Représentation compacte d'ensemble de solutions pour l'ordonnancement sous incertitude

Louis Rivière

THÈSE

En vue de l'obtention du
DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par

Louis RIVIERE

Le 11 décembre 2023

Représentation compacte d'ensembles de solutions pour l'ordonnancement sous incertitude

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et Télécommunications de Toulouse**

Spécialité : **Informatique et Télécommunications**

Unité de recherche :
LAAS - Laboratoire d'Analyse et d'Architecture des Systèmes

Thèse dirigée par
Christian ARTIGUES et Hélène FARGIER

Jury

M. Antoine JOUGLET, Rapporteur
M. Pierre MARQUIS, Rapporteur
Mme Marie PELLEAU, Examinatrice
M. Jean-Charles BILLAUT, Examinateur
M. Christian ARTIGUES, Directeur de thèse
Mme Hélène FARGIER, Directrice de thèse

"I understood then that it takes a powerful imagination to see a thing for what it really is."

-Norm Macdonald, Based on a True Story.

Représentation compacte d'ensembles de solutions pour l'ordonnancement sous incertitude

Résumé :

L'objectif de cette thèse est l'étude de méthodes permettant la prise en compte d'incertitude dans des problèmes d'ordonnancements à l'aide de structures de données représentant un ensemble de solutions de façon compacte.

La première partie de cette thèse présente les problèmes d'ordonnancement sous incertitude, et introduit les méthodes de résolution dont nous ferons usage dans cette thèse.

La seconde partie s'intéresse à des problèmes d'ordonnancement à une machine et des problèmes de jobshop, où l'incertitude porte sur les dates de disponibilité des tâches ou sur leurs durées. L'incertitude est modélisée par un ensemble de scénarios discrets, et nous supposons un modèle d'information minimaliste, dans lequel l'incertitude est levée lors du fait accompli. Nous considérons des objectifs stochastiques et robustes pour plusieurs critères réguliers, que nous utilisons pour guider la recherche de solution dans une phase hors-ligne, sur un ensemble de scénarios d'entraînement ; puis nous évaluons ces solutions sur un ensemble de scénarios de test dans une phase en ligne. Nous nous intéressons particulièrement aux différents types de décisions partielles qui peuvent être prises lors de la phase hors-ligne, telles que l'heuristique "First-in First-out", guidée par les informations révélées lors de la phase en ligne, complète la décision pour obtenir un ordonnancement. Les décisions partielles que nous considérons restreignent l'espace des solutions à des ensembles de séquences de tâches. Ces ensembles sont décrits par des structures de données comme les simples séquences (comme dans la littérature en ordonnancement stochastique ou robuste), des séquences de groupes d'opérations permutables, ou des diagrammes de décision multivalués. Nous proposons plusieurs méthodes pour prendre la meilleure décision partielle hors-ligne étant donnée la décision heuristique en ligne et le type de structure considérée. Des expérimentations évaluent les différentes approches selon les objectifs et les paramètres des instances considérées et montrent l'intérêt des nouvelles approches proposées.

Enfin, la troisième partie vise à formaliser l'étude, à la lumière du formalisme de la compilation de connaissance, des différentes structures utilisées (appelées langages de représentation dans ce contexte). Dans cette partie, nous considérons une variante décisionnelle d'un problème d'ordonnancement à une machine. L'incertitude n'y est pas modélisée explicitement, mais l'objectif est de déterminer, en fonction des structures de données utilisées pour représenter l'ensemble des solutions du problème, quels aléas il est possible de prendre en compte, par exemple en remettant à jour rapidement l'ensemble des solutions face à un aléa. L'objectif étant de calculer dans une phase hors-ligne une représentation du problème, qui serve d'outil à un décideur et permette la prise en compte d'un maximum d'aléas lors de la phase en ligne. Différents langages qui représentent le problème par un ensemble de séquences

de tâches correspondant à des solutions admissibles sont considérés (parmi lesquels les structures de données de la seconde partie). Les langages sont comparés selon des critères d'expressivité, de compacité, et en fonction des requêtes qu'ils supportent. Les résultats établis permettent de dresser une carte de compilation des langages étudiés et de guider un utilisateur dans le choix d'un langage de représentation.
Les résultats de l'étude menée dans cette thèse montrent expérimentalement que le calcul de solution en amont est très difficile, mais utiliser des départs à chaud semble prometteur. Les résultats théoriques permettent de comparer plusieurs langages, mais n'identifient pas de candidat pleinement satisfaisant pour le problème considéré. Enfin, les résultats suggèrent de nombreuses pistes de travail futures, en particulier à l'intersection des domaines de l'ordonnancement et de la compilation de connaissances, qui sont rarement considérés conjointement.

Mots clés : Ordonnancement sous incertitude, Compilation de connaissances, Proactif-réactif , Robuste/stochastique, Représentation compacte de solutions

Compact representation of sets of solutions for scheduling under uncertainty

Abstract : In this thesis, we study how data structures succinctly representing sets of sequences can be used to handle uncertainty within scheduling problems.
The first part of the thesis introduces scheduling problems and scheduling problems under uncertainty ; and describes the approaches that we will use to tackle them.
The second part takes interest in single machine scheduling problems and jobshop scheduling problems, in which uncertainty lies within release dates of tasks or on their duration. Uncertainty is modeled as a discrete set of scenarios, and we assume a minimalistic information model, in which uncertainty is lifted after the fact. We study stochastic as well as robust objectives for several regular criteria, used to guide search in an offline phase, on a training set of scenarios ; we then evaluate the solutions obtained on a testing set of scenarios in an online phase. We are specifically interested in the several types of partial decisions that can be taken during the offline phase, such that a simple "First-in First-out" heuristic, guided by information revealed during the online phase, takes the remaining decisions to reach a schedule. Partial decisions we consider restrict the solution space to sets of sequences of tasks. These sets are described by data structures such as sequences (as in the stochastic and robust scheduling literature), sequences of permutable groups, or multivalued decision diagrams. We introduce several methods to compute the best possible partial decision offline, given the online heuristic decision and the data structure considered. Experimentations evaluate the different approaches for the different objectives and instance parameters and show their relevance.
Finally, the third part aims at formalizing the study of the different data structures (called languages in that context) in the light of the tools of knowledge compilation. In this part, we consider a decision variant of a single machine scheduling problem. Uncertainty is not explicitly taken into account, but rather the goal is to find out which perturbations it is possible to handle, depending on the data structures used to represent the problem and its set of solutions, for example by quickly updating the data structure to reflect a change in problem data. The goal is to compute in an offline phase a representation of the problem that will be used as a tool for a decision-maker and allow them to handle as much perturbation as possible in the online phase. Several languages able to represent the problem as a set of sequences of tasks corresponding to admissible solutions are considered (among which the data structures of the second part). The languages are compared along their expressivity, succinctness, and by the requests they satisfy in polynomial time. The results achieved allow us to draw a compilation map of the studied languages and to guide a decision-maker in choosing a representation language.
The conclusions drawn in this thesis are a first step towards the formalization and usage of sets of sequences as partial decisions in the context of scheduling problems. Experimental results show that computing optimal solutions in the offline phase is very difficult, but that using warm-start strategies allows some gain over previous

methods. Theoretical results allow the comparison of several languages ; however, for the considered problem, no language was found to be completely suitable.
The results of this thesis suggest numerous leads for future works, specifically at the intersection of the domains of scheduling and knowledge compilation, two fields that are rarely considered jointly.
Keywords : Scheduling under uncertainty, Knowledge compilation, Proactive-reactive , Robust/Stochastic, Compact representation of solutions

Remerciements

Je souhaite tout d'abord remercier le jury de cette thèse et en particulier les rapporteurs, qui ont pris le temps de s'intéresser à mes travaux hétérogènes et dont les remarques ont permis, je l'espère, d'en faciliter la compréhension.

Je souhaite ensuite tout particulièrement remercier mes encadrants de thèse. Sans Hélène, cette thèse n'aurait jamais commencé. D'abord, parce que c'est elle qui est à l'origine du projet dans lequel s'inscrit ma thèse, mais aussi parce qu'elle m'a souvent révélé comment emprunter les pistes que je n'arrivais pas à distinguer. Tout au long de cette thèse, Hélène m'a montré de nouvelles façons de considérer les choses, et je pense avoir beaucoup appris de nos interminables discussions.

Sans Christian, cette thèse n'aurait jamais abouti. La disponibilité et les conseils pragmatiques de Christian m'ont guidé à travers les nombreuses épreuves qui ont jonché cette thèse. Il ne fait aucun doute que c'est son influence qui m'a permis, bon grès mal grès, de terminer ce que j'avais commencé. Il m'a montré comment voir la médaille avant son revers, et comment oser mettre un point final là où j'en aurai mis trois.

Quand j'ai quitté mon poste en 2020, décidé à commencer une thèse, je savais que ce ne serait pas une partie de plaisir, et que je m'apprêtais à passer les pires années de ma vie. Ah, comme j'ai maudit le Louis d'alors, qui s'est jeté à l'eau, en espérant apprendre à nager durant la chute. Bien des fois, j'ai considéré l'abandon, et autant de fois, j'ai été rattrapé par une main tendue, une discussion, ou même un sourire. Pour cette raison, je tiens à remercier toutes les personnes que j'ai côtoyées durant ces trois années de thèses, qui ont, parfois sans même le savoir, contribué à l'élaboration de cette thèse.

Je souhaite enfin remercier le lecteur, pour avoir exhumé ces travaux de leur étagère poussiéreuse, et d'avoir fait vivre, encore une fois, mes idées.

Merci,

Louis Rivière

Table des matières

Table des figures

Liste des tableaux

Liste des Algorithmes et des Modèles

Notations

— $\oplus$: Opérateur d'agrégation entre scénarios
— 1P : Problème à une machine
— d : Date de livraison d'une tâche
— D_χ domaine des variables de χ
— D : Vecteur des dates de livraison
— $D(\mathcal{G})$: Densité des contraintes de précédence d'un graphe
— E : Ensemble des contraintes de précédence
— G : Groupe de tâches
— H : Politique de second niveau
— I : Sémantique d'un langage
— JSP : Problème du Jobshop
— $\mathcal{L}$: Couche d'un MDD / Ensemble de langages
— L : Langage
— M : Ensemble de machines
— N : Ensemble de tâches
— s : Variable de décision de date
— S : Ensemble de scénarios
— p : Durée d'exécution / Chemin dans un MDD
— P : Vecteur des durées d'exécution / Nœuds permutables d'un arbre PQR / Polynôme
— Q : Nœuds réversibles d'un arbre PQR
— $\mathcal{Q}$: Fonction quantile / Ensemble des distributions de probabilité
— r : Date de disponibilité
— R : Vecteur des dates de disponibilité / Nœuds ordonnés d'un arbre PQR / Nœud racine d'un MDD
— T : Translation / Nœud terminal d'un MDD
— $\mathbb{T}$: Translation entre ensembles d'ordonnancements et ensembles de séquences
— $\mathcal{T}$: Translation entre ensembles de variables de séquence et ensembles de variables de position
— $\mathcal{U}$: Ensemble de tous les objets
— $\mathcal{X}$: Décision de premier niveau
— W : Largeur maximale d'un MDD
— γ : Critère d'optimisation
— Δ : Variabilité entre scénarios / Plus petit sous-arbre PQR commun
— ζ : Variable de décision de position
— σ : Variable de décision de séquence
— τ : Translation entre ordonnancements et séquences
— φ : Formule d'un langage
— Φ : Syntaxe d'un langage
— χ : Ensemble de variables de décision

Notions rencontrées

Par commodité, cette section regroupe un ensemble de notions abordées dans cette thèse qui peut servir de rappel lors de la lecture du manuscrit, ou rediriger vers la définition des concepts.

Cohérence d'une formule Voir définition 6.3.2.2
Une formule est cohérente lorsque ses données séquentielles (décrivant un ensemble de séquences) sont en correspondance avec ses données temporelles (décrivant des données associées aux tâches).

Decision de premier niveau Voir section 3.1.2.1
La décision de premier niveau restreint l'espace de décision. Dans cette thèse, cette restriction correspond à un ensemble de séquences. Les décisions de restriction qu'il est possible de prendre dépendent de la stratégie.

Démarrage à chaud Voir section 4.3.2
On parle de démarrage à chaud lorsqu'une solution de départ (par exemple une séquence) est fournie à une méthode. Le temps de chauffe détermine combien de temps est alloué à la recherche d'une bonne solution de départ, au détriment du temps restant à la méthode pour chercher à améliorer cette solution.

Langage-ensemble (SL) Voir section 6.6.6
Le langage-ensemble associé à un langage est le langage dont les formules sont des ensembles de formules du langage associé. La sémantique du langage ensemble se comprend comme l'union des ensembles représentés par les formules (ou la conjonction des fonctions représentées). Les langages-ensembles présentés dans cette thèse ont la particularité d'être complets.

Langage restreint (L^*) Voir section 6.5.4
Le langage restreint associé à un langage ne contient que les formules ayant une représentation équivalente pour le problème à une machine.

Ordonnancement semi-actif Voir section 1.2.1 et définition 6.3.1.1
Un ordonnancement est dit semi-actif (ou calé à gauche), si aucune tâche ne peut être commencée plus tôt sans changer l'ordre des tâches ou rendre l'ordonnancement inadmissible. Les ordonnancments semi-actifs sont dominants pour les objectifs réguliers.

Politique de second niveau Voir section 3.1.2.2
La politique de second niveau est définie par la stratégie. À chaque moment de décision, elle détermine quelle décision est prise. Nous utilisons exclusivement la politique FIFO dans cette thèse, dont l'implémentation varie subtilement en fonction de la stratégie.

Stratégie à deux niveaux Voir section 3.1.2
La stratégie définit à la fois l'ensemble des décisions de premier niveau qu'il est possible de prendre à l'issue de la phase proactive, et la politique de second niveau utilisée lors des moments de décision de la phase réactive.

Translation entre objets Voir définitions 6.4.3.1 et 6.4.2.1
Une translation établit une correspondance arbitraire entre des objets de types différents. Ces correspondances sont utilisées pour comparer des langages représentant des choses différentes, mais dont on établit la similarité.

Validité d'une décision de premier niveau Voir définition 3.1.2.1
On parle donc de décision de premier niveau valide *par rapport* à une politique de second niveau lorsque quel que soit le scénario, la politique de second niveau pourra extraire de la décision de premier niveau un ordonnancement admissible.

Introduction

Contexte

La résolution de problèmes d'optimisation se base souvent sur des données déterministes. Mais en raison d'une grande variété de facteurs comme l'erreur de mesure, la mauvaise estimation, ou même la mauvaise modélisation des objectifs, les données du problème posé sont souvent différentes de celles du problème que l'on aurait voulu résoudre. Ce phénomène mène parfois au calcul de solutions sous-optimales voire irréalisables et donc inapplicables à la situation réelle. La prise en compte de cette méconnaissance est donc cruciale et est l'objet de nombreux travaux visant à garantir un niveau de qualité des solutions mises en place et/ou en maximiser la qualité moyenne.

Les problèmes d'ordonnancement constituent une famille particulière de problèmes d'optimisation, très étudiés pour leur applications pertinentes dans de nombreux domaines de l'économie actuelle (fabrication, logistique, gestion de projet, ...) et sont sujets à de nombreuses perturbations. La mise en œuvre d'une solution étant séquentielle dans ces problèmes, il est parfois possible de raffiner la connaissance du problème au fur et à mesure de la mise en pratique de la solution, mais la complexité de ces problèmes peut rendre le calcul répété de solution à chaque nouvelle information obtenue prohibitif. Par conséquent, les études s'intéressant aux problèmes d'ordonnancement dits "sous incertitude" se multiplient.

Une approche permettant de s'adapter à la variabilité due à la méconnaissance des instances consiste en un calcul préalable d'un bon ensemble de solutions. Cet ensemble de solutions peut permettre, lors de la mise en application de la solution, de choisir parmi les solutions celle qui correspondra le mieux à l'instance qui se pose réellement. C'est un type de solution flexible, mêlant les méthodes de calcul de solution robuste, qui cherchent une unique solution de bonne qualité pour un ensemble de d'instances réelles possibles, et les techniques réactives qui modifient une solution courante lorsque de nouvelles connaissances sont disponibles. La mise en œuvre séquentielle des solutions nécessite que les ensembles de solutions soient structurés de telle sorte que l'on puisse passer d'une solution à une autre au cours du temps et de l'amélioration des connaissances.

Il existe de nombreuses façons de représenter un tel ensemble de solutions. Et plusieurs types de représentations ont déjà été présentés théoriquement et utilisés en pratique dans le contexte de l'ordonnancement sous incertitude. Ces ensembles de solutions peuvent être étudiés en utilisant le formalisme de la compilation de connaissance, un domaine de recherche s'intéressant aux capacités de représentation et aux propriétés des structures de données. Il est ainsi possible de comparer différentes façon de représenter des ensembles de solutions pour un problème selon des critères d'expressivité de la représentation, de compacité, et selon les opérations qu'il est facile de réaliser avec.

L'objectif de cette thèse est d'étudier expérimentalement les gains que peut apporter le calcul d'ensembles de solutions dans un contexte d'ordonnancement sous incertitude, en particulier de problèmes d'ordonnancement robustes ou stochastiques avec recours, et d'initier un rapprochement entre les domaines de l'ordonnancement et de la compilation de connaissances, pour étudier les possibilités théoriques de ces approches. A terme, l'objectif est aussi de pouvoir mettre à profit le facteur humain, qui est à la fois une source d'erreur dans un système, mais aussi un grand facteur de résilience. En mettant un ensemble de solutions à disposition d'un décideur humain, et en le munissant d'un ensemble d'opérations pour le manipuler, l'idée est qu'il soit possible d'adapter la mise en oeuvre des solutions à des perturbations n'ayant pas été imaginées lors de la résolution.

Organisation du manuscrit

Cette thèse est composée de trois parties.
La première partie constitue une introduction détaillée, visant à donner au lecteur les éléments nécessaires à la bonne compréhension de son contexte et son cadre. Y sont présentés de façon générale, dans deux chapitres distincts :

- Le concept d'ordonnancement déterministe (Chapitre 1) : dans ce chapitre, nous verrons ce qui caractérise un problème d'ordonnancement, et décrirons quelques-unes des nombreuses variantes existantes. Nous nous intéresserons aux types de solutions que l'on peut donner à ces problèmes. Puis nous présenterons les méthodes de résolution existantes.
- Le concept d'incertitude et sa prise en compte dans les problèmes d'ordonnancement (Chapitre 2) : dans ce chapitre, nous introduirons d'abord les façons dont il est possible de prendre en compte l'incertitude dans la modélisation d'un problème d'optimisation, ce qui dépend de la connaissance de cette incertitude, de l'objectif souhaité, et de la prise de connaissance possible. Nous nous pencherons aussi sur la manifestation de l'incertitude dans les problèmes d'ordonnancement. Nous montrerons l'importance de sa prise en compte, puis, nous étudierons les méthodes de résolution spécifiques aux problèmes sous incertitude. Enfin, nous répertorierons plusieurs structures de données utilisées dans le cadre de résolution de problèmes d'ordonnancement sous incertitude pour représenter, de façon plus ou moins compacte, des ensembles de solutions.

Dans la deuxième partie, nous décrivons les méthodes d'optimisation à deux niveaux basées sur des structures représentant des ensembles de solutions étudiées et développées dans cette thèse, ainsi que les gains qui peuvent êtres obtenus par ces méthodes. Des applications à des problèmes à une machine et d'atelier "jobshop" avec critère d'optimisation robuste et stochastique permettront de les évaluer expérimentalement.

Cette partie se divise en trois chapitres :

- Le premier chapitre (Chapitre 3) présente plus précisément et formellement

la famille d'approches à deux niveaux étudiée. Nous préciserons les objectifs et les hypothèses dans lesquelles nous opérons. Nous présenterons l'environnement d'expérimentation utilisé pour comparer différentes méthodes entre elles. Nous décrirons ce qu'est une décision partielle de premier niveau valide et les décision de recours du second niveau que nous utiliserons. Nous présenterons aussi des cas standards de décision partielle : lorsque aucune décision n'est prise ou qu'une décision complète de séquencement est prise. Des analyses de résultats expérimentaux initieront la comparaison des différentes méthodes et estimeront les limites de ces approches.

— Le second chapitre (Chapitre 4) s'intéresse au cas où la décision de premier niveau est déterminée par une séquence de groupes d'opérations permutables. Nous étudierons dans quel cas cette décision est valide, et proposerons des méthodes de calcul de solution. Nous terminerons par la présentation des résultats expérimentaux en comparant les nouvelles approches proposées avec les approches standards.

— Le troisième chapitre (Chapitre 5) s'intéresse au cas le plus large ou la décision de premier niveau est la plus libre possible et permet de décrire un ensemble arbitraire de séquences. En particulier lorsque la décision prend la forme d'un diagramme de décision multivalué, ou d'un simple ensemble de séquences. Dans ce cas encore, nous étudierons les conditions de validité d'une telle décision, puis nous présenterons plusieurs méthodes de résolution du problème, avant de proposer quelques résultats préliminaires.

Enfin, la troisième partie tente d'éclairer les capacités théoriques des différentes structures de représentation utilisées dans la seconde partie à la lumière du formalisme de la compilation de connaissances, afin d'en déduire leurs utilité pratique :

— Dans le premier chapitre (Chapitre 6) , nous présenterons le cas d'application considéré. Puis nous introduirons en détails les éléments et le formalisme utilisés dans cette partie : nous verrons comment les structures de représentation de solutions peuvent être considérées comme des langages, et comment ces langages, même s'ils décrivent des solutions de manières différentes, peuvent être comparés.

— Dans le second chapitre (Chapitre 7) nous démontrerons des résultats concernant ce qu'il est possible de représenter en utilisant ces structures (leur expressivité), ainsi que la taille nécessaire à cette représentation (la complexité spatiale), et les résultats concernant les capacités opératoires des différentes représentations, c'est-à-dire les opérations qu'il est facile ou non de réaliser lorsque l'information est représentée par une structure.

Pour terminer, nous discuterons les résultats présentés dans cette thèse ainsi que les perspectives qui s'en dégagent dans une conclusion.

On trouvera dans l'annexe A les résultats numériques détaillés des expériences menées en partie II.

Première partie

Contexte

Il est dans notre intérêt d'influencer l'état futur du monde. Toute influence consciente n'est possible que par l'intermédiaire du choix de nos actions. Par conséquent, nous souhaitons souvent que les actions dont nous décidons aient les propriété permettant d'accéder à un futur présentant certaines caractéristiques. Malheureusement, s'il est toujours possible de décrire les propriétés souhaitées d'une action dans notre langage, l'existence d'une action satisfaisant ces propriétés n'est pas garantie. De plus, même si une telle action existe, il est parfois difficile de savoir comment mettre en oeuvre cette action (en déduire des propriétés actionnables).

Par exemple, pour préparer un déménagement, j'ai à disposition 10 cartons. Je souhaite ranger toute mes affaires dans le plus petit nombre de cartons possible. Toutefois, je ne sais pas si un tel rangement existe (peut-être faudra-t-il plus de 10 cartons), et avoir décrit le rangement souhaité de cette façon (le rangement le plus économe en cartons) ne permet pas de savoir comment le réaliser. Les problèmes de décision et d'optimisation formalisent cette recherche en modélisant les propriétés actionnables des actions pertinentes par un ensemble de variables de décisions. Choisir une action, donner une valeur aux variables de décision, c'est proposer une solution au problème posé. Cette solution est admissible si elle possède bien les propriétés attendues. Par exemple, on va modéliser par des variables de décision le fait de ranger un objet dans un carton donné. Une affectation de chaque objet à un carton constitue une solution, un rangement candidat qui pourrait être celui que l'on cherche. Cette solution est admissible si elle affecte bien chaque objet à un carton en respectant sa capacité, et si c'est la solution admissible utilisant le moins de cartons, c'est bien le rangement que l'on cherchait ; et on sait désormais comment le mettre en action.

La résolution d'un problème d'optimisation nécessite en général des informations sur l'état du monde dans lequel l'action doit être prise (la taille des objets, des cartons, ...). Mais ces informations ne sont souvent connues que de manière imparfaite. Par conséquent, des décisions prises en supposant l'exactitude des informations peuvent ne pas êtres admissibles lors de leurs mise en action. Un objet peut être mal mesuré, un carton inutilisable, ... Le paradigme de l'optimisation sous incertitude tente de pallier à cet état de fait en considérant cette possible méconnaissance des données et en cherchant des solutions ayant certaines propriétés vis-a-vis de cet inconnu.

En particulier, les "problèmes d'ordonnancement" dénomment un large ensemble de problèmes de décision et d'optimisation visant à décider la façon dont un ensemble de tâches doivent être effectuées (ou ordonnancées). De tels problèmes se posent naturellement lors de toute organisation d'activités, mais c'est autour de la fin du 19ème siècle, avec la complexité grandissante des processus de production, que des méthodes scientifiques visant à l'optimisation de la production sont apparues, et que la question de l'ordonnancement à été posée de façon distincte. Herrmann [2006] identifie ensuite deux évènements clés dans la façon d'aborder ces

problèmes. Le premier est la création des graphes de Gantt au début du 20ème siècle, définissant les critères et décisions d'intérêt pour la production. Le second, est l'avènement des ordinateurs, menant au développement d'algorithmes d'optimisation spécialisés pour les problèmes d'ordonnancement, y compris sous incertitude, dans la deuxième moitié du 20ème siècle.

La partie qui suit vise à introduire les problèmes d'ordonnancement sous incertitude de façon générale pour donner au lecteur les éléments nécessaires à la bonne compréhension du contexte et du cadre de cette thèse. Dans le chapitre 1 nous définirons plus formellement ce que sont ces problèmes d'ordonnancement, ainsi que les notations qui seront utilisées tout au long de cette thèse ; nous verrons quelles décisions peuvent constituer une solution de ces problèmes ; et nous décrirons les approches exactes et approchées utilisées pour les résoudre.
Puis, dans le chapitre 2 nous introduirons plus particulièrement les problèmes d'ordonnancement sous incertitude. Nous commencerons par décrire différentes façon de modéliser l'inconnu dans un problème, ainsi que les différentes façon dont la connaissance peut être apportée ; puis nous verrons les méthodes existantes utilisées pour résoudre les problèmes d'ordonnancement sous incertitude, avant de décrire certaines structures de données que nous utiliserons dans cette thèse pour faire face à l'incertitude.

Chapitre 1

Ordonnancement

Sommaire

1.1 Le problème d'ordonnancement

1.1.1 Introduction et définitions

Le terme "problèmes d'ordonnancement" dénomme une grande variété de problèmes, plus ou moins difficiles à résoudre, plus ou moins expressifs et généraux. Pour Esquirol and Lopez [1999], les problèmes d'ordonnancement consistent à organiser dans le temps la réalisation de **tâches** compte tenu d'un ensemble de **contraintes**. Cet **ordonnancement** de tâches constitue une solution au problème d'ordonnancement. Cette solution est *admissible*, si elle respecte bien toutes les contraintes modélisées. Elle peut aussi être évaluée en fonction d'un ou plusieurs **critères** qui peuvent être agrégés au sein d'une fonction objectif à minimiser ou maximiser.

Une tâche (appelée opération dans le contexte de l'ordonnancement d'atelier présenté plus loin) est la représentation, dans un problème d'ordonnancement, d'une activité réelle. En fonction des contraintes ou critères considérés, différents aspects de cette activité seront modélisés dans le problème comme des données relatives à cette tâche. Par exemple, des informations concernant les ressources nécessaires à l'exécution de la tâche, concernant sa durée d'exécution dans telle ou telle condition, ou bien encore s'il est nécessaire ou non de l'effectuer d'une seule traite. L'ensemble des tâches est noté N.

Les contraintes s'appliquant à l'ordonnancement des tâches définissent plus spécifiquement encore le problème particulier considéré. Il n'y a pas plus de limite à la nature et au nombre de ces contraintes qu'à l'inventivité humaine. Toutefois, ces contraintes se divisent pour les plus courantes en deux catégories : les contraintes dites *temporelles*, limitant l'organisation des tâches dans le temps, et les contraintes dites de *ressources*, spécifiant les éléments nécessaires à l'exécution d'une tâche. Nous en présenterons dans la section 1.1.2 une sélection parmi les contraintes les plus courantes et les plus pertinentes pour nos travaux.

Enfin, comme évoqué plus haut, les problèmes d'ordonnancement différencient parfois les solutions admissibles à l'aide d'un ou plusieurs critères. Les critères établissent un ordre de préférence entre les différentes solutions en fonction des propriétés de celles-ci et peuvent être agrégés au sein d'une fonction objectif. On dit, lorsque l'on cherche une solution admissible d'un problème pour lequel l'objectif est défini, que l'on cherche à optimiser cet objectif. Il existe pléthore d'objectifs qu'il est possible d'optimiser. Certains seront présentés par la suite dans la section 1.1.2.

De par leurs grande versatilité, les problèmes d'ordonnancement permettent de modéliser et résoudre un grande nombre de problèmes de décision tirés de problématiques de domaines variés. On trouve par exemple : des problèmes de production industrielle [Herrmann, 2006], des problèmes d'ordonnancement dans les systèmes d'exploitations [Błażewicz et al., 2001], des problèmes de gestion de projet [Schwindt and Zimmermann, 2015], des problèmes de transport (aérien, ferroviaire, routier) [Ikli et al., 2021, Leutwiler and Corman, 2023, Castelli et al., 2004], etc ...

Cette variété d'applications, très pertinentes dans notre société aujourd'hui, explique la prolifération des articles scientifiques traitant de problèmes d'ordonnancement depuis le milieu du 20ème siècle. On relèvera néanmoins certaines critiques plus récentes (portées à l'encontre du domaine de l'aide à la décision en général) sur la façon dont cette recherche soutient les intérêts des décideurs, qui sélectionnent les contraintes et les critères des problèmes, souvent contraires aux intérêts des autres parties [Brans and Gallo, 2007, Ormerod and Ulrich, 2013, Bellenguez et al., 2023].

1.1.2 Variantes

L'abondance des problèmes d'ordonnancement rencontrés a rapidement motivé l'introduction de notations abrégées pour décrire les différents problèmes. Notablement, la notation de Graham [Graham et al., 1979] classifie les problèmes d'ordonnancement selon trois champs $\alpha|\beta|\gamma$. Depuis, de nombreuses propositions visent à étendre ces notations pour permettre la description et l'unification d'un éventail toujours plus large de problèmes distingués, qui dépassent nos besoins. Nous prenons donc pour base la notation originelle à trois champs de Graham, en y incorporant les extensions pertinentes pour notre problème décrites en plus de détails dans Billaut et al. [2013], Gourgand et al. [2000].

L'environnement machine est décrit dans le champ α. Par exemple, dans cette thèse nous nous intéresserons à des problèmes dans lesquels α vaut 1 ou J. La notation $\alpha = 1$ correspond à un problème dans lequel toutes les tâches sont exécutées l'une après l'autre par une seule machine (dit **problème à une machine**). Si $\alpha = J$ alors dans le problème considéré (dit **problème du jobshop**), les tâches sont regroupées en ensembles de tâches appelées **jobs**. Les tâches d'un job doivent êtres exécutées l'une après l'autre. Chaque tâche d'un job doit être exécutée sur une machine particulière parmi M machines.
On mentionne aussi le problème très général et très difficile (voir Ganian et al. [2020] pour une carte de complexité du problème) dit de "RCPSP" (Ressource Constrained Project Scheduling Problem), indiqué par $\alpha = PS$ [Billaut et al., 2013]. Dans ce problème, chaque tâche nécessite, pour son exécution, l'utilisation d'une quantité fixe d'un ensemble de ressources, dont les capacités sont connues. On peut remarquer qu'il est possible de modéliser les problèmes précédents en choisissant judicieusement ces capacités et utilisations de ressources.

Les (potentiellement multiples) caractéristiques des tâches considérées sont renseignées dans le champ β. Par exemple, dans cette thèse, nous serons confrontés aux cas où $prec \in \beta$, signifiant la considération de **contraintes de précédence** entre les tâches. Classiquement, lorsque toutes les tâches doivent êtres exécutés une seule fois, une contrainte de précédence $(i, j) \in N^2$, établit que la tâche j ne peut être ordonnancée qu'après la fin de la tâche i. L'ensemble des contraintes de précédence est noté E. Nous nous intéressons aussi au cas où $r_j \in \beta$, indiquant que chaque tâche possède une **date de disponibilité** à partir de laquelle il est possible de la démarrer. De façon similaire, la présence de la mention d_j dans β signale que chaque tâche possède une **date de livraison** à partir de laquelle elle est considérée en retard si elle n'est pas terminée. On note $\tilde{d}_j$ si ce retard n'est pas admis. Enfin, on peut aussi trouver dans β des précisions sur p_j : la **durée d'exécution** des tâches, mais par défaut, comme c'est le cas dans cette thèse, il n'y a pas de restriction particulière sur ces durées.

Les **critères** considérés, s'il y en a, sont décrits dans le champ γ. Il est utile, pour envisager certains de ces critères, d'introduire des variables additionnelles décrivant les propriétés des solutions. Ainsi, C_j représente, pour un ordonnancement donné, la **date de fin** de la tâche j. On note aussi, le cas échéant, $L_j = C_j - d_j$ le **retard** de la tâche j.
Étant données ces notations, on peut décrire les critères d'intérêt dans nos travaux. Le critère $\gamma = C_{MAX} = \max_{j \in N} C_j$ indique que l'on cherche à minimiser la date de fin de la tâche qui termine son exécution le plus tard. Similairement, si $\gamma = L_{MAX} = \max_{j \in N} L_j$, on cherche à minimiser le retard de la tâche la plus en retard. Enfin, nous verrons le critère $\gamma = \sum C_j = \sum_{j \in N} C_j$ visant à minimiser la somme des dates de fin des tâches. On remarquera que tout les critères présentés sont croissants en fonction des dates de fin C_j (ces critères sont dits *réguliers)*, par conséquent, on préférera toujours à un ordonnancement, toute choses égales par ailleurs, un

autre ordonnancement dans lequel les tâches terminent leurs exécutions plus tôt (cf section 1.2.1). Ce n'est toutefois pas toujours le cas, par exemple si l'on considère un critère pénalisant le fait de terminer une tâche en avance aussi bien qu'en retard, ce qui ne sera pas le cas dans cette thèse.

1.2 Solutions des problèmes d'ordonnancements

1.2.1 Solutions complètes : ordonnancements et notion de dominance

Comme discuté plus tôt, les problèmes d'optimisation impliquent la recherche de décisions, que l'on modélise par des variables de décision décrivant les actions envisagées. Pour les problèmes d'ordonnancement, la décision la plus informative et utilisée habituellement met en jeu les **variables de décision de date**, décrivant un ordonnancement. Un ordonnancement correspond à l'affectation d'une date de début pour chaque tâche.
Formellement, on note $\chi_{date} = \{s_1, s_2 \dots s_{|N|}\}$ l'ensemble des variables de décision d'ordonnancement. Pour chaque tâche j, la variable s_j représente sa date de début. A chacune de ces variables peut être affectée une valeur dans son **domaine** $D_{s_j} = \mathbb{R}^+$. La valeur affectée à une variable s_j se note $\check{s}_j$. Une solution est une affectation de toutes les variables de décision, notée $\check{s}_\chi = (\check{s}_1, \check{s}_2, \dots, \check{s}_{|N|})$, son domaine est $\mathbb{R}^{|N|}$.
Pour certains problèmes d'ordonnancement, l'affectation des variables de décision d'ordonnancement (décrivant la date de début des tâches) ne constitue pas une décision suffisante pour décrire une solution. C'est le cas par exemple si les tâches peuvent êtres exécutées selon plusieurs modes différents (il faut alors décider, par exemple, de la machine qui exécutera la tâche), ou si la préemption des tâches est autorisée (il faut alors décider de l'interruption et la reprise des tâches). Dans cette thèse, l'affectation des variables de décision d'ordonnancement décrit complètement une solution dans laquelle les tâches sont démarrées aux temps décidés, en utilisant les seules ressources possibles, puis doivent s'exécuter sans interruption jusqu'à leur complétion. Aucune décision supplémentaire n'est donc nécessaire pour caractériser complètement la solution.

Un ordonnancement est dit **admissible** s'il respecte toutes les contraintes du problème modélisé. Pour un critère donné, une solution admissible est dite **optimale** si aucune autre solution admissible ne lui est préférée. On appelle **semi-actif** un ordonnancement pour lequel il n'existe pas de solution admissible dont les tâches démarrent au moins aussi tôt, sans modifier l'ordre dans lequel les tâches démarrent. Un ordonnancement est **actif** s'il est semi-actif, et qu'il n'existe pas non plus de solutions admissibles dont les tâches démarrent au moins aussi tôt, quelque soit leur ordre. Enfin, un ordonnancement est dit **sans délai**, s'il correspond à une mise en oeuvre dans laquelle une tâche est toujours lancée lorsqu'il est possible de le faire. Plus formellement, soit un ordonnancement $\check{s}_\chi$, soit l'ensemble des tâches

$i_1, \ldots, i_q, \ldots i_{|N|}$ triées dans l'ordre croissant de leurs dates de début. $\check{s}_\chi$ est sans délai ssi pour toute tâche i_q, il n'existe pas $t < \check{s}_{i_q}$ telle que démarrer les tâches $i_1, ...i_{q-1}$ à leurs dates de début et démarrer i_q à t soit admissible. Les ordonnancements sans délai sont actifs.

Dominance pour les critères réguliers De façon générale, en optimisation, un sous-ensemble de solutions est dit **dominant** si l'on peut s'y restreindre pour la recherche de solution. C'est-à-dire qu'il contient au moins une solution admissible s'il en existe, et, si un critère est défini pour le problème, au moins une solution optimale.
Pour les problèmes d'ordonnancement avec critères réguliers, l'ensemble des ordonnancements actifs est dominant [Giffler and Thompson, 1960], donc l'ensemble des ordonnancements semi-actifs l'est aussi (le second incluant le premier). Mais l'ensemble des ordonnancements sans délai ne l'est pas. Lorsque l'ensemble des solutions semi-actives est dominant, puisque pour un ordre des tâches donné, il existe un seul ordonnancement "calé à gauche" semi-actif, on se limite souvent à la considération de la seule décision de l'ordre des tâches (ou **séquence**). Il est implicite que parmi tous les ordonnancements correspondant à cette séquence, c'est l'ordonnancement semi-actif que l'on décide de mettre en oeuvre.

Tout comme la dominance des ordonnancements semi-actifs permet d'accélérer la recherche de solution en ne considérant que l'ensemble des séquences (puisque sa taille est bien inférieure à celle de l'ensemble des ordonnancements), il existe d'autres notions de dominances qui permettent d'accélérer la résolution de problèmes d'ordonnancements (par exemple, dans Briand et al. [2007], Nowicki and Smutnicki [1996]). Une caractérisation plus rigoureuse du concept de dominance et de son intérêt est dressée dans Jouglet and Carlier [2011].

1.2.2 Décisions incomplètes : séquences

Formellement, on note $\chi_{seq} = \{\sigma_1, \sigma_2 \ldots \sigma_{|N|}\}$ l'ensemble des **variables de décision de séquence**. Pour la tâche i, la variable σ_i de domaine $[1 \ldots |N|]$ représente la position de la tâche dans la séquence. Ainsi, $\check{\sigma}_i = j$ correspond à la décision d'ordonnancer la tâche i en position j. On note que la décision des variables d'ordonnancement implique la décision des variables de séquence, tandis que la décision des variables de séquence ne font que restreindre le domaine des variables d'ordonnancement (le domaine résultant ne contient néanmoins qu'un seul ordonnancement semi-actif).

On déduit de ce qui précède qu'il n'est pas toujours nécessaire de fixer directement toute les décisions à effectuer, mais qu'il est possible de diviser le processus de décision en effectuant une sous-décision (ou **décision partielle**), qui devra être complétée par la suite. Cette division est particulièrement pertinente lorsque la complétion découle facilement de la décision initiale. C'est le cas dans l'exemple donné précédemment, où, après la décision initiale et partielle sur la séquence, une

procédure d'ordonnancement au plus tôt complète la décision de la meilleure des manières pour les objectifs réguliers. On notera aussi chez certains auteurs l'utilisation de la décision de séquence couplée à des algorithmes de listes pour le RCPSP [Sprecher et al., 1995] et pour le problème de machines parallèles [Graham, 1966] ne garantissant pas l'obtention de la meilleure complétion de décision mais une borne supérieure sur la valeur objectif. Nous verrons aussi dans le chapitre 2 que la prise de décision partielle permet plus de flexibilité pour adapter une décision à de nouvelles informations dans un contexte incertain. Notons qu'une décision n'est "partielle" que relativement à une décision "complète", et la décision partielle d'une séquence peut être une décision complète pour un autre problème.

Il faut distinguer les *décisions partielles*, qui sont des décisions impliquées par la décision complète (comme la décision de séquence par rapport à la décision d'ordonnancement), des *formulations alternatives*, qui prennent une décision équivalente, mais utilisent une modélisation différente. Par exemple, la décision concernant les temps de début de chaque tâche est parfois modélisée en utilisant la formulation alternative dite "time-indexed" [Pritsker et al., 1969, Koné et al., 2011, Ku and Beck, 2016] dans laquelle des variables de décision $x_{j,t}$ au domaine binaires indiquent si la tâche j commence au temps t. Mais des formulations alternatives des décisions partielles sont aussi possibles. Par exemple, les variables de décision de position $\chi_{pos} = \{\zeta_1, \zeta_2 \dots \zeta_{|N|}\}$ telles que $\check{\zeta}_i = j$ ssi la tâche j en position i permettent, comme les variables de χ_{seq}, de prendre une décision concernant l'ordre des tâches : c'est une formulation alternative de cette décision. Une autre formulation alternative pour la même décision met en jeu des variables booléenes $\delta_{i,j}$ qui indiquent si la tâche i précède la tâche j [Queyranne and Schulz, 1994].
Ces formulations alternatives ne sont pas toutefois complètement équivalentes, comme nous le verrons pour les méthodes de résolution dans la section 1.3.1.

1.3 Méthodes de résolution des problèmes d'ordonnancement

Les problèmes d'ordonnancement figurent en général au rang des problèmes difficiles (NP-difficiles au sens de la complexité). De nombreux travaux s'intéressent à la définition de la frontière entre les problèmes faciles et difficiles d'ordonnancement [Ganian et al., 2020, Lenstra et al., 1977, Lageweg et al., 1982]. Dans le contexte de cette thèse, nous nous intéressons principalement à des problèmes difficiles, qu'ils soient des problèmes à une machine, ou des problèmes de jobshop. Cette complexité justifie le grand nombre de travaux présentant des méthodes de résolution plus efficaces pour ces problèmes. Ces méthodes sont souvent tirées des techniques classiques d'optimisation combinatoire (programmation mathématique, programmation dynamique, procédures par séparation et évaluation, théorie des graphes), et des méthodes de résolution exactes ou approchées [Gotha, 1993].

1.3.1 Méthodes exactes

Lorsque l'on cherche à résoudre un problème d'optimisation, il ne suffit pas de trouver sa solution optimale , il faut aussi démontrer cette optimalité. Les méthodes de résolution qui garantissent à un utilisateur l'obtention de solution démontrée optimale sont appelées méthodes exactes. Dans le cas de problèmes difficiles, ces méthodes se basent en général sur l'exploration d'un arbre de recherche. Un arbre de recherche représente l'exploration des solutions par la restriction successives du domaine des variables de décision. L'ordre de ces décisions peut impacter fortement cette recherche, l'arbre n'est donc pas unique. Une fois les variables de décision suffisamment restreintes, l'admissibilité et la qualité des décisions possibles sont établies, et si nécessaire, la recherche se poursuit en modifiant les décisions effectuées. Des méthodes ingénieuses permettent de guider cette recherche et évitent souvent d'énumérer toutes les solutions de l'arbre.

1.3.1.1 Programmation linéaire en nombres entiers

Par exemple, il est possible de modéliser un problème d'optimisation combinatoire par la programmation linéaire en nombres entiers (PLNE). Cette modélisation consiste en la description du problème par un système d'équations linéaires pouvant impliquer des variables de décision entières. Ce système décrit les solutions admissibles du problème comme celles qui satisfont le système d'équations. Une fonction objectif linéaire peut permettre de capturer le critère d'optimisation.

$$\begin{aligned} \min \quad & c^T x \\ & Ax \leq b \\ & x \geq 0 \\ & x \in \mathbb{Z} \end{aligned}$$

Ces modèles de PLNE sont classiquement résolus par la méthode de séparation et d'évaluation (**branch and bound**). Cette méthode arborescente maintient au cours de la recherche des bornes inférieures et supérieures sur la valeur de la fonction objectif optimale. Les bornes supérieures (pour un problème de minimisation) sont obtenues lorsqu'une nouvelle meilleure solution admissible au problème est découverte. Les bornes inférieures sont établies par la **relaxation** du domaine des variables de décision entières. En permettant aux variables de décision entières de prendre des valeurs continues, on obtient un modèle de programmation linéaire (PL) dont la résolution est très efficace. Les solutions du PL obtenues, si elles sont entières, sont optimales. Sinon, elles permettent d'établir une borne inférieure. Les bornes ainsi obtenues permettent d'écarter les branches dont la meilleure solution (contenue dans le sous-arbre) est pire que la meilleure solution courante. On voit

bien maintenant comment le choix de variables impacte la résolution : d'une part de la façon dont elles divisent l'espace de l'arbre de recherche, et d'autre part par la qualité de leur relaxation.

Il existe de nombreuses modélisations PLNE pour les problèmes d'ordonnancement. Celles-ci se basent sur différentes variables de décision. Une comparaison de plusieurs formulations PLNE pour le RCPSP minimisant la date de fin est proposée par Koné et al. [2011]. Les auteurs montrent qu'il n'y a pas de formulation dominante pour toute les instances. Baker and Keller [2010] comparent différentes formulations pour le problème de minimisation du retard total sur une machine. Ils concluent qu'une formulation basée sur des variables de positions binaires ($\check{x}_{i,k} = 1$ ssi la tache i est en position k) obtient les meilleures performances sur les instances considérées. Queyranne and Schulz [1994] comparent les propriétés de plusieurs formulations de problèmes d'ordonnancement, dont certaines décrivent des dates de début de tâches, mais d'autres décrivent des séquences de tâches. Pour le problème de jobshop, Ku and Beck [2016] comparent plusieurs formulations existantes minimisant la date de fin au plus tard. Une dominance claire n'est pas établie entre les différents modèles. Toutefois, les auteurs comparent aussi les différentes formulations PLNE à une formulation de **programmation par contraintes (PPC)** et montrent que pour les instances les plus larges, l'approche PPC est plus efficace.

1.3.1.2 Programmation par contraintes

La programmation par contraintes permet de modéliser des problèmes d'optimisation en utilisant des contraintes, qui lient les affectations possibles des variables entre elles. On utilise le formalisme des problèmes de satisfactions de contraintes (CSP [1]). Un CSP est un triplet $(\chi, D, \mathtt{C})$ dans lequel χ est l'ensemble des variables de décision, D est l'ensemble de leurs domaine, et $\mathtt{C}$ est l'ensemble des contraintes. Une contrainte $\mathtt{C}_i = (\chi_i, \mathcal{R}_i)$ se définit par l'ensemble des variables qu'elle implique (χ_i) et par la relation définissant quelles valeurs de ces variables sont admissibles $(\mathcal{R}_i)$. Un critère d'optimisation peut aussi être défini.

Les modèles de programmation par contrainte sont habituellement résolus dans des méthodes arborescentes utilisant la **propagation** de contraintes. Propager les contraintes, c'est déduire, lors du branchement de l'arbre de recherche, des nouvelles restrictions de domaine, impliquées par les contraintes. On évite ainsi d'explorer des branches de l'arbre ne menant à aucune solution.

Les contraintes de PPC peuvent lier arbitrairement les variables et des algorithmes dédiés permettent une propagation plus ou moins efficace. Une des contraintes les plus courantes est la contrainte `"AllDifferent"`, qui s'applique à un ensemble de variables et n'admet que les affectations pour lesquelles chaque variable impliquée se voit affectée une valeur différente. Un exemple de propagation possible impliquant cette contrainte survient lorsqu'un ensemble de n variables doivent toute prendre des valeurs différentes parmi $k < n$ valeurs possibles dans l'union de leurs domaines.

1. Constraint Satisfaction Problem

La propagation peut alors conclure à la stérilité de la branche courante.

Selon Leung [2004], les méthodes de programmation par contrainte se prêtent particulièrement à la résolution de problèmes d'ordonnancement car ces derniers se modélisent par des contraintes nombreuses et qui impliquent relativement peu de variables ; des conditions propices à la propagation. De nombreux travaux utilisent la PPC pour résoudre différents problèmes d'ordonnancements [Fazel Zarandi et al., 2020]. L'efficacité des approches par PPC dépend aussi fortement des contraintes et propagateurs spécifiques.

Par exemple, l'outil **CP Optimizer** d'ILOG IBM permet de modéliser facilement les problèmes d'ordonnancement à l'aide notamment des **variables d'intervalle**. Une variable d'intervalle représente naturellement une tâche par sa date de début et sa date de fin. Ainsi, étant donnée deux variables d'intervalle A et B, on peut par exemple imposer une contrainte de précédence $A \prec B$ en écrivant :

$$\texttt{StartOf}(B) > \texttt{EndOf}(A)$$

L'autre type de variable qui nous intéresse particulièrement est la variable de séquence d'intervalles. En définissant une **variable de séquence d'intervalles** π sur les variables d'intervalle correspondant aux tâches, une contrainte de précédence $A \prec B$ devient simplement :

$$\texttt{Before}(\pi, A, B)$$

La contrainte "$\texttt{SameSequence}(\pi_1, \pi_2)$" sera d'intérêt dans cette thèse. Elle implique deux variables de séquence d'intervalles et étant donné une correspondance entre les variables d'intervalle qu'elles contiennent (implicite ici), n'admet que les solutions dans laquelle l'ordre respectif des variables d'intervalle correspondantes dans les deux séquences est le même.

Une variable de séquence permet donc d'établir des contraintes d'ordre sur les tâches, mais aussi des contraintes temporelles globales comme $\texttt{NoOverlap}(\pi)$, qui empêche deux tâches de la séquence de s'exécuter en parallèle, ce qui permet de modéliser des ressources de type machine (ressources dites disjonctives).

Il faut mentionner que de nombreux travaux cherchent à intégrer les approches de PLNE et de PPC au sein de méthodes exactes hybrides qui ont obtenus de très bons résultats pour des problèmes d'ordonnancement complexes [Lombardi and Milano, 2012].

Dans cette thèse, nous utilisons des solveurs "boîte noire" comme **CPLEX** et **CP Optimizer**. Ces solveurs implémentent respectivement les paradigmes arborescents de PLNE et de PPC, mais intègrent aussi de nombreuses heuristiques, astuces et autres diableries pour accélérer la recherche.

1.3.2 Heuristiques/Méta-heuristiques

Les méthodes exactes ne se prêtent pas souvent en pratique à la résolution de problèmes NP-difficiles de grande taille. C'est pourquoi les meilleurs résultats pour les problèmes d'ordonnancement comprenant un très grand nombre de tâches sont obtenus par l'emploi d'heuristiques ou méta-heuristiques. Contrairement aux méthodes exactes, une méthode heuristique ne fournit pas en général de preuves d'optimalité, et, même quant elle possède des propriétés de convergence, celle-ci n'est souvent que statistique. Un très grand nombre de ces heuristiques et en particulier de ces méta-heuristiques peuple la littérature. Nous présenterons ici deux méta-heuristiques parmi les plus reconnues, dont nous faisons l'usage au cours de cette thèse.

Il existe aussi des algorithmes dits "à garantie de performance" ou "algorithmes approchés", qui garantissent l'obtention d'une solution optimales à un facteur α de l'optimum dans le pire cas.

1.3.2.1 Algorithmes gloutons et heuristiques simples

En général, pour les problèmes d'ordonnancement qui nous intéressent, il est facile de trouver un ordonnancement faisable étant donnée une séquence mais la difficulté est de trouver la séquence optimale. Pour certains des problèmes les plus faciles d'ordonnancement, des équivalence avec des problèmes polynomiaux connus peuvent être établies, comme avec le problème de plus court chemin dans Brucker and Krämer [1996]. Pour d'autres de ces problèmes faciles, un **algorithme glouton** (qui prend des décisions successives sans jamais les remettre en cause) permet d'en trouver la solution optimale comme par exemple dans Lawler [1973]. Un cas particulier d'algorithme glouton est l'**algorithme de liste** qui ordonnance les tâches une par une en les prenant dans l'ordre défini par une liste fixée au préalable. Pour les problèmes plus difficiles, les algorithmes de liste ont parfois une garantie de performance, mais ne peuvent résoudre optimalement le problème en général [Kawaguchi and Kyan, 1986, Hall and Shmoys, 1992].

Les heuristiques "classiques" ne permettent la résolution que d'un nombre restreint de problèmes très similaires. Dans un effort de généralisation des méthodes heuristiques, de nombreuses "méta-heuristiques" ont vu le jour. Ces méthodes présentent une approche générale pour la résolution de problèmes d'optimisation, qui requiert peu de changements pour être appliquée à différents problèmes. Toutefois, pour être réellement compétitives, ces méthodes requièrent souvent une paramétrisation poussée et spécifique au problème étudié, qui n'est pas toujours triviale.

1.3.2.2 Algorithme tabou

La méthode tabou est une des méta-heuristiques parmi les plus simples mais qui pourtant a été très utilisée. Elle met en jeu un voisinage. Pour les problèmes d'optimisation combinatoire, le **voisinage** est une fonction $\mathcal{N} : (D_\chi) \to 2^{D_\chi}$ associant à chaque solution un ensemble de solutions voisines. Par extension on appelle

aussi $\mathcal{N}(\check{x})$ le voisinage de $\check{x}$. On suppose en général que si $\check{y} \in \mathcal{N}(\check{x})$, on à aussi $\check{x} \in \mathcal{N}(\check{y})$. La méthode tabou consiste à peu de choses près en une descente (une suite de mouvements d'une solution à une solution voisine améliorante, on parle aussi de recherche locale) dans un voisinage donné, couplée à un mécanisme de diversification par la "liste tabou". Son fonctionnement est décrit dans l'algorithme 1.1. En partant d'une solution initiale, on sélectionne dans son voisinage la meilleure solution non-tabou, qui devient la solution courante. La solution est ajoutée à la liste tabou ce qui permet de s'extraire de minimums locaux. La longueur de la liste tabou (le nombre d'itérations durant lesquelles une solution est conservée dans la liste) est un paramètre qui peut impacter la capacité de diversification de la méthode.

Algorithme 1.1 Algorithme tabou

Précondition : $\check{x}, l$

$X \longleftarrow \check{x}$
$X^* \longleftarrow X$
$L \longleftarrow [X]$
tant que Continue **faire**
  $V \longleftarrow \mathcal{N}(X)$
  $X \longleftarrow \underset{\forall X' \in V, X' \notin L}{\operatorname{argmin}} f(X')$
  $L \longleftarrow [X] + L[: l-1]$
  si $f(X) < f(X^*)$ **alors**
    $X^* \longleftarrow X$

L'efficacité du parcours de solution par la méthode tabou dépend aussi fortement du type de solution (de variable de décision) utilisé, puisqu'il définit aussi les voisinages qu'il est possible d'utiliser. Pour les problèmes d'ordonnancement que l'on considère, les solutions sous forme de séquence sont le plus souvent considérées, puisque, comme nous l'avons vu, elles sont associées à un ordonnancement semi-actif dominant, et permettent donc de réduire la taille de l'espace des solutions. Pour le problème du jobshop, Jain et al. [2000] présentent l'évolution chronologique des voisinages utilisés dans l'objectif de minimiser le makespan. Le voisinage le plus simple et intuitif consiste à échanger les positions de n'importe quelle paire de tâches adjacentes dans une séquence. Toutefois, le nombre de voisins que considère ce voisinage rend leurs évaluation laborieuse. Plusieurs résultats successifs caractérisant les mouvements qui sont susceptibles d'améliorer la solution courante ont permis de réduire grandement le nombre de voisins à évaluer menant à des voisinages dominants ne considérant qu'une fraction des voisins de l'approche naïve [Nowicki and Smutnicki, 1996].

1.3.2.3 Algorithme génétique

Les algorithmes génétiques (AG) figurent parmi les méta-heuristiques les plus utilisées et reconnues. Leur relative complexité est contrebalancée par la compréhension naturelle de leur efficacité. En effet, les AG s'inspirent du mécanisme de l'évolution naturelle. Nous sommes donc tentés d'utiliser ce mécanisme pour la résolution de nos problèmes. Les phases d'un algorithme génétique, dont certaines sont optionnelles, sont visibles dans l'algorithme 1.2.

Algorithme 1.2 Algorithme Génétique

```
1: Pop ⟵ INIT()
2: tant que Continue faire
3:    A, B ⟵ Select2(Pop)
4:    C ⟵ Croisement(A, B)
5:    C ⟵ Mutation(C)
6:    C ⟵ Repair(C)
7:    C ⟵ Education(C)
8:    Pop ⟵ Pop + [C]
9:    si |Pop| > maxPop alors
10:      Pop ⟵ Select(Pop)
```

Les algorithmes génétiques sont des algorithmes *à population*, c'est à dire qu'ils maintiennent un ensemble de solutions au cours de la résolution, par opposition aux algorithmes *à parcours* (comme la méthode tabou), qui ne considèrent qu'une seule solution courante. Le choix de la nature des "individus" (ou solutions) qui composent la population est un paramètre très important puisqu'il définit l'espace de recherche comme nous l'avons vu pour les algorithmes exacts. La première étape de l'algorithme est l'initialisation de la population ($INIT$). Celle-ci peut d'abord être composée d'individus aléatoires, ou bien d'individus de bonne qualité (dits "d'élite"), issus d'autres méthodes. Puis le processus générationnel est répété jusqu'à la fin de l'algorithme. Premièrement, deux individus A et B sont sélectionnés pour reproduction ($Select2$) dans la population Pop. La sélection peut être aléatoire, ou dépendre des performances relatives de la population. Ces individus sont croisés ($Croisement$) pour générer un nouvel individu "enfant". Les opérateurs de croisement doivent idéalement créer des enfants conservant la nature de la solution des parents. L'enfant est ensuite muté ($Mutation$), ce qui correspond généralement à un mouvement aléatoire dans un voisinage donné. Puis, dans certaines variantes de l'algorithme génétique, l'enfant est réparé ($Repair$) s'il n'est pas admissible. Dans le cas où il n'est pas réparé, on peut introduire une pénalité permettant de guider la population vers des régions admissibles de l'espace des solutions. Enfin, l'enfant est éduqué ($Educate$), souvent par une méthode de descente gloutonne, et il rejoint le reste de la population. La dernière phase survient lorsque la population devient trop nombreuse, c'est la phase de sélection ($Select$), qui sélectionne le sous-ensemble de la population à conserver, en fonction du score des individus, et parfois de leur diversité. On voit donc qu'il existe de nombreuses variations d'implémentation al-

gorithmes génétiques en fonction du problème d'intérêt. Le but étant de diversifier et d'intensifier suffisamment les solution au cours de la recherche pour s'extraire de minimums locaux et obtenir la meilleure solution possible.

De très nombreux travaux utilisent les algorithmes génétiques pour résoudre des problèmes d'ordonnancements [Çaliş and Bulkan, 2015, Cheng et al., 1996, 1999]. Là encore, se sont souvent des séquences qui sont choisies comme individus pour former la population.

Un opérateur très utilisé pour le croisement des séquences est le croisement à un point (*1-point crossover*). Ce dernier a l'avantage de conserver l'ordre des tâches provenant de chacun des parents. Ce croisement est présenté dans la figure 1.1. Un point X dans la première séquence est choisi, et l'enfant hérite de toute les tâches du premier parent jusqu'à ce point. Le reste de la séquence est construite en ajoutant toute les tâches manquantes dans l'ordre dans lequel elles apparaissent dans le deuxième parent.

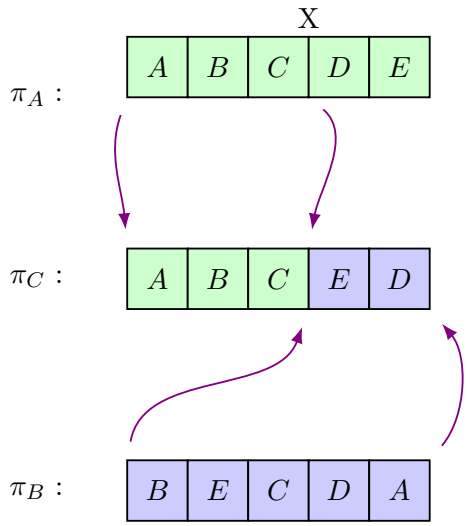


FIGURE 1.1 – Croisement à un point

Représenter les solutions sous forme de séquence permet souvent de garantir la faisabilité des individus. Mais parfois, comme quand le problème comprend des contraintes de précédence, il est quand même nécessaire de réparer les solutions obtenues comme nous le verrons dans le chapitre 3.

Chapitre 2

Ordonnancement sous incertitude

Sommaire

Nous avons décrit, dans le chapitre précédent, les problèmes d'ordonnancement déterministes ainsi que les méthodes permettant d'obtenir des solutions optimales ou de bonne qualité. Mais chacun sait qu'il est rare qu'un plan se déroule comme prévu, et une bonne solution en théorie ne l'est pas toujours (ni bonne, ni même admissible) en pratique. Dans Ben-Tal and Nemirovski [2000], il est montré que même de minuscules erreurs ou perturbations dans les données d'un problème peuvent mener à une solution présentant d'importantes violations des contraintes. C'est aussi le cas pour les problèmes d'ordonnancement. Par exemple, Goldratt [1997], mais aussi Hall and Posner [2004] montrent que les plannings déterministes peuvent donner de mauvaises performances lors de leurs exécutions. En conséquence, de nombreux travaux ont porté sur l'étude de l'optimisation sous incertitude et y compris l'ordonnancement sous incertitude [Pinedo and Schrage, 1982, Möhring et al., 1984, Daniels and Kouvelis, 1995, Aytug et al., 2005, Herroelen and Leus, 2005, Li and Ierapetritou, 2008, Davari and Demeulemeester, 2019].

Dans ce chapitre, nous introduisons les concepts de l'optimisation sous incertitude (et en particulier de l'ordonnancement sous incertitude) pertinents pour présenter le contenu de cette thèse. Nous nous intéressons à la modélisation des problèmes sous incertitude et leurs méthodes de résolution. Nous présentons aussi certaines structures qui peuvent être utilisées pour la résolution de problèmes sous incertitude.

2.1 Modéliser des problèmes sujets à l'incertitude

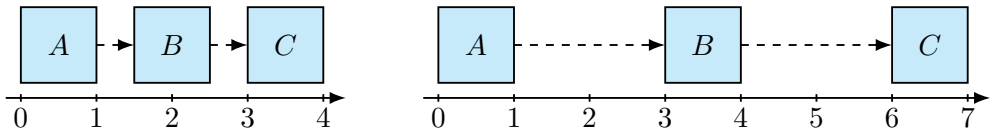
Pour pallier la méconnaissance des problèmes, il faut en premier lieu déterminer quelles informations sont méconnues, et dans quelles mesures. Toute les composantes d'un problème peuvent a priori être affectées par la méconnaissance ; non seulement les valeurs numériques des données des problèmes, mais aussi le nombre de contraintes, la nature des contraintes considérées, le critère d'intérêt, etc ...

Dans Billaut et al. [2013], les auteurs décrivent trois catégories de méconnaissance : la méconnaissance des données factuelles du problème (par exemple, due à des erreurs de mesures), la méconnaissance du contexte futur lors de la mise en oeuvre de la décision, et la méconnaissance de facteurs plus subjectifs de ce qui constitue une décision admissible ou préférée. Ils dénomment respectivement "instance","scénarios", et " interprétations" les valeurs possibles (ou réalisations) du problème pour ces trois catégories. Toutefois, ces trois catégories ne sont pas parfaitement distinctes. Dans cette thèse, nous nous concentrerons sur des méconnaissances se rapprochant du second type (des méconnaissances sur la date à laquelle une tâche peut être commencée, ou sur le temps nécessaire à l'exécution d'une tâche) que nous nommerons "incertitudes", et nous appellerons **scénarios** les différentes réalisations a posteriori de l'incertitude.

Nous nommons "instance" d'un problème modélisé un ensemble d'informations sur les données. Pour un problème déterministe, ce sont toutes les valeurs numériques des données. Pour un problème comprenant de l'incertitude, une instance comprend les valeur numériques de certains paramètres, et possiblement d'autres informations décrivant des connaissances imparfaites d'autres données, selon la modélisation (par exemple une distribution de probabilité d'une valeur, un ensemble de valeurs possibles...). Une fois l'incertitude levée, le scénario révèle une instance déterministe.

2.1.1 Niveau de modélisation de l'incertitude

Pour pallier les conséquences négatives de cette méconnaissance, lorsque l'on modélise un problème comportant des composantes y étant sensibles, il est possible de l'expliciter pour la prendre en compte lors de la recherche de solution. Cette modélisation peut prendre de nombreuses formes, et dépend principalement des informations qui sont ou ne sont pas disponibles, qu'il est possible d'obtenir, et du



(a) Grande incertitude : $p_i \in [1; 1.5], \forall i$ (b) Grande Imprécision : $p_i \in [1; 3], \forall i$

FIGURE 2.1 – Deux ordonnancements conséquence d'une modélisation très incertaine ou très imprécise de la durée d'exécution des tâches p_i.

critère que l'on souhaite prendre en compte pour évaluer une décision étant donnée cette méconnaissance. Puisque l'on peut rarement avoir une certitude absolue, cette modélisation doit faire des hypothèses plus ou moins prudentes dans un compromis entre **incertitude** et **imprécision**. Plus l'on veut être précis dans la description du problème, plus il sera possible de trouver de bonnes solutions théoriques, mais moins il est certain que la solution trouvée corresponde au problème effectif a posteriori. A l'inverse, une description très large et imprécise du problème permettra le calcul d'une solution qui s'adaptera bien au problème, souvent au coût de faibles performances. La figure 2.1 illustre ce phénomène pour un problème naïf d'ordonnancement dans lequel trois tâches doivent être exécutées sur la même machine et terminées le plus rapidement possible. Une date de début doit être fixée pour chacune d'elles. La durée d'exécution de ces tâches devrait être de 1 heure, mais cette durée est incertaine. Pour s'assurer de la faisabilité de la solution, il faut attribuer une marge suffisante entre chaque tâche pour garantir que chaque tâche sera terminée lorsque la suivante devra commencer (marges représentées par des flèches pointillées). Ainsi, plus l'on prévoit de marge, plus il est probable que la solution soit faisable, mais plus elle risque d'être mauvaise (du point de vue de la date de fin par exemple). Notons que malgré cette distinction entre incertitude et imprécision, nous perpétuons l'abus usuel et nommons incertitude toute méconnaissance.

On peut distinguer plusieurs niveaux de connaissance lorsque l'on modélise un problème sujet à l'incertitude.
Aux niveaux de connaissances les plus faibles, il est possible que certains aspects du problème soient méconnus, mais on ne sait rien de plus à leur sujet, ou du moins, on ne le modélise pas. Dans ce cas, il est toujours possible de tenter de se prémunir contre des effets néfastes d'imprévus. Par exemple, en utilisant des contraintes supplémentaires ou un objectif différent. On va chercher à trouver soit des solutions très "stables", c'est-à-dire insensibles aux variations des paramètres [Sotskov et al., 1998], soit des solutions flexibles, capables de s'adapter à plusieurs types d'imprévus, souvent à l'aide de l'expertise humaine [Artigues et al., 2005]. C'est à ce niveau de méconnaissance que l'on se placera dans la partie III. Il est aussi possible d'anticiper la mauvaise adéquation d'une solution déterministe en prévoyant de la modifier lorsque plus d'informations seront révélées. C'est le principe des *algorithmes prédictifs-réactifs* (voir section 2.2.2).

Les niveaux les mieux informés modélisent les réalisations possibles et y assignent une mesure de plausibilité d'occurrence. La modélisation "floue", issue de la théorie des possibilités, établit pour différentes valeurs leur niveau de possibilité, souvent déterminé par des experts [Herroelen and Leus, 2005]. Dubois et al. [2003] décrivent une utilisation de la théorie des possibilités pour la modélisation de problèmes d'optimisation. Elle exprime l'incertitude sur les données par plusieurs intervalles imbriqués dont chacun correspond à un niveau de possibilité de réalisation, par exemple : "tout à fait possible", "possible", et "impossible".
Une des modélisations la plus informée assigne à chaque réalisation possible une probabilité.

Dans un niveau intermédiaire, moins informé que les modélisations probabilistes ou même possibilistes, on connaît certaines ou toutes les options qui peuvent survenir, mais sans information a priori sur les réalisations les plus probables. Ces approches sont souvent regroupées sous le nom d'approches par scénarios, puisqu'elles décrivent un ensemble des scénarios de réalisation possibles. L'approche la plus courante consiste à donner, pour chaque paramètre du problème, un intervalle continu de valeurs possibles et parfois une valeur nominale [Ben-Tal and Nemirovski, 2002]. Cependant, pour certains critères de robustesse, Bertsimas and Sim [2004] montrent que la solution obtenue peut être trop prudente. Ce qui arrive en effet si l'on veut se prémunir contre le pire scénario (voir section suivante), qui surviendrait lorsque **tous** les paramètres prennent leur pire valeur, alors que ce scénario ne survient virtuellement jamais en pratique. Pour mitiger le pessimisme des solutions obtenues, on attribue en général un *budget* à l'incertitude. Habituellement, ce budget restreint l'ensemble des scénarios possibles soit en limitant la quantité totale de déviation par rapport à un scénario nominal, soit en limitant le nombre de paramètres qui peuvent s'en écarter de façon discrète [Bachtler et al., 2020], soit encore en limitant la quantité de déviation par une fonction continue sur l'ensemble des paramètres (c'est le cas par exemple pour la modélisation de l'incertitude par ellipsoïdes [Ben-Tal and Nemirovski, 2000]).

L'approche par scénarios que nous considérons dans la partie II utilise une modélisation discrète de l'incertitude. Elle consiste à proposer un ensemble de réalisations possibles de tout l'ensemble des paramètres à la fois. Ces scénarios peuvent être obtenus par échantillonnage aléatoire sur des occurrences passées du problème d'intérêt, auquel cas ils tendent à approximer la distribution de probabilité sous-jacente du problème, y compris quand les variables aléatoires ne sont pas indépendantes. En effet, le nombre d'occurrences de chaque scénario sur le nombre total de scénario tend vers la valeur de la probabilité du scénario si l'échantillonnage est suffisamment grand. Les scénarios peuvent aussi intégrer la notion de budget d'incertitude en choisissant des scénarios ne s'écartant pas trop d'un scénario nominal.

Bien que cela dépasse le cadre de cette thèse, on mentionnera aussi le paradigme plus large de l'optimisation distributionnellement robuste [Rahimian and Mehrotra,

2022], qui permet de modéliser l'incertitude sur la modélisation. Par exemple, si une distribution de probabilité est estimée par échantillonnage, celle-ci est elle même sujette aux erreurs et biais de mesure. Un ensemble de distributions peut donc prétendre à représenter l'incertitude. Les différent niveaux de modélisation s'appliquent de nouveau : on peut connaître l'ensemble des distributions possibles sans les distinguer, ou bien connaître la probabilité associée à chaque distribution etc.

2.1.2 Critères d'optimisation sous incertitude

Ayant modélisé l'incertitude à laquelle est soumis le problème, il faut adapter la définition des critères d'optimisation. En effet, une solution préférée dans un scénario peut ne pas l'être dans un autre. Il faut donc définir une nouvelle façon de comparer plusieurs solutions dans le contexte de l'optimisation sous incertitude. En fonction du niveau de connaissance modélisé, plusieurs critères sont considérés.

