Algorithmes exacts et approchés pour l'ordonnancement des travaux multiressources à intervalles fixes dans des systèmes distribués : approche monocritère et multiagent

Boukhalfa Zahout

UNIVERSITÉ DE TOURS

ÉCOLE DOCTORALE : Mathématiques, Informatique, Physique théorique, Ingénierie des systèmes

EA 6300 Laboratoire d'Informatique Fondamentale et Appliquées de Tours

THÈSE présentée par :

Boukhalfa ZAHOUT

soutenue le : **25 Juin 2021**

pour obtenir le grade de : **Docteur de l'université de Tours**

Discipline/ Spécialité : INFORMATIQUE

Algorithmes exacts et approchés pour l'ordonnancement des travaux multiressources à intervalles fixes dans des systèmes distribués : approche monocritère et multiagent

THÈSE dirigée par :

M. SOUKHAL Ameur	Professeur des Universités, université de Tours
M. MARTINEAU Patrick	Professeur des Universités, université de Tours

RAPPORTEURS :

M. ARTIGUES Christian	Directeur de Recherche, LAAS-CNRS, Toulouse
Mme. Kedad-Sidhoum Safia	Professeur des Universités, Conservatoire National des Arts et Métier

---

JURY :

M. ARTIGUES Christian	Directeur de Recherche, LAAS-CNRS, Toulouse
M. DELLACROCE Federico	Professeur, Politechnico di Torino, Italie **(Président du jury)**
M. DÉTIENNE Boris	Maître de Conférences, Université de Bordeaux
Mme. KEDAD-SIDHOUM Safia	Professeur des Universités, Conservatoire National des Arts et Métier
M. MARTINEAU Patrick	Professeur des Universités, Université de Tours
M. OULAMARA Ammar	Professeur des Universités, Université de Lorraine
M. SOUKHAL Ameur	Professeur des Universités, Université de Tours

Remerciements

En tout premier lieu, je remercie le bon Dieu, tout puissant, de m'avoir donné la force pour dépasser toutes les difficultés "El Hamdouli'Allah."

Je tiens d'abord à remercier mes deux directeurs de thèse : Ameur SOUKHAL et Patrick MARTINEAU, qui m'ont dirigé tout au long de la thèse. Leurs conseils ainsi que les nombreuses discussions enrichissantes ont été précieux dans la réalisation et l'aboutissement de ce travail de recherche. Qu'ils soient aussi remerciés pour leur gentillesse, disponibilité et les nombreux encouragements qu'ils m'ont prodigués.

Je tiens à exprimer mon respect et ma profonde reconnaissance à Ameur Soukhal et sa petite famille. Je le remercie pour tout ce qu'il m'a apporté tant au niveau personnel que professionnel. Je tiens à remercier également son épouse qui a sacrifiée son temps pour le bon avancement de la thèse et pour tout ce qu'elle a fait pour moi et ma famille, sans oublier ses petites filles qui ont ramené beaucoup de joie et de gaieté durant la thèse, un merci particulier pour Lyna.

J'adresse tous mes remerciements aux membres du jury qui m'ont fait l'honneur d'évaluer mon travail pour leur disponibilité, les conseils et remarques constructives. Je pense ici à :

Monsieur DELLACROCE Federico pour avoir accepté de présider le jury de thèse et pour sa bienveillance en dehors et pendant la soutenance,
Madame KEDAD-SIDHOUM Safia qui a accepté d'être la rapporteure de cette thèse, j'en suis honoré et je lui exprime toute ma profonde reconnaissance,
Monsieur ARTIGUES Christian qui a accepté d'être le rapporteur de cette thèse, je tiens à lui exprimer ma gratitude de m'avoir dirigé pendant deux stages de recherche qui m'ont donné l'envie d'aller plus loin et de connaître la richesse du métier d'Enseignant chercheur et je remercie toute l'équipe ROC du LAAS où j'ai passé des bons moments avec eux.
Monsieur OULAMARA Ammar qui a accepté de me consacrer son temps en examinant le manuscrit et de suivre mes travaux de recherche tout au long de la thèse, ses qualités pédagogiques remarquables m'ont permis de profiter de ses connaissances et ont contribué à l'avancement de mon travail en ne négligeant ni ses conseils avisés et ni ses critiques constructives,
Monsieur DETIENNE Boris pour l'honneur qu'il m'a fait pour sa participation à mon jury de thèse en qualité d'examinateur de mon travail et pour l'accueil chaleureux pendant mon séjour scientifique au sein de l'équipe de l'institut de mathématiques de Bordeaux.

Je voudrais remercier particulièrement Jean-Charles Billaut pour son aide précieuse, sa gentillesse, sa bonne humeur, le temps qui m'a consacré, les nombreux encouragements qu'il m'a prodigués et je tiens à souligner son humanisme à mon égard.

Je tiens également à remercier Emmanuel Neron, pour son intérêt permanent à mon égard et pour son soutien sur le plan humain.

Je tiens à adresser mes sincères remerciements à tous les permanents de l'équipe ROOT pour leurs soutiens mutuels, leurs bienveillances à mon égard, leurs encouragements, leurs savoirs scientifiques et leurs échanges amicaux. Un merci en particulier à Rault Tifenn, Vincent T'kindt, Yannick Kergosien et Ronan Bocquillon.

Je remercie aussi tous les permanents et les personnels du LIFAT, de Polytech'Tours et de l'école doctorale. Un merci particuliers à Yacine Sam et sa petite famille, Mohamed Slimane, Romain Raveaux, Jean-Yves Ramel, Carl Esswein, Alexis Rolland, Mathieu Delalandre, Sylvie Belair, Karine Romero, Julie Gasparini, Christelle Grange, Annie Simon, Cecile Boyer, Sébastien Beaufils, Gerald Mayaud, Bénédite Richard et Isabelle Foulon.

REMERCIEMENTS

Je remercie les anciens et futurs docteurs, je pense particulièrement à David Boas, Hugo Chevroton, Laura Echeverri Guzman, Mohamed Lemine Mohamed Habib, Olivier Ploton, Limeme Ben Ali, Meya Haroune, Romain Carletto, Alexis Robbes, Guillaume Lacharme, Kieu Diem Ho et Luong Phat Nguyen.

Mes vifs remerciements s'adressent à tous mes professeurs depuis l'école maternelle du lourd travail abattu dans mon cursus intellectuel.

Ces remerciements seraient incomplets si je n'en adressais pas à mes amis de France ou d'Algérie, j'exprime ma gratitude particulièrement à :

Abdnour Nounai mon ami d'enfance, à sa mère MAMA Timouche, à tous ses frères et soeurs et à la mémoire de son père Cheikh Hassen et son frère Arezki,
Azeddine Cheref pour tous les moments qu'on a passés ensemble, les longs débats sans fin, je tiens à remercier également son épouse pour tout ce qu'elle a fait pour ma petite famille et sans oublier ses deux petit garçons,
Baaziz Adlane pour tous les moments qu'on a passés ensemble et pour tout ce qu'il a fait pour moi,
Mounami Taibou Birgui Sekou pour tout les débats philosophiques et pour tout ce qu'il a fait pour moi,
Mostafa Darwiche, Frederic Rayar, Gaetan Galisot et Anubhav Gupta pour tous les moments qu'on a passés ensemble,
Salah Eddine Messekher, Lyes Guemit, Chems Eddine Zaidi et la promo de la Recherche opérationnelle 2011 de l'USTHB,
Mes amis de Toulouse : Timothée Renault, Nabil Lamhaddar, Charlene Ducor, Aglaé Lopez, Ania Meziani et la promo ISMAG 2016.

Mes plus profonds remerciements vont à mes très chers parents. Ils m'ont toujours soutenu, encouragé et aidé. Ils ont su me donner toutes les chances pour réussir. Qu'ils trouvent, dans la réalisation de ce travail, l'aboutissement de leurs efforts ainsi que l'expression de ma plus affectueuse gratitude.

Je remercie chaleureusement ma femme Malika qui a sacrifié ses rêves pour suivre les miens. Je tiens à la remercier pour la grande patience, l'encouragement et la confiance qu'elle m'a témoigné et surtout son soutien moral ininterrompu. Une pensée particulièrement affectueuse à la prunelle de mes yeux, ma petite fille Nelia.

Je tiens à adresser ma gratitude particulièrement à :

Ma soeur Djamila, son mari Karim ainsi qu'à leurs enfants Tarek et Farroudja Inès,
Mon frère Mustapha, sa femme Célia ainsi qu'à leurs enfants Anès et Mariya,
Mes deux derniers frères Kociela Karim et Sadek.

Un grand merci, à toute ma famille, qui porte le nom Zahout et Djouaher.

Je tiens à remercier la famille Ouaddour, particulièrement, mes beaux parents, beaux frères, ma belle soeur Rania.

Que tout ceux qui n'ont pas été cités nommément trouvent ici l'expression de mes remerciements et de ma sincère gratitude.

Je dédie ce modeste travail, à la mémoire de mes grands parents paternels et maternels, que Dieu, tout puissant, garde leurs âmes dans son vaste paradis.

Résumé

Les travaux de cette thèse proposent des méthodes d'optimisation sur des problèmes d'ordonnancement de travaux multiressources et à intervalles fixes sur machines parallèles identiques. Les "problèmes d'ordonnancement monocritère" et "problèmes d'ordonnancement multiagent" sont considérés. Nous développons un panel d'ordonnanceurs basés sur des méthodes exactes et approchées pour déterminer des solutions réalisables où l'objectif est de maximiser la somme totale pondérée des travaux ordonnancés (ou équivalent, minimisant le coût total pondérée des travaux rejetés), durant un horizon de planification. L'application des méthodes d'optimisation exactes s'avère illusoire dans la pratique en raison du temps de calcul, plus particulièrement lorsqu'il s'agit des systèmes distribués. En revanche ces méthodes exactes serviront comme références pour évaluer les méthodes approchées.

La première partie est consacrée à l'étude des problèmes d'ordonnancement monocritères. Après l'analyse de complexité, trois programmes linéaires en nombres entiers (PLNEs), un modèle basé sur la programmation par contrainte (PPC), une méthode hybride entre la PPC et un PLNE sont proposés pour résoudre à l'optimum le problème d'ordonnancement. Nous développons également une méthode exacte de type génération de colonnes basée sur la décomposition de Dantzig-Wolfe qui nous conduit à proposer un algorithme de type Branch & Price. Le Branch & Price offre de meilleures performances que les autres méthodes exactes sur des instances allant jusqu'à 150 travaux. Pour résoudre des instances de très grandes tailles, des heuristiques de liste basées sur des règles de priorité sont proposées. Afin d'éviter les choix myopes de ces algorithmes gloutons, nous introduisons une heuristique constructive PILOT combinant deux ou plusieurs règles d'ordonnancement et d'affectation ainsi qu'un algorithme évolutionnaire (AE). Les résultats expérimentaux mettent en évidence les performances de PILOT et AE.

La seconde partie de nos travaux est dédiée à l'étude de l'ordonnancement multiagent des travaux multiressources à intervalles fixes. Ce modèle considère plusieurs agents associés à des sous-ensembles de travaux disjoints, chacun d'eux cherche à maximiser la somme totale pondérée de ses travaux ordonnancés. Les trois approches suivantes sont considérées : combinaison linéaire des critères, l'approche ε-contrainte et l'énumération de l'ensemble des optima de Pareto. Après une analyse de la complexité des problèmes étudiés, des programmes dynamiques polynomiaux ont été développés pour résoudre des cas particuliers. Les fronts optimaux de Pareto sont obtenus par l'approche ε-contrainte utilisant le PLNE, la méthode hybride PPC & PLNE ou encore la méthode Branch & Price. Les résultats des expérimentations montrent que les méthodes exactes sont beaucoup moins performantes. Pour résoudre des problèmes de grande taille, des heuristiques de liste et une méthode évolutionnaire de type NSGAII sont développées. Toutes ces méthodes ont été implémentées et testées.

Mots-clefs : Recherche Opérationnelle, Ordonnancement multiagent, Machines parallèles, Travaux multiressource à intervalle fixe, Fronts de Pareto, Complexité, Programmation mathématique, Programmation par contraintes, Génération de colonnes, Branch&Price, Programmation Dynamique, Heuristiques, Algorithmes évolutionnaires.

Abstract

The work of this thesis deals with optimization methods and scheduling problems of multi-resource and fixed-interval jobs on identical parallel machines. Both "monocriterion scheduling problems" and "multiagent scheduling problems" are considered. We develop a panel of schedulers based on exact and heuristic methods to determine feasible solutions where the objective is to maximize the total weighted sum of scheduled jobs (or equivalently, minimizing the total weighted cost of rejected jobs), during a scheduling horizon. The application of exact optimization methods proves to be illusory in practice due to the computational time, especially when dealing with distributed systems. On the other hand, these exact methods will be used as references to evaluate the heuristic methods.

The first part is devoted to the study of monocriterion scheduling problems. After the complexity analysis, three integer linear programs (ILPs), a model based on constraint programming (CP), a hybrid method between CP and a ILP are proposed to solve optimally the scheduling problem. We also develop a column generation method based on the Dantzig-Wolfe decomposition which leads us to propose a Branch & Price algorithm. Branch & Price outperforms other exact methods (solves instances up to 150 jobs). To solve very large instances, heuristics based on priority rules are proposed. To avoid the myopic choices of these greedy algorithms, we design a constructive PILOT heuristic combining two or more scheduling and assignment rules and an evolutionary algorithm (EA). Experimental results highlight the performance of PILOT and AE.

The second part of our work is dedicated to the study of multi-agent scheduling problem of fixed intervals multiresource jobs. This model considers several agents associated with disjoint subsets of jobs, each of which seeks to maximize the total weighted of its scheduled jobs. The following three approaches are considered : linear combination of criteria, the ε-constraint approach and the enumeration of the set of optimal Pareto solutions. After an analysis of the complexity of the studied problems, polynomial dynamic programming algorithms have been developed to solve particular cases. The optimal Pareto fronts are obtained by the ε-constraint approach using the ILP, the hybrid PPC & ILP or the Branch & Price method. Experimental results show that the exact methods are less efficient. To solve large problems, heuristics and an evolutionary methods (NSGA-II) are developed. All these methods have been implemented and tested.

