme *semi-online* ne peut avoir un ratio de compétitivité inférieur à $R^{*''}$. □

Puisque les contraintes du problème $P''(v)$ sont différentes des contraintes du problème $P(v)$ définit par Hall et al. [39], il est nécessaire de définir l'expression de $R^{*''}$ avant de conclure sur l'utilité des informations partielles du problème considéré. Ainsi, nous définissons dans ce qui suit la valeur de $R^{*''}$ qui va donner lieu à la présentation d'un nouvel algorithme, nommé MCSWPT (*Modified Critical Shortest Weighted Processing Time).*

4.4.3 L'algorithme MCSWPT

Après quelques résultats préliminaires, nous définissons la solution optimale du problème $P''(v)$, notée $R''(v)$. Le lemme suivant établit une condition sous laquelle il est toujours possible d'améliorer une solution réalisable du problème $P''(v)$. Ceci conduit à des conditions d'optimalité pour $P''(v)$.

Lemme 11. Si (R, p) est une solution faisable du problème $P''(v)$ dans laquelle soit chaque contrainte de l'équation (4.67) est satisfaite en inégalité stricte, soit la contrainte (4.68) est satisfaite en inégalité stricte, alors $R < R''(v)$.

Démonstration. Tout d'abord, considérons que chaque contrainte (4.67) est satisfaite en inégalité stricte pour la solution (R, p). Considérons la nouvelle solution :

$$(R^{'}, p^{'}) = (R + \frac{\epsilon t_v}{p(p-\epsilon)}, p - \epsilon) \tag{4.72}$$

Pour une valeur de $\epsilon > 0$ suffisamment petite. Les contraintes (4.67) restent satisfaites. En outre,

$$\frac{t_v + p - \epsilon}{p - \epsilon} = \frac{t_v + p}{p} + \frac{\epsilon t_v}{p(p-\epsilon)} \geq R + \frac{\epsilon t_v}{p(p-\epsilon)} \tag{4.73}$$

La dernière inégalité est obtenue de la faisabilité de (R, p) pour la contrainte (4.68). Ainsi, la solution $(R^{'}, p^{'})$ est faisable pour le problème $P''(v)$.

Alternativement, supposons que la contrainte (4.68) soit satisfaite en inégalité stricte pour la solution (R, p). Considérons la solution :

$$(R^{'}, p^{'}) = (R + \frac{\epsilon}{t_{s+1} + p_{min}}, p + \epsilon) \tag{4.74}$$

Pour une valeur de ϵ suffisamment petite, les contraintes (4.67) sont toujours satisfaites car :

$$\frac{t_s + p + \epsilon + p_{min}}{t_{s+1} + p_{min}} = \frac{t_s + p + p_{min}}{t_{s+1} + p_{min}} + \frac{\epsilon}{t_{s+1} + p_{min}} \tag{4.75}$$

$$\geq R + \frac{\epsilon}{t_{s+1} + p_{min}} \tag{4.76}$$

Puisque la valeur de ϵ est choisie suffisamment petite, la contrainte (4.68) est toujours satisfaite. Ainsi, la solution $(R^{'}, p^{'})$ est réalisable pour le problème $P''(v)$.

Dans tous les cas, nous avons construit une solution réalisable $(R^{'}, p^{'})$ au problème $P''(v)$ dans laquelle $R^{'} > R$. Ceci implique que (R, p) n'est pas une solution optimale. Par conséquent, $R''(v) > R$. □

Corollaire 2. Dans une solution optimale au problème $P''(v)$, la contrainte (4.68) et au moins une des contraintes (4.67) sont satisfaites à égalité.

Démonstration. Le lemme 11 montre que si les conditions du corollaire ne sont pas satisfaites, alors une meilleure solution existe. □

Le corollaire 2 implique que dans une solution optimale de $P''(v)$, certaines contraintes s de l'inégalité (4.67) et la contrainte (4.68) sont satisfaites à égalité. Ainsi,

$$R''_s(v) = \frac{t_s + p + p_{min}}{t_{s+1} + p_{min}} = \frac{t_v + p}{p} \tag{4.77}$$

$$\iff t_s p + (p)^2 + p_{min} p = t_v t_{s+1} + t_v p_{min} + p t_{s+1} + p p_{min} \tag{4.78}$$

$$\iff (p)^2 - (t_{s+1} - t_s)\, p - t_v t_{s+1} - t_v p_{min} = 0 \tag{4.79}$$

donc,

$$p = \frac{t_{s+1} - t_s + \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1} + 4t_v p_{min}}}{2} \tag{4.80}$$

En conséquence

$$R''_s(v) = 1 + \frac{t_v}{p} \tag{4.81}$$

$$= 1 + \frac{t_v}{\frac{t_{s+1} - t_s + \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1} + 4t_v p_{min}}}{2}} \tag{4.82}$$

$$= 1 + \frac{2t_v \left(t_{s+1} - t_s - \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1} + 4t_v p_{min}} \right)}{-4t_v t_{s+1} - 4t_v p_{min}} \tag{4.83}$$

$$= 1 + \frac{-t_{s+1} + t_s + \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1} + 4t_v p_{min}}}{2(t_{s+1} + p_{min})} \tag{4.84}$$

$$= \frac{t_{s+1} + t_s + 2p_{min} + \sqrt{(t_{s+1} - t_s)^2 + 4t_v (t_{s+1} + p_{min})}}{2(t_{s+1} + p_{min})} \tag{4.85}$$

Ainsi,

$$R''_s(v) = \frac{t_{s+1} + t_s + 2p_{min} + \sqrt{(t_{s+1} - t_s)^2 + 4t_v (t_{s+1} + p_{min})}}{2(t_{s+1} + p_{min})} \quad pour\ s = \{0, ..., v-1\} \tag{4.86}$$

Théorème 11.

$$R^{*''} = \max_{v=1,..,T} \left\{ \min_{s=0,...,v-1} \left\{ \frac{t_{s+1} + t_s + 2p_{min} + \sqrt{(t_{s+1} - t_s)^2 + 4t_v(t_{s+1} + p_{min})}}{2(t_{s+1} + p_{min})} \right\} \right\} \tag{4.87}$$

Démonstration. Il suffit de démontrer que $R''(v) = \min_{s=0,...,v-1} \{R''_s(v)\}$. Considérons deux solutions faisables qui vérifient les conditions du Corollaire 2 $(R''_{s_1}(v), p''_{s_1}(v))$ et $(R''_{s_2}(v), p''_{s_2}(v))$ pour le problème $P''(v)$. Pour ces deux solutions, nous avons $R''_s(v) = \frac{t_s + p''_s(v) + p_{min}}{t_{s+1} + p_{min}}$ et $R''_s(v) = \frac{t_v + p''_s(v)}{p''_s(v)}$ pour $s = \{s_1, s_2\}$.
Supposons que $R''_{s_1}(v) < R''_{s_2}(v)$, ceci implique que $p''_{s_1}(v) > p''_{s_2}(v)$. Pour compléter la preuve, nous montrons que la solution $(R''_{s_2}(v), p''_{s_2}(v))$ est faisable pour le problème $P''(v)$.
La faisabilité de $(R''_{s_1}(v), p''_{s_1}(v))$ implique que,

$$\frac{t_{s_1} + p''_{s_2}(v) + p_{min}}{t_{s_1+1} + p_{min}} = \frac{R''_{s_1}(v)(t_{s_1+1} + p_{min}) - p''_{s_1}(v) - p_{min} + p''_{s_2}(v) + p_{min}}{t_{s_1+1} + p_{min}} \tag{4.88}$$

$$= R''_{s_1}(v) - \frac{p''_{s_1}(v) - p''_{s_2}(v)}{t_{s_1+1} + p_{min}} \tag{4.89}$$

$$< R''_{s_1}(v) < R''_{s_2}(v) \tag{4.90}$$

L'équation (4.90) est dû au fait que $p''_{s_2}(v) < p''_{s_1}(v)$.
Ainsi, la solution $(R''_{s_2}(v), p''_{s_2}(v))$ ne satisfait pas la contrainte (4.67) pour $s = s_1$, ce qui est une contradiction. □

Nous comparons la valeur obtenue de $R^{*''}$ avec R^* afin de vérifier l'utilité des informations partielles considérées.