Dans les cas où l'on modélise l'incertitude par un ensemble de scénarios, puisque l'on ne connaît pas leur probabilité d'occurrence, un décideur réticent au risque à souvent recours à l'*optimisation robuste*, qui consiste à se prémunir contre les pires scénario modélisés. L'approche la plus robuste du "minmax" consiste à chercher la solution x^* ayant le meilleur score pour un critère déterministe f dans le pire scénario s :

$$x^* = \underset{x}{\operatorname{argmin}} \max_{s} f(x, s)$$

Nous l'avons vu plus tôt, cette approche peut proposer des solutions trop pessimistes, puisqu'elles doivent s'accommoder de scénarios "pire-cas", très improbables en réalité, au détriment de gains possibles dans une proportion de scénarios plus probables. D'autres critères permettent de mitiger ce pessimisme. Par exemple le critère du "minmax-regret" vise a minimiser la plus grande différence entre le score obtenu par une solution sur un scenario et celui obtenu par la meilleure solution pour ce scenario :

$$x^* = \underset{x}{\operatorname{argmin}} \max_{s} \left(f(x, s) - \min_{y} f(y, s) \right)$$

Ce critère tend à fournir des solutions de meilleure qualité dans les "bons" scénarios, et de pire qualité dans les "mauvais", par rapport au critère minmax. C'est un compromis entre pessimisme (dont le minmax est le plus extrême représentant) et l'optimisme.
L'équivalent optimiste du critère minmax est le critère "minmin" , qui cherche la meilleure solutions dans le scenario le plus favorable :

$$x^* = \underset{x}{\operatorname{argmin}} \min_{s} f(x, s)$$

Bien que symétrique du minmax, ce critère est bien moins utilisé, ce qui est justifié par l'aversion au risque humaine. Le critère de Hurwicz constitue une approche directe de compromis entre optimisme et pessimisme dans laquelle un paramètre

$\alpha \in [0;1]$ détermine le niveau de pessimisme :

$$x^* = \underset{x}{\operatorname{argmin}} \left(\alpha \min_s f(x,s) + (1-\alpha) \max_s f(x,s) \right)$$

Dans le cas de scénarios discrets échantillonnés, et si l'on sait pour chaque scénario sa probabilité d'occurrence $\mathbb{P}(s)$, l'approche la plus complète est l'*optimisation stochastique*, qui consiste à optimiser l'espérance du score, soit :

$$x^* = \underset{x}{\operatorname{argmin}} \sum_s \mathbb{P}(s) f(x,s)$$

Toutefois, l'aversion au risque ne disparaît pas, et il est courant que l'on souhaite quand même se prémunir contre les mauvais scénarios. L'approche CVAR (*Conditional value at risk* [Rockafellar and Uryasev, 2000]) optimise l'espérance du critère sur une proportion α des pires scénarios. De façon similaire il est possible d'optimiser le score obtenu sur le meilleurs scenarios parmis les α pires scenarios, soit le score obtenu sur le α quantile des pires scenarios. En fonction du paramètre α, cet objectif offre un autre compromis entre le critère minmin et minmax.

Si l'on connaît la probabilité d'occurrence des scénarios, et que la faisabilité des solutions n'est pas garantie, ajouter des contraintes probabiliste de la forme

$$\mathbb{P}(Ax \leq b) \geq \alpha$$

permet de garantir la faisabilité des solutions pour un pourcentage α des réalisations.

Comme indiqué précédemment, lorsque l'incertitude n'est pas modélisée, il est aussi possible d'anticiper l'incertitude sur les données du problème. Pour une solution calculée à partir d'un modèle déterministe, une *étude de sensibilité* [Sotskov et al., 1998] permet d'établir quelle variation minimale des données d'entrée impacterait la qualité de la solution obtenue. On peut aussi choisir de maximiser la flexibilité d'une solution [Artigues et al., 2005]. C'est-à-dire que l'on va proposer non pas une solution unique mais un ensemble de solutions au décideur, qui pourra ensuite choisir la plus pertinente en fonction du scénario réalisé.
Dans la partie II de cette thèse, nous considérons une modélisation de l'incertitude par un ensemble de scénarios discrets et considérons des objectifs d'optimisation stochastiques et robustes. De plus, étant donné que nous ne considérons lors de la résolution qu'un nombre fini de scénarios "d'entraînement" échantillonnés, il sera naturel de se poser la question de la qualité des solutions produites sur d'autres scénarios, nous évaluons donc les solutions aussi sur des scénarios échantillons de "test".

2.1.3 Modèles d'information

Une solution flexible n'est utile que si de l'information pertinente est obtenue avant la fin de l'exécution d'une solution. C'est le cas dans les problèmes de *décision*

séquentielle comme les problèmes d'ordonnancement, où il est possible d'obtenir de l'information au cours de la mise en oeuvre de la solution.

Les informations obtenues avant toute décision peuvent être intégrées au modèle, on ne s'intéresse donc dans le modèle d'information qu'à celles qui sont portées à notre connaissance après une décision partielle.

Dans un contexte d'incertitude, il semble évident qu'obtenir de l'information supplémentaire ne peut pas mener à des décisions de pire qualité (du moins en moyenne, et si l'information obtenue est véridique). Obtenir de l'information revient à diminuer l'incertitude. Pourtant, même quand de l'information est disponible au cours de l'exécution, celle-ci n'est pas toujours utilisée, ni prise en compte dans le modèle. Outre le possible caractère fastidieux d'une replanification, d'autres considérations justifient parfois de ne pas complexifier le modèle en considérant les nouvelles informations. Premièrement, les décideurs rechignent à modifier trop et trop souvent une solution. On parle de *stabilité*, ou au contraire de *l'instabilité* du planning en ordonnancement. Secondement, pour de nombreux problèmes, obtenir de l'information n'est pas gratuit mais a un coût.

Si l'on considère possible d'obtenir de l'information, il faut donc prendre en compte ce coût dans le processus d'optimisation, et le comparer à la *valeur de l'information*. Plusieurs métriques peuvent mesurer cette valeur, mais celle ci dépend de la capacité de l'information à suggérer une modification de la solution, et de la mesure dans laquelle la solution est améliorée par la connaissance apportée par l'information [Russel and Norvig, 2010]. On peut ensuite définir quelle information est *accessible* à chaque instant, et le cas échéant, quel est son coût. Des décisions supplémentaires peuvent refléter des actions spécifiques nécessaires à l'obtention des informations, ou bien l'information peut être révélée en conséquence des précédentes prises de décision. Les moments auquel les nouvelles informations sont considérées peuvent être fixés à des temps donnés, ou dépendre des décisions prises. Ces moments sont appelés **moments de décision** (*décision moment*), et sont les instants auquel de nouvelles décisions sont prises (puisque, a priori, il n'y a pas d'intérêt à prendre de décision en l'absence de nouvelles données).

Dans cette thèse nous nous intéresserons à des problèmes d'ordonnancements dans lesquels l'incertitude porte sur la date de disponibilité et la durée des tâches. Une modélisation pour l'acquisition d'information consiste à révéler toute information sur l'état présent d'une tâche. Ainsi, on sait à chaque temps t si une tâche est disponible ou non, et si elle est terminée, ou pas encore. Le tout sans qu'il n'y ait de coût associé. Cette hypothèse semble très raisonnable pour un décideur pouvant suivre l'exécution d'un ordonnancement et constater l'état d'avancement du planning.

D'autres modèles d'information sont plus généreux. Dans Portoleau et al. [2020], les auteurs considèrent un modèle de prise d'information pour un problème dans lequel les dates de livraison sont incertaines et modélisées par un intervalle. Ils considèrent qu'il est possible d'obtenir des informations réponses à des questions de type "La date de livraison de la tâche t sera-t-elle plus grande ou inférieure à x ?"

lorsque cette date x est suffisamment imminente. Ils supposent donc l'accès à des informations portant sur le futur. Dans leur modèle, un paramètre limite le nombre d'informations qu'il est possible d'utiliser, cela permet entre autres de modéliser un coût à l'obtention de l'information. Les informations ainsi obtenues guident l'exécution des tâches à chacun des moments de décision, sélectionnant ainsi une solution particulière parmi un ensemble de solutions pré-calculées.

On commence donc à comprendre l'intérêt des solutions flexibles : en remettant une partie des décisions à plus tard, dans un contexte où plus d'information est révélée, il est possible d'obtenir une solution finale de meilleure qualité (c'est l'approche de moindre engagement). Nous verrons toutefois, que pour des problèmes difficiles, la résolution en temps réel est impossible. Il s'agit donc de faire un compromis en proposant une solution partielle telle que les décisions restantes soient suffisamment faciles, et permettent de tirer le plus possible parti des informations qui seront disponibles.

2.1.4 L'incertitude spécifique aux problèmes d'ordonnancement

Les techniques de modélisation précédentes s'appliquent généralement aux problèmes d'optimisation sous incertitude, mais nous nous intéressons spécifiquement aux problèmes d'ordonnancement. Nous l'avons vu, ces problèmes sont particulièrement sensibles à l'incertitude, ce qui motive sa prise en compte dans les modèles traduisant ces problèmes.

Pour les problèmes d'ordonnancement, on peut distinguer plusieurs sources d'incertitude. Dans la partie II nous nous intéresserons exclusivement à l'incertitude la plus classique : celle des valeurs des données du problème. Toutefois, d'autres aléas peuvent perturber l'exécution d'un ordonnancement, et sont régulièrement pris en compte. On trouve le cas des incertitudes portant sur les ressources (pannes de machines, absence d'opérateurs), mais aussi des incertitudes sur les éléments même du problème, par exemple lorsqu'une tâche imprévue doit être ajoutée en urgence dans la plan, des modifications des contraintes (e.g. l'ajout ou retrait de contrainte de précédence), ou des modifications concernant la préférence du décideur. Les méthodes présentées en partie III peuvent potentiellement prendre en compte ce genre d'incertitudes plus générales.

Nous avons déjà constaté la variété des problèmes d'ordonnancement déterministes. Si l'on y ajoute le nombre de paramètres qui peuvent être incertains, et le grand nombre de modélisations possibles, on se rend compte de la grande difficulté à répertorier l'ensemble des problèmes d'ordonnancement sous incertitude. Dans Gourgand et al. [2000], les auteurs proposent une extension de la description à trois champs de Graham [1966] pour permettre la description de certains problèmes d'ordonnancement sous incertitude. Par exemple ils étendent le champ β (voir section 1.1.2) pour permettre d'indiquer la nature de l'incertitude sur les données (e.g

$p_{i,j} \sim exp(\mu_{i,j})$ si la durée des tâches suit une distribution exponentielle de paramètre μ).

Cette multitude de problèmes étudiés donne lieu à autant de façons de les résoudre. Toutefois, des familles de méthode de résolution se dégagent. Nous en présentons une partie dans la section suivante

2.2 Méthodes de résolution

Dès lors qu'un problème d'optimisation sous incertitude est modélisé, avec les considérations développées plus tôt, les méthodes utilisées pour prendre les décisions dans les cas sous incertitude sont les mêmes méthodes exactes, heuristiques et méta-heuristiques que dans le cas déterministe, définies dans le chapitre précédent (section 1.3). La principale différence survient lorsqu'il est possible de diviser le processus de décision pour prendre en compte des informations révélées pour la suite des décisions.

Billaut et al. [2013] identifient trois phases dans la résolution de problèmes d'ordonnancement sous incertitude, représentées chronologiquement dans la figure 2.2. La phase 0 correspond à la modélisation du problème, de ses aspects déterministes et incertains, de ses critères, et de son modèle d'information. Une fois le problème modélisé, on entre dans la phase 1 de la résolution, aussi appelée phase proactive, *offline*, ou statique. Durant cette phase, on suppose généralement que le coût du temps de calcul est faible, c'est-à-dire que l'on dispose d'un temps presque illimité de calcul et que la complexité des méthodes mises en oeuvre n'est pas un facteur bloquant. Par contre, l'incertitude bat son plein. A l'issue de la phase 1, une décision partielle est fixée (parfois dénommée décisions *here & now*), qui sera complétée dans la phase 2, aussi appelée phase réactive, "online" ou dynamique. Cette dernière phase est composée des moments de décisions définis dans le modèle (décisions *wait & see*). Durant cette phase, on considère que le coût du temps de calcul est élevé, on préférera donc des méthodes dont la complexité est faible. Par contre, l'incertitude est partiellement levée. Au fur et à mesure de cette phase, la décision partielle est complétée jusqu'à l'obtention de la solution finale.
Les méthodes de résolution se différencient ensuite par leurs stratégies de mise en oeuvre des phases 1 et 2 (en termes de quelles décisions sont prises à quel moment, et de méthodes utilisées pour la prise de décision), qui dépend bien sûr de la modélisation en phase 0. Elles sont habituellement regroupées selon trois catégories : les approches proactives, les approches réactives, et les approches proactives-réactives, que nous illustrons par la suite. Pour une description plus exhaustive, nous référons le lecteur aux travaux de Davenport and Beck [2000], Herroelen and Leus [2004b, 2005], Van de Vonder et al. [2007].

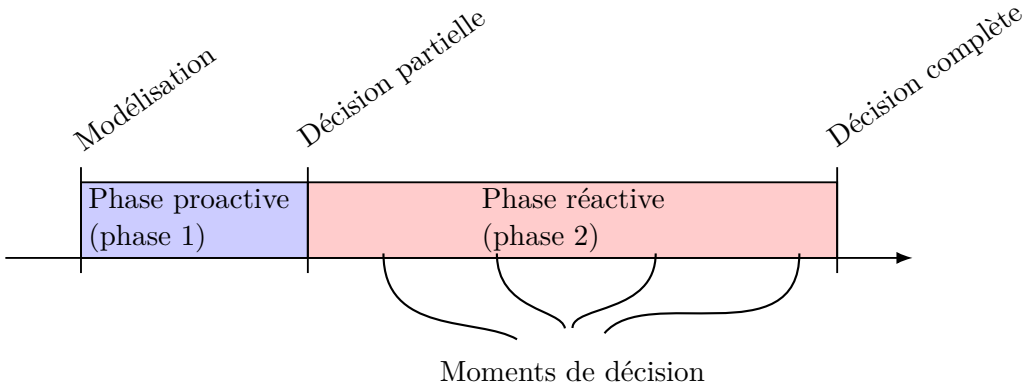


FIGURE 2.2 – Chronologie de l'ordonnancement sous incertitude

2.2.1 Approches proactives

On appelle **approches proactives** les stratégies de résolution qui se concentrent presque exclusivement sur la phase 1, au détriment de la phase 2. Ces approches se prémunissent contre l'incertitude en calculant une solution susceptible de bien se comporter, en termes de faisabilité et de qualité, quelque soit le scénario qui se réalise. On retrouve dans cette catégorie les méthodes d'optimisation robuste et stochastiques classiques des problèmes pour lesquels aucune information supplémentaire n'est révélée. Parmi les méthodes exactes, Herroelen and Leus [2004a] proposent une modélisation PLNE pour se prémunir contre les scénarios dans lesquels une activité au plus dévie de sa durée d'exécution nominale. Van de Vonder et al. [2008] proposent plusieurs heuristiques et méta-heuristiques pour générer des solutions robustes, notamment en ajoutant des marges d'exécutions aux tâches en fonctions de l'incertitude les concernant. Jensen [2001] utilise un algorithme génétique pour générer des solutions robustes pour le problème du jobshop confronté à l'incertitude sur la disponibilité des machines. Comme nous l'avons mentionné plus haut, bien qu'il semble que les approches proactives bénéficieraient de l'utilisation de l'information révélée durant la phase dynamique, celles-ci ont l'avantage pratique de proposer une solution robuste qui, sauf grandes perturbations, pourra être exécutée avec peu ou pas de modifications. L'approche proactive offre donc des solutions très stables. Toutefois, il est extrêmement rare qu'une méthode soit purement proactive en pratique. En effet, l'approche qui consiste à décider initialement de la date d'exécution de tâches et qui refuse de la modifier d'un iota est considérablement naïve. Dans un cas (lorsqu'une tâche finit en avance) parce que, comme nous l'avons vu, il est souvent contre-productif d'attendre une date de début si la prochaine tâche peut être lancée immédiatement, et dans l'autre cas (lorsqu'une tâche termine en retard), parce que lancer la prochaine tâche dont la date de lancement été manquée est préférable à déclarer l'ordonnancement impossible. Dans une application réelle, on effectue donc au minimum dans la phase 2 la procédure de "décalage à droite" qui retarde le lancement des prochaines tâches en cas de retard.

Dans cette optique, on considère en général qu'une décision de séquence dans la phase proactive associée à un ordonnancement au plus tôt dans la phase réactive est une approche "purement" proactive, alors qu'un traitement (certes minimal), est implémenté dans la phase réactive.

2.2.2 Approches réactives

A l'inverse des approches proactives, les stratégies de résolution qui se concentrent sur la phase 2 sont les **approches réactives**. Ces approches n'essaient pas de se prémunir à l'avance contre l'incertitude mais plutôt d'y réagir une fois l'incertitude levée. On nomme parfois la stratégie employée lors de la phase 2 une *politique*. Celle-ci décrit les règles qui décident l'action à effectuer à chaque moment de décision. Les approches purement dynamiques utilisent l'information disponible à chaque moment de décision pour choisir quelles tâches doivent être démarrées. Ces approches utilisent régulièrement des heuristiques simples comme les algorithmes de listes. On trouve, en sus des approches purement dynamiques, les approches dites "prédictives-réactives". Celles-ci proposent, à l'issue de la première phase, un ordonnancement prédictif, calculé sans considération pour la robustesse. Cet ordonnancement sert néanmoins de base aux managers pour préparer l'exécution des tâches, communiquer avec les différentes parties prenantes, et visualiser le travail à accomplir [Van de Vonder et al., 2007]. Lorsqu'un aléas survient, il faut alors "réparer" la solution courante. Vieira et al. [2003] décrivent les différents mode de réparation. Cela peut consister en des réparations élémentaires visant à restaurer la faisabilité (comme le "décalage à droite"), en un ré-ordonnancement partiel, ou à l'extrême en le ré-ordonnancement complet des tâches prenant en compte la nouvelle information. La fréquence et l'ampleur des réparations contribuent à l'instabilité de l'ordonnancement, on évite donc en général le ré-ordonnancement complet, d'autant plus que sa complexité est rédhibitoire dans cette phase.
Que la procédure de résolution mette en jeu une solution prédictive dès la fin de la phase proactive ne signifie pas qu'une décision complète à été prise. En effet, puisque cette "décision" est toujours soumise à modification, ce n'est qu'une indication de la décision qui sera prise dans le cas nominal lors des moments de décision de la phase réactive.

2.2.3 Approches proactives-réactives

Enfin, les stratégies "proactives-réactives" font usage des deux phases, proactives et réactives. Elles calculent donc dans la phase proactive une solution partielle (robuste par sa flexibilité), et définissent une politique pour décider de la marche à suivre lors des moments de décision de la phase réactive. On retrouve bien sur des extensions des approches (simplement proactives ou réactives) développées précédemment. Par exemple, Van de Vonder et al. [2007] comparent différentes combi-

naisons de méthodes proactives et réactive. Ils utilisent une méthode proactive pour générer un ordonnancement prédictif robuste, et une approche prédictive-reactive pour définir la politique de réaction.

Bien que l'approche proactive-réactive domine les approches réactives et les approches proactives, Davari and Demeulemeester [2019] déplorent que la plupart des stratégies proposées ne tirent pas pleinement partie des possibilités qu'offre cette approche. Premièrement, les stratégies proposées n'intègrent pas leurs partie proactive et réactive l'une avec l'autre. En effet, l'optimalité d'une solution (même partielle) calculée lors de la phase proactive dépend de la politique réactive qui sera employée, et inversement, la politique employée doit dépendre de la méthode de calcul de la phase proactive. Deuxièmement, les stratégies proposées ne prennent pas suffisamment en compte la stabilité de l'ordonnancement, alors que l'approche proactive-réactive le permet. Les auteurs proposent par la suite une stratégie de résolution pour le problème du RCPSP stochastique. Celle-ci calcule, lors de la phase proactive, une arbre de décision optimal étant donné une politique de décision lors de la phase réactive, qui prend en compte un coût associé à l'instabilité.

D'autres approches intègrent les phases proactives et réactives. La méthode "Just-in-case" de Drummond et al. [1994] anticipe les perturbations les plus probables pour un ordonnancement issu de la phase proactive et prévoit quel recours y apporter dans la phase réactive. Dans Portoleau et al. [2020], les auteurs calculent aussi un arbre de décision optimal pour la minimisation du retard maximal, définissant quelles informations obtenir et quelles décision prendre à chaque moment de décision, en fonction des informations recueillies. Dans van den Akker et al. [2018], le principe de l'optimisation robuste réparable est appliqué à un problème d'ordonnancement. Les auteurs cherchent à maximiser le nombre moyen de tâches exécutées en calculant une solution lors de la phase réactive, qui pourra être réparée lors de la phase réactive. Ces approches font écho à l'optimisation robuste ajustable [Ben-Tal et al., 2004, Yanıkoğlu et al., 2019] dans laquelle le modèle prévoit certaines variables de décision qui peuvent être modifiées lorsque de nouvelles informations sont révélées. Par exemple Bruni et al. [2018] proposent un modèle de PLNE d'optimisation stochastique à deux étapes, allouant des ressources aux tâches dans la première étape et décidant de l'ordonnancement lors de la deuxième étape.

La famille de méthodes proactives-réactives que nous proposons dans cette thèse produit, lors de la phase proactive, un grand ensemble de solutions. Cet ensemble de solutions est couplé à une politique qui décide, lors de la phase réactive, la solution qui est mise en oeuvre. A la différence des approches proactives-réactives présentées précédemment, qui prévoient explicitement pour chaque scénario ou ensemble de scénarios la solution associée, les méthodes auxquelles nous nous intéressons dans cette thèse produisent une représentation **compacte** d'un l'ensemble des solutions. Une représentation compacte permet de représenter plus de solutions et ainsi conserver un maximum de flexibilité. Plusieurs façons de représenter ces ensembles de solutions se trouvent dans la littérature. Par exemple, Erschler and Roubellat [1989] proposent, à l'issue de la phase proactive, de définir un ensemble

de solutions à l'aide d'une décision partielle particulière sur l'ordre des tâches. Dans Cheref et al. [2014], un modèle PLNE intègre cette phase proactive avec une politique réactive.
De façon similaire, Baptiste and Favrel [1993] proposent de considérer un ensemble d'ordonnancements équivalents permettant d'éviter les ré-ordonnancements trop fréquents lors de la phase réactive. Dans Aloulou and Portmann [2005], Aloulou and Artigues [2010], les auteurs caractérisent un ensemble de solutions à l'aide d'un ordre partiel. Une fonction de priorité choisit quelle tâche lancer parmi les tâches disponibles lors de la phase réactive.

Nous nous intéressons aux différentes façon de représenter des ensembles de solutions de façon compacte. La section suivante présente ces représentations plus en détails.

2.3 Structures de représentation d'ensembles de solutions

Les structures permettant de représenter des ensembles de solution sont souvent utilisées dans les problèmes d'ordonnancement. Non seulement dans le cadre d'approches proactives, mais aussi en tant qu'outil pour résoudre des problèmes déterministes.

Dans cette section, nous présentons certaines de ces structures, leurs utilisations dans la littérature, et nous ferons allusion à la façon dont ces structures *représentent* des ensembles des solution (ce concept sera développé dans la partie III) .

2.3.1 Groupes d'opérations permutables

Les séquences de groupes d'opérations permutables sont initialement proposés par Demmou [1977] et indépendamment par Wu et al. [1999]. Les deux travaux partent du constat qu'une décision partielle adéquate détermine la plus grande partie de la qualité de la solution finale, et que la flexibilité résultante permet de se prémunir contre les effets des aléas lors de la phase dynamique.

La structure des groupes d'opérations permutables est définie pour le problème à une machine et pour le problème du jobshop par une décision partielle particulière qui caractérise une partition ordonnée des tâches à exécuter sur chaque machine.

Définition 2.3.1.1. *Une **séquence de groupes d'opérations permutables** π^m pour une machine m est une séquence d'ensembles d'opérations G_i^m :*

$$\pi^m = [G_1^m, G_2^m \dots G_k^m], \quad t.q. \bigcup_{i=1\dots k} G_i^m = N \ et \ G_i^m \cap G_j^m = \emptyset \ \forall (i,j)$$

La décision restreint le domaine des séquences à celles dans lesquelles toutes les tâches de G_1^m sont exécutées avant celles de G_2^m, elles-même exécutées avant celles de G_3^m, etc... Mais l'ordre des tâches au sein des groupes n'est pas fixé. Ainsi, le

nombre de permutations de l'ensemble des tâches qu'il est possible d'atteindre pour chaque machine est $\prod_{m \in 1...M} \prod_{i \in 1...k} |G_i^m|!$. On comprend donc que la taille des groupes a une grande influence sur la flexibilité de la décision partielle. A un extrême, toutes les tâches sont dans un même et unique groupe, auquel cas la flexibilité est maximale, en fait, aucune décision n'est prise. A l'autre extrême, chaque groupe contient une unique tâche, auquel cas une unique séquence par machine est compatible avec la décision partielle, et la seule flexibilité restante réside dans les dates de début des tâches.

Les groupes d'opérations permutables ont été intégrés dans une approche proactive et implémentée industriellement par l'intermédiaire du logiciel ORDO [Billaut and Roubellat, 1996]. Dans Artigues et al. [2016] les auteurs proposent un modèle de PLNE et une méthode tabou pour intégrer la phase proactive de recherche de séquence de groupe avec la phase réactive. Cheref et al. [2016b] utilisent les séquences de groupe d'opérations permutables pour un problème d'ordonnancement et transport sous incertitude.

Le chapitre 4 s'intéresse particulièrement aux séquences de groupe d'opérations permutables pour l'ordonnancement sous incertitude.

2.3.2 Ordres partiels

Le coeur de la difficulté des problèmes d'ordonnancement provient du nombre limité des ressources, qui nécessite parfois d'arbitrer quelles tâches seront prioritaires sur quelles autres tâches concurrentes. Cet arbitrage revient à l'ajout de contraintes de précédence supplémentaires entre des paires de tâches. Un ensemble de contraintes de précédence définit un **ordre partiel** sur les tâches. On remarque que les séquences de groupes d'opérations permutables définissant un ordre partiel particulier, les ordres partiels généralisent les séquences de groupes d'opérations permutables.

La décision partielle que constitue un ordre partiel est le plus naturellement implémentée lorsque l'on considère les variables de décision $x_{i,j}$ dont la valeur est 1 si la tâche i est ordonnancée avant la tâche j. Pour cette raison on représente parfois cette décision par une matrice, mais dans cette thèse nous utiliserons la représentation inspirée des contraintes de précédence, un ordre partiel π est donc décrit par un ensemble de contraintes de précédence : $(i, j) \in \pi$ signifiant que i est avant j. Un ordre partiel ainsi décrit représente l'ensemble des ordonnancements admissibles pour ces contraintes de précédence.

Un ordre partiel peut constituer une décision partielle dans le cadre d'approches proactives. Par exemple dans Policella et al. [2007, 2009], les auteurs décrivent une approche proactive-réactive pour le RCPSP. Ils obtiennent un ordre partiel par l'ajout de contraintes de précédence, permettant de caractériser un ensemble de solutions respectant les contraintes de ressources. Dans Aloulou and Portmann [2005],

Aloulou and Artigues [2010], les auteurs utilisent un ordre partiel pour représenter un ensemble de solutions pour un problème à une machine.

2.3.3 Ordres partiels "And/Or"

Une plus grande généralisation des ordres partiels que nous étudierons dans cette thèse sont les ordres partiels "And/Or" (AOPO). Ils sont notamment décrits par Möhring et al. [2004], Gillies and Liu [1995], mais leurs origines peuvent être tracées à des travaux antérieurs. Ce paradigme permet d'exprimer des contraintes disjonctives telles que "la tâche i ou la tâche j doit être ordonnancée avant la tâche k". Les contraintes de précédence deviennent des "conditions d'attente", et un ordre partiel "And/Or" π est un ensemble de conditions d'attente : $(X, k) \in \pi$ signifiant que l'une des tâches de l'ensemble de tâche $X \subseteq N$ doit être ordonnancée avant que la tâche k le soit. Un cas particulier survient lorsque $|X| = 1$, qui correspond à une contrainte de précédence classique. Bien qu'à notre connaissance, cette structure n'ait pas été utilisée dans une approche proactive, on peut considérer un AOPO comme une décision partielle caractérisant l'ensemble des ordonnancements admissibles pour ces contraintes.

2.3.4 Arbres PQR

Dans Baptiste et al. [1991] les auteurs décrivent la structure des arbres PQR, augmentant les arbres PQ (utilisés antérieurement dans le cadre de problème d'ordonnancement avec temps de préparation [Baptiste and Favrel, 1984]) avec une capacité d'ordre. Un arbre PQR est un tuple (V, A) de nœuds et d'arcs formant un arbre. Les nœuds sont de quatre types distincts :

- Les nœuds permutables de $\mathbf{P} \subseteq V$: l'ordre de leurs enfants peut être choisi arbitrairement.
- Les nœuds réversibles de $\mathbf{Q} \subseteq V$: leurs enfants sont ordonnés, et cet ordre peut être inversé.
- Les nœuds ordonnés de $\mathbf{R} \subseteq V$: leurs enfants ont un ordre fixe.
- Les feuilles de $\mathbf{F} \subseteq V$: chacune est associée à une tâche dans N. L'ordre des feuilles impliqué par les choix d'ordres des enfants des autres nœuds de l'arbre définit la **frontière** de l'arbre PQR, dans laquelle on lit une séquence.

Un exemple d'arbre PQR dont la frontière actuelle est la séquence $ABCD$ est donné dans la figure 2.3.

Un arbre PQR constitue aussi une décision partielle concernant l'ordre des tâches. Les séquences de tâches admissibles sont les frontières qu'il est possible d'obtenir en permutant les nœuds P et en inversant les nœuds Q. Par exemple, inverser le nœud Q dans la figure donne la frontière $ABDC$.

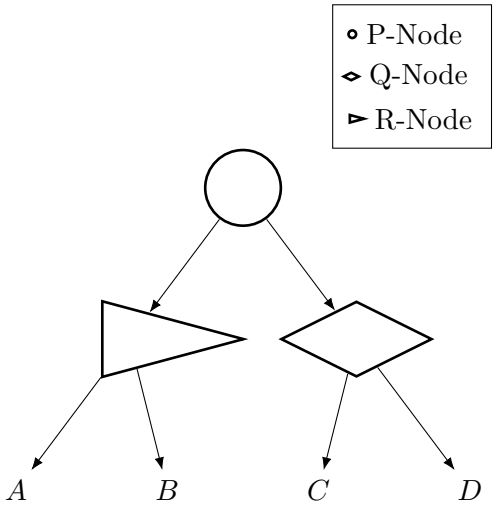


FIGURE 2.3 – Exemple d'un arbre PQR

2.3.5 Diagrammes de décision multivalués

Les diagrammes de décision ont initialement été imaginés pour l'étude de circuits logiques. Furent d'abord décrits les diagrammes de décision binaires (BDD), permettant de représenter n'importe quelle fonction booléenne sur des variables binaires sous une forme plus compacte que par exemple une table de vérité [Lee, 1959, Akers, 1978] ; puis les diagrammes de décision multivalués (MDD), permettant de représenter des fonctions booléennes sur des ensembles de variables entières. Un historique plus complet de l'origine des diagrammes de décision et des travaux qui ont mené à leur utilisation dans des problèmes d'optimisation et de satisfaction de contraintes est dressé dans Bergman et al. [2016a].

Dans cette thèse, un MDD est un graphe (V, A) dans laquelle l'ensemble des nœuds V est divisé en $|\chi| + 1$ couches ordonnées $\mathcal{L}_1, \ldots, \mathcal{L}_{|\chi|+1}$ (pour un ensemble de variables de décision χ). Chaque couche $\mathcal{L}$ est associée à une variable de décision $x \in \chi$. Les première et dernière couches sont composées d'un seul nœud (respectivement le nœud racine R et le nœud terminal T). Un arc orienté $a \in A$, lie un nœud u dans sa couche de départ $\mathcal{L}_k$, à un nœud v dans sa couche d'arrivée $\mathcal{L}_{k+1}$, et est étiqueté par un label $l_a \in D_x$, qui représente une valeur possible pour la variable associée à cette couche.

Un **chemin** entre deux nœuds u et v d'un MDD est une succession de nœuds $(u \to q_1 \to q_2 \to \cdots \to q_k \to v)$ tels qu'il existe un arc reliant chaque nœud du chemin à son successeur. S'il y a ambiguïté sur l'arc emprunté entre u et v, on peut préciser $u \xrightarrow{a} v$. La concaténation de deux chemins $p = (u \to \cdots \to v)$ et $p' = (u' \to \cdots \to v')$ se note $p \xrightarrow{a} p'$ si $a = (v, u')$ ou $p.p'$ si $v = u'$. Dans un chemin (dit **complet**) du nœud racine R au nœud terminal T : $(R \to \cdots \to T)$, les arcs a empruntés représentent chacun une affectation l_a, et par conséquent, puisque le chemin traverse toutes les couches, il représente une affectation de chaque variable :

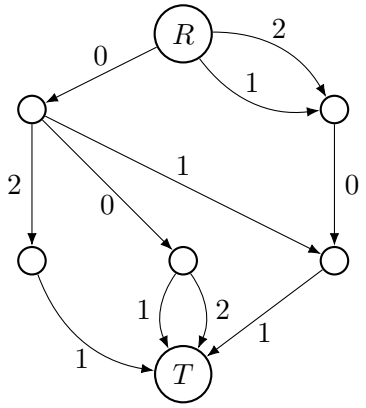


FIGURE 2.4 – Exemple de MDD

une solution.

L'ensemble des chemins complets d'un MDD représente donc un ensemble d'affectations d'un ensemble de variables de décision χ, qui peuvent être les solutions d'un problème. Un diagramme de décision peut donc être vu comme une fonction booléenne sur les variables de décision : $f(\check{x}) = \top$ ssi un chemin correspondant à $\check{x}$ est présent dans le MDD. Par exemple, la figure 2.4 représente les affectations $(0,2,1)$,$(0,0,1)$,$(0,0,2)$,$(0,1,1)$, $(1,0,1)$, et $(2,0,1)$.

Deux nœuds u et v d'une même couche sont dit équivalents lorsque l'ensemble des chemins de ces nœuds au terminal correspondent aux mêmes affectations des variables. Dans ce cas les nœuds sont redondants (on parle aussi de nœuds isomorphes). Un MDD est dit **réduit** lorsqu'il ne contient pas de nœuds redondants. Pour un ordre des variables donné, il existe un unique MDD réduit correspondant à un ensemble de solutions [Bergman et al., 2016a].

La **largeur** W d'un MDD est définie par le nombre de nœuds contenus dans sa plus grande couche : $W = \max_{k \in 1,\dots,|\chi|+1} |\mathcal{L}_k|$. Il est connu que pour certains problèmes, étant donné un ordre des variables de décisions associées à chaque couche, représenter l'ensemble de toutes les décisions admissibles par un MDD exact requiert un nombre de nœuds exponentiel en la taille du problème. Par conséquent, il peut être intéressant de limiter sa taille, ce qu'il est possible de faire en établissant une largeur maximale pour le MDD.

Dans une approche qui pourrait être qualifiée de proactive, Hadzic and Hooker [2006] utilisent un BDD pour représenter un ensemble de solutions presque optimales et proposent des outils pour effectuer une étude de sensibilité du problème. Dans Bergman et al. [2016b], Cire and van Hoeve [2013], les auteurs utilisent des diagrammes des décision pour représenter des ensembles de séquences de tâches, en associant une couche à chaque variable de décision de position pour laquelle $\check{\zeta}_i = j$ ssi la tâche j est en position i. Ils dérivent des bornes pour un problème d'ordonnancement, qu'ils intègrent à une méthode de séparation et évaluation.

Le chapitre 5 s'intéresse plus en détail aux diagrammes de décision multivalués pour l'ordonnancement sous incertitude.

Deuxième partie

Ensembles de solutions pour optimisation à deux niveaux

Nous avons démontré dans la partie précédente l'importance de la considération de l'incertitude dans la résolution de problèmes d'ordonnancement, et rapidement présenté des méthodes s'y attelant. Nous avons également répertorié plusieurs structures de données permettant la représentation d'ensembles de solutions.
De façon générale, il est préférable d'avoir le plus d'informations possible lors de la prise de décision. C'est pourquoi, dans les problèmes d'ordonnancement sous incertitude, pour lesquels de l'information supplémentaire est révélée lors de l'exécution d'une solution, il est prudent de ne pas arrêter complètement la décision sur une solution unique immédiatement, mais au contraire, de remettre certaines décisions à plus tard. Cependant, du à la complexité des problèmes d'ordonnancement, on ne peut pas prendre **toutes** les décisions dans le temps restreint de l'exécution, de façon optimale. Il s'agit donc de choisir quelles décisions seront prises au préalable, lorsque le moins d'information mais le plus de temps sont disponibles, et quelles décisions seront prises en temps réel, avec plus d'information, mais moins de temps.
Nous avons vu que l'on peut considérer que la décision partielle prise lors de la phase proactive restreint le domaine des solutions admissibles à un sous-ensemble de solutions. La décision restante sélectionne, parmi ces solutions, celle qui sera mise en oeuvre lors de la phase réactive. Les décisions partielles que l'on s'autorise à prendre influent sur la forme du sous-ensemble de solutions qu'il est possible d'obtenir.

Dans la partie suivante, nous proposons donc une méthode de décision à deux niveaux : le niveau de décision partielle préalable "offline", et le niveau de recours "online".
Dans le chapitre 3, nous présentons formellement la méthode d'optimisation à deux niveaux. Nous décrirons ce qui constitue une décision de premier niveau, et comment sont prises les décisions de second niveau. Nous verrons aussi quelles sont les limites de cette approche, en particulier, quels gains peuvent êtres obtenus, et quels en sont les facteurs limitant.
Ensuite, dans le chapitre 4, nous étudierons particulièrement cette méthode lorsque la décision de premier niveau constitue une séquence de groupes d'opérations permutables. Après avoir détaillé le contexte, nous décrirons les méthodes utilisées pour cette prise de décision, puis les résultats expérimentaux obtenus seront présentés et discutés.
Enfin, le chapitre 5 considérera une approche similaire pour une division différente de la décision ou la décision de premier niveau est la plus expressive possible. Nous étudierons comment cette décision peut être représentée par un diagramme de décision multivalué, et comment elle peut l'être par un ensemble de séquences. Nous décrirons des méthodes exactes et approchées pour le calcul de ces décisions. Les résultats expérimentaux de cette approche seront comparés aux résultats précédents.

Chapitre 3

Ordonnancement robuste/stochastique à deux niveaux

Sommaire

Ce chapitre introduit la famille d'approches à deux niveaux que nous proposons dans cette thèse. Nous présentons les problèmes que nous cherchons à résoudre, ainsi que le détail du principe de ces méthodes. Nous proposerons des métriques pour l'évaluation et la comparaison des méthodes de résolution, ainsi que plusieurs méthodes basiques qui serviront de points de comparaison.

3.1 Problème étudié, stratégies de résolution étudiées

Dans ce chapitre, nous nous intéressons à deux problèmes d'ordonnancement différents pour évaluer l'approche à deux niveaux plus généralement. Nous étudions des variantes sous incertitude de problèmes à une machine (1P) et de problèmes du jobshop (JSP).

Dans le problème 1P (voir chapitre 1), un ensemble de tâches doit être ordonnancé sur une unique machine. Chaque tâche a une durée d'exécution, une date de disponibilité, et une date de livraison. Les tâches doivent être ordonnancées en fixant une date de début pour chacune d'elles, sans préemption ni chevauchement, et ne peuvent être démarrées qu'après leurs date de disponibilité. Les tâches doivent aussi respecter un ensemble de contraintes de précédence. Un ordonnancement est évalué selon l'un de deux objectifs : le critère de retard maximal L_{MAX} d'un tâche (la différence entre la date de fin d'une tâche et sa date de livraison), ou le critère de somme des dates de fin des tâches $\sum C_i$.
Pour le problème du JSP (voir également le chapitre 1), un ensemble de jobs doit être ordonnancé sur un ensemble de machines. Chaque job est composé d'une séquence de tâches qui doivent se succéder sur des machines données. Les objectifs considérés pour ce problème sont la somme des temps de fin des jobs $\sum C_i$ et la durée totale de l'ordonnancement ou makespan C_{MAX}.

Pour le 1P, nous considérons une incertitude portant sur la date de disponibilité, pouvant provenir des aléas de livraison des fournisseurs par exemple [Wu et al., 2005]. Pour le JSP, nous considérons une incertitude plus classique sur la durée d'exécution des tâches. Dans les deux cas, cette incertitude est modélisée par un ensemble discret de scénarios $S \subseteq \overline{S}$, où $\overline{S}$ est l'ensemble de tous les scénarios possibles : S est un sous-ensemble de réalisations complètes de l'incertitude, qui pourraient être obtenues en échantillonnant des réalisations passées du problème, et ne nécessitent pas la connaissance des lois de distribution des paramètres incertains. L'ensemble des scénarios possibles, $\overline{S}$, est inconnu. Ainsi, pour le 1P par exemple, un scénario particulier s fixe une date de disponibilité r_i^s pour chaque job i, et un ensemble de scénarios représente plusieurs réalisations possibles de ces dates. Les scénarios ne sont pas choisis pour représenter des cas providentiels ou problématiques, on suppose donc que plus $|S|$ est grand, plus S approxime fidèlement la distribution sous-jacente (des doublons sont possibles, S est un multi-ensemble).
Nous supposons un modèle d'information qui correspond à la connaissance de l'état présent des tâches de l'atelier (indisponible, disponible, en cours d'exécution, terminée). Pour le problème 1P, cela revient à connaître la date de disponibilité des tâches au moment où elles sont disponibles, et pour le JSP à connaître la durée d'exécution des tâches au moment où elles sont terminées.
Nous étudions deux façons d'adapter les objectifs à la variante sous incertitude. Étant donnée une solution et un objectif γ, le critère peut être calculé pour chaque scénario $s \in S$ (comme il le serait dans le cas deterministe). Nous agrégeons les résultats obtenus sur chaque scénario soit par la moyenne des objectifs atteints, ce qui correspond à un objectif stochastique ; soit par le résultat atteint sur le pire scénario, ce qui correspond à un objectif robuste.

Pour tous les objectifs γ considérés, les problèmes déterministes correspondant au 1P $(1|prec, r_i|\gamma)$ et au JSP $(J||\gamma)$ sont connus pour être NP-difficiles [Garey et al., 1976, Lenstra et al., 1977, Lenstra and Rinnooy Kan, 1979], ce qui correspond au

cas ou $|S| = 1$. A plus forte raison, les problèmes d'optimisation sous incertitude associés, avec $|S| \geq 1$ sont donc aussi NP-difficiles.

Nous considérons pour la résolution de ces problèmes des stratégies de résolution à deux niveaux s'inscrivant dans les approches proactives-réactives intégrées. Le premier niveau correspond à la décision partielle proactive, qui cherche à restreindre le domaine de décision à un ensemble de solutions optimal compte tenu du second niveau de résolution sur l'ensemble des scénarios S. La stratégie F définit le choix du type de décision de premier niveau. Le deuxième niveau de résolution consiste en une politique s'appliquant à chaque moment de décision de la phase réactive. Les moments de décision sont les moments où une machine est libre, et des tâches sont prêtes à y être exécutées. Lors de ces moments de décision, une tâche doit être lancée. La politique de second niveau aboutit à un ordonnancement semi-actif, et ne peut donc pas introduire de temps d'attente. Nous verrons en fait dans le prochain chapitre (section 4.3) qu'il est en théorie possible d'obtenir de meilleurs scores en introduisant des temps d'attente, mais puisque nos objectifs sont réguliers (ordonnancer une tâche plus tôt n'empire pas le score, toutes choses égales par ailleurs, cf chapitre 1), il est contre-intuitif de les tolérer. Nous noterons H la politique de second niveau.

3.1.1 Formalisation du problème

Formellement, les problèmes sont définis comme suit :

3.1.1.1 Le problème à une machine

Le problème à une machine (1P) vise à ordonnancer un ensemble N de tâches étant données des dates de disponibilités incertaines décrites par une ensemble discret de scénarios S. Un problème avec $|N|$ tâches et $|S|$ scénarios est défini comme un tuple $\mathbf{1P} = (P, R, D, E, \oplus, \gamma)$ dans lequel :

- $p_i \in P$ est la **durée d'exécution** de la tâche i.
- $r_i^s \in R$ est la **date de disponibilité** de la tâche i dans le scénario $s \in S$.
- $d_i \in D$ est la **date de livraison** de la tâche i.
- E est l'ensemble des **contraintes de précédence** ($(i, j) \in E$ ssi la tâche i doit être ordonnancée avant la tâche j).
- Le paramètre $\oplus \in \{max, avg\}$ est l'**agrégateur d'objectifs** entre les scénarios.
- Le paramètre $\gamma \in \{\sum C_i, L_{MAX}\}$ est l' **objectif**.

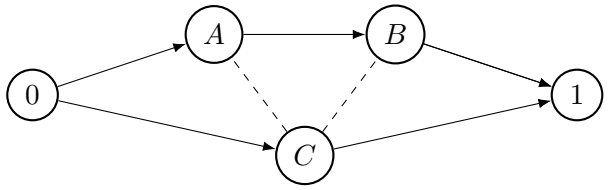
La figure 3.1 présente un exemple d'un problème 1P avec 3 tâches et 2 scénarios. Dans le graphe de précédence, un arc plein représente une contrainte de précédence. Les arcs pointillés représentent un ordre indécidé entre deux tâches (on omet les arcs de transitivité).

Job	$r_j^{s_1}$	p_j	d_j
A	0	2	3
B	3	3	5
C	2	3	5

(a) Scenario 1

Job	$r_j^{s_2}$	p_j	d_j
A	0	2	3
B	2	3	5
C	3	3	5

(b) Scenario 2



(c) Graphe de précédence

FIGURE 3.1 – (P1) Exemple de problème : une machine, deux scénarios et minimisation du pire plus grand retard.

3.1.1.2 Le problème du jobshop

Le problème du jobshop vise à ordonnancer un ensemble N de jobs sur un ensemble de machines M étant données des durées d'exécution des tâches incertaines décrites par une ensemble discret de scénarios S. Un problème avec $|N|$ jobs, $|M|$ machines et $|S|$ scénarios est défini comme un tuple **JSP** $= (P, M, \oplus, \gamma)$ dans lequel

- $p_{i,j}^s \in P$ est la **durée d'éxécution** de la tâche j du job i dans le scénario s.
- $m_{i,j} \in M$ est la **machine** sur laquelle la tâche j du job i doit être exécutée.
- $\oplus \in \{max, avg\}$ est l'**agrégateur d'objectif**.
- $\gamma \in \{C_{MAX}, \sum C_i\}$ est l'**objectif**.

La figure 3.2 présente un exemple d'un problème JSP avec 3 jobs et 2 scénarios. Le graphe de précédence représente la contrainte d'exécution séquentielle des tâches d'un même job. Les arc pointillés représentent l'ordre indécidé des tâches qui partagent la même machine (les tâches des différentes machines sont aussi indiquées par leur couleur).

3.1.2 Stratégie à deux niveaux

Le problème de décision à deux niveaux se résume pour une instance donnée et une stratégie $F = (\Omega_F, H_F)$ en la recherche de :

$$\mathcal{X}^* = \underset{\mathcal{X} \in \Omega_F}{\operatorname{argmin}} \underset{s \in S}{\oplus} \gamma(H_F(\mathcal{X}, s), s)$$

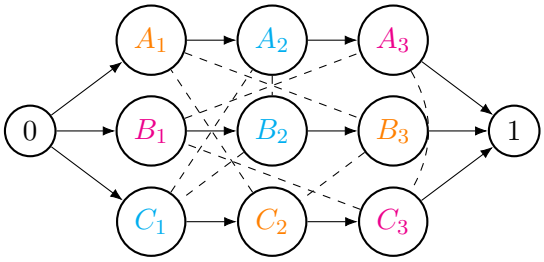
où Ω_F est un ensemble de décisions de premier niveau accessibles par la stratégie F, et H_F est la politique de second niveau de la stratégie. Les décisions de premier et second niveau sont décrites plus en détail dans ce qui suit. Toutefois, nous considérons seulement des stratégies pour lesquelles tout $\mathcal{X}$ est une décision partielle telle que H_F y est bien définie quelque soit le scénario $s \in \overline{S}$, et aboutit à un ordonnan-

$p_{i,j}^{s_1}$	1	2	3
A	3	2	1
B	2	2	1
C	1	2	1

(a) Scenario 1

$p_{i,j}^{s_2}$	1	2	3
A	2	2	1
B	3	2	1
C	1	2	1

(b) Scenario 2



(c) Graphe de précédence

FIGURE 3.2 – Exemple de problème (P2) : jobshop à 3 jobs, 3 machines, 2 scénarios et minimisation du pire makespan

cement admissible pour l'instance du problème considéré [1]. Puisque la décision de premier niveau est faite en connaissance de la politique de second niveau, on peut dire qu'une décision partielle $\mathcal{X}$ est une solution du problème à deux niveaux.

3.1.2.1 Décisions du premier niveau

Nous considérons au premier niveau que la stratégie F est implémentée par un algorithme A qui, étant donnée une instance d'un problème, fournit à l'issue de la phase proactive une décision $\mathcal{X}$ (dont la représentation peut avoir une importance) consistant en une restriction du domaine des variables de décision de séquence $\mathcal{X} \subseteq D_{\chi_{seq}}$. Cette restriction des séquences implique aussi une restriction du domaine des variables d'ordonnancement $\mathcal{X}' \subseteq D_{\chi_{date}}$ aux ordonnancements dont l'ordre d'exécution des tâches fait partie du domaine admissible des séquences.

Les différentes stratégies que nous étudions sont limitées par différentes restrictions sur la décision $\mathcal{X}$ qu'il leur est possible de prendre. L'ensemble des décisions accessibles à une stratégie F est $\Omega_F \subseteq 2^{D_{\chi_{seq}}}$. A un extrême, la stratégie JSEQ (que nous considérons dans la section 3.3.2) indique qu'au premier niveau une décision $\mathcal{X}$ restreint le domaine des séquences à un unique élément : une unique séquence (par exemple ABC) pour le 1P. A un autre extrême, la stratégie FIFO pure (section 3.3.1) indique qu'au premier niveau la décision $\mathcal{X}$ ne restreint pas du tout le domaine des séquences, et l'ordonnancement est déterminé uniquement par la politique de second niveau sur le scénario réalisé. A encore un autre extrême, une stratégie *complète* permet de décider au premier niveau de n'importe quelle restriction de séquence $\mathcal{X}$. Entre ces extrêmes se situent les stratégies qui ont accès

1. On exclut donc la stratégie qui fixerait les dates de début des tâches au premier niveau car il pourrait alors exister des scénarios qui rendraient cette solution non admissible

à certaines décisions partielles et pas à d'autres (par exemple, seulement les restrictions impliquées par des décisions de type "la tâche A doit être avant la tâche B").

On remarque donc que certaines stratégies en dominent d'autres en théorie, lorsque $\Omega_F \subset \Omega_{F'}$, alors F' domine F, jusqu'aux stratégies complètes, pour lesquelles $\Omega_F = 2^{D_{\chi_{seq}}}$, qui dominent toute les autres stratégies. Toutefois, dans un temps limité, les algorithmes utilisés pour implémenter les différentes stratégies ne sont pas garantis d'obtenir la meilleure décision possible. Et bien que le temps de calcul ne soit pas forcément bloquant lors de la phase proactive, en fonction de l'application, on peut préférer une stratégie incomplète, mais pour laquelle un algorithme donne de meilleurs résultats dans un temps limité, parce que menant à un espace de décision plus restreint et facile à explorer. De façon similaire, deux algorithmes implémentant la même stratégie peuvent avoir des performances différentes. Cette partie de la thèse est consacrée à l'étude des performances de diverses stratégies et leurs implémentations.

Définition 3.1.2.1. *Une décision de premier niveau est dite* ***valide*** *pour une politique de second niveau H ssi quelque soit le scénario, H aboutit à un ordonnancement admissible.*

3.1.2.2 Décisions du second niveau

La décision de premier niveau est faite en connaissance de la politique de décision du second niveau.

La décision de second niveau est une politique s'appliquant à chaque moment de décision de la phase réactive. Nous pouvons la considérer dans son ensemble comme une fonction $H_F : \Omega_F \times \overline{S} \to D_{\chi_{date}}$, qui, à partir d'une décision de premier niveau et étant donné un scénario, aboutit à une décision complète d'ordonnancement. Puisque nous considérons des objectifs réguliers, l'intérêt de H_F est d'aboutir à une séquence, l'ordonnancement au plus tôt des tâches est ensuite dominant (aucun autre ordonnancement n'est préféré, cf chapitre 1) pour l'ordonnancement des tâches. Il peut donc être utile de considérer que H_F est une fonction de $\Omega_F \times \overline{S} \to D_{\chi_{seq}}$

Dans cette thèse, H_F est la politique "Premier arrivé premier servi" (FIFO), correspondant aussi à la politique "*earliest release date*" (ERD) pour le problème à une machine [Bachtler et al., 2020]. A chaque moment de décision, la politique FIFO choisit, parmi les tâches pouvant être lancées selon la décision de premier niveau, celle qui était prête en premier. Pour le 1P, une tâche est prête après sa date de disponibilité, et pour le JSP, une tâche est prête lorsque la tâche précédente du job est terminée. Les égalités sont départagées par l'ordre lexicographique des tâches. La façon de déterminer les tâches qui peuvent êtres lancées, et donc le détail de l'implémentation de H_F, dépend de la décision de premier niveau de la stratégie

F et de la façon dont elle est représentée, mais la procédure générale de la politique FIFO est décrite dans l'algorithme 3.1.

Algorithme 3.1 Politique FIFO

1: **pour** $t = 0$ et à chaque moment de décision **faire**
2: **pour** chaque machine disponible $m \in M$ **faire**
3: recenser les tâches pouvant être lancées sur m.
4: ordonnancer la tâche qui était prête la première (le cas échéant).

Remarquons que l'utilisation de la politique FIFO n'est pas optimal. Toutefois, elle correspond à deux critères importants pour une politique de second niveau : c'est une politique computationnellement frugale, et dont le résultat dépend effectivement de l'information obtenue dans la phase réactive. En effet, une politique trop complexe serait difficile à mettre en oeuvre dynamiquement (par exemple le ré-ordonnancement complet des tâches compte tenu de la nouvelle information). D'autres politiques simples que l'on peut imaginer comme la politique consistant à prioriser les tâches les moins longues ne conviennent pas pour nos problèmes : dans le cas du 1P, les durées d'exécution des tâches ne sont pas des paramètres incertains, donc puisque la tâche la moins longue est toujours la même, la politique prendrait la même décision quelque soit le scénario, la flexibilité induite par la décision partielle ne serait pas utilisée ; et dans le cas du JSP, les durées des tâches ne sont révélées qu'après leurs complétions, l'information ne peut donc pas être utilisée pour décider laquelle exécuter. On pourrait toutefois utiliser les autres paramètres disponibles en sus de la date de disponibilité pour renseigner la prise de décision de second niveau, mais cela n'as pas été étudié dans cette thèse.

3.2 Évaluer les stratégies à deux niveaux

Les solutions obtenues sont évaluées selon les critères définis par l'agrégateur $\oplus$ et l'objectif γ définis antérieurement (section 3.1.1). Afin de comparer les différentes approches entre elles, il nous faut définir d'une part des métriques d'évaluation des solutions générées par ces approches pour un ensemble d'instances, et d'autre part un ensemble d'instances pour chacun des deux problèmes étudiés (le problème à une machine, et le problème du jobshop).
Notons que les différentes stratégies ne peuvent être évaluées que par l'intermédiaire d'algorithmes (ou méthodes) implémentant la recherche de solution du problème à deux niveaux.

3.2.1 Critères de performance des méthodes

3.2.1.1 Instances d'entraînement, instances de test

Les problèmes que nous considérons modélisent l'incertitude par un ensemble de $|S|$ scénarios discrets, que l'on suppose échantillonnés aléatoirement d'une distribution inconnue, à partir de réalisations passées de l'incertitude. Comme suggéré dans

le chapitre 2, bien que ces scénarios sont les seules informations disponibles concernant la distribution sous-jacente du problème, le véritable objectif est de fournir des solutions de bonne qualité non pas pour ces scénarios là, mais pour les scénarios correspondants à des réalisations futures de l'incertitude, d'autres scénarios tirés de la même distribution. Nous distinguons alors les scénarios **d'entraînement** (les scénarios S) et les scénarios **de test**. Les scénarios de tests sont utilisés exclusivement pour l'évaluation des solutions, et ne sont pas accessibles lors de la résolution, ni de la phase proactive, ni de la phase réactive. En évaluant les solutions obtenues sur un ensemble suffisamment grand de scénarios de tests, nous pouvons déterminer à quel point elles s'adaptent bien aux nouveaux scénarios.

3.2.1.2 Métriques

Pour chaque instance, les méthodes sont évaluées selon les métriques suivantes, dont les résultats moyens sont comparés pour des ensembles d'instances :

— Le **Score** obtenu par une méthode sur une instance est la valeur de l'objectif pour la solution $\mathcal{X}$ atteinte dans un temps maximum de 2 heures (évaluée sur les scénarios d'entraînement S).

$$\text{Score}(\mathcal{X}) = \underset{s \in S}{\oplus} \gamma(H(\mathcal{X}, s), s)$$

— Le **Score généralisé** est le score obtenu par la solution $\mathcal{X}$ obtenue par une méthode lorsque évaluée sur un ensemble de scénarios de test S', issus de la même distribution que les scénarios d'entraînement S utilisées pour le calcul de solution et du Score. Notons que pour $\oplus = avg$, plus le nombre de scénarios de test est grand, plus le score de généralisation obtenu approxime fidèlement l'espérance du score qu'obtiendra la solution sur une nouvelle instance.

$$\text{Score généralisé}(\mathcal{X}) = \underset{s \in S'}{\text{avg}} \gamma((H(\mathcal{X}, s), s)$$

Cependant, pour $\oplus = max$, lorsque le nombre de scénarios de test est très grand, considérer le pire scénario rend le score généralisé excessivement mauvais, et ne constitue pas une métrique intéressante pour la comparaison des méthodes. Pour évaluer les solutions sous un angle robuste, mais éviter de juger des solutions par rapport à des scénarios exceptionnels, nous considérons une fonction $\mathcal{Q}_{.9}$ extrayant la valeur du quantile 90% d'un ensemble pour évaluer notre solution et non le véritable pire scénario parmi tous les scénarios de test :

$$\text{Score généralisé}(\mathcal{X}) = \underset{s \in S'}{\mathcal{Q}_{.9}} \gamma(H(\mathcal{X}, s), s)$$

Pour un nombre suffisamment grand de scénarios de test, étant donné un scénario aléatoire s, $\mathcal{P}(\text{Score généralisé}(\mathcal{X})) \geq \gamma(H(\mathcal{X}, s), s)) = 0.9$

— La **Performance** (resp. **performance généralisée**) d'une méthode est le

rapport du meilleur score (resp. du score généralisé) atteint (par n'importe quelle méthode) m^* sur le score (resp. score généralisé) de la méthode évaluée.

$$\text{Performance}(\mathcal{X}) = \frac{m^*}{\text{Score}(\mathcal{X})}$$

Ainsi, une performance de 1 signifie que la méthode considérée obtient la meilleure solution sur toutes les instances. Une performance de 0.5 signifie qu'en moyenne le score obtenu par la méthode est deux fois plus grand que la meilleure solution connue. La métrique de performance nous permet d'évaluer l'impact de paramètres sans nécessairement distinguer les différents critères utilisés.

— La **Perte de généralisation** est le rapport du score généralisé sur le score.

$$\text{Perte}(\mathcal{X}) = \frac{\text{Score généralisé}(\mathcal{X})}{\text{Score}(\mathcal{X})}$$

Une perte de 1 signifie qu'il n'y a pas de différence en moyenne entre le score obtenu sur les scénarios d'entraînement et de test. Une perte plus grande que 1 signifie qu'en moyenne le score obtenu sur les scénarios de test est moins bon que celui obtenu sur les scénarios d'entraînement.

— La **Flexibilité** d'une solution Flexibilité($\mathcal{X}$) est une mesure du nombre de solutions qu'elle représente. La définition de cette mesure dépend de la stratégie implémentée, mais est une valeur comprise entre 0 et 1.

Nous cherchons à déterminer l'impact des variations de paramètres d'instances sur les résultats obtenus par les différentes méthodes. Pour étudier l'impact d'un paramètre, les méthodes sont évaluées sur un jeu d'instances qui fait varier ce paramètre. Nous présentons dans la suite les instances que nous utilisons pour l'évaluation.

3.2.2 Génération d'instances

En l'absence d'un jeu d'instances accessible pour nos problèmes, nous générons des instances personnalisées s'inspirant de celles décrites dans Artigues et al. [2016]. Les instances se basent sur les paramètres suivants :

— $|N|$: Le nombre de tâches
— $|M|$: Le nombre de machines (pour le JSP).
— $|S|$: Le nombre de scénarios d'entraînement.
— Δ : La variabilité relative des scénarios entre eux (portant sur la date de disponibilité dans le cas du problème à une machine, et sur les durées des tâches pour le problème du jobshop). Concrètement, la variabilité est un paramètre impactant la variance de la distribution de probabilité sous-jacente des paramètres incertains.
— $D(\mathcal{G})$: La densité du graphe de précédence (pour le 1P). Nous mesurons cette densité par le nombre de contraintes de précédence sur le nombre maximal de ces contraintes ($\frac{2|E|}{N(N-1)}$).

Nous choisissons des valeurs par défaut pour ces paramètres de manière à ce qu'elles nous permettent d'observer les limites de la performance de nos approches :

— $|N| = 100$, $|S| = 25$, $\Delta = 0.3$, $D(\mathcal{G}) = 0.01$ pour le 1P.

— $|N| = 10$, $|M| = 10$, $|S| = 25$, $\Delta = 0.5$ pour le JSP.

Ensuite, pour pouvoir étudier les variations de performance des approches étudiées et fonction des paramètres de instances, nous générons trois jeux d'instances pour le 1P et 4 pour le JSP en utilisant les paramètres par défaut sauf pour le paramètre étudié qui, lui, prend différentes valeurs dans un intervalle donné.

Les jeux d'instances 1P-N and JSP-N-M font varier le nombre de tâches (et de machines pour le JSP). Les jeux d'instances 1P-S and JSP-S font varier le nombre de scénarios d'entraînement. Les jeux d'instances 1P-Δ and JSP-Δ font varier la variabilité des scénarios entre eux. Le jeu d'instances 1P-$D(\mathcal{G})$ fait varier la densité du graphe de précédence.

Pour chaque valeur des paramètres, 5 instances différentes sont générées. En particulier, dans chaque jeu d'instance, il y a 5 instances qui partagent les mêmes paramètres par défaut, pour un total de 20 instances standards pour le 1P et 15 pour le JSP. Ces instances sont rassemblés en deux jeux d'instances virtuels 1P-A et JSP-A. Les tables 3.1 et 3.2 résument les caractéristiques des jeux d'instances.

Instances	$\|N\|$	$D(\mathcal{G})$	$\|S\|$	Δ	#inst.
1P-N	[10, 20, 50, 100, 150, 200]	0.01	25	0.3	30
1P-$D(\mathcal{G})$	100	[0, 0.001, 0.01, 0.05, 0.1, 0.2]	30	0.3	25
1P-S	100	0.01	[2, 5, 10, 15, 20, 25, 35, 50, 100]	0.3	45
1P-Δ	100	0.01	25	[0, 0.1, 0.3, 0.5, 0.7, 1]	30
1P-A	100	0.01	25	0.3	20

TABLE 3.1 – Jeux d'instances pour le problème à une machine

En partant d'un ensemble de paramètres fixés, chaque instance du problème 1P est générée comme suit. Premièrement, un scénario nominal est généré : pour chaque tâche, une durée p_i est choisie par un tirage uniforme dans l'intervalle $[50, 100]$, une date de disponibilité r_i dans l'intervalle $[0, \sum P/2]$, et une date de livraison d_i dans $[\sum P/2, \sum P]$.

A chaque scénario nominal est associé un graphe de précédence aléatoire. Nous utilisons la méthode décrite dans Gehrlein [1986] pour générer des ordres partiels aléatoires : en partant d'un ensemble de contraintes de précédence initialement vide E, une contrainte de précédence (i, j) aléatoire telle que $i < j$ (pour l'ordre lexicographique des tâches) est ajoutée. N'ajouter que des contraintes de précédence dans

Instances	$\|N\|$	$\|M\|$	$\|S\|$	Δ	#inst.
JSP-N-M	[5,10,20]	[2,5,10]	25	0.5	45
JSP-S	10	10	[2, 5, 10, 15, 20, 25, 35, 50, 100]	0.5	45
JSP-Δ	10	10	25	[0, 0.1, 0.3, 0.5, 0.7, 1]	30
JSP-A	10	10	25	0.5	15

TABLE 3.2 – Jeux d'instances pour le problème du jobshop

un sens donné (pour un ordre donné, sans perte de généralité) nous assure la faisabilité de l'ajout. La fermeture transitive de E génère ensuite toutes les contraintes de précédence impliquées par l'ajout. Des contraintes sont ajoutées ainsi jusqu'à ce que la densité du graphe de précédence atteigne ou dépasse la densité désirée $D(\mathcal{G})$. Secondement, on associe à chaque scénario nominal une densité de probabilité : pour chaque date de disponibilité incertaine, une loi normale $\mathcal{N}(r_i, r_i * \Delta)$ décrit la distribution sous-jacente du paramètre. Enfin, les $|S|$ scénarios d'entraînement et 1000 scénarios de tests sont tirés pour constituer respectivement l'instance d'entraînement et de test. L'instance d'entraînement est utilisée comme entrée pour l'approche à deux niveaux évaluée, tandis que l'instance de test n'est utilisée que pour l'évaluation.

Les instances du problème JSP sont générées de façon similaire, bien que ce soit la durée des tâches qui soit incertaine. Pour chaque tâche, la durée $p_{i,j}$ est choisie par un tirage uniforme dans l'intervalle $[50, 100]$, et la distribution associée est une loi exponentielle $p_{i,j} + \mathcal{E}(\Delta.p_{i,j})$.

3.3 Stratégies d'ordonnancement stochastiques/robustes de référence

Dans les chapitres suivants, nous nous proposons d'analyser les performances d'implémentation de deux stratégies à deux niveaux. L'une d'elle est basée sur les groupes d'opérations permutables (Chapitre 4) et l'autre est basée sur des diagrammes de décisions multivalués (Chapitre 5).

Avant de nous intéresser à ces stratégies, nous présentons dans cette section quelques stratégies de référence, qui serviront de point de comparaison pour la suite.

3.3.1 Stratégie FIFO pure

Le premier point de comparaison que nous considérons est la stratégie purement réactive "FIFO", qui correspond au cas où aucune décision n'est prise au premier

niveau : $\mathcal{X} = D_{\chi_{seq}}$. Dans ce cas, l'heuristique FIFO de second niveau prend toutes les décisions. A chaque moment de décision, l'heuristique démarre, sur chaque machine libre, les tâches disponibles (vis-à-vis des date de disponibilité, et du graphe de précédence) qui ont la plus grande priorité.

Ce point de comparaison présente un grand intérêt pour nous puisque elle met au défi l'idée qu'il soit possible (du moins dans un temps restreint) de trouver des décisions de premier niveau intéressantes. Si ce n'est pas le cas, c'est que le problème ne possède pas une structure suffisante pour que l'on puisse prendre des décisions pertinentes sans l'information obtenue lors de la phase réactive. Au contraire, s'il est possible de faire mieux que l'approche FIFO purement réactive, cela démontre l'intérêt des stratégies à deux niveaux lorsque la règle FIFO est utilisée, et l'intérêt des décisions en amont, malgré le contexte d'incertitude.

3.3.2 Stratégie JSEQ

Le second point de comparaison est la stratégie principalement proactive JSEQ, pour laquelle la décision de premier niveau $|\mathcal{X}| = 1$ restreint maximalement le domaine des variables de décision de séquence. Une solution du problème à deux niveaux définit donc une séquence unique. Comme pour la majorité des travaux consacrés au problème du jobshop stochastique et robuste [Kouvelis and Yu, 1997, Gu et al., 2009, van den Akker et al., 2013, Hao et al., 2013, Horng and Lin, 2015, Wang et al., 2018, 2019, Ghasemi et al., 2021] la décision de second niveau ne consiste qu'en l'ordonnancement au plus tôt des tâches dans l'ordre défini par la séquence. A chaque moment de décision t où une tâche termine son exécution sur une machine, la tâche suivante dans la séquence associée à cette machine est lancée immédiatement si elle est prête, ou alors la machine attend qu'elle le soit.

Plus précisément, on nomme "séquence de tâches", "JSEQ", ou simplement "séquence", le produit du premier niveau de la stratégie JSEQ. C'est un ordre total des tâches pour chaque machine (soit 1 séquence pour le 1P, et $|M|$ séquences pour le JSP). Les solutions JSEQ obtenues au premier niveau doivent être valides vis à vis de la politique de second niveau.

3.3.2.1 Solution JSEQ valide

D'après la définition de validité, une JSEQ est **valide** pour une instance si FIFO aboutit à un ordonnancement admissible. Nous voyons toutefois dans la suite que la validité d'une JSEQ ne dépend pas spécifiquement de l'heuristique utilisée. Dans le cas de la stratégie JSEQ, la validité et l'admissibilité se confondent.

Le graphe disjonctif [Roy and Sussmann, 1964, Balas, 1969] permet de définir formellement la validité d'une JSEQ pour le 1P et le JSP.
Le graphe disjonctif contient un nœud pour chaque tâche, plus les nœuds virtuels source et puits marquant le début et la fin de l'ordonnancement. Le graphe contient

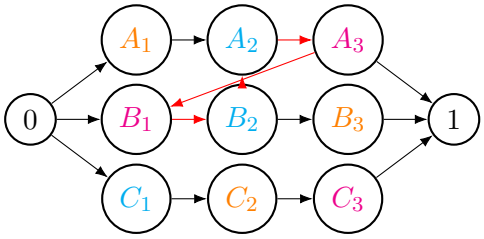


FIGURE 3.3 – Sélection complète d'un graphe disjonctif associé à une JSEQ invalide (par soucis de clarté, seuls sont représentés les arcs pertinents)

des arcs orientés et non-orientés, et représente les problèmes à une machine et du jobshop de la façon suivante :

- Le nœud source est connecté aux nœuds de chaque tâche par un arc orienté dont le coût est le date de disponibilité de la tâche pour le 1P et 0 pour le JSP.
- Chaque nœud-tâche est connecté au nœud puits par un arc orienté de coût $p_i - d_i$ pour le 1P et p_i pour le jobshop.
- Pour chaque contrainte de précédence (i, j), un arc orienté du nœud-tâche i à j de coût p_i.
- Pour chaque paire de tâche s'exécutant sur la même machine, un arc non-orienté en relie les nœuds.

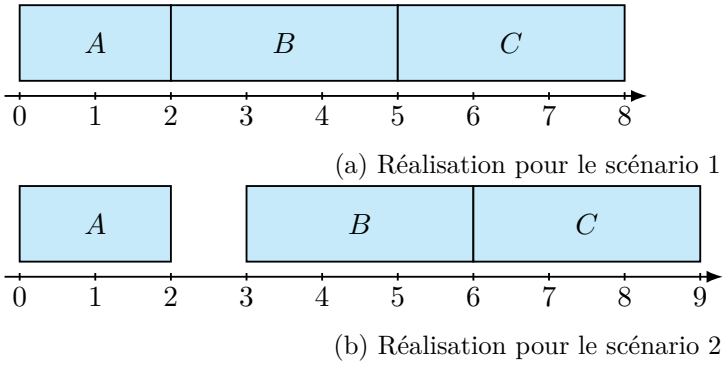
Ces arcs non-orientés sont appelés arcs *disjonctifs* car ils représentent la disjonction entre deux ordres possibles pour l'exécution des tâches sur la machine.
Une *sélection partielle* est une orientation d'une partie des arcs disjonctifs. Chaque arc ainsi orienté porte le coût de la durée de la tâche d'origine. Une *sélection complète* est une orientation de tous les arcs disjonctifs. Une sélection est sans circuit si elle n'introduit pas de circuit orienté dans le graphe disjonctif.

Une JSEQ correspond à une sélection complète telle que définie par l'ordre des tâches qu'elle impose. Ainsi, la validité d'une JSEQ est donnée par la définition :

Définition 3.3.2.1. *Une JSEQ est valide ssi la sélection complète associée est sans circuit.*

Pour le 1P, une JSEQ est invalide quand une paire de tâche impliquées dans une contrainte de précédence $(i, j) \in E$, et $j \prec i$ dans la JSEQ. Dans le cas du JSP, le circuit est produit par les contraintes de précédence internes aux jobs et celles provenant de l'orientation des arcs (définissant l'ordre d'exécution sur les machines). Un exemple correspondant à une JSEQ invalide pour le JSP est donné dans la figure 3.3.

Étant donné une JSEQ et un scénario, l'ordonnancement calé à gauche est dominant pour les objectifs réguliers que nous considérons [Kouvelis and Yu, 1997]. Cet ordonnancement est obtenu facilement par le calcul de la date de début au plus tôt des tâches, qui correspond au plus long chemin entre le nœud source et le nœud-tâche concerné dans le graphe de la sélection complète [Balas, 1969]. De



(a) Réalisation pour le scénario 1

(b) Réalisation pour le scénario 2

FIGURE 3.4 – La solution JSEQ ABC du problème P1 (Figure 3.1)

plus, le plus long chemin entre le nœud source et le nœud puits donne la valeur de l'objectif C_{MAX} pour le JSP et L_{MAX} pour le 1P. La décision de second niveau ne consiste donc qu'à suivre la JSEQ et démarrer chaque tâche dans l'ordre de la séquence dès qu'elle est disponible dans ce scénario.