Keywords : Operational research, Multiagent scheduling, Parallel machines, Multiresource fixed jobs, Pareto fronts, Complexity, Linear programming, Constraint programming, Column generation, Branch and Price, Dynamic programming, Heuristics, Evolutionary algorithms.

Table des matières

Liste des tableaux

Table des figures

Introduction générale

Le cloud computing est de plus en plus utilisé dans le domaine des technologies de l'information (IT), en proposant de nombreux services très utilisés. L'accès et l'utilisation de ces services engendrent plusieurs problèmes, l'un des plus importants est celui de l'ordonnancement des processus. Contrairement au contexte d'une plateforme homogène composée de ressources identiques, l'étude de l'ordonnancement des tâches soumises par les utilisateurs sur des ressources hétérogènes distribuées n'est pas assez soutenue du fait que les évolutions des architectures est souvent mal comprise (Beaumont et al., 2020).

En effet, le cloud computing est basé à la fois sur des concepts informatiques et sur des concepts économiques. Le concept informatique concerne l'infrastructure utilisée pour fournir les ressources. Le concept économique, quant à lui, est en charge de l'interaction commerciale du paradigme du cloud avec les clients.

Afin de permettre aux fournisseurs de cloud une meilleure gestion de l'infrastructure, en réponse aux attentes des clients, les travaux de cette thèse proposent des méthodes d'optimisation et d'ordonnancement. Nous proposons pour le gestionnaire des ressources un panel d'ordonnanceurs basé sur des méthodes exactes et approchées. L'application des méthodes d'optimisation exactes s'avère illusoire dans la pratique en raison du temps de calcul, en revanche ces méthodes exactes serviront comme références pour évaluer les méthodes approchées.

Particulièrement, au cours de cette thèse, nous nous sommes intéressés aux problèmes d'ordonnancement des travaux à intervalles fixes multiressources sur des machines parallèles identiques. Un travail s'exécute tout le long de sa fenêtre de temps (durée opératoire), de plus chaque travail exécuté mobilise une certaine quantité de ressources disponibles en quantités limitées sur les machines physiques. Cette limitation contraint l'ordonnanceur à décider quels travaux vont être exécutés durant l'horizon de planification considéré et lesquels seront "rejetés" (définitivement ou reportés pour la prochaine période de planification). L'objectif est de trouver une solution réalisable qui maximise la somme totale pondérée des travaux ordonnancés (ou équivalent à la minimisation du coût de rejet des travaux non ordonnancés).

Nous nous sommes intéressés à deux classes de problèmes d'ordonnancement : "l'ordonnancement des travaux multiressources à intervalles fixes monocritère" et "l'ordonnancement multiagent des travaux multiressources à intervalles fixes".

Pour l'ordonnancement multiagent, nous faisons face à un problème multicritère d'af-

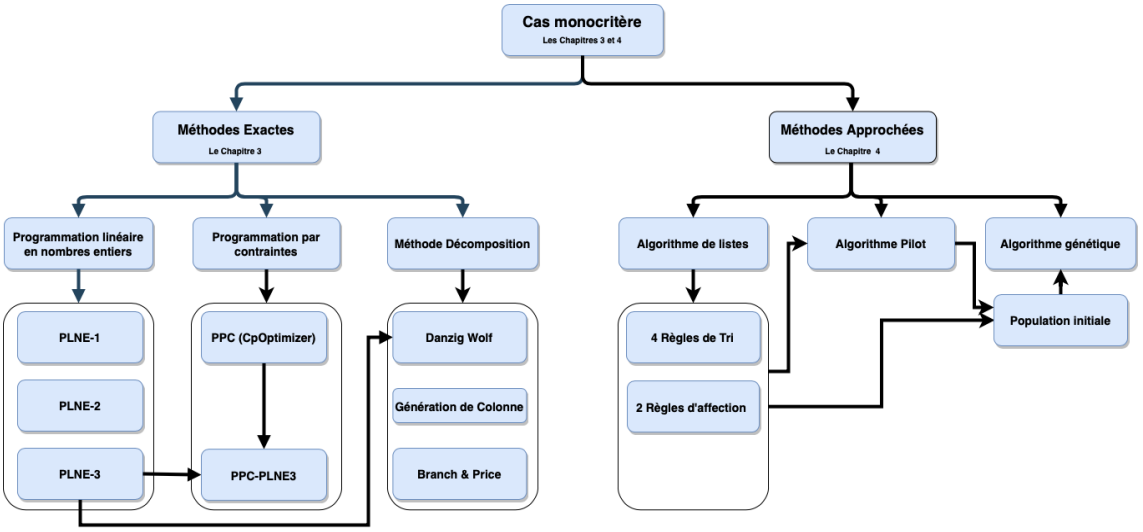
fectation des travaux sous contraintes de temps, différent du cas classique. Dans ce cas, la qualité de l'ordonnancement est mesurée par un ensemble de critères chacun est appliqué sur seulement un sous-ensemble de travaux. On dit que les agents (sous-ensembles de travaux) sont en compétition pour l'utilisation des ressources communes renouvelables et partagées. Dans notre étude, nous supposons que tous les travaux à ordonnancer sans préemption sont connus (ordonnancement hors ligne) et que les ressources sont disponibles tout le long de l'horizon de planification considéré.

Depuis l'introduction de l'ordonnancement multiagent par Agnetis et al. (2000), cette catégorie de problèmes d'ordonnancement connaît un intérêt particulier par les chercheurs, du point de vue théorique et pratique (Agnetis et al., 2014). Cependant, le premier article traitant cette problématique a été publié par (Peha, 1995) traitant un problème d'ordonnancement rencontré dans les réseaux, plus précisément, lors de la commutation de paquets dans un mode de transfert asynchrone **ATM**. Pour les réseaux ATM, dits à intégration de services, ce sont des réseaux qui transportent divers types de trafic tels que la voix, la vidéo, l'image par transfert, et divers types de données informatiques. Dans de tels systèmes, les informations portées par le réseau sont d'abord divisées en paquets de petites tailles. Ces paquets sont mis dans une zone tempo "*buffer*" en attente de transmission et un algorithme d'ordonnancement détermine l'ordre de leur transmission. Il est pertinent de classer ces types de trafic selon qu'ils disposent de contraintes sur leur délai d'achèvement (deadline, par exemple), ou qu'ils devraient être traités le plus tôt possible, i.e. sans contraintes sur le délai d'achèvement. Par conséquent, la diversité du trafic implique diverses fonctions objectifs. Par exemple, pour la plupart des types de données informatiques, les performances sont généralement mesurées en considérant le flux moyen de la file d'attente, ce qui est équivalent à minimiser la somme des dates de fin pondérées de traitement des travaux. Cependant, les paquets correspondants à la voix et à la vidéo, s'ils restent longtemps en file d'attente et n'atteignent pas leur destination à temps pour la lecture, seront perdus. L'objectif adéquat pour ces paquets est sûrement la minimisation du nombre moyen des travaux en retard pondérés.

L'organisation du manuscrit de thèse est comme suit.

Les deux premiers chapitres introductifs permettent de positionner le contexte et la problématique étudiée par rapport à l'existant. Les figures 1 et 2 résument les contributions de la thèse pour les cas monocritère et multiagent.

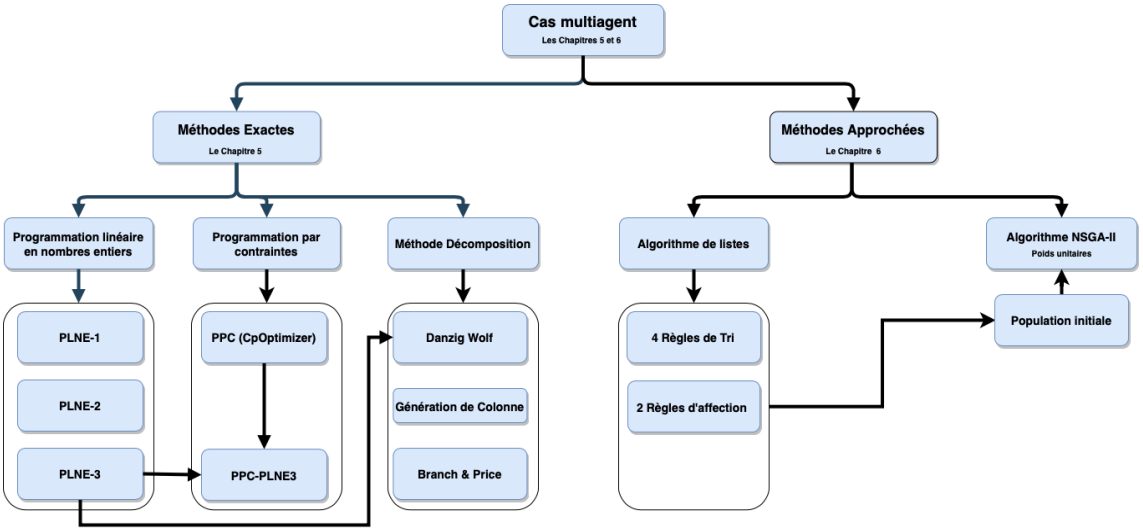
FIGURE 1 – Les contributions de la thèse pour le cas monocritère.



Le Chapitre 3 est consacré à la résolution exacte du cas monocritère du problème d'ordonnancement identifié. Nous présentons d'abord quelques définitions accompagnées des notations utilisées et analysons la complexité du problème étudié. Nous développons ensuite trois programmes linéaires en nombres entiers (PLNEs), un modèle basé sur la programmation par contrainte (PPC), une méthode hybride entre la PPC et un PLNE, où la PPC est utilisée pour générer une solution initiale réalisable. Une approche de type génération de colonnes basée sur la décomposition de Dantzig-Wolfe est proposée. Comme le problème maître de cette génération de colonnes présente un très grand nombre de colonnes, un algorithme de type Branch & Price est alors développé. Toutes ces méthodes ont été testées et évaluées.

Le Chapitre 4 est consacré à la résolution approchée des instances de très grandes tailles du problème d'ordonnancement étudié pour le cas mon-critère. Nous proposons d'abord des heuristiques de liste basées sur quatre règles d'ordonnancement et deux règles d'affectation. Ces heuristiques nous ont permis de proposer par la suite une nouvelle heuristique constructive inspirée de la méthode PILOT, en combinant deux ou plusieurs règles d'ordonnancement et d'affectation pour la construction d'une solution. Un algorithme évolutionnaire a été développé où toutes les heuristiques de liste ainsi que la méthode PILOT ont été utilisées pour générer une partie de la population initiale. L'analyse des résultats expérimentaux, montrant les performances de toutes ces méthodes approchées, clôture ce chapitre.

FIGURE 2 – Les contributions de la thèse pour le cas multiagent.



Les chapitres 5 et 6 sont dédiés à la version multicritère du problème étudié dans ce manuscrit de thèse. Plus précisément, ils focalisent sur l'ordonnancement multiagent où les travaux sont partitionnés en plusieurs sous-ensembles disjoints en compétition pour l'utilisation des ressources. Pour chaque sous-ensemble (agent), une fonction objectif est définie. Chaque agent souhaite maximiser le coût total de l'ordonnancement de ses propres travaux. Dans le Chapitre 5, nous proposons la généralisation des méthodes exactes développées pour l'ordonnancement monocritère aux cas multiagent. Nous présentons d'abord quelques définitions accompagnées des notations utilisées. Après une analyse de la complexité des problèmes étudiés, des programmes dynamiques polynomiaux ont été développés pour résoudre des cas particuliers. Dans notre étude, les trois approches suivantes sont considérées : combinaison linéaire des critères, l'approche ε-contrainte et l'énumération de l'ensemble des optima de Pareto. Les fronts de Pareto optimaux sont obtenus par l'approche ε-contrainte selon le PLNE, la méthode hybride entre la PPC et le PLNE ou encore la méthode par décomposition de type Branch & Price. Pour résoudre des problèmes avec des instances de grandes tailles, des heuristiques de listes et une méthode évolutionnaire de type NSGA-II sont développées et présentées dans le Chapitre 6. Toutes ces méthodes ont été implémentées et testées. L'analyse de leurs performances est présentée à la fin de chaque chapitre.

Une conclusion synthétique qui liste les apports principaux des travaux réalisés d'une part et identifie les problèmes qu'il reste à étudier, d'autre part.

Chapitre 1

Informatique dans les nuages : contexte et objectif de l'étude

1.1 Introduction

Le *cloud computing*, traduit le plus souvent en français par « *informatique dans les nuages* », « *informatique dématérialisée* » ou encore « *infonuagique* », est un domaine qui regroupe un ensemble de techniques et de pratiques permettant un accès à distance à du matériel ou à des logiciels informatiques, à travers une infrastructure réseau (Internet). Ce concept rend possible la distribution des ressources informatiques sous forme de services pour lesquels l'utilisateur paye uniquement pour ce qu'il utilise. Ces services peuvent être utilisés pour exécuter des applications scientifiques et commerciales. Ils sont souvent modélisés sous forme de machines virtuelles ou conteneurs.

Le cloud computing est un nouveau modèle d'entreprise et un nouveau modèle de service qui regroupe des tâches dans plusieurs centres de données informatiques (data center) différents, de sorte que toutes les applications peuvent obtenir la puissance de calcul, un accès au réseau, l'espace de stockage et les services d'information nécessaires. Le centre de données qui fournit des services est souvent appelé « *Cloud* ». Le cloud computing est considéré par les chercheurs comme le cinquième fluide d'utilité publique, après l'eau, l'électricité, le gaz et le pétrole. Faisant suite à la révolution informatique et internet, le cloud computing est considéré comme la troisième vague IT et constitue une composante stratégique importante des industries émergentes dans le monde (Wenhong and Yong (2015)).