Comparaison entre R^* et $R^{*''}$:

Afin de comparer R^* et $R^{*''}$, il suffit de comparer $R''_s(v)$ à $R_s(v)$, avec $R_s(v)$ la valeur obtenue par Hall et al. [39] pour le problème $P(v)$:

$$R_s(v) = \frac{t_{s+1} + t_s + \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1}}}{2t_{s+1}} \tag{4.91}$$

Supposons que $R''_s(v) \geq R_s(v)$ et trouvons une contradiction. Notons par $f_1 = \sqrt{(t_{s+1} - t_s)^2 + 4t_v(t_{s+1} + p_{min})}$ et par $f_2 = \sqrt{(t_{s+1} - t_s)^2 + 4t_v t_{s+1}}$.

$$R''_s(v) \geq R_s(v) \tag{4.92}$$
$$\iff \frac{t_{s+1}+t_s+2p_{min}+f_1}{2(t_{s+1}+p_{min})} - \frac{t_{s+1}+t_s+f_2}{2t_{s+1}} \geq 0 \tag{4.93}$$
$$\iff 2t_{s+1}(t_{s+1}+t_s) + 4t_{s+1}p_{min} + 2t_{s+1}f_1$$
$$- 2t_{s+1}(t_{s+1}+t_s) - 2p_{min}(t_{s+1}+t_s) - 2(t_{s+1}+p_{min})f_2 \geq 0 \tag{4.94}$$
$$\iff (p_{min}(t_{s+1}-t_s)+t_{s+1}f_1)^2 \geq ((t_{s+1}+p_{min})f_2)^2 \geq 0 \tag{4.95}$$
$$\iff p_{min}^2(t_{s+1}-t_s)^2 + t_{s+1}^2\left((t_{s+1}-t_s)^2 + 4t_v(t_{s+1}+p_{min})\right)$$
$$+ 2t_{s+1}p_{min}(t_{s+1}-t_s)f_1 - (t_{s+1}+p_{min})^2\left((t_{s+1}-t_s)^2 + 4t_vt_{s+1}\right) \geq 0 \tag{4.96}$$

$$\iff (t_{s+1}-t_s)f_1 \geq (t_{s+1}-t_s)^2 + 2t_v(t_{s+1}+p_{min}) \tag{4.97}$$
$$\iff (t_{s+1}-t_s)^2\left((t_{s+1}-t_s)^2 + 4t_v(t_{s+1}+p_{min})\right)$$
$$\geq \left((t_{s+1}-t_s)^2 + 2t_v(t_{s+1}+p_{min})\right)^2 \tag{4.98}$$
$$\iff 4t_v^2(t_{s+1}+p_{min})^2 \leq 0 \tag{4.99}$$

Cependant, par définition $t_v > 0$, $t_{s+1} > 0$ et $p_{min} > 0$ ce qui implique que l'inégalité (4.99) est une contradiction. Ainsi, $R''_s(v) < R_s(v)$. De plus, selon Hall et al. [39] la valeur de R^* vérifie $R^* < 2$. Ainsi,

$$R^{*''} < R^* < 2 \tag{4.100}$$

Nous pouvons conclure que les informations partielles étudiées sont utiles puisque la borne inférieure sur le ratio de compétitivité du le problème $1|online, r_j, t_k, p_{min}|\sum w_jC_j$ est meilleure que celle du problème $1|online, r_j,\ t_k|\sum w_jC_j$. Nous présentons alors un nouvel algorithme *semi-online*, nommé MCSWPT (*Modified Critical Shortest Weighted Processing Time)*, qui permet d'utiliser les informations partielles t_k et p_{min}.

L'algorithme MCSWPT : Tout d'abord, la valeur de $R^{*''}$ est calculée selon le Théorème 11. Ensuite, à un instant t, lorsque la machine est inactive et que certaines tâches sont disponibles, choisir la tâche J_j, avec la plus grande valeur de $\frac{w_j}{p_j}$, parmi toutes les tâches disponibles. En cas d'égalité, choisir la tâche arrivée en premier. De plus, notons par t_k la prochaine date d'arrivée. Ensuite, si

$$t + p \leq R^{*''}t_k + (R^{*''} - 1)p_{min} \tag{4.101}$$

Ordonnancer J_j à l'instant t. Sinon, attendre la date d'arrivée suivante et répéter la procédure.

L'expression $R^{*''} t_k + (R^{*''} - 1)p_{min}$ représente ici la date de fin critique de la tâche J_j, c'est-à-dire la date qu'il ne faut pas dépasser si la tâche J_j est ordonnancée. Si, en ordonnançant J_j, la date de fin critique est dépassée, alors MCSWPT attend la prochaine arrivée. Toutefois, si J_j peut être accomplie avant la date de fin critique, elle est alors ordonnancée immédiatement.

4.4.4 Étude expérimentale de MCSWPT

Nous présentons dans cette section une étude numérique comparative de trois algorithmes : DSWPT, CSWPT et MCSWPT. Le premier algorithme DSWPT (*Delayed Shortest Weighted Processing Time*) a été développé par Anderson et Potts [6] pour le problème *online* $1|online, r_j|\sum w_j C_j$. Ils ont prouvé que l'algorithme était 2-compétitif et que cette borne est la meilleure possible. Nous rappelons le déroulement de l'algorithme [6] :

DSWPT : À un instant t, lorsque la machine est inactive et que des tâches sont disponibles, choisir la tâche J_j, ayant la plus grande valeur de $\frac{w_j}{p_j}$. En cas d'égalité, choisir celle ayant le plus petit temps de traitement, sinon celle arrivée en premier. Si

$$p_j \leq t \tag{4.102}$$

Alors ordonnancer J_j. Sinon, attendre que la condition soit vérifiée ou qu'une nouvelle tâche arrive.

En ce qui concerne la génération des instances. Nous utilisons le schéma proposé par Hall et al. [38] avec des temps de traitement générés à partir d'une distribution uniforme $U[p_{min}, 100]$. Nous présentons les résultats pour $p_{min} = \{30, 50\}$. Concernant les dates d'arrivée, nous les générons comme suit [38] :

$$r_1 = 0 \text{ et } r_j = r_{j-1} + X_j,\ j = 2, ..., n$$

Où X_j est une variable aléatoire générée à partir d'une distribution uniforme $U[0, 100]$. Pour chaque taille d'instance $n \in \{10, 30, 50, 100\}$, 1000 instances ont été générées. Pour chaque instance, les trois algorithmes ont été testés et les fonctions objectives obtenues ont été enregistrées.