La figure 3.4 présente une solution pour l'instance P1 (figure 3.1) obtenue dans le paradigme de stratégie JSEQ. Dans le premier scénario, (a), la tâche C est le plus en retard, résultant en un retard $L_{MAX} = 3$. Dans le second scénario, (b), c'est aussi la tâche 3 qui est le plus en retard, avec $L_{MAX} = 4$. Il peut être montré que la séquence ABC est une décision de premier niveau optimale pour cette stratégie, puisqu'elle minimise le retard maximal obtenu sur les deux scénarios ($\max L_{MAX} = 4$).

3.3.2.2 Méthodes d'implémentation

La littérature des approches proactives pour la résolution du problème du job-shop est principalement constituée d'approches à voisinages ou méta-heuristiques [Gu et al., 2009, Hao et al., 2013, Horng and Lin, 2015, Ghasemi et al., 2021]. D'après van den Akker et al. [2013], ces approches par recherche locale obtiennent de meilleurs résultats lorsqu'elles sont évaluées sur un ensemble de scénarios échantillonnés plutôt qu'en fixant des valeurs nominales aux paramètres incertains en se ramenant à un problème déterministe (par exemple, avec un quantile de la distribution, ou une valeur moyenne). Conformément à ces résultats, nous proposons un algorithme génétique (**AG-JSEQ**) et un algorithme tabou (**TAB-JSEQ**) pour implémenter le premier niveau de la stratégie JSEQ pour le 1P et le JSP. Ces méta-heuristiques sont des cas particuliers des méthodes qui seront présentées plus tard dans le chapitre 4 (section 4.3.2).

Nous utilisons aussi une implémentation par une approche exacte : un modèle de programmation par contraintes directement adapté de l'exemple *"stochastic job-shop"* de **CP Optimizer** (voir "CP-Optimizer 20.1.0 User Manual") et dont les

1: **pour** $s \in S$ **faire**
2: $\texttt{StartOf}(\textsc{Job}[i,s]) \geq r_i^s \quad \forall i \in N$
3: $\texttt{EndOf}(\textsc{Job}[i,s]) \leq \texttt{StartOf}(\textsc{Job}[j,s]) \quad \forall (i,j) \in E$
4: $\textsc{Seq}[s] == \texttt{Sequence}(\textsc{Job}[:,s])$
5: $\texttt{NoOverlap}(\textsc{Seq}[s])$
6: $\textsc{Obj}[s] == \gamma(\textsc{Job}[:,s])$
7: **si** $s == 0$ **alors**
8: RefSeq $== Seq[s]$
9: **sinon**
10: $\texttt{SameSequence}(\textsc{Seq}[s], \text{RefSeq})$
11: $\texttt{Minimize}(\oplus(\textsc{Obj}[:])$

Modèle 3.1 – Modèle PPC implémentant la stratégie JSEQ pour le 1P

1: **pour** $s \in S$ **faire**
2: $\texttt{StartOf}(\textsc{Job}[m_{i,j-1}, i, s]) \leq \texttt{EndOf}(\textsc{Job}[m_{i,j}, i, s]) \quad \forall i \in N, \forall j \in M \setminus \{0\}$
3: $\textsc{Seqs}[s,m] == \texttt{Sequence}(\textsc{Job}[m,:,s]) \forall m \in M$
4: $\texttt{NoOverlap}(\textsc{Seqs}[s,m]) \ \forall m \in M$
5: $\textsc{Obj}[s] == \gamma(\textsc{Job}[:,:,s])$
6: **si** $s == 0$ **alors**
7: $\textsc{RefSeqs} == \textsc{Seqs}[s,:]$
8: **sinon**
9: **pour** $m \in M$ **faire**
10: $\texttt{SameSequence}(\textsc{Seqs}[s,m], \textsc{RefSeqs}[m])$
11: $\texttt{Minimize}(\oplus(\textsc{Obj}[:])$

Modèle 3.2 – Modèle PPC implémentant la stratégie JSEQ pour le JSP

versions 1P et JSP sont décrites dans les algorithmes 3.1 et 3.2.

Pour le 1P, une variable d'intervalle différente $\textsc{Job}[i,s]$ représente chaque tâche i dans chaque scénario s, puisque les tâches s'exécutent différemment en fonction du scénario. Pour chaque scénario d'entraînement, l'incertitude est levée, ce qui permet de fixer les contraintes spécifiques au scénario (par exemple la date de disponibilité des tâches dans ce scénario) ligne 2. Les contraintes de précédence entre les tâches de chaque scénarios sont fixées ligne 3. Toutes les tâches d'un même scénario $\textsc{Job}[:,s]$ sont rassemblées dans une variable de séquence (ligne 4), ce qui permet d'appliquer la contrainte `NoOverlap`[2] (ligne 5), et de forcer les séquences de chaque scénario à être égales (contrainte `SameSequence`) ligne(7-10). L'objectif γ est agrégé avec $\oplus$ et minimisé (lignes 6,11).

Le modèle est légèrement plus complexe mais conceptuellement identique pour le JSP. La variable d'intervalle $\textsc{Job}[m_{i,j}, i, s]$ représente la j-ème tâche du job i (sur la machine $m_{i,j}$). Les contraintes de précédence entre les tâches d'un même job sont décrites ligne 2. Les autres contraintes ne diffèrent que par le nombre des machines.

A l'inverse de la stratégie FIFO pure, la stratégie JSEQ constitue un point de

2. voir section 1.3.1.2

comparaison qui permet d'étudier s'il est intéressant de reporter certaines décisions à la phase réactive, malgré la non-optimalité de la politique FIFO. Si c'est le cas, c'est que l'information apportée dans la phase réactive est suffisamment significative pour compenser ce déficit d'optimalité et la complexité accrue de l'espace de résolution.

3.3.3 Stratégie à deux niveau IDEAL

Nous l'avons évoqué, bien qu'en théorie, certaines stratégies en dominent d'autres, en pratique, nous verrons dans le chapitre 4 que dans un temps limité, il est très difficile pour une stratégie dominante d'obtenir des décisions partielles de meilleures qualité. Nous aimerions savoir quel est le "manque à gagner", c'est-à-dire quel est le gain maximal qu'il est possible d'obtenir en théorie par une stratégie qui associerait à chaque scénario une séquence (et un ordonnancement semi-actif correspondant) optimale compte-tenu de ce scénario. En d'autres termes, la stratégie optimale produirait au premier niveau une décision $\mathcal{X}^*$, et sa politique de second niveau H^* serait telle que $H^*(\mathcal{X}^*, s)$ aboutisse à la séquence optimale $\pi_s^* = \underset{\pi}{\operatorname{argmin}}\ \gamma(\pi, s)$.

Pour implémenter cette stratégie idéale dans une approche exacte, il suffit d'enlever la contrainte `SameSequence` du modèle 3.1 et 3.2 (les lignes 7 à 10). Ainsi, les séquences sélectionnées pour chaque scénario ne sont plus liées et sont libres d'être différentes. Cela revient effectivement à résoudre $|S|$ modèles déterministes. Notons que cette stratégie n'est idéale que pour un ensemble de scénarios d'entraînement, et, si l'on tentait d'évaluer $\mathcal{X}^*$ sur des scénarios de test, la solution pourrait ne pas obtenir de bons résultats. Pour obtenir une borne pertinente en test, il faudrait résoudre tous les problèmes déterministes associés aux scénarios de test de chaque instance.

En fait, dans un contexte d'information limitée, on peut montrer que cette stratégie est impossible à mettre en œuvre. En effet, considérons le problème à une machine de la figure 3.5. Dans le scénario 1, il peut être montré que la séquence ABC est optimale (avec un retard maximal de 0), tandis que dans le scénario 2, c'est la séquence CBA qui l'est (avec un retard maximal de 1). Mais puisque l'incertitude porte sur la date de disponibilité de la tâche B, aucune information n'est disponible au moment de lancer la tâche A ou C. Les scénarios sont indistinguables avant le temps $t = 4$, et attendre cette date pour commencer l'exécution des tâches entraînerait un retard encore plus important. La stratégie idéale permet donc seulement d'obtenir un majorant (sous réserve de prouver l'optimalité de la solution obtenue).

3.4 Résultats

Dans cette section, nous présentons les résultats obtenus pour les stratégies JSEQ, FIFO et IDEAL qui sont représentées par les implémentations CP-JSEQ,

Job	r_j	p_j	d_j
A	0	4	11
B	4	2	8
C	0	6	12

(a) Scénario 1

Job	r_j	p_j	d_j
A	0	4	11
B	6	2	8
C	0	6	12

(b) Scénario 2

FIGURE 3.5 – Instance du 1P ($\oplus = max, \gamma = L_{MAX}$)

AG-JSEQ, TAB-JSEQ, FIFO, et IDEAL.
Toutes les instances décrites dans la section 3.2 sont résolues par les différentes méthodes présentées en se basant sur les instances d'entraînement, dans une limite de temps de 2 heures. Les instances sont résolues pour les 4 objectifs $\oplus$ γ présentés. Toutes les méthodes s'exécutent sur un unique processeur (Xeon E5-2695 v4 @ 2.10GHz). Les modèles de PPC sont résolus par IBM CP Optimizer 20.1. Les heuristiques sont implémentées en Python, tandis que les modèles PPC utilisent l'API Python de CPO. Les méthodes sont exécutées sur les instances d'entraînement et évalués sur les instances d'entraînement et de tests selon les métriques décrites dans la section 3.2.

Les tables A.1, A.2, A.3, A.4, A.5, A.6 et A.7 (en annexe) présentent les performances généralisées des méthodes pour différents paramètres d'instances [3].

3.4.1 Comparaison des implémentations de la stratégie JSEQ

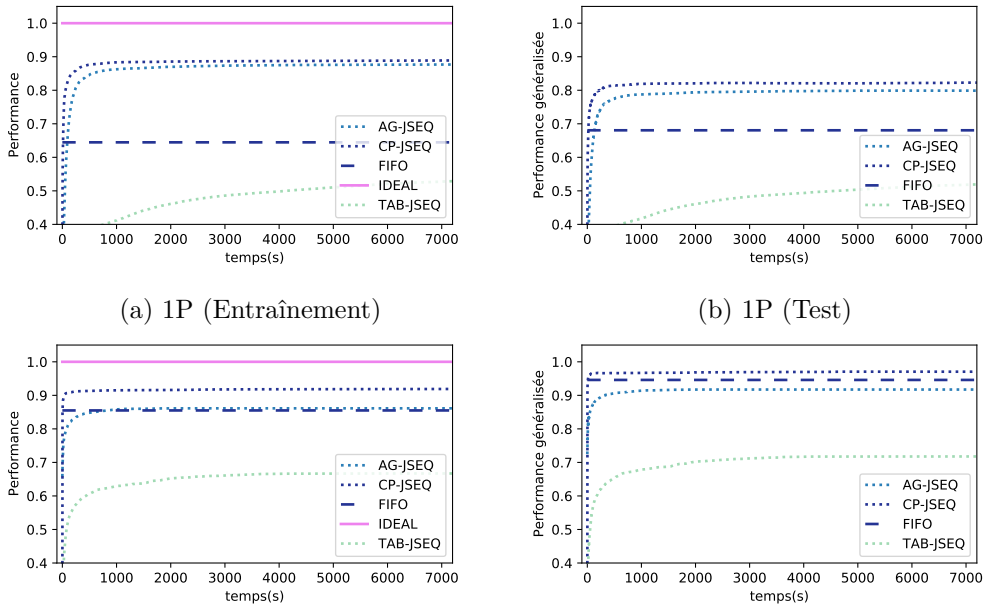
Les tables A.1, A.2, A.3, A.4, A.5, A.6 et A.7, permettent de comparer les performances généralisées de la méthode PPC avec les méthodes heuristiques (colonnes CP-JSEQ, AG-JSEQ et TAB-JSEQ) en fonction de la variation des différents paramètres des instances et de l'objectif.

La figure 3.6 résume l'évolution du score généralisé au cours du temps pour ces trois méthodes dans le cas standard (instances 1P-A et JSP-A).

On peut voir dans la figure et les tables mentionnées que la méthode CP-JSEQ obtient de meilleures performances et performances généralisées que les méthodes AG-JSEQ et TAB-JSEQ, pour les deux problèmes du 1P et JSP. En effet pour les deux problèmes, la méthode TAB-JSEQ n'obtient pas de bons résultats (autour de 50% de performance pour le 1P et 70% pour le JSP) ; les méthodes AG-JSEQ et CP-JSEQ obtiennent des résultats plus comparables, mais la méthode CP-JSEQ est meilleure en moyenne d'environ 5 points de pourcentage pour le JSP et d'entre 1 et deux points pour le 1P. Cet écart peut certainement être attribué à un manque de diversification par l'algorithme génétique et la méthode tabou. En effet, la figure 3.6 montre que les méthodes convergent autour d'un minimum local bien avant la fin du temps limite.

3. Les tables contiennent aussi les résultats obtenus par des stratégies qui seront présentées dans les chapitres suivants

Cet écart ne semble pas être influencé par les différences entre scénarios (tables A.1 et A.2) mais tend à s'accentuer pour les instances les plus grandes (tables A.3 et A.4), ce qui confirme la performance de CP optimizer sur ces problèmes. On peut aussi remarquer que la méthode GA-JSEQ semble obtenir de meilleurs résultats que la méthode TAB-JSEQ en moyenne. Le détail des résultats en fonction des objectifs révèle que pour le 1P, CP-JSEQ est plus efficace pour l'objectif L_{MAX}, mais les résultats sont plus mitigés pour l'objectif $\sum C_i$ (table A.5). Pour le JSP, la méthode CP-JSEQ domine hégémoniquement sur tous les objectifs (table A.6). On remarque que pour le 1P, lorsque la densité des précédences augmente, la différence de performance s'amenuise, mais la tendance reste inchangée (table A.7).
Ces résultats justifient l'utilisation de la méthode CP-JSEQ comme référence pour représenter la stratégie JSEQ dans les chapitres suivants.



(a) 1P (Entraînement)

(b) 1P (Test)

(c) JSP (Entraînement)

(d) JSP (Test)

FIGURE 3.6 – Performance et performance généralisée au cours du temps (instances *A*)

3.4.2 Gain possible à l'entraînement

La figure 3.7 représente les performances à l'entraînement des meilleurs représentant des stratégies FIFO, JSEQ, et IDEAL, pour chaque objectif, en fonction de la variabilité des scénarios et de la taille des instances.
Dans les résultats présentés, seule l'implémentation de la stratégie FIFO est garantie d'être optimale. Mais les résultats présentés suggèrent déjà les instances pour lesquelles il existe un gain possible par rapport à FIFO ou JSEQ. Les zones les plus intéressantes sont celles pour lesquelles la différence entre IDEAL et le meilleur de

FIFO et JSEQ est la plus grande.

De façon générale, on peut voir que pour le 1P et le JSP, lorsque la variabilité entre scénario est nulle, il n'y a pas de gain possible sur JSEQ, puisque si tous les scénarios sont identiques, une séquence unique est optimale pour tous. A l'inverse, lorsque la variabilité augmente, la stratégie JSEQ, ne peut s'adapter suffisamment et ses performances sont dégradées. On remarque toutefois que pour le 1P, lorsque $\gamma = L_{MAX}$, la tendance n'est pas claire, au contraire, en particulier pour $\oplus = max$, presque aucun gain n'est possible par rapport à JSEQ. On observe ce phénomène pour presque toute les valeurs de $|N|$. Nous supposons que par construction, les instances sont telles que une seule séquence peut-être suffisamment robuste sur tous les scénarios d'entraînement à la fois. Avec l'utilisation du quantile 90%, l'impact de ce phénomène peut être amoindri lors de l'évaluation sur les scénarios de tests, cependant, il faut réaliser que si une seule séquence peut-être une solution optimale, les stratégies plus complexes risquent de ne pas tirer beaucoup parti de la phase d'entraînement.

On peut aussi voir que pour le 1P, la taille des instances à un impact mitigé sur le gain possible. Cependant, pour le JSP, on remarque un impact clair du nombre de machines. Les gains possibles sur JSEQ sont égaux lorsque $\gamma = \sum C_i$, mais lorsque $\gamma = C_{MAX}$, aucun gain n'est possible si trop peu de machines sont impliquées. Une tendance similaire est observée quant aux gains possibles sur FIFO lorsque $\gamma = C_{MAX}$.

Enfin, remarquons que les amplitudes de gain possibles sont bien plus importantes pour le 1P que pour le JSP, en particulier pour la stratégie FIFO.

3.4.3 Perte de généralisation

Toutes les méthodes utilisent pour le calcul de solution les scénarios d'entraînement, et il est légitime de se demander à quel point les solutions obtenues peuvent être appliquées à des scénarios qui ne sont pas contenus dans l'ensemble d'entraînement. Pour vérifier si les solutions "généralisent" bien (c'est à dire si les scores obtenus sur les instances d'entraînement reflètent bien les scores obtenus sur les isntances de test), nous étudions la perte de généralisation.

La figure 3.8 montre que la perte de généralisation et sa variabilité diminuent rapidement avec le nombre de scénarios d'entraînement, pour toutes les méthodes considérées, et les deux problèmes 1P et JSP.

On remarque que pour l'agrégateur "max", la perte de généralisation semble converger vers une valeur inférieure à 1 (visible en particulier pour le JSP). En effet, bien que les solutions robustes soient calculées par rapport au pire scénario à l'entraînement, l'évaluation sur les scénarios de test utilise le scénario pire que 90% des scénarios. Les scores obtenus en test sont donc moins pessimistes qu'à l'entraînement.

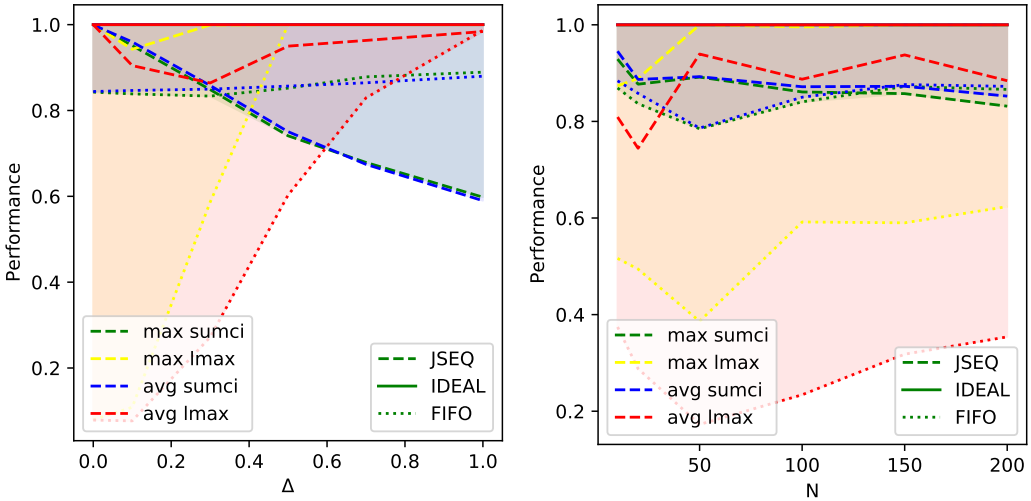
Il faut noter que la méthode FIFO n'apprend rien lors de la phase proactive, c'est pourquoi on observe une tendance à la baisse de la perte de généralisation pour l'agrégateur "max", puisque plus il y a de scénarios, plus l'évaluation sera pessimiste à l'entraînement. Mais pour l'agrégateur "avg", on observe une simple diminution de la variabilité.

3.4.4 Discussion

Les résultats obtenus par les implémentations de la stratégie JSEQ suggèrent qu'en particulier, CP-JSEQ est un bon représentant de la stratégie. Bien que l'optimalité des solutions ne soit pas prouvée, CP-JSEQ est significativement meilleure que les métaheuristiques que nous avons implémenté, ce qui laisse présager d'une bonne qualité des solutions trouvées pour la stratégie JSEQ.

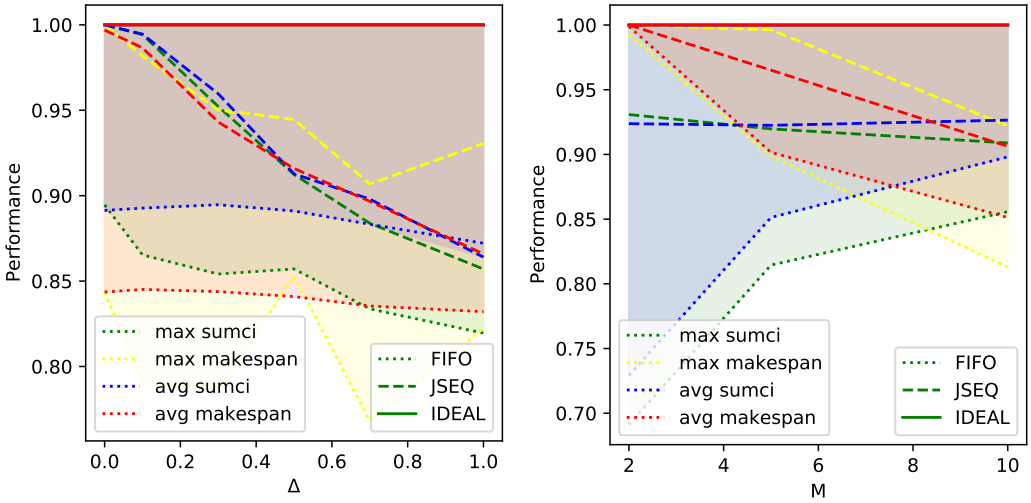
Auquel cas, l'étude des stratégies de référence révèle qu'il existe pour de nombreuses instances, un gap de gain possible allant de quelques pourcent à une dizaine de pourcent pour certaines instances, pour des stratégies dominant les stratégies JSEQ et FIFO. Cependant, pour certaines instances (en particulier les instances du 1P pour l'objectif $\max L_{MAX}$) aucun gain ne semble possible (sous réserve d'optimalité des solutions de la stratégie IDEAL). Cela peut être dû à la façon dont sont générées les instances. Une étude plus poussée des cas dans lesquels l'utilisation de stratégies dominantes peut apporter un gain semble donc nécessaire. Des tests basés sur des instances modifiées de la littérature, peuvent compléter les résultats actuels.

Ces gains possibles à l'entraînement ne reflètent pas directement les gains possibles lors de la phase de test, mais en sont des indicateurs. En effet, l'étude de la perte de généralisation suggère que même pour un nombre relativement restreint de scénarios, la perte de généralisation moyenne, n'est pas si importante, et sa variation non plus. Ces résultats témoignent d'un apprentissage possible à l'aide des scénario d'entraînement de la distribution des scénarios de test, et donc d'une bonne corrélation entre les résultats d'entraînement et de tests. La question du gain possible en test pourrait être estimé par l'utilisation de la stratégie IDEAL, en calculant pour chaque scénario de test la séquence optimale. La qualité de cette borne est toutefois difficile à estimer, et peut être que de meilleures bornes peuvent elles être calculées pour affiner l'estimation.



(a) Performance en fonction de la variabilité entre scénarios Δ (1P)

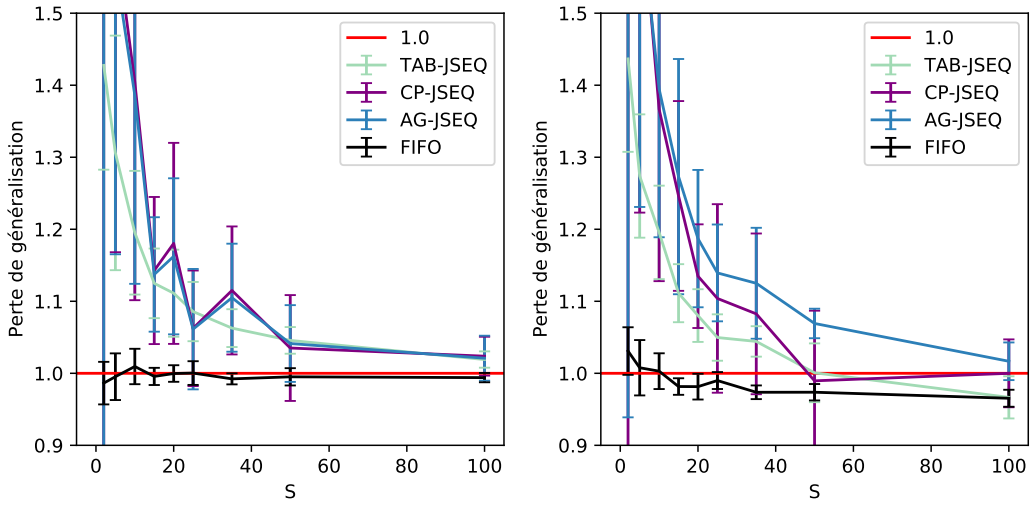
(b) Performance en fonction du nombre de tâches $|N|$ (1P)



(c) Performance en fonction de la variabilité entre scénarios Δ (JSP)

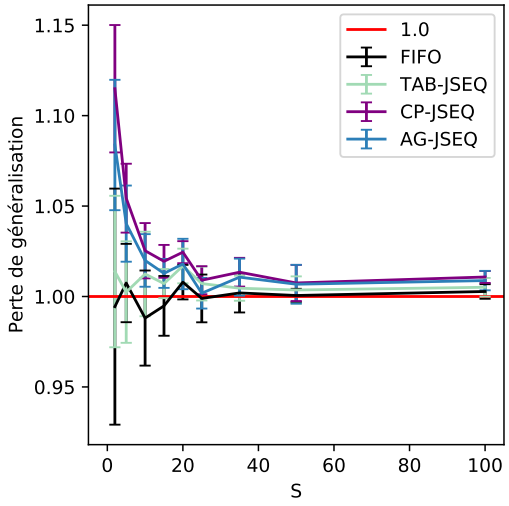
(d) Performance en fonction du nombre de machines $|M|$ (JSP)

FIGURE 3.7 – Performance des stratégies FIFO/JSEQ/IDEAL pour différentes valeurs de paramètres

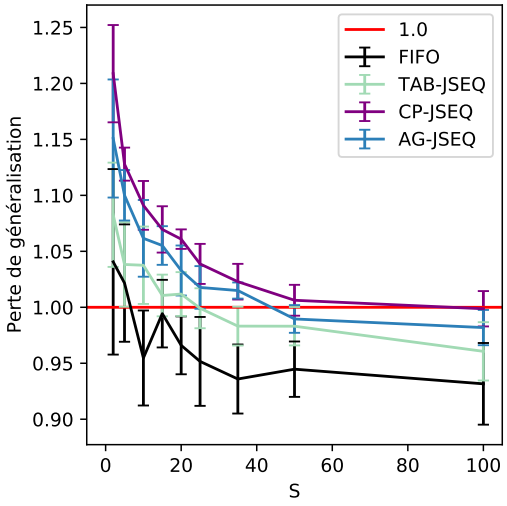


(a) 1P ($\oplus = avg$)

(b) 1P ($\oplus = max$)



(c) JSP ($\oplus = avg$)



(d) JSP ($\oplus = max$)

FIGURE 3.8 – Perte de généralisation en fonction du nombre de scénarios d'entraînement pour différents agrégateurs (instances A)

Chapitre 4

Ordonnancement robuste et stochastique à deux niveaux via les séquences de groupes d'opérations permutables

Sommaire

Nous cherchons dans ce chapitre à étudier les performances de l'approche à deux niveaux "GSEQ", pour laquelle la décision de premier niveau produit, sur chaque machine, une partition ordonnée des tâches (une séquence de groupes d'opérations permutables, voir définition 2.3.1.1). La politique FIFO au second niveau démarre, à chaque moment de décision, les tâches les plus prioritaires qui peuvent être lancées, mais doit ordonnancer toutes les tâches d'un groupe avant celles de son successeur. Après une courte introduction situant nos travaux par rapport à la littérature, ce chapitre propose une définition d'une décision valide de premier niveau pour cette stratégie, et propose une extension de l'algorithme de pire cas des GSEQ (décrit par

exemple dans Artigues et al. [2005]) compatible avec cette nouvelle définition. Nous présentons ensuite des méthodes pour le calcul de solution GSEQ : une méthode exacte, un algorithme tabou, un algorithme génétique et des méthodes hybrides de démarrage à chaud. Enfin, nous comparons ces implémentations entre elles et avec les méthodes implémentant les stratégies de référence.

Les travaux présentés dans ce chapitre ont été publiés dans une revue internationale [Rivière et al., 2023].

4.1 Introduction

Nous avons évoqué dans le chapitre 2 des approches proactives faisant usage des séquences de groupes d'opérations permutables, approches qui ont déjà été implémentées dans un contexte industriel [Billaut and Roubellat, 1996]. Pourtant, peu de travaux étudient les gains apportés par la flexibilité qu'apporte cette approche. Dans Pinot et al. [2007], Cardin et al. [2013], les auteurs étudient les performances sur l'objectif C_{MAX} d'une approche utilisant les séquences de groupes par rapport à des approches purement réactives ou purement proactives, pour différents niveaux d'incertitudes. Ils produisent dans un premier temps une séquence correspondant à un ordonnancement optimal dans le cas nominal, puis utilisent l'heuristique gloutonne définie par Esswein [2003] pour générer une séquence de groupes contenant la séquence optimale qui cherche à maximiser la flexibilité de la solution, tout en garantissant une borne supérieure sur la qualité de la solution. La politique FIFO est utilisée pour la phase réactive.

L'approche développée par Wu et al. [1999] utilise les séquences de groupes d'opérations permutables dans l'optique de minimiser le retard pondéré dans un problème de jobshop. Cette approche utilise une méthode de séparation et évaluation pour générer une séquence de groupes, mais se limite à deux groupes seulement (ce qui peut être vu comme une restriction supplémentaire de la stratégie : $\Omega_{2-GSEQ} \subseteq \Omega_{GSEQ}$). Ils utilisent l'heuristique ATC (*Apparent Tardiness Cost*) proposée par Vepsalainen and Morton [1987] dans la phase réactive.

Dans ces travaux qui étudient les performances des séquences de groupes d'opérations permutables [Wu et al., 1999, Pinot et al., 2007, Cardin et al., 2013], les expérimentations utilisent lors de la phase réactive une politique comme FIFO ou ATC, mais la phase proactive vise à maximiser la flexibilité, ou une mesure de qualité dans le pire cas sur un scénario nominal. Cela permet de proposer à un décideur un ensemble de solutions ayant une qualité garantie pour le scénario nominal, mais n'intègre pas la connaissance de la politique réactive (simulée lors de l'évaluation des solutions sur les scénarios) à la décision proactive. Nous nous intéressons dans cette thèse à la quantification du gain possible lorsque l'utilisation de la politique FIFO dans la phase réactive est anticipée dans la phase proactive. Le but n'est donc pas de produire dans la phase proactive la solution la plus flexible possible, mais la solution qui obtiendra les meilleures performances lorsque couplée à la politique

FIFO.

Dans cette veine, Artigues et al. [2016], Cheref et al. [2016a,b] étudient un problème à une machine avec un ensemble de scénarios discrets. Ils proposent une méthode PLNE pour le calcul de la séquence de groupes optimale pour la minimisation du retard maximal dans le pire scénario, sachant que les décisions de second niveau seront prises par la politique FIFO. Ils proposent un algorithme glouton et une méthode tabou pour implémenter les stratégies GSEQ et JSEQ et les comparer. Ils utilisent un jeu de scénarios d'entraînement dans leurs méthodes PLNE et tabou, et évaluent les solutions obtenues sur un jeu de scénarios de test. Les auteurs concluent positivement à l'intérêt de la stratégie GSEQ pour ce problème. C'est-à-dire que même pour un temps de calcul limité, la stratégie GSEQ peut fournir des solutions plus robustes que la stratégie JSEQ, malgré la plus grande complexité.

Ces travaux ont défini un cadre pour la comparaison des stratégies JSEQ et GSEQ, et établi les premiers résultats. On observe toutefois plusieurs limitations. Le modèle PLNE utilisé dans Artigues et al. [2016], Cheref et al. [2016a,b] pour l'implémentation de la stratégie GSEQ ne résout que de très petites instances, et seule une méthode tabou à été proposée comme alternative. De plus, seule l'optimisation robuste sur le problème à une machine a été étudiée. Enfin, aucune comparaison n'a été menée avec les approches visant à maximiser la flexibilité au premier niveau évoquées ci-dessus. De même, nous venons de voir que les études menées par Pinot et al. [2007] et Cardin et al. [2013] ne s'intéressent qu'au problème du jobshop, et ne considèrent pas non plus la politique de second niveau lors de la décision de premier niveau.

Les travaux menés dans cette thèse s'appuient sur ces travaux pour affiner la comparaison entre les stratégies GSEQ et JSEQ, notamment en proposant un nouveau modèle de PPC, un algorithme génétique, et une méthode de démarrage à chaud à partir d'une séquence de tâches pour le calcul de séquence de groupes. Nos travaux comparent ces stratégies plus généralement pour le problème à une machine et du jobshop, pour les différents agrégateurs et les différents objectifs présentés dans la section 3.1.1. Nous étudions aussi l'impact de la prise en compte intégrée du second niveau par rapport aux études qui se contentent de maximiser la flexibilité de la solution de premier niveau.

4.2 Une approche à deux niveaux basée sur des séquences de groupes d'opérations permutables

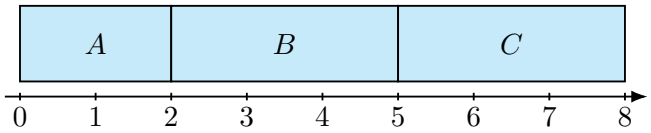
Dans la stratégie GSEQ, le premier niveau aboutit à une décision $\mathcal{X} \in \Omega_{GSEQ}$ décrite par une séquence de groupes d'opérations permutables (que l'on appelle aussi séquence de groupes, ou GSEQ), qui restreint l'ensemble du domaine des séquences à un sous-ensemble de séquences. On peut dire que la séquence de groupes *représente*

Job	r_j	p_j	d_j
A	0	2	3
B	3	3	5
C	2	3	5

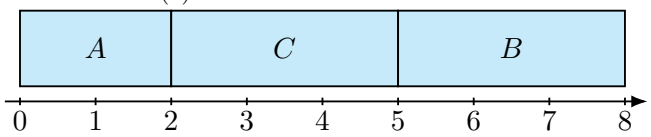
(a) Scenario 1 de P1

Job	r_j	p_j	d_j
A	0	2	3
B	2	3	5
C	3	3	5

(b) Scenario 2 de P1



(c) Réalisation du scénario 1



(d) Réalisation du scénario 2

FIGURE 4.1 – Exemple de solution de la stratégie GSEQ : $\pi = [\{A\}, \{B, C\}]$ pour le problème P1 ($A \prec B$)

un ensemble (potentiellement grand) de séquences.

Un exemple de la stratégie GSEQ est illustré dans la figure 4.1, pour le problème P1 (Figure 3.1, rappelé ici). La décision de premier niveau définit une séquence de deux groupes. Le premier groupe contient la tâche A, et le second les tâches B et C. La politique de second niveau ordonnance donc A en premier, au temps 0 dans les deux scénarios. Au temps 2, la tâche A termine. Il n'y a pas d'autres tâches dans le premier groupe, donc c'est la première tâche disponible dans le second groupe qui est lancée le plus tôt possible. Dans le scénario 1, c'est la tâche B (disponible en premier au temps 2) ce qui aboutit à l'ordonnancement (a), de séquence ABC. Dans le scénario 2, c'est la tâche C qui est disponible en premier, ce qui aboutit à l'ordonnancement (b), de séquence ACB. Au final, le score obtenu est $\max L_{MAX} = 3$, un progrès par rapport à la solution optimale de la stratégie JSEQ présentée dans la figure 3.4.

On remarque que cette stratégie domine à la fois les stratégies JSEQ et FIFO, puisque une séquence de groupes peut représenter n'importe quelle séquence unique (à la manière de la stratégie JSEQ) si chaque groupe ne contient qu'une tâche, et peut aussi ne pas restreindre le domaine des séquences (à la manière de la stratégie FIFO) si toutes les tâches sont dans le même groupe. Nous verrons toutefois que cette stratégie n'est pas complète au sens de la section 3.1.2.1), car toutes les restrictions possibles de l'ensemble des séquences ne sont pas accessibles.

Nous utilisons la politique FIFO au second niveau. Nous souhaitons donc que pour toute décision de premier niveau, FIFO soit capable d'aboutir à un ordonnancement admissible. Nous devons établir les conditions de *validité* (au sens de la

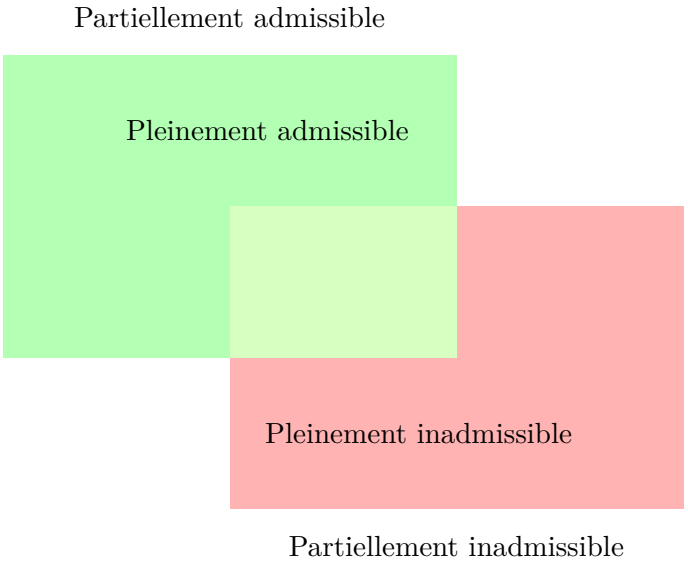


FIGURE 4.2 – Cas particuliers de GSEQ

définition 3.1.2.1) d'une séquence de groupes pour FIFO.

4.2.1 Séquence de groupes valide

Dans les travaux précédents utilisant les groupes d'opérations permutables [Erschler and Roubellat, 1989, Billaut and Roubellat, 1996, Artigues et al., 2005, 2016, Cheref et al., 2016a,b], une séquence de groupes est une solution admissible ssi **toutes** les séquences représentées (que l'on peut construire en ordonnant les tâches au sein de chaque groupe) sont admissibles (ce sont des JSEQ valides au sens de la définition 3.3.2.1). C'est ce que nous appellerons une séquence de groupes *pleinement* admissible. Notre définition de la validité modulo une heuristique nous permet de relâcher la contrainte de pleine validité. Nous commençons par distinguer plusieurs cas particuliers :

Définition 4.2.1.1. *Une GSEQ est :*

- *Pleinement admissible si elle représente seulement des JSEQ admissibles.*
- *Pleinement inadmissible si elle ne représente aucune JSEQ admissible.*
- *Partiellement admissible si elle représente au moins une JSEQ admissible.*
- *Partiellement inadmissible si elle représente au moins une JSEQ inadmissible.*

La figure 4.2 illustre les relations entre ces cas.

Nous pouvons démontrer qu'une GSEQ est valide pour la politique FIFO ssi elle est partiellement admissible. C'est-à-dire que pour toute GSEQ partiellement admissible, la politique de second niveau FIFO aboutit à un ordonnancement admissible.

L'intérêt de considérer des séquences de groupes partiellement inadmissibles est d'obtenir un espace de solutions de séquences de groupes admissibles plus grand et donc potentiellement, des ordonnancements de meilleure qualité après la phase réactive. Cela sera vérifié expérimentalement dans la section 4.4.8.

Nous avons vu dans la figure 3.3 comment un circuit dans le graphe de précédence peut mener à une JSEQ inadmissible pour le JSP. Durant la phase réactive, cela se manifeste par l'impossibilité de lancer une tâche sans violer l'ordre des tâches d'un job, ou l'ordre des tâches fixé par une séquence sur une machine. Cette impossibilité ne dépend pas du scénario, mais seulement de l'ordre imposé par la séquence. Soit un moment de décision lors de la phase réactive. Soit T l'ensemble des tâches qui sont terminées, R l'ensemble des tâches prêtes vis-à-vis des contraintes de précédence des jobs (elles sont les premières tâches d'un job ou bien toutes les tâches précédentes sont dans T). Soit aussi G l'ensemble des groupes "courants", c'est-à-dire sur chaque machine le premier groupe dont toute les tâches ne sont pas dans T (ces tâches sont prêtes vis à vis du planning). Soit G^- et G^+ les groupes qui précèdent et suivent les groupes dans G. Enfin, soit H une politique de second niveau telle que si une tâche t_i est prête vis a vis des contraintes de précédence de job et de séquencement ($t_i \in R \cap G$), H ordonnance t_i dans un nombre fini d'étapes. H peut être la politique FIFO.

Proposition 4.2.1.1. *Une GSEQ π est partiellement admissible ssi $\forall s \in \overline{S}$, $H(\pi, s)$ aboutit à un ordonnancement admissible.*

Démonstration. Soit $s \in \overline{S}$ un scénario tel qu'à un moment donné, H ne peut décider de la tâche à ordonnancer : c'est donc que $R \cap G = \emptyset$, sinon H finirait par lancer une tâche. Il y a donc une tâche dans chaque groupe courant dont un prédécesseur n'est pas prêt : $\forall g \in G, \exists t_i \in g$ tq $\exists t_j \in G^+, j < i$. H ne peut donc pas aboutir quel que soit le scénario. Et, puisque les tâches d'un groupe partagent les mêmes contraintes de séquencement, aucun ordre des tâches d'un groupe ne peut mener à une séquence admissible, et donc π est pleinement inadmissible.
Inversement, si π est pleinement inadmissible, aucune solution admissible n'est représentée, et donc H ne peut aboutir à un ordonnancement admissible. Par contraposition, étant donnée une GSEQ partiellement admissible, H aboutit à une séquence admissible quelque soit le scénario s. □

Par conséquent FIFO établit la validité partielle d'une GSEQ en temps polynomial.

La figure 4.3 illustre le comportement d'une GSEQ partiellement admissible et pleinement inadmissible pour le problème P2 (figure 3.2). La figure représente pour chaque machine, les groupes de tâches contenues dans G^-, G, et G^+ à un moment clé de l'ordonnancement. Les tâches de T sont colorées en vert et les tâches de R en bleu. On remarque que les deux GSEQ contiennent en particulier la JSEQ inadmissible

de la figure 3.3. La GSEQ de la figure 4.3a est pleinement inadmissible, en effet, aucune tâche ne peut être ordonnancée, en raison du circuit entre les tâches de G et G^+ sur les machines M_1 et M_2, toutes les tâches de R sont dans G^+. Cependant, la GSEQ de la figure 4.3b est partiellement admissible : puisque B_2 et A_2 sont dans le même groupe, il est possible d'ordonnancer A_2 en premier, ce qui libère A_3, puis B_1 et enfin B_2.

	G^-	G	G^+
M_1		$\{C_1,B_2\}$	$\{A_2\}$
M_2		$\{A_3\}$	$\{B_1\},\{C_3\}$
M_3	$\{A_1,C_2\}$	$\{B_3\}$	

(a) Comportement d'une GSEQ pleinement inadmissible

	G^-	G	G^+
M_1	$\{C_1\}$	$\{B_2,\ A_2\}$	
M_2		$\{A_3\}$	$\{B_1\},\{C_3\}$
M_3	$\{A_1,C_2\}$	$\{B_3\}$	

(b) Comportement d'une GSEQ partiellement admissible

FIGURE 4.3 – Comportement d'une GSEQ pleinement inadmissible / partiellement admissible

4.2.1.1 Validité des GSEQ par le graphe pire cas

Le **graphe pire cas** (WC-$\mathcal{G}$) d'une GSEQ pour un scénario donné est défini dans Artigues et al. [2005]. Ce graphe est composé d'un nœud source, d'un nœud puits, et de deux nœuds étiquetés i et i' pour chaque tâche i, qui représentent respectivement le début et la fin de la tâche. Pour chaque tâche i, il y a un arc entre la source et le nœud i de coût r_i, un arc du nœud i au nœud i' de coût p_i, et un arc du nœud i' au nœud puits de coût $p_i - d_i$. Pour chaque contrainte de précédence entre une tâche i et j, le graphe contient un arc du nœud i' au nœud j de coût nul. Pour chaque paire de tâche i et j appartenant à deux groupes consécutifs, un arc de coût nul lie le nœud i' au nœud j. Enfin, pour chaque paire de tâches i et j appartenant au même groupe G, un arc de coût $\sum_{k\in G} p_k$ lie les nœuds i et j'. Deux exemples de graphes pire cas sont représentés dans la figure 4.4 (les coûts des arcs sont omis dans un souci de lisibilité).
Le graphe de pire cas permet de calculer la date de fin des tâches dans le scénario pour la pire permutation des tâches des groupes (la pire JSEQ) représentée par une GSEQ (nous y reviendrons dans la section 4.2.2). Mais de plus, Artigues et al. [2005] montrent que le graphe de pire cas peut être utilisé pour établir la pleine validité d'une GSEQ.

Théorème 4.2.1.1. *Artigues et al. [2005] Une GSEQ est pleinement admissible ssi le graphe de pire cas est sans circuit.*

La présence de circuit dans le graphe de pire cas peut être établie par simple parcours en profondeur. On remarque que par contraposée, si le graphe de pire cas contient un circuit, la GSEQ est partiellement inadmissible. Par exemple, la figure 4.4 présente le graphe pire cas des deux GSEQ correspondant à la figure 4.3.

Les deux graphes contiennent un circuit (en rouge dans la figure), et sont donc partiellement inadmissibles.

Théorème 4.2.1.2. *Une GSEQ est pleinement inadmissible ssi le graphe de pire cas sans les arcs entre tâches d'un même groupe (WC-$\mathcal{G}$*) contient un circuit.*

Démonstration. ($\Longleftarrow$) Après avoir enlevé les arcs entre les tâches d'un même groupe dans le WC-$\mathcal{G}$ associé à une GSEQ π, seules subsistent les les précédences de séquencement fixées par la GSEQ (la sélection partielle) π, les précédences entre les tâches des jobs, et les arcs entre les nœuds de début et fin de tâches i et i'. Un circuit dans ce graphe WC-$\mathcal{G}$* implique un circuit dans le graphe disjonctif sans les arcs disjonctifs. Donc aucune sélection des arcs disjonctifs ne peut être sans circuit. Par conséquent, aucune séquence contenue dans la GSEQ n'est admissible, et donc π est pleinement inadmissible.

($\Longrightarrow$) Soit une GSEQ π pleinement inadmissible, et supposons que le WC-$\mathcal{G}$* associé est sans circuit. Nous montrons une contradiction. Si le WC-$\mathcal{G}$* est sans circuit, le graphe disjonctif sans arcs disjonctifs est sans circuit. Or, il est toujours possible d'orienter un arc disjonctif sans introduire de premier circuit dans un graphe. En effet, pour orienter un arc (i, j), s'il existe un chemin de i à j, orienter l'arc de i vers j n'introduit pas de circuit, sauf s'il existe un chemin de j vers i (auquel cas le graphe contenait déjà un circuit). Sinon, orienter l'arc de j vers i n'introduit pas de circuit. Nous avons vu qu'une orientation sans circuit des arcs correspond à une JSEQ admissible. Donc π serait partiellement admissible, ce qui amène à une contradiction. □

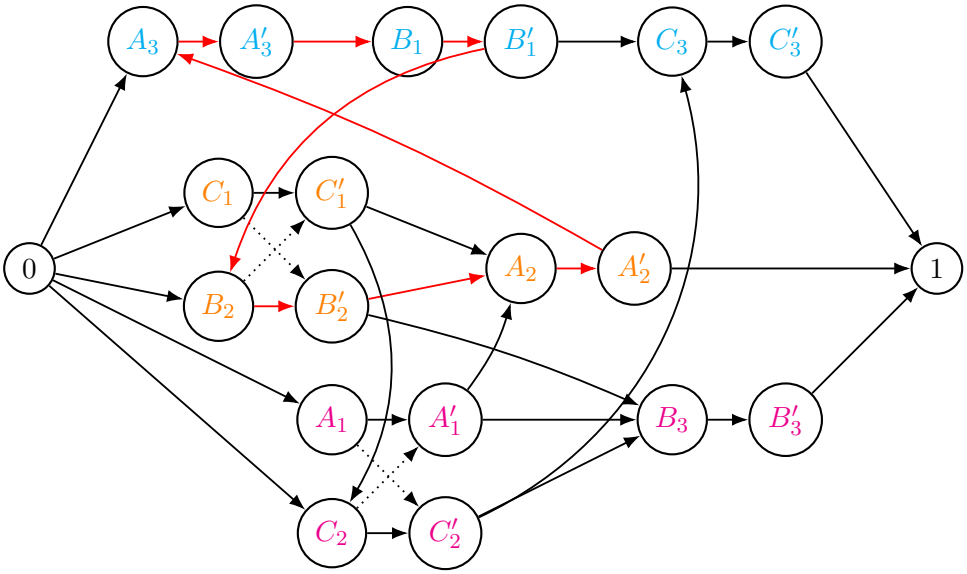
Par contraposition, WC-$\mathcal{G}$* est sans circuit ssi la GSEQ est partiellement admissible.

Dans la figure 4.4, une fois les arcs entre tâches d'un même groupe enlevés (en pointillés dans la figure), il ne subsiste de circuit que pour la figure 4.4a. La GSEQ associée est donc pleinement inadmissible, tandis que la GSEQ associée au graphe de la figure 4.4b est donc partiellement admissible.

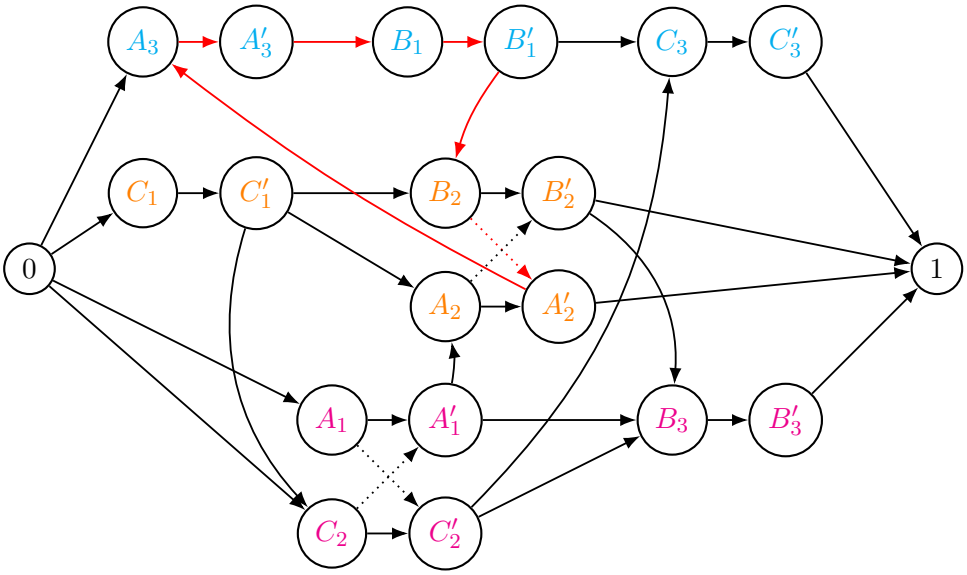
Pour le problème 1P, on remarque qu'une GSEQ est partiellement admissible (resp. pleinement admissible) ssi pour toute paire de tâches $(i, j) \in E$, le groupe de i est ordonné avant le groupe de j (resp. strictement avant) : $G(i) \preceq G(j)$.

4.2.2 Calcul du score dans le pire cas d'une GSEQ partiellement admissible

Une des propriétés intéressantes des séquences de groupes d'opérations permutables est la capacité d'établir facilement, pour chaque tâche, sa date de fin dans le pire cas pour un scénario donné, c'est-à-dire la pire date de fin de la tâche pour un ordonnancement dominant (semi-actif) correspondant à l'une des séquences représentées par la séquence de groupe. En effet, Artigues et al. [2005] montrent qu'il existe un algorithme polynomial utilisant le graphe de pire cas capable de calculer



(a) Graphe pire cas pour une GSEQ pleinement inadmissible
$\pi = [[\{C_1, B_2\}, \{A_2\}]], [\{A_3\}, \{B_1\}, \{C_3\}], [\{A_1, C_2\}, \{B_3\}]]$



(b) Graphe pire cas pour une GSEQ partiellement admissible
$\pi = [[\{C_1\}, \{B_2, A_2\}]], [\{A_3\}, \{B_1\}, \{C_3\}], [\{A_1, C_2\}, \{B_3\}]]$

FIGURE 4.4 – Graphes de pire cas pour une GSEQ pleinement inadmissible et une GSEQ partiellement admissible

cette date de fin dans le pire cas. En utilisant les dates de fin dans le pire cas, des informations sur les valeurs pire cas pour les objectifs réguliers peuvent être obtenues. Pour certains objectifs comme le retard maximum L_{MAX} ou le makespan C_{MAX}, la valeur objectif pire cas peut directement être obtenue (puisque ne met-

tant en jeu que la date de fin d'une unique tâche), et pour d'autres objectifs comme la somme des dates de fin $\sum C_i$, une borne supérieure sur l'objectif pire cas est obtenue (puisque le pire cas pour la date de fin n'est pas nécessairement le même pour toutes les tâches).

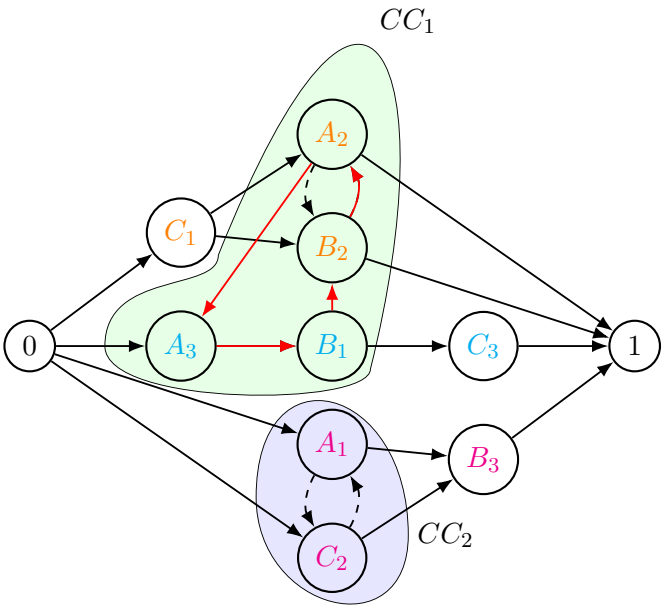
Il est opportun ici de rappeler que les évaluations des objectifs dans le pire des cas peuvent être utilisées comme objectif robuste lors du calcul de séquence de groupes dans la phase proactive. Bien sûr, les séquences de groupes correspondant à de simples séquences sont dominantes pour la minimisation de l'objectif dans le pire cas, mais en maximisant la flexibilité de la séquence de groupes dans un second objectif, on peut générer une séquence de groupes à flexibilité maximale respectant une garantie sur l'objectif, ce qui est une approche classique de la littérature, alternative à celle proposée dans ce chapitre [Kouvelis and Yu, 1997, Gu et al., 2009, van den Akker et al., 2013, Hao et al., 2013, Horng and Lin, 2015, Wang et al., 2018, 2019, Ghasemi et al., 2021]. Cette approche alternative propose en effet un ensemble de solutions pour la phase réactive mais sans évaluer a priori la politique de second niveau. L'extension du calcul du score dans le pire des cas d'une séquence partiellement admissible permet ainsi d'évaluer la pertinence de l'approche de maximisation de la flexibilité par rapport à l'approche proposée dans cette thèse sur les séquences partiellement admissibles.

L'algorithme pire cas fonctionne par le calcul de plus longs chemins dans le graphe de pire cas. En effet, le plus long chemin du nœud source au nœud de fin de tâche i' donne la date de fin pire cas de la tâche i parmi tous les ordonnancements associés à une séquence représentée par une GSEQ, pour un scénario donné.

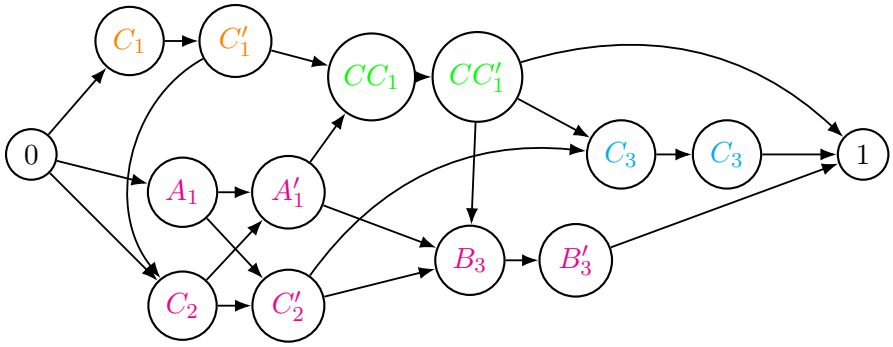
Algorithme 4.1 Algorithme de pire cas généralisé

1: Prendre $\mathcal{G}$ le graphe de précédence associé à π
2: **si** $\mathcal{G}$ contient un circuit **alors**
3: **retour** $\perp$ %π *est pleinement inadmissible*
4: Ajouter les arcs non orientés des groupes de π à $\mathcal{G}$
5: Calculer les composantes fortement connexes C de $\mathcal{G}$
6: Séparer les composantes de C contenant seulement des tâches d'une même machine
7: Construire le graphe de pire cas de π agrégé par C
8: **retour** Les dates de fin dans le pire cas d'après Artigues et al. [2005]

L'algorithme n'est plus applicable dans le cas d'une GSEQ partiellement inadmissible et partiellement admissible (que nous considérons valide pour la stratégie GSEQ), car des circuits peuvent être présents dans le graphe pire cas (l'algorithme de plus long chemin ne peut plus y être exécuté). Pour résoudre ce problème, nous décrivons dans ce qui suit un algorithme calculant une borne sur la date de fin dans le pire cas des tâches, compatible avec notre utilisation des séquences de groupes partiellement inadmissibles.



(a) Graphe disjonctif et ses composantes fortement connexes (simplifié)



(b) Graphe de pire cas agrégé

FIGURE 4.5 – Illustration de l'algorithme de pire cas étendu (4.1) pour $\pi = [[\{C_1\}, \{B_2, A_2\}], [\{A_3\}, \{B_1\}, \{C_3\}], [\{A_1, C_2\}, \{B_3\}]]$

L'algorithme 4.1 commence par construire le graphe de précédence $\mathcal{G}$ associé à la séquence de groupes π considérée (étape 1). $\mathcal{G}$ ne contient que les contraintes de précédence imposées par l'ordre des groupes de π et celles des tâches d'un job, donc les tâches au sein d'un même groupe ne sont liées par aucun arc. $\mathcal{G}$ contient un circuit ssi π est pleinement inadmissible (voir théorème 4.2.1.2), auquel cas l'algorithme s'arrête : il n'y a pas de "pire cas" puisque aucune séquence admissible n'est représentée.
Les arcs non orientés représentant les disjonctions possibles sont ensuite ajoutés

entre les tâches d'un même groupe (étape 4), résultant en un graphe disjonctif. L'algorithme de Tarjan est utilisé pour obtenir la liste C des composantes fortement connexes du graphe (étape 5). Cette étape est illustrée par un exemple présenté dans la figure 4.5a, qui représente le graphe disjonctif associé à la GSEQ $\pi = [[\{C_1\}, \{B_2, A_2\}], [\{A_3\}, \{B_1\}, \{C_3\}], [\{A_1, C_2\}, \{B_3\}]]$, pour le JSP avec 3 machines et 3 jobs du problème P2 (figure 3.2) (certaines contraintes de précédence sont omises dans un souci de clarté). L'analyse de composantes fortement connexes donne deux composantes à plus d'un élément (CC_1 et CC_2) et des composantes contenant un seul élément.
Ensuite, la liste C des composantes fortement connexes est modifiée pour ne conserver que les composantes à plus d'un élément qui contiennent des tâches provenant d'au moins deux machines différentes (étape 6). Dans notre exemple, seule la composante CC_1 est conservée telle quelle. Les autres composantes sont séparées. Si, à l'issue de cette étape de séparation, aucune composante groupée ne subsiste, et que la liste C ne contient que des composantes avec un unique élément, alors la GSEQ est pleinement admissible (dans notre exemple, la GSEQ est partiellement inadmissible). Enfin, le graphe pire cas agrégé est construit (étape 7) en utilisant les composants de C (voir la figure 4.5b). Les tâches de chaque composante c sont regroupées en une tâche unique qui hérite de toute les contraintes de précédence des tâches qui la composent.

Proposition Le graphe de pire cas agrégé sans arcs intra-groupe est sans circuit.

Démonstration. Par construction, puisque les nœuds-tâches impliquées dans des circuits forment des composants fortement connexes, ils ont été agrégés à l'étape (étape 5). Puis, n'ont été désagrégés que les cycles sur une unique machine, dûs aux arcs disjonctifs, qui n'introduisent pas de circuits après que les arcs intra-groupes aient été enlevés. □

La date de fin dans le pire cas des tâches contenues dans un regroupement sont fixées à la date de fin dans le pire cas du regroupement entier, et la durée d'exécution du regroupement est la somme des durée d'exécution des tâches qui le composent.

Proposition 4.2.2.1. *Le plus long chemin de la racine au nœud de fin des tâches est une borne supérieure de la date de fin dans le pire cas de cette tâche.*

Démonstration. Puisque la GSEQ est partiellement admissible (étape 2), il existe une façon d'ordonnancer les tâches pour obtenir une solution admissible. Puisque les regroupements héritent des contraintes de séquencement des machines des tâches qui la composent, les tâches ne peuvent commencer que lorsque toutes les machines impliquées sont libres, et le temps d'exécution égal à la somme des durées des tâches qui composent le regroupement implique une absence totale de parallélisation : les tâches sont exécutées les unes à la suite des autres, dans un ordre convenable

(dont on sait l'existence). En fixant la date de fin dans le pire cas des tâches d'un regroupement à le date de fin de toutes les tâches de ce regroupement, on s'assure d'obtenir une borne supérieure pour la date de fin pire cas. Par propagation dans le graphe pire cas, la date de fin pire cas des autres tâches est aussi une borne supérieure sur leurs dates de fin dans le pire cas. □

Le regroupement dégrade la qualité de la borne obtenue pour la date de fin dans le pire cas, mais on note que si aucun regroupement n'est nécessaire (quand la GSEQ est pleinement admissible), le score pire cas obtenu est identique à l'algorithme originel de pire cas. On remarque aussi que pour le 1P, l'étape (étape 6) sépare toujours les regroupements, donc aucune perte n'est introduite. On retrouve le résultat de Aloulou et al. [2004] établissant l'existence d'algorithmes polynomiaux pour la maximisation d'objectifs de maximisation des dates de fin de tâches pour les problèmes à une machine avec des contraintes de précédence et dates de disponibilité.
La complexité de cet algorithme est la même que celle de l'algorithme originel ($O(|N|^2)$). Bien que nous ne proposions qu'une borne sur la date de fin dans le pire cas, la question de savoir si le calcul de la date de fin dans le pire cas est un problème NP-difficile pour le JSP est ouverte.

Une étape supplémentaire de généralisation est nécessaire pour calculer la date de fin dans le pire cas compte tenu des multiples scénarios que nous considérons. Nous pouvons, en appliquant l'algorithme décrit précédemment à chaque scénario, obtenir une borne sur la date de fin de chaque tâche dans le pire cas pour chaque scénario, que nous pouvons agréger en utilisant $\oplus$ et obtenir une borne sur l'objectif agrégé, avec une complexité $O(|S|.|N|^2)$. Nous avons ainsi un outil nous permettant par exemple de maximiser la flexibilité d'une séquence de groupes partiellement admissible en imposant une borne sur la fonction objectif dans le pire des cas.

4.3 Calcul de solutions

La stratégie GSEQ domine les stratégies JSEQ et FIFO en théorie, mais la question de sa performance en pratique doit être établie par l'intermédiaire de son implémentation. Dans ce qui suit, nous présentons des méthodes existantes et de nouvelles méthodes pour l'implémentation de la stratégie GSEQ à deux niveaux. Ces méthodes seront ensuite comparées entre elles dans la section 4.4.

4.3.1 Modèle de programmation par contraintes

A notre connaissance, la seule méthode exacte existante pour la résolution du problème de la stratégie GSEQ à deux niveaux que nous considérons est le modèle PLNE proposé par Artigues et al. [2016], Cheref et al. [2016a,b], qui ne résout que de très petites instances. L'efficacité des approches par contraintes pour les problèmes

d'ordonnancement de grande taille suggère l'utilisation de la programmation par contrainte (PPC).

Nous proposons dans cette section un modèle de PPC implémentant la stratégie GSEQ pour les problèmes 1P et JSP. Dans l'esprit du modèle JSEQ (cf algorithme 3.1 et 3.2), chaque tâche peut avoir une date de début différente dans chaque scénario, mais elles doivent être consistantes avec une solution GSEQ, identique dans tous les scénarios. Les modèles PPC qui suivent (Modèles 4.1 et 4.2) décrivent les contraintes clés permettant le calcul de la GSEQ, respectivement pour le 1P et le JSP, en utilisant les contraintes de **CP Optimizer**. Il faut remarquer que la simulation de la politique FIFO sur l'ensemble de scénarios d'entraînement est intégrée aux modèles.

1: **pour** $s \in S$ **faire**
2: $\quad$ `StartOf`$(\text{JOB}[i,s]) \geq r_i^s \quad \forall i$
3: $\quad$ `NoOverlap`(`Sequence`$(\text{JOB}[:,s]))$
4: $\quad$ **pour** $(i,j) \in N^2$ **faire**
5: $\quad\quad$ $(i,j) \in E \rightarrow g_i < g_j$ $\%g_i$ *is the group number of job* i
6: $\quad\quad$ $g_i < g_j \rightarrow$ `EndBeforeStart`$(\text{JOB}[i,s], \text{JOB}[j,s])$
7: $\quad\quad$ **si** $r_i^s \leq r_j^s$ **alors**
8: $\quad\quad\quad$ $g_i == g_j \rightarrow$ `EndBeforeStart`$(\text{JOB}[i,s], \text{JOB}[j,s])$
9: $\quad\quad$ **sinon**
10: $\quad\quad\quad$ $g_i == g_j \rightarrow$ `EndBeforeStart`$(\text{JOB}[j,s], \text{JOB}[i,s])$
11: **pour** $g \in N$ **faire**
12: $\quad$ $u_g = \vee(g_i == g, \forall i \in N)$ $\%$ u_g *marks a group as "used"*
13: $\quad$ **si** $g > 0$ **alors**
14: $\quad\quad$ $u_{g-1} \geq u_g$

Modèle 4.1 – Modèle PPC de calcul de solution GSEQ pour le 1P

Le modèle 4.1 contient $O(|S|.|N|)$ variables entières, et $O(|S|.|N|^2)$ contraintes. Les variables g_i sont des entiers décrivant l'index du groupe contenant la tâche i ($g_i = k$ signifiant que la tâche i est dans le k-ième groupe de la séquence de groupe) ; $\text{JOB}[i,s]$ est une variable d'intervalle représentant les dates de début et de fin d'exécution de la tâche i dans le scénario s ; u_g est une variable booléenne marquant l'utilisation du groupe g.
Les lignes (2 - 10) décrivent le modèle CPO pour chaque scénario. Pour un scénario fixé, les données sont déterministes, ce qui permet de fixer les contraintes de disponibilité (ligne 2). Toutes les tâches sont rassemblées dans une variable de séquence, et la contrainte de non chevauchement `NoOverlap` est appliquée ligne (3). Ensuite, chaque paire de tâche est considérée. Si elles sont liées par une contrainte de précédence (i,j), le groupe de la tâche i doit être d'index inférieur [1] à l'index du groupe de la tâche j ligne (5). A l'inverse, si une tâche i est dans un groupe d'index

1. Strictement pour obtenir une GSEQ pleinement admissible, ou au sens large pour partiellement admissible

inférieur au groupe d'une tâche j, la contrainte ligne (6) force la précédence $i \prec j$. Cette contrainte synchronise les séquences de chaque scénario avec la GSEQ définie par les variables g. Si deux tâches sont dans le même groupe, il faut les ordonner selon la politique FIFO de second niveau, donc la tâche disponible en premier est ordonnancée en premier lignes (7 - 10).

Seul l'ordre des groupes utilisé est important, et non leurs indices, il est donc possible de briser une symétrie en forçant l'utilisation des groupes de plus petits index en priorité. Les lignes (11-14) utilisent les variables booléennes u à cette fin. La ligne (12) donne la valeur "1" à la variable u_g ssi le groupe g est utilisé, c'est-à-dire s'il existe une tâche dont le groupe est g. Les contraintes des lignes (13-14) ne permettent à un groupe g d'être utilisé que si $g = 0$ ou si le groupe $g-1$ est utilisé. Ainsi, si k groupes sont utilisés, ce seront les k premiers (les groupes 0 à $k-1$).

1: **pour** $s \in S$ **faire**
2: $\texttt{StartOf}(\textsc{Job}[m_{i,j-1}, i, s]) \geq \texttt{EndOf}(\textsc{Job}[m_{i,j}, i, s]) \quad \forall i \in N, \forall j \in M \setminus \{0\}$
3: $\texttt{NoOverlap}(\texttt{Sequence}(\textsc{Job}[m, :, s])) \forall m \in M$
4: $\textsc{Ready}[m_{i,0}, i, s] == 0 \quad \forall i \in N$
5: $\textsc{Ready}[m_{i,j}, i, s] == \texttt{EndOf}(\textsc{Job}[m_{i,j-1}, i, s] \quad \forall i \in N, \forall j \in M \setminus \{0\}$
6: **pour** $m \in M$ **faire**
7: **pour** $(i, j) \in N^2$ **faire**
8: $g_i^m < g_j^m \rightarrow \texttt{EndBeforeStart}(\textsc{Job}[m, i, s], \textsc{Job}[m, j, s])$
9: **si** $\textsc{Ready}[m, i, s] \leq \textsc{Ready}[m, j, s]$ **alors**
10: $g_i^m == g_j^m \rightarrow \texttt{EndBeforeStart}(\textsc{Job}[m, i, s], \textsc{Job}[m, j, s])$
11: **sinon**
12: $g_i^m == g_j^m \rightarrow \texttt{EndBeforeStart}(\textsc{Job}[m, j, s], \textsc{Job}[m, i, s])$
13: **pour** $m \in M$ **faire**
14: $\texttt{StartOf}(\textsc{Jobs}[m, i, s]) == \texttt{max}(\texttt{EndOfPrev}(\texttt{Sequence}[m],$
$\textsc{Jobs}[m, i, s]), \textsc{Ready}[m, i, s]), \forall i \in N$
15: **pour** $m \in M$ **faire**
16: **pour** $g^m \in N$ **faire**
17: $u_g^m = \vee(g_i^m == g^m, \forall i \in N)$
18: **si** $g^m > 0$ **alors**
19: $u_{g-1}^m \geq u_g^m$

Modèle 4.2 – Modèle PPC de calcul de solution GSEQ pour le JSP

La structure du problème JSP implique quelques modifications du modèle PPC par rapport au modèle du problème 1P. Il est décrit dans le modèle 4.2 et contient $O(|N|.|M|)$ variables entières, $O(|S|.|N|.|M|)$ variables continues et $O(|S|.|M|.|N|^2)$ contraintes.
Une variable g_i^m représente l'index du groupe de chaque job i sur chaque machine m. Comme précédemment, un modèle est créé pour chaque scénario, et synchronisé par l'intermédiaire des variables g. Les contraintes de la ligne (2) correspondent aux

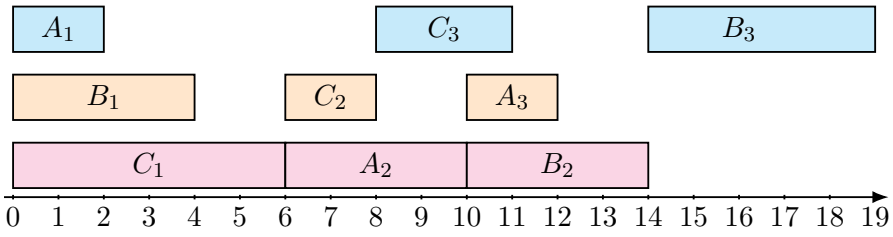
contraintes de précédence entre les tâches d'un job. Une variable de séquence est créée pour chaque machine dans chaque scénario et la contrainte `NoOverlap` assure le non-chevauchement des tâches sur chaque machine (3). Des variables supplémentaires $\textsc{Ready}[m, i, s]$ sont nécessaires. $\textsc{Ready}[m, i, s]$ correspond à la date à laquelle la tâche du job i sur la machine m est prête dans le scénario s : lorsque la tâche précédente du même job est terminée (4-5). Pour chaque machine m et chaque paire de job (i, j), la politique FIFO est simulée pour séquencer les tâches dans l'ordre attendu conformément aux valeurs des variables g_i^m et g_j^m, similairement au cas du 1P (6-15). A la différence du 1P, ce sont les variables Ready qui sont utilisées pour déterminer l'ordre des tâches d'un groupe. Remarquons que le modèle JSP peut produire des solutions partiellement invalides.
Les contraintes (16-18) doivent être présentes pour assurer l'ordonnancement semi-actif des tâches. En leur absence, CPO pourrait retarder l'exécution d'une tâche pour influencer les valeurs des variables Ready, et par conséquent, les décisions futures d'ordre de la politique FIFO au sein des groupes, ce qui va à l'encontre de l'hypothèse de non-anticipation. Le théorème suivant établit formellement ce résultat :

Théorème 4.3.1.1. *Pour une solution GSEQ et la politique FIFO de second niveau, les ordonnancements semi-actifs ne sont pas dominants dans la minimisation du C_{MAX}.*

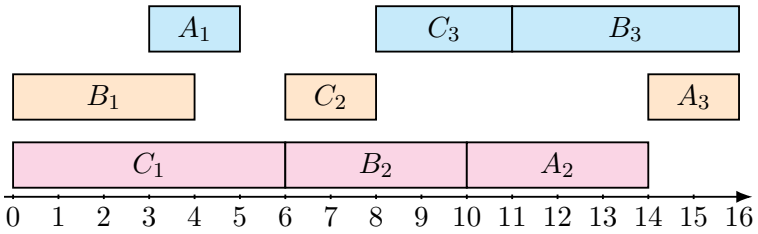
Démonstration. La figure 4.6 propose un contre-exemple qui illustre le comportement indésirable qui peut survenir lorsque les contraintes (16-18) sont absentes, dans lequel un ordonnancement semi-actif n'est pas dominant (on lui préfère un ordonnancement non semi-actif). La figure 4.6a correspond au comportement attendu pour une GSEQ $\pi = [[\{A_1\}, \{C_3, B_3\}], [\{B_1\}, \{C_2\}, \{A_3\}], [\{C_1\}, \{A_2, B_2\}]]$ pour un problème avec 3 jobs et 3 machines : les tâches A_1 et B_2 sont lancées sans délai, donc la tâche A_1 termine en premier, et par conséquent, lors de la considération du groupe $G_2^3 = \{A_2, B_2\}$, la politique FIFO ordonnance A_2 en premier. Toutefois, cette décision s'avère pénalisante pour la valeur objectif, car lancer la tâche B_2 en premier permet d'atteindre une meilleure date de fin C_{MAX}. Anticipant cela, CPO peut retarder la tâche A_1 comme dans la figure 4.6b afin qu'elle termine après B_1, et par conséquent permettre à la politique FIFO de lancer B_2 avant A_2, et d'atteindre la meilleure valeur de C_{MAX}. □

Une politique de décision en temps réel qui pourrait retarder des tâches pour améliorer la solution de cette façon devrait avoir connaissance d'informations futures du scénario (la durée d'exécution des tâches A_1 et B_1 dans l'exemple), ce qui viole notre modèle d'information : les durée d'exécution des tâches sont révélées lorsqu'elles se terminent.
Par conséquent, un calage à gauche "manuel" est imposé dans le modèle. L'expression `EndOfPrev`(seq,i) représente la date de fin de la tâche qui précède une tâche i dans une séquence seq. Donc la contrainte ligne (17) force le lancement dès que

possible en fixant la date de début des tâches au maximum entre la date de fin de la tâche précédente sur la machine, et la fin de la tâche précédente du job.