C'est en 2009 que la réelle explosion du cloud survint avec l'arrivée sur le marché de sociétés comme Google (Google App Engine), Microsoft (Microsoft Azure), IBM (IBM Smart Business Service), Sun (Sun cloud) et Canonical Ltd (Ubuntu Enterprise cloud). Cette explosion du Cloud engendre des problèmes bien connus dans le domaine de l'informatique. Parmi les problèmes, on peut considérer le problème d'ordonnancement comme étant un problème majeur.

Ce chapitre présente une introduction au cloud computing et aux machines virtuelles ou conteneurs, nécessaire pour la compréhension générale du manuscrit.

Tout d'abord, nous présentons dans la section 1.2 un aperçu général du cloud computing, y compris sa définition, ses caractéristiques principales et une comparaison avec les technologies connexes. Nous présentons les différents modèles de service, les différents modèles de déploiement, ainsi que les différents acteurs du cloud computing. Nous résumons quelques challenges de recherche en cloud computing.

Ensuite, nous présentons, dans la section 1.3, un aperçu des différents niveaux d'ordonnancement dans le cloud et nous présentons aussi le fonctionnement global du gestionnaire de conteneur.

Finalement, dans la section 1.4, nous définissons le problème général traité dans cette thèse et introduisons des exemples illustratifs.

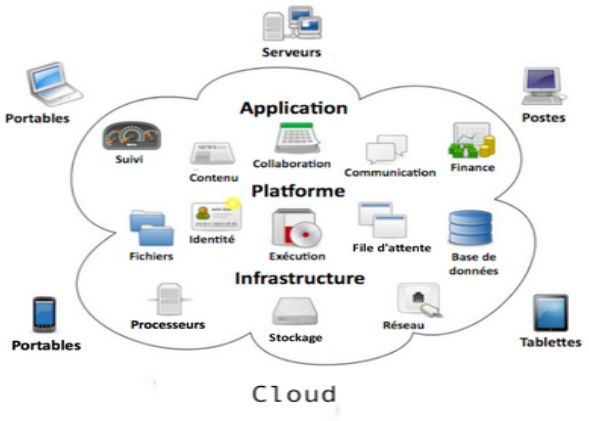
1.2 Cloud Computing

1.2.1 Définition

Le cloud computing n'a pas encore de définition claire et complète dans la littérature. Avoir une définition précise et acceptée, aidera à bien déterminer les domaines de recherche et à explorer de nouveaux domaines d'application du cloud. Pour cette raison, il y a des travaux sur la normalisation de la définition du cloud computing, à l'exemple de (Vaquero et al., 2008) qui ont comparé plus de vingt définitions différentes et ont proposé une définition globale. En guise de synthèse des différentes propositions données dans la littérature, nous introduisons une définition mixte, qui correspond aux différents types de cloud considérés dans les travaux réalisés dans cette thèse.

Nous définissons le cloud (voir la figue1.1) comme un modèle informatique qui permet d'accéder à un pool de ressources hétérogènes physiques ou virtualisées. Ces ressources sont allouées pour une utilisation optimale, sous forme de services reconfigurables dynamiquement qui permet le passage à l'échelle. Ce pool de ressources est généralement exploité par un modèle de paiement à l'utilisation dans lequel des garanties sont offertes par le fournisseur d'infrastructure au moyen d'accords de niveau de service (SLA, Service Level Agreement) personnalisés.

FIGURE 1.1 – Le cloud computing.



1.2.2 Modèles du cloud computing

Cette section présente les modèles du cloud. Nous commençons par détailler les modèles de service, puis nous donnons les modèles de déploiement et enfin nous introduisons les acteurs du cloud.

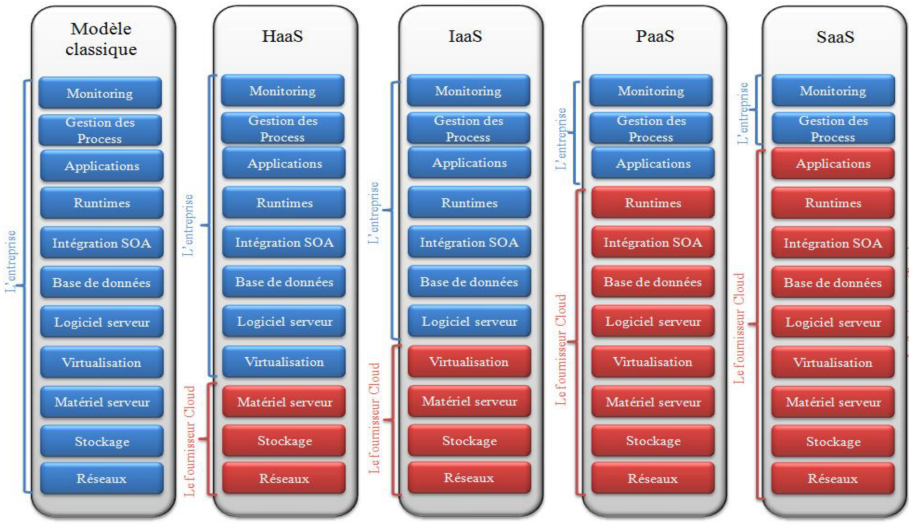
1.2.2.1 Modèles de service du cloud computing

XaaS (X as a Service) représente la base du paradigme du cloud computing, où X représente un service tel qu'un logiciel, une plateforme, une infrastructure, un Business Process, etc. Nous présentons, dans cette section, quatre modèles de services (Rimal et al., 2009), à savoir :

1. Logiciel en tant que services SaaS (Software as a Service), où le matériel, l'hébergement, le framework d'application et le logiciel sont dématérialisés,
2. Plateforme en tant que service PaaS (Platform as a Service), où le matériel, l'hébergement et le framework d'application sont dématérialisés,
3. Infrastructure en tant que service IaaS (Infrastructure as a Service) et
4. Matériel en tant que service HaaS (Hardware as a Service),

où seuls les serveurs ou les machines physiques sont dématérialisés dans les deux derniers cas. La figure 1.2 montre le modèle classique et les différents modèles de service de cloud.

FIGURE 1.2 – Les modèles de services du cloud computing.



— **Software as a Service (SaaS) :** Le SaaS communément appelé le modèle ASP (Application Service Provider), est considéré par beaucoup comme la nouvelle vague de distribution de logiciels d'application et il offre aux utilisateurs des applications sous forme de services en ligne déjà déployés dans le cloud. Il utilise des ressources communes et une seule instance du code objet d'une application ainsi que de la base de données pour prendre en charge plusieurs clients simultanément. Les utilisateurs

ne savent ni où, ni comment ces applications sont déployées. Ils ne payent pas pour les posséder, mais pour les utiliser. Ils peuvent accéder à ces applications à tout moment via internet par le biais de n'importe quel dispositif connecté, tel que les ordinateurs portables, tablettes, smartphones. Les applications sont utilisées soit directement via l'interface disponible, soit via des API fournies (souvent réalisées grâce aux Web Services ou à l'architecture REST (REpresentational State Transfer)). Les principales applications actuelles de ce modèle sont la vidéo conférence, les communications unifiées, les outils collaboratifs, la messagerie. Les principaux fournisseurs de cette catégorie sont Salesforce.com (logiciels CRM), Google (Gmail, Google Apps), NetSuite, Oracle, IBM, Microsoft, etc.

— **Platform as a Service (PaaS) :** L'idée principale est de fournir aux développeurs une plateforme comprenant tous les systèmes et environnements pour développer les applications. Il offre une plateforme entièrement configurée et gérée, sur laquelle l'utilisateur peut développer, tester et exécuter ses applications. Le déploiement des solutions PaaS est automatisé et évite à l'utilisateur d'avoir à acheter des logiciels ou d'avoir à réaliser des installations supplémentaires. Le PaaS offre une grande flexibilité, permettant notamment de tester rapidement un prototype ou encore d'assurer un service informatique sur une période de courte durée. Il favorise la mobilité des utilisateurs, puisque l'accès aux données et aux applications peut se faire à partir de n'importe quel périphérique connecté. Les principaux fournisseurs du PaaS sont SalesForce.com (Force.com), Google (Google App Engine), Microsoft (Windows Azure), Facebook (Facebook Platform).

— **Hardware as a Service (HaaS) :** Le fournisseur de cloud Haas loue essentiellement du matériel physique (par exemple serveur, stockage). Les utilisateurs accèdent au HaaS via Internet pour installer et configurer les serveurs loués. Ils ont le contrôle sur le système d'exploitation et la pile logiciel/matériel. Les communautés de recherche louent du HaaS pour leurs applications et les configurent selon leurs besoins. Par conséquent, les applications orientées calculs-intensifs et/ou traitement intensifs de données (Rimal et al., 2009), qui étaient traditionnellement exécutés sur des systèmes HPC, peuvent maintenant être exécutés dans le cloud. Parmi les principaux fournisseurs qui offrent du HaaS, on peut citer Baremetalcloud (Baremetal2014, 2014) et Softlayer (SoftLayer2014, 2014). Pour la recherche, Grid5000 (Grid2014, 2014) peut servir de support aux expérimentations à grande échelle.

— **Infrastructure as a Service (IaaS) :** L'IaaS est similaire au HaaS, mais les ressources offertes sont virtualisées. L'IaaS permet la mise à disposition des ressources d'infrastructure telles que des capacités de calcul, des moyens de stockage, du réseau sous forme de services publics. Les utilisateurs d'IaaS sont libérés des charges causées par la possession, la gestion ou le contrôle du matériel sous-jacent. Par conséquent, ils n'ont pas accès directement aux machines physiques, mais indirectement à travers les machines virtuelles (VMs). Les utilisateurs ont la plupart du temps un contrôle presque complet sur les VMs qu'ils louent : ils peuvent choisir des images de systèmes d'exploitation préconfigurées par le fournisseur, ou bien des images de machines personnalisées contenant leurs propres applications, bibliothèques et paramètres de configuration. Les utilisateurs peuvent également choisir les différents ports d'Internet à travers lesquels les machines virtuelles seront accessibles, etc. Les

utilisateurs peuvent héberger et exécuter des logiciels, des applications quelconques, ou encore stocker des données sur l'infrastructure et ne paient que les ressources qu'ils consomment.
La particularité de l'IaaS est de fournir un approvisionnement en ressources informatiques, qui soit dynamique, flexible, extensible et qui possède de bonnes propriétés de passage à l'échelle pour répondre aux besoins spécifiques des utilisateurs. Le cloud est vu ainsi comme une source infinie de ressources de calcul, de stockage et de réseau accessibles en un modèle de paiement à l'usage. Internet devient une place de marché où l'infrastructure informatique est distribuée, et consommée en tant que marchandise.
Il existe de nombreux fournisseurs de l'IaaS. Le tableau 1.1 résume quelques plateformes commerciales et open-source.

TABLE 1.1 – Fournisseurs IaaS

Fournisseurs IaaS	
Commerciaux	**Open-sources**
Amazon EC2 (Amazon2012, 2012)	Eucalyptus (Eucalyptus2014, 2014)
Rackspace (Rackspace2014, 2014)	Nimbus(Nimbus2014, 2014)
GoGrid (Gogrid2014, 2014)	OpenNebula (OpenNebula2014, 2012)
Microsoft Azure (Microsoft2014, 2014)	OpenStack(OpenStack2014, 2014)

Notre travail se concentre sur l'Infrastructure as a Service (IaaS) et le Hardware as a Service (HaaS), car ils fournissent tous les services de base nécessaires pour les applications, les processeurs, le stockage et l'accès aux services réseau à travers les machines virtuelles. En outre, l'IaaS et le HaaS sont des technologies riches en fonctionnalités et matures actuellement.

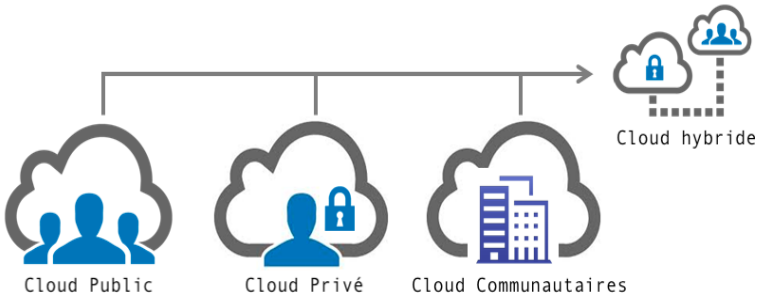
1.2.2.2 Modèles de déploiement

Selon la définition du cloud computing donnée part le NIST (Mell et al., 2011), il existe quatre modèles de déploiement des services de cloud, à savoir : cloud privé, cloud communautaire, cloud public et cloud hybride, comme illustré dans la figure 1.3.

Les travaux de cette thèse peuvent s'appliquer à ces quatre modèles de déploiement des services de cloud.

— **cloud privé :** L'ensemble des ressources d'un cloud privé est exclusivement mis à disposition pour un usage exclusif par une seule organisation comprenant plusieurs consommateurs (par exemple, des unités commerciales). Le cloud privé peut être géré par l'entreprise elle-même (cloud privé interne) ou par une tierce partie (cloud privé externe). Les ressources d'un cloud privé se trouvent généralement dans les locaux de l'entreprise ou bien chez un fournisseur de services. Dans ce dernier cas, l'infrastructure est entièrement dédiée à l'entreprise et y est accessible via un réseau sécurisé (de type VPN). L'utilisation d'un cloud privé permet de garantir, par exemple, que les ressources matérielles allouées ne seront jamais partagées par deux clients différents.

FIGURE 1.3 – Modèles de déploiement du cloud computing.