Nous avons choisis trois indicateurs de performance. Le premier indicateur est le *RPD moyen* (*Relative Percent Difference*), il défini en moyenne le pourcentage du gap entre la solution de l'algorithme et la meilleure solution obtenue parmi les algorithmes

considérés. Les deux autres indicateurs sont : le *nombre de fois meilleur* et le *nombre de fois en avance*, qui indiquent en moyenne combien de fois la solution de l'algorithme a été la meilleure parmi les solutions des algorithmes testés et le nombre de fois en avance sur les autres solutions, respectivement.
Les figures 4.16, 4.17 et 4.18 illustrent les performances des algorithmes pour $p_{min} = 30$ selon les indicateurs *RPD moyen*, *nombre de fois meilleur* et *nombre de fois en avance*, respectivement. Les figures 4.19, 4.20 et 4.21 présentent les performances des algorithmes pour $p_{min} = 50$. Nous pouvons remarquer que l'algorithme proposé MCSWPT garantit la meilleure performance en moyenne et il est en avance sur les autres algorithmes CSWPT et DSWPT. Nous pouvons également constater qu'en augmentant la valeur de p_{min} en considérant $p_{min} = 50$, l'avantage de MCSWPT sur les autres algorithmes devient de plus en plus pertinent, que ça soit pour des instances de petite taille ou des instances de plus grande taille. De plus, pour $p_{min} = 50$ les performances des algorithmes existants DSWPT et CSWPT deviennent équivalentes.

4.5 Conclusion

Dans ce chapitre, nous avons étudié dans un premier temps le problème $1|online, known\ r_j|\sum C_j$ où les dates d'arrivée des tâches sont connues à l'avance, tandis que les temps de traitement des tâches demeurent inconnus. Nous avons construit deux algorithmes *semi-online*, VD-SPT et CSPTn, qui utilisent l'information partielle dans la prise de décision. L'étude expérimentale nous a permis d'évaluer ces algorithmes selon différents caractéristiques des instances. En général, VD-SPT est très performant face à des instances de petites tailles, alors que CSPTn montrent un avantage pour les instances avec une grande variabilité en termes de temps de traitement. Finalement, nous présentons une borne inférieure générale sur le ratio de compétitivité du problème $1|online, known\ r_j\ |\sum C_j$ afin de prouver l'utilité de l'information partielle. Dans un second temps, nous avons étudié le problème $1|online, known\ r_j|\sum w_jC_j$. Nous avons démontré que l'information sur les dates d'arrivée des tâches exactes n'apporte aucune amélioration quand les tâches ont un poids différent. Ensuite, nous avons étudié le problème $1|online, r_j, t_k, p_{min}|\sum w_jC_j$, où une combinaison d'information sur les dates d'arrivée potentielles et une borne inférieure sur le plus petit temps de traitement, permet d'améliorer la prise de décision. Pour ce problème, nous avons développé un nouvel algorithme *semi-online*, nommé MCSWPT, dont la performance a été évaluée par une étude expérimentale comparative.

Les travaux présentés dans ce chapitre ont fait l'objet des publications suivantes :

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. New Heuristic For Single Machine

Semi-online Total Completion Time Minimization, In Proceedings of IFAC World Congress, volume 1, 2020. [63]

— Hajar Nouinou, Taha Arbaoui, Alice Yalaoui. Semi-online Scheduling for Minimizing the Total completion Time with Known Release Datee. 17th IFAC Symposium on Information Control Problems in Manufacturing, Hungary, 2021. [66]

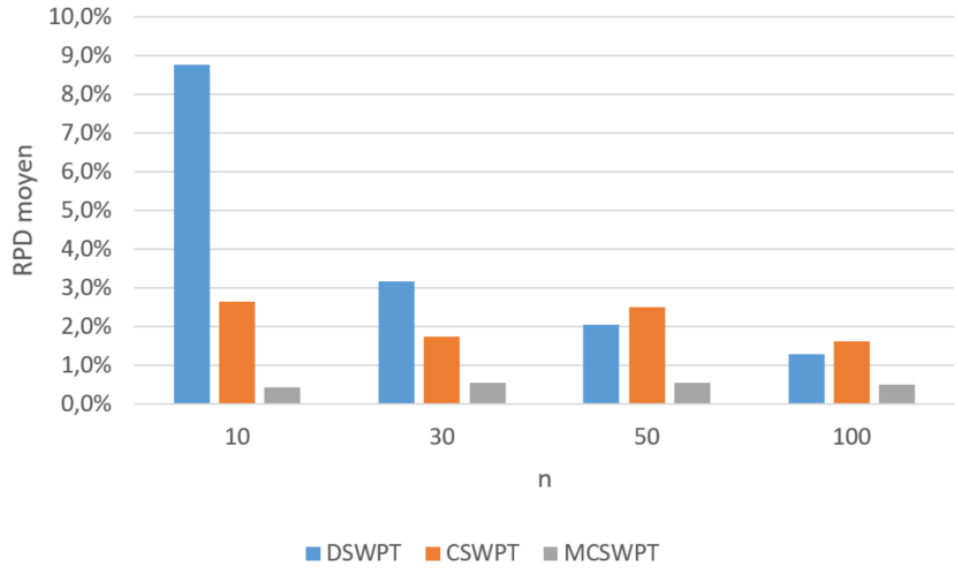

Figure 4.16 – RPD moyen pour DSWPT, CSWPT et MCSWPT ($p_{min} = 30$)

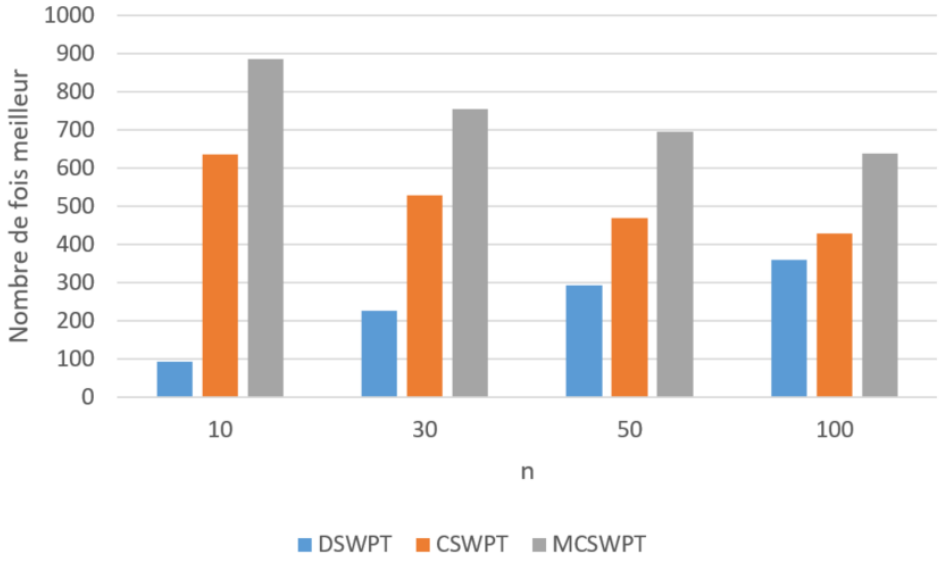

Figure 4.17 – Nombre de fois où un algorithme est meilleur pour DSWPT, CSWPT et MCSWPT ($p_{min} = 30$)