(a) Ordonnancement semi-actif



(b) Ordonnancement non semi-actif

FIGURE 4.6 – Exemple de non dominance des ordonnancement semi-actifs

4.3.2 Heuristiques avec démarrage à chaud

Nous verrons dans la section 4.4, que bien que la stratégie GSEQ domine en théorie la stratégie JSEQ, dans un temps limité, pour les instances de taille moyenne et grandes, le modèle PPC implémentant la stratégie JSEQ obtient de bien meilleurs résultats que le modèle PPC de la stratégie GSEQ. Pour tirer parti à la fois de la bonne performance du modèle JSEQ, mais aussi de la supériorité théorique des GSEQ, nous proposons des heuristiques "à chaud" implémentant la stratégie GSEQ en utilisant comme point de départ une solution JSEQ (qui peut être vue comme un cas particulier de solution GSEQ) de bonne qualité. Nous comparerons l'implémentation de la stratégie JSEQ dans un temps donné avec plusieurs heuristiques consistant à utiliser une partie du temps disponible dans une stratégie JSEQ, et le reste dans une heuristique GSEQ "à chaud" utilisant le produit de la stratégie JSEQ.

Nous comparons trois méthodes à chaud :
- L'heuristique gloutonne de Esswein [2003] (EW-GSEQ).
- Un algorithme tabou inspiré de Artigues et al. [2016] (TAB-GSEQ).
- Un nouvel algorithme génétique (GA-GSEQ).

De plus, ces méthodes sont comparées pour les paramètres suivants :
- L'**optimisation pire-cas** (WC = $\top$ [2]) : l'optimisation pire cas vise à produire au premier niveau la GSEQ qui minimise le score de la pire séquence représentée pour chaque scénario. Dans ce cas, la robustesse est apportée

2. On note $\top$ pour la valeur booléenne "Vrai" et $\bot$ pour la valeur "Faux".

par la maximisation d'un critère secondaire de flexibilité (voir point suivant). C'est l'approche la plus courante dans la littérature sur le calcul de GSEQ [Esswein, 2003, Pinot, Cardin, and Mebarki, 2007, Cardin, Mebarki, and Pinot, 2013]. On rappelle que le score pire-cas d'une séquence de groupes (ou une borne sur ce score) est obtenu en temps polynomial en utilisant l'algorithme décrit dans la section 4.2.2.
Quand l'optimisation n'est pas pire cas (WC = $\bot$), la décision de premier niveau prend en compte l'utilisation de l'heuristique FIFO au second niveau pour simuler le comportement et évaluer une GSEQ. Le premier niveau produit la GSEQ qui minimise le score obtenu par la politique FIFO sur les scénarios d'entraînement. C'est l'approche utilisée par Cheref et al. [2016b] et reprise dans ce chapitre.

— Utiliser un **objectif secondaire de flexibilité** : Dans le cas où on utilise l'optimisation pire cas, si l'on ne considère pas un objectif secondaire de flexibilité (OF = $\bot$), alors les simples séquences dominent les solutions GSEQ. Il faut donc explicitement maximiser la flexibilité (OF=$\top$) pour obtenir une solution dont le pire cas est garanti, mais dont la flexibilité peut être avantageuse lors de l'évaluation sur les scénarios de tests. Dans le cas où on utilise l'optimisation FIFO, il peut aussi être intéressant de déterminer si la maximisation de la flexibilité est avantageuse.
— Tolérer les **solutions partiellement inadmissibles** : pour le JSP, nous étudions l'impact de la définition étendue de validité d'une GSEQ (TPI=$\top$) qui ne sont habituellement pas tolérées, comparée à la définition classique de la littérature nécessitant la pleine admissibilité des solutions produites (TPI=$\bot$).
— Le **temps de chauffe** (INIT) : le temps consacré à la résolution de la stratégie JSEQ. Le temps restant est le temps alloué à la méthode à chaud de la stratégie GSEQ.

4.3.2.1 Algorithme tabou : TAB-GSEQ

Nous proposons une méthode tabou inspirée de l'algorithme tabou de Artigues et al. [2016], Cheref et al. [2016b] dans laquelle nous utilisons une solution JSEQ pour le démarrage à chaud, et non une solution GSEQ obtenue par une heuristique gloutonne. La solution JSEQ initiale est transformée en solution séquence de groupes GSEQ équivalente, ne contenant qu'une seule tâche dans chaque groupe. La solution courante est représentée par une liste de groupes de tâches.
La méthode initiale de Artigues et al. [2016], Cheref et al. [2016b] explore 4 voisinages à chaque itération correspondant aux mouvements suivants :

— Échange de groupe : échange des positions d'une paire de groupes parmi tous les groupes (toutes les tâches d'un groupe i appartiennent désormais au groupe j et vice versa)
— Insertion de groupe : Déplacement d'une unique tâche d'un des groupes vers un autre groupe, ou dans un nouveau groupe situé entre deux groupes

existants.
— Division de groupe : Division d'un des groupes G en deux groupes G^- et G^+. Il y a $2^{|G|}$ façons de diviser un groupe en deux, donc pour réduire la taille de ce voisinage, une métrique est associée à chaque tâche et ne sont considérées que les $|G|$ divisions dans lesquelles les tâches de G^- ont une métrique inférieure à celles de G^+ (en d'autres termes, les tâches sont divisées selon l'ordre défini par la métrique). La métrique utilisée originellement est la date moyenne de livraison (un paramètre incertain dans les travaux originels). L'idée étant qu'il est préférable d'avoir les tâches les plus urgentes dans G^-.
— Fusion de groupes : fusion de deux groupes consécutifs de la solution.

Une liste tabou de taille $10 * |N|$ est utilisée pour stocker les solutions atteintes précédemment mais les expériences menées dans Artigues et al. [2016] montrent un impact limité de la liste tabou sur la performance de l'heuristique. La procédure tabou termine lorsque les conditions d'arrêt sont atteintes.

Des tests préliminaires ont montré que l'utilisation du voisinage d'insertion de groupe uniquement était préférable à l'utilisation simultanée de tous les voisinages décrits. Notre implémentation de l'algorithme tabou n'utilise donc que ce voisinage. A chaque itération, le meilleur voisin non-tabou devient la solution courante. Pour accélérer l'exploration du voisinage, nous utilisons aussi un paramètre de diversification, proportionnel au nombre d'itérations sans amélioration, qui définit à quelle distance du groupe d'origine (en terme de nombre de groupes) on considère l'insertion des tâches. Ainsi, on ne considère d'abord que le voisinage composé de solutions pour lesquelles les tâches sont insérées dans un groupe voisin, puis l'on considère des voisins de plus en plus distants (5 itérations sans améliorations augmentent la portée des insertions de 1). L'algorithme s'arrête à la fin du temps imparti ou après 1000 itérations sans améliorations du meilleur score.

Notons aussi que l'algorithme tabou évoqué dans le chapitre 3 pour l'implémentation de la stratégie JSEQ est une restriction de l'algorithmes TAB-GSEQ aux insertions entre les groupes (ce qui correspond aux voisinages classiques *"swap"* lorsque la diversification est minimale, et "insertion" lorsqu'elle est plus importante).

4.3.2.2 Greedy heuristic : EW-GSEQ

L'heuristique gloutonne EW-GSEQ, intitulée "EBGJ" dans Esswein [2003], prend une JSEQ en entrée. Puis, une descente est effectuée en considérant seulement les fusions de groupes. Les groupes fusionnés sont choisis pour minimiser le score (ou ne pas le détériorer quand on considère le score pire cas) et fusionnent en priorité les groupes plus petits. La procédure termine lorsque aucune fusion n'est avantageuse, ou au temps limite. La procédure est décrite dans l'algorithme 4.2.

Algorithme 4.2 Schéma de l'heuristique gloutonne EW-GSEQ

1: Depuis une solution de départ x
2: **tant que** il existe des voisins non-déteriorants et dans le temps limite imparti **faire**
3: Calculer l'ensemble $\mathcal{N}$ des voisins "fusion de groupe" valides .
4: la solution courante x devient le meilleur voisin (de score minimal, dont la taille du plus grand groupe est minimale).
5: **retour** x

4.3.2.3 Algorithm génétique : GA-GSEQ

L'algorithme génétique que nous implémentons reprend la structure classique décrite dans l'algorithme 1.2.

Algorithme 4.3 Schéma d'exécution de l'algorithme GA-GSEQ

1: $Population \leftarrow InitPopulation(PopSize)$
2: **tant que** $(time < MaxTime) \wedge (stagnate < MaxStagnate)$ **faire**
3: $A, B \leftarrow Select2(Population)$
4: $C \leftarrow Crossover(A, B)$
5: **si** $random() < MutateRate$ **alors**
6: $C \leftarrow Mutate(C)$
7: **si** $Invalid(C) \wedge (random() < RepairRate)$ **alors**
8: $C \leftarrow Repair(C)$
9: **si** $random() < EducateRate$ **alors**
10: $C \leftarrow Educate(C)$
11: $Population \leftarrow Population + C$
12: **si** $PopulationLimit < Population$ **alors**
13: $Population \leftarrow SelectSurvivors(Population)$

Dans l'algorithme 4.3, la population initiale contient la solution JSEQ du démarrage à chaud, ainsi que $\alpha_p - 1$ solutions GSEQ générées aléatoirement (étape 1). Ensuite, à chaque itération, deux individus sont choisis aléatoirement et croisés en utilisant une extension de l'opérateur de croisement à un point (voir section 1.1), adapté aux GSEQ, et illustré dans la figure 4.8 : les groupes de π_A sont préservés par l'opérateur jusqu'au point de croisement X, les tâches restantes sont ensuite ajoutées en conservant l'ordre imposé par π_B (étape 3 et 4). L'individu résultant du croisement est muté avec une probabilité α_m, par un mouvement aléatoire dans les voisinages décrits dans la section 4.3.2.1 (étape 6).

L'individu est ensuite réparé s'il est invalide avec une probabilité α_r (étape 8). La procédure de réparation transforme une solution invalide en une solution valide tout en conservant au mieux ses caractéristiques (l'ordre des tâches qu'elle prescrit). Pour le 1P, la réparation consiste à itérer sur les tâches dans l'ordre défini par le GSEQ, et chaque tâche j rencontrée impliquée dans une contrainte de précédence (i, j) violée est repoussée à une position acceptable dans la séquence.

	G_1	G_2	G_3
M_1	$[C_1,B_2]$	$[A_2]$	
M_2	$[A_3]$	$[B_1]$	$[C_3]$
M_3	$[A_1,C_2]$	$[B_3]$	
M	$[C, B, A, A, C]$	$[A, B, B]$	$[C]$
M_1	$[C_1,A_2]$	$[B_2]$	
M_2	$[B_1]$	$[A_3]$	$[C_3]$
M_3	$[A_1,C_2]$	$[B_3]$	

FIGURE 4.7 – Réparation d'une GSEQ pleinement inadmissible pour le JSP

Pour le JSP, la procédure, illustrée dans la figure 4.7, est un peu plus complexe : les groupes au même index sur chaque machine sont fusionnés dans un unique "super-groupe" de tâches, et seul le job de chaque tâche est conservé, puis, les jobs du super-groupe sont insérés dans des groupes sur les machines dans l'ordre des tâches des jobs. Les groupes de chaque machine sont fermés, et la procédure est itérée avec les groupes suivants, jusqu'à ce que tout les groupes soient fusionnés et divisés de cette manière.

Proposition 4.3.2.1. *La GSEQ produite à l'issue de cette procédure de réparation est partiellement admissible.*

Démonstration. Puisque les tâches de chaque job sont insérées dans l'ordre croissant, une tâche dans un groupe donné n'aura pas de prédécesseur dans un groupe d'index strictement supérieur. Par conséquent, il est toujours possible d'ordonnancer toutes les tâches des groupes d'un index donné, puis celles de l'index suivant, et ainsi de suite, jusqu'à l'ordonnancement complet de toutes les tâches. □

L'individu est éduqué avec une probabilité α_e par descente (étape 10). La procédure d'éducation applique le premier mouvement améliorant dans le voisinage d'insertion de groupe, jusqu'à ce qu'aucun mouvement ne soit améliorant, pour un maximum de 100 mouvements. Enfin, l'individu rejoint la population (étape 11). Si celle ci dépasse la taille maximale autorisée α_{max}, la moitié des individus les moins bons en sont exclus (étape 13). L'algorithme se poursuit ainsi jusqu'à la date limite, ou si aucune amélioration n'est obtenue durant 1000 d'itérations.

Les méta-paramètres de l'algorithme ont été fixés empiriquement par un test en grille préliminaire. pour le 1P et le JSP, ces paramètres sont fixés à (α_p, α_{max}, α_m, α_e, α_r) = (10, 20, 0.5, 0.01, 0.5).

Remarquons que l'algorithme GA-JSEQ (l'algorithme génétique implémentant la stratégie JSEQ) est, de même que pour l'algorithme tabou, une restriction du GA-GSEQ. Les voisinages de division et de fusion de groupe ne s'appliquent pas lors de la phase de mutation dans ce cas. On remarque que la procédure de croisement dans ce cas reviens à l'opérateur classique de croisement à un point présenté dans la figure 1.1.

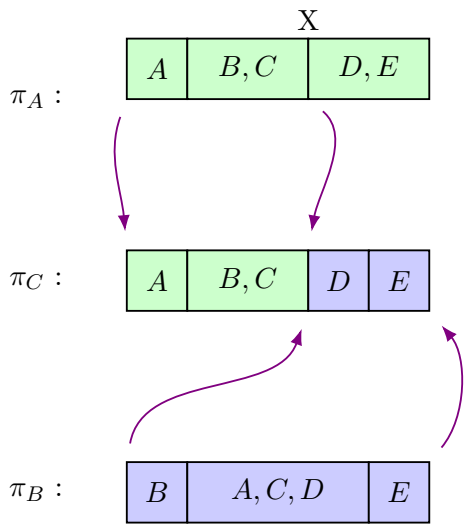


FIGURE 4.8 – Le croisement à un point version GSEQ

4.4 Résultats

Nous présentons dans cette section les résultats des expérimentations visant à évaluer les performances des différentes méthodes implémentant la stratégie GSEQ, et à la comparer aux stratégies de références du chapitre 3.

4.4.1 Récapitulatif des expériences

Toutes les instances décrites dans la section 3.2 sont résolues par les différentes méthodes présentées en se basant sur les instances d'entraînement, dans une limite de temps de 2 heures. Les instances sont résolues pour les 4 objectifs $\oplus$ γ présentés, et pour différentes combinaisons de paramètres (WC$\in\{\bot,\top\}$, OF$\in\{\bot, \top\}$, TPI$\in\{\bot, \top\}$ et INIT vaut 0 si le démarrage à chaud n'est pas utilisé, ou une valeur en minute indiquant la durée de chauffe). Puisque la méthode CP-JSEQ est un bon représentant de la stratégie JSEQ (voir section 3.4), nous utilisons exclusivement cette méthode pour la chauffe.

La table 4.1 résume les différentes configurations testées sur les instances décrites. C1 correspond aux méthodes sans démarrage à chaud. C2 évalue les méthodes avec démarrage à chaud (indiqué par "*") pour différents temps de chauffe. C3 évalue les méthodes avec 90 minutes de chauffe, pour différentes valeurs de WC, OF [3] et TPI.

Sauf indication spécifique, les paramètres par défaut des méthodes sont ¬WC, ¬OF (OF pour les méthodes à chaud), TPI(¬TPI pour le 1P), INIT = 0 (90 pour

3. OF=$\bot$ n'est pas considéré pour EW-GSEQ

Conf.	Méthode X-SEQ	Valeur des paramètres			
		WC	OF	TPI	INIT
C1	{CP-G,AG-G,TAB-G}	$\perp$	$\perp$	$\perp$ (1P) | $\top$ (JSP)	0
C2	{AG-G*,EW-G*,TAB-G*}	$\perp$	$\perp$	$\perp$ (1P) | $\top$ (JSP)	{1,2,5,10,30 60, 90, 110}
C3	{AG-G*,EW-G*,TAB-G*}	{$\top$,$\perp$}	$\top$	$\perp$ (1P) | $\top$ (JSP)	90
C4	{AG-G*,EW-G*,TAB-G*}	$\perp$	$\perp$	$\top$ (1P) | $\perp$ (JSP)	90

TABLE 4.1 – Méthodes évaluées pour le 1P et le JSP

les méthodes à chaud).

Comme précédemment, toutes les méthodes s'exécutent sur un unique processeur (Xeon E5-2695 v4 @ 2.10GHz). Les modèles de PPC sont résolus par IBM CP Optimizer 20.1. Pour les méthodes à chaud, la solution obtenue par la méthode CP-JSEQ après INIT est utilisé comme entrée pour le temps restant.
Les heuristiques sont implémentées en Python, tandis que les modèles PPC utilisent l'API Python de CPO.

Les méthodes sont exécutées sur les instances d'entraînement et évaluées sur les instances d'entraînement et de tests selon les métriques décrites dans la section 3.2. Les tables A.1, A.2, A.3, A.4, A.5, A.6 et A.7 présentent les performances généralisées des méthodes pour différents paramètres d'instances. Les tables A.8, A.9, A.10, A.11, A.12, A.13, et A.14 présentent l'impact du paramètre WC en fonction des paramètres d'instances. Les tables A.15,A.16, A.17, A.18 et A.19 quant à elles présentent l'impact du paramètre TPI pour différentes valeurs de Δ et différents objectifs. Enfin, la table A.20 présente l'impact du paramètre OF en fonction de la densité des contraintes de précédence.

4.4.2 Comparaison des méthodes CP-JSEQ et CP-GSEQ

Les tables A.1, A.2, A.3, A.4, A.5, A.6 et A.7 permettent de comparer les résultats obtenus par les deux méthodes PPC : CP-GSEQ et CP-JSEQ. Les données montrent que dans l'ensemble, la méthode CP-JSEQ obtient de bien meilleurs résultats que la méthode équivalente GSEQ. La seule exception survenant sur les plus petites instances sur le 1P, pour lesquelles CP-GSEQ obtient de meilleurs résultats (voir par exemple les colonnes CP-GSEQ and CP-JSEQ dans la table A.3 pour $N = 10$). Ces résultats sont cohérents avec la dominance théorique de la stratégie GSEQ sur JSEQ : lorsque la méthode est capable de résoudre les instances à l'optimum les résultats de la stratégie GSEQ devraient être au moins aussi bons que ceux de la stratégie JSEQ. Le phénomène n'apparaît pas pour le JSP, dans lequel, même pour les plus petites instances, le CP-GSEQ n'est parfois pas capable de résoudre les instances à l'optimum (colonnes CP-GSEQ et CP-JSEQ de la table A.4).

On peut remarquer dans les tables A.1 et A.2 que lorsque la variabilité entre les scénario augmente (Δ est plus grand), vient un point ou la méthode CP-GSEQ devient plus intéressante que CP-JSEQ ($\Delta = 0.5$ pour le 1P et $\Delta = 0.7$ pour le JSP). Comme le montre les bons résultats de la stratégie FIFO sur les mêmes instances, lorsque la variabilité devient trop importante, l'accent mis sur la phase réactive donne de meilleurs résultats. En effet, la méthode CP-GSEQ commence en général par trouver une solution très flexible (proche de la solution FIFO), et arrive rarement à l'améliorer en réduisant le nombre de groupes, mais ces solutions sont déjà de bonne qualité quand la variabilité est importante.

L'analyse comparée des différents objectifs (tables A.5 et A.6) révèle que CP-JSEQ domine pour tout les objectifs sauf le $max \sum C_i$ pour le JSP, tandis que pour le 1P, CP-GSEQ obtient de meilleurs résultats pour l'objectif $\sum C_i$ mais de moins bons résultats pour l'objectif L_{MAX}. Toutefois pour ces objectifs, la stratégie FIFO obtient aussi de remarquables performances. Ces résultats montrent la difficulté d'obtenir des solutions proactives pour optimiser la somme des temps de fin des tâches dans un contexte robuste.

Pour le 1P, la densité des contraintes de précédence (table A.7) a un impact important sur la performance relative de CP-GSEQ et CP-JSEQ : lorsque la densité est proche de 0, CP-GSEQ obtient de relativement meilleurs résultats, mais lorsque la densité augmente, ces résultats se détériorent, tandis que les performances de CP-JSEQ augmentent dans ces conditions.

En résumé, malgré la dominance théorique de CP-GSEQ, cette méthode ne parvient à obtenir de meilleurs résultats que dans certains cas particuliers, et même dans ce cas, la stratégie FIFO obtiens souvent de meilleurs résultats. On peut donc conclure au minimum que le modèle PPC proposé doit être amélioré, puisqu'il doit pouvoir être au moins aussi bon que le modèle CP-JSEQ, qu'il domine en théorie.

4.4.3 Comparaison de CP-JSEQ avec les méthode GSEQ à chaud

La figure 4.9 présente le comportement typique des différentes méthodes à chaud avec les paramètres standards, et de la méthode CP-JSEQ sur les instances d'entraînement et de test A. Les résultats détaillés sur les instances de test A figurent sur les tables A.1, A.2, A.3, A.4, A.5, A.6 et A.7.

La figure 4.9 montre comment, en démarrant d'une bonne solution JSEQ calculée par le CP-JSEQ à INIT=90, les heuristiques GSEQ à chaud (AG-GSEQ*, EW-GSEQ* and TAB-GSEQ*) améliorent rapidement les performances des solutions en ajoutant de la flexibilité et améliorent les performances à l'entraînement, ce que l'on observe aussi sur les instances de test.

Les tables A.1 et A.2 révèlent que les méthodes à chaud améliorent légèrement la qualité des solutions quand Δ est petit, mais bien plus significativement quand Δ est grand. Cependant, pour le 1P, quand la variabilité augmente trop, les méthodes à chaud n'arrivent plus à obtenir de meilleures solutions que l'approche

purement réactive FIFO. L'analyse des performances en fonction de la variabilité des scénarios révèle que TAB-GSEQ* est presque toujours dominé par AG-GSEQ* et EW-GSEQ*, tandis que EW-GSEQ* est compétitif avec AG-GSEQ*, sauf pour les très grandes variabilités, pour lesquelles ses performances s'effondrent. La taille du problème (tables A.3 et A.4) ne semble pas avoir d'impact significatif dans la performance relative des méthodes à chaud. Au contraire, la table A.5 montre des différences en fonction de l'objectif pour le 1P : GA-GSEQ* obtient de meilleures performances que CP-JSEQ et EW-GSEQ* pour tous les objectifs sauf $\max L_{\max}$, où EW-GSEQ* obtient de meilleurs résultats. Cette différence n'est pas présente pour le JSP (Table A.6) et est certainement due à l'absence de gain relatif entre les stratégie JSEQ et GSEQ pour de nombreuses instances avec cet objectif. Enfin, la table A.7 montre qu'une plus grande densité des contraintes de précédence ne détériore pas les performances des méthodes GSEQ à chaud autant que celles de CP-GSEQ : les méthodes à chaud obtiennent de meilleurs résultats que CP-JSEQ de façon consistante (bien que le gain tende à diminuer avec la densité des contraintes de précédence). Cela est bien sur dû au fait qu'il utilisent comme point de départ une bonne solution JSEQ. Ces résultats montrent donc la supériorité des méthodes GSEQ sur les méthodes JSEQ dans la majorité des cas, sous réserve d'un démarrage à chaud.

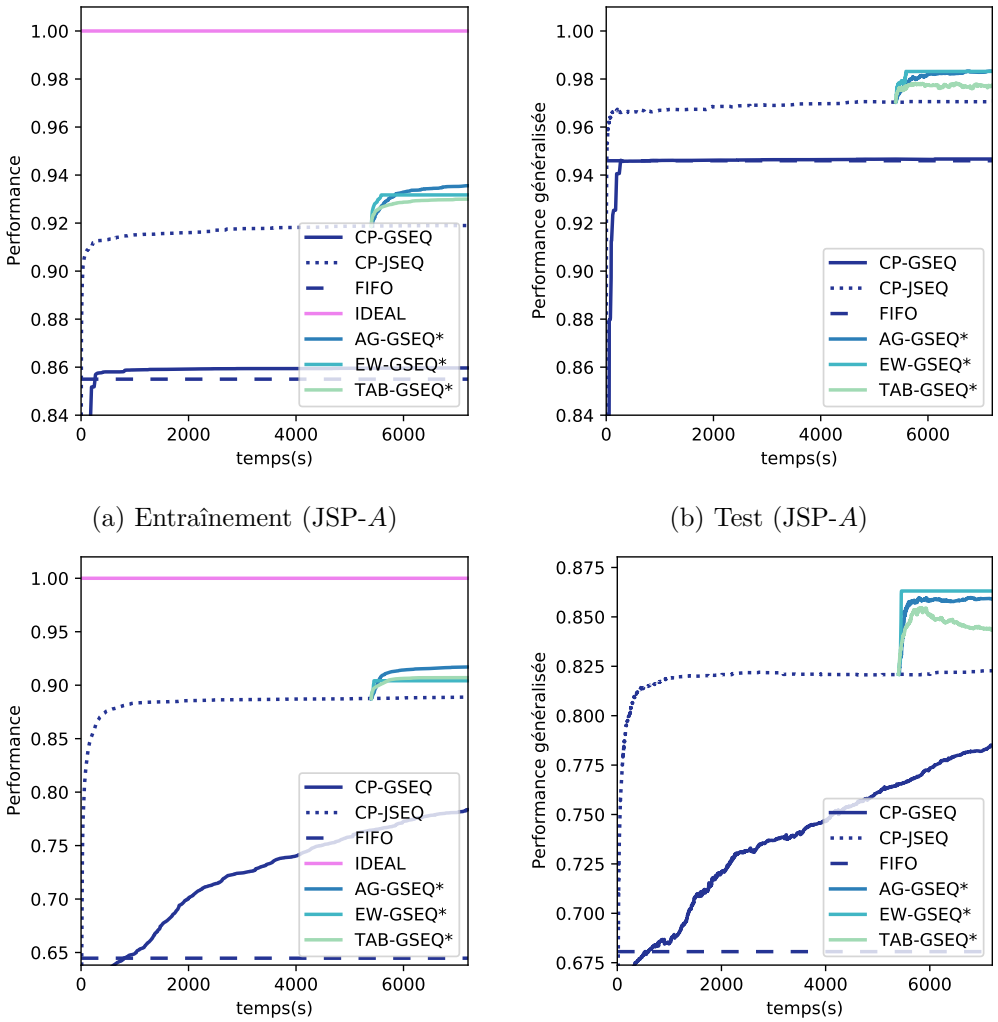
4.4.4 Flexibilité des solutions obtenues

Nous définissons la **flexibilité** d'une solution $\mathcal{X} \in \Omega_{GSEQ}$, représentée par une séquence de groupes π, par la valeur :

$$\text{Flexibilité}(\pi) = \frac{\sum_{m \in M} \sum_{G \in \pi^m} |G|.\log(|G|)}{|M|.|N|.\log(|N|)}$$

Cette valeur de flexibilité est comprise entre 0 et 1, et vaut exactement 1 lorsque toutes les tâches sur chaque machine sont comprises dans un unique groupe (c'est le cas pour les séquence de groupes correspondant à la stratégie FIFO) et 0 lorsque tout les groupes contiennent une seule tâche (les séquences de groupes correspondant à la stratégie JSEQ). L'utilisation du log permet de contrebalancer la croissance exponentielle du nombre de séquences représentées avec la taille des groupes pour obtenir une répartition plus équilibrée des valeurs de flexibilité.

Une corrélation importante est obtenue pour les méthodes à chaud entre la mesure de flexibilité et les paramètres OF (de l'ordre de 0.6 pour les deux problèmes) et dans une moindre mesure pour TPI (de l'ordre de 0.4). Ces paramètres influencent donc grandement la flexibilité des solutions obtenues. Cependant, cette plus grande flexibilité ne se corrèle pas nécessairement avec une moindre perte de généralisation. En effet, pour le JSP, la corrélation est presque nulle (environ -0.03) et pour le 1P, elle est plus importante (environ -0.25), mais une analyse plus poussée suggère que la cause de la perte de généralisation moins importante n'est pas seulement la flexibilité mais l'influence du paramètre TPI lorsque la densité des contraintes de précédence augmente. En effet, la figure 4.10 montre, pour la méthode AG-GSEQ*,



(a) Entraînement (JSP-A)

(b) Test (JSP-A)

(c) Entraînement (1P-A)

(d) Test (1P-A)

FIGURE 4.9 – Performance moyenne au cours du temps pour les méthodes (Instances A, paramètres standard)

que la paramétrisation par défaut et ¬OF ont une perte de généralisation similaire pour la valeur standard de densité de précédence (0.1) et au delà, malgré une flexibilité très différente. Tandis que la paramétrisation TPI produit des solutions qui peuvent être très flexibles lorsque la densité des contraintes augmente, et dont la perte de généralisation est moindre. On remarque aussi que maximiser la flexibilité des solutions semble avoir un impact plus important pour de très petites densité de contraintes de précédence, et dans ce cas, on observe une une différence entre les paramètres par défaut et ¬OF. Il semblerait que si l'on tolère les solutions partiellement inadmissibles, on puisse d'une part obtenir des solutions plus flexibles, mais aussi mieux s'adapter aux scénarios de tests. Toutefois, la flexibilité des solutions en soi ne semble pas être l'unique facteur déterminant pour minimiser la perte de

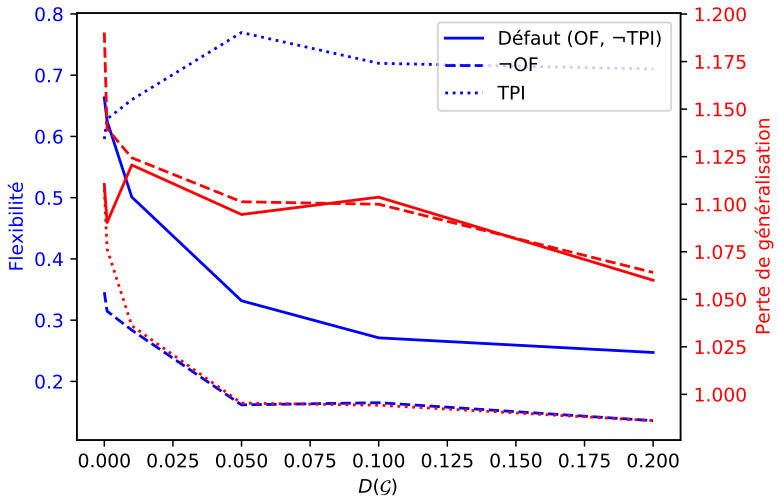


FIGURE 4.10 – Flexibilité et perte de généralisation moyenne des solutions AG-GSEQ* en fonction de la densité des contraintes de précédence (Instances 1P-$D(\mathcal{G})$)

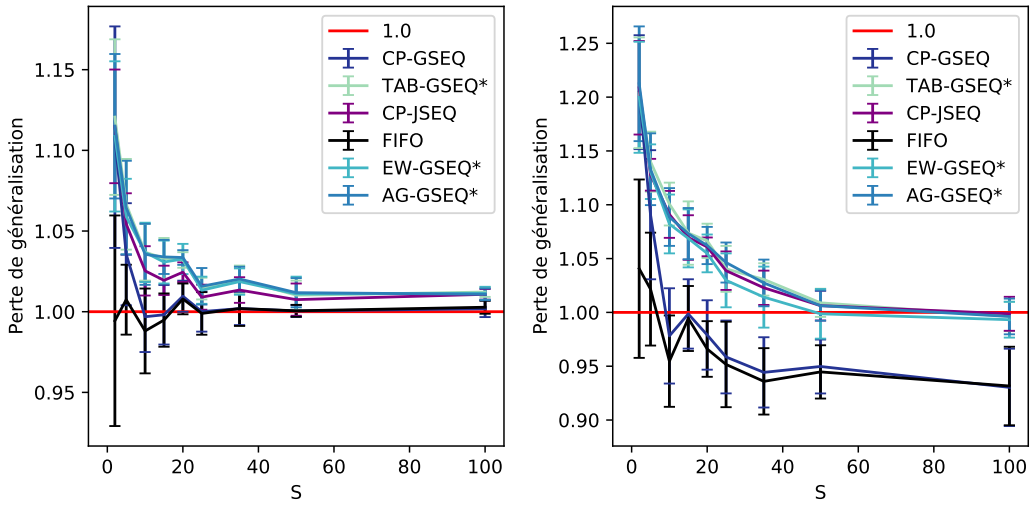
généralisation des solutions obtenues.

4.4.5 Score de généralisation

L'étude de la perte de généralisation des différentes méthodes semble confirmer que les méthodes GSEQ ont une perte de généralisation équivalente aux méthodes JSEQ. On peut comparer, sur la figure 4.11 l'évolution de la perte de généralisation en fonction du nombre de scénarios d'entraînement pour les méthodes GSEQ avec la méthode FIFO et la méthode CP-JSEQ. On observe en effet une diminution de la perte de généralisation et de la variance avec le nombre de scénario d'entraînement équivalente à celle obtenue pour la stratégie JSEQ (voir figure 3.8). On remarque toutefois une variation particulièrement importante dans le cas $\oplus = max$ pour le 1P. De plus, la perte de généralisation de la méthode CP-GSEQ s'assimile plus volontiers à celle de la méthode FIFO qu'à celle des autres méthodes GSEQ. Cela tend encore à montrer que les solutions obtenues par CP-GSEQ tendent à êtres similaires aux solutions purement réactives.

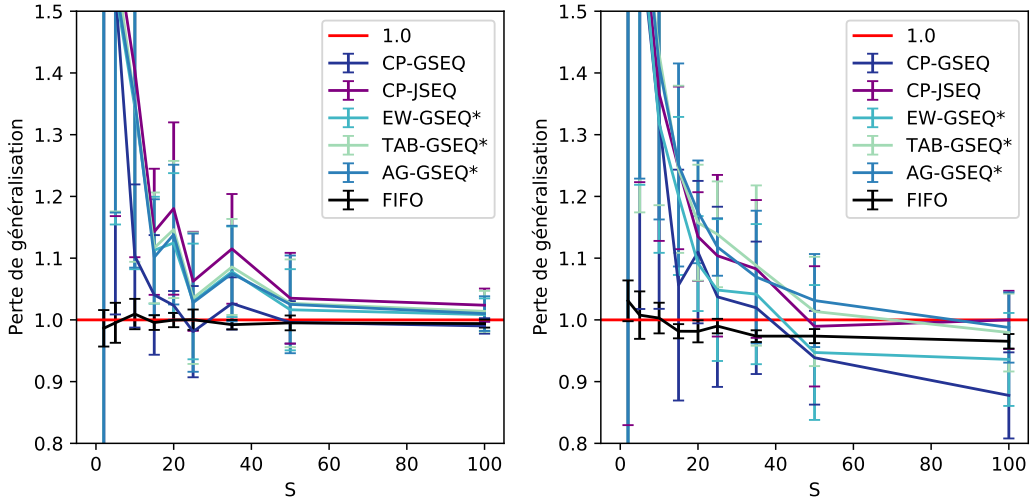
4.4.6 Impact de INIT

La figure 4.12 compare les performances généralisées au cours du temps de AG-GSEQ et EW-GSEQ à chaud. Les méthodes utilisent les paramètres par défaut, mais AG-GSEQ* utilise ¬OF.
On y remarque que EW-GSEQ* obtient de meilleures performances lorsque la solution JSEQ initiale est extraite quand CP-JSEQ atteint un plateau, alors que AG-GSEQ* est capable de diversifier un peu plus ses solutions et est donc moins sensible à la qualité de la solution initiale. Pour toutes les valeurs INIT, les mé-



(a) JSP-S ($\oplus = avg$)

(b) JSP-S ($\oplus = max$)

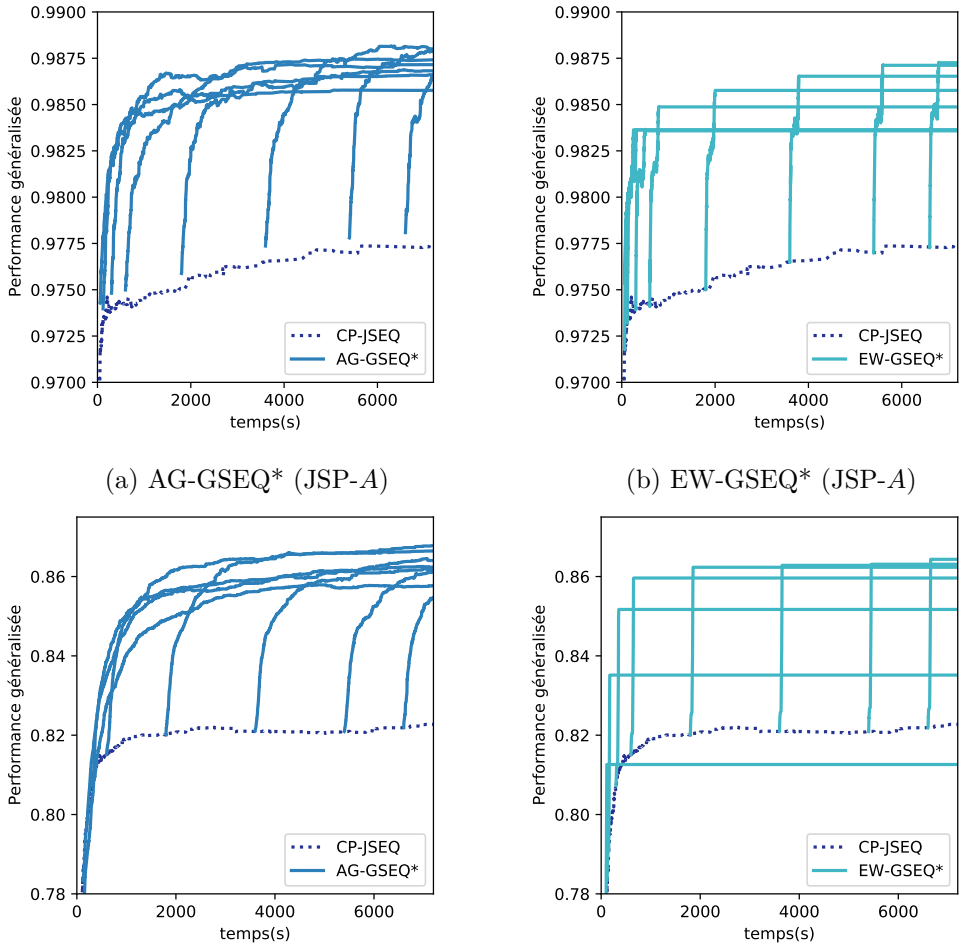


(c) 1P-S ($\oplus = avg$)

(d) 1P-S ($\oplus = max$)

FIGURE 4.11 – Perte de généralisation pour différentes méthodes en fonction du nombre de scénarios d'entraînement

thodes à chaud améliorent substantiellement les solutions JSEQ. Cependant, il est nécessaire de réserver un temps suffisant à l'algorithme à chaud pour en tirer pleinement parti. Dans un temps limité, on observe que même si JSEQ n'atteint pas un plateau, il peut être bénéfique d'utiliser les méthodes à chaud, sous réserve de diviser adéquatement le temps disponible entre les deux phases. Il faut aussi remarquer que même si AG-GSEQ* obtient de très légèrement meilleures solutions que EW-GSEQ*, la simplicité de l'heuristique gloutonne ainsi que le peu de temps nécessaire à l'amélioration des solutions la rendent très intéressante.



(a) AG-GSEQ* (JSP-A)

(b) EW-GSEQ* (JSP-A)

(c) AG-GSEQ* (1P-A)

(d) EW-GSEQ* (1P-A)

FIGURE 4.12 – Performance généralisée en fonction du temps de chauffe sur les instances standard. (¬OF pour AG-GSEQ*)

4.4.7 Impact du paramètre WC

Nous étudions l'impact du paramètre "pire-cas" (WC). C'est-à-dire que nous voulons déterminer s'il est plus intéressant à l'entraînement d'évaluer une solution dans la phase proactive en utilisant la politique FIFO de second niveau ou bien en utilisant l'évaluation pire cas décrite dans la section 4.1. Comme suggéré plus tôt, il faut aussi maximiser la flexibilité ($OF = \top$) en objectif secondaire dans ce cas.

Les tables A.8 et A.9 décrivent les performances généralisées pour différentes variabilités Δ. Les résultats montrent que plus les scénarios sont variables, plus l'utilisation de l'heuristique FIFO est préférable. La table A.10 montre que le gain est plus petit pour les instances les plus grandes du 1P, alors que la taille des instances n'a pas beaucoup d'impact sur le JSP. La table A.14 montre que l'existence de nombreuses contraintes de précédence diminue l'intérêt de l'évaluation FIFO, car

des solutions moins flexibles réduisent la différence entre les deux d'évaluation. On remarque que pour le 1P et l'objectif max L_{MAX}, ainsi que pour le JSP et l'objectif max C_{MAX}, l'optimisation WC est moins préjudiciable, et peut parfois obtenir de meilleures performances généralisées en moyenne (tables A.12 et A.13).

4.4.8 Impact de la tolérance des solutions partiellement inadmissible.

Nous avons montré plus tôt qu'il est possible de tolérer les GSEQ partiellement admissibles sans craindre de ne pas pouvoir obtenir un ordonnancement admissible. Cependant, la façon dont ce changement impacte l'espace de résolution des GSEQ, et donc l'impact sur les performances obtenues, est inconnu.
Les résultats sur le JSP (table A.16) montrent que pour la majorité des heuristiques à chaud, il est en moyenne préférable de tolérer les solutions partiellement admissibles quand Δ est grand, et légèrement pire quand Δ est petit. Le détail des objectifs de la table A.19 révèle que TPI est bénéfique pour l'objectif $\sum C_i$ mais pas pour C_{MAX}. Pour le 1P cependant, l'impact de TPI est extrêmement favorable pour tous les objectifs (table A.18), en particulier lorsque la variabilité entre scénarios est grande (table A.15). Cette différence d'impact s'explique certainement par la présence de contraintes de précédence. En effet, la table A.17 montre qu'en l'absence de contraintes de précédence, le paramètre TPI donne de moins bons résultats (car il ne fait qu'élargir l'espace des solutions), mais plus la densité des contraintes de précédence augmente, plus l'impact du paramètre est bénéfique. On suppose que cela est du à un espace des solutions plus permissif, plus facile à explorer étant donné le voisinage utilisé. Cela pourrait aussi expliquer l'effondrement des performances de CP-GSEQ quand la densité des contraintes de précédence augmente pour le 1P (Table A.7). En effet, puisque par défaut, la tolérance n'est pas implémentée pour le 1P, imposer la validité de toute les solutions représentées pourrait être trop contraignant dans le cas de nombreuses contraintes de précédence. Ce qui explique pourquoi la flexibilité apportée par la stratégie FIFO est plus efficace dans ce cas.

4.5 Discussion

Les expérimentations montrent que prendre des décisions séquentielles de premier niveau partielles et "laisser faire" une heuristique online pour compléter les décisions au fur et à mesure de la réalisation des incertitudes peut être bénéfique. Ces résultats confirment donc ceux de Cheref et al. [2014, 2016b] sur l'intérêt des groupes d'opérations permutables en ordonnancement stochastique et robuste et les résultats décrits par Aloulou and Portmann [2005] pour une stratégie basée sur les ordres partiels, qui concluent que c'est pour les instances avec une variation petite à moyenne que le plus grand gain de performances peut être obtenu par rapport à la stratégie JSEQ. Toutefois deux découvertes vont à l'encontre de la littérature classique sur les groupes : premièrement, il n'est pas nécessaire d'assurer que toutes les séquences représentées par les groupes soient réalisables (le paramètre TPI), ce

qui permet d'atteindre des solutions intéressantes même en présence de contraintes de précédence ; secondement, la maximisation de la flexibilité n'est pas forcément un objectif qui permet de mieux réagir face aux aléas dans le cas ou la décision de second niveau est prise par la politique FIFO. En moyenne, prendre en compte la phase réactive lors de la phase proactive obtient de meilleures performances généralisées que l'optimisation en considérant le pire cas, bien que ce ne soit pas toujours le cas pour certains objectifs.

Pour les différentes méthodes GSEQ présentées, la paramétrisation doit se faire en fonction des instances que l'on souhaite résoudre, puisque l'impact des différents paramètres varie fortement en fonction de la nature de l'instance considérée. Toutefois, les méthodes qui se démarquent sont AG-GSEQ*, qui obtient les meilleurs résultats, et EW-GSEQ*, qui obtient très rapidement des résultats compétitifs, et parfois meilleurs que AG-GSEQ*.

Au delà de la paramétrisation et du choix des méthodes, nous avons vu que bien qu'obtenir des solutions GSEQ de bonne qualité à partir de zéro est très difficile, si l'on utilise un démarrage à chaud, il est possible d'obtenir des résultats de meilleures qualité qu'avec la stratégie JSEQ ou la stratégie FIFO. Il est donc naturel de se demander si une stratégie dominant GSEQ peut obtenir des résultats encore supérieurs.

Chapitre 5

Représentation d'ensembles de séquences en utilisant les diagrammes de décisions multivalués

Sommaire

Une des limitations potentielles de la stratégie GSEQ présentée dans les chapitres précédents est due à son d'expressivité. Du fait de son incomplétude, il n'est pas possible de restreindre arbitrairement l'ensemble des séquences possibles avec une décision GSEQ au premier niveau. Par exemple, imaginons un problème 1P dans lequel la séquence optimale est ABC dans un scénario, et CBA dans un autre scénario. Pour avoir une chance d'obtenir ces solutions optimales au final, il faut au moins que la décision de premier niveau ne les exclut pas. Or, nous verrons qu'en utilisant la stratégie GSEQ, la seule décision de premier niveau satisfaisante est la

décision d'un seul groupe complètement permutable, qui n'exclut aucune solution (nous le montrerons formellement dans le théorème 7.1.4.1). Par conséquent, les chances que la décision de second niveau aboutissent à la solution optimale sont faibles. Pour cette raison, on peut s'attendre à ce qu'une stratégie en dominant une autre obtienne de strictement meilleurs résultats, surtout si la décision de premier niveau est prise en connaissance de la politique de second niveau.

Dans ce chapitre, nous nous intéressons donc à la stratégie complète, pouvant représenter n'importe quelle restriction des séquences au premier niveau, et qui domine toutes les autres stratégies en théorie. Dans ces premiers travaux, nous n'étudions l'implémentation de la stratégie complète que pour le problème 1P.

La décision associée à la stratégie complète peut être définie de nombreuses façons. Par exemple, la représentation la plus explicite consiste en l'ensemble des séquences compatibles avec la décision de premier niveau. On remarque alors que cet ensemble peut être *très* grand, et donc difficile à exploiter au second niveau. Remarquons que ce problème ne se posait pas avec la représentation par les séquences de groupe permutables de la stratégie GSEQ, dont la taille restait constante quelque soit le nombre de solutions représentées. Nous choisissons pour la représentation de la décision de premier niveau les diagrammes de décision multivalués (MDDs, voir section 2.3.5), qui permettent une représentation plus compacte de l'ensemble des séquences (nous verrons toutefois dans le chapitre 7 que dans certains cas, cette représentation n'est pas compacte non plus).
Après une courte introduction présentant des travaux pertinents utilisant les diagrammes de décision, nous commençons par établir les bases nécessaires à l'utilisation des MDDs comme représentation de décisions de premier niveau pour le problème à une machine et plus précisément les conditions de validité d'un MDD en tant que décision de premier niveau (au sens de la définition 3.1.2.1 : les conditions nécessaires pour qu'une heuristique FIFO puisse toujours en tirer un ordonnancement admissible). Nous proposons ensuite une implémentation de la stratégie en PPC, puis exploitons une particularité du cas où $\oplus = max$ (optimisation robuste) pour proposer une approche heuristique dans ce cas. Nous comparons ensuite les résultats obtenus par cette méthode aux stratégies et implémentations précédentes. Ce chapitre est issu d'une collaboration avec Willem-Jan van Hoeve (Tepper School of Business, Carnegie Melon University).

5.1 Introduction

Les MDDs sont un candidat intéressant pour la représentation de séquences en raison de leur compacité et des propriétés intéressantes qui sont exploitées dans la littérature. De plus, nous avons vu dans le chapitre 3 que plusieurs approches proactives-réactives utilisent des arbres pour leur décision de premier niveau, dans lesquels chaque chemin de la racine vers une feuille représente une conjecture, un ensemble de scénarios pour lesquels une décision est prise au second niveau. Toute-

fois, le nombre de scénarios différents rend souvent ces arbres trop grands. Utiliser des MDDs permet de profiter des mêmes principes, avec un potentiel gain de compacité due aux fusions de nœuds équivalents.

Récemment, une nouvelle perspective sur les problèmes d'optimisation ont été apportée par l'utilisation des diagrammes de décision [Bergman et al., 2016a]. Cette perspective permet notamment l'amélioration des méthodes générales de résolution comme la propagation de contraintes de PPC [Andersen et al., 2007, Hoda et al., 2010, Cire and van Hoeve, 2012, Bergman et al., 2014], mais aussi des concept de relaxation de PLNE [Andersen et al., 2007, Hooker, 2019, Bergman and Cire, 2018]. Les diagrammes de décisions valués sont aussi utilisés comme des heuristiques primales, permettant l'obtention de bornes pouvant être utilisées dans des approches de séparation et évaluation [Lozano and Smith, 2022, Bergman et al., 2016b]. De façon générale, les MDDs semblent permettre la modélisation efficace des problèmes.

Castro et al. [2022] cataloguent les nombreux travaux qui utilisent les diagrammes de décision comme outil dans la résolution de problèmes d'optimisation combinatoire. Les auteurs remarquent que les diagrammes de décision sont particulièrement adaptés pour les applications nécessitant de répéter des opérations, puisque en général, la compilation d'un MDD peut être difficile, mais que l'extraction de résultats est ensuite plus facile. En particulier, les diagrammes de décisions peuvent être utilisés pour représenter un ensemble d'objets de façon plus compacte qu'une simple liste, ce qui permet par la suite d'accélérer les requêtes. Par exemple, Malalel et al. [2023] utilisent un diagramme de décision dans un contexte multi-objectif pour stocker des informations sur le front de Pareto d'un problème, et ainsi accélérer leurs mise à jour au cours de la recherche. Dans Tran et al. [2023], un diagramme de décision est utilisé dans le cadre d'un problème de routage pour stocker un ensemble de clauses générées itérativement.

Toutefois, une des principales limitations déplorées par Castro et al. [2022] est la tendance des MDDs à grandir exponentiellement avec la taille des problèmes malgré tout, ce qui peut être mitigé à l'aide des techniques de restriction et relaxation (voir section 5.1.1.1), mais au coût d'approximations de moindre qualité.

Pour les problèmes d'ordonnancement, Cire and van Hoeve [2012] présentent plusieurs méthodes de propagation de contraintes pour les contraintes `AllDifferent` et les contraintes de précédence dans les diagrammes de décision multivalués. Cire and van Hoeve [2013] utilisent les diagrammes de décision relâchés pour l'obtention de bornes dans une approche par séparation et évaluation s'appliquant à une grande variété de problèmes d'ordonnancement sur une machine (avec contraintes de précédence, fenêtres de livraison, temps de préparation...). Hooker [2019] présente une méthode intégrant la relaxation lagrangienne dans des diagrammes de décision et donne des indications sur les problèmes d'ordonnancement susceptibles de bénéficier de la méthode.

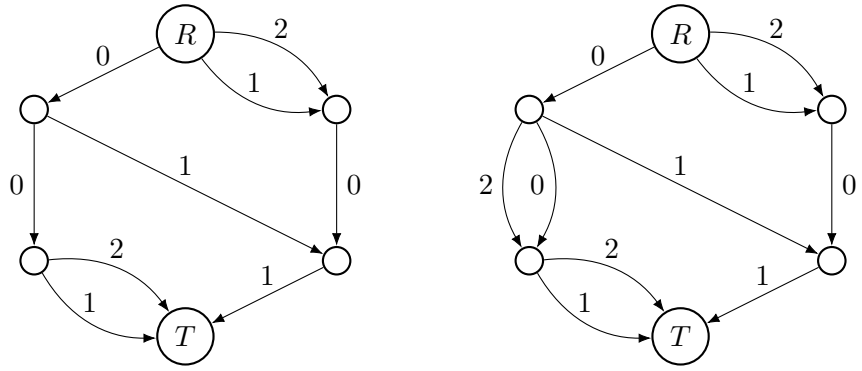
Bergman et al. [2016a] notent que les méthodes développées autour des dia-

grammes de décision ne s'adaptent pas directement au cas non-déterministe, dans lequel une décision à un nœud pourrait mener à différents états. Les travaux plus recents de Hooker [2022] proposent cette extension non-déterministe avec les diagrammes de décision stochastiques (SDD) dans lesquels une décision à un nœud mène à différents états, avec une probabilité donnée. Ils définissent des conditions suffisantes pour étendre le relâchement des diagrammes de décision aux SDDs, et appliquent notamment cette nouvelle structure pour un problème de séquencement de tâches de durées incertaines.

Quelques autres travaux tentent d'utiliser les diagrammes de décision dans des problèmes d'optimisation sujets aux incertitudes. Hadzic and Hooker [2006] tirent parti de la capacité des diagrammes de décision à répondre facilement à des requêtes répétées, une fois la structure calculée, pour mener une étude de sensibilité, et déterminer l'impact de la modifications de paramètres du problème sans avoir à recalculer les solutions de ces nouvelles instances (remarquons que ces travaux sont plus proches des méthodes que nous étudions dans la partie III).

Salemi and Davarnia [2023] proposent une méthode qui utilise un diagramme de décision en tant que problème maître dans la décomposition d'un problème particulier de flot avec une demande incertaine. MacNeil and Bodur [2022] proposent une décomposition pour des problèmes sous incertitude à deux étapes. La décomposition se base sur l'idée que pour la classe de problème étudiée, la décision de premier niveau permet de paramétrer un diagramme de décision dans lequel un algorithme de plus court chemin donne la meilleure solution pour les variables de recours. L'utilisation d'un algorithme de plus court chemin est une différence majeure entre beaucoup de travaux présents dans la littérature et notre approche (qui se restreint à la politique FIFO au second niveau) puisque dans notre cas, représenter un sur-ensemble de solutions à l'issue du premier niveau peut dégrader la qualité de la solution finale, ce qui n'est pas le cas avec une politique optimale.

Certaines approches ne considèrent pas les MDDs comme un outil pour obtenir une solution, mais *en tant* que solution du problème. Par exemple, Bergman and Cire [2017] proposent un modèle de PLNE pour calculer la meilleure relaxation d'un diagramme de décision binaire. Dans Jiang et al. [2022], les auteurs proposent une approche mêlant diagrammes de décision et réseaux de neurones pour apprendre dans un espace de décision structuré. L'idée est de générer un MDD pour représenter des solutions admissibles d'un espace de décision, puis d'utiliser l'apprentissage machine pour déterminer ensuite quelle solution sera extraite. On peut identifier ces deux phases à une stratégie à deux niveaux dans laquelle la génération du MDD est la phase proactive et l'extraction d'une solution constitue la phase réactive. Les auteurs s'intéressent à la validité de la politique de second niveau et montrent que pour un MDD exact, il est garanti d'obtenir une solution admissible. Ils proposent aussi une procédure de recherche du meilleur MDD relâché, mais ne précisent pas si il est garanti d'obtenir une solution dans ce cas.



(a) MDD restreint (b) MDD relâché

FIGURE 5.1 – MDD relâché et restreints

5.1.1 Quelques propriétés supplémentaires des MDDs

Nous introduisons dans cette section quelques propriétés supplémentaires utiles dans ce chapitre.

5.1.1.1 Diagrammes de décision exacts, restreints, relâchés

Un MDD est dit **exact** pour un problème P lorsque toutes les solutions admissibles et seulement les solutions admissibles de P sont représentées dans le MDD. Si le MDD de la figure 2.4 est un MDD exact pour un problème donné, les MDDs de la figure 5.1 en présentent une version restreinte et relâchée de largeur 2.

Un MDD est dit **restreint** pour un problème P lorsque seulement des solutions admissibles de P (mais pas nécessairement toutes) sont représentées dans le MDD. Le MDD restreint de la figure 5.1a représente les affectations $(0, 0, 1)$, $(0, 0, 2)$, $(0, 1, 1)$, $(1, 0, 1)$, et $(2, 0, 1)$ (Il omet donc la solution admissible $(0, 2, 1)$).

Un MDD est dit **relâché** pour un problème P lorsque toutes les solutions admissibles de P (mais possiblement d'autres) sont représentées dans le MDD. Le MDD relâché de la figure 5.1b représente les affectations $(0, 0, 1)$, $(0, 0, 2)$, $(0, 1, 1)$, $(1, 0, 1)$, $(2, 0, 1)$, $(0, 2, 1)$ et $(0, 2, 2)$ (Il représente donc la solution non admissible $(0, 2, 2)$).

5.1.1.2 Données supplémentaires

Un MDD est parfois enrichi d'informations supplémentaires de deux types. L'ajout d'un coût à chaque arc permet de représenter des MDDs valués, permettant le calcul d'un score associé à chaque chemin. L'ajout d'informations à chaque nœud est utilisé par exemple pour filtrer efficacement les arcs du MDD lors de la propagation de contraintes. Dans cette thèse, nous nous intéressons à des MDDs basés sur les variables de décision de position, pour lesquels chaque arc représente l'affectation d'une tâche à une position. U n chemin représente donc un ensemble d'affectations. Nous utilisons dans ce chapitre les informations "*All*" et "*Some*" [Andersen et al., 2007, Cire and van Hoeve, 2013, Bergman et al., 2016a], qui décrivent

pour chaque nœud du MDD, respectivement l'ensemble des tâches rencontrées dans *tous* les chemins de la racine au nœud et l'ensemble des tâches rencontrées dans *au moins un* chemin de la racine au nœud.

Pour chaque nœud d'un MDD donné, si $In(u)$ est l'ensemble des arcs entrants dans u, l'information All peut être calculée récursivement depuis le nœud racine $All(R) = \emptyset$ par l'équation

$$All(v) = \bigcap_{a=(u,v)\in In(v)} (All(u) \cup \{l_a\})$$

De façon similaire, l'information $Some$ est obtenue par l'équation suivante avec $Some(R) = \emptyset$:

$$Some(v) = \bigcup_{a=(u,v)\in In(v)} (Some(u) \cup \{l_a\})$$

5.2 L'approche à deux niveaux avec les MDDs

Comme nous l'avons mentionné, il est d'ordinaire possible d'extraire, par un algorithme de plus court chemin, la meilleure solution d'un MDD exact, et des bornes (respectivement inférieures et supérieures) de MDDs relâchés et restreints. Mais cela n'est pas le cas lorsque l'on utilise une politique de décision pour le choix de la solution au second niveau. Par conséquent, contrairement aux applications classiques des MDDs dans les problèmes d'optimisation et d'ordonnancement, notre objectif n'est pas de représenter *toutes* les séquences admissibles, mais seulement un ensemble de séquences constituant une décision partielle de premier niveau, de sorte que la meilleure solution possible puisse être obtenue par la politique FIFO au second niveau. On peut donc qualifier les MDDs qui nous intéressent de "restreints". Toutefois, cette restriction n'est accompagnée d'aucune garantie quant à la taille du MDD, habituellement limitée par une largeur maximale W. Nous considérons alors la possibilité de relâcher un MDD restreint, ce qui pourrait permettre de représenter de façon plus compacte l'ensemble des solutions restreint par la décision partielle.

Nous considérons donc dans cette partie une décision de premier niveau consistant en un MDD capable de représenter une restriction arbitraire de l'ensemble des séquences. Nous utilisons les variables de décision de position $\check{\zeta}_i = j$ ssi la tâche j est en position i pour représenter cet ensemble de séquences (comme suggéré section 2.3.5). Rappelons que dans un MDD basé sur les variables de décision de position, la couche $\mathcal{L}_i$ est associé à la variable de décision ζ_i. L'emprunt d'un arc a, étiqueté l_a, d'un nœud u sur la couche $\mathcal{L}_i$ à un nœud v sur la couche $\mathcal{L}_{i+1}$, représente l'affectation $\check{\zeta}_i = l_a$. Un chemin du nœud racine R sur la couche $\mathcal{L}_0$ au nœud terminal T sur la couche $\mathcal{L}_{|N|}$ (dit chemin *complet*) représente donc une affectation complète des variables de décision : une séquence.
Rappelons aussi qu'il est possible d'associer à chaque nœud l'information complémentaire All et $Some$, décrivant les tâches affectées dans respectivement *tous* et *certains* chemins depuis la racine au nœud.

Étant donné un MDD, la politique FIFO de second niveau procède de la façon suivante : en partant initialement du nœud racine, et à chaque moment de décision, FIFO ordonnance la tâche disponible en premier parmi les tâches figurant sur les arcs issus du nœud courant. Considérons par exemple que le MDD de la figure 5.2 est la décision de premier niveau pour un problème à une machine et trois tâches A, B et C (remarquons que la décision représente les séquences pour lesquelles $A \prec C$). Soit un scénario s dans lequel $r^s_C < r^s_A < r^s_B$. Les décisions de la politique FIFO sont indiquées en rouge. Au nœud racine, les tâches A et B sont disponibles d'après la décision de premier niveau. Étant donné le scénario, la tâche A est ordonnancée en premier au plus tôt, et la politique suit l'arc associé. Lorsque la tâche termine son exécution, la décision de premier niveau indique que le choix est à faire entre les tâches B et C. D'après le scénario, c'est la tâche C qui est choisie, elle est donc exécutée au plus tôt, puis il ne reste plus que la tâche B.

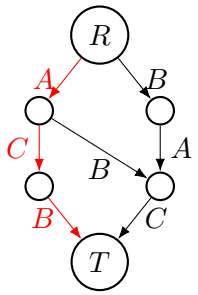


FIGURE 5.2 – Exemple de MDD pour la représentation de séquences

5.2.1 Conditions de validité d'un MDD comme solution de premier niveau

Il est facile de montrer qu'un MDD exact représentant seulement des séquences admissibles est valide au sens de la définition 3.1.2.1 pour toute politique H ordonnançant les tâches selon un chemin de la racine au terminal. En particulier la politique de second niveau FIFO pourra en extraire une séquence pour tout scénario. Mais nous aimerions savoir s'il est possible de relâcher la contrainte d'exactitude du MDD à la façon des séquences de groupes d'opérations permutables partiellement admissibles pour produire un MDD relâché qui resterait valide pour la procédure FIFO décrite. Ces solutions sont potentiellement plus intéressantes pour les MDD qu'elles ne le sont pour les séquences de groupes, puisqu'en plus de modifier l'espace des solutions, relâcher un MDD permet de compacter arbitrairement la représentation des séquences en fixant une largeur maximale W (mais peut aussi introduire de nouvelles séquences). Cependant, est-il toujours possible pour la politique FIFO d'extraire une solution d'un MDD relâché ? Cette section apporte une réponse partielle négative à cette question en étudiant les conditions de validité d'un MDD comme solution de premier niveau.

5.2.1.1 Caractérisations utiles des MDD

Nous avons défini dans la section 5.1.1.1 ce qu'est un MDD exact et relâché, ce qui indique si les solutions représentées par les chemins complets du MDD sont admissibles ou non. Mais il est possible de raffiner ces définitions pour préciser quelles sont les contraintes qui sont relâchées.

Définition 5.2.1.1. *Si* $\mathtt{C}$ *est une contrainte sur les variables du problème, nous disons qu'un MDD est* ***C-restreint*** *ssi tout chemin complet de sa racine à son nœud terminal représente une séquence qui satisfait la contrainte* $\mathtt{C}$*.*

Définition 5.2.1.2. *Nous disons qu'un MDD est* ***C-consistant*** *ssi pour chacun de ses arcs, il existe un chemin complet de sa racine à son nœud terminal passant par cet arc qui représente une séquence qui satisfait la contrainte* $\mathtt{C}$*.*

Définition 5.2.1.3. *Nous disons qu'un MDD est* ***C-relâché*** *ssi il existe un chemin complet de sa racine à son nœud terminal qui satisfait la contrainte* $\mathtt{C}$*.*

On remarque qu'un MDD $\mathtt{C}$-restreint est $\mathtt{C}$-consistant, et qu'un MDD $\mathtt{C}$-consistant est $\mathtt{C}$-relâché. De plus, pour une conjonction de contraintes $\mathtt{C} = \{\mathtt{C}_1 \wedge \mathtt{C}_2 \wedge \ldots \mathtt{C}_k\}$, si un MDD est $\mathtt{C}$-restreint (resp. $\mathtt{C}$-consistant, resp. $\mathtt{C}$-relâché), alors il est $\mathtt{C}_i$-restreint (resp. $\mathtt{C}_i$-consistant, resp. $\mathtt{C}$-relâché) $\forall i$. La réciproque n'est vraie que pour la $\mathtt{C}$-restriction.

5.2.1.2 Validité de MDD relâchés

Rappelons qu'une séquence $\check{\zeta}$, représentée par une affectation des variables de position $\chi_{pos} = \{\zeta_1, \ldots, \zeta_{|N|}\}$, est admissible pour le problème à une machine (1P) ssi chaque tâche est affectée à une unique position dans la séquence, et que l'ensemble des contraintes de précédence E sont respectées. Autrement dit, la séquence satisfait les contraintes $\mathtt{AllDifferent}(\chi_{pos})$ et $\mathtt{Prec}(E)$ (où $\mathtt{Prec}(E)$ représente la conjonction des contraintes $\mathtt{C}_{i \prec j}, \forall (i,j) \in E$) [1]. Un ordonnancement admissible est ensuite obtenu par l'ordonnancement au plus tôt des tâches.

Nous savons qu'un MDD $\{\mathtt{AllDifferent} \wedge \mathtt{Prec}\}$-restreint est valide. Mais est-il possible pour FIFO d'obtenir une séquence admissible à partir d'un MDD qui ne soit pas restreint ?
Nous envisageons dans la suite différents degrés de relâchement de la $\mathtt{AllDifferent}$-restriction et $\mathtt{Prec}$-restriction de notre MDD. Nous étudions en particulier les trois degrés de granularité de relaxation : la $\mathtt{C}$-restriction, la $\mathtt{C}$-consistance, et le $\mathtt{C}$-relâchement (pour lequel la seule contrainte est l'existence d'une solution satisfaisant $\mathtt{C}$ dans le MDD). L'objectif est de déterminer les conditions minimales de validité d'un MDD selon ces degrés, c'est-à-dire sous quelle conditions une politique FIFO

1. On note simplement "$\mathtt{AllDifferent}$" et "$\mathtt{Prec}$" dans la suite

est capable d'obtenir un ordonnancement admissible a partir d'un MDD quel que soit le scénario. Nous supposons aussi que la politique FIFO doit pouvoir prendre une décision gloutonne à l'aide de l'information localement disponible à chaque nœud, et en connaissance des décisions passées.

Validité d'un MDD `AllDifferent`-consistant : Nous montrons ici que relâcher la `AllDifferent`-restriction d'un MDD pour imposer seulement sa `AllDifferent`-consistance ne permet plus à une politique gloutonne FIFO d'établir une séquence valide, et ce même en utilisant les informations locales *All* et *Some* associées aux nœuds. Considérons le MDD de la figure 5.3. Il est bien `AllDifferent`-consistant puisque l'on peut vérifier que chaque arc fait partie d'une chemin contenant toute les tâches. Mais supposons que dans un scénario s, $r^s_D < r^s_A < r^s_B < r^s_C$. La politique FIFO sélectionnerait dans les premières étapes la sous-séquence DAB, puis ne pourrait continuer la séquence, puisque aucun des arcs disponibles ($\{B, D\}$) ne peut être emprunté sans violer la contrainte `AllDifferent`. On remarque qu'aucune séquence valide représentée ne commence par la sous-séquence DA, cependant, il existe des chemins commençant par DA qui contiennent B et de chemins qui contiennent C, et de plus, tous les chemins commençant par DA ne contiennent pas D ni A dans leur suite, donc l'information locale ne permet pas d'établir que A était un choix ne menant à aucune séquence valide. Pour pouvoir détecter le problème, il faudrait savoir qu'aucun chemin commençant par DA ne contient *à la fois* B et C.
Par conséquent, nous voyons que la `AllDifferent`-consistance ne suffit pas pour la validité du MDD. Étant donné nos degrés de granularité, nous concluons donc à la nécessité de la `AllDifferent`-restriction de nos MDDs. On notera que l'absence de contraintes de précédence étant un cas particulier de `Prec` (avec $E = \emptyset$), l'exemple de la figure 5.3 est aussi un MDD `AllDifferent`-consistant et `Prec`-restreint.

Validité d'un MDD `AllDifferent`-restreint Soit donc un MDD `AllDifferent`-restreint. Est-ce un condition suffisante pour sa validité en utilisant FIFO ? La figure 5.4 donne un exemple d'un MDD `AllDifferent`-restreint et `Prec`-relâché (pour une unique contrainte de précédence $C_{B \prec C}$) dont une solution est faisable (BCA). Cependant, supposons que dans un scénario s, $r^s_A < r^s_B$: la décision A est prise au nœud racine. Cette décision ne pose pas de problème localement, puisque les informations locales *All* et *Some* ne décrivent pas dans quel ordre les tâches B et C apparaîtront dans la suite. Toutefois, la politique FIFO ne pourra pas obtenir de séquence valide après cette décision.

Validité d'un MDD `AllDifferent`-restreint et `Prec`-consistant Le degré de relaxation minimal suivant est de considérer un MDD `AllDifferent`-restreint et `Prec`-consistant. Nous montrons qu'un tel MDD est aussi `Prec`-restreint, et donc $\{\texttt{AllDifferent} \wedge \texttt{Prec}\}$-restreint, dont nous savons déjà la validité.