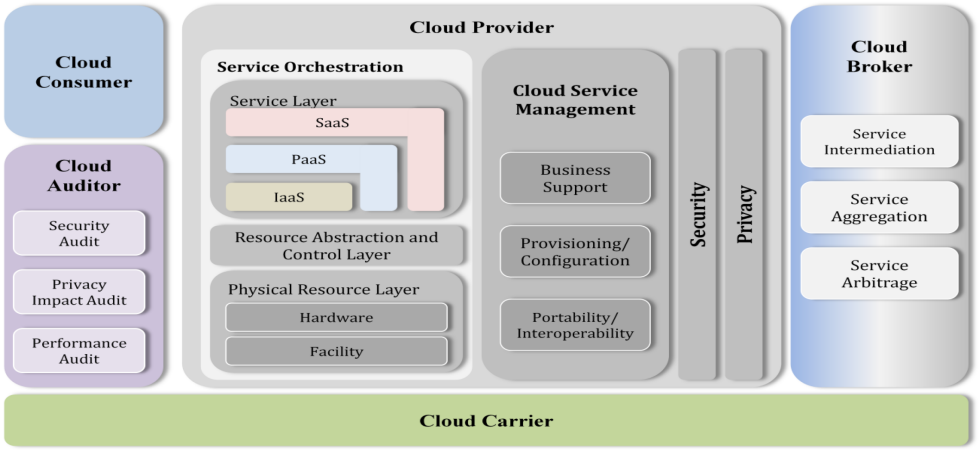
— **cloud communautaire :** L'infrastructure d'un cloud communautaire est prévue pour être utilisée exclusivement par une communauté spécifique de consommateurs issus d'organisations qui ont des préoccupations communes (par exemple, mission, exigences de sécurité, politique et considérations de conformité). Elle peut être détenue, gérée et exploitée par une ou plusieurs organisations de la communauté, un tiers ou une combinaison des deux, et elle peut être située au sein des dites organisations ou à l'extérieur, chez un fournisseur de services.
— **cloud public :** L'infrastructure d'un cloud public est accessible à un large public et appartient à un fournisseur de services. Ce dernier facture les utilisateurs selon la consommation et garantit la disponibilité des services via des contrats SLA.
— **cloud hybride :** L'infrastructure d'un cloud hybride est une composition de plusieurs clouds (privé, communautaire ou public). Les différents clouds composant l'infrastructure restent des entités uniques, mais sont reliés par une technologie standard ou propriétaire permettant ainsi la portabilité des données ou des applications déployées sur les différents clouds.

1.2.2.3 Acteurs du cloud computing

Comme le montre la figure 1.4, l'architecture de référence du NIST (Liu et al., 2011) pour le cloud computing définit cinq grands acteurs :

— **Consommateur ou utilisateur (cloud Consumer) :** une personne ou une organisation qui entretient une relation d'affaires avec des fournisseurs de services dans le cloud et qui utilise les services de ces derniers.
— **Fournisseur (cloud Provider) :** une personne, une organisation ou une entité chargée de mettre un service à la disposition des parties intéressées.
— **Courtier (cloud broker) :** une entité qui gère l'utilisation, les performances et l'approvisionnement de services, et négocie les relations entre les fournisseurs et les consommateurs.
— **Auditor (cloud auditor) :** une entité qui peut faire une évaluation indépendante des services de cloud, des opérations du système d'information, de la performance et de la sécurité de la mise en œuvre de cloud.

FIGURE 1.4 – Modèle de référence conceptuel du cloud(Liu et al., 2011)



— **Carrier (cloud carrier) :** un intermédiaire qui assure la connectivité et le transport des fournisseurs aux consommateurs de services cloud.

1.2.3 Caractéristiques du cloud computing

Dans cette section nous allons voir les principales caractéristiques du cloud computing :

— **Accès en libre-service à la demande :** Le cloud computing offre des ressources et services aux utilisateurs à la demande. Les services sont fournis de façon automatique, sans nécessiter d'interaction humaine avec chaque fournisseur de services(Mell et al., 2011).

— **Un large accès au réseau :** Les services de cloud computing sont facilement accessibles au travers du réseau par des mécanismes standard qui permettent une utilisation depuis de multiples types de terminaux (par exemple, les ordinateurs portables, tablettes, smartphones)(Mell et al., 2011).

— **Mutualisation de ressources (Pooling) :** Les ressources du fournisseur de cloud peuvent être mutualisées pour répondre aux besoins des différents utilisateurs. En fonction de leurs demandes, différentes ressources physiques et virtuelles sont affectées et réaffectées dynamiquement (Mell et al., 2011). En général, les utilisateurs n'ont aucun contrôle ou connaissance sur l'emplacement exact des ressources fournies. Toutefois, ils peuvent imposer de spécifier l'emplacement à un niveau d'abstraction plus haut.

— **Approvisionnement dynamique en ressources :** L'approvisionnement dynamique des ressources peuvent être automatiquement mises à disposition des utilisateurs en cas d'accroissement de la demande, et peuvent être libérées lorsqu'elles ne sont plus nécessaires (Zhang et al., 2010). L'utilisateur a l'illusion d'avoir accès à des ressources illimitées à n'importe quel moment, bien que le fournisseur en définisse généralement un seuil (par exemple : 20 instances par zone est le maximum possible pour Amazon EC2).

— **Paiement à l'usage :** Le cloud computing utilise un modèle de paiement à l'usage.

Le système de paiement peut varier d'un service à l'autre. Par exemple, un fournisseur SaaS peut louer une machine virtuelle auprès d'un fournisseur IaaS sur la base d'un tarif horaire. D'autre part, un fournisseur SaaS qui fournit une gestion de la relation client à la demande, peut facturer ses clients en fonction du nombre d'utilisateurs qui utilisent le service (Zhang et al., 2010). La tarification en fonction de l'utilisation réduit les coûts d'exploitation du service, car elle facture les clients à l'utilisation.

1.2.4 Technologies connexes

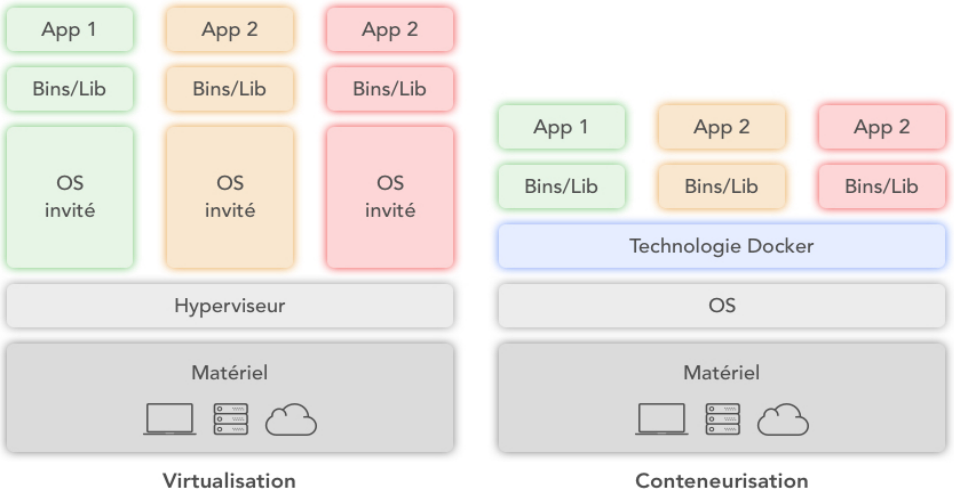
Le cloud est souvent confondu avec plusieurs paradigmes informatiques, tels que le Grid computing, l'Utility computing ...etc. Dans cete section, nous allons voir quelques technologies connexes aux clouds.

- **Grid computing :** Le grid computing est un paradigme de calcul réparti qui coordonne des ressources en réseau et géographiquement distribuées pour atteindre un objectif de calcul commun. Le grid computing a été développé pour exécuter des applications scientifiques, qui sont généralement des calculs intensifs ou des traitements de très gros volumes de données. Le cloud computing est similaire au grid computing : les deux adoptent le concept d'offrir les ressources sous forme de services. Toutefois, le cloud computing va plus loin en exploitant les technologies de virtualisation à plusieurs niveaux (matériel et plate-forme d'application) pour réaliser le partage des ressources et l'approvisionnement dynamique des ressources.
- **Utility computing (informatique utilitaire) :** L'utility computing représente le modèle consistant à fournir des ressources à la demande et à facturer les clients sur la base de l'utilisation plutôt qu'un tarif forfaitaire. Le cloud computing peut être perçu comme une réalisation de l'informatique utilitaire. Il adopte un système de tarification basé sur l'utilité, entièrement pour des raisons économiques. Avec l'approvisionnement des ressources à la demande et le paiement à l'usage, les fournisseurs de services peuvent réellement optimiser l'utilisation des ressources et minimiser leurs coûts d'exploitation.
- **la virtualisation :** La virtualisation est une technologie qui permet d'abstraire les détails du matériel physique et de fournir des ressources virtualisées pour des applications de haut niveau. En effet, la virtualisation regroupe l'ensemble des techniques matérielles ou logicielles permettant de faire fonctionner, sur une seule machine physique, plusieurs configurations informatiques (systèmes d'exploitation,...), de sorte à former plusieurs machines virtuelles, qui reproduisent le comportement des machines physiques. La virtualisation constitue le fondement du cloud computing, car elle permet de mettre en commun les ressources informatiques de grappes de serveurs et d'affecter ou de réaffecter dynamiquement des ressources virtuelles à des applications à la demande. Les ressources physique du serveur physique sont partitionnées par le moniteur de machine virtuelle, également appelé hyperviseur, en plusieurs machines virtuelles différentes, chacune fonctionnant sous un système d'exploitation distinct et une pile de logiciels utilisateur.
- **La conteneurisation :** est un type de virtualisation au niveau de l'application, qui permet de créer plusieurs instances d'espace utilisateur isolées sur un même

noyau. Ces instances sont appelées conteneurs. La conteneurisation propose une manière de virtualiser des ressources de manière légère, avec une isolation garantie par le système d'exploitation. Ces ressources sont ainsi plus facilement portables d'un système à un autre.

La principale différence entre la conteneurisation et la virtualisation réside dans les moyens mis en œuvre pour réaliser l'isolation entre le système à l'intérieur du conteneur et le reste. La figure 1.5 montre la différence d'architecture entre les deux systèmes. D'un côté, on voit que les conteneurs reposent sur le noyau du système d'exploitation qui les accueille. De l'autre, les machines virtuelles, utilisent le noyau du système d'exploitation à l'intérieur de la VM. Celui-ci n'a pas directement accès aux ressources physiques et utilise l'hyperviseur (le logiciel utilisé pour la virtualisation) pour accéder aux ressources physiques (processeur, RAM, disque dur). L'hyperviseur va émuler du matériel virtuel que la machine virtuelle utilisera au lieu d'accéder directement aux ressources présentes dans la machine. Cela ajoute une couche supplémentaire qui n'existe pas avec les conteneurs. L'isolation procurée par la conteneurisation est réalisée par le noyau du système d'exploitation hôte. Tous les conteneurs et l'hôte se partagent un même noyau Linux dans lequel des espaces de nommages sont créés.

FIGURE 1.5 – Une représentation des technologies de virtualisation et conteneurisation.



La conteneurisation et virtualisation ne rentrent pas forcément en concurrence et il est possible d'utiliser les deux technologies conjointement. Alors, conteneurs ou machines virtuelles ?

Pour faire tourner un maximum d'applications particulières sur un minimum de serveurs, il est préférable d'utiliser des conteneurs tout en gardant un œil sur les systèmes qui font tourner des conteneurs tant que leur sécurité n'est pas verrouillée. Pour exécuter plusieurs applications sur des serveurs et avoir une grande variété de systèmes d'exploitation, il est préférable d'utiliser les machines virtuelles. Et si la sécurité est la priorité numéro un d'une entreprise, alors il faut préférer pour l'instant les machines virtuelles.

Dans la pratique, il est possible d'utiliser à la fois des conteneurs et des machines

virtuelles dans le cloud et dans les centres de données. L'économie d'échelle que procurent les conteneurs ne peut être ignorée. Dans le même temps, les machines virtuelles conservent toujours leurs avantages. Alors que la technologie de conteneurs arrive à maturité, comme le dit Thorsten von Eicken, CTO de RightScale, une entreprise spécialisée dans la gestion des plateformes cloud « l'association VM/conteneurs constituera le nirvana de la portabilité cloud » (Vaughan-Nichols, 2016). Nous n'en sommes pas encore là, mais c'est vers cela que nous nous dirigeons.

Les travaux de cette thèse peuvent s'appliquer sur les deux technologies, la conteneurisation et la virtualisation. Dans le reste de ce manuscrit nous allons nous focaliser sur la conteneurisation pour plusieurs raisons, la principale étant axée sur sur l'accélération de développement d'applications. Un autre atout, toutes ou quasiment toutes les entreprises de haute technologie, investissent dans des conteneurs, par exemple : Google, IBM et Microsoft ...etc.

1.2.5 Défis de recherche dans le cloud computing

Bien que le cloud computing a été largement adopté par l'industrie, la recherche sur le domaine de cloud en est encore à ses débuts. De nombreux problèmes existants n'ont pas été entièrement résolus, tandis que de nouveaux défis continuent d'émerger (Zhang et al., 2010). Dans cette section, nous résumons certains des enjeux de la recherche dans le cloud computing :

— **Gestion de l'énergie :** L'amélioration de l'efficacité énergétique est un autre enjeu majeur dans le cloud computing. Il a été estimé que le coût de l'alimentation et le refroidissement représente 53% du total des dépenses opérationnelles de centres de données (Hamilton, 2009a). Les centres de données aux États-Unis a consommé plus de 1,5% de l'énergie totale produite en 2006 , et le pourcentage devrait augmenter de 18% par an (Li et al., 2009). Les fournisseurs d'infrastructures sont donc sous pression pour réduire la consommation d'énergie. L'objectif est non seulement de réduire le coût de l'énergie dans les centres de données, mais aussi pour respecter les réglementations gouvernementales et les normes environnementales.
La conception de centres de données efficaces sur le plan énergétique a reçu une attention considérable. Ce problème peut être abordé de plusieurs façons. Par exemple, l'efficacité énergétique des architectures matérielles qui permet de ralentir la vitesse du processeur et la désactivation partielle des composants matériels (Brooks et al., 2000), est devenue très pertinente. L'ordonnancement des tâches tenant compte de l'énergie (Vasić et al., 2009) et la consolidation des serveurs (Srikantaiah et al.) sont deux autres manières de réduire la consommation d'énergie en éteignant les machines non utilisées. Un défi majeur dans toutes les méthodes ci-dessus visent à obtenir un bon compromis entre les économies d'énergie et la performance des applications.