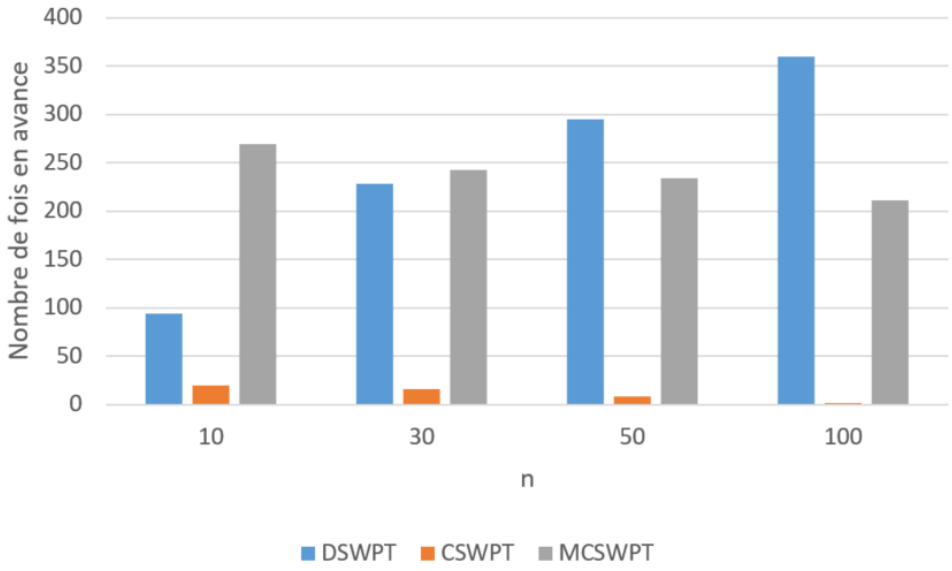

Figure 4.18 – Nombre de fois où un algorithme est en avance pour DSWPT, CSWPT et MCSWPT ($p_{min} = 30$)

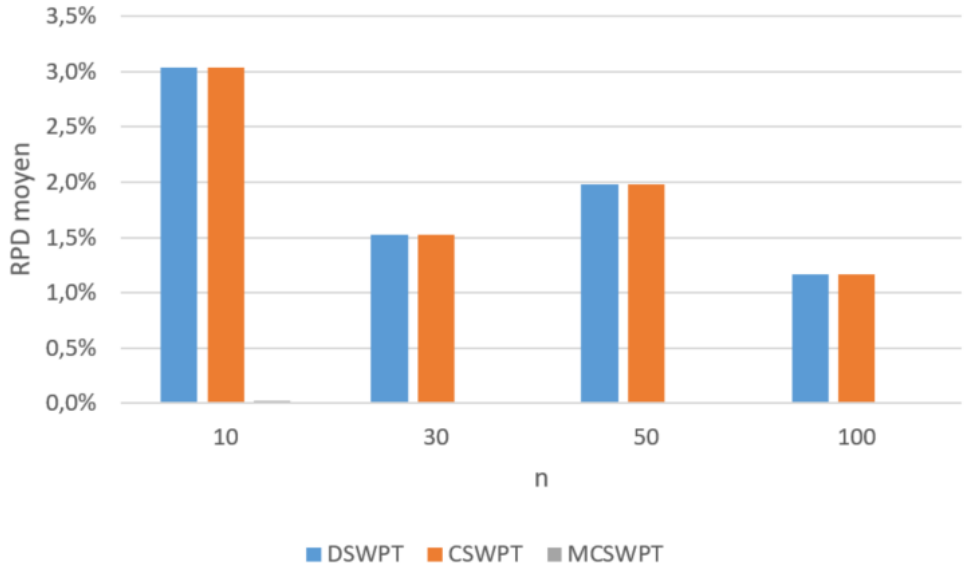

Figure 4.19 – RPD moyen pour DSWPT, CSWPT et MCSWPT ($p_{min} = 50$)

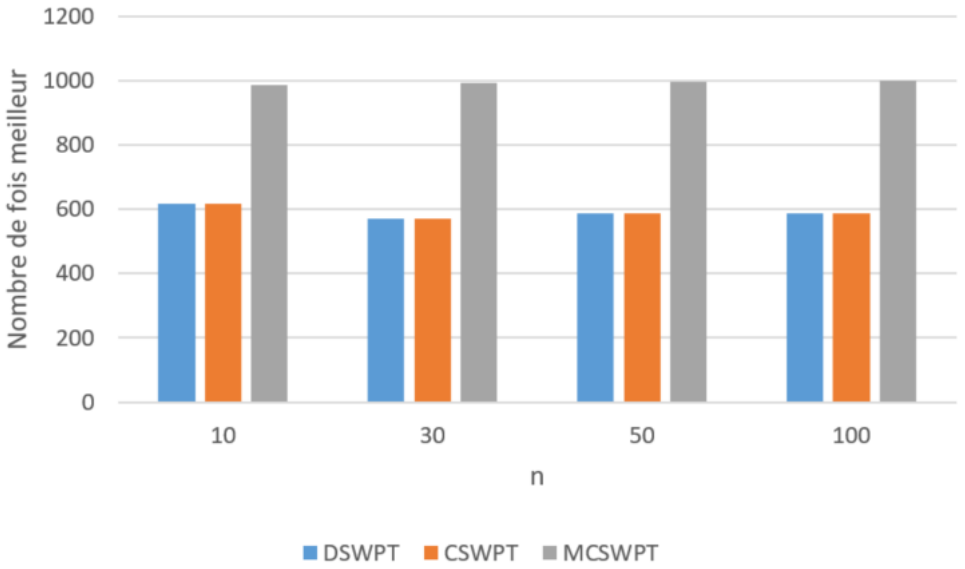

Figure 4.20 – Nombre de fois où un algorithme est meilleur pour DSWPT, CSWPT et MCSWPT ($p_{min} = 50$)