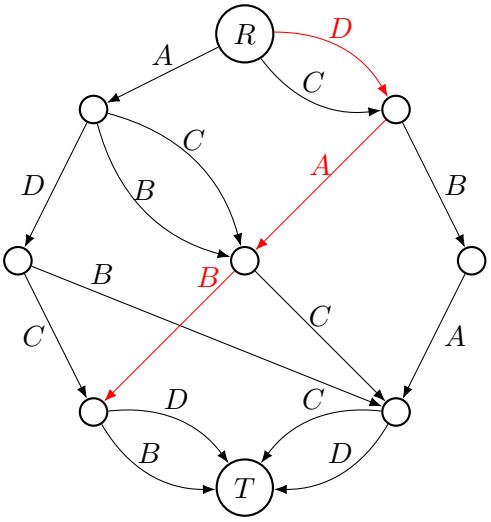


FIGURE 5.3 – Exemple d'un MDD `AllDifferent`-consistant pouvant égarer la politique FIFO.

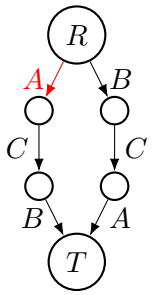


FIGURE 5.4 – Exemple d'un MDD `AllDifferent`-restreint pouvant égarer la politique FIFO pour un contrainte de précédence $B \prec C$.

Définition 5.2.1.4. *Un MDD (V, A) est* ***C-chemin-indépendant*** *ssi pour chaque nœud u du MDD, quel que soit deux chemins p_1 et p_2 de la racine à u faisant chacun partie d'un chemin admissible, pour tout chemin p de u à T, $p_1.p$ est admissible ssi $p_2.p$ l'est aussi. Formellement :*

$$\forall u \in A, \forall p_1, p_2 = (R \to \cdots \to u) \ t.q. \ \exists p'_1, p'_2 = (u \to \cdots \to T), \mathtt{C}(p_1.p'_1) = \mathtt{C}(p_2.p'_2) = \top, \\ \forall p = (u \to \cdots \to T), \mathtt{C}(p_1.p) = \mathtt{C}(p_2.p) \tag{5.1}$$

Nous montrons d'abord qu'un MDD `C`-consistant et `C`-chemin-indépendant est `C`-restreint. Puis nous montrons qu'un MDD `AllDifferent`-restreint et `Prec`-consistant est `Prec`-chemin-indépendant.

Théorème 5.2.1.1. *Un MDD `C`-consistant et `C`-chemin-indépendant est `C`-restreint.*

Démonstration. Soit un MDD (V, A) `C`-consistant. Par définition, il existe pour

chaque arc a entre deux nœuds u et v un chemin complet p de la racine R au terminal T qui satisfait la contrainte $\mathtt{C}$.

$$\forall a = (u, v) \in A, \exists p = (R \to \cdots \to u \xrightarrow{a} v \to \cdots \to T) \text{ t.q. } \mathtt{C}(p) = \top \tag{5.2}$$

En identifiant les chemins p et p', on a $\exists p \xrightarrow{a} p'$ tel que $\mathtt{C}(p \xrightarrow{a} p') = \top$. Si le MDD est $\mathtt{C}$-chemin-indépendant, l'équation 5.1 donne que pour tout autre chemin p_o de la racine à u qui fait partie d'un chemin qui satisfait $\mathtt{C}$, on a $\mathtt{C}(p_o \xrightarrow{a} p') = \top$. Autrement dit, toute décision prise depuis un nœud à l'issue d'un chemin p faisant partie d'un chemin complet satisfaisant $\mathtt{C}$ est cohérente avec un chemin $\mathtt{C}$-valide. Par propagation depuis le nœud racine, puisqu'il existe au moins une solution satisfaisant $\mathtt{C}$ dans le MDD, tous les chemins complets satisfont $\mathtt{C}$.

□

Lemme 5.2.1.1. *Un MDD `AllDifferent`-restreint et `Prec`-consistant est `Prec`-chemin-indépendant.*

Démonstration. Soit une unique contrainte de précédence $i \prec j$. Si $\mathcal{L}^p(i)$ représente la couche à laquelle un arc étiqueté i est sélectionné dans le chemin p (puisque le MDD est `AllDifferent`-restreint, cette couche existe pour chaque chemin complet p), alors :

$$\mathtt{C}_{i \prec j}(p) = \begin{cases} \top \iff \mathcal{L}^p(i) < \mathcal{L}^p(j) \\ \bot \iff \mathcal{L}^p(i) > \mathcal{L}^p(j) \end{cases} \tag{5.3}$$

Supposons qu'un MDD $M = (V, A)$ est $\mathtt{C}_{i \prec j}$-consistant, mais que l'équation 5.1 n'est pas vérifiée (i.e. $\exists u \in V, p1, p2$ (tels que présentés), $\exists p = (u \to \cdots \to T)$, $\mathtt{C}_{i \prec j}(p1.p) \neq \mathtt{C}_{i \prec j}(p2.p)$) (voir la figure 5.5).

Soit N l'ensemble de toutes les tâches et N_p l'ensemble des tâches rencontrées sur le chemin (partiel) p. Puisque M est `AllDifferent`-restreint, on sait que $N_{p_1} = N_{p_2}$ et $N_{p_1} \cup N_p = N$. Sans perte de généralité, supposons $\mathtt{C}_{i \prec j}(p_1.p) = \top$ et $\mathtt{C}_{i \prec j}(p_1.p) = \bot$. La seule façon pour que $\mathcal{L}^{p_1.p}(i) < \mathcal{L}^{p_1.p}(j)$ et $\mathcal{L}^{p_2.p}(i) > \mathcal{L}^{p_2.p}(j)$ est que $\{i, j\} \subseteq N_{p_1} \cap N_{p_2}$. Mais on sait que $\exists p'_1, \mathtt{C}_{i \prec j}(p_1.p'_1) = \top$, donc $\mathcal{L}(i) < \mathcal{L}(j)$ dans $p_1.p'_1$. Une contradiction.
Il suit que l'équation 5.1 est vérifiée : Le MDD est $\mathtt{C}$-chemin-indépendant. □

Par conséquent, par le théorème 5.2.1.1, tout MDD `AllDifferent`-restreint et $\mathtt{C}_{i \prec j}$-consistant est aussi $\mathtt{C}_{i \prec j}$ restreint. Et donc, s'il est $\mathtt{C}$-consistant pour un ensemble de contraintes de précédence E On en déduit qu'il est aussi $\mathtt{Prec}(E)$-restreint.

5.2.2 Validité d'un MDD relâché vis à vis d'une politique de second niveau opérant un filtrage répété

La section précédente conclut qu'étant donné une politique FIFO utilisant des informations locales de façon gloutonne, les niveaux de relâchement du MDD consi-

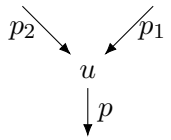


FIGURE 5.5 – Schéma d'un nœud de MDD

dérée ne sont pas suffisants pour une décision de premier niveau valide. Peut on considérer une autre implémentation de la politique FIFO qui permette la validité de MDDs non-restreints ?
Imaginons une politique FIFO+, plus complexe, qui à chaque moment de décision lors de son parcours dans le MDD, en filtre les arcs non empruntés, ainsi que les arcs qui ne font partie d'aucune solution valide compatible avec les choix fait précédemment.
Par exemple, étant donné le MDD de la figure 5.2, une fois la décision A prise sur la couche $\mathcal{L}_0$, l'arc correspondant à la décision B est supprimé, et tous les arcs ne faisant plus partie de chemins complets admissibles sont supprimés (l'arc au label A de la couche $\mathcal{L}_1$ dans l'exemple).

Proposition 5.2.2.1. *La procédure FIFO+ par filtrage répété permet d'étendre la validité d'un MDD à tout MDD contenant une séquence admissible.*

Démonstration. Soit un MDD $M = (V, A)$ tel que M soit individuellement $\mathtt{C_i}$-relâché pour une conjonction de contraintes $\mathtt{C} = \bigwedge_i \mathtt{C_i}$. Supposons qu'à un moment de décision, après un chemin partiel faisant partie d'un chemin complet admissible p la procédure FIFO+ soit à un nœud $u \in V : \exists p',\ \mathtt{C}(p.p') = \top$. Le processus de filtrage supprime tout les arcs non-empruntés des couches déjà traversées et établit la $\mathtt{C}$-consistance sur le MDD restant. Puisque p fait partie d'un chemin complet admissible, il reste au moins un chemin complet dans le MDD. De plus, supposons que la décision prise à ce moment mène au nœud $v \in Out(u)$ par l'intermédiaire de l'arc $a = (u, v)$. Par la définition de la $\mathtt{C}$-consistance, a vérifie $\exists p_a = (R \to \cdots \to u), p'_a = (v \to \cdots \to T), C(p_a \xrightarrow{a} p'_a) = \top$. Or, un seul chemin de la racine à u subsiste après filtrage, donc $\exists p'_a, C(p \xrightarrow{a} p'_a) = \top$: quel que soit la décision prise par FIFO+, le chemin partiel fait parie d'u chemin complet admissible.
Par induction, si M contient une solution admissible, cette procédure FIFO+ avec filtrage répété extrait une solution d'un tel MDD.

□

Toutefois, il est connu qu'établir la `AllDifferent`-consistance est NP-difficile [Andersen et al., 2007]. Et comme nous l'avons vu, établir la `Prec`-consistance sur un MDD `AllDifferent`-restreint reviens à établir sa `Prec`-restriction.

On considère donc que cet effort lors de la phase réactive serait trop important, et ne considérons pas l'implémentation de la politique FIFO+.

5.2.3 Implémentation PPC de la stratégie complète basée sur les MDD restreints : CP-COMPL

Dans cette section, nous proposons une implémentation de la stratégie complète dont la décision de premier niveau est représentée par un MDD. La section précédente conclut à la non validité des MDDs non restreints étant donné la politique FIFO de second niveau que nous avons décrit. L'implémentation de la stratégie doit donc se limiter à la recherche d'un MDD **restreint**. Comme dans les chapitres précédent, l'objectif est de chercher le MDD qui obtient les meilleurs résultats possibles lorsqu'il est évalué sur un ensemble de scénarios d'entraînement, en espérant qu'il obtienne aussi de bons résultats sur un ensemble de scénarios de tests.

Pour bien définir le problème de recherche de MDD, nous proposons dans un premier temps un modèle de PPC [2], inspiré du modèle de la stratégie GSEQ (modèle 4.1) et du modèle PLNE de Bergman and Cire [2017].

Le modèle CP-COMPL (Modèle 5.1) utilise des variables de *construction* et des variables d'*évaluation*. Les variables de construction décrivent la décision valide de premier niveau , elles correspondent aux variables de groupe "g", qui décrivent la composition des groupes d'une séquence de groupes dans le modèle GSEQ (modèle 3.1). Un paramètre W limite la largeur du MDD (le nombre de nœuds par couche) produit et donc la taille du modèle. Les variables d'évaluation simulent l'ordonnancement obtenu par la politique de second niveau et calculent le score obtenu par la solution pour chaque scénario. Ces variables correspondent aux variables d'intervalles *Jobs* dans le modèle GSEQ.

La figure 5.6 illustre le fonctionnement du modèle et la fonction des variables de construction et d'évaluation.

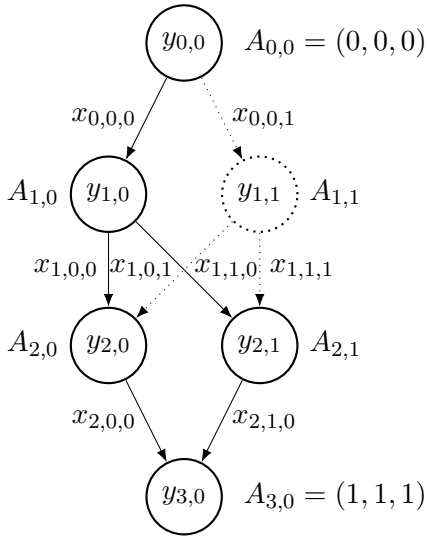
Les variables de décision binaires utilisées pour la construction (lignes (23-25) du modèle) sont les suivantes :

- $y_{l,i}$ indique si le nœud d'index i de la couche l est utilisé. Par exemple, dans la figure, le nœud correspondant à $y_{1,1}$ n'est pas utilisé.
- $x^j_{l,i,k}$ indique si un arc étiqueté j lie le nœud d'index i de la couche l au nœud d'indice k de la couche $l+1$.
- $a^j_{l,i}$ indique si tous les chemins p de la racine au nœud d'indice i de la couche l (correspondant à la variable $y_{l,i}$) contiennent un arc étiqueté par la tâche j (ce qui s'exprime par $j \in N_p$). Les variables $a^j_{l,i}$ associées à un nœud $y_{l,i}$ représentent ses données *All*. Par exemple, dans la figure, $a^A_{1,0} = 1$ car tous les chemins de la racine au nœud $y_{1,0}$ empruntent un arc étiqueté A.

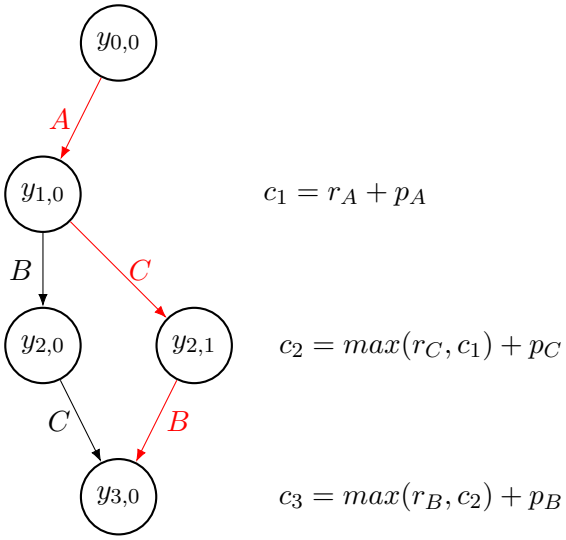
Les lignes (26-27) présentent les variables de décision utilisées pour l'évaluation

2. Le modèle décrit peut aussi être implémenté en PLNE, mais des résultats préliminaires suggèrent que la modélisation PPC est plus efficace

3. Dans la figure, par soucis de lisibilité, on utilise des variables entières équivalentes $x_{l,i,k} = j$, par exemple $x_{1,0,0} = B$. De plus, les variables a sont regroupées dans un tuple A. Par exemple, $A_{2,0} = (a^1_{2,0}, a^1_{2,0}, a^0_{2,0}) = (1, 1, 0)$



(a) Construction



(b) Évaluation ($r_c < r_b$)

FIGURE 5.6 – Illustration du modèle PPC[3](|N|=3, W=2)

du MDD sur chaque scénario :

— $t^j_{l,i,k,s} \in \{0, 1\}$ indique si l'arc $x^j_{l,i,k}$ est emprunté (i.e si la décision correspondante est prise) par la politique FIFO dans le scénario s dans le MDD représenté. Dans la figure, la sélection pour l'évaluation pour un scénario donné est représentée par l'arc en rouge.

— $c_{l,i}$ indique la date de fin de la l-ième tâche dans le scénario s. Les valeurs obtenues pour le scénario considéré sont explicitées dans la figure.

Au total, pour une instance avec $|N|$ tâches, $|S|$ scénarios, et une largeur maxi-

male W, le modèle comporte $O(|N|^2W^2|S|)$ variables binaires, $O(W|S|)$ variables entières, et $O((|N|+|S|)|N|^2W^2)$ contraintes.

Le modèle peut se diviser selon les deux parties : les variables x, y, et a, ainsi que les contraintes (6-16) fixent les règles pour la construction du MDD valide au premier niveau, tandis que les autres contraintes et les variables c et t permettent l'évaluation du MDD par la politique FIFO au second niveau.

construction du MDD Les variables x et y, et les contraintes (6-12) assurent que le graphe à bien la forme d'un MDD. (6-7) ne permettent d'arc qu'entre des nœuds utilisés ($y_{l,i} = 1$). (8-10) assurent que chaque arc fait partie d'un chemin de la racine au nœud terminal. (11) stipule qu'il ne peut y avoir qu'un seul arc entre deux nœuds. (12) interdit à deux arcs sortants d'un nœud d'avoir le même label.

Les variables a et les contraintes (13-16) imposent la $\{\texttt{AllDifferent} \wedge \texttt{Prec}\}$-restriction. (13) permet le calcul des données *All* : pour que tous les chemins p menant à un nœud u aient $j \in N_p$, il faut que pour chaque nœud parent v de u, soit tout les chemins menant à v contiennent j, soit que l'arc de v à u soit étiqueté j. (14-15) établissent la valeur de *All* au nœud racine et au nœud terminal pour assurer que tous les chemins complet p aient $N_p = N$. La `AllDifferent`-restriction est donc assurée. (16) assure que pour chaque contrainte de précédence $A \prec B$, si un nœud possède un arc sortant étiqueté B, tous les chemins p menant au nœud ont $A \in N_p$. La contrainte assure donc la `Prec`-restriction [4].

Évaluation du MDD construit Les contraintes (17-20) modélisent le comportement de la politique FIFO. Les variables t doivent former, pour chaque scénario, un chemin complet de la racine au terminal. (17) assure que le chemin fait bien partie du MDD. (18-19) sont des contraintes de flot qui contraignent la sélection d'un chemin par scénario. (20) implémente la politique FIFO en contraignant le choix de l'arc sortant à celui dont la date de disponibilité est la plus petite. Les contraintes (21-22) établissent les dates de fin. A chaque couche, la tâche est commencée au plus tôt après la date de fin de la couche précédente et la date de disponibilité de la tâche lancée à cette couche. Enfin les contrainte (4-5) capturent l'objectif à partir des dates de fin.

Symétries Nous proposons en sus de briser certaines symétries identifiées dans le modèle :

La contrainte (28) brise la symétrie de sélection des nœuds actifs de chaque couche en forçant l'utilisation des nœuds d'indice inférieur en premier. Les contraintes (29-30) brisent partiellement la symétrie de permutation des nœuds actifs de chaque couche en imposant un ordre lexicographique sur les informations "All" de chaque

4. Notons que nous avons montré qu'il est suffisant d'établir la `Prec`-consistance : pour chaque chemin étiqueté B, il existe un chemin entrant ayant rencontré A. Cette contrainte nécessiterait toutefois l'introduction de variables pour représenter l'état *Some* de chaque nœud.

$$\min \quad z \tag{5.4}$$

$$z \geq \oplus\, \gamma(c_l^s) \tag{5.5}$$

$$x_{l,i,k}^j \leq y_{l,i} \qquad \forall l, i, k, j \tag{5.6}$$

$$x_{l,i,k}^j \leq y_{l+1,k} \qquad \forall l, i, k, j \tag{5.7}$$

$$\sum_{\forall j,k} x_{0,0,k}^j \geq 1 \tag{5.8}$$

$$\sum_{\forall j,k} x_{l,i,k}^j \geq y_{l,i} \qquad \forall l, \forall i \tag{5.9}$$

$$\sum_{\forall j,k} x_{l-1,i,k}^j \geq y_{l,k} \qquad \forall l, \forall i \tag{5.10}$$

$$\sum_{\forall j} x_{l,i,k}^j \leq 1 \qquad \forall l, i, k \tag{5.11}$$

$$\sum_{\forall k} x_{l,i,k}^j \leq 1 \qquad \forall l, i, j \tag{5.12}$$

$$x_{l-1,k,i}^{j'} \leq 1 - a_{l,i}^j + a_{l-1,k}^j \qquad \forall l, i, k, j \forall j' \neq j \tag{5.13}$$

$$a_{|N|+1,0}^j = 1 \qquad \forall j \tag{5.14}$$

$$a_{0,0}^j = 0 \qquad \forall j \tag{5.15}$$

$$a_{l,i}^A \geq \sum_{\forall k} x_{l,i,k}^B \qquad \forall (A,B) \in E, \forall l, i \tag{5.16}$$

$$t_{l,i,k,s}^j \leq x_{l,i,k}^j \qquad \forall s \forall l, i, k, j \tag{5.17}$$

$$\sum_{k,j} t_{0,0,k,s}^j = 1 \qquad \forall s \tag{5.18}$$

$$\sum_{k,j} t_{l-1,k,i,s}^j = \sum_{k,j} t_{l,i,k,s}^j \qquad \forall s, l, i \tag{5.19}$$

$$t_{l,i,k,s}^j \geq t_{l,i,k',s}^{j'} - (2 - x_{l,i,k,s}^j - x_{l,i,k',s}^{j'}) \qquad \forall s, \forall k, k' \forall j, j', r_j^s < r_{j'}^s \tag{5.20}$$

$$c_l^s \geq c_{l-1}^s + \sum_{\forall j,i,k} p_j^s t_{l,i,k,s}^j \qquad \forall s, l \tag{5.21}$$

$$c_l^s \geq \sum_{\forall j,i,k} (r_j^s + p_j^s) t_{l,i,k,s}^j \qquad \forall s, l \tag{5.22}$$

$$x_{l,i,k}^j \in \mathbb{B} \tag{5.23}$$

$$y_{l,i} \in \mathbb{B} \tag{5.24}$$

$$a_{l,i}^j \in \mathbb{B} \tag{5.25}$$

$$c_l^s \in \mathbb{R} \tag{5.26}$$

$$t_{l,i,k,s}^j \in \mathbb{B} \tag{5.27}$$

Modèle 5.1 – Modèle CP-COMPL

$$y_{l,i} \geq y_{l,i+1} \qquad \forall l, i \qquad (5.28)$$

$$a^0_{l,i} \leq a^0_{l,i+1} \qquad \forall l, i \qquad (5.29)$$

$$\bigwedge_{j=0}^{k} (a^j_{l,i} = a^j_{l,i+1}) \implies (a^{k+1}_{l,i} \leq a^{k+1}_{l,i+1}) \qquad \forall l, \forall i, \forall k \qquad (5.30)$$

Modèle 5.2 – Symétries du modèle CP-COMPL

nœud. Nous utilisons la décomposition "and" de Frisch et al. [2006] pour exprimer cette contrainte. On remarque qu'il est possible pour deux nœuds d'avoir les mêmes informations "All" sans qu'il soit possible de les fusionner (car l'information ignore l'ordre d'apparition des tâches dans les chemins menant au nœud). Les permutations de ces nœuds sont donc encore possibles.

Remarquons que lorsque W tend vers l'infini, le modèle tend effectivement vers l'implémentation de la stratégie complète (dont les décisions de premier niveau peuvent représenter n'importe quel ensemble de séquences), mais pour de petites valeurs de W, c'est une stratégie incomplète. Toutefois, à l'entraînement, quelque soit la décision de premier niveau $\mathcal{X}$, au maximum, une séquence différente sera extraite par FIFO pour chaque scénario dans S et évaluée. Une décision de premier niveau $\mathcal{X}'$ ne représentant que ces séquences ($|S|$ au maximum) est équivalente à $\mathcal{X}$ du point de vue de l'entraînement. En effet, retirer des séquences non-utilisées ne peut changer le résultat de la politique FIFO, à chaque moment de décision de la phase réactive, moins d'options sont disponibles, mais la tâche la plus prioritaire l'est toujours. Notons que ce n'est pas vrai pour une stratégie non-complète comme GSEQ, pour laquelle il n'est pas toujours possible de retirer certaines séquences sans en retirer d'autres que l'on souhaite conserver.

Par conséquent, il est garanti qu'il existe un MDD optimal à l'entraînement de largeur au plus $|S|$. Cependant, il n'est pas évident que celui-ci sera optimal lorsque évalué sur les scénarios de tests, au contraire, on pourrait penser qu'un phénomène de sur-entraînement risque de survenir, et que limiter la largeur W pourrait améliorer les performances en général. Dans la section suivante, nous tirons profit du nombre limité de séquences nécessaires à l'entraînement pour développer des approches plus simples basées sur la représentation de la stratégie complète par un ensemble de séquences.

5.3 Propriétés des stratégies complètes à l'entraînement et algorithme pour le critère $L_{\max}$

Puisque nous avons vu que lors de la phase d'entraînement, un nombre restreint séquences est suffisant, la question de la compacité de la représentation d'une dé-

cision de premier niveau complète n'y est pas pertinente. Nous considérons donc dans cette section l'implémentation de la stratégie complète dont la décision de premier niveau est représentée par un ensemble de séquences $\mathcal{X}$ plutôt que par un diagramme de décision multivalué. Il sera toujours possible de passer d'une représentation "ensemble de séquence" à une représentation "MDD" équivalente de taille raisonnable par compilation des séquence.

5.3.1 Ordre lexicographique et front des séquences

Il est intéressant, afin d'obtenir une nouvelle perspective sur le problème, de définir un ordre lexicographique des séquence pour chaque scénario $s \in S$. On définit l'ordre $<_s$ entre séquence à partir de l'ordre des dates de disponibilité des tâches, et en comparant les tâches des séquences position par position dans l'ordre d'apparition. Par exemple, soit un problème 1P à trois tâches tels que pour un scénario s, $r^s_B < r^s_A < r^s_C$. L'ordre lexicographique associé vérifie que $BAC <_s BCA <_s \cdots <_s CBA <_s CAB$. Dans le cas où deux tâches ont une date de disponibilité identique, l'ordre est départagé par l'ordre lexicographique des tâches ($A < B < C$).

Étant donné cet ordre, si l'on considère la séquence retournée par FIFO, on remarque :

Proposition 5.3.1.1. $FIFO(\mathcal{X}, s) = min_{<_s}\mathcal{X}$

Dans l'exemple de la figure 5.7, $\mathcal{X} = \{ACB, ABC, BAC\}$. Comme s_1 vérifie $r_A < r_B < r_C$, on a $ABC <_{s_1} ACB <_{s_1} BAC$. De même on a pour les autres scénarios $BAC <_{s_2} ACB <_{s_2} ABC$, et $BAC <_{s_3} ABC <_{s_3} ACB$.

Pour un ensemble de scénarios donné, nous nommons le *front*[5] de $\mathcal{X}$ (noté $\overline{\mathcal{X}}$) le sous ensemble de ses séquences $\{min_{<_s}\mathcal{X}, \forall s \in S\}$. Seul le front est utilisé dans l'évaluation d'une décision de premier niveau. Dans notre exemple, précédent avec 3 scénarios, le front est composé de deux séquences : $\overline{\mathcal{X}} = \{ABC, BAC\}$.
On remarque au passage que pour chaque séquence $\sigma \in \mathcal{X}$, il existe un scénario $s_\sigma \in \overline{S}$ pour lequel $\sigma = min_{<_{s_\sigma}}\mathcal{X}$: il suffit que $r^{s_\sigma}_i < r^{s_\sigma}_j$ si $\sigma_i < \sigma_j$. Dans nos expériences, étant donné suffisamment de scénario de tests tirés d'une distribution à support non-borné (comme une loi normale ou une loi exponentielle) la probabilité d'une séquence particulière de faire partie du front de test tend vers 1.

Étant donné deux décisions de premier niveau (ensembles de séquences) $\mathcal{X}$ et $\mathcal{X}'$, il est possible de déterminer facilement le front de l'union des ensembles de séquences $\overline{\mathcal{X} \cup \mathcal{X}'}$ et donc d'évaluer $\mathcal{X} \cup \mathcal{X}'$ en comparant les fronts $\overline{\mathcal{X}}$ et $\overline{\mathcal{X}'}$ selon l'équation suivante :

$$min_{<_s}\mathcal{X} \cup \mathcal{X}' = min_{<_s}(min_{<_s}\mathcal{X}, min_{<_s}\mathcal{X}')$$

5. Dans l'absence de précision contraire, le front réfère au front défini par les scénarios d'entraînement

5.3.2 Un algorithme polynomial d'extraction du meilleur sous-ensemble de séquences

Dans le cas ou $\oplus = \max$ (correspondant à l'optimisation robuste), étant donné un ensemble de séquences $\mathcal{X}$, nous proposons un algorithme polynomial en $|\mathcal{X}|$ pour extraire la meilleure décision de premier niveau $\mathcal{X}^* \subseteq \mathcal{X}$, soit

$$\mathcal{X}^* = \underset{X \subseteq \mathcal{X}}{\operatorname{argmin}} \max_{s \in S} \gamma(FIFO(X, s), s) \tag{5.31}$$

Algorithme 5.1 Algorithme "Best-Of" d'extraction de la meilleure sous-décision pour $\oplus = max$

1: Depuis une solution de départ $\mathcal{X}$
2: $\mathcal{X}^* = \mathcal{X}$
3: $B^* = \max_s \gamma(FIFO(\mathcal{X}, s))$
4: **tant que** $|\mathcal{X}| > 0$ **faire**
5: $\quad \underline{s} = \operatorname{argmax}_s \gamma(FIFO(\mathcal{X}, s), s)$
6: $\quad \underline{\pi} = \min_{<_{\underline{s}}} \mathcal{X}$
7: $\quad \mathcal{X} = X \setminus \{\underline{\pi}\}$
8: $\quad$ **si** $\max_s \gamma(FIFO(\mathcal{X}, s), s) < B^*$ **alors**
9: $\quad\quad \mathcal{X}^* = \mathcal{X}$
10: $\quad\quad B^* = \max_s \gamma(FIFO(\mathcal{X}, s), s)$
11: Return $\mathcal{X}^*$

L'algorithme 5.1 commence par initialiser la meilleure solution jusqu'à présent $\mathcal{X}^*$ et son score B^* obtenu en prenant la pire valeur de l'objectif obtenue par FIFO sur l'ensemble des scénarios (2-3). Ensuite, tant que l'ensemble des séquences n'est pas vide, l'algorithme sélectionne la séquence limitante $\underline{s}$ pour le score (i.e. la séquence que FIFO choisit dans le scénario pire cas) et la supprime de l'ensemble (5-7). A chaque itération, la nouvelle solution est évaluée et la meilleure solution trouvée mise à jour le cas échéant (8-10). L'algorithme termine en retournant la meilleure solution obtenue $\mathcal{X}^*$ Une décision équivalente contenant moins de séquences est son front $\overline{\mathcal{X}^*}$. Cet algorithme à une complexité en $O(|S||\mathcal{X}|^2)$ (qui peut être réduite en $O(|S||\mathcal{X}| \log(|\mathcal{X}|))$ en pré-ordonnant les séquences pour chaque scénario).

	$<_s$		
$s_1 : r_A < r_B < r_C$	4	5	~~2~~
$s_2 : r_C < r_B < r_A$	~~1~~	5	6
$s_3 : r_B < r_A < r_C$	~~6~~	3	2

FIGURE 5.7 – Illustration de l'algorithme Best-Of ($|S| = 3$, $\mathcal{X} = \{ACB, ABC, BAC\}$)

La figure 5.7 illustre une étape du fonctionnement de l'algorithme pour un exemple fictif avec trois scénarios. La figure montre, pour chaque scénario, le score

obtenu pour chaque séquence de l'ensemble $\mathcal{X}$. Chaque séquence (indiquée par une couleur) est positionnée selon sa priorité induite par $<_s$ pour chaque scénario. Le front se lit donc donc en prenant, pour chaque scénario, la séquence la plus à gauche. Initialement, le front est $\overline{\mathcal{X}} = \{ABC, BAC\}$. Lors de la première itération, $\underline{s} = s_3$ avec $\gamma(FIFO(BAC, s_3)) = 6$. La séquence BAC est supprimée (en vert dans la figure), et le front devient $\overline{\mathcal{X}} = \{ABC, ACB\}$, avec un score de 5. Ce front est optimal pour l'exemple, mais on remarque que le front $\overline{\mathcal{X}} = \{ACB\}$ le serait aussi.

Proposition 5.3.2.1. *L'algorithme de Best-Of retourne bien le meilleur sous-ensemble $\mathcal{X}^*$ tel que défini dans l'équation 5.31.*

Démonstration. Soit $\mathcal{X}'$ la solution courante ($\mathcal{X}$ dans l'algorithme lors d'une itération donnée), soit $\mathcal{X}^*$ et B^* la meilleure solution et le meilleur score obtenu jusqu'a présent, et soit $\underline{\Pi} = \mathcal{X} \setminus X'$ l'ensemble des séquences supprimées jusqu'à présent. Réinsérer des séquences de $\underline{\Pi}$ dans $\mathcal{X}'$ ne peut mener à une solution meilleure que $\mathcal{X}^* : \forall \mathcal{Z} \subseteq \underline{\Pi}, \max_s \gamma(FIFO(\mathcal{X}' \cup \mathcal{Z}, s)) \geq B^*$. En effet, à chaque itération, est enlevée une séquence du front (le $min_{<_s}$ pour un scénario s) dont le score est au moins aussi mauvais que la borne courante $B' \geq B^*$. Considérons la séquence $\underline{\pi} \in \mathcal{Z}$ ayant été supprimée la première. Il existe $\underline{s}$ tel que $\underline{\pi} = min_{<_s}\mathcal{X}' \cup \mathcal{Z}$ et $\gamma(\underline{\pi}, \underline{s}) \geq B^*$, donc $\max_s \gamma(FIFO(\mathcal{X}' \cup \mathcal{Z}, s), s) \geq B^*$. En particulier lorsque l'algorithme termine, $|\mathcal{X}'| = 0$, $\underline{\Pi} = \mathcal{X}$, et par conséquent $\forall \mathcal{Z} \subseteq \mathcal{X}, \max_s \gamma(FIFO(\mathcal{Z}, s), s) \geq B^* : \mathcal{X}^*$ est un sous-ensemble optimal. □

5.3.3 Un algorithme génétique basé sur l'algorithme Best-Of : AG-COMPL

Toujours dans le cas où $\oplus = max$, nous proposons un algorithme génétique faisant usage de l'algorithme de Best-Of pour résoudre le problème de recherche de l'ensemble optimal de séquences comme solution de premier niveau.

Il est basé sur l'idée qu'étant donnée une population de séquences, il est facile d'en extraire le meilleur sous-ensemble. Par conséquent, le coeur du problème réside en la génération d'un ensemble de séquences de bonne qualité, et non pas dans le choix des séquences à intégrer dans la solution de premier niveau.

L'algorithme reprend la structure classique décrite dans l'algorithme 1.2. La population est initialement constituée d'un ensemble de séquences aléatoires ainsi que d'une solution JSEQ dans le cas d'un démarrage à chaud. La présence de contraintes de précédence rend toutefois certaines de ces séquences invalides. Elles sont alors réparées selon le principe présenté dans la section 4.3.2.3, en repoussant les tâches impliquées à des positions acceptables.
A chaque itération, les séquences sont croisées en utilisant le croisement à un point. La séquence-enfant générée est mutée dans le voisinage défini par l'inversion de deux tâches consécutives aléatoires dans la séquence, puis éduquée par descente de gradient dans ce même voisinage, et ajoutée à la population. Lorsque la taille de

la population dépasse la taille maximum autorisée, le processus de sélection est enclenché. Premièrement, l'algorithme Best-Of est utilisé pour déterminer le meilleur front $\overline{\mathcal{X}}$ et son score B^*. Sont supprimées de la population toutes les séquences dont le score est pire que B^* dans tous les scénarios. La nouvelle population se compose des séquences du front, d'une sélection aléatoire parmi les séquences de la population précédente, et d'une part de nouvelles séquences aléatoires.
Le processus est répété jusqu'au temps limite. L'output de l'algorithme est le meilleur front obtenu lors de la résolution.

5.4 Résultats

Nous présentons dans cette section les résultats obtenus par les implémentations de la stratégie complète présentées dans ce chapitre, et les comparons aux stratégies des chapitres précédents. Le modèle de PPC présenté dans la section 5.2.3 est nommé CP-COMPL, tandis que l'algorithme génétique présenté dans la section 5.3.3 est nommé AG-COMPL.

Sauf indication spécifique, par défaut, CP-COMPL n'implémente pas les contraintes anti-symétries, et le paramètre de largeur maximale W est fixé à 2. Le temps de chauffe est fixé à $INIT = 90$ par défaut pour AG-COMPL*.

Comme précédemment, toutes les méthodes s'exécutent dans une limite de temps de 2 heures, sur un unique processeur (Xeon E5-2695 v4 @ 2.10GHz). Les modèles de PPC sont résolus par IBM CP Optimizer 20.1. Les heuristiques sont implémentées en Python, tandis que les modèles PPC utilisent l'API Python de CPO.

5.4.1 Résultats de CP-COMPL

5.4.1.1 Performances de CP-COMPL

Les performances de CP-COMPL peuvent être observées dans la table A.3, pour le cas où $|N| = 10$ seulement, car dès $|N| = 20$, le modèle n'a été capable d'obtenir une solution que pour une seule instance, les autres instances dépassant la limite mémoire de 16Go. On observe pour $|N| = 10$ une performance raisonnablement bonne, puisque meilleure que celle de CP-JSEQ, mais moins bonne que celle de CP-GSEQ. Outre les faibles performances du modèle, il faut rappeler qu'en limitant la largeur du MDD ($W = 2$), on implémente une stratégie incomplète, qui ne domine pas la stratégie GSEQ, ce qui peut aussi expliquer les résultats. L'optimalité n'étant prouvée pour aucune instance, on ne peut conclure avec certitude.

5.4.1.2 Impact des symétries

La table 5.1 décrit l'impact des contraintes 5.28 (S1) et 5.30 (S2) sur les performances à l'entraînement du modèle CP-COMPL en fonction de la largeur maximale

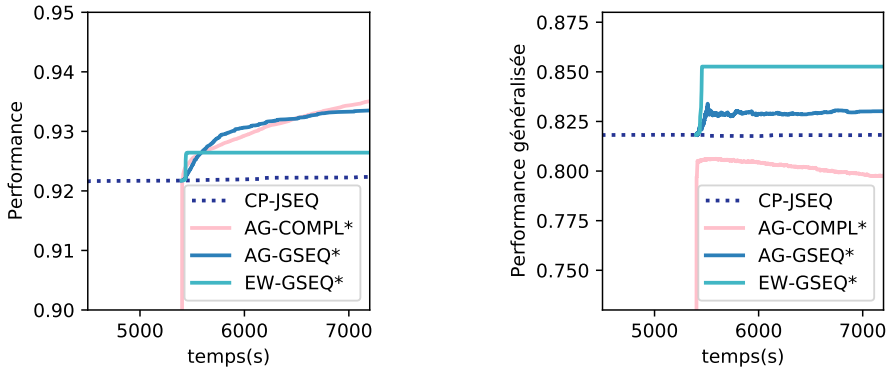
W. On remarque que lorsque $W = 2$, les contraintes ne bénéficient pas aux performances du modèle, mais elles ont toutes les deux un impact positif lorsque $W = 5$. En effet, pour des petites largeurs du MDD, le facteur de symétrie n'est pas très important (à l'extrême, il n'y a pas de symétries lorsque $W = 1$), mais le nombre de symétries augmente selon 2^W pour S1 (le nombre de façons de sélectionner des noeuds sur une couche) et $W!$ pour S2 (les permutations des nœuds sélectionnés). On peut donc s'attendre à ce que l'impact de S2 soit encore plus marqué pour de plus grandes valeurs de W.

Contraintes W	∅	{S1}	{S2}	{S1,S2}
2	0.9326	0.9108	0.9275	0.9205
5	0.8414	0.9047	0.8965	0.9135

TABLE 5.1 – Performance à l'entraînement pour CP-COMPL, avec différentes combinaisons de contraintes anti-symétries, en fonction de la largeur maximale du MDD (instances 1P-N avec $|N| = 10$)

5.4.2 Résultats de AG-COMPL

5.4.2.1 Performance à l'entraînement/ en test



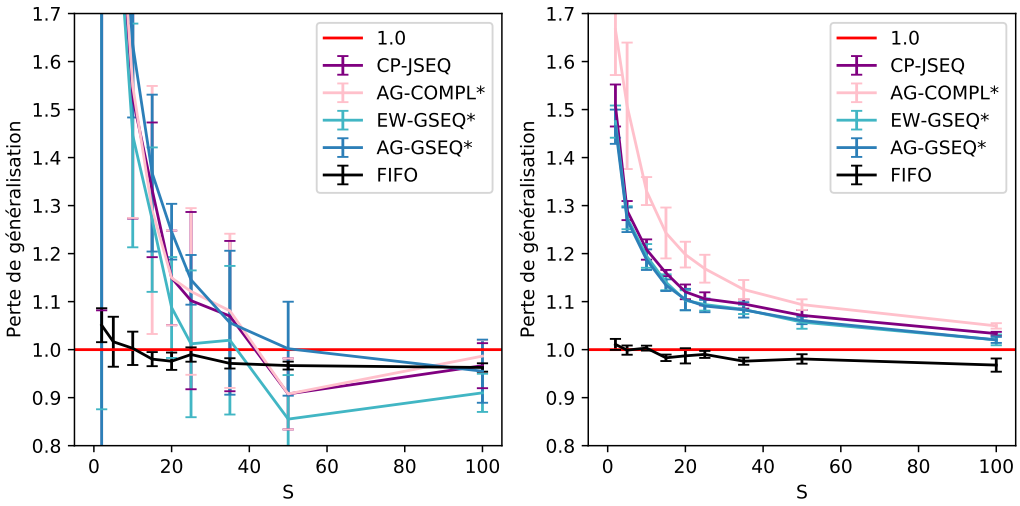
(a) 1P-A (Entraînement) (b) 1P-A (Test)

FIGURE 5.8 – Performance en fonction du temps à l'entraînement et en test pour les méthodes à chaud ($\oplus = max$)

La figure 5.8 illustre le comportement sur les instances standard (1P-A) de la méthode AG-COMPL* comparée aux méthodes GSEQ à chaud. On observe qu'à l'entraînement, les performances de AG-COMPL* sont comparables à celles des meilleures méthodes GSEQ à chaud, le détail des tables A.21,A.23 et A.22 révèle qu'à l'entraînement, AG-COMPL* obtient presque toujours de très bonnes, voir les meilleures performances. En particulier, on remarque (table A.21) que de bonnes

performances à l'entraînement sont obtenues pour $\gamma = \sum C_i$ comparativement aux autres méthodes. Pour $\gamma = L_{MAX}$, les résultats obtenus sont un peu inférieurs, mais il faut se rappeler que pour la plupart de ces instances, aucun gain n'est possible à l'entraînement comparé à la stratégie JSEQ (cf figure 3.7). De bonnes performances sont obtenues à l'entraînement en particulier pour des instances avec $|N| \leq 100$, pour lesquelles AG-COMPL* obtient les meilleures performances (table A.22), mais la méthode reste compétitive pour les plus grandes instances.

Toutefois, les performances généralisées en test ne reflètent pas cette compétitivité, on observe sur la figure 5.8 que dans le cas standard, en moyenne les solutions obtenues ne font qu'empirer au cours du temps. En effet, on peut voir sur les tables A.21, A.23 et A.22 que les performances généralisées de la méthode sont au final moins bonnes que celles de EW-GSEQ* en moyenne pour toutes les valeurs des paramètres considérés.



(a) $\gamma = L_{MAX}$ (b) $\gamma = \sum C_i$

FIGURE 5.9 – Perte de généralisation pour différentes méthodes en fonction du nombre de scénarios d'entraînement ($\oplus = max$)

La figure 5.9 montre la perte de généralisation de la méthode AG-COMPL* ainsi que d'une sélection de méthode des chapitres précédents. On remarque que la perte de généralisation de AG-COMPL* est significativement plus importante que celle des autres méthodes pour $\gamma = \sum C_i$, cela explique l'inversion des tendances entre les performances à l'entraînement et en test des méthodes à chaud.

5.5 Discussion

Dans ce chapitre, nous avons présenté deux implémentations de la stratégie complète qui représentent les décisions de premier niveau de deux façons différentes :

par les diagrammes de décision multivalués et par les ensembles de séquences.

Nos travaux sur les MDDs se distinguent de la plupart des applications usuelles des diagrammes de décision puisque nous ne considérons pas les MDDs comme des outils mais comme le produit : la décision partielle de premier niveau que nous cherchons. Nous avons établi qu'il n'était pas possible d'exploiter pour notre approche à deux niveaux avec la politique FIFO les propriétés de relâchement habituelles des MDDs qui semblent constituer l'attrait majeur de ces structures dans la littérature. Toutefois, Les diagrammes de décisions restent une façon intuitive de représenter un ensemble de séquences tout en permettant parfois un gain en espace par rapport aux ensembles de séquences ou aux arbres.

Les mauvaises performances du modèle CP-GSEQ, traduisant l'explosion de l'espace de recherche associé, semble trouver un echo amplifié dans les performances du modèle CP-COMPL. Cela n'est pas très surprenant, puisque sans même considérer les difficultés de construction spécifiques aux diagrammes de décision, la taille de l'espace des solutions, c'est-à-dire le nombre d'ensembles de séquences qu'il est possible de représenter, est en $O(|N|!)$ pour la stratégie JSEQ, $O(|N|^{|N|})$ pour la stratégie GSEQ, et $O(2^{|N|!})$ pour la stratégie complète en général. Toutefois, on pouvait s'attendre à ce que le paramètre W de largeur maximale réduise suffisamment l'espace de recherche, mais ce ne semble pas être le cas pour le modèle actuel. De plus, nous avons vu qu'à l'entraînement, l'espace de recherche peut être réduit en $O(\binom{|N|!}{S})$. Les performances de l'algorithme AG-COMPL*, qui tire partie de cette réduction de l'espace de recherche, semblent prometteuses à l'entraînement mais sont décevantes en test, ce qui montre que la bonne corrélation entre l'amélioration du score à l'entraînement et en test pour un nombre relativement restreint de scénarios d'entraînement n'est pas garantie, et une attention particulière doit être portée au choix des solutions représentées pour éviter le sur-apprentissage. Comme nous l'avons vu dans le chapitre 4, un ajout aveugle de flexibilité risque de ne pas être efficace, mais puisque la stratégie GSEQ par exemple est capable de tirer partie de la phase d'entraînement pour améliorer les performances d'une solution JSEQ avec un démarrage à chaud, la stratégie complète doit aussi en être capable. Ajouter un ensemble de séquences au front est une piste envisageable pour améliorer les performances des solutions en test, toutefois, si le nombre de séquences à rajouter est trop important, il est possible que la taille de la représentation de l'ensemble de séquences explose. Dans ce cas, la représentation de l'ensemble par un MDD pourrait être intéressante.

Troisième partie

Mise en contexte théorique avec la compilation de connaissances

Nous avons vu dans la partie I que les solutions flexibles, représentant des ensembles de solutions, permettent d'anticiper les effets négatifs de l'incertitude, dont l'impact est notable sur les problèmes d'ordonnancements. Nous avons également répertorié plusieurs structures de données, utilisées dans des problèmes d'ordonnancement pour représenter ces ensembles de solutions. Dans la partie II, nous avons étudié une famille d'approches proactives-réactives intégrées qui tentent de minimiser un objectif étant donné la connaissance d'un paramètre incertain et d'un ensemble de scénarios. Ces approches utilisent des structures de données, comme les séquences de groupes d'opérations permutables ou les diagrammes de décision multivalués, pour constituer une décision partielle représentant un ensemble de séquences. A cette occasion nous avons remarqué que le choix de la structure utilisée impacte les ensembles de solution qu'il est possible de représenter, mais peut aussi impacter la taille de cette représentation.

Les propriétés impliquées par différentes façon de représenter des instances d'un problème sont le champ d'études de la *compilation de connaissances*. Dans le formalisme de ce domaine, les structures de données peuvent être vues comme des **langages de représentation**. Ces langages seront définis formellement dans la suite, mais il est opportun de réaliser comment une famille de structures (par exemple, les séquences de groupes d'opérations permutables) est un langage dont la **syntaxe** définit les mots bien formés (pour les GSEQ, une partition ordonnée des tâches). La **sémantique** du langage associe à chaque mot son interprétation (une GSEQ *représente* un ensemble de séquences).
L'objectif de la compilation de connaissance est de factoriser le fardeau de la complexité d'un problème par la **compilation** des données d'une instance depuis un langage source (dans lequel une opération est difficile) vers un langage cible (dans lequel une opération est plus facile). Cette méthode est particulièrement adaptée lorsque l'on souhaite répéter l'opération de nombreuses fois, mais est aussi pertinente dès lors que le coût du temps de calcul en amont est moindre que le coût en aval (comme c'est le cas par exemple dans des approches proactives-réactives). Une représentation compilée d'un problème permet aussi à un décideur d'interagir en temps réel avec le problème, pour simuler par exemple l'impact de décisions ou prendre en compte de nouvelles contraintes conséquences de circonstances imprévues. C'est une façon plus générale d'anticiper des aléas divers, qui peut faire usage de la flexibilité proposée par un ensemble de solutions que nous avions évoquée.

Pour déterminer quelles représentations sont les plus adaptées, les langages sont comparés selon trois aspects : la capacité à représenter des instances d'un problème, la capacité à le faire de façon compacte, et la complexité des opérations d'intérêt sur ces représentations. A travers l'étude des propriétés comparées des langages de représentation, nous espérons aussi obtenir une meilleure compréhension des mécanismes de la représentation d'ensembles de séquences, et par ce biais permettre l'amélioration des approches à deux niveaux présentées dans la partie II.

Dans la partie suivante, nous étudions plusieurs des structures de données utilisées pour les problèmes d'ordonnancements, et en particulier celles utilisées dans notre approche à deux niveaux, sous l'angle de la formalisation apportée par la compilation de connaissance.
Dans le chapitre 6, nous introduisons des éléments de compilation de connaissance nécessaires à la bonne compréhension de l'étude menée. En particulier, nous commençons par présenter formellement le concept de langage de représentation. Nous proposons ensuite une équivalence entre la représentation d'ensembles de séquences et d'ordonnancements, ce qui nous permet de décrire comment les différents langages sont comparés entre eux. Au cours de cette introduction, nous présentons le langage source du problème à une machine étudié $(1|prec, r_i, \tilde{d}_i|-)$ [6], et les langages cibles candidats à la compilation. Nous définissons également les requêtes d'intérêt dans notre étude.

Dans le chapitre 7, nous mettons à profit le formalisme définit dans le chapitre 6 pour comparer les différents langages présentés entre eux. Nous commençons par présenter les résultats d'expressivité des langages, puis les résultats portant sur leurs compacité, avant de terminer par l'étude comparée des requêtes qu'ils supportent. Nous proposons une première carte de compilation des langages de représentation pour les problèmes d'ordonnancements, et tirons quelques conclusions quant à leur utilisation.

6. Remarquons que c'est un problème similaire au 1P de la partie précédente, mais les dates de livraison y sont strictes

Chapitre 6

Introduction à la compilation de connaissances pour les problèmes d'ordonnancement

Sommaire

Dans ce chapitre, nous commençons par développer les motivations évoquées précédemment pour l'introduction de techniques de compilation de connaissances pour les problèmes d'ordonnancement, et situons nos travaux par rapport aux travaux similaires de la littérature. La suite du chapitre s'organise autour de l'exemple de la compilation d'un problème à une machine vers le langage des séquences de groupes d'opérations permutables. Nous rappelons d'abord les définitions des variables de décision de dates et de séquence que nous utilisons. Puis présentons formellement le concept de langage. A ce stade, l'exemple montrera la nécessité de l'introduction de la notion de translation pour la comparaison de langages basés sur différentes variables de décisions. Nous définissons ensuite la façon dont les langages sont comparés à l'aide de la translation. Nous terminons par la présentation des autres langages que nous étudions, ainsi que les requêtes intéressantes dans un contexte d'ordonnancement sous incertitude.

La figure 6.1 décrit le principe général de la compilation de connaissances. Une instance du problème d'intérêt est décrite formellement par une formule φ_1 dans un langage L_1, et un décideur souhaite par exemple connaître une solution à cette instance, mais cette opération est difficile dans ce langage. Une étape de compilation préalable permet de transformer la formule φ_1 en une formule φ_2 d'un autre langage L_2, pour lequel cette opération est ensuite facile (dans ce cas, l'étape de compilation doit donc être difficile).

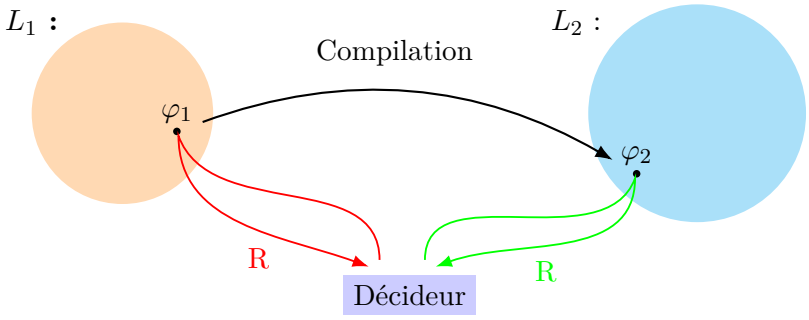


FIGURE 6.1 – Principe de la compilation de connaissances

6.1 Introduction

6.1.1 Motivation

Comme évoqué précédemment, nous nous intéressons dans ce chapitre à un problème d'ordonnancement à une machine ($1|prec, r_i, \tilde{d}_i|-$ dans la notation de Graham). Dans ce problème, un ensemble N de tâches doivent être ordonnancées

sans préemption ni chevauchement. Chaque tâche a une date de disponibilité r_i et une date de livraison stricte $\tilde{d}_i$ avant laquelle elle doit terminer pour que l'ordonnancement soit admissible (on notera simplement d_i dans la suite). On considère aussi un ensemble de contraintes de précédence E.

Déterminer un ordonnancement admissible pour ce problème est connu pour être NP-difficile (puisque le problème de minimisation du retard est difficile (Lenstra et al. [1977]). Dans un contexte d'incertitude, cela implique qu'il ne sera pas toujours possible de le résoudre efficacement lorsque des aléas modifieront le problème. Pour cette raison, les techniques de la compilation de connaissance semblent pertinentes. Notons que contrairement à la partie II, nous ne considérons pas un ensemble de scénarios discret, mais plutôt des valeurs nominales de paramètres, que nous souhaitons pouvoir modifier dynamiquement dans une phase réactive. L'idée étant qu'il est peut-être possible de pré-calculer hors-ligne une composante clé du problème hors-ligne, ce qui permettra ensuite de ré-ordonnancer facilement en ligne les tâches pour intégrer de nouvelles informations ou contraintes.

A terme, l'objectif visé est de fournir au décideur une structure de données représentant le problème. Cette structure devrait permettre au décideur d'effectuer des opérations facilement, comme obtenir un ordonnancement admissible au problème, simuler et suivre l'exécution d'un ordonnancement, mettre à jour le problème pour prendre en compte l'occurrence d'aléas. Nous nommons ces opérations des **requêtes**, auxquelles des algorithmes doivent fournir une réponse satisfaisante a partir des informations contenues dans la structure de données. La nature de la structure de données est donc cruciale pour déterminer la facilité avec laquelle les requêtes peuvent être exécutées, et donc, dans un contexte dynamique, les requêtes qui peuvent raisonnablement être supportées.

6.1.2 Travaux similaires

Le domaine de recherche de la compilation de connaissance étudie les capacités théorique de différentes structures de données pouvant représenter par exemple des solutions d'un problème. Pourtant, il n'existe à notre connaissance que très peu de travaux qui tentent d'appliquer le formalisme et les méthodes de la compilation de connaissance aux problèmes d'ordonnancement. En particulier la seule application directe de ce formalisme à des problèmes d'ordonnancements dont nous ayons connaissance sont les travaux de Dymitrowski [2020], qui tentent de représenter des ensembles d'ordonnancements admissibles pour le problème du RCPSP, en utilisant des diagrammes de décision multivalués et des graphes de décision multivalués décomposables. Ces structures supportent effectivement de nombreuses opérations d'intérêt, comme la capacité d'ajouter de nouvelles contraintes de précédence, la possibilité de prendre en compte un retard de livraison ou une indisponibilité de ressource ... Toutefois, les instances de plus de 30 tâches n'ont pu être compilées en utilisant des représentations basées sur la discrétisation du domaine temporel des dates de débuts des tâches.

Nous avons vu dans le chapitre 2 que de nombreuses structures de données différentes sont utilisées pour représenter des séquences dans le cadre de la résolution de problèmes d'ordonnancement. Nous avons insisté dans la partie II sur les structures des séquences de groupes d'opérations permutables et des diagrammes de décision multivalués. Nous analyserons formellement certaines de ces structures dans cette partie.

Le domaine de la compilation de connaissance s'intéresse à la représentation de problèmes dans des structures de données alternatives. Ces types de structures de données sont appelées "langages" dans ce contexte : ils sont définis par leur représentation (syntaxe), interprétation (sémantique) et leurs taille. Ce formalisme permet de comparer ces structures de données par l'intermédiaire de leurs propriétées.

Les travaux de Gogic et al. [1995] décrivent comment deux langages peuvent être comparés : suivant leur *expressivité* (qu'est ce qu'ils permettent de représenter), de leur *compacité* (quelle est la taille de ces représentations), et leur capacité à *supporter des requêtes* (quelles est la complexité des opérations sur ces structures). Cette comparaison souligne les avantages et inconvénients des différents langages, en fonction de l'utilisation qu'il est prévu d'en faire. La comparaison de plusieurs langages peut être résumée dans ce que l'on appelle une *carte de compilation*, tirant parti de la transitivité des relations d'expressivité et de compacité. Une telle carte est dressée dans Fargier et al. [2015] dans le contexte de problèmes de planification, proches des problèmes d'ordonnancements. Les auteurs proposent une carte de compilation pour quelques langages temporels tels que les problèmes temporels simples (STP : *simple temporal problems*), les problèmes de satisfaction de contraintes temporelles (TCSP : *temporal constraint satisfaction problem*), les problèmes temporels disjonctifs (DTP : *disjunctive temporal problem*), et divers sous-langages des DDDs (*difference decision diagrams*) proposés par Møller et al. [1999]. Ces langages représentent des fonctions booléennes sur des variables de décision de date qui peuvent être comprises comme un ensemble de contraintes liants les valeurs relatives de points temporels. Les STP contraignent simplement l'écart entre des paires de points temporels à un intervalle donné, tandis que les TCSP, les DTP et les DDD permettent des contraintes plus complexes, en particulier en contraignant l'écart entre deux points à une disjonction d'ensembles possibles. Les conclusions de l'article sont pessimistes concernant les langages qu'il présente. En effet, les langages décrits sont soit trop peu expressifs, soit ne supportent pas une requête essentielle compte tenu d'une application en planification (établir l'existence d'une solution admissible, extraire un plan, et en suivre l'exécution).

Dans d'autres travaux en planification, il est courant de passer outre la complexité de manipulation des TCSP en les divisant en un ensemble de taille potentiellement exponentielle de STPs (pour chaque intervalle disjoint d'un TCSP). Cela permet le support de plusieurs requêtes, mais une taille exponentielle en fonction du nombre de contraintes disjonctives. Afin d'améliorer les performances, Shah and Williams [2008] proposent une représentation plus compacte des TCSPs. Bien que les résultats expérimentaux montrent que la méthode permet de réduire la taille mémoire et d'améliorer les performances (d'un facteur allant jusqu'à trois ordres de magni-

tude), certaines requêtes essentielles ne sont pas supportées. Des travaux ultérieurs généralisent l'approche aux DTPs [Conrad and Williams, 2011], au prix d'un support encore moindre des requêtes (en particulier, établir l'existence d'une solution devient difficile).

Habituellement, les langages que l'on compare à travers ce formalisme représentent les mêmes objets. Classiquement, ces objets sont des fonctions à valeurs booléennes, qui peuvent représenter l'ensemble des solutions d'un problème. Cependant, nous avons vu que pour un même problème, on peut considérer plusieurs objets pour constituer une solution (plusieurs types de variables de décision). Les travaux de Fargier et al. [2013] présentent le concept de *compilation hétérogène*, formalisant la correspondance implicite entre des objets de types différents (comme par exemple les variables de décision entières avec les variables de décision binaires correspondant à leurs encodage binaire) pour la comparaison de langages. Nous utilisons ce formalisme pour comparer des représentations temporelles de solutions (basée sur les variables de décision de date de début des tâches) et les représentation séquentielles (basée sur les variables de décision de séquence).

6.2 Variables de décision

Nous avons vu dans le chapitre 1 plusieurs types de variables de décision pour la représentation de solutions. Nous définissons dans cette section formellement les variables de décision et décrivons plus en détail les variables de décision que nous utilisons.

Formellement, soit $\chi = \{x_1, x_2, \cdots, x_k\}$ un ensemble de k **variables de décisions**. Le domaine de chaque variable est D_{x_i} et pour chaque sous-ensemble de variables $X \subseteq \chi$, on note $\check{x}_X$ une affectation des variables de décision de X à une valeur dans leur domaine $D_X = \prod_{x \in X} D_x$.

Bien que de nombreux ensembles de variables de décision puissent êtres considérés (voir chapitre 1), nous nous concentrons sur seulement quelques-uns d'entre eux.

6.2.1 Variables de décision de dates

Pour un ensemble de $|N|$ tâches, l'ensemble des **variables de décision de dates** est l'ensemble $\chi_{date} = \{s_1, s_2 \dots s_{|N|}\}$. Pour chaque tâche j, la variable s_j représente sa date de début. A chacune de ces variables peut être affectée une valeur dans son domaine $D_{s_j} : \mathbb{R}^+$. Une affectation de toute les variables de décision, notée $\check{s}_\chi = (\check{s}_1, \check{s}_2, \dots, \check{s}_{|N|})$, représente un *ordonnancement*.

Par exemple, pour un problème à trois tâches $N = \{A, B, C\}$ [1] l'ordonnancement $\check{s} = (\check{s}_A, \check{s}_B, \check{s}_C) = (1, 2, 0)$ décrit une date de début pour chaque tâche.

6.2.2 Variables de décision de séquence

Pour un ensemble de $|N|$ tâches, l'ensemble des **variables de décision de séquence** est l'ensemble $\chi_{seq} = \{\sigma_1, \sigma_2 \dots \sigma_{|N|}\}$. Pour chaque tâche j, la variable σ_j représente sa position dans la séquence. A chacune de ces variables peut être affectée une valeur dans son domaine $D_{\sigma_j} : [\![1, |N|]\!]$. On suppose qu'à deux tâches distinctes on ne peut affecter la même position : $\check{\sigma}_i \neq \check{\sigma}_j \ \forall (i, j) \in N^2, i \neq j$. Une affectation de toute les variables de séquence, notée $\check{\sigma}_\chi$, représente une *séquence* : un ordre sur les tâches.
Par exemple la séquence $\check{\sigma} = (\check{\sigma}_A, \check{\sigma}_B, \check{\sigma}_C) = (2, 3, 1)$ décrit la position de chaque tâche. Par commodité, on note aussi une séquence par la représentation des tâches dans l'ordre définit par leurs positions (soit "CAB" pour l'exemple précédent).

Une séquence $\check{\sigma}$ de taille $|N|$ définit aussi $2^{|N|}$ **sous-séquences** $\check{\sigma}_{|E}$ correspondant à la sélection d'un sous-ensemble E des tâches et définit par

$$\check{\sigma}_{|E,i} = |\{\check{\sigma}_j \leq \check{\sigma}_i, j \in E\}|, \forall i \in E$$

Par exemple, $\check{\sigma}_{|\{A\}} = (1)$ (la séquence "A") et $\check{\sigma}_{|\{B,C\}} = (2, 1)$ (la séquence "CB") sont des sous-séquences de "CAB". On parle aussi de projection sur E.
De plus, on définit la **concaténation** des séquences $\check{\sigma}_{E_1}$ et $\check{\sigma}_{E_2}$ (pour deux ensembles distincts de variables de séquence E_1 et E_2) par $\check{\sigma}_{E_1}.\check{\sigma}_{E_2}$avec

$$\check{\sigma}_{E_1}.\check{\sigma}_{E_2} = \begin{cases} \check{\sigma}_{E_1,i} & \forall i \in E_1 \\ \check{\sigma}_{E_2,i} + |E_1| & \forall i \in E_2 \end{cases}$$

Par exemple, pour deux ensembles $E_1 = \{\sigma_A, \sigma_B\}$ et $E_2 = \{\sigma_C, \sigma_D\}$, si on a $\check{\sigma}_{E_1} = (1, 2) = AB$ et $\check{\sigma}_{E_2} = (1, 2) = CD$, alors $\check{\sigma}_{E_1}.\check{\sigma}_{E_2} = (1, 2, 3, 4) = ABCD$.

On remarque qu'une affectation des variables de date $\check{s}$ correspond à une affectation unique des variables de séquences $\check{\sigma}$, donnée par la fonction $\check{\sigma} = order(\check{s})$:

$$\check{\sigma}_i = |\{\check{s}_j, \check{s}_j \leq \check{s}_i \forall j\}| \quad \forall i$$

6.2.3 Variables de décision de position

L'ensemble des **variables de décision de position** est l'ensemble $\chi_{pos} = \{\zeta_1, \zeta_2 \dots \zeta_{|N|}\}$. Chaque variable à pour domaine $D_{\zeta_i} : [\![1, |N|]\!]$, et $\check{\zeta}_i = j$ signifie que la tâche en position i est j. On suppose que deux positions ne peuvent êtres associées à la même tâche. Nous utilisons alternativement les variables de séquences

1. Pour plus de commodité, nous identifions les tâches de façon équivalente par leur numéro ou par la lettre de l'alphabet correspondante, on note donc aussi cet ensemble $N = \{1, 2, 3\}$

et de position pour représenter des séquences puisqu'il existe une bijection entre leurs affectations. On note cette fonction bijective *position* :

$$position(\check{\sigma}) = \{\check{\zeta}_{\check{\sigma}_i} = i,\ \forall i \in N\}$$
$$position^{-1}(\check{\zeta}) = \{\check{\sigma}_{\check{\zeta}_i} = i,\ \forall i \in N\}$$

En utilisant les variables de position, la séquence CAB est représentée par $\check{\zeta} = (\check{\zeta}_1, \check{\zeta}_2, \check{\zeta}_3) = (C, A, B)$.

6.3 Langages de représentation

Les **langages de représentation** sont définis par leurs syntaxes, leurs sémantiques, et leur taille.
Nous inspirant des travaux de Fargier et al. [2013], soit $\mathcal{U}$ l'ensemble de tous les **objets** que nous souhaitons représenter. Dans notre cas, les objets sont principalement des **fonctions booléennes** sur les variables de décision, que l'on représente par des structures de données, ou "**formules**". Remarquons qu'une fonction booléenne $f : D_\chi \longrightarrow \mathbb{B}$ peut aussi être vue comme l'ensemble des éléments qu'elle accepte $\{\check{x}, f(\check{x}) = \top, \check{x} \in D_\chi\}$, on dira donc aussi des formules qu'elles représentent des ensembles. Soit également Σ^* l'ensemble de toutes les formules, (ou représentations) possibles. Il est pertinent ici de faire l'analogie avec la théorie des langages formels. On peut dire que les formules que nous considérons sont des **mots** du langage de représentation. Par exemple, une séquence de groupes d'opérations permutables particulière peut être vue comme un mot du langage des séquences de groupes. Nous avons vu qu'une séquence de groupes d'opérations permutables peut être associée à un ensemble des séquences qu'elle représente. De façon équivalente on pourrait définir une fonction qui détermine, pour chaque séquence, si elle est représentée par la séquence de groupe ou non : c'est la fonction qui est représentée par la formule qu'est la séquence de groupe.

Un langage L est un tuple (Φ_L, I_L, s_L) dans lequel :
— $\Phi_L \subseteq \Sigma^*$ est la *syntaxe* de L : l'ensemble des formules du langage, parmi toutes les formules de Σ^*.
— I_L : La *sémantique* de L : $I_L \subseteq \Phi_L \times \mathcal{U}$ associe à chaque formule $\varphi \in \Phi_L$ l'objet dans $\mathcal{U}$ qu'elle représente, que l'on note $[[\varphi]]_L$.
Quand cet objet est une fonction booléenne de $D_\chi \longrightarrow \mathbb{B}$, on le note aussi f_L^φ. On dit qu'une affectation $\check{x}$ est admissible par, ou satisfait φ, et inversement que φ couvre, ou accepte $\check{x}$, lorsque $f_L^\varphi(\check{x}) = \top$ (noté $\check{x} \models \varphi$, ou alternativement, $\check{x} \in [[\varphi]]$).
— $s_L : \Phi_L \longrightarrow \mathbb{N}$ donne pour chaque formule $\varphi \in \Phi_L$ sa taille $|\varphi|$.

On note $L(\chi)$ un langage de représentation de fonctions booléennes défini pour une famille de variables de décision χ.

Définition 6.3.0.1. *Nous disons qu'un* ***langage*** L *est* ***complet*** *sur l'ensemble* $A \subseteq \mathcal{U}$ *ssi* $\forall a \in A, \exists \varphi \in \Phi_L, [[\varphi]] = a$.

Définition 6.3.0.2. *On dit qu'un langage* $L(\chi)$ *représentant des fonctions booléennes sur les variables de décision* χ *est* ***complet*** *s'il est complet sur* 2^{D_χ} *i.e* $\forall D \subseteq D_\chi, \exists \varphi \in \Phi_L, \forall \check{x} \in D_\chi, f_L^\varphi(\check{x}) = \top$ *ssi* $\check{x} \in D$.

Les problèmes de recherche de solutions admissibles peuvent êtres vus comme des langages dans lesquels la syntaxe est l'ensemble des formulations correctes du problème (les instances possibles), et la sémantique interprète les instances du problème comme la fonction booléenne qui accepte les affectations qui sont des solutions admissibles de l'instance.

6.3.1 Le problème de décision à une machine

Dans ce chapitre, nous nous intéressons à un problème d'ordonnancement à une machine dans lequel un ensemble N de tâches doivent être ordonnancées. Chaque tâche $i \in N$ possède une date de disponibilité $r_i \in \mathbb{R}^+$, un temps d'exécution $p_i \in \mathbb{R}^{+*}$ et une date de livraison stricte $d_i \in \mathbb{R}^+$. Ces informations sur les tâches sont les **données temporelles** : on note PRD le tuple (P, R, D) avec $P \in \mathbb{R}^{+|N|}, R \in \mathbb{R}^{+|N|}, D \in \mathbb{R}^{+|N|}$ qui sont les vecteurs de durées, de dates de disponibilité et de livraison pour toutes les tâches. Toutes les tâches doivent être ordonnancées sans chevauchement ni préemption (donc une seule tâche peut être exécutée à la fois, et les tâches ne peuvent être interrompues une fois commencées). De plus, les tâches sont soumises à un ensemble E de contraintes de précédence. Contrairement au problème à une machine de la partie II, les contraintes de livraison sont strictes. L'objectif sera ici de trouver un ordonnancement admissible étant données ces contraintes. En d'autre termes, une instance du problème représente les ordonnancements qui sont admissibles. On peut formellement décrire le langage de ce problème $L_{1M}(\chi_{date})$:

- Φ_{1M} est l'ensemble des $\varphi = (E, PRD)$ dans lesquels PRD sont les données temporelles et $E \subset N^2$ est l'ensemble des contraintes de précédence. Ces formules sont appelées **instances du problème à une machine** ou simplement **instances**.
- I_{1M} : associe à chaque instance φ dans Φ_{1M} la fonction $f_{L_{1M}}^\varphi : D_{\chi_{date}} \to \mathbb{B}$ telle que $f_{L_{1M}}^\varphi(\check{s}) = \top$ ssi les équations suivantes sont vérifiées :

$$\begin{aligned} r_i \leq \check{s}_i \leq d_i - p_i \quad & \forall i \in N \\ \check{s}_i + p_i \leq \check{s}_j \vee \check{s}_j + p_j \leq \check{s}_i \quad & \forall (i,j) \in N^2, i \neq j \\ \check{s}_i < \check{s}_j \quad & \forall (i,j) \in E \end{aligned}$$

— $s_{L_{1M}}(\varphi) = |E| + 3|N|$.

On rappelle qu'un ordonnancement admissible $\check{s} \in D_{\chi_{date}}$ est **semi-actif** par rapport aux données temporelles PRD d'une instance φ ssi pour cette instance, aucune tâche ne peut être démarrée plus tôt sans changer l'ordre des tâches ou rendre l'ordonnancement non admissible vis à vis des contraintes de non chevauchement ou de disponibilité des tâches.

Définition 6.3.1.1. *Formellement, si* $position(order(\check{s})) = \check{\zeta}$, $\check{s}$ *est semi-actif ssi* $\check{s}_{\check{\zeta}_1} = r_{\check{\zeta}_1}$ *et* $\check{s}_{\check{\zeta}_i} = max(r_{\check{\zeta}_i}, \check{s}_{\check{\zeta}_{i-1}} + p_{\check{\zeta}_{i-1}}), \forall i \in N \setminus \{1\}$.

Pour chaque séquence et données temporelles, il existe donc un unique ordonnancement semi-actif correspondant.