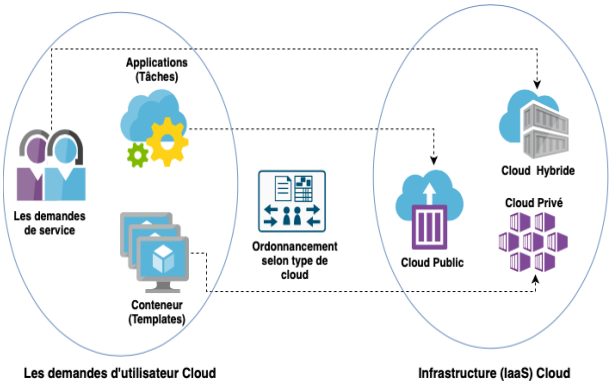
— **Sécurité de données :** La sécurité des données est un autre sujet de recherche important dans le domaine du cloud computing. Comme les fournisseurs de services n'ont généralement pas accès au système de sécurité physique des centres de données, ils doivent compter sur le fournisseur d'infrastructure pour assurer une sécurité totale des données.

— **Ordonnancement :**
L'ordonnancement est un enjeu important qui influence considérablement les performances de l'environnement de cloud computing. L'ordonnancement des tâches fait référence au processus d'ajustement des ressources entre différents utilisateurs de cloud computing, il est l'un des problèmes d'optimisation les plus connus dans littérature et joue un rôle clé dans l'amélioration du bon fonctionnement de cloud computing. Il existe plusieurs niveaux d'ordonnancement dans le cloud, notamment : l'ordonnancement au niveau application et l'ordonnancement au niveau infrastructure. Le premier consiste en l'ordonnancement (affectation) des tâches composant les applications des utilisateurs sur les services IaaS ou HaaS du cloud, et le deuxième niveau concerne l'affectation de machines virtuelles sur les infrastructures physiques (machines physiques) du cloud. Les deux niveaux d'ordonnancement sont des problèmes complexes.
Dans cette thèse, notre intérêt porte sur le deuxième niveau d'ordonnancement, où les demandes de l'utilisateur sont exprimées sous forme de conteneurs d'application.

1.3 Ordonnancements dans le cloud

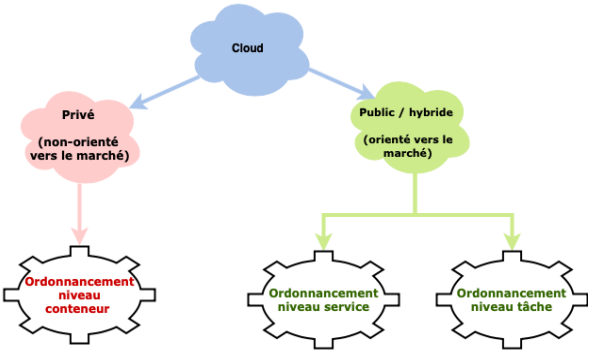
Les environnements informatiques distribués et/ou parallèle, tels que le cloud computing sont nécessairement confrontés aux problèmes d'ordonnancement qui peuvent affecter les performances du cloud de manière positive ou négative. Par conséquent, il est essentiel qu'un cloud ait un ordonnanceur efficace pour optimiser différents critères en fonction des besoins des fournisseurs et utilisateurs de cloud. Le processus d'ordonnancement peut se faire à différent niveaux du cloud computing selon la nature du service proposé (voir figure 1.6).

FIGURE 1.6 – Ordonnancement dans le cloud.



Par conséquent, selon l'orientation ou non vers le marché du service proposé par le cloud, les critères abordés et le niveau hiérarchique de l'ordonnancement ne sont pas les mêmes. Nous avons identifié trois niveaux d'ordonnancement, voir la figure 1.7, deux d'entre eux sont orientés vers le marché (ordonnancement au niveau du service et de la tâche), et le troisième n'est pas orienté vers le marché (ordonnancement au niveau de conteneur). Nous présentons ci-dessous chaque niveau d'ordonnancement dans le cloud.

FIGURE 1.7 – Différents niveaux d'ordonnancement dans le cloud.



— **Ordonnancement niveau de service :** ce niveau d'ordonnancement est statique et concerne une partie de la couche d'infrastructure pour la gestion des ressources. L'objectif de ce type d'ordonnancement est d'attribuer des grandes quantités de demandes qui représentent des services.
Dans ce type d'ordonnancement, l'architecture de cloud est composée de plusieurs clouds géographiquement distribués. L'ordonnanceur doit conaître la configuration générale de l'architecture. Une vue globale est nécessaire pour permettre à l'ordonnanceur d'assigner les processus sur les différents clouds. Un tel ordonnanceur est appelé méta-ordonnanceur et procède en trois étapes.
  — La première étape : En fonction des paramètres fonctionnels, l'ordonnanceur est responsable de trouver la collection de clouds qui correspond le mieux au service demandé sur le marché des clouds. Si aucun cloud proposant ce service n'est trouvé, il peut être demandé à l'utilisateur de télécharger son propre environnement pour faire fonctionner son service.
  — La deuxième étape : est liée à la granularité de la qualité de service demandée par les clients.
  En effet, en fonction de la qualité de service à petite ou grande échelle, le méta-ordonnanceur pourrait devoir déduire des paramètres non fonctionnels (temps, coût, ...) à partir d'un ensemble de paramètres fonctionnels.
  — La troisième étape : est introduite pour allouer le service demandé en effectuant la réservation sur le cloud approprié. Le méta-ordonnanceur peut alors estimer les valeurs obtenues pour les besoins des utilisateurs en fonction de l'affectation choisie. Le contrat entre l'utilisateur et le fournisseur est signé et représenté par l'accord sur le niveau de service (SLA). De plus, comme nous l'avons dit précédemment, cette mission est statique. Ainsi, il est nécessaire de proposer des ordonnanceurs locaux au sein de chaque centre de données (cloud) pour pouvoir s'adapter à l'évolution du système comme la volatilité des ressources et la surcharge des ressources. Il faut donc prévoir une planification dynamique, qui est offerte par l'ordonnancement au niveau des tâches.
— **Ordonnancement niveau de tâche :** l'ordonnancement au niveau des tâches concerne une couche de gestion des ressources moins importante que l'ordonnancement au niveau des services. Ce type d'ordonnancement est dynamique et s'adapte

aux changements du cloud. L'objectif est d'optimiser l'affectation des tâches aux conteneurs sous des contraintes de qualité de service de chaque tâche de chaque client, tout en minimisant le prix d'exécution global et le makespan. À l'instar du méta-orodonnanceur, l'ordonnanceur de tâches est dédié à un seul centre de données (cloud), il ne peut pas gérer les ressources des autres clouds d'autres fournisseurs. Le processus de mise en correspondance entre la tâche et la machine virtuelle est effectué par un tiers appelé courtier.
De plus, comme l'ordonnanceur au niveau des services, l'ordonnanceur au niveau des tâches comporte trois étapes précises.

- La première étape : l'ordonnanceur vise à calibrer les contraintes de qualité de service des tâches. En effet, par exemple, lorsqu'une charge de travail est planifiée au niveau du service, toutes les tâches sont envoyées ensemble vers le cloud qui est sélectionné par le méta-ordonnanceur. Cependant, ces tâches ne seront pas traitées en même temps par l'ordonnanceur au niveau des tâches. Par conséquent, de nouvelles contraintes doivent être attribuées aux tâches en fonction du temps réel où elles vont être traitées efficacement.
- La deuxième étape : consiste à optimiser le processus de correspondance entre la tâche et les conteneurs. Cela se fait de manière dynamique en fonction des besoins en ressources, de l'accord de niveau de service (SLA), du prix et de la disponibilité des conteneurs.
  Les conteneurs sont créées à la demande en les réservant au fournisseur pour répondre aux besoins précédemment cités. Le problème peut même être plus dynamique et compliqué lorsque le prix des mêmes instances de conteneurs peut changer au cours du temps. Cela conduit l'ordonnanceur à modifier l'affectation de chaque tâche en permanence en fonction du prix du conteneur et de la charge générale pour optimiser la qualité de service. En général, les objectifs visés pour la qualité de service dans ce type d'ordonnancement sont à la fois le temps de réponse global et le coût lié au traitement des tâches. L'objectif est de réduire ces deux éléments tout en offrant un profit intéressant au courtier.
- La troisième étape : consiste à mettre en œuvre un ordonnancement optimal ou quasi optimal sur l'architecture après avoir été calculé par la deuxième étape. Il est à noter que l'ordonnancement au niveau du service et l'ordonnancement au niveau de la tâche peuvent être complémentaires et faire partie du cloud orienté vers le marché.

— **Ordonnancement niveau de conteneur :** l'ordonnancement au niveau de conteneur est le niveau d'ordonnancement le plus bas, utilisé pour fournir les conteneurs qui sont demandés par la tâche dans l'ordonnancement au niveau de la tâche. Cette planification gère les conteneurs d'un cloud fonctionnant sous un gestionnaire de cloud en utilisant les possibilités de conteneurisation offertes par Docker par exemple. L'objectif est de trouver le meilleur ordonnancement des conteneurs sur les hôtes qui composent le centre de données (cloud). Ceci est fait en fonction des caractéristiques de conteneur en termes de ressources pour respecter les contraintes de faisabilité, tout en optimisant les objectifs du fournisseur de cloud. Ce type d'ordonnancement est hérité de l'infrastructure de grille qui remplace les conteneurs par des jobs. Ainsi, contrairement à l'ordonnancement au niveau des services et des

tâches, l'ordonnancement au niveau des conteneurs n'est pas une approche orientée vers le marché. En d'autres termes, il n'y a aucune référence au profit à ce niveau d'ordonnancement.
Les objectifs sont exclusivement orientés vers l'amélioration des performances en matière de consommation énergétique des centres de données et de temps de réponse des conteneurs. Le processus d'ordonnancement est réalisé en trois étapes.

— La première étape : l'ordonnanceur récupère les caractéristiques des conteneurs en fonction des besoins spécifiés (par exemple, une tâche qui nécessite un conteneur avec une certaine quantité de CPU, de mémoire et bande passante ...etc). Une fois cela fait, il filtre à la fois la liste des conteneurs et des hôtes en ne gardant que les hôtes disponibles et les conteneurs déployables.
— La deuxième étape : consiste à attribuer le pool de conteneur déployables aux hôtes disponibles. Cette affectation se fait en tenant compte des contraintes des conteneurs et des capacités des hôtes. En outre, comme indiqué précédemment, l'affectation doit être optimale ou quasi-optimale au regard des objectifs.
— La troisième étape : le meilleur ordonnancement est validé et le conteneur est déployé sur l'infrastructure. Les capacités des hôtes sont mises à jour par Docker en attendant les autres conteneurs. Notez que le problème évolue dans le temps. En effet, à chaque fois qu'un conteneur termine son traitement, de nouvelles possibilités d'affectation peuvent apparaître. De plus, il est possible de rendre le problème entièrement dynamique en déplaçant le conteneur d'un hôte à l'autre pendant qu'il est encore en cours de traitement. C'est ce que l'on appelle la migration en direct. Cependant, cette technique se heurte à certains problèmes liés à la flexibilité des applications s'exécutant sur le conteneur (interruptibilité, droits des utilisateurs, espace de travail,...), au coût du transfert de données et à la complexité du temps processeur des conteneurs. Elle entraîne également des frais généraux d'ordonnancement non négligeables qui doivent être pris en compte.

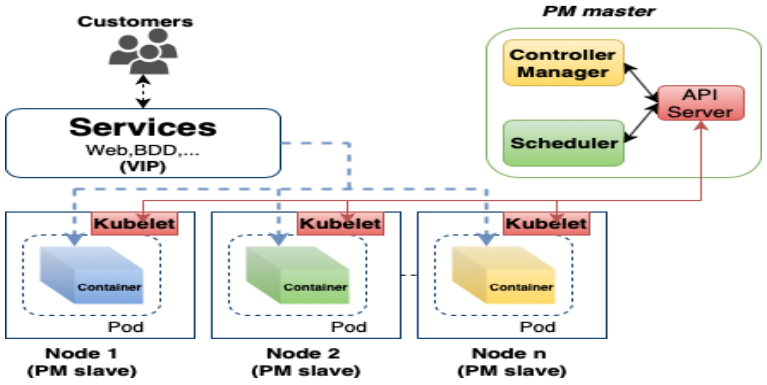
Dans ce manuscrit, nous nous intéressons à l'ordonnancement au niveau de conteneur et au niveau-service. Comme un conteneur est fortement relié au kernel, le conteneur n'a pas conscience de ce qui se passe en dehors de ce kernel et donc de la machine physique (serveur hôte), c'est là que le gestionnaire des conteneurs intervient, il va apporter l'orchestration et la gestion des conteneurs sur des clusters de serveurs, autrement dit le gestionnaire des conteneurs va prendre en charge plusieurs kernels et donc pouvoir gérer les conteneurs sur ces différents serveurs.

Avec le gestionnaire de conteneur, le développeur n'a plus à s'occuper de la gestion des machines virtuelles. Il a à disposition directement son environnement d'exécution qui est le conteneur pour pouvoir y déployer son code, et c'est le gestionnaire de conteneur qui va s'occuper des couches infrastructure sous-jacentes.

La figure 1.8 illustre l'orchestration des conteneurs. Le gestionnaire de conteneur s'exécute au dessus de l'OS (système d'exploitation) et interagit avec les pods (les pods sont les environnements d'exécution d'un ou de plusieurs conteneurs) de conteneurs qui s'exécutent sur les noeuds (les noeuds sont des machines physiques (serveurs) hébergeant les conteneurs qui exécutent les tâches qui leur sont assignées), Le gestionnaire Master (Le gestionnaire Master est le serveur maître (machine physique maître) permettant de contrôler

les noeuds) reçoit les commandes de la part d'un administrateur et va relayer ces instructions aux noeuds. Ce système de transfert fonctionne avec des services et le noeud le plus adapté pour cette tâche va être choisi automatiquement et il va ensuite allouer les ressources aux pods désignés dans ce noeud pour qu'ils effectuent la tâche requise. Lorsque le master planifie un pod dans un noeud, le kubelet (le kubelet est un composant exécuté sur des noeuds et qui s'assure que les conteneurs définis ont démarré et fonctionnent comme prévu) de ce noeud ordonne à docker par exemple de lancer les conteneurs spécifiés et c'est docker qui va démarrer ou arrêter les conteneurs. Le kubelet collecte ensuite en continu le statut de ces conteneurs via docker et rassemble ces informations sur le serveur master, on voit donc que les ordres proviennent d'un système automatisé et non d'un administrateur qui va assigner manuellement des tâches à tous les noeuds pour chaque conteneur.