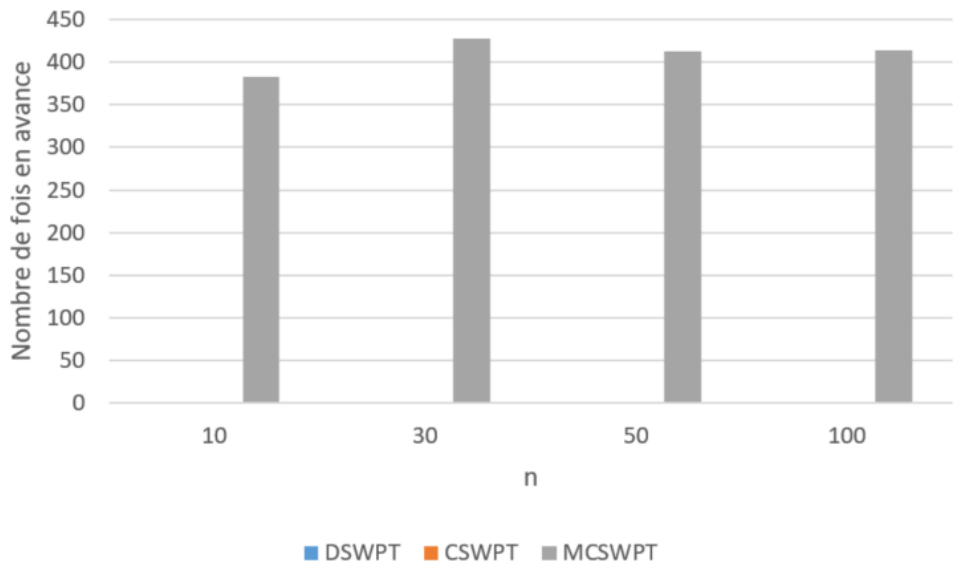

Figure 4.21 – Nombre de fois où un algorithme est en avance pour DSWPT, CSWPT et MCSWPT ($p_{min} = 50$)

Conclusion générale

L'objectif principal de cette thèse a été d'étudier la valeur des informations partielles dans un contexte d'ordonnancement *semi-online* sur machine unitaire. Nous avons utilisé la méthode d'analyse de compétitivité ainsi que la méthode d'analyse expérimentale pour mesurer la performance des algorithmes *semi-online*. En déterminant des bornes inférieures sur le ratio de compétitivité, nous avons abordé la question : combien perdons-nous face à l'incertitude ? ou à quel point cela vaut-il la peine de connaître le futur ? Nous avons fait le choix de nous intéresser aux problèmes de minimisation de la somme des dates de fin des tâches et la somme des dates de fin pondérées sur machine unitaire. Notre étude a permis l'identification des informations partielles utiles de celles non-utiles. Pour les problèmes d'ordonnancement *semi-online* où l'information partielle permet d'améliorer la prise de décision, nous avons proposé des algorithmes *semi-online* permettant l'utilisation de ces informations. Ensuite, nous avons analysé les performances de ces algorithmes en utilisant l'analyse de compétitivité et/ou l'analyse expérimentale.

Dans ce travail, les problèmes d'ordonnancement *semi-online* dont les algorithmes proposés sont étudiés à l'aide d'une analyse de compétitivité, correspondent plus à des cas généraux qu'à des cas particuliers des problèmes d'ordonnancement *online* existants. En effet, que ce soit dans la prise de décision de l'algorithme proposé ou dans son ratio de compétitivité, en éliminant l'information partielle nous retrouvons le modèle *online*.

Un état de l'art sur les problèmes d'ordonnancement *semi-online* existants nous a permis d'apercevoir le grand manque des travaux considérants la minimisation de la somme des dates de fin des tâches (ou la somme des dates de fin pondérées). Nous nous sommes alors intéressés à étudier plusieurs problèmes *semi-online* sur un environnement à machine unitaire. En effet, les résultats obtenus en considérant cet environnement pourraient apporter une meilleure vision sur comment aborder des environnements plus complexes.

La première partie de ce travail consistait à étudier plusieurs variantes *semi-online* où nous avons considéré des informations partielles sur les temps de traitement des tâches. Pour chaque problème *semi-online* considéré, nous avons déterminé la borne inférieure sur le ratio de compétitivité. Ainsi, une classification des informations utiles et des informations non-utiles a été présentée.

Dans le chapitre 3, nous avons étudié le problème où les tâches arrivent en ordre décroissant des temps de traitement et une borne inférieure sur le temps de traitement de la dernière tâche est connue à l'avance. Nous avons présenté un nouvel algorithme *semi-online* pour le problème, nommé ϕD-SPT, avec ϕ un paramètre du problème exprimé en fonction des informations partielles considérées. Une étude de compétitivité de cet algorithme a permis de prouver que le ratio de compétitivité est de $1 + \frac{1}{\phi}$, avec $\phi > 1$. La différence entre la borne inférieure et la borne supérieure sur le ratio de compétitivité est très légère et est justifiée par l'utilisation d'une borne inférieure sur la solution *offline* optimale dans notre étude de compétitivité. Finalement, une étude numérique montre que l'algorithme garantit une meilleure performance que les autres algorithmes *online* et *semi-online* existants.

Dans le chapitre 4, nous avons étudié dans un premier temps le problème $1|online, known\ r_j|\sum C_j$ où les dates d'arrivée des tâches sont connues à l'avance, tandis que les temps de traitement demeurent inconnus. Pour ce problème, nous avons déterminé la borne inférieure sur le ratio de compétitivité qui prouve l'utilité de cette information. En ce qui concerne l'intégration de cette information dans la prise de décision, nous avons présenté deux algorithmes *semi-online*, VD-SPT et CSPTn. Le premier a montré une très bonne performance sur des instances de petites tailles, tandis que le deuxième montre un avantage pour des instances de plus grandes tailles présentant une grande variabilité dans les temps de traitement. Dans un second temps, nous avons considéré le problème $1|online, known\ r_j|\sum w_jC_j$ où chaque tâche J_j a un poids w_j. Nous avons ainsi prouvé que la connaissance des dates d'arrivée des tâches exactes n'apporte aucun avantage. Ensuite, nous avons étudié le problème $1|online, r_j, t_k, p_{min}|\sum w_jC_j$ avec une combinaison d'information sur les dates d'arrivée potentielles et une borne inférieure sur le plus petit temps de traitement. Nous avons constaté que cette combinaison d'information est utile et nous avons développé un algorithme *semi-online*, nommé MCSWPT, intégrant la combinaison d'information dans sa prise de décision. Pour l'ensemble des algorithmes *semi-online* développés dans ce chapitre, nous avons évalué leurs performances par une étude expérimentale.

Les différents travaux présentés dans cette thèse laissent envisager plusieurs possibilités de recherches futures. Dans le court terme, nous envisageons d'étudier la compétitivité de l'algorithme MCSWPT présenté dans le chapitre 4. En effet, le fait d'ajouter les poids des tâches avec la considération de la minimisation de la somme des dates de fin pondérées permet de faciliter l'identification de la pire instance du problème. Plus précisément, au lieu d'appliquer des modifications sur les temps de traitement des tâches afin d'obtenir la pire instance, nous pouvons modifier les poids, et cette modification n'influence que l'ordre des tâches dans l'instance. En ce qui concerne les autres algorithmes proposés dans cette partie, une analyse de compétitivité semble dramatiquement difficile du fait que ces algorithmes sont plus complexes. Ainsi, la caractérisation de l'instance pire-cas qui permet de calculer le ratio de compétitivité devient une tâche très difficile voire même impossible.

Nous pouvons envisager de rapprocher le modèle *semi-online* encore plus du modèle *offline* en étudiant la combinaison de l'information sur les dates d'arrivée exactes et la borne inférieure sur le plus petit temps de traitement dans l'instance.