Pour plus de commodité, pour décrire une instance particulière, nous utilisons une notation par accolade pour donner les valeurs de différents composants de l'instance (qui décrit explicitement les données temporelles pour chaque tâche P,R, D et l'ensemble des contraintes de précédence E). Par exemple, la notation suivante :

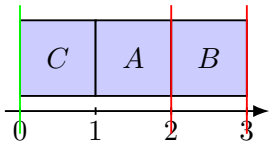
$$\varphi = \begin{cases} p_i = 1 & \forall i \in N \\ r_i = 0 & \forall i \in N \\ d_B = 3 & \\ d_i = 2 & \forall i \in N \setminus \{B\} \\ E = \emptyset & \end{cases} \tag{6.1}$$

désigne une instance $\varphi = (E, PRD) \in L_{1M}$ pour un problème à une machine avec $N = \{A, B, C\}$, $E = \emptyset$, $p_i = 1, \forall i \in N$, etc.
On remarque que pour cette instance, tout ordonnancement semi-actif tel que B est exécuté en dernier est une solution admissible telle que $f_{\varphi}^{L_{1M}}(\check{s}) = \top$. En particulier, l'ordonnancement $\check{s} = (1, 2, 0)$ est admissible (voir la figure 6.2a).

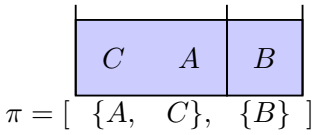
Il est connu qu'en général, trouver un ordonnancement admissible pour ce langage est NP-difficile. Donc même s'il est possible de calculer une solution à ce problème au préalable, si un aléa entraînant une modification des données survient lors de la phase réactive, il ne sera pas toujours possible de calculer rapidement une nouvelle solution (si $P \neq NP$). Nous considérons par conséquent des représentations alternatives du même ensemble de solutions.

6.3.2 Langages PRD et cohérence des formules

Nous considérons par la suite des langages de représentation de séquences, qui s'abstraient des dates précises d'ordonnancement des tâches pour se concentrer sur des **données séquentielles** comme l'ordre des tâches. L'objectif étant de considérer un espace des solutions de taille réduite. Nous allons donc nous concentrer sur des fonctions booléennes basées sur les variables de décision χ_{seq} ou χ_{pos} plutôt que χ_{date}. Cependant, nous sommes toujours intéressés par les données temporelles, en



(a) Ordonnancement admissible pour l'instance φ (eq. 6.1) de L_{1M}



(b) Séquence admissible pour une formule π (eq. 6.2) de L_{POG}

FIGURE 6.2 – Représentation d'une formule de L_{1M} avec un ordonnancement admissible, ainsi que d'une formule de L_{POG} avec une séquence admissible

particulier lorsque nous étudierons des requêtes pour lesquelles elles sont nécessaires (dans la section 6.7).

Pour cette raison, les formules $\varphi = (\pi, PRD)$ des langages que nous étudions sont composées de données séquentielles π, décrivant des contraintes des relations d'ordre entre les tâches d'un ensemble N, ainsi que des données temporelles PRD qui comportent des informations concernant les tâches de N. Nous appelons ces langages des "**langages PRD**". L'ensemble des langages PRD est noté $\mathcal{L}_{PRD}$. Les données temporelles n'interviennent pas nécessairement dans la sémantique de la représentation. Cependant, pour les langages de représentation d'ensemble de séquences que nous considérons, nous ne nous intéressons qu'aux formules pour lesquelles les données séquentielles π sont **cohérentes** avec les données temporelles PRD. C'est une condition analogue à la cohérence inverse globale [Bessiere et al., 2013].

Définition 6.3.2.1. *On dit d'une séquence $\check{\sigma}$ qu'elle est **cohérente avec les données temporelles** PRD, ssi $\exists \check{s} \in D_{\chi_{date}}, order(\check{s}) = \check{\sigma}, \check{s} \models \psi$ avec $\psi = (\emptyset, PRD) \in L_{1M}$. On note $\check{\sigma} \models PRD$.*

Définition 6.3.2.2. *On dit d'une formule $\varphi = (\pi, PRD) \in L(\chi_{seq})$ qu'elle est **cohérente**, ssi $\forall \check{\sigma} \in D_{\chi_{seq}}, f^{\varphi}(\check{\sigma}) = \top, \check{\sigma} \models PRD$. On note $\varphi \models PRD$.*

En d'autres termes, pour chaque séquence représentée, il existe un ordonnancement dont l'ordre des tâches correspond et qui satisfait les données temporelles.
On définit de façon similaire la cohérence d'une séquence représentée par les variables de décision de position et la cohérence d'une formule de $L(\chi_{pos})$.
Dans la suite, nous ne nous intéressons qu'aux fragments cohérents des langages PRD. C'est à dire, pour chacun des langages $L \in \mathcal{L}_{PRD}$ présentés dans la suite, au langage issue de la restriction de Φ_L à $\{(\varphi, PRD) \in \Phi_L, \varphi \models PRD\}$.

Du fait de la dominance des ordonnancement semi-actifs, si $\check{\sigma} \models PRD$, alors en particulier, l'ordonnancement semi-actif correspondant satisfait les données temporelles.

Dans un léger abus de notation, on omet, lorsque c'est sans ambiguïté, les données temporelles d'une formule $\varphi = (\pi, PRD)$ et on nomme alors "formule" les seules

données séquentielles π. Remarquons que toute donnée séquentielle π est cohérente avec des données temporelles :

$$PRD_\infty = \begin{cases} p_i = 1 & \forall i \in N \\ r_i = 0 & \forall i \in N \\ d_i = \infty & \forall i \in N \end{cases}$$

En effet, on peut remarquer que tout ordonnancement est admissible par une formule $(\emptyset, PRD_\infty)$ de L_{1M}, et il existe donc un ordonnancement admissible correspondant à chaque séquence.

6.3.3 Séquences de groupes d'opération permutables

Nous considérons le langage de séquence de groupes d'opérations permutables L_{POG}. Nous avons évoqué que ce langage représente des ensembles de séquences, et non des ensembles d'ordonnancements. Le langage des séquences de groupes d'opérations permutables $L_{POG}(\chi_{seq}) \in \mathcal{L}_{PRD}$ est formellement défini comme suit :

- Les formules de Φ_{POG} sont composées de deux parties : la description des séquences et les données temporelles $\varphi_{POG} = (\pi, PRD)$. π est une partition ordonnée de l'ensemble des tâches N :
  $\pi = [G_1, G_2 \dots G_k],\ k \leq |N|,\ \bigcup_{i=1\dots k} G_i = N$ et $G_i \cap G_j = \emptyset\ \forall (i,j)$.
  Pour un POG π, on note $G^\pi(i)$ le groupe qui contient la tâche i.
- I_{POG} : associe chaque formule $\varphi \in \Phi_{POG}$ avec la fonction $f^\varphi : D_{\chi_{seq}} \to \mathbb{B}$ telle que $f^\varphi(\check{\sigma}) = \top$ ssi chaque tâche dans la séquence est à une position admissible pour le groupe auquel elle appartient dans π, formellement : $f^\varphi(\check{\sigma}) = \top$ ssi $\forall (i,j) \in N^2, (G(i) \prec_\pi G(j) \implies \check{\sigma}_i < \check{\sigma}_j)$
- $s_{POG}(\varphi)$ est la taille du POG plus la taille des données temporelles soit $|\varphi| = |N| + 3|N|$.

Comme indiqué plus tôt, nous nous intéressons au fragment cohérent de L_{POG}. Puisque toutes les formules de Φ_{POG} doivent être cohérentes, une transformation comme par exemple des modifications des données temporelles, peut impliquer des changements nécessaires des données séquentielles pour prendre en compte la survenue d'aléas. Garantir la cohérence des formules permet donc de s'assurer que la représentation compilée correspond toujours à l'instance d'intérêt.

Par exemple, soit la formule de L_{POG} suivante :

$$\pi = [\{A, C\}, \{B\}] \tag{6.2}$$

On remarque que, encore une fois, $f^\pi_{L_{POG}}(\check{\sigma}) = \top$ lorsque B est en dernière position. En particulier, la séquence $\check{\sigma} = (2, 3, 1)$ (soit CAB) est admissible (voir la figure 6.2b) car toutes les tâches du premier groupe sont situées avant celles du second groupe.

On remarque dans l'exemple figure 6.2 des similarités frappantes dans les objets

représentés. D'une part, entre l'ordonnancement admissible $\check{s}$ de la figure 6.2a et la séquence admissible $\check{\sigma}$ de la figure 6.2b (en fait $order(\check{s}) = \check{\sigma}$). D'autre part entre ce qui est représenté par l'instance du problème à une machine $[[\varphi]]$ et par la séquence de groupes d'opérations permutables $[[\pi]]$ de la figure 6.2 : pour chaque ordonnancement $\check{s}$ admissible pour φ, la séquence correspondante $\check{\sigma} = order(\check{s})$ est admissible pour π. Nous voudrions comparer les deux langages L_{1M} et L_{POG}, mais le paradigme classique de la compilation de connaissance ne le permet pas puisque les objets représentés sont de nature différentes : des fonctions booléennes sur des ordonnancements, et des fonctions booléennes sur des séquences. Nous requérons l'usage des translations.

6.4 Translations entre objets

Une **translation** [Fargier et al., 2013] est une relation liant des paires d'objets de $\mathcal{U}$. Lorsque deux objets a et b sont liés par une translation T, on note aTb.

6.4.1 Translations entre ordonnancements et séquences

Définition 6.4.1.1. *Nous définissons la* ***translation*** τ ***entre ordonnancements et séquences*** *comme l'ensemble des paires* $(\check{s}, \check{\sigma})$ *avec* $\check{s} \in D_{\chi_{date}}, \check{\sigma} \in D_{\chi_{seq}}$ *qui lie tout ordonnancement* $\check{s}$ *avec la séquence* $\check{\sigma}$ *telle que* $order(\check{s}) = \check{\sigma}$.

Cette relation lie tous les ordonnancements qui partagent le même ordre des tâches et la séquence correspondante.
Par exemple, l'ordonnancement $\check{s}$ et la séquence $\check{\sigma}$ présentés dans les figures 6.2a et 6.2b sont liés à travers cette translation : $\check{s}\tau\check{\sigma}$.

6.4.2 Translations entre ensembles d'ordonnancements et ensembles de séquences

Définition 6.4.2.1. *Nous définissons ensuite une* ***translation entre ensembles d'ordonnancements et ensembles de séquences*** $\mathbb{T} \subseteq 2^{D_{\chi_{date}}} \times 2^{D_{\chi_{seq}}}$ *qui lie des ensembles d'ordonnancements et les ensembles de séquences* (X, Y) *tels que* $\forall \check{s} \in X, \exists \check{\sigma} \in Y, \check{s}\tau\check{\sigma}$ *et que inversement* $\forall \check{\sigma} \in Y, \exists \check{s} \in X, \check{s}\tau\check{\sigma}$.

Intuitivement, $\mathbb{T}$ lie l'ensemble des paires (X, Y) telles que $Y = \{order(\check{s}), \forall \check{s} \in X\}$.
Dans l'exemple, on a effectivement que les formules φ et π des eq. 6.1 et 6.2 sont telles que $[[\varphi]]_{1M}\mathbb{T}[[\pi]]_{POG}$.

Remarque : Si $\varphi = (\pi, PRD)$ est une formule pour un langage $L(\chi_{seq}) \in \mathcal{L}_{PRD}$ cohérent, et que $\psi = (E, PRD')$ est une formule de L_{1M} telle que $[[\psi]]\mathbb{T}[[\varphi]]$, alors

en particulier, si la sémantique de $L(\chi_{seq})$ n'implique pas les données temporelles (ce qui est le cas pour les langages de représentation de séquence considérés), alors $\varphi' = (\pi, PRD')$ représente le même objet que φ, et de plus φ' est cohérent (puisque pour chaque séquence représentée, ψ contient un ordonnancement correspondant admissible). Donc on à aussi $[[\psi]]\mathbb{T}[[\varphi']]$. On peut donc ramener une formule représentant une instance compilée à des données temporelles pertinentes pour l'instance représentée sous-jacente.

6.4.3 Translations entre ensembles de séquences

Nous avons vu plus tôt l'existence d'une bijection "*position*" entre affectation des variables de décision de séquence χ_{seq} et affectation des variables de décision de position χ_{pos}.

Définition 6.4.3.1. *Nous établissons donc aussi une* ***translation entre ensemble de séquences*** *représentées par les variables de séquence et par les variables de position :* $\mathcal{T} \subseteq 2^{D_{\chi_{seq}}} \times 2^{D_{\chi_{pos}}}$ *qui lie des ensembles* (X, Y) *tels que* $\forall \breve{\sigma} \in X, \exists \breve{\zeta} \in Y$, *position*$^{-1}(\breve{\zeta}) = \breve{\sigma}$ *et donc inversement* $\forall \breve{\zeta} \in Y, \exists \breve{\sigma} \in X$, *position*$(\breve{\sigma}) = \breve{\zeta}$.

On remarque que $\mathcal{T}$ est aussi une bijection. De plus, la translation $\mathbb{T} \circ \mathcal{T}$ lie les ensembles d'ordonnancements de $2^{D_{\chi_{date}}}$ avec les ensembles de séquences représentées par les variables de position de $2^{D_{\chi_{pos}}}$

6.5 Comparer des langages de représentation

En utilisant les translations, nous pouvons désormais comparer des langages représentant différents types d'objets. Les langages sont comparés selon trois aspects :

- L'expressivité : un langage peut-il représenter plus d'objets qu'un autre ?
- La compacité : Si deux langages peuvent représenter les mêmes objets, y-a-t-il des objets pour lesquels la taille de la formule nécessaire pour les représenter est exponentiellement plus grande pour un langage que pour l'autre ?
- Le support de requêtes : Est-ce qu'une opération peut être exécutée facilement dans un langage alors qu'elle est difficile dans l'autre ?

Puisque nous souhaitons pouvoir comparer des langages représentant des objets de différents types, nous utilisons les concepts d'expressivité et de compacité *modulo une translation* T, tel que définis dans Fargier et al. [2013]. On retrouve les définitions classiques d'expressivité et compacité (pour la comparaison d'objets de même type) par la translation identité "Id", qui lie chaque objet à lui même.

6.5.1 Expressivité

Définition 6.5.1.1. *Pour une translation donnée* T, *on dit que le langage* L_1 *est* ***au moins aussi expressif*** *que le langage* L_2 *modulo* T *(que l'on note* $L_1 \leq_e^T L_2$*)*

ssi :

$$\forall \varphi_2 \in \Phi_{L2},\ \exists \varphi_1 \in \Phi_{L1},\ [[\varphi_1]]_{L1} T [[\varphi_2]]_{L2}$$

C'est à dire, que pour chaque formule de L_2, il existe une formule "équivalente" de L_1, c'est à dire telle que les objets que représentent les formules sont liés par T. Par exemple, nous montrerons que $L_{POG} \geq_e^{\mathbb{T}} L_{1M}$: le langage L_{1M} est au moins aussi expressif que le langage L_{POG} modulo $\mathbb{T}$ (définition 6.4.2.1).

6.5.2 Compacité

Définition 6.5.2.1. *Pour une translation donnée T, on dit que le langage L_1 est **au moins aussi compact** que le langage L_2 modulo T (et on note $L_1 \leq_s^T L_2$) ssi :*

$$\exists P \in \mathbb{R}[X],\ \forall \varphi_2 \in \Phi_{L2},\ \exists \varphi_1 \in \Phi_{L1},\ [[\varphi_1]]_{L1} T [[\varphi_2]]_{L2} \text{ et } |\varphi_1| < P(|\varphi_2|)$$

C'est à dire, qu'il existe un polynôme de $\mathbb{R}[X]$ tel que, pour chaque formule de L_2, il existe une formule équivalente de L_1 polynomialement plus petite, telle que les objets que représentent les formules sont liés par T.
On utilise aussi les notations suivantes pour les relations d'expressivité et de compacité :

- $L_1 <^T L_2$, lorsque $L_1 \leq^T L_2$ et $L_1 \not\geq^T L_2$
- $L_1 \sim^T L_2$, lorsque $L_1 \leq^T L_2$ et $L_1 \geq^T L_2$
- $L_1 \lesseqgtr^T L_2$, lorsque $L_1 \not\leq^T L_2$ et $L_1 \not\geq^T L_2$

6.5.3 Requêtes

Informellement, les *requêtes* sont des opérations sur les objets. La complexité du calcul associé avec l'opération dépend de langage dans lequel l'objet est représenté. Si ce calcul est polynomial, on dit que le langage *supporte* la requête. On distingue deux catégories de requêtes : les **transformations**, dont la sortie est un objet représenté par le même langage que l'entrée, et les **queries** qui ne sont pas des transformations.

Formellement, une query RQ (resp. une transformation RT) est une relation liant des objets d'intérêt et des paramètres avec des sorties satisfaisantes. On a $RQ \subseteq (\Omega \times \Phi_{PRM} \times \Phi_{ANS})$ (resp. $RT = (\Omega \times \Phi_{PRM} \times \Omega)$) avec $\Omega \subseteq \mathcal{U}$ l'ensemble des objets d'intérêt (des fonctions booléennes des variables de décision dans notre cas), PRM est un langage capable de représenter tous les paramètres pertinents pour la requête, et ANS un langage capable de représenter les sorties pertinentes (dans le cas des transformations, ce langage est le langage d'entrée).

Définition 6.5.3.1. *On dit que le langage L supporte une query RQ (resp. une transformation RT) ssi le calcul correspondant peut être executé en temps polynomial quand les objets dans Ω sont représentés par le langage L ; i.e ssi il existe un*

algorithme polynomial A tel que $\forall \varphi \in \Phi_L, p \in \Phi_{PRM}, A(\varphi, p)$ *retourne* $a \in \Phi_{ANS}$ *(resp.* $\varphi' \in \Phi_L$*) tel que* $([[\varphi]]_L, p, a) \in RQ$ *(resp.* $([[\varphi]]_L, p, [[\varphi']]_L) \in RT$*).*

Par exemple, nous nous rappelions plus tôt que L_{1M} ne supporte pas la requête consistant à déterminer s'il existe une solution admissible, tandis que L_{POG} la supporte, puisque toute séquence représentée est admissible. Les autres requêtes d'intérêt pour nous seront présentées dans la section 6.7.

6.5.4 Comparer des langages d'expressivité différentes

Nous présentons dans la prochaine section d'autres langages candidats pour la compilation d'instances du problème à une machine.

Quand ces langages cible sont moins expressifs que le langage source (L_{1M}), ils ne conviennent pas pour la compilation. En effet, nous voulons au moins être capable de compiler toutes les instances du langage source dans le langage cible. Nous verrons par exemple que L_{POG} n'est pas suffisamment expressif.
A l'inverse, lorsqu'un langage est *plus* expressif que le langage source, il convient pour la compilation, toutefois, il n'est pas possible a priori de comparer la compacité des langages (dont la définition requiert des langages d'expressivité équivalente). Par conséquent, pour chacun de ces langages "trop" expressifs L, nous définissons aussi le langage L^* : le fragment de L restreint aux formules qui correspondent aux instances du problème à une machine, modulo une translation pertinente T. Formellement,

$$\Phi_{L^*} = \{\varphi \in \Phi_L | \exists \varphi' \in \Phi_{L_{1M}}, [[\varphi']]_{L_{1M}} T [[\varphi]]_L\}$$

On note $\mathcal{L}^*$ l'ensemble des langages restreints de cette manière.

6.6 Autres langages de représentation de séquences

Dans cette section, nous présentons les langages candidats que nous étudions pour la compilation des instances de L_{1M}, en sus du langage L_{POG}, correspondant à des structures que nous avons évoquées dans les chapitres précédents (voir section 2.3). Ces langages seront comparés dans le chapitre 7. On rappelle que pour tous les langages de $\mathcal{L}_{PRD}$ suivants, nous ne nous intéresserons par la suite qu'à leur fragment cohérent.

6.6.1 Langage de séquences

Tout d'abord, considérons le langage de séquences, basé sur les variables de décision de séquence : $L_{SEQ}(\chi_{seq}) \in \mathcal{L}_{PRD}$:

— Une formule de Φ_{SEQ}, contient une séquence et les données temporelles : $\varphi = (\pi, PRD)$. Avec $\pi \in D_{\chi_{seq}}$.

— I_{SEQ} : Chaque séquence φ dans Φ_{SEQ} est associée à la fonction $f^{\varphi}_{L_{SEQ}}$: $D_{\chi_{seq}} \to \mathbb{B}$ telle que $f^{\varphi}_{L_{SEQ}}(\check{\sigma}) = \top$ ssi $\check{\sigma} = \pi$.
— $s_{SEQ}(\varphi) = |N| + 3|N|$.

6.6.2 Langage des ordres partiels

Considérons le langage des ordres partiels $L_{PO}(\chi_{seq}) \in \mathcal{L}_{PRD}$:
— $\varphi = (\pi, PRD) \in \Phi_{PO}$ avec π un ensemble de contraintes de précédence $e = (i, j) \in N^2$, et PRD les données temporelles.
— I_{PO} : associe à tout ensemble de contraintes de précédence $\varphi \in \Phi_{PO}$ la fonction $f^{\varphi}_{PO} : D_{\chi_{seq}} \to \mathbb{B}$ telle que $f^{\varphi}_{PO}(\check{\sigma}) = \top$ ssi $\forall (i, j) \in \pi, \check{\sigma}_i < \check{\sigma}_j$.
— $s_{PO}(\varphi) = 2|\pi| + 3|N|$.

6.6.3 Ordres partiels "And/Or"

Considérons le langage des ordres partiels And/Or $L_{AOPO}(\chi_{seq}) \in \mathcal{L}_{PRD}$:
— Les formules de Φ_{AOPO} sont les ensembles de contraintes de précédence généralisées. $\varphi = (\pi, PRD)$ avec π un ensemble de conditions d'attente $w = (X, i) \in 2^N \times N$ signifiant qu'au moins une des tâches de l'ensemble de tâches $X \subseteq N$ doit être ordonnancée avant que la tâche i le soit. PRD sont les données temporelles .
— I_{AOPO} : Associe à chaque ensemble de conditions d'attente $\varphi \in \Phi_{AOPO}$ la fonction $f^{\varphi}_{AOPO} : D_{\chi_{seq}} \to \mathbb{B}$ telle que $f^{\varphi}_{AOPO}(\check{\sigma}) = \top$ ssi $\forall (X, i) \in \pi, \exists j \in X, (\check{\sigma}_j < \check{\sigma}_i)$
— $s_{AOPO}(\varphi) = \sum_{(X,i)\in\pi} (|X| + 1) \ + 3|N|$.

6.6.4 Langage des arbres PQR

On rappelle qu'un arbre PQR est un tuple (V, A) où V est un ensemble de nœuds et A un ensemble d'arcs formant un arbre (voir chapitre 2). Les nœuds sont de quatre types distincts :
— Les nœuds permutables de $\mathbf{P} \subseteq V$: l'ordre de leurs enfants peut être choisi arbitrairement.
— Les nœuds réversibles de $\mathbf{Q} \subseteq V$: leurs enfants sont ordonnés, et cet ordre peut être inversé.
— Les nœuds ordonnés de $\mathbf{R} \subseteq V$: leurs enfants ont un ordre fixe.
— Les feuilles de $\mathbf{F} \subseteq V$: chacune est associée à une tâche dans N. L'ordre des feuilles impliqué par les choix d'ordres des enfants des autres nœuds de l'arbre définit la **frontière** de l'arbre PQR, dans laquelle on lit une séquence.

Formellement, le langage des arbres PQR $L_{PQR}(\chi_{seq}) \in \mathcal{L}_{PRD}$ est défini comme suit :
— Les formules de Φ_{PQR} sont $\varphi = (\pi, PRD)$ avec π un arbre PQR et PRD les données temporelles.

— I_{PQR} : associe à chaque arbre PQR $\varphi \in \Phi_{PQR}$ la fonction $f^{\varphi}_{PQR} : D_{\chi_{seq}} \to \mathbb{B}$. Formellement, pour tout nœud n, soit $Jobs(n)$ l'ensemble des tâches aux feuilles du sous arbre dont la racine est n. Soit $Child(n)$ l'ensemble ordonné des nœuds enfants de n. Soit $perm(x)$ l'ensemble des permutations de l'ensemble x. Pour chaque permutation $y \in perm(x)$ on note l'ordre définit par y par $<_y$. Et soit $reverse(y)$ la permutation inverse d'une permutation y. On définit récursivement la fonction associée au nœud n d'un sous-arbre PQR par :

$$f^n_{PQR}(\check{\sigma}) = g^n(\check{\sigma}) \wedge (\bigwedge_{m \in Child(n)} f^m_{PQR}(\check{\sigma}_{|Jobs(m)}))$$

avec :

$$g^n(\check{\sigma}) = \begin{cases} \top \quad \forall n \in F \\ \exists y \in perm(Child(n)), \\ \quad \forall m_1 <_y m_2 \in Child(n)^2, \forall i,j \in Jobs(m_1), Jobs(m_2), \check{\sigma}_i < \check{\sigma}_j \quad \forall n \in P \\ \exists y \in \{Child(n), reverse(Child(n))\}, \\ \quad \forall m_1 <_y m_2 \in Child(n)^2, \forall i,j \in Jobs(m_1), Jobs(m_2), \check{\sigma}_i < \check{\sigma}_j \quad \forall n \in Q \\ y = Child(n), \\ \quad \forall m_1 <_y m_2 \in Child(n)^2, \forall i,j \in Jobs(m_1), Jobs(m_2), \check{\sigma}_i < \check{\sigma}_j \quad \forall n \in R \end{cases}$$

Intuitivement, les séquences acceptées sont celles qu'il est possible d'obtenir en réorganisant les enfants des nœuds de l'arbre en fonction de leur type, pour obtenir différents ordres des feuilles sur la frontière.

— $s_{PQR}(\varphi)$ est donné par le nombre de nœud et d'arcs de φ (plus les données temporelles) : $|V| + |A| + 3|N|$.

6.6.5 Langage des diagrammes de décision multivalués

Rappelons que pour notre usage, un diagramme de décision multivalué (MDD) est une paire (V, A), dont les nœuds V sont partitionnés en $|N|+1$ couches, avec la première et la dernière couche comprenant un unique nœud (resp. le nœud racine R et le nœud terminal T). Chaque arc a étiqueté l_a de la couche k à la couche $k+1$ représente l'affectation de la variable $\check{\zeta}_k$ ($\check{\zeta}_k = i$ ssi la tâche i est en position k) à la valeur l_a. Par conséquent, un chemin du nœud racine au nœud terminal représente une séquence (voir chapitres 2 et 5).

$L_{MDD}(\chi_{pos}) \in \mathcal{L}_{PRD}$ est basé sur les variables de décision de position dans l'ordre naturel, et défini comme suit :

— Les formules de Φ_{MDD} : $\varphi = (\pi, PRD)$ sont telles que π est un MDD, et PRD sont les informations temporelles.

— I_{MDD} : associe à tout MDD $\varphi \in \Phi_{MDD}$ la fonction $f^{\varphi}_{MDD} : D_{\chi_{pos}} \to \mathbb{B}$ telle que $f^{\varphi}_{MDD}(\check{\zeta}) = g^R(\check{\zeta})$ avec g la fonction récursive :

$$\begin{cases} g^T(\check{\zeta}) = \top \\ g^u(\check{\zeta}_X) = \exists v \in V, \exists a = (u,v) \in A, l_a = \check{\zeta}_1 \wedge g^v(\check{\zeta}_{|X\setminus\{1\}}) \quad \forall u \in V \setminus \{T\} \end{cases}$$

— $s_{MDD}(\varphi) = |V| + |A| + 3|N|$.

6.6.6 Langages-ensemble

Nous avons déjà évoqué et démontrerons ultérieurement dans le chapitre 7 que certains des langages présentés ne sont pas aussi expressifs que le langage du problème à une machine (spécifiquement, les langages L_{SEQ},L_{POG},L_{PO},L_{AOPO}, et L_{PQR}).
Pour tous ces langages, nous proposons de les étendre de façon à les rendre suffisamment expressifs tout en conservant certaines de leur propriétés. De façon similaire à la fermeture disjonctive de Fargier and Marquis [2014], nous définissons les "langages-ensemble", qui représentent l'union de formules d'un langage originel. Pour un langage $L(\chi) \in \mathcal{L}_{PRD}$, le langage-ensemble associé, noté $SL(\chi) \in \mathcal{L}_{PRD}$ est défini comme suit :

— Les formules de Φ_{SL} sont composées d'un ensembles de formules du langage originel L et des données temporelles. $\varphi = (\Pi, PRD) \in \Phi_{SL}$, $\Pi = \{\pi_1, \ldots, \pi_k\}$ avec $(\pi_i, PRD) \in \Phi_L, \forall i$.
— I_{SL} : Une formule-ensemble représente la disjonction des fonctions associées aux sous-formules du langage originel (i.e. l'union des ensembles représentés) : $f^{\varphi}_{SL}(\check{x}) = \bigvee_{\pi_i \in \Pi} f^{\pi_i}_{L}(\check{x})$.
— $s_{SL}(\varphi) = \sum_{\pi_i \in \Pi} |\pi_i| (+3|N|)$.

On remarque que, si L est capable de représenter n'importe quel unique élément de D_χ, alors $SL(\chi)$ est complet, et par conséquent suffisamment expressif (voir th 7.1.1.4). Par exemple, nous avons évoqué qu'une séquence de groupes ne pouvait pas représenter les séquences ABC et CBA sans représenter toutes les séquences de trois tâches (voir théorème 7.1.4.1), mais on peut voir que l'ensemble de séquences de groupes $\pi = \{[\{A\}, \{B\}, \{C\}], [\{C\}, \{B\}, \{A\}]\}$ représente bien ces deux séquences.

6.7 Requêtes d'intérêt

Nous voulons fournir un ensemble de solutions compilées à un décideur en tant qu'outil pour superviser le processus d'ordonnancement. Le langage cible devrait donc supporter plusieurs requêtes utiles. Les sections suivantes présentent les requêtes auxquelles nous nous intéressons, mais il faut noter que ces requêtes ne constituent qu'une première sélection que nous avons choisie pour être pertinente pour l'ordonnancement réactif, et de nombreuses autres requêtes peuvent être pertinentes.

6.7.1 Suivi d'exécution

Les requêtes suivantes sont nécessaires pour suivre l'exécution de l'ordonnancement :

CO **: Existence d'une solution** Étant donné une formule φ qui représente une fonction booléenne sur les variables de décision χ, le décideur souhaite savoir si φ contient une solution.

Définition 6.7.1.1. *Formellement, L supporte la query* ***CO*** *(consistency) ssi il existe un algorithme polynomial A tel que*
$\forall \varphi \in \Phi_L$,$A(\varphi)$ *retourne* $\top$ *ssi* $\exists \check{x} \in D_\chi, f_L^\varphi(\check{x}) = \top$.

MX **: Extraire une solution** Étant donné une formule φ, le décideur souhaite extraire une solution admissible.

Définition 6.7.1.2. *Formellement, L supporte la query* ***MX****(model extration) ssi il existe un algorithme polynomial A tel que*
$\forall \varphi \in \Phi_L$,$A(\varphi)$ *retourne* $\check{x} \in D_\chi$ *tel que* $f_L^\varphi(\check{x}) = \top$.

CD_0 **: ordonnancement d'une tâche** A un temps t où la machine est libre, le décideur souhaite ordonnancer une tâche i au plus tôt (supposons que c'est au temps t_0, donné par $t_0 = max(r_i, t)$). Cela correspond à une modification des données temporelles de l'instance à PRD' dans lesquelles, $r'_i = t_0$, $d'_i = t_0 + p_i$ et $r'_j = max(r_j, t_0 + p_i), \forall j \neq i$. La formule φ doit être modifiée pour rester cohérente avec PRD', puisque seuls les ordonnancements dans lesquels la tâche i est ordonnancée en premier peuvent satisfaire les données temporelles.

Définition 6.7.1.3. *Formellement, L supporte la transformation* CD_0*(conditioning) ssi il existe un algorithme polynomial A tel que*
$\forall \varphi \in \Phi_L, i \in N$,
$A(\varphi, i)$ *retourne* $\varphi' = (\pi', PRD') \in L$ *(si un tel* φ' *existe)) avec* $[[\varphi']] = \{x, x \models PRD', \forall x \in [[\varphi]]\}$, *et* PRD' *les données temporelles modifiées de* φ *(telles que* $r'_i = t_0$, $d'_i = t_0 + p_i$ *et* $r'_j = max(r_j, t_0 + p_i), \forall j \neq i$*).*

6.7.2 Prise en compte des aléas

Les requêtes suivantes peuvent être utilisées pour prendre en compte les aléas :

$R\uparrow$ **: Délai de disponibilité** De façon imprévue, la date de disponibilité d'une tâche i est retardée à $u \geq r_i$. Le décideur souhaite modifier la formule courante φ_L pour prendre en compte le changement des données temporelles de PRD à PRD'. Soit PRM un langage représentant des affectations de valeur à des dates de disponibilité (par exemple, $[r_i = u]$ indique que la date de disponibilité de la tâche i est affectée la valeur u) .

Définition 6.7.2.1. *Formellement, L supporte la transformation $R\uparrow$ ssi il existe un algorithme polynomial A tel que*
$\forall\varphi \in \Phi_L, \forall i, \forall u > r_i, p = [r_i = u] \in \Phi_{PRM}$, $A(\varphi, p)$ *retourne* $\varphi' = (\pi', PRD') \in L$ *avec avec* $[[\varphi']] = \{x, x \models PRD', \forall x \in [[\varphi]]\}$, *et* PRD' *les données temporelles modifiées de* φ *(telles que* $r'_i = u$, *et* $r'_j = r_j, \forall j \neq i$*).*

$E\uparrow$: Ajout de contraintes de précédence Suite à un changement imprévu, le décideur souhaite ordonnancer une tâche i avant une autre tâche j. Il faut donc modifier la formule courante pour prendre en compte le changement.
Soit PRM un langage représentant des contraintes de précédence (par exemple, (i, j) pour signifier que i doit être ordonnancé avant j).

Définition 6.7.2.2. *Formellement, L supporte la transformation $E\uparrow$ ssi il existe un algorithme polynomial A tel que*
$\forall\varphi_L \in \Phi_L, \forall p = (i, j) \in \Phi_{PRM}$,
$A(\varphi, p)$ *retourne* $\varphi' = (\pi', PRD) \in L$ *, tel que* $[[\varphi']] = \{\check{x}, \check{x}_i < \check{x}_j, \forall\check{x} \in [[\varphi]]\}$.

6.7.3 Autres requêtes

Les requêtes suivantes peuvent aussi être intéressantes pour un décideur :

$BC/WC(F)$: Meilleure/Pire solution représentée Le décideur souhaite connaître le meilleur (resp. le pire) score obtenu par la mise en oeuvre d'une solution semi-active représentée par la formule courante, étant donné une fonction objectif F : $D_{\chi_{date}} \longrightarrow \mathbb{R}$.

Définition 6.7.3.1. *Formellement, L supporte la query* ***BC(F) (resp. WC(F))*** *ssi il existe un algorithme polynomial A tel que*
$\forall\varphi = (\pi, PRD) \in \Phi_L$,
$A(\varphi)$ *retourne* $\min\limits_{\check{s}\in S} F(\check{s})$ *(resp. max) où S est l'ensemble des ordonnancements* $\check{s}$ *semi-actifs pour PRD tel que* $ST[[\varphi]]$ *pour une translation adéquate T (*$\mathbb{T}$ *si* φ *est une formule d'un langage* $L(\chi_{seq})$ *ou* $\mathbb{T}\circ\mathcal{T}$ *si* φ *est une formule d'un langage* $L(\chi_{pos})$*)*

Par exemple :

1. BC/WC($\sum C_{MAX}$) où $\sum C_i(\check{s}) = \max\limits_{i\in N}\check{s}_i + p_i$
2. BC/WC($\sum w_iC_i$) où $\sum w_iC_i(\check{s}) = \sum\limits_{i\in N} w_i.(\check{s}_i + p_i)$ pour un ensemble de poids W.
3. BC/WC(L_{MAX}) où $L_{MAX}(\check{s}) = \max\limits_{i\in N}\check{s}_i + p_i - \delta_i$ avec δ une date de livraison *souples.*

6.8 Discussion

En utilisant la translation $\mathbb{T}$ nous pouvons comparer formellement le langage "naturel" de description du problème à une machine basé sur les dates de début des tâches, et des langages basés sur les séquences, la composante clé des informations d'ordonnancement, qui peuvent possiblement êtres représentés de façon plus compacte. Les requêtes présentées dans la section 6.7 ne sont pas exhaustives, mais constituent un exemple d'application possible. Concernant les requêtes de prise en compte d'aléas, on remarquera que nous ne considérons que les changements qui restreignent l'ensemble des solutions admissibles. La raison est que les formules que nous considérons sont cohérentes, donc toutes les séquences représentées satisfont les données temporelles, mais il peut exister d'autres séquences, non-représentées, qui les satisfont aussi. Prenons par exemple une formule de $L_{PO} : \varphi = (\pi, PRD)$ Si une contrainte est relâchée, par exemple qu'une date de disponibilité est avancée $(R\downarrow)$, il faudrait distinguer dans π les contraintes de précédence qui sont dues aux données temporelles de celles qui ont été ajouté par des requêtes $E\uparrow$ par exemple... On peut aussi considérer des aléas ciblant les autres paramètres. On peut remarquer que la diminution d'une date de livraison est symétrique avec l'augmentation d'une date de disponibilité (requête $R\uparrow$), mais la requête d'augmentation de la durée d'exécution par exemple pourrait être étudiée.
Dans le chapitre suivant, nous considérons les définitions et les requêtes présentées pour comparer les langages entre eux.

Chapitre 7

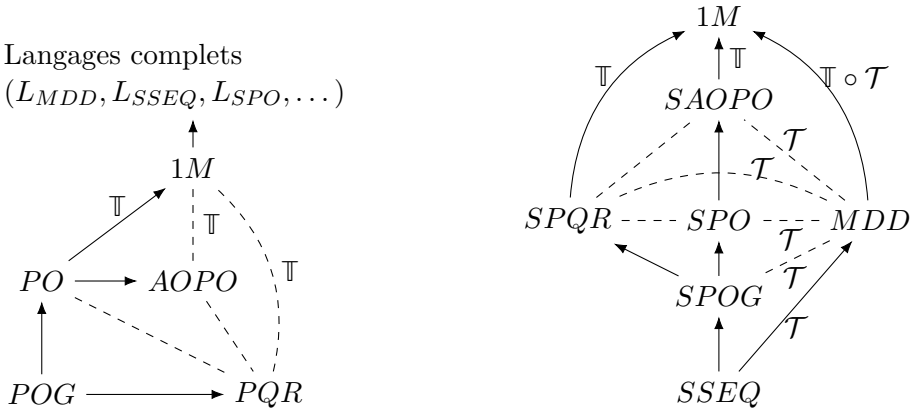
Comparaison des langages de représentation

Sommaire

Nous avons décrit dans le chapitre précédent les langages que nous comparons, ainsi que le façon dont nous les comparons. Nous avons vu que la comparaison d'expressivité et de compacité nécessite parfois l'intermédiaire d'une translation. Enfin nous avons proposé quelques requêtes d'intérêt dont nous étudions la complexité.

Les résultats d'expressivité des langages L_{1M}, L_{SEQ}, L_{PO}, L_{POG}, L_{AOPO}, L_{PQR}, L_{MDD} et des langages-ensemble (L_{SSEQ}, L_{SPO}, L_{SPOG}, L_{SAOPO}, L_{SPQR}) sont résumés dans la figure 7.1a, tandis que les résultats de compacité des langages d'expressivité équivalente (L_{1M}, L^*_{SSEQ}, L^*_{SPO}, L^*_{SPOG}, L^*_{SAOPO}, L^*_{MDD}, L^*_{SPQR}) sont résumés dans la figure 7.1b. Un arc de L_1 à L_2 signifie $L_1 > L_2$ (L_2 est strictement

plus expressif, ou strictement plus compact que L_1, selon la relation considéré). Les arrêtes pointillées relient deux langages incomparables $L_1 \lesseqgtr L_2$. Les arcs correspondants aux résultats modulo une translation sont étiquetés en conséquence. Les résultats de complexité des requêtes sont résumés dans la table 7.1. Les requêtes supportées par les langages sont indiquées par ✓, les symboles • et ∘ dénotent respectivement le non-support des requêtes [1] et le non-support sous réserve $P \neq NP$, et un point d'interrogation dénote un résultat inconnu. Les colonnes des requêtes BC et WC indiquent les fonctions pour lesquelles la requête est supportée (voir les détails dans la section 7.4).



(a) Carte d'expressivité (b) Carte de compacité (L^*)

FIGURE 7.1 – Récapitulatif des résultats d'expressivité et de compacité

	CO	MX	CD_0	$R\uparrow$	$E\uparrow$	BC	WC
L_{1M}	∘	∘	✓	✓	✓	∘	∘
L_{SSEQ}	✓	✓	✓	✓	✓	✓	✓
L_{SPOG}	✓	✓	✓	•	•	C_{MAX}	C_{MAX}, L_{MAX}
L_{SPO}	✓	✓	✓	•	✓	C_{MAX}	C_{MAX}, L_{MAX}
L_{SAOPO}	✓	✓	✓	?	✓	?	?
L_{SPQR}	✓	✓	?	?	?	?	?
L_{MDD}	✓	✓	✓	?	✓	C_{MAX}	C_{MAX}, L_{MAX}

TABLE 7.1 – Récapitulatif de complexité des requêtes
✓ : Supportée ; • : Pas supportée ;
∘ :Pas supportée, sauf si P=NP
? : Inconnu/Conjecture

Ces résultats théoriques peuvent guider un utilisateur vers le choix d'un langage cible pour la compilation de problème. Dans ce chapitre, les résultats contenus dans la carte de compilation et la table de complexité des requêtes sont démontrés. Nous établissons dans un premier temps un ensemble de résultats intermédiaires utili-

1. Ce qui signifie usuellement que la taille de l'output explose.

sés dans la suite, puis nous penchons sur les résultats d'expressivité comparée des langages, les résultats de compacité des langages, et les résultats de complexité des requêtes d'intérêts des langages. Enfin, nous concluons le chapitre par une discussion autour des résultats établis.

7.1 Résultats intermédiaires

Dans cette section, nous établissons quelques résultats intermédiaires qui seront utilisés dans les sections suivantes. Certains résultat plus généraux concernent tous les langages, ou tous les langages-ensemble, d'autres seulement les langages PRD de représentation d'ensembles de séquences cohérents, ou les langages restreints L^*, d'autres résultats encore sont des propriétés spécifiques à un langage particulier.
Dans ce qui suit, lorsque l'on parle de langage de représentation d'ensembles de séquence, on utilise χ_{seq} pour l'ensemble des variables, $\check{\sigma}$ pour une séquence, et $\mathbb{T}$ pour une translation entre ensembles d'ordonnancements et ensembles de séquences. Des preuves similaires peuvent être établies pour les autres langages de représentation d'ensembles de séquences (en l'occurrence cela s'applique à L_{MDD}) en utilisant à la place les variables χ_{pos}, les séquences $\check{\zeta}$, et la translation $\mathbb{T} \circ \mathcal{T}$ (entre ensembles d'ordonnancements et ensembles de séquences représentés par des variables de décision de position).

7.1.1 Propriétés générales des langages

Théorème 7.1.1.1. *Soit $T \subseteq A \times B$ une translation telle que $\forall a \in A, \exists b \in B, aTb \in T$ (i.e. une relation totale). Soit L_1 un langage complet sur B. Alors L_1 est au moins aussi expressif modulo T que tout langage L_2 représentant exclusivement des objets de A :*
$\forall L_2, (\forall \varphi \in \Phi_{L_2}, [[\varphi]] \in A) \implies L_1 \leq_e^T L_2$.

Démonstration. Si $\forall \varphi \in \Phi_{L_2}, [[\varphi]] \in A$, et $\forall a \in A, \exists b \in B, aTb \in T$, et puisque L_1 est complet sur B, $\forall b \in B, \exists \varphi' \in \Phi_{L_1}, [[\varphi']] = b$.
On a bien $\forall \varphi \in \Phi_{L_2}, \exists \varphi' \in \Phi_{L_1}, [[\varphi]] T [[\varphi']]$. □

Remarquons que la translation identité Id vérifie les conditions nécessaires à l'application de ce théorème. Nos translations τ, $\mathcal{T}$ et $\mathbb{T}$ vérifient aussi les conditions nécessaires : chaque ordonnancement et chaque séquence ont un équivalent par τ, chaque ensemble de séquence représentée par les variables χ_{seq} à un ensemble équivalent de séquence représentées par les variables χ_{pos}, et chaque ensemble d'ordonnancements et de séquences ont un équivalent par $\mathbb{T}$. Par conséquent, tout langage de représentation d'ensemble de séquences complet sera au moins aussi expressif que tout autre langage modulo nos translations.

Théorème 7.1.1.2. *Soient L_1 et L_2 deux langages et T une translation entre L_1 et L_2. Si L_1 est au moins aussi compact que L_2, alors le langage-ensemble SL_1 est au moins aussi compact que le langage ensemble SL_2 (modulo T) :*
$(L_1 \leq_s^T L_2) \implies (SL_1 \leq_s^T SL_2)$

Démonstration. Soit φ une formule de SL_2. On sait $\exists P \in \mathbb{R}[X],\ \forall \varphi_i \in \Phi_{L_2},\ \exists \pi_i \in \Phi_{L_1},\ [[\pi_i]]T[[\varphi_i]], |\pi_i| < P(|\varphi_i|)$.
Donc il existe une formule $\pi = \{\pi_1, \ldots, \pi_k\}$ de SL_1 et $[[\pi]]T[[\varphi]]$.
On a aussi $|\pi| = \sum_{i=1\ldots k} |\pi_i| \leq \sum_{i=1\ldots k} P(|\varphi_i|)$.
En toute généralité $P(x) = a_n.x^n + \cdots + a_1.x + a_0$. Donc puisque $|\varphi_i| \geq 0$, $\sum_{i=1\ldots k} P(|\varphi_i|) \leq |a_n|.(\sum_{i=1\ldots k} |\varphi_i|)^n + \cdots + |a_1|.(\sum_{i=1\ldots k} |\varphi_i|) + k.|a_0|$
En identifiant le polynôme $P' \in \mathbb{R}[X]$, on a bien $\forall \varphi \in \Phi_{SL_2},\ \exists \pi \in \Phi_{SL_1},\ [[\pi]]T[[\varphi]], |\pi| \leq P'(|\varphi|)$. □

Théorème 7.1.1.3. *de Fargier et al. [2013]*
Soient trois langages L_1, L_2, et L_3, ainsi que T une translation entre L_1 et L_2, et T' une translation entre L_2 et L_3 :
$(L_1 \geq^T L_2) \wedge (L_2 \geq^{T'} L_3) \implies (L_1 \geq^{T' \circ T} L_3)$

Ce théorème permet d'établir la transitivité des relations de comparaison des cartes de compilation présentées lorsque l'une des translation est $\mathbb{T}$ et l'autre est Id par exemple.

Par contraposition, nous obtenons aussi les deux corollaires suivants :

Corollaire 7.1.1.1. $(L_1 \not\geq^{T \circ T'} L_3) \wedge (L_1 \geq^T L_2) \implies (L_2 \not\geq^{T'} L_3)$

Corollaire 7.1.1.2. $(L_1 \not\geq^{T \circ T'} L_3) \wedge (L_2 \geq^{T'} L_3) \implies (L_1 \not\geq^T L_2)$

Théorème 7.1.1.4. *Soit 2^A l'ensemble d'ensembles d'éléments de A. Soit L un langage de représentation d'ensembles d'éléments de A. Si L peut représenter individuellement chaque ensemble de cardinalité 1 de l'ensemble 2^A, alors le langage-ensemble correspondant SL est complet sur 2^A :*
$\forall L, (\forall \{a\} \in 2^A, \exists \varphi \in \Phi_L, [[\varphi]] = \{a\}) \implies (\forall B \in 2^A, \exists \varphi' \in \Phi_{SL}, [[\varphi']] = B)$

Démonstration. Si $\forall a \in A, \exists \varphi_a \in L, [[\varphi_a]] = \{a\}$ alors $\forall B \subseteq A, \exists \varphi' \in SL, [[\varphi']] = B$. En particulier, l'ensemble $\{\varphi_b, \forall b \in B\}$ est un tel φ'. □

Ainsi, puisque nos langages de représentation d'ensembles de séquences sont capables de représenter n'importe quelle unique séquence, les langages-ensembles associés sont complets.

7.1.2 Propriété des langages de de représentation de séquences

Le théorème suivant nécessite la définition préalable des notions de fermeture pour les opérateurs de concaténation et de projection.

Définition 7.1.2.1. *Soit un langage de représentation d'ensembles de séquences L. Soient E_1 et E_2 des ensembles de variables de décision. On dit que le langage L est* ***fermé pour la concaténation*** *"." ssi*

$$\forall E_1, E_2 (E_1 \cap E_2 = \emptyset), \forall \varphi_1 \in L(E_1), \varphi_2 \in L(E_2),\ \exists \varphi_3 \in L(E_1 \cup E_2)$$

avec

$$[[\varphi_3]] = \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1 \in [[\varphi_1]], \forall \check{\sigma}_2 \in [[\varphi_2]]\}$$

On note $\varphi_3 = \varphi_1.\varphi_2$.

Définition 7.1.2.2. *De façon similaire, soit un langage L de représentation d'ensembles de séquences et E un ensemble de variables de décision. On dit que L est* ***fermé pour la projection*** *ssi*

$$\forall F \subseteq E\ \forall \varphi \in L(E),\ \exists \varphi' \in L(F),\ [[\varphi']] = \{\check{\sigma}_{|F}, \forall \check{\sigma} \in [[\varphi]]\}$$

On note $\varphi' = \varphi_{|F}$.

Étant donné ces définitions, on a le théorème suivant :

Théorème 7.1.2.1. *Soit L_1 un langage de représentation d'ensembles de séquences fermé pour la concaténation, avec $|\varphi_1.\varphi_2| \sim O(|\varphi_1| + |\varphi_2|)$ i.e. dont la taille des formules augmente linéairement avec la concaténation. Soit L_2 un langage de représentation d'ensemble de séquences fermé pour la concaténation et pour la projection tel que SL_2 soit complet.*
S'il existe des formules dans le langage L_1 qui n'ont pas d'équivalent dans le langage L_2, alors le langage-ensemble SL_2 n'est pas plus compact que SL_1 :
$L_1 \not\geq_e L_2 \implies SL_1 \not\geq_s SL_2$.

Démonstration. Puisque $L_1 \not\geq_e L_2$, il existe un ensemble de variables de décision E tel que $L_1(E) \not\geq_e L_2(E)$. Puisque les variables de décision sont muettes, l'exemple peut être reproduit pour une infinité d'ensembles de variables E_k. Pour chacun de ces ensembles de variables distinctes, on a $L_1(E_k) \not\geq_e L_2(E_k) : \exists \varphi_k \in L_1(E_k), \forall \pi_k \in L_2(E_k), [[\varphi_k]] \neq [[\pi_k]]$.
Considérons la formule $\varphi^i = \varphi_1.\varphi_2.\ldots.\varphi_i$. Posons $E^i = \bigcup_{k=1\ldots i} E_k$. On a $\varphi^i \in L_1(E^i)$ car L_1 est fermé pour la concaténation. Donc, $\{\varphi^i\} \in SL_1(E^i)$ par définition des langages-ensemble.
Soit Π_i^* une formule de $SL_2(E^i)$ telle que $|\Pi_i^*| = \min_{\Pi \in SL_2(E^i)} \{|\Pi|,\ [[\Pi]] = [[\varphi^i]]\}$ c'est à dire que Π_i^* est un ensemble de formules représentant les mêmes séquences que $[[\varphi^i]]$ de cardinalité minimale (un tel Π_i^* existe puisque SL_2 est complet).
Cette formule Π_i^* est composée de formules de L_2, i.e. $\forall \pi \in \Pi_i^*, \pi \in L_2(E^i)$. De plus, pour chacune de ces sous-formules, π contenues dans Π_i^*, soit la formule alternative $\pi' = \pi_{|E_1}.\pi_{|E_2}.\ldots.\pi_{|E_i}$. On sait $\pi' \in L_2(E^i)$ car L_2 est fermé pour la projection et la concaténation. On a $[[\pi]] \subseteq [[\pi']]$ et de plus on remarque $[[\pi_{|E_k}]] \subset [[\varphi_k]],\ \forall k$, car si $[[\pi_{|E_k}]] \not\subseteq [[\varphi_k]]$, alors $[[\Pi_i^*]] \neq [[\varphi^i]]$, et on sait qu'il n'existe pas de π_k tel que $[[\pi_k]] = [[\varphi_k]]$. Donc on a aussi $[[\pi']] \subset [[\Pi_i^*]]$.
Il est donc possible de remplacer π par π' dans Π_i^* sans modifier l'ensemble représenté, ni augmenter la cardinalité de Π_i^*. C'est à dire qu'il existe une formule de $SL_2(E^i)$ de cardinalité minimale composé uniquement de formules issues de la concaténation de formules des langages $L_2(E_k)$. On conclut donc qu'il est dominant (on peut s'y restreindre vis à vis de la minimisation de la cardinalité) pour Π_i^* de

n'être composé que de formules de $L_2(E^i)$ issues de la concaténation de formules π_k de $L_2(E_k)$ telles que $[[\pi_k]] \subset [[\varphi_k]]$. Nommons $\Phi^i_{L_2} \subset \Phi_{L_2(E^i)}$ l'ensemble de telles formules.

Nous reprenons ici la logique du théorème 1.a de Ahlswede [2006] [2] :

Soit donc Π^*_{i+1} composé de formules de $\Phi^{i+1}_{L_2}$, qui représente toutes les séquences de $[[\varphi^{i+1}]]$ et de cardinalité minimale pour une telle formule de $SL_2(E^{i+1})$.

Notons $x[[\varphi^{i+1}]]$ le sous ensemble de $[[\varphi^{i+1}]]$ définit par $\{\check{\sigma} \in [[\varphi^{i+1}]], \check{\sigma}_{|E_1} = x\}$. Enfin, soit une distribution de probabilité $q^* \in \mathcal{Q}$ sur $\Phi^1_{L_2}$ telle que

$$q^*_\pi = \frac{|\{\pi_{i+1}, \pi_{i+1} \in \Pi^*_{i+1}, [[\pi_{i+1}]]_{|E_1} = [[\pi]]\}|}{|\Pi^*_{i+1}|} \forall \pi \in \Phi^1_{L_2}$$

Pour représenter toutes les séquences de l'ensemble $x[[\varphi^{i+1}]]$, il faut au moins $|\Pi^*_i|$ éléments de Π^*_{i+1} dont la projection sur E_1 contient x. Donc en considérant la définition de q^*, on obtient :

$$|\Pi^*_{i+1}| \sum_{\pi \in \Phi^1_{L_2}} \mathbb{1}_\pi(x) q^*_\pi \geq |\Pi^*_i|$$

Avec $\mathbb{1}_\pi(x) = 1$ si $x \in [[\pi]]$ et 0 sinon. Ensuite, puisque cela vaut pour tout x,

$$|\Pi^*_{i+1}| \min_{x \in [[\varphi_1]]} \sum_{\pi \in \Phi^1_{L_2}} \mathbb{1}_\pi(x) q^*_\pi \geq |\Pi^*_i|$$

Et, à plus forte raison,

$$|\Pi^*_{i+1}| \max_{q \in \mathcal{Q}} \min_{x \in [[\varphi_1]]} \sum_{\pi \in \Phi^1_{L_2}} \mathbb{1}_\pi(x) q_\pi \geq |\Pi^*_i|$$

Si l'on pose $K = \max\limits_{q \in \mathcal{Q}} \min\limits_{x \in [[\varphi_1]]} \sum_{\pi \in \Phi^1_{L_2}} \mathbb{1}_\pi(x) q_\pi$ on obtient

$$|\Pi^*_{i+1}| K \geq |\Pi^*_i|$$

Or, K est une constante strictement inférieure à 1.
En effet, supposons $\exists q, \min\limits_{x \in [[\varphi_1]]} \sum\limits_{\pi \in \Phi^1_{L_2}} \mathbb{1}_\pi(x) q_\pi = 1$.
Alors $\forall x \in [[\varphi_1]], \forall \pi \in \Phi^1_{L_2}, \ \mathbb{1}_\pi(x) = 0 \implies q_\pi = 0$.
D'autre part, $\forall q, \exists \pi, q_\pi > 0$. Soit donc un tel π, si $\exists x, x \notin [[\pi]]$, alors on doit avoir $q_\pi = 0$, ce qui n'est pas le cas. Mais à l'inverse, si $\forall x, x \in [[\pi]]$ alors $[[\pi]] = [[\varphi_1]]$,

2. Ce théorème établit une borne inférieure pour un problème de couverture d'ensembles issu du produit cartésien de problèmes de couverture d'ensembles identiques. Dans notre cas, nous cherchons à "couvrir" (représenter) toutes les séquences représentées par φ^i par des ensembles de séquences représentés par des formules de $\Phi^i_{L_2}$

ce qui n'est pas possible non plus. Donc $K < 1$.

Par récurrence, on peut établir la croissance exponentielle en i du nombre de formules de $L_2(E^i)$ nécessaires pour représenter l'ensemble $[[\varphi^i]]$. Par conséquent, $\forall \Pi \in L_2(E^i)$, même si $\forall \pi \in \Pi, |\pi| = 1$, $\forall P \in \mathbb{R}[X], ([[\Pi]] = [[\varphi^i]]) \implies |\Pi| > P(|\varphi^i|)$: SL_2 n'est pas plus compact que SL_1.

□

En essence, ce résultat nous permet de déduire d'une différence d'expressivité entre langages, une différence de compacité entre langages-ensemble correspondants. L'idée est de concaténer des formules d'un langage qui n'ont pas de représentation dans un autre langage de façon à ce que le nombre de formules nécessaires à la représentation grandisse exponentiellement.

7.1.3 Propriété des langages PRD et des langages restreints en expressivité

Théorème 7.1.3.1. *Soit un langage restreint $L^* \in \mathcal{L}^*$. Soit T une translation entre L_{1M} et L^*. Le langage à une machine est au moins aussi expressif que L^* modulo T :*
$L^* \geq_e^T L_{1M}$

Démonstration. La preuve suit de la définition de L^* : une restriction des formules de L à celles pour lesquelles L_{1M} possède une formule équivalente modulo T. □

Théorème 7.1.3.2. *Soit un langage L et son fragment restreint $L^* \in \mathcal{L}^*$. Le langage L est au moins aussi expressif que son fragment L^* :*
$\forall L, L^* \geq_e L$

Démonstration. Suit de la définition de L^* : une restriction des formules de L. □

Théorème 7.1.3.3. *Soient L_1 et L_2 deux langages de représentation des mêmes objets, et T une translation entre L_{1M} et L_1 (et entre L_{1M} et L_2). Si, pour une relation $\leq \in \{\leq_e, \leq_s\}$, $(L_1 \leq L_2)$ alors $(L_1^* \leq L_2^*)$*

Démonstration. Pour la relation d'expressivité : soit $\varphi_2 \in \Phi_{L_2^*}$, par définition, il existe une formule φ de L_{1M} telle que $[[\varphi]]T[[\varphi_2]]$. Or, par définition, si $L_1 \leq_e L_2$, alors $\forall \varphi_2 \in \Phi_{L_2}, \exists \varphi_1 \in \Phi_{L_1}, [[\varphi_1]] = [[\varphi_2]]$. Nous avons donc aussi $[[\varphi]]T[[\varphi_1]]$. Par conséquent, $\varphi_1 \in L_1^*$, et $\forall \varphi_2 \in L_2^*, \exists \varphi_1 \in L_1^*, [[\varphi_1]] = [[\varphi_2]]$.
La preuve pour la relation de compacité est similaire. □

Remarquons la contraposée de ce théorème $(L_1^* \not\leq L_2^*) \implies (L_1 \not\leq L_2)$. Ce théorème nous permet d'obtenir les résultats d'expressivité et de compacité pour les langages originel et restreints en une unique preuve.

Théorème 7.1.3.4. *Soit L un langage PRD de représentation de séquence. Soit R un requête parmi {CO,MX,CD_0,$R\uparrow$,$E\uparrow$,BC/WC}.*
Si L supporte la requête R, alors L^* la supporte aussi

Démonstration. Pour toutes les query RQ, si L supporte RQ, alors $\exists A, \forall \varphi \in \Phi_L$, A retourne une sortie satisfaisante en temps polynomial. Or L^* est un fragment de L, donc en particulier, L^* supporte RQ.

D'un autre côté, toutes les transformations RT que nous considérons (CD_0,$R\uparrow$,$E\uparrow$) correspondent à des modifications de l'instance du problème à une machine originelle vers une autre instance similaire (en d'autres termes, L_{1M} supporte ces transformations).
Soient $\psi = (E, PRD) \in \Phi_{1M}$, et $\varphi = (\pi, PRD^{3}) \in L$, tel que $[[\psi]]\mathbb{T}[[\varphi]]$. Puisque L_{1M} supporte RT, soit $A(\psi) = (E', PRD')$ (une modification de l'instance du problème, soit des données temporelles par CD_0 ou $R\uparrow$, soit des contraintes de précédence par $E\uparrow$) et puisque L supporte RT, supposons aussi $A'(\varphi) = (\pi', PRD')$.
Tout d'abord, $\forall \check{s} \in [[A(\psi)]]$, $\check{s}$ satisfait à la fois les contraintes de précédence E' et les contraintes temporelles PRD' de $A(\psi)$. De plus on sait $\exists \check{\sigma} \in \varphi, \check{s}\tau\check{\sigma}$, donc $\check{\sigma}$ vérifie les contraintes séquentielles, et $\check{\sigma}$ est cohérent avec PRD'. Donc $\check{\sigma} \in [[A'(\varphi)]]$.
Donc pour tout ordonnancement $\check{s} \in [[A(\psi)]]$, il existe une séquence $\check{\sigma} \in [[A'(\varphi)]]$ telle que $\check{s}\tau\check{\sigma}$.
D'autre part, $\forall \check{\sigma} \in [[A(\varphi)]]$, $\check{\sigma}$ est cohérent avec PRD', car L est un langage cohérent, donc $\exists \check{s} \models PRD', order(\check{s}) = \check{\sigma}$. De plus, puisque $\check{\sigma} \in [[\varphi]]$, $\check{s}$ satisfait aussi E'.
Donc pour toute séquence $\check{\sigma} \in [[A'(\varphi)]]$, il existe un ordonnancement $\check{s} \in [[A(\psi)]]$, tel que $\check{s}\tau\check{\sigma}$.
On en conclut $[[A(\psi)]]\mathbb{T}[[A'(\varphi)]]$, donc $A(\varphi)$ est une formule de L^*, et donc L^* supporte RT. □

Il suffira donc dans notre cas de montrer qu'un langage supporte une requête pour savoir que le langage restreint associé la supporte aussi.

7.1.4 Propriétés de langages de représentation spécifiques

7.1.4.1 Propriétés des séquences de groupes

Théorème 7.1.4.1. *Soient $\check{\sigma}$ and $\check{\sigma}'$ deux séquences couvertes par une unique formule φ de L_{POG}. Si deux tâches apparaissent dans des ordres différents dans $\check{\sigma}$ et $\check{\sigma}'$, alors les deux tâches appartiennent au même groupe :*
$\forall A, B \in N, (\check{\sigma}_A > \check{\sigma}_B \wedge \check{\sigma}'_B > \check{\sigma}'_A) \implies (G(A) = G(B))$

Démonstration. Si $\check{\sigma} \models \varphi \in \Phi_{L_{POG}}$, on a pour toute paire de tâche i et j que $G(i) < G(j) \implies \check{\sigma}_i < \check{\sigma}_j$. Par contraposition, on a donc $\check{\sigma}_i \geq \check{\sigma}_j \implies G(i) \geq G(j)$.
Donc en particulier, $G(A) \geq G(B)$ mais aussi $G(B) \geq G(A)$. D'où le résultat. □

7.1.4.2 Propriétés des arbres PQR

Théorème 7.1.4.2. *Soit une formule φ de L_{PQR}. Soit $\Delta(A, B)$ le nœud racine du plus petit sous arbre de φ contenant les tâches A et B ($\{A, B\} \subseteq Jobs(\Delta(A, B))$).*

3. Nous avons évoqués plus tôt que puisque $[[\psi]]\mathbb{T}[[\varphi]]$, on peut se ramener aux données temporelles PRD

Si pour toutes les séquences admissibles par φ , A et B sont dans le même ordre, alors $\Delta(A,B)$ est un nœud de type R :

$$(\forall \check{\sigma} \models \varphi, \check{\sigma}_A < \check{\sigma}_B) \vee (\forall \check{\sigma} \models \varphi, \check{\sigma}_B < \check{\sigma}_A) \Leftrightarrow \Delta(A,B) \in R$$

Démonstration. Supposons $(\forall \check{\sigma} \models \varphi, \check{\sigma}_A < \check{\sigma}_B)$ (clairement on n'a pas en même temps $(\forall \check{\sigma} \models \varphi, \check{\sigma}_B < \check{\sigma}_A)$, mais remarquons que les deux cas sont symétriques). Soit $Child$[4]$(\Delta(A,B)) = c_1, \ldots, c_k$, par définition, $\forall i, \{A,B\} \not\subseteq Jobs(c_i)$, sinon on aurait $\Delta(A,B) = c_i$. Donc $\exists i,j$, avec $A \in Child(c_i)$ et $b \in Child(c_j)$. Par conséquent si $\Delta(A,B)$ est un nœud de type Q ou P, $\exists \check{\sigma}' \models \varphi, \check{\sigma}'_A > \check{\sigma}'_B$, ce qui contredit notre hypothèse. De plus, si $\Delta(A,B)$ est un nœud de type R, alors on voit que $(\forall \check{\sigma} \models \varphi, \check{\sigma}_A < \check{\sigma}_B)$ si $i < j$ et $(\forall \check{\sigma} \models \varphi, \check{\sigma}_B < \check{\sigma}_A)$ sinon. □

7.1.4.3 Propriétés du langage à une machine

Théorème 7.1.4.3. *Soit une formule φ d'un langage de représentation d'ensembles de séquences $L(\chi_{seq})$ et φ' une formule de L_{1M} telle que $[[\varphi']]\mathbb{T}[[\varphi]]$. Soit une séquence $\check{\sigma}$ et position$(\check{\sigma}) = \check{\zeta}$. Si $\check{\sigma}$ est admissible pour φ alors il existe une façon d'ordonnancer les tâches de toutes ses sous-séquences dans l'ordre sans violer les contraintes de φ' :*
$\check{\sigma} \models \varphi \implies \forall F \subseteq N, \check{\zeta}_{|F} = \check{\zeta}_1, \ldots \check{\zeta}_{|F|},\ r_{\check{\zeta}_1} + \sum_{i=1\ldots k} p_{\check{\zeta}_i} \leq d_{\check{\zeta}_{|F|}}$

Démonstration. En effet, on sait que $[[\varphi]]\mathbb{T}[[\varphi']]$ donc $(\check{\sigma} \models \varphi) \implies (\exists \check{s} \models \varphi', order(\check{s}) = \check{\sigma})$. Puisque $\check{s} \models \varphi'$, on a $r_i \leq \check{s}_i \leq d_i - p_i \quad \forall i \in N$ et $\check{s}_{\check{\zeta}_i} + p_{\check{\zeta}_i} \leq \check{s}_{\check{\zeta}_{i+1}} \forall i \in N$ car position$(order(\check{s})) = \check{\zeta}$.
En particulier, considérons une sous-séquence des tâches F avec $\check{\zeta}_{|F} = \check{\zeta}_1, \ldots \check{\zeta}_{|F|}$. On a $r_{\check{\zeta}_1} \leq \check{s}_{\check{\zeta}_1}$, et $\check{s}_{\check{\zeta}_{|F|}} + p_{\check{\zeta}_{|F|}} \leq d_{\check{\zeta}_{|F|}}$, en appliquant itérativement $\check{s}_{\check{\zeta}_i} + p_{\check{\zeta}_i} \leq \check{s}_{\check{\zeta}_{i+1}}$, on obtient au final $r_{\check{\zeta}_1} + \sum_{i=1\ldots k} p_{\check{\zeta}_i} \leq d_{\check{\zeta}_{|F|}}$.
□

Par exemple, si AB est admissible pour une formule φ, les trois inégalités suivantes sont vérifiées :
- $A : r_A + p_A \leq d_A$
- $B : r_B + p_B \leq d_B$
- $AB : r_A + p_A + p_B \leq d_B$

Théorème 7.1.4.4. *Soit toujours une formule φ d'un langage de représentation d'ensembles de séquences $L(\chi_{seq})$ et φ' une formule de L_{1M} telle que $[[\varphi']]\mathbb{T}[[\varphi]]$. Mais supposons maintenant $\varphi' = (\emptyset, PRD)$. Soit une séquence $\check{\sigma}$ et position$(\check{\sigma}) = \check{\zeta}$. Si $\check{\sigma}$ n'est pas admissible pour φ alors il existe une sous-séquence critique pour laquelle il n'est pas possible d'ordonnancer les tâches dans l'ordre sans violer les contraintes de φ' :*
$\check{\sigma} \not\models \varphi \implies \exists F \subseteq N, \check{\zeta}_{|F} = \check{\zeta}_1, \ldots \check{\zeta}_{|F|}, d_{\check{\zeta}_{|F|}} < r_{\check{\zeta}_1} + \sum_{i=1\ldots k} p_{\check{\zeta}_i}$

4. On rappelle que $Child(n)$ sont les nœuds enfants du nœud n et $Jobs(n)$ sont les tâches-feuille du sous arbre de racine n.

Démonstration. De façon similaire , $(\check{\sigma} \not\models \varphi) \implies (\forall \check{s}, order(\check{s}) = \check{\sigma} \implies \check{s} \not\models \varphi')$. Considérons le cas particulier dans lequel $\check{s}$ est l'ordonnancement semi-actif correspondant à la séquence $\check{\sigma}$. Dans l'ordonnancement semi-actif, les dates de disponibilité sont respectées $\check{s}_i > r_i, \forall i$, et les contraintes disjonctives aussi puisqu'un ordre est défini par la séquence $\check{s}_{\check{\zeta}_i} + p_{\check{\zeta}_i} \leq \check{s}_{\check{\zeta}_{i+1}} \forall i$. De plus, on sait que $E = \emptyset$, donc

$$\check{s} \not\models \varphi' \implies \bigvee_i \check{s}_i + p_i > d_i$$

.

Puisque $\check{s}$ est semi-actif, $\check{s}_{\check{\zeta}_i} = \max(r_{\check{\zeta}_i}, \check{s}_{\check{\zeta}_{i-1}} + p_{\check{\zeta}_{i-1}})$. On a que si $(\check{s}_{\check{\zeta}_i} + p_{\check{\zeta}_i} > d_{\check{\zeta}_i})$ alors $(r_{\check{\zeta}_i} + p_{\check{\zeta}_i} > d_{\check{\zeta}_i} \vee \check{s}_{\check{\zeta}_{i-1}} + p_{\check{\zeta}_{i-1}} + p_{\check{\zeta}_i} > d_{\check{\zeta}_i})$. En développant itérativement le côté droit, on obtiens que $\check{s} \not\models \varphi'$ implique l'existence d'une sous séquence de tâches (plus spécifiquement, une sous-séquence successive de tâches [5]) de $\check{\sigma}$, $\check{\zeta}_{|F} = \check{\zeta}_1, \ldots \check{\zeta}_{|F|}$ telle que $r_{\check{\zeta}_1} + \sum_{i=1\ldots k} p_{\check{\zeta}_i} > d_{\check{\zeta}_{|F|}}$.

□

Par exemple, si BA est inadmissible pour une formule φ, l'une des trois inégalités suivantes doit être vérifiée :

— $B : r_B + p_B > d_B$
— $A : r_A + p_A > d_A$
— $BA : r_B + p_B + p_A > d_A$

On peut en général écarter certaines possibilitées en considérant les séquences admissibles qui contiennent certaines sous-séquences. Par exemple, si la première inégalitée est vérifiée, aucune séquence contenant B ne pourrait être admissible. Si la séquence AB est admissible, on sait que les sous séquences A et B sont admissibles, c'est donc que c'est la troisième inégalité qui est vérifiée.

7.2 Résultats d'expressivité

Dans cette section, nous justifions les résultats résumés dans la figure 7.1b.

7.2.1 Expressivité du langage à une machine

Le langage à une machine est strictement moins expressif que les langages complets de séquences (En particulier, il est moins expressif que les langages-ensemble complets, et le langage des MDDs).