FIGURE 1.8 – L'orchestration des conteneurs.



1.4 Problématique de la thèse

Le cloud computing est basé à la fois sur des concepts informatiques et économiques. Le concept informatique concerne l'infrastructure utilisée pour fournir les ressources. D'autre part, le concept économique est en charge de l'interaction commerciale du paradigme du cloud avec les clients. En effet, le concept économique ajoute des contraintes et des objectifs totalement nouveaux affectant l'ordonnancement des applications dans le cloud. En outre, la flexibilité et la variabilité de la taille d'un cloud pose des problèmes d'ordonnancement aux différents niveaux que nous avons définis dans la section 1.3.

Les travaux de cette thèse proposent un pont entre les méthodes d'optimisation et l'ordonnancement dans le cloud. Dans ce manuscrit, nous proposons un panel d'ordonnanceurs basés sur des méthodes exactes et approchées. L'application des méthodes d'optimisation exactes s'avère illusoire dans la pratique en raison du temps de calcul, en revanche ces méthodes exactes seront des références pour les fournisseurs de cloud afin d'évaluer les différentes méthodes approchées proposées. Toutes ces méthodes peuvent s'adapter aux contraintes et caractéristiques de chaque niveau d'ordonnancement défini dans la section 1.3 avec une mise à jour des critères abordés sur les différents niveaux du cloud.

Dans ce manuscrit, nous nous intéressons à l'ordonnancement au niveau-service et au niveau conteneur, en particulier à l'ordonnancement de conteneurs d'application dans le

cloud computing. Nous proposons des nouveaux modèles d'optimisation avec de nouvelles contraintes qui englobent tous les critères des différents niveaux d'ordonnancement évoqués dans la section 1.3 où les fournisseurs et les utilisateurs de services cloud ont des exigences et des objectifs à atteindre.

Comme nous l'avons déjà dit, le cloud computing semble aujourd'hui être de plus en plus adopté dans de nombreux domaines en proposant plusieurs services. Dans ce manuscrit, nous considérons comme une étude de cas, le domaine du calcul haute performance (HPC) où les infrastructures HPC sont proposées comme un service. Le principal critère à optimiser que nous avons observé pour l'ordonnancement au niveau-service (section 1.3) est le profit. Cependant, la consommation d'énergie pour le type de service proposé dans ce cas est croissante. Par conséquent, un ordonnancement tenant compte de l'énergie est crucial pour les systèmes à grande échelle qui consomment une quantité considérable d'énergie.

L'étude de (Koomey et al., 2007) montre qu'en 2005, la puissance utilisée par les serveurs représente environ 0,6% de la consommation totale d'électricité aux États-Unis. Ce pourcentage passe à 1,2% si l'on tient compte du refroidissement et des infrastructures auxiliaires. La même année, aux États-Unis, la facture d'électricité globale pour le fonctionnement de ces serveurs et des infrastructures associées s'élevait à environ 2,7 milliards de dollars et 7,2 milliards de dollars pour le monde entier. L'électricité totale consommée par les serveurs a doublé au cours de la période 2000-2005 dans le monde entier et cette augmentation s'est encore confirmée au cours de ces dernières années (Laros III et al., 2013).

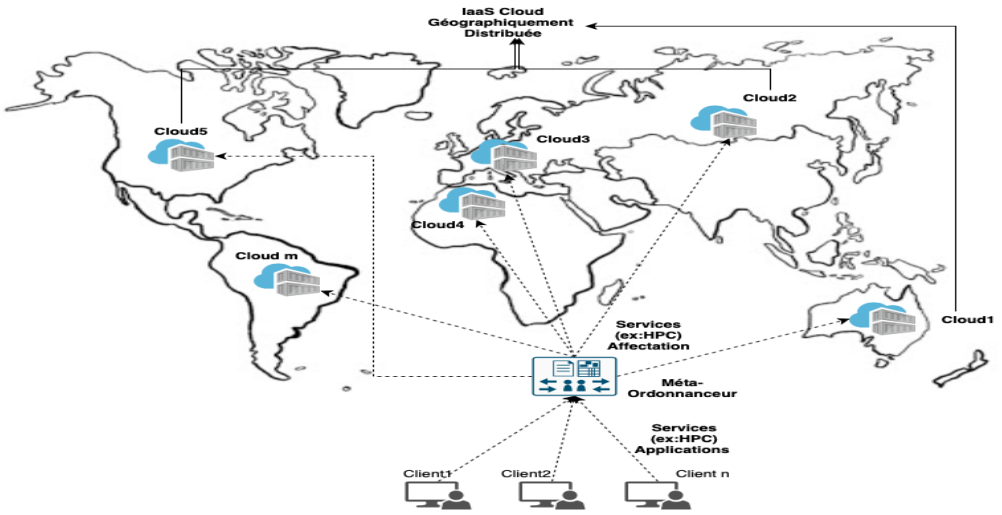
La consommation d'énergie a un impact qui affecte le profit des fournisseurs, de plus, plus une activité est gourmand en énergie, plus il génère des gaz à effet de serre et a un impact négatif sur le climat. En effet, selon l'estimation d'Amazon (Hamilton, 2007, 2009b), le montant des coûts énergétiques représente 42% du budget total d'un centre de données. Ainsi, les fournisseurs et les clients ont le souci de réduire leur consommation d'énergie, ce qui se traduit par une réduction des coûts. Donc, il est crucial de proposer des solutions efficaces pour les grands systèmes gourmands en énergie.

1.4.1 Formulation générale du problème

Le type de cloud considéré dans ce manuscrit peut être un cloud public et/ou privé du modèle de service IaaS. Plus précisément, il s'agit d'un cloud de services HPC à deux niveaux, orienté vers le marché ou non. Au premier niveau, nous trouvons le fournisseur et à l'autre niveau, les clients. Le modèle de notre architecture est inspirée du projet Open Cirrus (Campbell et al., 2009), c'est un ensemble de cloud géographiquement répartis dans le monde entier(Coady et al., 2015). Les clients ont accès au cloud en demandant des ressources au fournisseur (voir figure 1.9).

Le service proposé par le fournisseur de cloud dans notre approche offre des infrastructures aux clients afin d'exécuter des applications HPC par exemple. Comme pour l'ordonnancement de niveau de service sur un ensemble de cloud, l'ordonnanceur doit avoir une vue globale (méta-vue) de l'ensemble de clouds. Ainsi, l'originalité de ce modèle consiste à développer des algorithmes afin de trouver le meilleur méta-ordonnancement pour les requêtes dans le temps.

FIGURE 1.9 – Une représentation de notre modèle cloud à deux niveaux.



Le modèle de cloud à deux niveaux que nous abordons est géographiquement distribué. Nous nous intéressons plus particulièrement au modèle Infrastructure as a Service (IaaS). En effet, nous avons affaire dans notre cas à un modèle avec, d'un côté le fournisseur du cloud distribué et de l'autre côté, les clients. Ces derniers ont accès au Cloud en demandant des ressources au fournisseur. Le service offert par le fournisseur du Cloud consiste en des infrastructures (ressources) pour permettre aux clients d'exécuter leurs applications. Cependant, malgré les nombreux avantages avérés de l'utilisation d'une infrastructure en nuage pour l'exécution des processus d'entreprise, elle est toujours confrontée à un problème majeur qui peut compromettre son succès : le manque de conseils pour choisir entre plusieurs offres. De plus, lors de l'exécution des processus métier, il est difficile d'automatiser toutes les tâches et plusieurs objectifs souvent contradictoires doivent être pris en compte (Bessai et al., 2013). Bessai et al. (2013) proposent trois approches complémentaires bi-critères pour l'ordonnancement des processus métier sur les ressources distribuées du Cloud tout en tenant compte de sa caractéristique de calcul élastique qui permet aux utilisateurs d'allouer et de libérer des ressources de calcul (machines virtuelles) à la demande et de son modèle économique basé sur le paiement à l'utilisation.

Notre étude aide le fournisseur à optimiser l'utilisation des ressources tout en offrant le meilleur ordonnancement des applications pour assurer la meilleure qualité de service (QoS) en respectant les contraintes du modèle et les contraintes spécifiées par les clients, telles que la date de début et de fin d'exécution des applications définies par le client, le budget alloué exprimé en kilowatts par heure (Kwh). Aujourd'hui, le fournisseur fixe un prix d'allocation des ressources par client en Kwh. Les clients peuvent donc estimer leur facture en fonction de leurs besoins en ressources selon, par exemple, la formule suivante (Ishihara and Yasuura, 1998) :

$$\text{Kwh} = \text{Puissance du CPU (en KW)} \times \text{Durée (en h)} \tag{1.1}$$

Avec la puissance du CPU est égale à $(C \times fr \times V)$, où C est la capacité de commutation,

fr est la fréquence du CPU et V est la tension d'alimentation.

Avec une telle modélisation, un client peut soumettre sa demande avec les informations suivantes : une date de début s, une date de fin f, la quantité d'énergie estimée pour sa demande. Bien entendu, il peut également spécifier d'autres besoins pour son application comme la quantité de mémoire nécessaire, la quantité de stockage, ou la bande passante par exemple. En outre, les applications n'ont pas nécessairement la même priorité. Cette priorité est souvent attribuée par le fournisseur de ressources informatiques (en fonction de l'importance du client / de l'application) ou automatiquement par le système.

Plus formellement, nous considérons le cas où nous avons un fournisseur de Cloud avec un certain nombre de centres de données répartis géographiquement sur différents sites dans le monde. Chaque centre de données est considéré comme une méta-machine qui possède principalement types de ressources : l'énergie, une quantité limitée de mémoire, une quantité limitée de stockage et une quantité limitée de bande passante, etc. En outre, chaque centre de donnée est soumis à une consommation d'énergie maximale à ne pas dépasser, à savoir le *Power Capping* exprimé en Kwh. Le plafonnement de la puissance représente donc la limite maximale de la consommation d'énergie d'un système. Dans ce contexte, les clients soumettent donc des demandes (applications).

Les applications doivent donc être traités. Chaque application sera affecté à une machine (noeud) à exécuter. Dans cette étude, les applications doivent être ordonnancés sans préemption pendant un horizon de temps d'ordonnancement fixé à l'avance. Nous supposons que les machines sont disponibles en permanence. Ainsi, la soumission d'une demande par l'utilisateur pour exécuter son application est définie par le vecteur (Début, fin, poids, RAM, Bandwidth, EC). Les éléments du vecteur représentent respectivement la date de début, la date de fin de l'exécution de l'application, son poids, la quantité de mémoire requise par l'utilisateur, la bande passante et la consommation d'énergie limitée par unité de temps. *Durée = date de fin - date de début* est le temps d'exécution d'une application.

Fixer la date de début, la date de fin, la fréquence du CPU ainsi que des besoins en ressources d'une application est aujourd'hui de plus en plus fréquent. L'objectif principal de cette étude est de rechercher un sous-ensemble maximum d'application qui peuvent être ordonnancés sur l'horizon de temps tout en respectant toutes les contraintes du système et les spécifications des clients. Le problème d'ordonnancement est donc un problème multi-ressources à intervalle fixe (`FISP (Fixed Interval Scheduling Problem)`). Nous reviendrons plus en détail dans le chapitre 2 sur ce problème `FISP` .

Exemple cas monocritère

Prenons un exemple avec 8 applications doivent être ordonnancés sur 2 machines (sur le même ou différents centres de données) sur un horizon de temps $[0, 10]$. 3 types de ressources sont nécessaires, par exemple : RAM, Bandwidth et power capping (un plafonnement de l'énergie), noté par EC. Les quantités de ces ressources sont limitées : $(RAM, Bandwidth, EC) = (2, 2, 2)$. Pour chaque applications , la date de début, la date de fin, le poids de priorité pour chaque application et les besoins en ressource sont données dans le tableau 1.2.

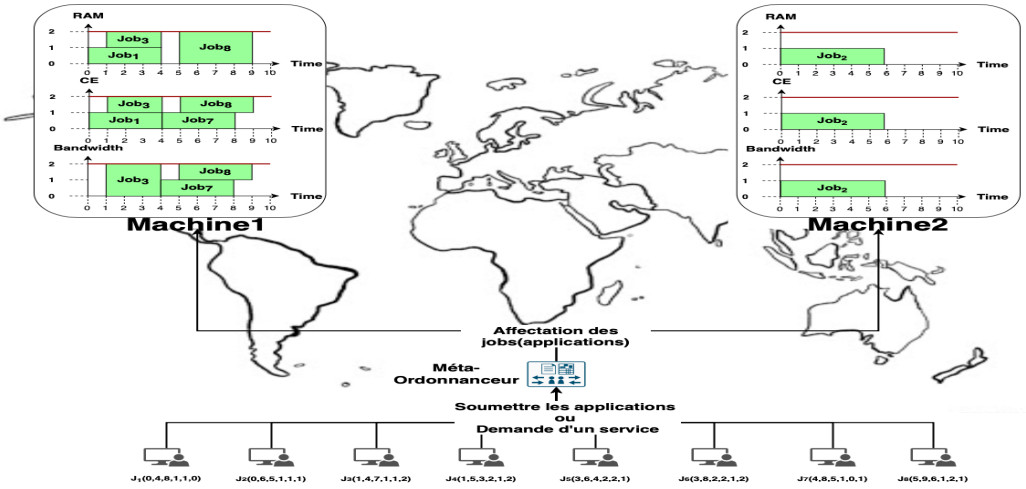
La figure 1.10 présente une solution optimale pour le problème mono-critère, représenté

TABLE 1.2 – Une instance de 8 applications et 3 types de ressources (cas monocritère).