Une autre possibilité est de considérer des environnements machine plus complexes. En effet, dans l'ordonnancement *offline*, les performances des algorithmes se détériorent généralement lorsque la complexité de l'environnement machine augmente. En revanche, dans l'ordonnancement *online* ou *semi-online* et en raison de l'indisponibilité d'information sur les tâches futures, un environnement machine plus complexe peut donner plus d'opportunités pour corriger des erreurs commises dans le passé. Une preuve solide de cette hypothèse est présentée par Leonardi et Raz [57], qui ont prouvé que pour minimiser le *flow time* total, la meilleure borne de performance possible pour les instances avec un nombre fixe de tâches est décroissante en fonction du nombre de machines.

Pour le problème de minimisation de la somme des dates de fin pondérées des tâches sur machine unitaire, nous pouvons également envisager de combiner encore plus d'informations en ajoutant des informations sur les poids des tâches par exemple, jusqu'à ce qu'une limite puisse être identifiée entre le modèle *semi-online* et le modèle *offline*, c'est-à-dire identifier la combinaison d'information adéquate qui va permettre de développer un algorithme *semi-online* 1-compétitif (voir Figure 4.22).

Finalement, il serait intéressant d'évaluer les performances des algorithmes proposés avec les autres méthodes comme l'augmentation de ressources, la compétitivité locale amortie, etc.

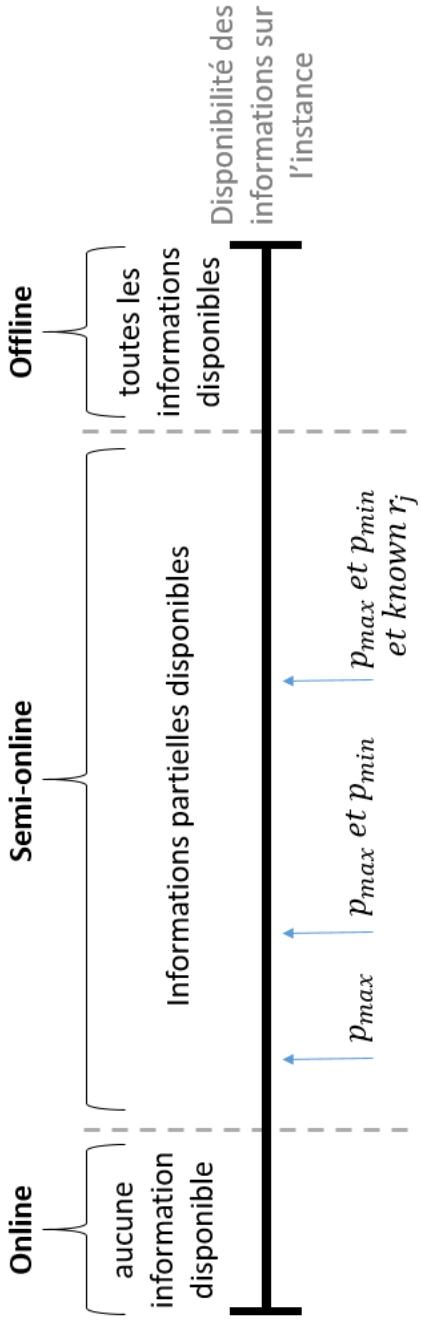

Figure 4.22 – Limites entre le *online*, le *semi-online* et le *offline* en fonction des informations connues

Bibliographie

[1] Susanne Albers. Better bounds for online scheduling. *SIAM Journal on Computing*, 29(2) :459–473, 1999.

[2] Susanne Albers. Online algorithms : a survey. *Mathematical Programming*, 97(1) :3–26, 2003.

[3] Susanne Albers. Recent advances for a classical scheduling problem. In *International Colloquium on Automata, Languages, and Programming*, pages 4–14. Springer, 2013.

[4] Susanne Albers and Matthias Hellwig. Semi-online scheduling revisited. *Theoretical Computer Science*, 443 :1–9, 2012.

[5] Susanne Albers and Bianca Schröder. An experimental study of online scheduling algorithms. In *International Workshop on Algorithm Engineering*, pages 11–22. Springer, 2000.

[6] Edward J Anderson and Chris N Potts. Online scheduling of a single machine to minimize total weighted completion time. *Mathematics of Operations Research*, 29(3) :686–697, 2004.

[7] Enrico Angelelli, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line scheduling on three processors with known sum of the tasks. *Journal of Scheduling*, 10(4) :263–269, 2007.

[8] Elnaz Asadollahi-Yazdi, Paulin Couzon, Nhan Quy Nguyen, Yassine Ouazene, and Farouk Yalaoui. Industry 4.0 : Revolution or evolution ? *American Journal of Operations Research*, 10(06) :241, 2020.

[9] James Aspnes, Yossi Azar, Amos Fiat, Serge Plotkin, and Orli Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 623–631, 1993.

[10] Yossi Azar and Oded Regev. On-line bin-stretching. In *In Proceedings of the 2nd RANDOM*. Citeseer, 1997.

[11] Yossi Azar and Oded Regev. On-line bin-stretching. *Theoretical Computer Science*, 268(1) :17–41, 2001.

[12] Nikhil Bansal, Tracy Kimbrel, and Kirk Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM (JACM)*, 54(1) :1–39, 2007.

[13] Hocine Belouadah, Marc E Posner, and Chris N Potts. Scheduling with release dates on a single machine to minimize total weighted completion time. *Discrete applied mathematics*, 36(3) :213–231, 1992.

[14] Shai Ben-David, Allan Borodin, Richard Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1) :2–14, 1994.

[15] Vittorio Bilò, Michele Flammini, and Roberto Giovannelli. Experimental analysis of online algorithms for the bicriteria scheduling problem. *Journal of Parallel and Distributed Computing*, 64(9) :1086–1100, 2004.

[16] Jacek Blazewicz, Klaus Ecker, Erwin Pesch, Günter Schmidt, and J Weglarz. *Handbook on scheduling*. Springer, 2019.

[17] Qian Cao, TCE Cheng, Guohua Wan, and Yi Li. Several semi-online scheduling problems on two identical machines with combined information. *Theoretical Computer Science*, 457 :35–44, 2012.

[18] Qian Cao, Zhaohui Liu, and TC Edwin Cheng. Semi-online scheduling with known partial information about job sizes on two identical machines. *Theoretical computer science*, 412(29) :3731–3737, 2011.

[19] Qian Cao and Guohua Wan. Semi-online scheduling with combined information on two identical machines in parallel. *Journal of Combinatorial Optimization*, 31(2) :686–695, 2016.

[20] Xin Chen, Zhenzhen Xu, György Dósa, Xin Han, and He Jiang. Semi-online hierarchical scheduling problems with buffer or rearrangements. *Information Processing Letters*, 113(4) :127–131, 2013.

[21] TC Edwin Cheng, Hans Kellerer, and Vladimir Kotov. Semi-on-line multiprocessor scheduling with given total processing time. *Theoretical computer science*, 337(1-3) :134–146, 2005.

[22] TC Edwin Cheng, Hans Kellerer, and Vladimir Kotov. Algorithms better than lpt for semi-online scheduling with decreasing processing times. *Operations Research Letters*, 40(5) :349–352, 2012.

[23] Fabricia Silva da Rosa, Sandra Rolim Ensslin, Leonardo Ensslin, and Rogério Joao Lunkes. Environmental disclosure management : a constructivist case. *Management Decision*, 2012.