Théorème 7.2.1.1. *Soit L un langage de représentation d'ensembles de séquences complet : $L_{1M} >_e^{\mathbb{T}} L$*

Démonstration. En effet, d'un côté on à $L_{1M} \geq_e^{\mathbb{T}} L$ puisque L est complet et $\mathbb{T}$ satisfait les conditions du théorème 7.1.1.1.
Mais de l'autre côté, $L \not\geq_e^{\mathbb{T}} L_{1M} : \exists \varphi \in \Phi_L, \forall \varphi' \in \Phi_{L_{1M}}, ([[\varphi']], [[\varphi]]) \notin \mathbb{T}$. En effet soit $[[\varphi]] = \{ABC, BCA, CAB\} \in L$ (un tel φ existe car L est complet), nous

5. Ce qui correspond à tous les chemins critiques possibles

montrons qu'il n'existe pas $\varphi' = (E, PRD') \in L_{1M}$ tel que $[[\varphi']]\mathbb{T}[[\varphi]]$.
Premièrement, un tel φ' aurait $E = \emptyset$, car il doit y avoir un ordonnancement $\check{s} \models \varphi'$ tel que $order(\check{s}) = \check{\sigma}, \forall \check{\sigma} \in [[\varphi]]$, par exemple, il doit y avoir $\check{s}$ tel que $order(\check{s}) = ABC$, donc $(B, A) \notin E, (C, A) \notin E, (C, B) \notin E$. Dans l'ensemble des séquences représentées par φ, toutes les paires de tâches apparaissent dans tous les sens, donc aucune contrainte de précédence ne peut être dans E.
Ensuite, les inégalités suivantes sont vérifiées :

$$d_B < r_A + \sum_{k \in A,B,C} p_k \leq d_C$$
$$d_C < r_B + \sum_{k \in A,B,C} p_k \leq d_A$$
$$d_A < r_C + \sum_{k \in A,B,C} p_k \leq d_B$$

En effet, par exemple, puisque $ABC \models \varphi$, le théorème 7.1.4.3 donne $r_A + \sum_{k \in A,B,C} p_k \leq d_C$. Il en est de même pour les séquences BCA et CAB.
De plus, $ACB \not\models \varphi$, d'après le théorème 7.1.4.4 il existe un sous ensemble successif des tâches de ACB qui doit ne pas satisfaire les contraintes du problème. Or, nous avons vu plus tôt que toutes les sous séquences de taille 2 sont des sous séquences de séquences acceptées et le théorème 7.1.4.3 établit donc qu'elles doivent satisfaire les contraintes du problème. Par conséquent, il doit être vrai que $d_B < r_A + \sum_{k \in A,B,C} p_k$.
Et il en est de même pour les autres séquences rejetées BAC et CBA.

Le jeu d'inégalité présenté doit donc être vérifié, mais nous pouvons voir qu'il n'admet pas de solutions. Par conséquent il n'existe pas de telle instance φ' ; et L est strictement plus expressif que L_{1M}.

□

Par conséquent, tout langage au moins aussi expressif que L_{1M}, une fois restreint à L^*, est aussi expressif. le théorème suivant nous permettra donc de comparer les langages dans la section 7.3 :

Théorème 7.2.1.2. *Soit L un langage de représentation d'ensembles de séquences :* $(L_{1M} \geq_e^{\mathbb{T}} L) \implies (L_{1M} \sim_e^{\mathbb{T}} L^*)$

Démonstration. La preuve découle du théorème 7.1.3.1. □

Dans la section suivante (7.3), nous étudions la compacité de langages ayant la même expressivité, ce sont donc les langages L^* que nous comparerons (voir théorème 7.2.1.2). Mais dans cette section également, lorsque l'on compare l'expressivité de deux langages, il est pertinent de le faire pour les langages restreints associés, car des différences d'expressivité concernant des objets que notre langage source ne permet pas d'exprimer n'est pas pertinent. Par la suite, lorsque nous comparons

deux langages, nous proposons une représentation équivalente dans le langage à une machine des objets pour lesquels on observe une différence d'expressivité. Remarquons que ces résultats négatifs (lorsqu'un langage ne peut représenter un objet) s'étendent aux langages non-restreints par la contraposée du théorème 7.1.3.3.

7.2.2 Expressivité des langages L_{POG}, L_{PO} et L_{AOPO}

Nous pouvons tout d'abord établir que L_{AOPO} est au moins aussi expressif que L_{PO}, qui est au moins aussi expressif que L_{POG}, lui même au moins aussi expressif que L_{SEQ}.

Théorème 7.2.2.1. $L_{AOPO} \leq_e L_{PO} \leq_e L_{POG} \leq_e L_{SEQ}$

Démonstration. Il est connu que L_{AOPO} généralise L_{PO} (nous avons vu qu'une condition d'attente (X, i) est équivalente à une contrainte de précédence lorsque $|X| = 1$), que L_{PO} généralise L_{POG} (une séquence de groupes d'opérations permutables est un ordre partiel particulier), et que L_{POG} généralise L_{SEQ} (une séquence correspond au cas ou il y a une seule tâche par groupe). Donc par construction, on peut facilement trouver, pour chaque formule d'un langage, une formule équivalente dans un langage qui le généralise. D'où le résultat. □

Notons que ces résultats sont aussi valables pour les langages L^* correspondants, en vertu du théorème 7.1.3.3.

Le langage des ordres partiel n'est pas aussi expressif que le langage des "And/Or" ordres partiels.

Théorème 7.2.2.2. $L^*_{PO} \not\leq_e L^*_{AOPO}$

Démonstration. Soit l'instance du problème à une machine suivante, avec $N = \{A, B, C\}$:

$$\psi_1 = \begin{cases} p_i = 1 & \forall i \in N \\ r_A = r_B = 0 & \\ r_C = 1 & \\ d_i = 3 & \forall i \in N \\ E = \emptyset & \end{cases}$$

On peut vérifier que les ordonnancements acceptés par $\psi_1 = (\emptyset, PRD)$ correspondent aux séquences ABC, ACB, BCA, et BAC (C ne peut être en première position sans qu'une tâche termine après sa date de livraison). Soit $\varphi_{AOPO} = (\{(A, B), C\}, PRD) \in L_{AOPO}$. On peut voir que $[[\psi_1]]\mathbb{T}[[\varphi_{AOPO}]]$ (et par conséquent φ_{AOPO} est une formule de L^*_{AOPO}.
Supposons maintenant qu'il existe $\varphi = (\pi, PRD') \in L_{PO}$ tel que $[[\varphi_{AOPO}]] = [[\varphi]]$. $\forall (i, j) \in N^2$, il ne peut y avoir $(i, j) \in \pi$, car tous les ordres de deux tâches sont possibles parmi $\{ABC, ACB, BCA, BAC\}$. Par conséquent $\pi = \emptyset$, Mais dans ce

cas on aurait par exemple $CBA \models \varphi$ (car toutes les séquences sont admissibles pour l'ensemble vide de contraintes de précédence), donc $[[\varphi_{AOPO}]] \neq [[\varphi]]$. En conclusion, il n'existe pas un tel φ. □

Ce résultat implique $L_{PO} \not\leq_e L_{AOPO}$ par la contraposée du théorème 7.1.3.3.

Le langage des séquences de groupes d'opérations permutables n'est pas aussi expressif que le langage des ordres partiels.

Théorème 7.2.2.3. $L^*_{POG} \not\leq_e L^*_{PO}$

Démonstration. De façon similaire, soit l'instance suivante ($N = \{A, B, C\}$) :

$$\psi_2 = \begin{cases} p_i = 1 & \forall i \in N \\ r_i = 0 & \forall i \in N \\ d_i = 3 & \forall i \in N \\ E = \{(A, B)\} \end{cases}$$

Les ordonnancements acceptés par $\psi_2 = (E, PRD)$ correspondent aux séquences dans lesquelles A est avant B : ABC, ACB, et CAB. On voit que $\varphi_{PO} = (\{(a, b)\}, PRD)$ par exemple est tel que $[[\psi_2]]\mathbb{T}[[\varphi_{PO}]]$ (donc φ_{PO} est une formule de L^*_{PO}). Soit $\varphi = (\pi, PRD') \in L_{POG}$ tel que $[[\varphi_{PO}]] = [[\varphi]]$. En particulier, $ABC \models \varphi$ et $CAB \models \varphi$. Par le théorème 7.1.4.1, $G(A) = G(B) = G(C)$, toutes les tâches sont dans le même groupe : $\pi = [\{A, B, C\}]$, mais dans ce cas, on aurait aussi par exemple $CBA \models \varphi$, donc $[[\varphi_{PO}]] \neq [[\varphi]]$. En conclusion, il n'existe pas un tel φ. □

Ce résultat implique $L_{POG} \not\leq_e L_{PO}$ par la contraposée théorème 7.1.3.3.

Le langage des "And/Or" ordres partiels est incomparable au langage à une machine (modulo $\mathbb{T}$).

Théorème 7.2.2.4. $L_{AOPO} \leqslant^{\mathbb{T}}_e L_{1M}$

Démonstration. Nous montrons d'abord $L_{1M} \not\leqslant^{\mathbb{T}}_e L_{AOPO}$: Soit l'instance suivante ($N = \{A, B, C\}$) :

$$\psi_3 = \begin{cases} p_i = 1 & \forall i \in N \\ r_B = 1 & \\ d_B = 2 & \\ r_i = 0 & \forall i \in \{A, C\} \\ d_i = 3 & \forall i \in \{A, C\} \\ E = \emptyset & \end{cases} \tag{7.1}$$

On peut voir que dans cette instance, la tâche B doit être ordonnancé au temps 1 et toutes les tâches de N doivent toute être ordonnancées entre les temps 0 et 3, donc

les ordonnancements admissibles par $\psi_3 = (\emptyset, PRD)$ correspondent aux séquences ABC et CBA. Supposons qu'il existe $\varphi = (\pi, PRD') \in L_{AOPO}$ tel que $[[\psi_3]]\mathbb{T}[[\varphi]]$. Si π contient une condition d'attente de la forme (X, A) ou (X, C), aucune séquence admissible ne pourrait avoir la tâche A ou C en première position. Ce ne peut donc pas être le cas. Mais π ne peut pas non plus contenir de condition d'attente de la forme (X, B) avec $|X| = 1$, puisque ni A ni C ne doivent nécessairement être positionnés avant B. Ne reste plus que la condition d'attente $(\{A, C\}, B)$, mais celle ci-ne suffit pas puisque si π contient seulement cette condition d'attente, la séquence ACB est admissible par exemple. On conclut qu'il n'existe pas de tel φ.

Montrons ensuite $L_{AOPO} \not\geq_e^{\mathbb{T}} L_{1M}$: Soit la formule suivante de L_{AOPO}, avec $N = \{A, B, C, D\}$ (on rappelle que PRD_∞ correspond à des données temporelles pour lesquelles toutes les données séquentielles sont cohérentes) :

$$\varphi = (\{(\{A, C\}, B), (\{B, D\}, A)\}, PRD_\infty)$$

Montrons qu'il n'y a pas d'instance équivalente ψ dans L_{1M} telle que $[[\psi]]\mathbb{T}[[\varphi]]$. On peut vérifier que φ couvre les séquences $CBAD$, $CBDA$, $CDAB$, $CDBA$, $DABC$, $DACB$, $DCAB$, et $DCBA$; et ne couvre pas les 16 autres séquences de 4 tâches. Remarquons qu'en particulier, $CBAD \models \varphi$ et $DABC \models \varphi$, donc toutes les sous-séquences de taille 2 apparaissent dans des séquences acceptées, et donc nécessairement, $\psi = (\emptyset, PRD)$.
Remarquons derechef que puisque $CBAD \models \varphi$, le théorème 7.1.4.3 donne en particulier $r_B + \sum_{A,B,D} p_i \leq d_D$ (sous séquence BAD)). D'autre part, $CABD \not\models \varphi$, donc le théorème 7.1.4.4 donne qu'il existe une sous séquence critique de $CABD$. Nous avons vu que toutes les sous-séquences consécutives de 2 tâches ne sont pas critiques, et parmi les sous séquences consécutives de taille 3, CAB fait partie des sous-séquences de $CDAB$ (qui est accepté), et donc n'est pas critique. De plus, on ne peut avoir que la "sous-séquence" de taille 4 est critique puisque $CBAD$ est accepté, et donc $r_C + \sum_{A,B,C,D} p_i \leq d_D$. C'est donc que ABD est critique : $r_A + \sum_{A,B,D} p_i > d_D$.
Un argument similaire peut être établi à l'aide des séquences $DABC$, $DBAC$ et $CDBA$; mais il est plus simple de constater la symétrie de A avec B et D avec C dans φ pour établir au final que l'instance $\psi = (\emptyset, PRD) \in L_{1M}$ doit vérifier :

$$r_A + \sum_{A,B,D} p_i > d_D \geq r_B + \sum_{A,B,D} p_i$$
$$r_B + \sum_{A,B,C} p_i > d_C \geq r_A + \sum_{A,B,C} p_i$$

Ce qui implique $r_A > r_B$ et $r_B > r_A$. En conclusion, il n'existe pas de telle instance ψ. □

On obtiens aussi $L^*_{AOPO} >_e^{\mathbb{T}} L_{1M}$ puisque $(L \not\leq_e^{\mathbb{T}} L_{1M}) \implies (L^* \not\leq_e^{\mathbb{T}} L_{1M})$, et par le théorème 7.1.3.1.

Au final, on a donc :

$$L_{1M} \lesseqgtr_e^{\mathbb{T}} L_{AOPO} <_e L_{PO} <_e L_{POG}$$

mais aussi

$$L_{1M} <_e^{\mathbb{T}} L^*_{AOPO} <_e L^*_{PO} <_e L^*_{POG}$$

Puisque la relation d'incomparabilité n'est pas transitive, nous montrons ensuite que le langage à une machine est strictement plus expressif que le langage des ordres partiels :

Théorème 7.2.2.5. $L_{1M} <_e L_{PO}$

Démonstration. D'une part , on a $L_{PO} \not\leq_e^{\mathbb{T}} L_{1M}$ par le corollaire 7.1.1.2, puisque $L_{AOPO} \not\leq_e^{\mathbb{T}} L_{1M}$ et $L_{AOPO} \leq_e L_{PO}$.
D'autre part, $L_{PO} \geq_e^{\mathbb{T}} L_{1M}$. Par construction, soit $\varphi = (\pi, PRD) \in L_{PO}$. Alors $\psi = (\pi, PRD) \in L_{1M}$ est tel que $[[\psi]]\mathbb{T}[[\varphi]]$. Donc $\forall \varphi \in \Phi_{L_{PO}}, \exists \psi \in L_{1M}, [[\psi]]\mathbb{T}[[\varphi]]$ □

On a donc aussi $L_{1M} <_e L^*_{PO}$.

Théorème 7.2.2.6. *Les langages L_{SSEQ}, L_{SPOG}, L_{SPO} et L_{SAOPO} sont complets.*

Démonstration. La preuve découle du théorème 7.1.1.4. Puisque le langage L_{SEQ} par définition peut représenter toute les séquences, et par le résultat précédent $L_{AOPO} \leq_e L_{PO} \leq_e L_{POG} \leq_e L_{SEQ}$, donc ces langages peuvent aussi représenter des séquences uniques. □

7.2.3 Expressivité des arbres PQR

Le langage des arbres PQR est incomparable au langage à une machine.

Théorème 7.2.3.1. $L_{PQR} \lesseqgtr_e^{\mathbb{T}} L_{1M}$

Démonstration. Montrons en premier lieu $L_{PQR} \not\geq_e^{\mathbb{T}} L_{1M}$.
Soit $\varphi = (\pi, PRD_\infty) \in L_{PQR}$, avec π l'arbre PQR de la figure 7.2.

On remarque que φ couvre les séquences dans lesquelles A et B sont adjacents et C et D sont adjacents. Soit $\psi = (E, PRD) \in L_{1M}$ tel que $[[\psi]]\mathbb{T}[[\varphi]]$.
Puisque en particulier les séquences $ABCD$ et $CDBA$ sont couvertes par φ, $E = \emptyset$. Ensuite, puisque en particulier, φ couvre $ABCD$ et $CDAB$, le théorème 7.1.4.3 donne d'une part $d_B \geq r_C + \sum_{i \in N} p_i$ et $d_D \geq r_A + \sum_{i \in N} p_i$, et d'autre part que les sous-séquences CAB,ABD,ACD, et CDB ne sont pas critiques. Or $CABD \not\models \varphi$ et $ACDB \not\models \varphi$, donc par le théorème 7.1.4.4, et puisque leurs sous-séquences de taille 2 et 3 ne sont pas critiques, on a $r_C + \sum_{i \in N} p_i > d_D$ et $r_A + \sum_{i \in N} p_i > d_B$. L'instance ψ doit donc vérifier :

$$d_B \geq r_C + \sum_{i \in N} p_i > d_D \geq r_A + \sum_{i \in N} p_i > d_B \tag{7.2}$$

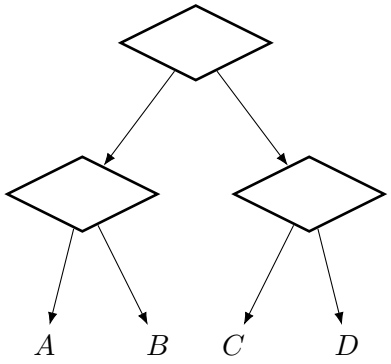


FIGURE 7.2 – Arbre PQR n'ayant pas d'instance de L_{1M} équivalente

On en déduit qu'une telle instance n'existe pas, et donc $L_{PQR} \not\geq_e^{\mathbb{T}} L_{1M}$.

Nous montrons ensuite $L^*_{PO} \not\geq_e L^*_{PQR}$: soit un ordre partiel $\varphi = ([(A, B)], PRD_\infty)$ avec $N = \{A, B, C\}$. Remarquons que les séquences que couvre φ sont ABC, ACB, et CAB. Remarquons de plus que puisque $L_{1M} \leq_e L_{PO}$, il existe une instance de L_{1M} correspondante, donc φ est une formule de L^*_{PO}. Soit φ' une formule de L_{PQR} tel qu'il couvre les mêmes séquences : $[[\varphi']] = [[\varphi]]$. Le théorème 7.1.4.2 donne que $\Delta(A, B)$ est de type R, et sa contraposée donne que $\Delta(A, C)$ et $\Delta(B, C)$ ne sont pas de type R. Trois cas sont possibles :

- Soit $\Delta(A, B)$ est un sous arbre de $\Delta(A, C) = \Delta(B, C)$, auquel cas $ACB \not\models \varphi'$.
- Soit $\Delta(B, C)$ est un sous arbre de $\Delta(A, B)$, auquel cas $CAB \not\models \varphi'$
- Soit $\Delta(A, C)$ est un sous arbre de $\Delta(A, B)$, auquel cas $ABC \not\models \varphi'$

Donc il n'existe pas un tel φ', donc $L^*_{PO} \not\geq_e L^*_{PQR}$ et $L_{PO} \not\geq_e L_{PQR}$ (corollaire du th. 7.1.3.3).
Par le corollaire 7.1.1.1, puisque $L_{PO} \geq_e^{\mathbb{T}} L_{1M}$ et $L_{PO} \not\geq_e L_{PQR}$, on a $L_{1M} \not\geq_e^{\mathbb{T}} L_{PQR}$. D'où la conclusion. □

On a aussi $L^*_{PQR} >_e^{\mathbb{T}} L_{1M}$ par le théorème 7.1.3.1.

Le langage des arbres PQR est incomparable au langage des ordres partiels et des "And/Or" ordres partiels.

Théorème 7.2.3.2. *$L^*_{PQR} \leqslant_e L^*_{AOPO}$ et $L^*_{PQR} \leqslant_e L^*_{PO}$*

Démonstration. Tout d'abord, nous savons que $L^*_{PO} \not\geq_e L^*_{PQR}$ donc puisque $L^*_{AOPO} \leq_e L^*_{PO}$, on a $L^*_{AOPO} \not\geq_e L^*_{PQR}$ par le corollaire 7.1.1.1.
Ensuite, nous montrons $L^*_{PQR} \not\geq_e L^*_{AOPO}$. Nous avons montré plus tôt que l'instance du problème à une machine $\psi_3 = (\emptyset, PRD)$ (voir 7.1) n'a pas de représentation équivalente dans L_{AOPO}. Cependant, la figure 7.3 montre comment l'arbre PQR d'une formule $\varphi = (\pi, PRD) \in L^*_{PQR}$ permet de représenter cette instance à l'aide d'un nœud Q.

De plus, puisque $L^*_{AOPO} \leq_e L^*_{PO}$, le corollaire 7.1.1.1 donne $L^*_{PQR} \not\geq_e L^*_{PO}$. □

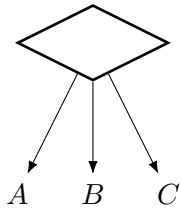


FIGURE 7.3 – Arbre PQR π correspondant à ψ_3

On a aussi $L_{PQR} \lesseqgtr_e^{\mathbb{T}} L_{AOPO}$ et $L_{PQR} \lesseqgtr_e^{\mathbb{T}} L_{PO}$ par la contraposée du théorème 7.1.3.3

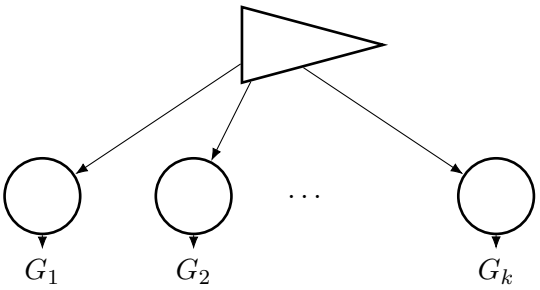
Le langage des séquences de groupes d'opérations permutables est strictement moins expressif que le langage des arbres PQR.

Théorème 7.2.3.3. $L^*_{PQR} <_e L^*_{POG}$

Démonstration. Tout d'abord, puisque $L^*_{PO} \leq_e L^*_{POG}$ et $L^*_{PO} \not\leq_e L^*_{PQR}$, on a $L^*_{POG} \not\leq_e L^*_{PQR}$ (corollaire 7.1.1.2).
De plus, montrons $L^*_{PQR} \leq_e L^*_{POG}$. En effet, la figure 7.4 montre que pour toute formule φ de L_{POG}, il est possible de construire une formule φ' de L_{PQR} telle que $[[\varphi]] = [[\varphi']]$ (en d'autres termes, les arbres PQR généralisent les séquences de groupe d'opération permutables).

$\pi = [G_1, G_2, \ldots, G_k]$

(a) Séquence de groupes permutables



(b) Arbre PQR

FIGURE 7.4 – Séquence de groupes permutables et arbre PQR correspondant

Donc $L_{PQR} \leq_e L_{POG}$, et par le théorème 7.1.3.3, on a aussi la conclusion. □

7.3 Résultats de compacité

Dans cette section nous justifions les résultats de compacité de la figure 7.1b. Rappelons que le théoreme 7.2.1.2 donne que les langages complets (Les langages-ensemble et L_{MDD}, une fois restreint à L^*, ont une expressivité équivalente à L_{1M}, et donc une expressivité équivalente entre eux. On peut donc comparer leurs compacité.

Avant toute chose, il faut remarquer que le langage à une machine est au moins aussi compact que tout autre langage aussi expressif.

Théorème 7.3.0.1. *Soit L un langage de représentation d'ensembles de séquences et L^* son fragment restreint :*
$L_{1M} \leq_s^{\mathbb{T}} L^*$

Démonstration. Le résultat suit de la définition de la taille d'une formule de L_{1M} : $\forall \varphi \in \Phi_{1M}, |\varphi| \leq |N|^2 + 3|N|$. □

Par conséquent, si la représentation d'une instance par un langage peut exploser, ce dernier n'est pas aussi compact que le langage à une machine.

Théorème 7.3.0.2. *Soit L un langage de représentation d'ensembles de séquences, s'il existe un langage L' pour lequel $L^* \not\leq_s L'^*$, alors $L^* \not\leq_s L_{1M}$.*

Démonstration. On a $\forall P \in \mathbb{R}[X], \exists \varphi' \in L'^*, \forall \varphi \in L^*, [[\varphi']] = [[\varphi]] \implies |\varphi| \geq P(|\varphi'|)$. Donc puisque $\exists \varphi'' \in L_{1M}, [[\varphi'']] = [[\varphi]]$, on a aussi $\forall \varphi \in L^*, [[\varphi'']] = [[\varphi]] \implies |\varphi| \geq P(|\varphi''|)$ par la définition de la taille d'une formule de L_{1M}. □

7.3.1 Fermeture des langages étudiés

Nous montrons dans cette section la fermeture pour la concaténation et la projection des langages L_{SEQ}, L_{POG}, et L_{PO} afin de permettre l'application du théorème 7.1.2.1 (pour lequel la fermeture des langage est une hypothèse). Notons que puisque le langage L_{1M} est au moins aussi expressif que ces langages, les preuves de fermeture sont aussi valables pour les langages L^*_{SEQ}, L^*_{POG}, et L^*_{PO}.

7.3.1.1 Fermeture de L_{SEQ}

Théorème 7.3.1.1. *L_{SEQ} est fermé pour la concaténation*

Démonstration. Soit deux séquences $\varphi_1 = (\pi_1, PRD_1) \in L_{SEQ}(E_1)$ et $\varphi_2 = (\pi_2, PRD_2) \in L_{SEQ}(E_2)$, avec E_1 et E_2 des ensembles disjoints de variables de décision. L'opérateur de concaténation est défini pour les séquences. Donc on a simplement $\varphi_3 = (\pi_1.\pi_2, PRD_\infty) \in L_{SEQ}(E_1 \cup E_2)$ et $[[\varphi_3]] = \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1 \in [[\varphi_1]], \forall \check{\sigma}_2 \in [[\varphi_2]]\}$. □

Théorème 7.3.1.2. *L_{SEQ} est fermé pour la projection*

Démonstration. De façon similaire, soit $\varphi = (\pi, PRD) \in L_{SEQ}(E)$. Avec E un ensemble de variables de décision. La projection d'une séquence sur l'ensemble $F \subseteq E$ correspond à la sous séquence définie par F. On a donc $\varphi' = (\pi_{|F}, PRD_\infty) \in L_{SEQ}(F)$ et $[[\varphi']] = \{\check{\sigma}_{|E}, \forall \check{\sigma} \in [[\varphi]]\}$ □

7.3.1.2 Fermeture de L_{POG}

Théorème 7.3.1.3. *L_{POG} est fermé pour la concaténation*

Démonstration. Soit deux séquences de groupes $\varphi_1 = (\pi_1, PRD_1) \in L_{POG}(E_1)$ et $\varphi_2 = (\pi_2, PRD_2) \in L_{POG}(E_2)$. Avec E_1 et E_2 des ensembles disjoints de variables de décision.
Soit $\varphi_3 = ([G_1^{\pi_1}, \ldots, G_k^{\pi_1}, G_1^{\pi_2}, \ldots, G_l^{\pi_2}], PRD_\infty)$. $\varphi_3 \in L_{POG}(E_1 \cup E_2)$.
On a d'un côté $[[\varphi_3]] \supseteq \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1, \check{\sigma}_2 \in [[\varphi_1]] \times [[\varphi_2]]\}$. En effet $\forall \check{\sigma}_1 \in [[\varphi_1]]$, on a $\forall i, j \in E_1^2, (G(i) \prec_{\varphi_3} G(j) \implies \check{\sigma}_i < \check{\sigma}_j$ et de même $\forall \check{\sigma}_2 \in [[\varphi_2]]$. De plus, $\check{\sigma}_1.\check{\sigma}_2$ vérifie $\check{\sigma}_i < \check{\sigma}_j, \forall (i,j) \in E_1 \times E_2$. Donc $\check{\sigma}_1.\check{\sigma}_2 \in [[\varphi_3]]$.
De l'autre côté, soit $\check{\sigma}_3 \in [[\varphi_3]]$. $\forall i, j \in E_1 \times E_2, \check{\sigma}_{3,i} < \check{\sigma}_{3,j}$. On sait donc $\exists \check{\sigma}_1 \in D_{E_1}, \check{\sigma}_2 \in D_{E_2}, \check{\sigma}_1.\check{\sigma}_2 = \check{\sigma}_3$. Supposons $\check{\sigma}_1 \notin [[\varphi_1]]$, alors $\exists i, j \in E_1, G(i) \prec G(j)$ et pourtant $\check{\sigma}_{1,i} > \check{\sigma}_{1,j}$. Mais alors on a aussi $\check{\sigma}_{3,i} > \check{\sigma}_{3,j}$, donc $\check{\sigma}_3 \notin [[\varphi_3]]$. On obtient le même résultat si on suppose $\check{\sigma}_2 \notin [[\varphi_2]]$. Par conséquent, on a aussi $[[\varphi_3]] \subseteq \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1, \check{\sigma}_2 \in [[\varphi_1]] \times [[\varphi_2]]\}$. □

Théorème 7.3.1.4. *L_{POG} est fermé pour la projection*

Démonstration. Soit $\varphi = (\pi, PRD) \in L_{POG}(E)$. Avec E un ensemble de variables de décision.
Soit $\varphi' = ([G_1^\pi \cap F, \ldots, G_k^\pi \cap F], PRD_\infty)$.
D'un coté, $[[\varphi']] \supseteq \{\check{\sigma}_{|F}, \forall \check{\sigma} \in [[\varphi]]\}$: soit $\check{\sigma} \in [[\varphi]]$, en particulier on a $\forall i, j \in F, (G(i) \prec G(j) \implies \check{\sigma}_i < \check{\sigma}_j$, donc $\check{\sigma}_{|F} \in [[\varphi']]$.
De l'autre côté, soit une séquence $\check{\sigma}'$ telle que $\check{\sigma}' \in [[\varphi]]'$. On peut facilement construire une "sur-séquence" $\check{\sigma}$ telle que par exemple

$$\begin{cases} \check{\sigma}_i = \check{\sigma}'_i + \sum\limits_{G \prec G^\varphi(i)} |G \setminus F| & \forall i \in F \\ \sum\limits_{G \prec G^\varphi(i)} |G| \leq \check{\sigma}_i \leq \sum\limits_{G \preceq G^\varphi(i)} |G| & \forall i \notin F \\ \check{\sigma}_i \neq \check{\sigma}_j & \forall i, j \in E^2 \end{cases}$$

On a bien $\check{\sigma} \in [[\varphi]]$ et $\check{\sigma}_{|F} = \check{\sigma}'$. Donc $[[\varphi']] \subseteq \{\check{\sigma}_{|F}, \forall \check{\sigma} \in [[\varphi]]\}$. □

7.3.1.3 Fermeture de L_{PO}

Théorème 7.3.1.5. *L_{PO} est fermé pour la concaténation*

Démonstration. Soit deux séquences $\varphi_1 = (\pi_1, PRD_1) \in L_{PO}(E_1)$ et $\varphi_2 = (\pi_2, PRD_2) \in L_{PO}(E_2)$. Avec E_1 et E_2 des ensembles disjoints de variables de décision.
Soit $\varphi_3 = (\pi_1 \cup \pi_2 \cup \{(i,j), \forall (i,j) \in E_1 \times E_2\}, PRD_\infty)$.
$\varphi_3 \in L_{PO}(E_1 \cup E_2)$. On a d'un côté $[[\varphi_3]] \supseteq \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1, \check{\sigma}_2 \in [[\varphi_1]] \times [[\varphi_2]]\}$. En effet, soit $\check{\sigma}_1 \in [[\varphi_1]]$, alors $\forall (i,j) \in (E_1^2) \cap \pi_1, \check{\sigma}_{3,i} < \check{\sigma}_{3,j}$, soit aussi $\check{\sigma}_2 \in [[\varphi_2]]$, alors $\forall (i,j) \in (E_2^2) \cap \pi_2, \check{\sigma}_{3,i} < \check{\sigma}_{3,j}$, et enfin soit $\check{\sigma}_3 = \check{\sigma}_1.\check{\sigma}_2$ par définition de la concaténation, $\forall (i,j) \in (E_2 \times E_2), \check{\sigma}_{3,i} < \check{\sigma}_{3,j}$. Donc $\check{\sigma}_3 \in [[\varphi_3]]$.
De l'autre côté, soit $\check{\sigma}_3 \in [[\varphi_3]]$. En particulier $\forall i, j \in E_1 \times E_2, \check{\sigma}_{3,i} < \check{\sigma}_{3,j}$. On sait

donc $\exists \check{\sigma}_1 \in D_{E_1}, \check{\sigma}_2 \in D_{E_2}, \check{\sigma}_1.\check{\sigma}_2 = \check{\sigma}_3$. Supposons $\check{\sigma}_1 \notin [[\varphi_1]]$, alors $\exists i, j \in \pi_1, \check{\sigma}_{1,i} > \check{\sigma}_{1,j}$. Mais alors on a aussi $\check{\sigma}_{3,i} > \check{\sigma}_{3,j}$, donc $\check{\sigma}_3 \notin [[\varphi_3]]$. On obtient le même résultat si on suppose $\check{\sigma}_2 \notin [[\varphi_2]]$. Par conséquent, on a aussi $[[\varphi_3]] \subseteq \{\check{\sigma}_1.\check{\sigma}_2, \forall \check{\sigma}_1, \check{\sigma}_2 \in [[\varphi_1]] \times [[\varphi_2]]\}$. □

Théorème 7.3.1.6. *L_{PO} est fermé pour la projection*

Démonstration. Soit $\varphi = (\pi, PRD) \in L_{PO}(E)$. Avec E un ensemble de variables de décision. Soit π^* la fermeture transitive de π. Remarquons d'abord que $[[(\pi, PRD]]_{PO} = [[(\pi^*, PRD]]_{PO}$.
Soit $\varphi' = (\pi', PRD_\infty) \in L_{PO}(F)$. Avec $\pi' = \pi^* \cap \{(i,j), i \in F \vee j \in F\}$
D'un coté, $[[\varphi']] \supseteq \{\check{\sigma}_{|F}, \forall \check{\sigma} \in [[\varphi]]\}$: soit $\check{\sigma} \in [[\varphi]]$, alors $\forall (i,j) \in \pi^*, \check{\sigma}_i < \check{\sigma}_j$. Par conséquent, $\forall (i,j) \in \pi', \check{\sigma}_{|F,i} < \check{\sigma}_{|F,j} : \check{\sigma}_{|F} \in [[\varphi']]$.
De l'autre côté, soit une séquence $\check{\sigma}'$ telle que $\check{\sigma}' \in [[\varphi']]$. L'ensemble de contraintes de précédence constitué par $\pi \cup \{(i,j), \check{\sigma}'_i < \check{\sigma}'_j\}$ est sans circuit car π est sans circuit. Nous avons vu à l'occasion du théorème 4.2.1.2 qu'il est toujours possible d'extraire une séquence admissible d'un ensemble de contraintes de précédence sans circuit. On peut donc construire une séquence $\check{\sigma} \in [[\varphi]]$ avec $\check{\sigma}_{|F} = \check{\sigma}'$. Donc $[[\varphi']] \subseteq \{\check{\sigma}_{|F}, \forall \check{\sigma} \in [[\varphi]]\}$. □

7.3.2 Compacité des langages L_{SSEQ}, L_{SPOG}, L_{SPO} et L_{SAOPO}

Nous montrons d'abord que le langage à une machine est strictement plus compact que le langage des ensembles de "And/Or" ordres partiels.

Théorème 7.3.2.1. $L_{1M} <_s^{\mathbb{T}} L^*_{SAOPO}$

Démonstration. D'une part, $L_{1M} \leq_s^{\mathbb{T}} L^*_{SAOPO}$ tient par le théorème 7.3.0.1. D'autre part nous verrons par la suite qu'il existe des instances pour lesquelles aucune représentation compacte équivalente n'existe dans L_{SAOPO} (voir par exemple le théoreme 7.3.3.1). On obtient le résultat par le théorème 7.3.0.2. □

Nous pouvons établir que L^*_{SAOPO} est strictement plus compact que L^*_{SPO}, qui est strictement plus compact que L^*_{SPOG}, lui même strictement plus compact que L^*_{SSEQ}.

Théorème 7.3.2.2. $L^*_{SPO} <_s L^*_{SPOG} <_s L^*_{SSEQ}$

Démonstration. Comme nous l'avons vu pour les résultats d'expressivité, puisque ces langages sont des généralisations successives, il est facile de trouver pour chaque formule, une formule équivalente de taille équivalente dans un langage généralisant, donc on a $L_{SPO} \leq_s L_{SPOG} \leq_s L_{SSEQ}$. (résultat qui est aussi valable pour les langages restreints L^* par le théorème 7.1.3.3). D'autre part, nous avons montré résultats d'expressivité $L^*_{SPO} \not\geq_e L^*_{SPOG}$, et $L^*_{SPOG} \not\geq_e L^*_{SSEQ}$. De plus, ces langages sont fermés pour la concaténation et la projection (voir section 7.3.1). Par conséquent, le théorème 7.1.2.1 donne les résultats $L^*_{SPO} \not\geq_s L^*_{SPOG}$, et $L^*_{SPOG} \not\geq_s L^*_{SSEQ}$. □

Nous avons pu utiliser le théorème 7.1.2.1 précédemment puisque nous avons montré la fermeture des langages considérés. Pour certaines preuves qui suivent, il semblerait que le théorème s'applique, mais puisque nous n'avons pas montré que les langages impliqués sont fermés, nous utilisons une autre preuve [6]. Par exemple pour L^*_{SAOPO} :

Théorème 7.3.2.3. $L^*_{SAOPO} <_s L^*_{SPO}$

Démonstration. D'une part, comme précédemment, $L^*_{SAOPO} \leq_s L^*_{SPO}$, car les ordres partiels "And/Or" généralisent les ordres partiels.
D'autre part, montrons $L^*_{SAOPO} \not\geq_s L^*_{SPO}$: Soit l'instance suivante de $n = 3k$ tâches (avec $\mathbb{1}(x) = 1$ si $x = \top$ et 0 sinon, et % l'opérateur modulo) :

$$\psi_4 = \begin{cases} p_i = 1 & \forall i \\ r_i = 3 * (i/3) + \mathbb{1}_{i\%3=0} & \forall i \\ d_i = 3 * (i/3 + 1) & \forall i \\ E = \emptyset \end{cases}$$

On remarque que lorsque $k = 1$, $\psi_4 = (\emptyset, PRD')$ correspond à l'instance $\psi_1 = (\emptyset, PRD)$ (qui représente les séquences $\{ABC, BCA, ACB\}$). Nous avons vu dans le théoreme 7.2.2.2 que l'ordre partiel "And/Or" $\varphi = \{((a, b), c)\}, PRD)$ correspond à ψ_1, mais qu'il n'y a pas de représentation équivalente par un ordre partiel (il existe une formule équivalente dans L_{SPO} puisque le langage-ensemble est complet, mais celle-ci contient au moins deux ordres partiels). L'instance ψ_4 est répétée pour chacun des k triplets (X, Y, Z) de tâche $((A, B, C), (D, E, F), \dots)$. Les séquences correspondant aux ordonnancements admissibles sont tous les choix d'une sous-séquence parmi $\{XYZ, YZX, XZY\}$ pour chaque triplet. On remarque facilement que l'ordre partiel "And/Or" $\varphi = (\{((a, b), c), ((d, e), f), \dots ((n-2, n-1), n)\}, PRD')$ vérifie $[[\psi_4]]\mathbb{T}[[\varphi]]$, avec $|\varphi| \sim O(n)$.
Toutefois, un unique ordre partiel ne peut représenter deux séquences correspondant à deux choix de sous-séquences sans représenter aussi des séquences inadmissibles. Donc chaque nouvelle instance concaténée multiplie par deux le nombre minimal d'ordres partiels nécessaires à couvrir toutes les séquences représentées. Au final un minimum de 2^k ordres partiels sont nécessaires. D'où le résultat.

□

7.3.3 Compacité des ensembles d'arbres PQR

Le langage-ensemble des arbres PQR n'est pas moins compact que le langage ensemble des "And/Or" ordres partiels.

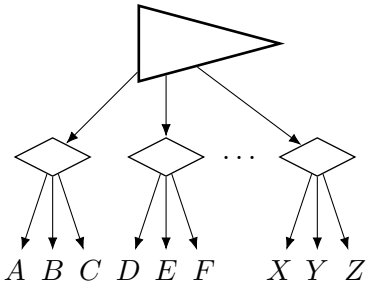
Théorème 7.3.3.1. $L^*_{SPQR} \not\geq_s L^*_{SAOPO}$

Démonstration. Considérons l'instance du problème à une machine de la figure 7.5 avec $n = 3k$ tâches. On voit que par exemple pour $k = 1$ et les tâches A, B, C, les

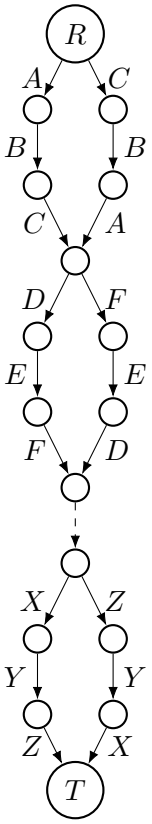
6. Ces preuves peuvent illustrer l'idée reprise dans le théorème 7.1.2.1

$$\psi_5 = \begin{cases} p_i = 1 & \forall i \\ r_i = 3 * (i/3) + \mathbb{1}_{i\%3=0} & \forall i \\ d_i = 3 * (i/3 + 1) - \mathbb{1}_{i\%3=0} & \forall i \\ E = \emptyset \end{cases}$$

(a) Instance de problème à une machine



(b) Arbre PQR



(c) MDD

FIGURE 7.5 – Trois représentations équivalentes d'un ensemble de séquences (modulo translation)

ordonnancements admissibles de l'instance correspondent aux séquences ABC et CBA (dans ce cas, l'instance correspond à ψ_3 (7.1)). Lorsque $k > 1$, l'instance est répétée pour chacun des k triplets (X, Y, Z) de tâche $((A, B, C), (D, E, F), \dots)$. Les séquences correspondants aux ordonnancements admissibles sont donc tous les choix possible de la séquence XYZ ou la séquence inverse ZYX pour chaque triplets. L'arbre PQR de la figure 7.5 représente ces séquences et est de taille $\sim O(n)$
Maintenant considérons $\varphi = (\Pi, PRD) \in L_{SAOPO}$. Nous avons vu à l'occasion du théoreme 7.2.2.4 que pour chaque triplet (X, Y, Z), une seule sous-formule $\pi \in \Pi$ ne peut représenter les deux séquences sans en représenter d'autres. Donc chaque sous formule ne peut représenter plus d'une des séquences. On conclut que Π doit contenir un minimum de 2^k sous-formules. D'où le résultat. □

Par le corollaire 7.1.1.2, on a aussi $L^*_{SPQR} \not\geq_s L^*_{SPO}$ et $L^*_{SPQR} \not\geq_s L^*_{SPOG}$.

Le langage ensemble des arbres PQR n'est pas plus compact que le langage ensemble des ordres partiels.

Théorème 7.3.3.2. $L^*_{SPQR} \not\leq_s L^*_{SPO}$

Démonstration. Soit l'ordre partiel pour $n = 3k$ tâches constitué de k triplet de tâches $T_i = (X_i, Y_i, Z_i) : \varphi = (\{(X_i, Y_i), \forall i\} \cup \{(a, b), \forall a \in T_i, b \in T_{i+1}, \forall i \in 1 \dots k-1\}, PRD_\infty)$. Lorsque $k = 1$, cela correspond à l'ordre partiel du théoreme 7.2.3.1, avec les séquences admissibles $\{ABC, ACB, CAB\}$. Nous avons montré qu'il n'existe pas d'arbre PQR correspondant. Lorsque $k > 1$, les séquences représentées sont tous les choix parmi les trois sous-séquences pour chaque triplet.
Soit $\varphi' = (\Pi, PRD) \in L_{SPQR}$. Pour chaque sous-formule $\pi \in \Pi$, le théorème 7.1.4.2 est vérifié pour chaque tâche de chaque triplet successifs, un nœud R est donc le plus petit sous arbre commun à chaque triplet. De plus, puisque pour chaque triplet, Z_i peut être à toutes les positions, les tâches du triplet sont dans le même sous arbre de R. Or, puisqu'un sous arbre est un arbre PQR, il ne peut représenter seulement les sous-séquences souhaitées.
Par conséquent un minimum de 2^k arbres PQR doivent constituer Π . □

Par le corollaire 7.1.1.1, on a aussi $L^*_{SPQR} \not\leq_s L^*_{SAOPO}$ et $L^*_{SPQR} \not\leq^{\mathbb{T}}_s L_{1M}$.

Enfin, le langage ensemble des arbres PQR est tout de même plus compact que le langage ensemble des séquences de groupes.

Théorème 7.3.3.3. $L^*_{SPQR} \leq_s L^*_{SPOG}$

Démonstration. Nous avons vu que dans la figure 7.4 comment, pour chaque séquence de groupe, on trouve un arbre PQR équivalent de taille équivalente, on obtient donc la conclusion par construction. □

Au final, on déduit des résultats présentés les relations qui figurent dans la carte de compilation : $L^*_{SPQR} >^{\mathbb{T}}_s L_{1M}$ (par le théoreme 7.3.0.1), $L^*_{SPQR} \lesseqgtr_s L^*_{SAOPO}$, $L^*_{SPQR} \lesseqgtr_s L^*_{SPO}$, et $L^*_{SPQR} <_s L^*_{SPOG}$.

7.3.4 Compacité des diagrammes de décision multivalués

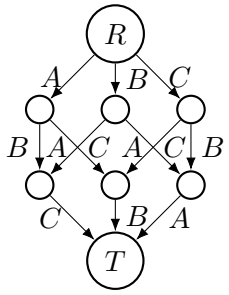
Le langage des MDD n'est pas plus compact que le langage des ensembles de séquences de groupes.

Théorème 7.3.4.1. $L^*_{MDD} \not\leq^{\mathcal{T}}_s L^*_{SPOG}$

Démonstration. Considérons une instance du problème à une machine non contrainte (avec par exemple $d_i = \infty, \forall i$). La séquence de groupes totalement permutables qui contient toutes les tâches dans un seul groupe correspond à cette instance et est de taille 3. La figure 7.6 montre un exemple pour $n = 3$. Le MDD le plus compact qui représente toutes les permutations est l'hypercube de dimension n, puisque pour chacun de ses nœuds, l'ensemble des tâches des chemins y menant est différent, donc les chemins sortant de deux nœuds différents ne peuvent être identiques (car tous les chemins vérifient `AllDifferent`), et par conséquent aucun nœud ne peut être fusionné, le MDD est réduit, et de taille minimale. La taille de ce MDD correspond donc au nombre de sommets (nœuds) et d'arêtes (arcs) d'un hypercube de taille n : $2^{n-1}(n+2)$. □

$$\pi = [\{A, B, C\}]$$

(a) Séquence de groupes pleinement permutables



(b) "Hypercube" MDD

FIGURE 7.6 – MDD de taille minimale équivalent à une séquence de groupes totalement permutables ($n = 3$)

Par le corollaire 7.1.1.1, on a aussi $L^*_{MDD} \not\leq^{\mathcal{T}}_s L^*_{SPO}$, $L^*_{MDD} \not\leq^{\mathcal{T}}_s L^*_{SAOPO}$, $L^*_{MDD} \not\leq^{\mathcal{T}}_s L^*_{SPQR}$ et $L^*_{MDD} \not\leq^{\mathbb{T}\circ\mathcal{T}}_s L_{1M}$.

Le langage des MDDs n'est pas moins compact que le langage ensemble des "And/Or" ordres partiels.

Théorème 7.3.4.2. $L^*_{MDD} \not\geq^{\mathcal{T}}_s L^*_{SAOPO}$

Démonstration. Nous avons vu à l'occasion du théoreme 7.3.3.1 qu'aucune formule de L^*_{SAOPO} ne peut représenter de façon compacte l'instance ψ_5 de la figure 7.5. On peut voir que le MDD représenté dans la même figure correspond à cette instance et sa taille est en $O(|N|)$). D'où le résultat. □

Par le corollaire 7.1.1.2, on a aussi $L^*_{MDD} \not\geq^{\mathcal{T}}_s L^*_{SPO}$, $L^*_{MDD} \not\geq^{\mathcal{T}}_s L^*_{SPOG}$ et $L^*_{MDD} \not\geq^{\mathcal{T}}_s L^*_{SSEQ}$.

Le langage des MDDs n'est pas moins compact que le langage ensemble des arbres PQR.

Théorème 7.3.4.3. $L^*_{MDD} \not\geq^{\mathcal{T}}_s L^*_{SPQR}$

Démonstration. On sait $L^*_{SPQR} \leq_s L^*_{POG}$ et $L^*_{MDD} \not\leq^{\mathcal{T}}_s L^*_{SPOG}$, donc par le corollaire 7.1.1.1 on a le résultat. □

Enfin, le langage des MDD est tout de même plus compact que le langage-ensemble des séquences .

Théorème 7.3.4.4. $L^*_{MDD} \leq^{\mathcal{T}}_s L^*_{SSEQ}$

Démonstration. Il est facile de construire, pour chaque ensemble de séquences Π, un MDD équivalent π dont la taille est au pire équivalente en ajoutant, pour chaque séquence $\breve{\sigma} \in \Pi$ un chemin de la racine au terminal correspondant à l'affectation $position(\breve{\sigma})$. □

Au final, on déduit des résultats présentés les relations qui figurent dans la carte de compilation : $L^*_{MDD} >^{\mathbb{T}\circ\mathcal{T}}_s L_{1M}$ (par le théoreme 7.3.0.1), $L^*_{MDD} \leqslant^{\mathcal{T}}_s L^*_{SAOPO}$, $L^*_{MDD} \leqslant^{\mathcal{T}}_s L^*_{SPO}$, $L^*_{MDD} \leqslant^{\mathcal{T}}_s L^*_{SPOG}$, et $L^*_{MDD} <^{\mathcal{T}}_s L^*_{SSEQ}$.

7.4 Résultats de support des requêtes

Dans cette section, nous justifions les résultats de complexité des requêtes présentés dans la table 7.1. Rappelons qu'en vertu du théorème 7.1.3.4, montrer qu'un langage L supporte une requête suffit à montrer que le langage L^* la supporte aussi pour nos requêtes.

7.4.1 Support de CO et de MX

Théorème 7.4.1.1. *L_{SEQ}, L_{AOPO},L_{PO}, L_{POG}, L_{PQR}, L_{MDD}, ainsi que les langages-ensemble basés sur ces langages supportent CO et MX.*
Seul L_{1M} ne supportent pas CO ni MX.

Démonstration. Il est connu que L_{1M} ne supporte par CO ni MX [Lenstra et al., 1977]. On peut voir que trivialement, L_{SEQ} supporte CO et MX (la séquence représentée est une solution admissible).
De plus, Möhring et al. [2004] donnent un algorithme linéaire pour extraire une séquence admissible d'un ordre partiel "And/Or" s'il en existe une. Par conséquent, puisque les ordres partiels et les séquences de groupe en sont des cas particuliers, L_{AOPO},L_{PO}, et L_{POG} supportent CO et MX.
L_{PQR} supporte aussi CO et MX, puisque une solution admissible peut être lue en considérant les feuilles de l'arbre (sa frontière [Baptiste and Favrel, 1993]). Enfin, une solution admissible, si elle existe, peut être facilement extraite d'un MDD en considérant un chemin de la racine au terminal. L_{MDD} supporte donc ces requêtes.
Si L supporte CO, alors SL supporte CO (il suffit de lancer l'algorithme polynomial A sur les sous-formules du langage-ensemble, et retourner $\top$ si une sous-formule représente une solution admissible). Le même argument s'applique à MX. □

7.4.2 Support de CD_0

Théorème 7.4.2.1. *L_{1M}, L_{AOPO},L_{PO}, L_{POG}, L_{PQR}, L_{MDD}, ainsi que les langages-ensemble basés sur ces langages supportent CD_0.*

Démonstration. L_{1M} supporte trivialement CD_0 qui correspond à une simple modification des données temporelles.
Pour les langages séquentiels, la requête correspond à ne conserver que les séquences où la tâche i que l'on veut démarrer est en première position. Pour les langages L_{PO} et L_{AOPO}, il suffit d'ajouter les contraintes de précédence supplémentaires $\{(i,j)\forall j \neq i \in N\}$.

Pour L_{POG}, il faut d'abord vérifier que i est dans le groupe G_1 de la formule $\pi = [G_1, G_2, \dots]$ considérée. Si c'est le cas, la séquence de groupes $\pi' = [\{i\}, G_1 \setminus \{i\}, G_2, \dots]$ est satisfaisante pour la requête.
De façon similaire, pour L_{MDD}, on peut vérifier facilement si i fait partie des labels des arcs sortants du nœud racine R, si c'est le cas, une formule satisfaisante est obtenue en supprimant les arcs correspondant aux autres options sur la première couche (ainsi que les nœuds et les arcs se retrouvant orphelins).

Si L supporte CD_0, alors SL supporte CD_0 (il suffit de lancer l'algorithme polynomial A sur les sous-formules du langage-ensemble, la nouvelle formule est l'ensemble des sous-formules transformées par A). □

7.4.3 Support de $R\uparrow$

Théorème 7.4.3.1. *L_{SPOG} et L_{PO} ne supportent pas $R\uparrow$*

Démonstration. Soit une formule $\varphi = (\{[\{1, \dots, |N|\}]\}, PRD) \in L_{SPOG}$ représentant toutes les permutations de $|N|$ tâches avec les données temporelles $PRD = (p_i = 1, r_i = 0, d_i = |N|, \forall i)$. La formule est bien cohérente : tous les ordonnancements semi-actifs sont admissibles.
Supposons ensuite que la tâche 1 soit en retard, sa nouvelle date de disponibilité est $r'_1 = |N|/2 - 1$. Il faut modifier φ pour assurer la cohérence avec les nouvelles données temporelles. On remarque qu'en fait, ne doivent être conservées que les séquences représentées par φ telles que $\check{\sigma}_1 \geq n/2$.
Nous montrons que tout SPOG φ' tel que $[[\varphi']] = \{\check{\sigma}, \check{\sigma} \models PRD', \forall \check{\sigma} \in [[\varphi]]\}$ est de taille $|\varphi'|$ non polynomiale en n.
En effet, soit un tel $\varphi' = (\Pi, PRD')$. Π doit couvrir toutes les séquences pertinentes. En particulier, on a

$$f^{\Pi}(\check{\sigma}_{|A}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B}) = \top, \forall A \subset N$$
$$\text{avec}$$
$$|A| = n/2$$
$$\{A, B, \{1\}\} \text{ une partition de } N$$
$$\check{\sigma}_1 = n/2$$

C'est à dire que pour chaque séquence de cette famille particulière de séquences (issue de toutes les sélection possibles des $n/2$ tâches de A), il existe une sous formule $\pi \in \Pi$ qui la couvre i.e $f^{\pi}_{L_{POG}}(\check{\sigma}_{|A}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B}) = \top$.

Nous montrons par l'absurde que deux séquences de cette famille issues de deux sélections **distinctes** A et A' ne peuvent être couvertes par la même sous-formule π.
Supposons $\exists \pi \in \Pi, f^{\pi}(\check{\sigma}_{|A}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B}) = f^{\pi}(\check{\sigma}_{|A'}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B'}) = \top$.
On remarque que puisque $A \neq A'$, $\exists t \in A \cap B'$. De plus $f^{\pi}(\check{\sigma}_{|A}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B}) = \top$ et

$\check{\sigma}_t < \check{\sigma}_1$ dans cette séquence. De même, $f^{\pi}(\check{\sigma}_{|A'}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B'}) = \top$ et $\check{\sigma}_t > \check{\sigma}_1$ dans $\check{\sigma}_{|A'}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B'}$
Par le théorème 7.1.4.1, π vérifie $G(t) = G(1)$, et donc $f^{\Pi}(\check{\sigma}_{|A\setminus t}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|\{t\}}.\check{\sigma}_{|B}) = f^{\pi}(\check{\sigma}_{|A\setminus t}.\check{\sigma}_{|\{t\}}.\check{\sigma}_{|\{1\}}.\check{\sigma}_{|B}) = \top$, mais dans cette séquence, $\check{\sigma}_1 = n/2 - 1$, or Π ne devrait pas couvrir de séquences dans laquelle $\check{\sigma}_1 < n/2$. C'est donc une contradiction. Puisque Π doit contenir une sous-formule π distincte pour chaque séquence issue d'une sélection $A \subset N$, et qu'il existe $\binom{|N|}{|N|/2}$ telles sélections. $|\Pi| \geq \binom{|N|}{|N|/2}$. Par conséquent, L_{SPOG} ne supporte pas $R\uparrow$.

Nous pouvons montrer de façon similaire que L_{SPO} ne supporte pas $R\uparrow$. En effet si un ordre partiel π couvre deux séquences issues de sélections distinctes, il couvre une séquence dans laquelle $\check{\sigma}_t < \check{\sigma}_1$ et une dans laquelle $\check{\sigma}_t > \check{\sigma}_1$, donc il n'y a pas de contraintes de précédence liant t et 1. On pourrait donc les permuter et π couvrirait aussi une séquence interdite.

□

Théorème 7.4.3.2. *L_{1M}, L_{SEQ}, et L_{SSEQ} supportent $R\uparrow$.*

Démonstration. L_{1M} supporte trivialement $R\uparrow$ par modification des données temporelles.
Il est aussi facile de montrer que L_{SEQ} supporte $R\uparrow$, puisqu'il suffit de vérifier que la séquence représentée est cohérente avec les nouvelles données temporelles (c'est le cas ssi l'ordonnancement semi-actif associé est toujours admissible).
De la même façon que pour CD_0, si un langage L supporte $R\uparrow$, le langage SL la supporte. □

7.4.4 Support de $E\uparrow$

Théorème 7.4.4.1. *L_{1M}, L_{PO}, L_{AOPO}, L_{SEQ}, L_{MDD}, et les langages-ensemble correspondant supportent $E\uparrow$.*

Démonstration. L_{1M}, L_{PO} et L_{AOPO}, supportent trivialement $E\uparrow$ par l'ajout direct d'une contrainte de précédence. L_{SEQ} supporte aussi $E\uparrow$ puisqu'il suffit de vérifier que la contrainte est respectée dans la séquence représentée.
Enfin, il est connu [Cire and van Hoeve, 2012] que les MDDs supportent l'ajout de contraintes de précédence.
Comme précédemment, si un langage supporte $E\uparrow$, le langage-ensemble correspondant la supporte. □

Toutefois de façon similaire à la preuve de non support de $R\uparrow$ de L_{SPOG} dans la section 7.4.3, nous pouvons montrer le résultat suivant :

Théorème 7.4.4.2. *L_{POG} ne supporte pas $E\uparrow$*

Démonstration. En effet, tentons d'ajouter une contrainte de précédence (i, j) à la séquence de groupe totalement permutable. Considérons la famille de séquences $\check{\sigma}_{|A}.\check{\sigma}_{|\{i\}}.\check{\sigma}_{|\{j\}}.\check{\sigma}_{|B}$ pour des sélections $A \subset N$ de taille $n/2$. Si $\pi \in L_{POG}$ peut couvrir

deux sélections distinctes A et A' on a $\exists t \in A \cap B'$, et donc par le théorème 7.1.4.1, $G(i) = G(t) = G(j)$, et donc π couvre aussi la séquence interdite $\check{\sigma}_{|A}.\check{\sigma}_{|\{j\}}.\check{\sigma}_{|\{i\}}.\check{\sigma}_{|B}$. On conclut comme précédemment. □

7.4.5 Support de BC/WC

Théorème 7.4.5.1. *Les résultats de support des requêtes BC/WC sont comme indiqué dans la figure 7.1.*

Démonstration. Puisque L_{1M} ne supporte pas CO, il ne peut supporter BC/WC. Trivialement, L_{SEQ} supporte $BC(F)$ et $WC(F)$ ssi F évalue une séquence en temps polynomial. Aloulou et al. [2004] montrent que la maximisation de toute fonction $f_{MAX} = \max_{i \in N} f_i(C_i)$ avec f_i une fonction non-décroissante et C_i la date de fin de la tâche i peut être déterminée en temps polynomial étant donné un ensemble de contraintes de précédence. En particulier, L_{PO} supporte $WC(C_{MAX})$ et $WC(L_{MAX})$. De même pour L_{POG}, un cas particulier de L_{PO}.

Il est aussi connu que la minimisation du makespan par exemple est facile étant donné un ensemble de contraintes de précédence et des dates de disponibilité [Aloulou et al., 2004]. Donc L_{PO} et L_{POG} supportent $BC(C_{MAX})$.

Bergman et al. [2016a] montrent comment il est possible de calculer, pour chaque arc a du nœud u à v d'un MDD (V, A), sa date de fin au plus tôt $\underline{C_a}$ récursivement à partir des arcs arcs entrants dans u (noté par $In(u)$), par l'équation :

$$\underline{C_a} = \max(r_{l_a}, \min_{b \in In(u)} \underline{C_b}) + p_{l_a}$$

On peut calculer similairement la date de fin au plus tard d'un arc $\overline{C_a}$. Par conséquent, $\min_{a \in In(T)} \underline{C_a}$ donne la valeur minimale du makespan, $\max_{a \in In(T)} \underline{C_a}$ donne sa valeur maximale, et $\max_{a \in A} \underline{C_a} - d_{l_a}$ donne la valeur du plus grand retard maximum. En résumé, L_{MDD} supporte donc les requêtes $BC(C_{MAX})$, $WC(C_{MAX})$ et $WC(L_{MAX})$. □

7.5 Discussion

Nous avons démontré dans ce chapitre les résultats résumés dans les figures 7.1a, 7.1b et la table 7.1. Malheureusement, le seul langage étudié qui supporte les requêtes d'intérêt que nous avons identifié est le langage trivial d'ensemble de séquences L_{SSEQ}. Donc aucun langage ne nous satisfait pleinement pour la représentation de solutions du problème à une machine.
De plus, les résultats de compacité montrent que même modulo notre translation, aucun des langages étudiés ne permet de représenter de façon compacte les instances du problème à une machine.
Néanmoins, les quelques résultats présentés permettent déjà la comparaison des différents langages. Le résultat le plus intéressant semble être que les ensembles

d'ordres partiels supportent strictement plus de requêtes que les ensembles de séquences de groupes d'opérations permutables, alors qu'ils leurs sont plus compacts ! Pour les requêtes considérées, on peut donc déduire une forme de dominance de L_{SPO} sur L_{SPOG}. Ces résultats de dominance pourraient toutefois être mitigés par la considération de plus de requêtes d'intérêt, qui pourraient révéler une incomparabilité dans le support des requêtes.
Les résultats non-démontrés de support des requêtes pourraient aussi se révéler très intéressants. En particulier, les ensembles d'ordres partiels "And/Or" pourraient à leur tour dominer les ensembles d'ordres partiels. Ou encore les diagrammes de décision multivalués pourraient supporter toutes les requêtes d'intérêt.
Les nombreux résultats d'incomparabilité que présente la carte de compilation suggèrent les différent types de contraintes qui peuvent caractériser des ensembles de séquences et sont certainement dus aux origines hétéroclites des langages. Recueillir et cartographier d'autres langages de représentation de séquences pourrait permettre de compléter notre compréhension des propriétés des séquences.

Les travaux présentés suggèrent surtout la quantité de travaux qu'il reste à faire pour recenser et cartographier les langages de représentation pour les problèmes d'ordonnancement. Tout d'abord, nous n'avons pas démontré de nombreux résultats de complexité pour le support des requêtes d'intérêt par les langages d'intérêt. Mais de plus, d'autres langages, et d'autres requêtes, peuvent être pertinentes. En particulier, il serait intéressant d'intégrer d'autres langages correspondants à des problèmes d'ordonnancement. Des problèmes à une machine, et possiblement des problèmes d'ordonnancement plus complexes pour lesquels un algorithme de liste permet d'obtenir un ordonnancement à partir d'une séquence.

Quatrième partie

Conclusion et perspectives

Chapitre 8

Conclusion et perspectives

8.1 Résumé des travaux présentés

Dans cette thèse, nous avons présenté des approches pour la prise en compte d'incertitudes dans des problèmes d'ordonnancement faisant usage du calcul préalable et de la représentation compacte d'ensembles de séquences.

Synthèse des contributions de la partie II

Dans la partie II, nous avons présenté une famille d'approches s'inscrivant dans le paradigme des approches proactives-réactives intégrées, dans laquelle une décision de premier niveau restreint hors-ligne l'espace des solutions en caractérisant un ensemble de séquences, et les décisions de second niveau sont prises par une politique en ligne. Les approches de cette famille se définissent par ce que nous avons appelé une "stratégie", qui décrit à la fois l'espace des restrictions possibles au premier niveau, et la politique de second niveau.
Nous avons étudié plusieurs stratégies pour une même politique de second niveau correspondant à l'heuristique FIFO (*First-in First-out*). En particulier nous avons étudié les stratégies pour lesquelles la décision de premier niveau correspond à une unique séquence (la stratégie JSEQ), à l'ensemble de toutes les séquences (la stratégie FIFO pure), à une restriction arbitraire (la stratégie complète) et à un ensemble particulier de restrictions (la stratégie GSEQ par exemple).
Les résultats expérimentaux obtenus montrent que bien que certaines stratégies en dominent d'autres en théorie, du fait de la complexité associée à la tailles des différents espaces de recherche, il n'est pas évident d'en tirer des solutions de meilleure qualité dans un temps restreint. Dans cette thèse, nous proposons l'utilisation des démarrages à chaud pour tirer parti du potentiel d'une stratégie dominante ainsi que de l'espace de recherche restreint des stratégies dominées, une approche qui semble prometteuse.

La question de l'implémentation des différentes stratégies reste essentielle pour tirer pleinement partie de leurs potentiel. Nous avons en particulier proposé et comparé plusieurs approches pour l'implémentation de la méthode GSEQ. Les résultats obtenus suggèrent que la méthode gloutonne EW-GSEQ est efficace en tant que méthode à chaud, mais l'algorithme génétique AG-GSEQ proposé est moins sensible au choix du temps de chauffe et compétitif avec la méthode gloutonne sur de nombreuses instances. Pour la stratégie complète, nous avons d'une part proposé

un premier modèle de PPC (CP-COMPL) dont les résultats laissent à désirer et, d'autre part, l'algorithme génétique AG-COMPL, dont les résultats à l'entraînement sont encourageants, mais produisent des solutions qui n'obtiennent pas de bons scores en test. De façon générale, les résultats expérimentaux confirment l'intérêt des démarrages à chaud. Ce résultat est concordant avec ceux de l'approche *"solve and robustify"* [Policella et al., 2009], pour une stratégie basée sur les ordres partiels.

Cela pose la question de comment un entraînement portant sur un ensemble restreint de scénarios peut produire une solution de bonne qualité sur un ensemble plus importants de scénarios de tests. Les résultats de cette thèse montrent que pour de nombreuses approches, malgré un nombre restreint de scénarios d'entraînement, la phase d'entraînement peut être profitable, et l'entraînement porte ses fruits : c'est notamment le cas des stratégies GSEQ. L'approche classique consistant à proposer un maximum de flexibilité pour assurer la robustesse des solutions proposées (par exemple dans Artigues et al. [2005]) n'est pas nécessairement intéressante lorsque les décisions au second niveau sont prises automatiquement par une politique comme FIFO. Une plus grande attention doit donc être portée aux caractéristiques des solutions dont la perte de généralisation est importante pour améliorer les solutions produites.

Pour pouvoir efficacement utiliser le résultat de la décision du premier niveau des stratégies, se pose aussi la question de son mode de représentation. Une question qui rejoint les travaux menés en partie III.

Synthèse des contributions de la partie III

Dans la partie III, étudions différentes structures de données pour représenter des ensembles de solutions, l'objectif étant de calculer hors-ligne une représentation d'un problème fournissant à un décideur un outil pour réagir aux aléas en ligne. Nous avons proposé une formalisation permettant la comparaison de différents types de structures de données en tant que langages de représentation d'ensembles de séquences pour un problème d'ordonnancement à une machine. Ce formalisme, issu du domaine de la compilation de connaissance, permet de comparer différents langages selon leur expressivité, leur compacité, et leur capacité à supporter des requêtes. Cette comparaison permet de guider l'utilisateur vers un choix de langage de représentation. L'utilisation de translations permet aussi de comparer des langages ne représentant pas le même type d'objets.

Nous nous sommes en particulier intéressés aux langages correspondant aux structures utilisées dans la partie II, mais aussi à d'autres langages issus de la littérature en ordonnancement comme les ordres partiels, ou les arbres PQR. Nous avons identifié un premier ensemble de requêtes d'intérêt permettant le suivi de l'exécution d'un ordonnancement et la prise en compte de certains aléas.
Afin de comparer ces langages d'expressivités différentes, nous utilisons d'un côté

une restriction aux formules correspondant à des problèmes d'ordonnancement à une machine, afin de limiter leur expressivité et, de l'autre, nous utilisons la famille des "langages-ensemble", qui permet de rendre complet tous les langages considérés. Les résultats obtenus permettent d'établir une première carte de compilation des langages étudiés.

8.2 Pistes de travaux futurs

Pour une grande partie, les travaux présentés dans cette thèse sont toujours en cours, et de nombreuses pistes se dégagent pour continuer l'exploration des sujets présentés.

Pistes issues de la partie II

Concernant la partie II, un ensemble de directions possibles pour des travaux futurs se dégagent. La première catégorie de pistes réside dans l'amélioration des implémentations des différentes stratégies étudiées. En effet, l'étude de ces stratégies ne peut a priori se faire que par l'intermédiaire d'implémentations efficaces en l'absence desquelles la comparaison relative est difficile.
Pour la stratégie JSEQ, nous avons donné des arguments soutenant la bonne performance de la méthode PPC, mais celle ci pourrait être confirmée par l'utilisation des meilleures méthodes de la littérature. Les implémentations de la stratégie GSEQ peuvent certainement aussi être améliorées, puisque l'une des meilleures méthodes actuelle est une heuristique gloutonne, mais à ce jour, aucune méthode ne semble exploiter efficacement l'espace de recherche associé. Une piste possible serait la restriction du nombre de groupes utilisés à la façon de Wu et al. [1999] dont les solutions se limitent à deux groupes. La stratégie résultante est dominée par la stratégie GSEQ, mais associée à un espace de résolution réduit.
A plus forte raison, les implémentations de la stratégie complète, notamment ceux de la méthode exacte de PPC, doivent être améliorées. En particulier il semble qu'une analyse de l'impact de la `AllDifferent`-restriction des MDD permette de construire des diagrammes de décision qui soient exacts par construction, sans qu'il soit nécessaire d'assurer cette propriété, ce qui pourrait alléger le modèle. D'autre part, bien que les degrés de relâchements étudiés pour la définition des MDDs valides n'aient pas été satisfaisants, d'autres relâchements pourraient être possibles. Une piste intéressante est celle des "*sound MDDs*" [Hadžić and Hooker, 2007] : des MDDs dans lesquels toutes les solutions non-admissibles représentées sont ignorées pour certaines requêtes.
Dans le cas robuste, les propriétés de l'algorithme de Best-Of suggèrent une possible méthode exacte pour l'ajout itératif de séquences à un ensemble restreint de séquences dont le meilleur sous-ensemble peut être déterminé en temps polynomial. Le développement d'une implémentation efficace dans le cas stochastique est aussi une piste de travail future, tout comme la question de savoir si l'adaptation de la

stratégie complète au problème du jobshop se fait aussi naturellement que pour la stratégie GSEQ ou encore pour les arbres PQR [Baptiste and Favrel, 1993].

La seconde catégorie de pistes réside dans les autres stratégies et variantes qu'il est possible d'étudier selon le paradigme proposé des méthodes à deux niveau. Les nombreux langages étudiés dans le partie III en particulier pourraient être utilisés pour définir des stratégies, ainsi que diverses restrictions ou développements autour de ces langages (par exemple les langages-ensemble). Les résultats d'expressivité des langages L_{SEQ}, L_{POG}, L_{PO}, L_{AOPO}, et L_{MDD} suggèrent que des stratégies correspondantes suivraient le même schéma de dominance, on peut alors imaginer une procédure de démarrage à chaud à plusieurs vitesses, qui utiliserait successivement les différentes stratégies pour améliorer les solutions dans des espaces de recherche imbriqués.
Une piste importante est aussi l'étude de différentes politiques de second niveau. Tout d'abord, d'autres informations comme la durée des tâches pour le 1P devrait être intégrée à l'information sur laquelle se base la politique, en plus de la composante incertaine, et de meilleurs résultats pourraient sans doutes être atteints. De manière générale, plus la politique de second niveau est capable de discriminer parmi les séquences sélectionnées au premier niveau pour en extraire une meilleure en fonction du scénario, plus la décision de premier niveau peut comporter un grand nombre de séquences sans craindre une mauvaise décision ultérieure, ce qui permet à terme une meilleure adaptation possible. Dans cette lignée, en relâchant le modèle d'information, d'autres politiques peuvent être utilisées. Par exemple on pourrait appliquer une règle de type SPT sur le problème du jobshop si l'on révélait la durée des tâches lorsqu'elles sont prêtes à être exécutées. Il serait intéressant de comparer les performances de stratégies en fonction de leurs politiques de second niveau.

Pistes issues de la partie III

Concernant la partie III, plusieurs pistes se dégagent aussi.