Applications	Date de début	date de fin	Poids	EC	RAM	$Bandwidth$
1	0	4	8	1	1	0
2	0	6	5	1	1	1
3	1	4	7	1	1	2
4	1	5	3	2	1	2
5	3	6	4	2	2	1
6	3	8	2	2	1	2
7	4	8	5	1	0	1
8	5	9	6	1	2	1

avec le diagramme de Gantt sur chaque machine, pour l'exemple ci-dessus, la valeur de la fonction objectif est de : $Z = 31$ (la somme des poids des applications ordonnancés $1, 2, 3, 7, 8$), les applications $4, 5, 6$ ne seront pas ordonnancées (pas assez de ressources), elles seront ordonnancés plus tard.

FIGURE 1.10 – Une solution optimale monocritère.



De plus, nous traitons le cas où différentes fonctions objectives doivent être optimisées. Nous sommes donc confrontés à des problèmes d'ordonnancement multicritères. Les problèmes classiques d'ordonnancement multicritères ont été largement étudiés dans la littérature (voir (T'kindt and Billaut, 2006; Hoogeveen, 2005)) où l'on suppose que la qualité de la solution est mesurée par une ou plusieurs fonctions objectives appliquées à tous les travaux sans distinction. Dans certaines situations réelles, cette hypothèse peut être écartée ou ne pas être appropriée au problème plus général de l'allocation des ressources. Au lieu d'utiliser un ou plusieurs critères pour tous les travaux, il peut être nécessaire de considérer que chaque fonction objectif dépend d'un sous-ensemble de travaux. Par exemple, le cas de notre étude. Il s'agit d'un problème d'ordonnancement multicritères pour lequel un nouveau type de compromis doit être trouvé. Ces problèmes d'ordonnancement sont appe-

lés dans la littérature "ordonnancement multiagents" (Cheng et al., 2006; Kovalyov et al., 2012) ou "ordonnancement avec des agents concurrents" (Agnetis et al., 2009a) ou "problèmes d'ordonnancement de travaux interférents" (Hoogeveen, 2005; Tuong et al., 2012), Nous reviendrons plus en détail dans le chapitre 2 sur ces problème d'ordonnancement multiagent.

Nous définissons le terme "agent" comme une entité associée à un sous-ensemble de travaux. Cette entité peut être associée à un autre décideur qui intervient dans le choix de la solution finale. Chaque agent vise à chercher un sous-ensemble maximum de travaux qui lui est propre, et qui peuvent être ordonnancés sur l'horizon de temps tout en respectant toutes les contraintes du système et les spécifications des clients. Ces agents sont en concurrence puisqu'ils partagent les mêmes ressources (Agnetis et al., 2014, 2000). Nous recherchons donc des solutions de compromis. Ces problèmes sont proches de l'optimisation combinatoire et de la théorie des jeux coopératifs (Agnetis et al., 2009b).

Dans ce manuscrit, pour illustrer nos approches, nous traitons le cas de deux agents. Toutes les approches développées peuvent être généralisées à plusieurs agents.

Exemple cas mulicritère (multiagent)

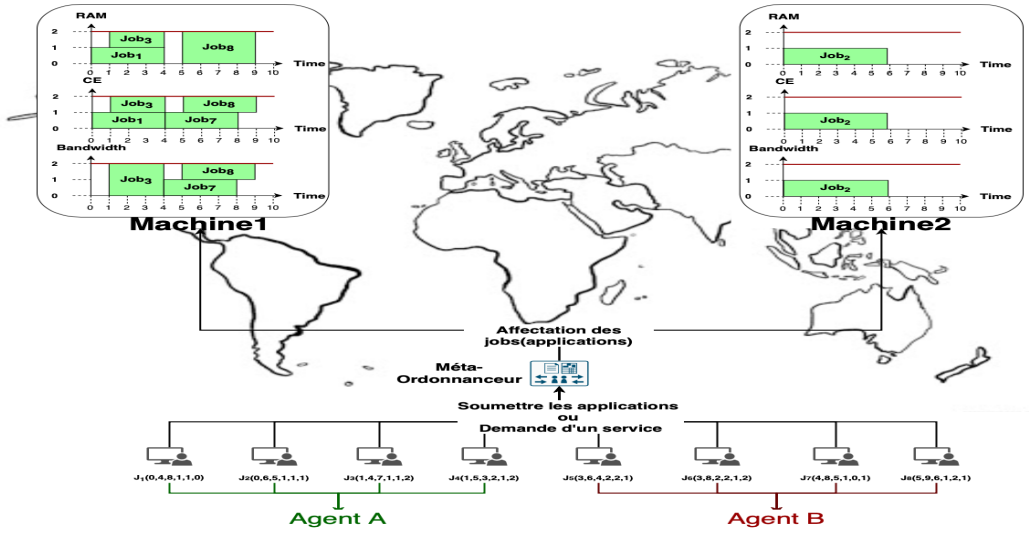
Prenons un exemple avec 8 applicationss doivent être ordonnancés sur 2 machines (sur le même ou différent centre de donnée) sur un horizon de temps $[0, 10]$. On considère deux agents ($Application_A = 1, 2, 3, 4$ et $Application_B = 5, 6, 7, 8$). 3 types de ressources sont nécessaires, par exemple : RAM, Bandwidth et power capping (EC). Les quantités de ces ressources sont limitées : $(RAM, Bandwidth, EC) = (2, 2, 2)$. Pour chaque application, la date de début, la date de fin, le poids de priorité pour chaque job et les besoins en ressources sont données dans le tableau 1.3.

Table 1.3 – Une instance de 8 Applications et 3 types de ressources (cas multiagent).

Applications	Date de début	date de fin	Poids	EC	RAM	$Bandwidth$
1	**0**	**4**	**8**	**1**	**1**	**0**
2	**0**	**6**	**5**	**1**	**1**	**1**
3	**1**	**4**	**7**	**1**	**1**	**2**
4	**1**	**5**	**3**	**2**	**1**	**2**
5	3	6	4	2	2	1
6	3	8	2	2	1	2
7	4	8	5	1	0	1
8	5	9	6	1	2	1

La figure 1.11 présente une solution optimale pour le problème multiagent, représenté avec le diagramme de gantt sur chaque machine, pour l'exemple ci-dessus, la valeur de la fonction objectif pour l'agent A est de : $Z_A = 20$ (pour l'agent B est de : $Z_B = 11$).

FIGURE 1.11 – Une solution optimale multiagent (2 agents).



1.5 Conclusions du chapitre

Dans ce chapitre, nous avons présenté tous les concepts et les éléments et hypothèses nécessaires à la compréhension générale de nos contributions. Nous avons exploré divers défis de recherche dans le cloud. Un de ces défis porte sur l'ordonnancement de conteneurs d'application, qui présente un élément essentiel des systèmes de gestion de conteneurs. Nous avons proposé un nouveau modèle d'optimisation avec de nouvelles contraintes qui englobe tous les critères des différents niveaux d'ordonnancement évoqués précédemment où les fournisseurs et les utilisateurs de services cloud ont des exigences et des objectifs à atteindre.

Chapitre 2

Ordonnancement sur des ressources cumulatives et disjonctives : État de l'art

2.1 Introduction

Les problèmes d'ordonnancement sont des problèmes d'optimisation combinatoire dans lesquels certaines activités doivent être exécutées en utilisant les ressources dont elles ont besoin. Une allocation faisable des ressources aux activités dans le temps est appelée un ordonnancement. La qualité d'un ordonnancement est mesurée par divers critères d'optimisation qui sont en général des fonctions de délais d'exécution des activités et des quantités de ressources utilisées.

Dans un problème d'ordonnancement traditionnel, il est possible de déterminer librement les dates de début des activités. L'ordonnanceur utilise cette liberté pour construire un ordonnancement tout en satisfaisant certaines contraintes ou même en optimisant un certain critère. Concernant le problème d'ordonnancement traité dans cette thèse, l'ordonnanceur n'a pas de liberté pour déterminer la date de début d'une activité ; c'est plutôt une donnée du problème.

Dans la section 2.2 de ce chapitre, nous dressons quelques travaux connexes des problèmes d'ordonnancement des travaux à intervalle fixe, également connus sous le nom de problèmes d'ordonnancement des travaux fixes. Dans la section 2.5.2 nous rappelons les définitions des problèmes d'ordonnancement multiagents, ainsi un certain nombres de notions issues de l'optimisation multicritère, nécessaires pour nos travaux. Dans la section 2.3 nous présentons les problèmes d'ordonnancement dans un contexte *cloud computing* qui est actuellement de plus en plus utilisé dans le domaine des technologies de l'information (IT). Ce chapitre s'achève par la section 2.6 présentant une conclusion.

2.2 Problème d'ordonnancement d'intervalles fixes

Les problèmes d'ordonnancement sont formulés en termes de machines et de travaux. Dans notre étude, les machines dotées d'un certain nombre de ressources renouvelables doivent exécuter les tâches. Dans un problème d'ordonnancement traditionnel, il est possible de déterminer librement les dates de début des jobs, qui sont des variables de décisions. L'ordonnanceur utilise cette liberté pour construire un ordonnancement tout en satisfaisant certaines contraintes ou même en optimisant un ou plusieurs critères. Les mesures de performance pour ce type de problème d'ordonnancement reflètent la qualité de la solution et donc de l'utilisation des ressources. Par exemple, la minimisation de la durée de totale de traitement ou de production (makespan) mesure le temps nécessaire aux machines pour traiter tous les travaux. On trouve dans la littérature de la recherche opérationnelle plusieurs travaux sur ces problèmes d'ordonnancement classiques (Brucker, 2004; Leung, 2004).

Les décisions auxquelles l'ordonnanceur doit faire face sont le traitement ou non du travail et l'allocation de la (des) ressource(s) nécessaire(s) à son exécution. Les mesures de performance peuvent se focaliser sur les travaux ; par exemple, on peut souhaiter maximiser le coût total pondéré des travaux traités (acceptés ou le bénéfice global du traitement de certains travaux), ou équivalent le coût total pondéré de rejet de certains travaux (la perte globale totale liée au rejet de traitement de certains travaux). Bien qu'il existe plusieurs travaux dans la littérature sur les problèmes d'ordonnancement à intervalles fixes, les recherches sur ces problèmes ont souvent été adaptées à une grande diversité d'applications. Par exemple, dans Fischetti et al. (1989) les auteurs s'intéressent à la minimisation du nombre de chauffeurs de bus nécessaires pour effectuer toutes les tâches, quand une équipe ne peut pas travailler plus qu'un temps de travail donné T dans une journée.

2.2.1 Description du problème de base de l'ordonnancement des intervalles fixes et ses variantes

Le problème de base d'ordonnancement d'intervalles fixes peut être décrit comme suit : les données sont n intervalles fixes $[s_i, f_i)$ avec $s_i < f_i \, , \forall i \ldots n$. Chaque intervalle fixe correspond au temps d'exécution d'un travail sur toute la durée de l'intervalle. La préemption des travaux n'est pas autorisée. Nous supposons que s_i et f_i sont des entiers non négatifs. Deux intervalles (ou travaux) se chevauchent si l'intersection est non vide, sinon ils sont dits disjoints. En outre, il existe m machines. Dans le cas classique (problème de base), chaque machine peut traiter au maximum un travail à la fois et elle est toujours disponible, i.e. chaque machine est disponible en permanence dans $[0,\infty)$. Nous supposons que chaque travail est affecté à une seule machine. Le problème de base consiste ainsi à traiter tous les travaux en utilisant un nombre minimum de machines. En d'autres termes, un ordonnancement est l'affectation des travaux aux machines, i.e une affectation de travaux sur un ensemble minimum de machines de sorte que les intervalles de temps de deux travaux affectés à la même machine ne se chevauchent pas.

Il est bien connu que les problèmes d'ordonnancement des intervalles fixes peuvent être modélisés par des graphes d'intervalles. En effet, un graphe dit d'intervalles se construit comme suit : on a un nœud pour chaque intervalle et un arc entre deux nœuds si et

seulement si les intervalles correspondant se chevauchent. Par conséquent, le problème de base de l'ordonnancement des intervalles fixes correspond au problème de coloration d'un graphe d'intervalles (cf par exemple (Golumbic, 1980)) ; le nombre chromatique d'un graphe d'intervalles correspond au nombre minimum de machines utilisées dans le contexte d'ordonnancement des intervalles fixes.

Par définition des graphes d'intervalles, l'ensemble des travaux peut être partitionné en sous-ensembles de travaux où un sous-ensemble est défini par les travaux qui se chevauchent. Dans le cas où la taille de ces sous-ensembles est de deux, on parle de chevauchement par paire. Il est trivial que la taille de cet ensemble est une borne inférieure pour le nombre de machines nécessaires pour traiter tous les travaux.

Pour déterminer un ordonnancement optimal minimisant le nombre de machines à utiliser, nous pouvons utiliser l'algorithme de Ford et Fulkerson dit d'*escalier* (Ford and Fulkerson, 1962). Cet algorithme est basé sur le théorème de Dilworth avec une complexité temporelle en $O(n^2)$, où n est le nombre de travaux. Gupta et al. (1979) proposent un autre algorithme avec une meilleure complexité en $O(n \log n)$.

La littérature traitant l'ordonnancement d'intervalles fixes est riche par les modèles et méthodes de résolution. Selon les critères et paramètres considérés et les études menées, nous pouvons définir principalement quatre variantes :

1. Ordonnancement d'intervalles fixes de travaux à exécuter : il s'agit d'une classe de problèmes où tous les travaux doivent être exécutés, i.e. pas de rejet. Dans ce cas, le nombre de machines est une variable de décision. L'objectif est de trouver un ordonnancement qui minimise le coût total d'exécution des travaux, et principalement le nombre de machines utilisées.

2. Ordonnancement d'intervalles fixes avec nombre de machines fixé : contrairement au cas précédent, ici le nombre de machines est une donnée du problème et il n'est donc pas nécessaire d'affecter tous les travaux aux machines. Dans cette classe de problèmes, si une machine i exécute un travail j un profit $c_{i,j}$ est dégagé. L'objectif est donc de trouver un ordonnancement à profit maximum.