[24] Stéphane Dauzère-Pérès and Jean-Bernard Lasserre. On the importance of sequencing decisions in production planning and scheduling. *International transactions in operational research*, 9(6) :779–793, 2002.

[25] Sandra Rolim Ensslin, Leonardo Ensslin, Rogério Tadeu de Oliveira Lacerda, Victor Hugo Aurélio de Souza, et al. Disclosure of the state of the art of performance evaluation applied to project management. *American Journal of Industrial and Business Management*, 4(11) :677, 2014.

[26] Leah Epstein. A survey on makespan minimization in semi-online environments. *Journal of Scheduling*, 21(3) :269–284, 2018.

[27] Leah Epstein and Lene M Favrholdt. Optimal non-preemptive semi-online scheduling on two related machines. *Journal of Algorithms*, 57(1) :49–73, 2005.

[28] Leah Epstein and Rob van Stee. Lower bounds for on-line single-machine scheduling. *Theoretical Computer Science*, 299(1-3) :439–450, 2003.

[29] Amos Fiat and Gerhard Woeginger. *Online algorithms : The state of the art*. Springer, 1998.

[30] Amos Fiat and Gerhard J Woeginger. Competitive analysis of algorithms. In *Online Algorithms*, pages 1–12. Springer, 1998.

[31] Marshall L Fisher. A dual algorithm for the one-machine scheduling problem. *Mathematical programming*, 11(1) :229–251, 1976.

[32] Rudolf Fleischer and Michaela Wahl. On-line scheduling revisited. *Journal of Scheduling*, 3(6) :343–353, 2000.

[33] Michel X Goemans. Improved approximation algorthims for scheduling with release dates. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 591–598, 1997.

[34] Todd Gormley, Nicholas Reingold, Eric Torng, and Jeffery Westbrook. Generating adversaries for request-answer games. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 564–565, 2000.

[35] Ronald L Graham. Bounds for certain multiprocessing anomalies. *Bell System Technical Journal*, 45(9) :1563–1581, 1966.

[36] Ronald L. Graham. Bounds on multiprocessing timing anomalies. *SIAM journal on Applied Mathematics*, 17(2) :416–429, 1969.

[37] Ronald L Graham, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

[38] Nicholas G Hall and Marc E Posner. Generating experimental data for computational testing with machine scheduling applications. *Operations Research*, 49(6) :854–865, 2001.

[39] Nicholas G Hall, Marc E Posner, and Chris N Potts. Online scheduling with known arrival times. *Mathematics of Operations Research*, 34(1) :92–102, 2009.

[40] Yong He and György Dósa. Extension of algorithm list scheduling for a semi-online scheduling problem. *Central European Journal of Operations Research*, 15(1) :97–104, 2007.

[41] Yong He and Guochuan Zhang. Semi on-line scheduling on two identical machines. *Computing*, 62(3) :179–187, 1999.

[42] Johannes A Hoogeveen and Arjen PA Vestjens. Optimal on-line algorithms for single-machine scheduling. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 404–414. Springer, 1996.

[43] Sungjin Im, Benjamin Moseley, and Kirk Pruhs. A tutorial on amortized local competitiveness in online scheduling. *ACM SIGACT News*, 42(2) :83–97, 2011.

[44] Srikanth K Iyer and Barkha Saxena. Improved genetic algorithm for the permutation flowshop scheduling problem. *Computers & Operations Research*, 31(4) :593–606, 2004.

[45] David S Johnson. Near-optimal bin packing algorithms [ph. d. thesis]. *Boston : Massachusetts Institute of Technology*, 1973.

[46] David S Johnson. Fast algorithms for bin packing. *Journal of Computer and System Sciences*, 8(3) :272–314, 1974.

[47] David S Johnson and Michael R Garey. *Computers and intractability : A guide to the theory of NP-completeness*. WH Freeman, 1979.

[48] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[49] Hans Kellerer, Vladimir Kotov, Maria Grazia Speranza, and Zsolt Tuza. Semi on-line algorithms for the partition problem. *Operations Research Letters*, 21(5) :235–242, 1997.

[50] Jacques Labetoulle, Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Preemptive scheduling of uniform machines subject to release dates. In *Progress in combinatorial optimization*, pages 245–261. Elsevier, 1984.

[51] BJ Lageweg, Jan Karel Lenstra, and AHG Rinnooy Kan. Minimizing maximum lateness on one machine : computational experience and some applications. *Statistica Neerlandica*, 30(1) :25–41, 1976.

[52] Eugene L Lawler. Sequencing to minimize the weighted number of tardy jobs. *Rev. Française Automat. Informat. Recherche opérationnelle*, 10 :27–33, 1976.

[53] Eugene L Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. In *Annals of discrete Mathematics*, volume 1, pages 331–342. Elsevier, 1977.

[54] Eugene L Lawler and J Michael Moore. A functional equation and its application to resource allocation and sequencing problems. *Management Science*, 16(1) :77–84, 1969.

[55] Kangbok Lee, Joseph Y-T Leung, and Michael L Pinedo. Makespan minimization in online scheduling with machine eligibility. *Annals of Operations Research*, 204(1) :189–222, 2013.

[56] Jan K Lenstra, Alexander HG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[57] Stefano Leonardi and Danny Raz. Approximating total flow time on parallel machines. *Journal of Computer and System Sciences*, 73(6) :875–891, 2007.

[58] Ming Liu, Feifeng Zheng, Shijin Wang, and Jiazhen Huo. Optimal algorithms for online single machine scheduling with deteriorating jobs. *Theoretical Computer Science*, 445 :75–81, 2012.

[59] Xiwen Lu, René A Sitters, and Leen Stougie. A class of on-line scheduling algorithms to minimize total completion time. *Operations Research Letters*, 31(3) :232–236, 2003.

[60] Ran Ma, Jiping Tao, and Jinjiang Yuan. Online scheduling with linear deteriorating jobs to minimize the total weighted completion time. *Applied Mathematics and Computation*, 273 :570–583, 2016.

[61] Nicole Megow. Coping with incomplete information in scheduling—stochastic and online models. In *Operations Research Proceedings 2007*, pages 17–22. Springer, 2008.

[62] J Michael Moore. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management science*, 15(1) :102–109, 1968.

[63] Hajar Nouinou, Taha Arbaoui, and Alice Yalaoui. New heuristic for single machine semi-online total completion time minimization. *IFAC-PapersOnLine*, 53(2) :10676–10681, 2020.

[64] Hajar Nouinou, Taha Arbaoui, and Alice Yalaoui. Semi-online scheduling for minimizing the total completion time : Information value. In *International Conference on Optimization and Learning*, page Spain, 2020.

[65] Hajar Nouinou, Taha Arbaoui, and Alice Yalaoui. Minimising the total completion time for a class of semi-online single machine scheduling problems. *Theoretical Computer Science*, 2021.

[66] Hajar Nouinou, Taha Arbaoui, and Alice Yalaoui. Semi-online scheduling for minimizing the total completion time with known release datee. *17th IFAC Symposium on Information Control Problems in Manufacturing*, 2021.

[67] Hervé Panetto, Benoit Iung, Dmitry Ivanov, Georg Weichhart, and Xiaofan Wang. Challenges for the cyber-physical manufacturing enterprises of the future. *Annual Reviews in Control*, 47 :200–213, 2019.