La première piste consiste à cataloguer et formaliser les différentes requêtes correspondant à des opérations utilisées dans la littérature en ordonnancement, ainsi que déterminer lesquelles sont des compositions d'autres requêtes. Des requêtes relatives à l'exploitation des structures par les stratégies à deux niveaux permettraient aussi un rapprochement plus étroit avec la partie II.
D'autres pistes pourraient permettre ce rapprochement. En effet nous ne nous intéressons dans la partie III qu'à l'admissibilité des solutions, et cherchons à représenter *toutes* les solutions admissibles pour un problème déterministe. Alors que dans la partie II, nous somme intéressés par le score des solutions, et l'ensemble représenté n'est pas déterminé par l'admissibilité mais on souhaite plutôt représenter un sous-ensemble particulier des solutions ; ce qui correspond plutôt à la compilation *partielle valuée*.

Étant donné que les résultats établis n'ont pas révélé de langages satisfaisants

pour le support des requêtes d'intérêt, il semble aussi nécessaire d'étudier de nouveaux langages. Là encore, cataloguer les structures de données utilisées dans la littérature en ordonnancement est pertinent, puisque ces structures sont certainement utilisées car elles admettent des propriétés souhaitables. Mais d'autres domaines comme celui de la planification, voire des domaines plus éloignés peuvent aussi inspirer l'utilisation de nouveaux langages pour la représentation d'ensembles de solutions. On peut aussi envisager l'étude de restrictions des langages, comme par exemple les séquences de groupes à k groupes, les diagrammes de décision de largeur k, ou les ordres partiels dont le graphe de précédence est un type de graphe particulier.

Une autre piste consiste à considérer non pas de nouveaux langages cibles, mais un nouveau langage source, correspondant à un autre problème d'ordonnancement dont les solutions peuvent être stockées sous forme de séquences. Cela peut être d'autres problèmes à une machine, ou bien des problèmes d'ordonnancement plus complexes, pour lesquels un algorithme de liste permettrait d'établir une correspondance entre séquence et ordonnancement.

Enfin, certains résultats théoriques qui établissent des propriétés des langages pourraient certainement être étendus. Par exemple, pour le théorème 7.1.2.1 établissant la compacité de langages ensemble par l'expressivité des langages originels, une correspondance peut peut-être aussi être établie entre les hypothèses de fermeture pour la concaténation et la projection avec le support de requêtes des langages, auquel cas le théorème pourrait être appliqué à une grande variété de langages. On peut aussi envisager l'étendre modulo une translation.

Annexes

Annexe A

Résultats expérimentaux détaillés

A.1 Performances généralisées des méthodes de résolution à deux niveaux

Δ	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
0.0000	0.9828	0.9931	0.8406	0.9997	0.4613	0.3950	0.4242
0.1000	0.8898	0.8358	0.7074	0.8599	0.4919	0.4259	0.4452
0.3000	0.8355	0.8021	0.7875	0.8335	0.6746	0.5200	0.5117
0.5000	0.8146	0.7588	0.7850	0.7731	0.8965	0.5666	0.5601
0.7000	0.7914	0.7217	0.7632	0.7346	0.9656	0.5698	0.5647
1.0000	0.7277	0.6715	0.7494	0.6913	0.9992	0.5915	0.5778

Δ	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
0.0000	0.9985	0.9998	0.9987
0.1000	0.9180	0.9231	0.9159
0.3000	0.8516	0.8624	0.8396
0.5000	0.8037	0.8042	0.7964
0.7000	0.7845	0.7575	0.7697
1.0000	0.7398	0.7117	0.7140

Table A.1 – Performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances 1P-Δ)

Δ	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
0.0000	0.9594	0.8984	0.9579	1.0000	0.8689	0.9396	0.5415
0.1000	0.9671	0.9054	0.8756	0.9973	0.8696	0.9388	0.6536
0.3000	0.9738	0.9189	0.9036	0.9841	0.9138	0.9504	0.6910
0.5000	0.9730	0.9117	0.9457	0.9683	0.9443	0.9627	0.7168
0.7000	0.9826	0.9081	0.9605	0.9603	0.9624	0.9736	0.7048
1.0000	0.9870	0.8954	0.9720	0.9420	0.9720	0.9787	0.7134

Δ	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
0.0000	1.0000	1.0000	1.0000
0.1000	0.9921	0.9919	0.9916
0.3000	0.9888	0.9841	0.9847
0.5000	0.9814	0.9820	0.9738
0.7000	0.9849	0.9753	0.9712
1.0000	0.9798	0.9687	0.9590

TABLE A.2 – Performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances JSP-Δ)

|N|	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	CP-COMPL	TAB-GSEQ	TAB-JSEQ
10	0.9805	0.9068	0.9796	0.9136	0.6848	0.9534	0.9659	0.9080
20	0.9519	0.8258	0.9395	0.8285	0.6430	-	0.9038	0.7939
50	0.8985	0.8641	0.8397	0.8966	0.5653	-	0.8312	0.7839
100	0.8545	0.8099	0.7995	0.8344	0.6761	-	0.5570	0.5353
150	0.7597	0.7980	0.6691	0.8373	0.7101	-	0.4077	0.4237
200	0.6270	0.8055	0.6403	0.8439	0.7530	-	0.3892	0.3999

|N|	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
10	0.9873	0.9913	0.9880
20	0.9501	0.9436	0.9138
50	0.9244	0.9360	0.9165
100	0.8760	0.8793	0.8635
150	0.8435	0.8305	0.8354
200	0.8564	0.8655	0.8593

TABLE A.3 – Performance généralisée en fonction de la taille de l'instance (paramètres par défaut, instances JSP-$|N|$)

|N|	|M|	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
5	2	0.9949	0.9937	0.9914	0.9993	0.9459	0.9906	0.9540
	5	0.9833	0.9543	0.9846	0.9854	0.9509	0.9778	0.8909
	10	0.9912	0.9575	0.9847	0.9696	0.9781	0.9869	0.8003
10	2	0.9937	0.9767	0.9842	0.9928	0.9113	0.9808	0.9637
	5	0.9791	0.9523	0.9280	0.9822	0.9436	0.9717	0.8769
	10	0.9812	0.9191	0.9536	0.9738	0.9521	0.9735	0.7090
20	2	0.9897	0.9708	0.8193	0.9966	0.8710	0.9771	0.9590
	5	0.9828	0.9437	0.8592	0.9874	0.9003	0.9589	0.8700
	10	0.9710	0.9069	0.9396	0.9822	0.9522	0.9750	0.5793

|N|	|M|	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
5	2	0.9985	0.9994	0.9985
	5	0.9947	0.9897	0.9897
	10	0.9900	0.9853	0.9884
10	2	0.9970	0.9970	0.9973
	5	0.9899	0.9890	0.9844
	10	0.9863	0.9834	0.9796
20	2	0.9984	0.9982	0.9982
	5	0.9917	0.9909	0.9877
	10	0.9864	0.9885	0.9855

TABLE A.4 – Performance généralisée en fonction de la taille de l'instance (paramètres par défaut, instances JSP-N-M)

$\oplus$	γ	AG-COMPL	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
avg	L_{MAX}	-	0.7938	0.7339	0.5686	0.7769	0.2910	0.3289	0.3007
	$\sum C_i$	-	0.9366	0.8798	0.9149	0.8776	0.9218	0.7762	0.7948
max	L_{MAX}	0.4186	0.7211	0.7034	0.7035	0.7555	0.5264	0.3118	0.3228
	$\sum C_i$	0.6737	0.9427	0.8782	0.9535	0.8809	0.9832	0.7174	0.6580

$\oplus$	γ	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
avg	L_{MAX}	0.8608	0.8411	0.8402
	$\sum C_i$	0.9274	0.9060	0.9111
max	L_{MAX}	0.7316	0.8046	0.7279
	$\sum C_i$	0.9160	0.9006	0.8946

TABLE A.5 – Performance généralisée en fonction de l'objectif (paramètres par défaut, instances 1P-A)

$\oplus$	γ	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
avg	C_{MAX}	0.9845	0.9426	0.9285	0.9828	0.9245	0.9730	0.8588
	$\sum C_i$	0.9925	0.9198	0.9472	0.9664	0.9474	0.9788	0.7984
max	C_{MAX}	0.9547	0.9206	0.9398	0.9763	0.9405	0.9376	0.5626
	$\sum C_i$	0.9817	0.8862	0.9712	0.9581	0.9714	0.9736	0.6511

$\oplus$	γ	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
avg	C_{MAX}	0.9920	0.9896	0.9896
	$\sum C_i$	0.9863	0.9823	0.9796
max	C_{MAX}	0.9752	0.9813	0.9726
	$\sum C_i$	0.9797	0.9795	0.9681

TABLE A.6 – Performance généralisée en fonction de l'objectif (paramètres par défaut, instances JSP-A)

$D(\mathcal{G})$	AG-GSEQ	AG-JSEQ	CP-GSEQ	CP-JSEQ	FIFO	TAB-GSEQ	TAB-JSEQ
0.0000	0.9198	0.8269	0.8587	0.8366	0.6245	0.6383	0.5032
0.0010	0.8862	0.7914	0.8211	0.8250	0.6339	0.6614	0.5356
0.0100	0.8565	0.7850	0.7668	0.8043	0.6905	0.5376	0.5112
0.0500	0.7771	0.7642	0.6194	0.7670	0.7382	0.5330	0.5428
0.1000	0.7450	0.7431	0.6038	0.7503	0.7464	0.5635	0.5723
0.2000	0.7915	0.7897	0.6356	0.7906	0.7847	0.6218	0.6477

$D(\mathcal{G})$	AG-GSEQ*	EW-GSEQ*	TAB-GSEQ*
0.0000	0.9425	0.9464	0.9078
0.0010	0.9095	0.9084	0.8736
0.0100	0.8607	0.8533	0.8394
0.0500	0.7938	0.7820	0.7873
0.1000	0.7620	0.7634	0.7605
0.2000	0.7986	0.7996	0.7950

TABLE A.7 – Performance généralisée en fonction de la densité de précédence (paramètres par défaut, instances JSP-$D(\mathcal{G})$)

A.2 Performances généralisées des méthodes à chaud

A.2.1 Impact du paramètre WC

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
Δ						
0.0000	0.9985	0.9997	0.9998	0.9997	0.9987	0.9997
0.1000	0.9180	0.8822	0.9231	0.8816	0.9159	0.8676
0.3000	0.8516	0.8346	0.8624	0.8445	0.8396	0.8366
0.5000	0.8037	0.7853	0.8042	0.7843	0.7964	0.7767
0.7000	0.7845	0.7440	0.7575	0.7406	0.7697	0.7386
1.0000	0.7398	0.7009	0.7117	0.6974	0.7140	0.6937

TABLE A.8 – Impact du paramètre WC sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances 1P-Δ)

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
Δ						
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1000	0.9921	0.9978	0.9919	0.9978	0.9916	0.9978
0.3000	0.9888	0.9838	0.9841	0.9839	0.9847	0.9838
0.5000	0.9814	0.9672	0.9820	0.9688	0.9738	0.9685
0.7000	0.9849	0.9613	0.9753	0.9609	0.9712	0.9603
1.0000	0.9798	0.9440	0.9687	0.9437	0.9590	0.9441

TABLE A.9 – Impact du paramètre WC sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances JSP-Δ)

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
N						
10	0.9873	0.9153	0.9913	0.9152	0.9880	0.9152
20	0.9501	0.8456	0.9436	0.8368	0.9138	0.8585
50	0.9244	0.9311	0.9360	0.9197	0.9165	0.9306
100	0.8760	0.8483	0.8793	0.8493	0.8635	0.8389
150	0.8435	0.8165	0.8305	0.8112	0.8354	0.8114
200	0.8564	0.8418	0.8655	0.8421	0.8593	0.8430

TABLE A.10 – Impact du paramètre WC sur la performance généralisée en fonction de la taille des instances (paramètres par défaut, instances 1P-N)

	Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
	WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
N	|M|						
5	2	0.9985	0.9997	0.9994	0.9994	0.9985	0.9996
	5	0.9947	0.9860	0.9897	0.9851	0.9897	0.9851
	10	0.9900	0.9709	0.9853	0.9709	0.9884	0.9709
10	2	0.9970	0.9972	0.9970	0.9972	0.9973	0.9968
	5	0.9899	0.9835	0.9890	0.9834	0.9844	0.9845
	10	0.9863	0.9753	0.9834	0.9754	0.9796	0.9753
20	2	0.9984	0.9968	0.9982	0.9982	0.9982	0.9982
	5	0.9917	0.9864	0.9909	0.9882	0.9877	0.9869
	10	0.9864	0.9825	0.9885	0.9825	0.9855	0.9819

TABLE A.11 – Impact du paramètre WC sur la performance généralisée en fonction de la taille des instances (paramètres par défaut, instances JSP-N-M)

	Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
	WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$\oplus$	γ						
avg	L_{MAX}	0.8608	0.7808	0.8411	0.7798	0.8402	0.7792
	$\sum C_i$	0.9274	0.8776	0.9060	0.8772	0.9111	0.8774
max	L_{MAX}	0.7316	0.7942	0.8046	0.7993	0.7279	0.7674
	$\sum C_i$	0.9160	0.8805	0.9006	0.8808	0.8946	0.8808

TABLE A.12 – Impact du paramètre WC sur la performance généralisée en fonction de l'objectif (paramètres par défaut, instances 1P-A)

	Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
	WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$\oplus$	γ						
avg	C_{MAX}	0.9920	0.9817	0.9896	0.9817	0.9896	0.9817
	$\sum C_i$	0.9863	0.9664	0.9823	0.9664	0.9796	0.9664
max	C_{MAX}	0.9752	0.9785	0.9813	0.9809	0.9726	0.9808
	$\sum C_i$	0.9797	0.9580	0.9795	0.9581	0.9681	0.9580

TABLE A.13 – Impact du paramètre WC sur la performance généralisée en fonction de l'objectif (paramètres par défaut, instances JSP-A)

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
WC	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$D(\mathcal{G})$						
0.0000	0.9425	0.8632	0.9464	0.8651	0.9078	0.8488
0.0010	0.9095	0.8423	0.9084	0.8449	0.8736	0.8332
0.0100	0.8607	0.8176	0.8533	0.8135	0.8394	0.8094
0.0500	0.7938	0.7754	0.7820	0.7729	0.7873	0.7701
0.1000	0.7620	0.7550	0.7634	0.7533	0.7605	0.7523
0.2000	0.7986	0.7919	0.7996	0.7927	0.7950	0.7922

TABLE A.14 – Impact du paramètre WC sur la performance généralisée en fonction de la densité de contraintes de précédence (paramètres par défaut, instances 1P-$D(\mathcal{G})$)

A.2.2 Impact du paramètre TPI

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
TPI	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
Δ						
0.0000	0.9985	0.9989	0.9998	0.9998	0.9987	0.9997
0.1000	0.9180	0.9694	0.9231	0.9928	0.9159	0.9483
0.3000	0.8516	0.9524	0.8624	0.9805	0.8396	0.8911
0.5000	0.8037	0.9740	0.8042	0.9828	0.7964	0.8363
0.7000	0.7845	0.9581	0.7575	0.9909	0.7697	0.8057
1.0000	0.7398	0.9435	0.7117	0.9932	0.7140	0.7750

TABLE A.15 – Impact du paramètre TPI sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances 1P-Δ)

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
TPI	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
Δ						
0.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
0.1000	0.9943	0.9921	0.9935	0.9919	0.9929	0.9916
0.3000	0.9838	0.9888	0.9850	0.9841	0.9856	0.9847
0.5000	0.9795	0.9814	0.9808	0.9820	0.9746	0.9738
0.7000	0.9741	0.9849	0.9758	0.9753	0.9726	0.9712
1.0000	0.9649	0.9798	0.9612	0.9687	0.9588	0.9590

TABLE A.16 – Impact du paramètre TPI sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances JSP-Δ)

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
TPI	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$D(\mathcal{G})$						
0.0000	0.9425	0.9348	0.9464	0.9464	0.9078	0.8976
0.0010	0.9095	0.9169	0.9084	0.9574	0.8736	0.8877
0.0100	0.8607	0.9469	0.8533	0.9804	0.8394	0.8879
0.0500	0.7938	0.9792	0.7820	0.9785	0.7873	0.8716
0.1000	0.7620	0.9785	0.7634	0.9921	0.7605	0.8606
0.2000	0.7986	0.9781	0.7996	0.9896	0.7950	0.9101

TABLE A.17 – Impact du paramètre TPI sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances 1P-Δ)

	Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
	TPI	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$\oplus$	γ						
avg	L_{MAX}	0.8608	0.9918	0.8411	0.9969	0.8402	0.8775
	$\sum C_i$	0.9274	0.9999	0.9060	0.9682	0.9111	0.9247
max	L_{MAX}	0.7316	0.8615	0.8046	0.9884	0.7279	0.8551
	$\sum C_i$	0.9160	0.9678	0.9006	0.9602	0.8946	0.8990

TABLE A.18 – Impact du paramètre TPI sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances 1P-A)

	Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
	TPI	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$\oplus$	γ						
avg	C_{MAX}	0.9945	0.9920	0.9908	0.9896	0.9899	0.9896
	$\sum C_i$	0.9812	0.9863	0.9796	0.9823	0.9794	0.9796
max	C_{MAX}	0.9790	0.9752	0.9856	0.9813	0.9755	0.9726
	$\sum C_i$	0.9741	0.9797	0.9699	0.9795	0.9663	0.9681

TABLE A.19 – Impact du paramètre TPI sur la performance généralisée en fonction de la variabilité entre scénarios (paramètres par défaut, instances JSP-A)

A.2.3 Impact du paramètre OF

Méthode	AG-GSEQ*		EW-GSEQ*		TAB-GSEQ*	
OF	$\perp$	$\top$	$\perp$	$\top$	$\perp$	$\top$
$D(\mathcal{G})$						
0.0000	0.8863	0.9425	-	0.9464	0.8494	0.9078
0.0010	0.8733	0.9095	-	0.9084	0.8429	0.8736
0.0100	0.8564	0.8607	-	0.8533	0.8318	0.8394
0.0500	0.7883	0.7938	-	0.7820	0.7824	0.7873
0.1000	0.7630	0.7620	-	0.7634	0.7601	0.7605
0.2000	0.7959	0.7986	-	0.7996	0.7958	0.7950

TABLE A.20 – Impact du paramètre OF sur la performance généralisée en fonction de la densité des contraintes de précédence (paramètres par défaut, instances 1P-$D(\mathcal{G})$)

A.2.4 Performances de AG-COMPL*

Performance	Entraînement			Test		
Méthode	AG-COMPL*	AG-GSEQ*	EW-GSEQ*	AG-COMPL*	AG-GSEQ*	EW-GSEQ*
γ						
L_{MAX}	0.9669	0.9989	0.9989	0.7295	0.7316	0.8046
$\sum C_i$	0.8701	0.8676	0.8539	0.8655	0.9160	0.9006

TABLE A.21 – Performance de méthodes à chaud en fonction du critère (paramètres par défaut, instances 1P-A, $\oplus = max$))

Performance	Entraînement			Test		
Méthode	AG-COMPL*	AG-GSEQ*	EW-GSEQ*	AG-COMPL*	AG-GSEQ*	EW-GSEQ*
|N|						
10	0.9851	0.9801	0.9778	0.9263	0.9747	0.9846
20	0.9807	0.9747	0.9167	0.7813	0.9130	0.9284
50	0.9579	0.9512	0.9484	0.8607	0.8721	0.9153
100	0.9402	0.9379	0.9316	0.8141	0.8451	0.8596
150	0.9324	0.9333	0.9302	0.8102	0.8033	0.8271
200	0.9191	0.9195	0.9190	0.8369	0.8288	0.8557

TABLE A.22 – Performance de méthodes à chaud en fonction du nombre de tâches (paramètres par défaut, instances 1P-$|N|$, $\oplus = max$))

Performance	Entraînement			Test		
Méthode	AG-COMPL*	AG-GSEQ*	EW-GSEQ*	AG-COMPL*	AG-GSEQ*	EW-GSEQ*
|S|						
2	0.9619	0.9774	0.9374	0.6283	0.7190	0.7267
5	0.9766	0.9512	0.9420	0.7383	0.7707	0.7874
10	0.9565	0.9374	0.9318	0.7985	0.8070	0.8471
15	0.8752	0.9341	0.9245	0.7447	0.8118	0.8274
20	0.9393	0.9366	0.9264	0.8096	0.8103	0.8535
25	0.9295	0.9251	0.9191	0.8025	0.8159	0.8605
35	0.9300	0.9334	0.9230	0.8406	0.8699	0.8734
50	0.9219	0.9211	0.9182	0.8186	0.7984	0.8537
100	0.9124	0.9144	0.9089	0.8254	0.8549	0.8675

TABLE A.23 – Performance de méthodes à chaud en fonction du nombre de scénarios d'entraînement (paramètres par défaut, instances 1P-S, $\oplus = max$)

Bibliographie

R. Ahlswede. Appendix : On Set Coverings in Cartesian Product Spaces. In Rudolf Ahlswede, Lars Bäumer, Ning Cai, Harout Aydinian, Vladimir Blinovsky, Christian Deppe, and Haik Mashurian, editors, *General Theory of Information Transfer and Combinatorics*, Lecture Notes in Computer Science, pages 926–937. Springer, Berlin, Heidelberg, 2006. ISBN 978-3-540-46245-3. doi : 10.1007/11889342_58. URL https://doi.org/10.1007/11889342_58. (Cité en page 154.)

Akers. Binary Decision Diagrams. *IEEE Transactions on Computers*, C-27(6) :509–516, June 1978. ISSN 1557-9956. doi : 10.1109/TC.1978.1675141. Conference Name : IEEE Transactions on Computers. (Cité en page 38.)

Mohamed Aloulou and Marie-Claude Portmann. An Efficient Proactive-Reactive Scheduling Approach to Hedge Against Shop Floor Disturbances. In *Multidisciplinary Scheduling : Theory and Applications*, pages 223–246. January 2005. ISBN 978-0-387-25266-7. doi : 10.1007/0-387-27744-7_11. (Cité en pages 35, 36 et 96.)

Mohamed Ali Aloulou and Christian Artigues. Flexible solutions in disjunctive scheduling : General formulation and study of the flow-shop case. *Computers & Operations Research*, 37(5) :890–898, May 2010. ISSN 03050548. doi : 10.1016/j.cor.2009.03.021. URL https://linkinghub.elsevier.com/retrieve/pii/S0305054809000884. (Cité en pages 35 et 37.)

Mohamed Ali Aloulou, Mikhail Y. Kovalyov, and Marie-Claude Portmann. Maximization Problems in Single Machine Scheduling. *Annals of Operations Research*, 129(1-4) :21–32, July 2004. ISSN 0254-5330. doi : 10.1023/B:ANOR.0000030679.25466.02. URL http://link.springer.com/10.1023/B:ANOR.0000030679.25466.02. (Cité en pages 79 et 176.)

H. R. Andersen, T. Hadzic, J. N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In Christian Bessière, editor, *Principles and Practice of Constraint Programming – CP 2007*, Lecture Notes in Computer Science, pages 118–132, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-74970-7. doi : 10.1007/978-3-540-74970-7_11. (Cité en pages 101, 103 et 110.)

Christian Artigues, Jean-Charles Billaut, and Carl Esswein. Maximization of solution flexibility for robust shop scheduling. *European Journal of Operational Research*, 165(2) :314–328, September 2005. ISSN 0377-2217. doi : 10.1016/j.ejor.2004.04.004. URL https://www.sciencedirect.com/science/article/pii/S0377221704002425. (Cité en pages 25, 28, 68, 71, 73, 74, 76 et 182.)

Christian Artigues, Jean-Charles Billaut, Azeddine Cheref, Nasser Mebarki, and Zakaria Yahouni. Robust Machine Scheduling Based on Group of Permutable Jobs. In Doumpos M, Zopounidis C, and Grigoroudis E, editors, *Robustness Analysis in Decision Aiding*, volume 241 of *Optimization, and Analytics. International Series in Operations Research & Management Science*, pages 191–220. Springer, 2016. doi : 10.1007/978-3-319-33121-8_9. URL https://hal.laas.fr/hal-01875889. (Cité en pages 36, 53, 69, 71, 79, 83, 84 et 85.)

Haldun Aytug, Mark A. Lawley, Kenneth McKay, Shantha Mohan, and Reha Uzsoy. Executing production schedules in the face of uncertainties : A review and some future directions. *European Journal of Operational Research*, 161(1) :86–110, February 2005. ISSN 0377-2217. doi : 10.1016/j.ejor.2003.08.027. URL https://www.sciencedirect.com/science/article/pii/S0377221703005307. (Cité en page 23.)

Oliver Bachtler, Sven O. Krumke, and Huy Minh Le. Robust single machine makespan scheduling with release date uncertainty. *Operations Research Letters*, 48(6) :816–819, November 2020. ISSN 0167-6377. doi : 10.1016/j.orl.2020.10.003. URL https://www.sciencedirect.com/science/article/pii/S0167637720301516. (Cité en pages 26 et 50.)

Kenneth R. Baker and Brian Keller. Solving the single-machine sequencing problem using integer programming. *Computers & Industrial Engineering*, 59(4) :730–735, November 2010. ISSN 0360-8352. doi : 10.1016/j.cie.2010.07.028. URL https://www.sciencedirect.com/science/article/pii/S0360835210002196. (Cité en page 16.)

Egon Balas. Machine Sequencing Via Disjunctive Graphs : An Implicit Enumeration Algorithm. *Operations Research*, 17(6) :941–957, December 1969. ISSN 0030-364X. doi : 10.1287/opre.17.6.941. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.17.6.941. Publisher : INFORMS. (Cité en pages 56 et 57.)

Baptiste, Cho, Favrel, and Zouhri. Une caractérisation analytique des ordonnancements admissibles sous contraintes hétérogènes en flow-shop. *Journal Européen des Systèmes Automatisés*, 25 :87–102, 1991. (Cité en page 37.)

PIERRE Baptiste and JOEL Favrel. Résolution de problèmes d'ordonnancmeent par les treillis de Galois et les graphes d'intervalle. *RAIRO - Operations Research - Recherche Opérationnelle*, 18 :405–430, 1984. (Cité en page 37.)

PIERRE Baptiste and JOEL Favrel. Taking into account the rescheduling problem during the scheduling phase. *Production Planning & Control*, 4(4) :349–360, January 1993. ISSN 0953-7287. doi : 10.1080/09537289308919457. (Cité en pages 35, 173 et 184.)

O. Bellenguez, N. Brauner, and A. Tsoukiàs. Is there an ethical operational research practice ? And what this implies for our research ? *EURO Journal on Decision Processes*, 11 :100029, January 2023. ISSN 2193-9438. doi : 10.1016/j.ejdp.2023.100029. (Cité en page 10.)

A. Ben-Tal, A. Goryashko, E. Guslitzer, and A. Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2) :351–376, March 2004. ISSN 1436-4646. doi : 10.1007/s10107-003-0454-y. (Cité en page 34.)

Aharon Ben-Tal and Arkadi Nemirovski. Robust solutions of Linear Programming problems contaminated with uncertain data. *Mathematical Programming*, 88(3) : 411–424, September 2000. ISSN 1436-4646. doi : 10.1007/PL00011380. URL https://doi.org/10.1007/PL00011380. (Cité en pages 23 et 26.)

Aharon Ben-Tal and Arkadi Nemirovski. Robust optimization – methodology and applications. *Mathematical Programming*, 92(3) :453–480, May 2002. ISSN 1436-4646. doi : 10.1007/s101070100286. URL https://doi.org/10.1007/s101070100286. (Cité en page 26.)

David Bergman and Andre A. Cire. Discrete Nonlinear Optimization by State-Space Decompositions. *Management Science*, 64(10) :4700–4720, October 2018. ISSN 0025-1909. doi : 10.1287/mnsc.2017.2849. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.2017.2849. Publisher : INFORMS. (Cité en page 101.)

David Bergman and Andre Augusto Cire. On Finding the Optimal BDD Relaxation. In Domenico Salvagnin and Michele Lombardi, editors, *Integration of AI and OR Techniques in Constraint Programming*, Lecture Notes in Computer Science, pages 41–50, Cham, 2017. Springer International Publishing. ISBN 978-3-319-59776-8. doi : 10.1007/978-3-319-59776-8_4. (Cité en pages 102 et 111.)

David Bergman, Andre Cire, and Willem-Jan van Hoeve. MDD Propagation for Sequence Constraints. *The Journal of Artificial Intelligence Research (JAIR)*, 50, July 2014. doi : 10.1613/jair.4199. (Cité en page 101.)

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and John Hooker. *Decision Diagrams for Optimization*. Springer International Publishing, October 2016a. ISBN 978-3-319-42847-5. Google-Books-ID : o6uvDAEACAAJ. (Cité en pages 38, 39, 101, 103 et 176.)

David Bergman, Andre A. Cire, Willem-Jan van Hoeve, and J. N. Hooker. Discrete Optimization with Decision Diagrams. *INFORMS Journal on Computing*, 28(1) :47–66, January 2016b. ISSN 1091-9856. doi : 10.1287/ijoc.2015.0648. URL https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2015.0648. Publisher : INFORMS. (Cité en pages 39 et 101.)

Dimitris Bertsimas and Melvyn Sim. The Price of Robustness. *Operations Research*, 52(1) :35–53, February 2004. ISSN 0030-364X. doi : 10.1287/opre.1030.0065. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.1030.0065. Publisher : INFORMS. (Cité en page 26.)

Christian Bessiere, Hélène Fargier, and Christophe Lecoutre. Global Inverse Consistency for Interactive Constraint Satisfaction. In Christian Schulte, editor, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 159–174, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-40627-0. doi : 10.1007/978-3-642-40627-0_15. (Cité en page 136.)

Jean-Charles Billaut and F. Roubellat. A new method for workshop real time scheduling. *International Journal of Production Research*, 34(6) :1555–1579, June 1996. ISSN 0020-7543. doi : 10.1080/00207549608904984. URL https://doi.org/10.1080/00207549608904984. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/00207549608904984. (Cité en pages 36, 68 et 71.)

Jean-Charles Billaut, Aziz Moukrim, and Eric Sanlaville. *Flexibility and Robustness in Scheduling*. John Wiley & Sons, March 2013. ISBN 978-1-118-62339-8. Google-Books-ID : 5R6nxKJp2hMC. (Cité en pages 10, 11, 24 et 31.)

Jean-Pierre Brans and Giorgio Gallo. Ethics in OR/MS : past, present and future. *Annals of Operations Research*, 153(1) :165–178, September 2007. ISSN 1572-9338. doi : 10.1007/s10479-007-0177-1. URL https://doi.org/10.1007/s10479-007-0177-1. (Cité en page 10.)

Cyril Briand, H. Trung La, and Jacques Erschler. A robust approach for the single machine scheduling problem. *Journal of Scheduling*, 10(3) :209–221, June 2007. ISSN 1099-1425. doi : 10.1007/s10951-007-0010-3. URL https://doi.org/10.1007/s10951-007-0010-3. (Cité en page 13.)

Peter Brucker and Andreas Krämer. Polynomial algorithms for resource-constrained and multiprocessor task scheduling problems. *European Journal of Operational Research*, 90(2) :214–226, April 1996. ISSN 0377-2217. doi : 10.1016/0377-2217(95)00350-9. URL https://www.sciencedirect.com/science/article/pii/0377221795003509. (Cité en page 18.)

Maria Elena Bruni, Luigi Di Puglia Pugliese, Patrizia Beraldi, and Francesca Guerriero. A Two-stage Stochastic Programming Model for the Resource Constrained Project Scheduling Problem under Uncertainty :. In *Proceedings of the 7th International Conference on Operations Research and Enterprise Systems*, pages 194–200, Funchal, Madeira, Portugal, 2018. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-285-1. doi : 10.5220/0006612601940200. URL http://www.scitepress.org/DigitalLibrary/Link.aspx?doi=10.5220/0006612601940200. (Cité en page 34.)

Jacek Błażewicz, Klaus H. Ecker, Erwin Pesch, Günter Schmidt, and Jan Weglarz. *Scheduling Computer and Manufacturing Processes*. Springer Science & Business Media, June 2001. ISBN 978-3-540-41931-0. Google-Books-ID : 0rrq1xlJ1dsC. (Cité en page 10.)

Olivier Cardin, Nasser Mebarki, and Guillaume Pinot. A study of the robustness of the group scheduling method using an emulation of a complex FMS. *International Journal of Production Economics*, 146(1) :199–207, November 2013. ISSN 0925-5273. doi : 10.1016/j.ijpe.2013.06.023. URL https://www.sciencedirect.com/science/article/pii/S0925527313002934. (Cité en pages 68, 69 et 84.)

Lorenzo Castelli, Raffaele Pesenti, and Walter Ukovich. Scheduling multimodal transportation systems. *European Journal of Operational Research*, 155(3) :603–615, June 2004. ISSN 0377-2217. doi : 10.1016/j.ejor.2003.02.002. URL https://www.sciencedirect.com/science/article/pii/S0377221703004818. (Cité en page 10.)

Margarita P. Castro, Andre A. Cire, and J. Christopher Beck. Decision Diagrams for Discrete Optimization : A Survey of Recent Advances. *INFORMS Journal on Computing*, March 2022. ISSN 1091-9856. doi : 10.1287/ijoc.2022.1170. URL https://pubsonline.informs.org/doi/abs/10.1287/ijoc.2022.1170. Publisher : INFORMS. (Cité en page 101.)

Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms—I. representation. *Computers & Industrial Engineering*, 30(4) :983–997, September 1996. ISSN 0360-8352. doi : 10.1016/0360-8352(96)00047-2. URL https://www.sciencedirect.com/science/article/pii/0360835296000472. (Cité en page 21.)

Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms, part II : hybrid genetic search strategies. *Computers & Industrial Engineering*, 36(2) :343–364, April 1999. ISSN 0360-8352. doi : 10.1016/S0360-8352(99)00136-9. URL https://www.sciencedirect.com/science/article/pii/S0360835299001369. (Cité en page 21.)

Azeddine Cheref, Teddy Bouchard, Jean-Charles Billaut, and Christian Artigues. Un algorithme tabou pour résoudre, grâce aux groupes d'opérations permutables, un problème d'ordonnancement et de routing robuste. November 2014. (Cité en pages 35 et 96.)

Azeddine Cheref, Christian Artigues, and Jean-Charles Billaut. A new robust approach for a production scheduling and delivery routing problem. In *8th IFAC Conference on Manufacturing Modelling, Management and Control*, 8th IFAC Conference on Manufacturing Modelling, Management and Control, Troyes, France, June 2016a. URL https://hal.archives-ouvertes.fr/hal-01350826. (Cité en pages 69, 71 et 79.)

Azeddine Cheref, Christian Artigues, and Jean-Charles Billaut. Online recoverable robustness based on groups of permutable jobs for integrated production scheduling and delivery routing. August 2016b. URL https://hal.archives-ouvertes.fr/hal-01351496. (Cité en pages 36, 69, 71, 79, 84 et 96.)

A Cire and Willem-Jan van Hoeve. MDD Propagation for Disjunctive Scheduling. pages 11–19, January 2012. (Cité en pages 101 et 175.)

Andre Cire and Willem-Jan van Hoeve. Multivalued Decision Diagrams for Sequencing Problems. *Operations Research*, 61, December 2013. ISSN 978-3-319-10427-0. doi : 10.1287/opre.2013.1221. (Cité en pages 39, 101 et 103.)

P. R. Conrad and B. C. Williams. Drake : An Efficient Executive for Temporal Plans with Choice. *Journal of Artificial Intelligence Research*, 42 :607–659, December 2011. ISSN 1076-9757. doi : 10.1613/jair.3478. URL https://www.jair.org/index.php/jair/article/view/10735. (Cité en page 131.)

Richard L. Daniels and Panagiotis Kouvelis. Robust Scheduling to Hedge Against Processing Time Uncertainty in Single-Stage Production. *Management Science*, 41(2) :363–376, February 1995. ISSN 0025-1909. doi : 10.1287/mnsc.41.2.363. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.41.2.363. Publisher : INFORMS. (Cité en page 23.)

Morteza Davari and Erik Demeulemeester. The proactive and reactive resource-constrained project scheduling problem. *Journal of Scheduling*, 22, April 2019. doi : 10.1007/s10951-017-0553-x. (Cité en pages 23 et 34.)

Davenport and Beck. A Survey of Techniques for Scheduling with Uncertainty, 2000. URL https://tidel.mie.utoronto.ca/pubs/uncertainty-survey.pdf. (Cité en page 31.)

Rabah Demmou. *Etudes de familles remarquable d'ordonnancements, en vue d'une aide à la décision*. PhD thesis, Université Paul Sabatier - Toulouse III, 1977. (Cité en page 35.)

Mark Drummond, John Bresina, and Keith Swanson. Just-In-Case Scheduling. *Proceedings of the 12th National Conference on Artificial Intelligence*, 1994. (Cité en page 34.)

Didier Dubois, Helene Fargier, and Philippe Fortemps. Fuzzy scheduling : Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147(2) :231–252, June 2003. ISSN 0377-2217. doi : 10.1016/S0377-2217(02)00558-1. URL https://www.sciencedirect.com/science/article/pii/S0377221702005581. (Cité en page 26.)

Stanislaw Dymitrowski. Compilation de connaissances pour le problème de gestion de projet à contraintes de ressources, August 2020. (Cité en page 129.)

Jacques Erschler and François Roubellat. An Approach to Solve Workshop Real Time Scheduling Problems. In Shimon Y. Nof and Colin L. Moodie, editors, *Advanced Information Technologies for Industrial Material Flow Systems*, NATO ASI Series, pages 651–679, Berlin, Heidelberg, 1989. Springer. ISBN 978-3-642-74575-1. doi : 10.1007/978-3-642-74575-1_27. (Cité en pages 34 et 71.)

Patrick Esquirol and Pierre Lopez. *L'ordonnancement*. ECONOMICA, 1999. (Cité en page 9.)

Carl Esswein. *Un apport de flexibilité séquentielle pour l'ordonnancement robuste*. These de doctorat, Tours, January 2003. URL http://www.theses.fr/2003TOUR4022. (Cité en pages 68, 83, 84 et 85.)

Helene Fargier, Pierre Marquis, and Alexandre Niveau. Towards a Knowledge Compilation Map for Heterogeneous Representation Languages. page 8, 2013. (Cité en pages 131, 133, 138, 139 et 152.)

Hélène Fargier and Pierre Marquis. Disjunctive closures for knowledge compilation. *Artificial Intelligence*, 216 :129–162, November 2014. ISSN 0004-3702. doi : 10.1016/j.artint.2014.07.004. URL https://www.sciencedirect.com/science/article/pii/S0004370214000873. (Cité en page 144.)

Hélène Fargier, Frédéric Maris, and Vincent Roger. Temporal Constraint Satisfaction Problems and Difference Decision Diagrams : A Compilation Map. In *27th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015)*, pages 429–436, Vietri sul Mare, Italy, November 2015. IEEE. doi : 10.1109/ICTAI.2015.71. URL https://hal.archives-ouvertes.fr/hal-01303829. (Cité en page 130.)

Mohammad Hossein Fazel Zarandi, Ali Akbar Sadat Asl, Shahabeddin Sotudian, and Oscar Castillo. A state of the art review of intelligent scheduling. *Artificial Intelligence Review*, 53(1) :501–593, January 2020. ISSN 1573-7462. doi : 10.1007/s10462-018-9667-6. URL https://doi.org/10.1007/s10462-018-9667-6. (Cité en page 17.)

Alan M. Frisch, Brahim Hnich, Zeynep Kiziltan, Ian Miguel, and Toby Walsh. Propagation algorithms for lexicographic ordering constraints. *Artificial Intelligence*, 170(10) :803–834, July 2006. ISSN 0004-3702. doi : 10.1016/j.artint.2006.03.002. URL https://www.sciencedirect.com/science/article/pii/S0004370206000385. (Cité en page 115.)

Robert Ganian, Thekla Hamm, and Guillaume Mescoff. The Complexity Landscape of Resource-Constrained Scheduling. volume 2, pages 1741–1747, July 2020. doi : 10.24963/ijcai.2020/241. URL https://www.ijcai.org/proceedings/2020/241. ISSN : 1045-0823. (Cité en pages 11 et 14.)

M. R. Garey, D. S. Johnson, and Ravi Sethi. The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2) :117–129, 1976.

ISSN 0364-765X. URL https://www.jstor.org/stable/3689278. Publisher : INFORMS. (Cité en page 46.)

William V. Gehrlein. On methods for generating random partial orders. *Operations Research Letters*, 5(6) :285–291, December 1986. ISSN 0167-6377. doi : 10.1016/0167-6377(86)90066-0. URL https://www.sciencedirect.com/science/article/pii/0167637786900660. (Cité en page 54.)

Amir Ghasemi, Amir Ashoori, and Cathal Heavey. Evolutionary Learning Based Simulation Optimization for Stochastic Job Shop Scheduling Problems. *Applied Soft Computing*, 106 :107309, July 2021. ISSN 1568-4946. doi : 10.1016/j.asoc.2021.107309. URL https://www.sciencedirect.com/science/article/pii/S1568494621002325. (Cité en pages 56, 58 et 76.)

B. Giffler and G. L. Thompson. Algorithms for Solving Production-Scheduling Problems. *Operations Research*, 8(4) :487–503, 1960. ISSN 0030-364X. URL https://www.jstor.org/stable/167291. Publisher : INFORMS. (Cité en page 13.)

Gillies and Liu. Scheduling Tasks with AND/OR Precedence Constraints. 1995. doi : 10.1137/S0097539791218664. URL https://epubs.siam.org/doi/epdf/10.1137/S0097539791218664. (Cité en page 37.)

Goran Gogic, Henry Kautz, Christos Papadimitriou, and Bart Selman. The Comparative Linguistics of Knowledge Representation. page 8, 1995. (Cité en page 130.)

Goldratt. *Critical Chain : A business Novel*. Gower Publishing, 1997. ISBN 0-88427-153-6. (Cité en page 23.)

Gotha. Les problèmes d'ordonnancement. *RAIRO - Operations Research - Recherche Opérationnelle*, 27(1) :77–150, 1993. ISSN 0399-0559. URL https://eudml.org/doc/105053. Publisher : EDP-Sciences. (Cité en page 14.)

Michel Gourgand, Nathalie Grangeon, and Sylvie Norre. A review of the static stochastic flow-shop scheduling problem. *Journal of Decision Systems*, 9(2) : 1–31, January 2000. ISSN 1246-0125. doi : 10.1080/12460125.2000.9736710. URL https://doi.org/10.1080/12460125.2000.9736710. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/12460125.2000.9736710. (Cité en pages 10 et 30.)

R. L. Graham. Bounds for certain multiprocessing anomalies. *The Bell System Technical Journal*, 45(9) :1563–1581, November 1966. ISSN 0005-8580. doi : 10.1002/j.1538-7305.1966.tb01709.x. Conference Name : The Bell System Technical Journal. (Cité en pages 14 et 30.)

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and Approximation in Deterministic Sequencing and Scheduling : a Survey. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Annals*

of Discrete Mathematics, volume 5 of *Discrete Optimization II*, pages 287–326. Elsevier, January 1979. doi : 10.1016/S0167-5060(08)70356-X. URL https://www.sciencedirect.com/science/article/pii/S016750600870356X. (Cité en page 10.)

Jinwei Gu, Xingsheng Gu, and Manzhan Gu. A novel parallel quantum genetic algorithm for stochastic job shop scheduling. *Journal of Mathematical Analysis and Applications*, 355(1) :63–81, July 2009. ISSN 0022-247X. doi : 10.1016/j.jmaa.2008.12.065. URL https://www.sciencedirect.com/science/article/pii/S0022247X0900016X. (Cité en pages 56, 58 et 76.)

Tarik Hadzic and J N Hooker. Postoptimality Analysis for Integer Programming Using Binary Decision Diagrams. 2006. (Cité en pages 39 et 102.)

Tarik Hadžić and J. N. Hooker. Cost-Bounded Binary Decision Diagrams for 0-1 Programming. In Pascal Van Hentenryck and Laurence Wolsey, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 84–98, Berlin, Heidelberg, 2007. Springer. ISBN 978-3-540-72397-4. doi : 10.1007/978-3-540-72397-4_7. (Cité en page 183.)

Leslie A. Hall and David B. Shmoys. Jackson's Rule for Single-Machine Scheduling : Making a Good Heuristic Better. *Mathematics of Operations Research*, 17 (1) :22–35, February 1992. ISSN 0364-765X. doi : 10.1287/moor.17.1.22. URL https://pubsonline.informs.org/doi/abs/10.1287/moor.17.1.22. Publisher : INFORMS. (Cité en page 18.)

Nicholas G. Hall and Marc E. Posner. Sensitivity Analysis for Scheduling Problems. *Journal of Scheduling*, 7(1) :49–83, January 2004. ISSN 1094-6136. doi : 10.1023/B:JOSH.0000013055.31639.f6. URL http://link.springer.com/10.1023/B:JOSH.0000013055.31639.f6. (Cité en page 23.)

Xinchang Hao, Lin Lin, Mitsuo Gen, and Katsuhisa Ohno. Effective Estimation of Distribution Algorithm for Stochastic Job Shop Scheduling Problem. *Procedia Computer Science*, 20 :102–107, January 2013. ISSN 1877-0509. doi : 10.1016/j.procs.2013.09.246. URL https://www.sciencedirect.com/science/article/pii/S1877050913010466. (Cité en pages 56, 58 et 76.)

Jeffrey W. Herrmann. A History of Production Scheduling. In Jeffrey W. Herrmann, editor, *Handbook of Production Scheduling*, International Series in Operations Research & Management Science, pages 1–22. Springer US, Boston, MA, 2006. ISBN 978-0-387-33117-1. doi : 10.1007/0-387-33117-4_1. URL https://doi.org/10.1007/0-387-33117-4_1. (Cité en pages 7 et 10.)

Willy Herroelen and Roel Leus. The construction of stable project baseline schedules. *European Journal of Operational Research*, 156(3) :550–565, August 2004a. ISSN 0377-2217. doi : 10.1016/S0377-2217(03)00130-9. URL https:

//www.sciencedirect.com/science/article/pii/S0377221703001309. (Cité en page 32.)

Willy Herroelen and Roel Leus. Robust and reactive project scheduling : A review and classification of procedures. *Katholieke Universiteit Leuven, Open Access publications from Katholieke Universiteit Leuven*, 42, April 2004b. doi : 10.1080/00207540310001638055. (Cité en page 31.)

Willy Herroelen and Roel Leus. Project scheduling under uncertainty : Survey and research potentials. *European Journal of Operational Research*, 165(2) :289–306, September 2005. ISSN 0377-2217. doi : 10.1016/j.ejor.2004.04.002. URL https://www.sciencedirect.com/science/article/pii/S0377221704002401. (Cité en pages 23, 26 et 31.)

Samid Hoda, Willem-Jan van Hoeve, and John Hooker. A Systematic Approach to MDD-Based Constraint Programming. volume 6308, pages 266–280, September 2010. ISBN 978-3-642-15395-2. doi : 10.1007/978-3-642-15396-9_23. (Cité en page 101.)

J. N. Hooker. Stochastic Decision Diagrams. In Pierre Schaus, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, pages 138–154, Cham, 2022. Springer International Publishing. ISBN 978-3-031-08011-1. doi : 10.1007/978-3-031-08011-1_11. (Cité en page 102.)

John N. Hooker. Improved Job Sequencing Bounds from Decision Diagrams. In Thomas Schiex and Simon de Givry, editors, *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 268–283, Cham, 2019. Springer International Publishing. ISBN 978-3-030-30048-7. doi : 10.1007/978-3-030-30048-7_16. (Cité en page 101.)

Shih-Cheng Horng and Shieh-Shing Lin. Integrating Ant Colony System and Ordinal Optimization for Solving Stochastic Job Shop Scheduling Problem. In *Modelling and Simulation 2015 6th International Conference on Intelligent Systems*, pages 70–75, February 2015. doi : 10.1109/ISMS.2015.9. ISSN : 2166-0670. (Cité en pages 56, 58 et 76.)

Sana Ikli, Catherine Mancel, Marcel Mongeau, Xavier Olive, and Emmanuel Rachelson. The aircraft runway scheduling problem : A survey. *Computers & Operations Research*, 132 :105336, August 2021. ISSN 0305-0548. doi : 10.1016/j.cor.2021.105336. URL https://www.sciencedirect.com/science/article/pii/S0305054821001192. (Cité en page 10.)

Anant Jain, Balasubramanian Rangaswamy, and Sheik Meeran. New and "Stronger" Job-Shop Neighbourhoods : A Focus on the Method of Nowicki and Smutnicki (1996). *J. Heuristics*, 6 :457–480, September 2000. doi : 10.1023/A:1009617209268. (Cité en page 19.)

Jensen. *Robust and Flexible Scheduling with Evolutionary Computation*. PhD thesis, 2001. URL https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=9387e1e870ed1a5e87e9a3ef8a00bf72fa33c886. (Cité en page 32.)

Nan Jiang, Maosen Zhang, Willem-Jan van Hoeve, and Yexiang Xue. Constraint Reasoning Embedded Structured Prediction. *Journal of Machine Learning Research*, 23(345) :1–40, 2022. ISSN 1533-7928. URL http://jmlr.org/papers/v23/21-1484.html. (Cité en page 102.)

Antoine Jouglet and Jacques Carlier. Dominance rules in combinatorial optimization problems. *European Journal of Operational Research*, 212(3) :433–444, August 2011. ISSN 0377-2217. doi : 10.1016/j.ejor.2010.11.008. URL https://www.sciencedirect.com/science/article/pii/S0377221710007654. (Cité en page 13.)

Tsuyoshi Kawaguchi and Seiki Kyan. Worst Case Bound of an LRF Schedule for the Mean Weighted Flow-Time Problem. *SIAM Journal on Computing*, 15(4) :1119–1129, November 1986. ISSN 0097-5397. doi : 10.1137/0215081. URL https://epubs.siam.org/doi/10.1137/0215081. Publisher : Society for Industrial and Applied Mathematics. (Cité en page 18.)

Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1) :3–13, January 2011. ISSN 0305-0548. doi : 10.1016/j.cor.2009.12.011. URL https://www.sciencedirect.com/science/article/pii/S0305054809003360. (Cité en pages 14 et 16.)

Panos Kouvelis and Gang Yu. *Robust Discrete Optimization and Its Applications*, volume 14 of *Nonconvex Optimization and Its Applications*. Springer US, Boston, MA, 1997. ISBN 978-1-4419-4764-2 978-1-4757-2620-6. doi : 10.1007/978-1-4757-2620-6. URL http://link.springer.com/10.1007/978-1-4757-2620-6. (Cité en pages 56, 57 et 76.)

Wen-Yang Ku and J. Christopher Beck. Mixed Integer Programming models for job shop scheduling : A computational analysis. *Computers & Operations Research*, 73 :165–173, September 2016. ISSN 0305-0548. doi : 10.1016/j.cor.2016.04.006. URL https://www.sciencedirect.com/science/article/pii/S0305054816300764. (Cité en pages 14 et 16.)

B. Lageweg, J. Lenstra, E. Lawler, and A. Kan. Computer-Aided complexity classification of combinational problems. *Communications of the ACM*, 25 :817–822, November 1982. doi : 10.1145/358690.363066. (Cité en page 14.)

E. L. Lawler. Optimal Sequencing of a Single Machine Subject to Precedence Constraints. *Management Science*, 19(5) :544–546, 1973. ISSN 0025-1909. URL https://www.jstor.org/stable/2629451. Publisher : INFORMS. (Cité en page 18.)

C. Y. Lee. Representation of Switching Circuits by Binary-Decision Programs. *Bell System Technical Journal*, 38(4) :985–999, 1959. ISSN 1538-7305. doi : 10.1002/j.1538-7305.1959.tb01585.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1959.tb01585.x. _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1959.tb01585.x. (Cité en page 38.)

J. K. Lenstra and A. H. G. Rinnooy Kan. Computational Complexity of Discrete Optimization Problems. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Annals of Discrete Mathematics*, volume 4 of *Discrete Optimization I*, pages 121–140. Elsevier, January 1979. doi : 10.1016/S0167-5060(08)70821-5. URL https://www.sciencedirect.com/science/article/pii/S0167506008708215. (Cité en page 46.)

J. K. Lenstra, A. H. G. Rinnooy Kan, and P. Brucker. Complexity of Machine Scheduling Problems. In P. L. Hammer, E. L. Johnson, B. H. Korte, and G. L. Nemhauser, editors, *Annals of Discrete Mathematics*, volume 1 of *Studies in Integer Programming*, pages 343–362. Elsevier, January 1977. doi : 10.1016/S0167-5060(08)70743-X. URL https://www.sciencedirect.com/science/article/pii/S016750600870743X. (Cité en pages 14, 46, 129 et 173.)

Joseph Y.-T. Leung, editor. *Handbook of Scheduling : Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, New York, April 2004. ISBN 978-0-429-20564-4. doi : 10.1201/9780203489802. (Cité en page 17.)

Florin Leutwiler and Francesco Corman. A review of principles and methods to decompose large-scale railway scheduling problems. *EURO Journal on Transportation and Logistics*, 12 :100107, January 2023. ISSN 2192-4376. doi : 10.1016/j.ejtl.2023.100107. URL https://www.sciencedirect.com/science/article/pii/S2192437623000043. (Cité en page 10.)

Zukui Li and Marianthi Ierapetritou. Process scheduling under uncertainty : Review and challenges. *Computers & Chemical Engineering*, 32(4) :715–727, April 2008. ISSN 0098-1354. doi : 10.1016/j.compchemeng.2007.03.001. URL https://www.sciencedirect.com/science/article/pii/S0098135407000580. (Cité en page 23.)

Michele Lombardi and Michela Milano. Optimal methods for resource allocation and scheduling : a cross-disciplinary survey. *Constraints*, 17(1) :51–85, January 2012. ISSN 1572-9354. doi : 10.1007/s10601-011-9115-6. URL https://doi.org/10.1007/s10601-011-9115-6. (Cité en page 17.)

Leonardo Lozano and J. Cole Smith. A binary decision diagram based algorithm for solving a class of binary two-stage stochastic programs. *Mathematical Programming*, 191(1) :381–404, January 2022. ISSN 1436-4646. doi : 10.1007/s10107-018-1315-z. URL https://doi.org/10.1007/s10107-018-1315-z. (Cité en page 101.)

Moira MacNeil and Merve Bodur. Leveraging Decision Diagrams to Solve Two-stage Stochastic Programs with Binary Recourse and Logical Linking Constraints, November 2022. URL http://arxiv.org/abs/2211.02169. arXiv :2211.02169 [math]. (Cité en page 102.)

Steve Malalel, Arnaud Malapert, Marie Pelleau, and Jean-Charles Régin. MDD Archive for Boosting the Pareto Constraint. In Roland H. C. Yap, editor, *29th International Conference on Principles and Practice of Constraint Programming (CP 2023)*, volume 280 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24 :1–24 :15, Dagstuhl, Germany, 2023. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISBN 978-3-95977-300-3. doi : 10.4230/LIPIcs.CP.2023.24. URL https://drops.dagstuhl.de/opus/volltexte/2023/19061. ISSN : 1868-8969. (Cité en page 101.)

R. H. Möhring, F. J. Radermacher, and G. Weiss. Stochastic scheduling problems I — General strategies. *Zeitschrift für Operations Research*, 28(7) : 193–260, November 1984. ISSN 1432-5217. doi : 10.1007/BF01919323. URL https://doi.org/10.1007/BF01919323. (Cité en page 23.)

Rolf H. Möhring, Martin Skutella, and Frederik Stork. Scheduling with AND/OR Precedence Constraints. *SIAM Journal on Computing*, 33(2) :393–415, January 2004. ISSN 0097-5397, 1095-7111. doi : 10.1137/S009753970037727X. URL http://epubs.siam.org/doi/10.1137/S009753970037727X. (Cité en pages 37 et 173.)

Jesper Møller, Jakob Lichtenberg, Henrik Reif Andersen, and Henrik Hulgaard. Difference Decision Diagrams. In Jörg Flum and Mario Rodriguez-Artalejo, editors, *Computer Science Logic*, Lecture Notes in Computer Science, pages 111–125, Berlin, Heidelberg, 1999. Springer. ISBN 978-3-540-48168-3. doi : 10.1007/3-540-48168-0_9. (Cité en page 130.)

Eugeniusz Nowicki and Czeslaw Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, June 1996. doi : 10.1287/mnsc.42.6.797. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.42.6.797. Publisher : INFORMS. (Cité en pages 13 et 19.)

Richard J. Ormerod and Werner Ulrich. Operational research and ethics : A literature review. *European Journal of Operational Research*, 228(2) :291–307, July 2013. ISSN 0377-2217. doi : 10.1016/j.ejor.2012.11.048. URL https://www.sciencedirect.com/science/article/pii/S0377221712009009. (Cité en page 10.)

Michael Pinedo and Linus Schrage. Stochastic Shop Scheduling : A Survey. In M. A. H. Dempster, J. K. Lenstra, and A. H. G. Rinnooy Kan, editors, *Deterministic and Stochastic Scheduling*, NATO Advanced Study Institutes Series, pages 181–196, Dordrecht, 1982. Springer Netherlands. ISBN 978-94-009-7801-0. doi : 10.1007/978-94-009-7801-0_9. (Cité en page 23.)

Guillaume Pinot, Olivier Cardin, and Nasser Mebarki. A study on the group sequencing method in regards with transportation in an industrial FMS. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pages 151–156, October 2007. doi : 10.1109/ICSMC.2007.4414227. ISSN : 1062-922X. (Cité en pages 68, 69 et 84.)

Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen Smith. From precedence constraint posting to partial order schedules : A CSP approach to Robust Scheduling. *AI Commun.*, 20 :163–180, January 2007. (Cité en page 36.)

Nicola Policella, Amedeo Cesta, Angelo Oddi, and Stephen F. Smith. Solve-and-robustify. *Journal of Scheduling*, 12(3) :299–314, June 2009. ISSN 1099-1425. doi : 10.1007/s10951-008-0091-7. URL https://doi.org/10.1007/s10951-008-0091-7. (Cité en pages 36 et 182.)

Tom Portoleau, Christian Artigues, and Romain Guillaume. Robust Predictive-Reactive Scheduling : An Information-Based Decision Tree Model. In Marie-Jeanne Lesot, Susana Vieira, Marek Z. Reformat, João Paulo Carvalho, Anna Wilbik, Bernadette Bouchon-Meunier, and Ronald R. Yager, editors, *Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Communications in Computer and Information Science, pages 479–492, Cham, 2020. Springer International Publishing. ISBN 978-3-030-50153-2. doi : 10.1007/978-3-030-50153-2_36. (Cité en pages 29 et 34.)

A. Alan B. Pritsker, Lawrence J. Watters, and Philip M. Wolfe. Multiproject Scheduling with Limited Resources : A Zero-One Programming Approach. *Management Science*, 16(1) :93–108, 1969. ISSN 0025-1909. URL https://www.jstor.org/stable/2628369. Publisher : INFORMS. (Cité en page 14.)

Maurice Queyranne and Andreas S Schulz. Polyhedral Approaches to Machine Scheduling. *Berlin : TU*, Fachbereich 3, 1994. (Cité en pages 14 et 16.)

Hamed Rahimian and Sanjay Mehrotra. Distributionally Robust Optimization : A Review. *Open Journal of Mathematical Optimization*, 3 :1–85, July 2022. ISSN 2777-5860. doi : 10.5802/ojmo.15. URL http://arxiv.org/abs/1908.05659. arXiv :1908.05659 [cs, math, stat]. (Cité en page 26.)

Louis Rivière, Christian Artigues, and Hélène Fargier. Two-stage stochastic/robust scheduling based on permutable operation groups. *Annals of Operations Research*, 2023. URL https://hal.science/hal-04229958. Publisher : Springer Verlag. (Cité en page 68.)

R. Tyrrell Rockafellar and Stanislav Uryasev. Optimization of conditional value-at-risk. *The Journal of Risk*, 2(3) :21–41, 2000. ISSN 14651211. doi : 10.21314/JOR.2000.038. URL http://www.risk.net/journal-of-risk/technical-paper/2161159/optimization-conditional-value-risk. (Cité en page 28.)

Roy and Sussmann. Les problemes d'ordonnancement avec contraintes disjonctives. *SEMA Montrouge*, 9, 1964. (Cité en page 56.)

Russel and Norvig. *Artificial Intelligence A Modern Approach*. 2010. (Cité en page 29.)

Hosseinali Salemi and Danial Davarnia. Solving Unsplittable Network Flow Problems with Decision Diagrams. *Transportation Science*, 57(4) :937–953, July 2023. ISSN 0041-1655. doi : 10.1287/trsc.2022.1194. URL https://pubsonline.informs.org/doi/abs/10.1287/trsc.2022.1194. Publisher : INFORMS. (Cité en page 102.)

Christoph Schwindt and Jürgen Zimmermann, editors. *Handbook on Project Management and Scheduling Vol. 2*. Springer International Publishing, Cham, 2015. ISBN 978-3-319-05914-3 978-3-319-05915-0. doi : 10.1007/978-3-319-05915-0. URL https://link.springer.com/10.1007/978-3-319-05915-0. (Cité en page 10.)

Julie A Shah and Brian C Williams. Fast Dynamic Scheduling of Disjunctive Temporal Constraint Networks through Incremental Compilation. page 8, 2008. (Cité en page 130.)

Yuri N. Sotskov, Vyacheslav S. Tanaev, and Frank Werner. Stability Radius of an Optimal Schedule : A Survey and Recent Developments. In Gang Yu, editor, *Industrial Applications of Combinatorial Optimization*, Applied Optimization, pages 72–108. Springer US, Boston, MA, 1998. ISBN 978-1-4757-2876-7. doi : 10.1007/978-1-4757-2876-7_4. URL https://doi.org/10.1007/978-1-4757-2876-7_4. (Cité en pages 25 et 28.)

Arno Sprecher, Rainer Kolisch, and Andreas Drexl. Semi-active, active, and non-delay schedules for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 80(1) :94–102, January 1995. ISSN 0377-2217. doi : 10.1016/0377-2217(93)E0294-8. URL https://www.sciencedirect.com/science/article/pii/0377221793E02948. (Cité en page 14.)

Trong-Hieu Tran, Cédric Pralet, and Hélène Fargier. Combining Incomplete Search and Clause Generation : An Application to the Orienteering Problems with Time Windows. In Andre A. Cire, editor, *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Lecture Notes in Computer Science, pages 493–509, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-33271-5. doi : 10.1007/978-3-031-33271-5_32. (Cité en page 101.)

Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. A classification of predictive-reactive project scheduling procedures. *Journal of Scheduling*, 10 (3) :195–207, June 2007. ISSN 1099-1425. doi : 10.1007/s10951-007-0011-2. URL https://doi.org/10.1007/s10951-007-0011-2. (Cité en pages 31 et 33.)

Stijn Van de Vonder, Erik Demeulemeester, and Willy Herroelen. Proactive heuristic procedures for robust project scheduling : An experimental analysis. *European Journal of Operational Research*, 189(3) :723–733, September 2008. ISSN 0377-2217. doi : 10.1016/j.ejor.2006.10.061. URL https://www.sciencedirect.com/science/article/pii/S0377221706011805. (Cité en page 32.)

Marjan van den Akker, Kevin van Blokland, and Han Hoogeveen. Finding Robust Solutions for the Stochastic Job Shop Scheduling Problem by Including Simulation in Local Search. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Experimental Algorithms*, Lecture Notes in Computer Science, pages 402–413, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-38527-8. doi : 10.1007/978-3-642-38527-8_35. (Cité en pages 56, 58 et 76.)

Marjan van den Akker, Han Hoogeveen, and Judith Stoef. Combining two-stage stochastic programming and recoverable robustness to minimize the number of late jobs in the case of uncertain processing times. *Journal of Scheduling*, 21 (6) :607–617, December 2018. ISSN 1099-1425. doi : 10.1007/s10951-018-0559-z. URL https://doi.org/10.1007/s10951-018-0559-z. (Cité en page 34.)

Ari P. J. Vepsalainen and Thomas E. Morton. Priority Rules for Job Shops with Weighted Tardiness Costs. *Management Science*, 33(8) :1035–1047, August 1987. ISSN 0025-1909. doi : 10.1287/mnsc.33.8.1035. URL https://pubsonline.informs.org/doi/abs/10.1287/mnsc.33.8.1035. Publisher : INFORMS. (Cité en page 68.)

Guilherme E. Vieira, Jeffrey W. Herrmann, and Edward Lin. Rescheduling Manufacturing Systems : A Framework of Strategies, Policies, and Methods. *Journal of Scheduling*, 6(1) :39–62, January 2003. ISSN 1099-1425. doi : 10.1023/A:1022235519958. URL https://doi.org/10.1023/A:1022235519958. (Cité en page 33.)

Bing Wang, Xiaozhi Wang, Fengming Lan, and Quanke Pan. A hybrid local-search algorithm for robust job-shop scheduling under scenarios. *Applied Soft Computing*, 62 :259–271, January 2018. ISSN 1568-4946. doi : 10.1016/j.asoc.2017.10.020. URL https://www.sciencedirect.com/science/article/pii/S1568494617306257. (Cité en pages 56 et 76.)

Bing Wang, Xiaozhi Wang, and Hanxin Xie. Bad-scenario-set robust scheduling for a job shop to hedge against processing time uncertainty. *International Journal of Production Research*, 57(10) :3168–3185, May 2019. ISSN 0020-7543. doi : 10.1080/00207543.2018.1555650. URL https://doi.org/10.1080/00207543.2018.1555650. Publisher : Taylor & Francis _eprint : https ://doi.org/10.1080/00207543.2018.1555650. (Cité en pages 56 et 76.)

Christine Wei Wu, Kenneth N Brown, and J Christopher Beck. Scheduling with Uncertain Release Dates. 2005. (Cité en page 46.)

S. David Wu, Eui-Seok Byeon, and Robert H. Storer. A Graph-Theoretic Decomposition of the Job Shop Scheduling Problem to Achieve Scheduling Robustness. *Operations Research*, February 1999. URL https://pubsonline.informs.org/doi/abs/10.1287/opre.47.1.113. Publisher : INFORMS. (Cité en pages 35, 68 et 183.)

İhsan Yanıkoğlu, Bram L. Gorissen, and Dick den Hertog. A survey of adjustable robust optimization. *European Journal of Operational Research*, 277(3) :799–813, September 2019. ISSN 0377-2217. doi : 10.1016/j.ejor.2018.08.031. URL https://www.sciencedirect.com/science/article/pii/S0377221718307264. (Cité en page 34.)

Banu Çaliş and Serol Bulkan. A research survey : review of AI solution strategies of job shop scheduling problem. *Journal of Intelligent Manufacturing*, 26(5) : 961–973, October 2015. ISSN 1572-8145. doi : 10.1007/s10845-013-0837-8. URL https://doi.org/10.1007/s10845-013-0837-8. (Cité en page 21.)

Résumé :

L'objectif de cette thèse est l'étude de méthodes permettant la prise en compte d'incertitude dans des problèmes d'ordonnancements à l'aide de structures de données représentant un ensemble de solutions de façon compacte.

La première partie de cette thèse présente les problèmes d'ordonnancement sous incertitude, et introduit les méthodes de résolution dont nous ferons usage dans cette thèse.

Le seconde partie s'intéresse à des problèmes d'ordonnancement à une machine et des problèmes de jobshop, où l'incertitude porte sur les dates de disponibilité des tâches ou sur leurs durées. L'incertitude est modélisée par un ensemble de scénarios discrets, et nous supposons un modèle d'information minimaliste, dans lequel l'incertitude est levée lors du fait accompli. Nous considérons des objectifs stochastiques et robustes pour plusieurs critères réguliers, que nous utilisons pour guider la recherche de solution dans une phase hors-ligne, sur un ensemble de scénario d'entraînement ; puis nous évaluons ces solutions sur un ensemble de scénarios de test dans une phase en ligne. Nous nous intéressons particulièrement aux différent types de décisions partielles qui peuvent être prises lors de la phase hors-ligne, telles que l'heuristique "First-in First-out", guidée par les informations révélées lors de la phase en ligne, complète la décision pour obtenir un ordonnancement. Les décisions partielles que nous considérons restreignent l'espace des solutions à des ensembles de séquences de tâches. Ces ensembles sont décrits par des structures de données comme les simples séquences (comme dans la littérature en ordonnancement stochastique ou robuste), des séquences de groupes d'opérations permutables, ou des diagrammes de décision multivalués. Nous proposons plusieurs méthodes pour prendre la meilleure décision partielle hors-ligne étant donnée la décision heuristique en ligne et le type de structure considérée. Des expérimentations évaluent les différentes approches selon les objectifs et les paramètres des instances considérées et montrent l'intérêt des nouvelles approches proposées.

Enfin, le troisième partie vise à formaliser l'étude, à la lumière du formalisme de la compilation de connaissance, des différentes structures utilisées (appelées langages de représentation dans ce contexte). Dans cette partie, nous considérons une variante décisionnelle d'un problème d'ordonnancement à une machine. L'incertitude n'y est pas modélisée explicitement, mais l'objectif est de déterminer, en fonction des structures de données utilisées pour représenter l'ensemble des solutions du problème, quels aléas il est possible de prendre en compte, par exemple en remettant à jour rapidement l'ensemble des solutions face à un aléa. L'objectif étant de calculer dans une phase hors-ligne une représentation du problème, qui serve d'outil à un décideur et permette la prise en compte d'un maximum d'aléas lors de la phase en ligne. Différents langages qui représentent le problème par un ensemble de séquences de tâches correspondant à des solutions admissibles sont considérés (parmi lesquels les structures de données de la seconde partie). Les langages sont comparés selon des critères d'expressivité, de compacité, et en fonction des requêtes qu'ils supportent. Les résultats établis permettent de dresser une carte de compilation des langages

étudiés et de guider un utilisateur dans le choix d'un langage de représentation.
Les résultats de l'étude menée dans cette thèse montrent expérimentalement que le calcul de solution en amont est très difficile, mais utiliser des départs à chaud semble prometteur. Les résultats théoriques permettent de comparer plusieurs langages, mais n'identifient pas de candidat pleinement satisfaisant pour le problème considéré.
Enfin, les résultats suggèrent de nombreuses pistes de travail futures, en particulier à l'intersection des domaines de l'ordonnancement et de la compilation de connaissances, qui sont rarement considérés conjointement.
Mots clés : Ordonnancement sous incertitude, Compilation de connaissances, Proactif-réactif , Robuste/stochastique, Représentation compacte de solutions

Abstract : In this thesis, we study how data structures succinctly representing sets of sequences can be used to handle uncertainty within scheduling problems.
The first part of the thesis introduces scheduling problems and scheduling problems under uncertainty ; and describes the approaches that we will use to tackle them.
The second part takes interest in single machine scheduling problems and jobshop scheduling problems, in which uncertainty lies within release dates of tasks or on their duration. Uncertainty is modeled as a discrete set of scenarios, and we assume a minimalistic information model, in which uncertainty is lifted after the fact. We study stochastic as well as robust objectives for several regular criteria, used to guide search in an offline phase, on a training set of scenarios ; we then evaluate the solutions obtained on a testing set of scenarios in an online phase. We are specifically interested in the several types of partial decisions that can be taken during the offline phase, such that a simple "First-in First-out" heuristic, guided by information revealed during the online phase, takes the remaining decisions to reach a schedule. Partial decisions we consider restrict the solution space to sets of sequences of tasks. These sets are described by data structures such as sequences (as in the stochastic and robust scheduling literature), sequences of permutable groups, or multivalued decision diagrams. We introduce several methods to compute the best possible partial decision offline, given the heuristic decision online and the considered data structure. Experimentations evaluate the different approaches for the different objectives and instance parameters and show their relevance.
Finally, the third part aims at formalizing the study of the different data structures (called languages in that context) in the light of the tools of knowledge compilation. In this part, we consider a decision variant of a single machine scheduling problem. Uncertainty is not explicitly taken into account, but rather the goal is to find out, depending on the data structures used to represent the problem and its set of solutions, which perturbations it is possible to handle, for example by quickly updating the data structure to reflect a change in problem data. The goal is to compute in an offline phase a representation of the problem that will be used as a tool for a decision-maker and allow them to handle as much perturbation as possible in the online phase. Several languages able to represent the problem as a set of sequences of tasks corresponding to admissible solutions are considered (among which the data structures of the second part). The languages are compared along their expressivity, succinctness, and by the requests they satisfy. The results achieved allow us to draw a compilation map of the studied languages and guide a decision-maker in choosing a representation language.
The conclusions drawn in this thesis are a first step towards the formalization and usage of sets of sequences as partial decisions in the context of scheduling problems. Experimental results show that computing optimal solutions in the offline phase is very difficult, but that using warm-start strategies allows some gain over previous methods. Theoretical results allow the comparison of several languages ; however, for the considered problem, no language was found to be completely suitable.
The results of this thesis suggest numerous leads for future works, specifically at

the intersection of the domains of scheduling and knowledge compilation, two fields that are rarely considered jointly.

Keywords : Scheduling under uncertainty, Knowledge compilation, Proactive-reactive , Robust/Stochastic, Compact representation of solutions