[68] Cynthia Phillips, Clifford Stein, and Joel Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82(1-2) :199–223, 1998.

[69] Cynthia A Phillips, Cliff Stein, Eric Torng, and Joel Wein. Optimal time-critical scheduling via resource augmentation. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 140–149, 1997.

[70] Michael Pinedo. *Scheduling*, volume 29. Springer, 2012.

[71] Marc E Posner. Minimizing weighted completion times with deadlines. *Operations Research*, 33(3) :562–574, 1985.

[72] Chris N Potts and LN Van Wassenhove. Algorithms for scheduling a single machine to minimize the weighted number of late jobs. *Management Science*, 34(7) :843–858, 1988.

[73] Prabhakar Raghavan and Marc Snir. Memory versus randomization in on-line algorithms. In *International Colloquium on Automata, Languages, and Programming*, pages 687–703. Springer, 1989.

[74] AHG Rinnooy Kan, BJ Lageweg, and Jan Karel Lenstra. Minimizing total costs in one-machine scheduling. *Operations research*, 23(5) :908–927, 1975.

[75] Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Mathematics of Operations Research*, 34(2) :481–498, 2009.

[76] Steve Seiden, Jiri Sgall, and Gerhard Woeginger. Semi-online scheduling with decreasing job sizes. *Operations Research Letters*, 27(5) :215–221, 2000.

[77] Marc Sevaux and Stéphane Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European journal of operational research*, 151(2) :296–306, 2003.

[78] CAI Sheng-yi. Semi online scheduling on three identical machines [j]. *Journal of Wenzhou Teachers College*, 3, 2002.

[79] Linus Shrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16 :687–690, 1968.

[80] René Sitters. Competitive analysis of preemptive single-machine scheduling. *Operations Research Letters*, 38(6) :585–588, 2010.

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

[82] Francielly Hedler Staudt, Gülgün Alpan, Maria Di Mascolo, and Carlos M Taboada Rodriguez. Warehouse performance measurement : a literature review. *International Journal of Production Research*, 53(18) :5524–5544, 2015.

[83] Zhiyi Tan and Yong He. Semi-on-line problems on two identical machines with combined partial information. *Operations Research Letters*, 30(6) :408–414, 2002.

[84] Zhiyi Tan and Shaohua Yu. Online scheduling with reassignment. *Operations Research Letters*, 36(2) :250–254, 2008.

[85] Zhiyi Tan and An Zhang. Online and semi-online scheduling. *Handbook of combinatorial optimization*, pages 2191–2252, 2013.

[86] Jiping Tao. A better online algorithm for the parallel machine scheduling to minimize the total weighted completion time, 2014.

[87] Jiping Tao, Zhijun Chao, and Yugeng Xi. A novel way to analyze competitive performance of online algorithms. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, volume 1, 2009.

[88] Jiping Tao, Zhijun Chao, and Yugeng Xi. Comments on "competitive analysis of a better on-line algorithm to minimize total completion time on a single-machine". *Journal of Global Optimization*, 49(2) :359–362, 2011.

[89] Jiping Tao, Zhijun Chao, Yugeng Xi, and Ye Tao. An optimal semi-online algorithm for a single machine scheduling problem with bounded processing time. *Information Processing Letters*, 110(8-9) :325–330, 2010.

[90] Ji Tian, Ruyan Fu, and Jinjiang Yuan. Online over time scheduling on parallel-batch machines : a survey. *Journal of the Operations Research Society of China*, 2(4) :445–454, 2014.

[91] Wajdi Trabelsi, Christophe Sauvey, and Nathalie Sauer. Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems. *Computers & Operations Research*, 39(11) :2520–2527, 2012.

[92] RN Uma, Joel Wein, and David P Williamson. On the relationship between combinatorial and lp-based lower bounds for np-hard scheduling problems. *Theoretical computer science*, 361(2-3) :241–256, 2006.

[93] Rob van Stee and Han La Poutré. Minimizing the total completion time on-line on a single machine, using restarts. *Journal of Algorithms*, 57(2) :95–129, 2005.

[94] Arjen PA Vestjens. *On-line machine scheduling*. Citeseer, 1997.

[95] Yong Wu, Yi-kun Huang, and QF Yang. Semi-online multiprocessor scheduling with the longest given processing time. *Journal of Zhejiang University : Science Edition*, 35 :23–26, 2008.

[96] Bo Xiong and Christine Chung. Completion time scheduling and the wsrpt algorithm. In *International Symposium on Combinatorial Optimization*, pages 416–426. Springer, 2012.

Hajar NOUINOU

Doctorat : Optimisation et Sûreté des Systèmes

Année 2021

Ordonnancement *semi-online* sur machine unitaire pour l'industrie du futur

Nous étudions la valeur de l'information dans des problèmes d'ordonnancement *semi-online* sur machine unitaire. Nous proposons ainsi des algorithmes *semi-online* pour résoudre ces problèmes et nous évaluons leurs performances. Contrairement aux problèmes d'ordonnancement classiques *offline* où le décideur connaît toutes les caractéristiques de l'instance à ordonnancer, dans les problèmes d'ordonnancement *online* ou *semi-online* la prise de décision est effectuée sans aucune information ou uniquement avec des informations partielles sur l'instance. Notre travail consiste à distinguer les informations qui peuvent améliorer la prise de décision dans un contexte d'ordonnancement *semi-online* des informations qui, même disponibles, n'apportent aucune amélioration. Nous traitons principalement les problèmes *semi-online* dont la fonction objective est la minimisation de la somme des dates de fin des tâches sur machine unitaire. Nous commençons par étudier plusieurs modèles *semi-online* avec des informations partielles sur les temps de traitement des tâches. Ensuite, nous considérons le problème avec une information sur les dates d'arrivée et finalement la combinaison de l'information sur les temps de traitements et les dates d'arrivée des tâches. Pour chaque problème étudié où l'information partielle est identifiée comme utile, nous proposons un algorithme *semi-online* intégrant l'information dans la prise de décision. Ensuite, nous évaluons sa performance à l'aide d'une analyse de compétitivité ou par étude expérimentale comparative.

Mots clés : ordonnancement (gestion) – algorithmes en ligne – optimisation combinatoire – temps réel (informatique).

Single Machine Semi-online Scheduling for Smart Manufacturing

We study the value of information in semi-online single machine scheduling problems. We propose semi-online algorithms to solve these problems and evaluate their performances. Unlike the classical offline scheduling problems where the decision maker knows all characteristics of the instance to be scheduled, in online scheduling problems the decision-making is performed without previous information or only with partial information about the instance. Our work consists in distinguishing information that can improve the decision-making in a semi-online scheduling context from information that, even if available, does not bring any improvement. We mainly deal with semi-online problems for minimising the total completion time on a single machine. We start by studying several semi-online models with partial information on processing times of jobs. Then, we consider the problem with information on jobs release dates and finally the combination of information on processing times and jobs release dates. For each studied problem where partial information is identified as useful, we propose a semi-online algorithm integrating the information into the decision-making. We then evaluate its performance using a competitive analysis or a comparative experimental study.

Keywords: production scheduling – online algorithms – combinatorial optimization – real-time data processing.

Thèse réalisée en partenariat entre :