3. Ordonnancement d'intervalles fixes discrets : pour cette classe de problèmes d'ordonnancement, au lieu d'une seule date de début s_j pour chaque travail j, un ensemble discret d'heures de début d'exécution de chaque travail est fourni, i.e. $s_j \in \{s_{j_1}, s_{j_2}, \ldots, s_{j_K}\}$. On peut rencontrer ces problèmes dans certains centres de consolidation des commandes et de livraison où le client indique ses préférences de livraison. Le manager du système doit lisser les ordres de préparation sur l'horizon du temps considéré (généralement, la semaine).

4. Ordonnancement d'intervalles fixes en ligne où la préemption est autorisée : lorsque la préemption est autorisée, certains travaux traitent le cas où la réaffectation du reste du travail à réaliser sur une autre machine est interdite en se justifiant par la perte du rendement, par exemple. Des critères d'optimisation tels que la minimisation des encours, l'équilibrage de charge ou encore la satisfaction du client (respect du contrat de service) peuvent être considérés.

Nous allons reprendre chaque classe de problèmes d'ordonnancement citée ci-dessus et indiquer - à notre connaissance - les principaux travaux correspondants.

Le problème de base d'ordonnancement d'intervalles sur machines identiques appartient à la première classe. Ces machines peuvent avoir des coûts d'exploitation et/ou de disponibilité différents.

Citons par exemple les travaux de Kolen and Kroon (1991), chaque machine $i, 1 \leq i \leq m$ est disponible dans un intervalle $[a_i, b_i)$ avec un coût d'utilisation ct_i. Pour chaque intervalle $[a_i, b_i)$, une seule machine est disponible. On parle d'ordonnancement d'intervalles fixes avec disponibilités des machines (Kolen and Kroon, 1991, 1994).

Bhatia et al. (2003) généralisent les travaux de Kolen and Kroon (1991) en étudiant une autre variante où pour chaque intervalle de disponibilité $[a_i, b_i)$, un nombre illimité de machines de type i sont disponibles. Chaque machine de type i utilisée a un coût ct_i et l'objectif est de minimiser le coût total pour l'exécution de tous les travaux. Les auteurs ont proposé un algorithme 3-approché et un algorithme 2-approché dans le cas où l'objectif est de minimiser le nombre de machines utilisées, c'est-à-dire dans le cas où $ct_i = 1$ pour tous les i).

La deuxième classe d'ordonnancement d'intervalles fixes où le nombre de machines est donné traite principalement de la maximisation du nombre de travaux (pondérés ou non) à ordonnancer. Cette variante peut être résolue par la recherche d'un flot à coût minimum (voir (Arkin and Silverberg, 1987; Bouzina and Emmons, 1996). Dans le cas où chaque travail a un poids unitaire, Faigle and Nawijn (1995); Carlisle and Lloyd (1995) ont proposé un algorithme glouton pour trouver le nombre maximum de travaux à ordonnancer. Par contre, si les machines ne sont pas toutes qualifiées pour traiter un travail donné (un travail peut être exécuté sur un sous-ensemble de machines), le problème devient $\mathcal{NP}$-difficile (Arkin and Silverberg, 1987). Nous pouvons dans ce cas nous référer aux travaux de Kroon et al. (1995) qui ont proposé des heuristiques et des méthodes exactes pour résoudre ce dernier problème.

Si chaque machine i, $1 \leq i \leq m$ est disponible dans un intervalle $[a_i, b_i)$, Brucker and Nordmann (1994) proposent un algorithme de programmation dynamique de complexité $O(n^{m-1})$ permettant de maximiser le nombre de travaux ordonnancés. (Bhatia et al., 2003) proposent un algorithme randomisé avec une approximation de ratio $1 - 1/e$ pour l'ordonnancement des travaux avec un profit donné $w_j, j = 1, \ldots, n$.

La troisième classe de problèmes constitue une généralisation intéressante des problèmes d'ordonnancement d'intervalles fixes puisque chaque travail j est défini entre autres par un ensemble discret de dates de début d'exécution possibles. Bien entendu, on peut choisir au plus une date de début dans chaque ensemble S_j. Les problèmes d'ordonnancement de cette classe sont liés aux problèmes d'ordonnancement à contraintes de temps (*Time-Constrained Scheduling Problems* "TCSP") où chaque travail j a une date de début au plus tôt (*release date*) r_j, une date de fin souhaitée (*due date*) d_j et une durée opératoire p_j. Si pour chaque travail j, $d_j = r_j + p_j$, un intervalle de temps d'exécution pour chaque réalisation possible de j est considéré. Nous pouvons ainsi modéliser un problème d'ordonnancement à contraintes de temps comme un problème d'ordonnancement à intervalles fixes discrets. Cette problématique est confrontée à la difficulté de traiter un ensemble d'intervalles pour toutes les dates de début possibles d'un travail (notez que le mot "travail" ici fait référence à un ensemble d'intervalles). Nous pouvons citer par exemple les travaux de Berman and DasGupta (2000); Chuzhoy et al. (2004) proposant de résoudre par des méthodes exactes

et approchées pour ce type de problèmes d'ordonnancement.

Plus généralement, des résultats de complexité pour les problèmes d'ordonnancement d'intervalles fixes discrets sont introduits dans (Nakajima and Hakimi, 1982; Nakajima et al., 1982). Berman and DasGupta (2000) ont proposé un algorithme d'approximation pour traiter le cas pondéré du problème d'ordonnancement d'intervalles fixes discrets, et montrent comment cela peut être utilisé pour déduire des paramètres d'approximation pour des problèmes d'ordonnancement à contraintes de temps sur machines identiques et non-identiques, en améliorant l'approche basée sur la programmation linéaire décrite par (Bar-Noy et al., 2001).

La quatrième classe porte sur les problèmes d'ordonnancement d'intervalles fixes en ligne où les intervalles d'une longueur donnée $f_j - s_j$ sont ordonnés dans l'ordre de leur date de début s_j (Lipton and Tomkins, 1994). L'ordonnanceur doit décider d'ordonnancer ou non chaque intervalle avant qu'un nouveau travail (intervalle) arrive. L'objectif est de maximiser la longueur totale des intervalles ordonnancés en s'assurant qu'aucune paire d'intervalles ordonnancés ne se chevauche. (Lipton and Tomkins, 1994) montrent qu'aucun algorithme randomisé ne peut atteindre un ratio compétitif meilleur que $O(\log \Delta)$, où le rapport entre l'intervalle le plus long et le plus court est Δ. Ils proposent alors un algorithme randomisé compétitif [1] de complexité $O((\log \Delta)^{1+\epsilon})$. Ils ont aussi proposé un algorithme 2-compétitif pour le cas de deux longueurs. Ces derniers résultats ont été généralisés lorsque les retards sont considérés Goldman et al. (2000). Un retard δ_j d'un intervalle j signifie que si un intervalle est ordonnancé, il doit commencer entre s_j et $s_j + \delta_j$.

Le cas préemptif a été abordé dans (Woeginger, 1994). Les auteurs ont considéré le cas où chaque intervalle a un poids donné (pas nécessairement égal à sa longueur), le poids de l'intervalle préempté n'est pas comptabilisé. L'objectif est de maximiser le poids total des intervalles ordonnancés sans préemption. Woeginger (1994) montre qu'aucun algorithme déterministe ne peut atteindre un ratio fini k-compétitif, même pour un algorithme randomisé (Canetti and Irani, 1998). Woeginger (1994) montre aussi que si le poids d'un intervalle est fonction de sa longueur, il existe alors des classes de fonctions pour lesquelles on peut trouver des bornes supérieures et inférieures avec ratio fini k-compétitif pour des algorithmes déterministes. (Seiden, 1998) proposent un algorithme randomisé pour une classe spécifique de fonctions qui améliore le ratio fini k-compétitif obtenu par un algorithme déterministe. Miyazawa and Erlebach (2004) étudient un cas particulier où les longueurs des intervalles sont égales et où l'ordre d'arrivée des intervalles correspond à l'ordre croissant de leurs poids. Les auteurs montrent que les algorithmes randomisés peuvent donner un meilleur ratio fini k-compétitif par rapport aux meilleurs algorithmes déterministes possibles.

Dans le cas où chaque intervalle a un poids unitaire et m machines sont disponibles, Faigle and Nawijn (1995) et Carlisle and Lloyd (1995) indépendamment ont noté qu'un algorithme glouton est en fait un algorithme en ligne qui produit toujours une solution optimale. (Faigle et al., 1999) ont adapté cet algorithme pour traiter le cas avec fenêtres de temps.

Enfin, des versions préemptives où un travail interrompu doit être repris immédiatement

1. On dit qu'un algorithme en ligne est k-compétitif si la solution calculée est au pire k fois la valeur optimale obtenue pour le cas le cas hors ligne.

sur une autre machine ont été étudiées principalement par Dondeti and Emmons (1993); Kroon et al. (1997); Fischetti et al. (1989).

2.2.2 Applications des problèmes d'ordonnancement des intervalles fixes

Outre le contexte du travail de cette thèse présenté dans le chapitre précédent, nous souhaitons insister dans cette section sur d'autres domaines d'application montrant ainsi l'intérêt de nos contributions dans le domaine de l'ordonnancement d'intervalles fixes.

En effet, les problèmes d'ordonnancement d'intervalles fixes ont une grande diversité d'applications, ces problèmes sont des versions abstraites de certains défis qui peuvent survenir dans différentes situations du monde réel, illustrés par les exemples suivants.

Le problème fondamental de l'affectation des équipages peut être formulé comme suit (voir par exemple (Beasley and Cao, 1998) ou (Mingozzi et al., 1999)) : on connaît un ensemble d'équipes, un ensemble de lieux et un ensemble de tâches. Pour chaque tâche, une date de début, une date de fin, un lieu et une matrice des distances entre chaque paire de lieux sont donnés. Le problème de l'affectation de tâches à des équipages en minimisant le nombre d'équipages peut être formulé dans la terminologie de l'ordonnancement d'intervalles fixes en considérant les tâches comme des intervalles, les équipages comme des machines, et en tenant compte d'une distance entre toute paire d'intervalles. Les problèmes d'ordonnancement des équipages sont parmi les problèmes les plus célèbres de la recherche opérationnelle. Ils apparaissent également dans les applications pour les conducteurs de bus (voir (Martello and Toth, 1986)).

Un autre exemple est celui des utilisateurs qui communiquent entre eux en utilisant un réseau. Un utilisateur communique en demandant la capacité d'un lien (appelé bande passante) dans le réseau pendant un intervalle donné. Bien entendu, les demandes de différents utilisateurs peuvent ne pas être compatibles en raison de la quantité de capacité demandée ou des intervalles demandés. La question principale de cette application est de savoir comment utiliser le réseau de manière optimale en allouant la bande passante disponible aux utilisateurs. Notez que cette application est particulièrement pertinente dans un contexte en ligne. En utilisant la terminologie de l'ordonnancement des intervalles fixes, nous pouvons considérer les demandes comme des intervalles, et les liens comme des machines (pour plus de détails, voir (Bar-Noy et al., 1999), (Kumar, 1998), (Bhatia et al., 2003)).

(Gabrel, 1995) considère le problème de prises de photos par satellite. Un satellite tourne sur orbite autour de la terre. Sa tâche consiste à photographier des morceaux de la surface de la terre. Pour répondre à une demande de photographie d'un morceau spécifique de la terre, l'appareil photo doit commencer et terminer le tournage à des moments donnés. En utilisant la terminologie de l'ordonnancement d'intervalles fixes, une demande devient une tâche et l'appareil photo est la machine.

(Anthonisse and Lenstra, 1984) décrivent un problème de location de chalet. Étant donné un ensemble de chalets identiques et une période, peut-on répondre immédiatement à la demande d'un client pour une réservation ? Et que se passe-t-il si certaines périodes ont déjà été pré-affectées ? De toute évidence, dans un contexte d'ordonnancement d'intervalles fixes, un chalet est une machine et une demande est un intervalle.

(Gupta et al., 1979) et (Hashimoto and Stevens, 1971) décrivent un problème d'attribution des canaux. Les données sont des paires de composants électroniques. Chaque composant doit être placé sur un circuit imprimé à des coordonnées (x, y) données. Ensuite, les deux composants sont inter-connectés en utilisant un canal. Les canaux ne peuvent pas se chevaucher. La minimisation du nombre de canaux revient à résoudre le problème de base d'ordonnancement d'intervalles fixes.

(Kolen and Kroon, 1991, 1994) abordent un problème de maintenance dans l'industrie aéronautique. Plus précisément, les ingénieurs doivent effectuer des tâches de maintenance sur les avions. Il existe différents types d'avions, et un ingénieur n'est autorisé à effectuer une tâche sur un type d'avion spécifique que s'il possède une licence pour ce type. Les tâches ont des heures fixes de début et de fin. Les ingénieurs titulaires d'une licence jouant le rôle de machines, le problème est un problème d'ordonnancement d'intervalles fixes où chaque machine peut traiter un ensemble donné de tâches.

(Carter and Tovey, 1992) décrivent un problème d'affectation de classe à des salles. Le problème est d'affecter n classes qui se réunissent pendant des périodes données aux salles $salle_j, j = 1 \dots m$, sous diverses contraintes et objectifs. L'une des variantes qu'ils considèrent est le réglage de l'ordre des salles de telle sorte que si une classe accepte la $salle_j$, alors elle accepte également toute $salle_k$ avec $k > j$. Donc, les classes ont le rôle des intervalles et les salles ont le rôle des machines, le problème revient à résoudre le problème d'ordonnancement d'intervalles fixes.

(Brehob et al., 2004) étudient les politiques de remplacement des mémoires caches non standard (une mémoire cache stocke des éléments dans des emplacements de cache). Ils établissent un parallèle avec l'ordonnancement d'intervalles fixes en faisant correspondre chaque accès à un certain élément à une heure de début d'un intervalle, et l'accès suivant à cet élément sert d'heure de fin de cet intervalle. Une machine correspond à un emplacement de cache.

Un autre problème vient de la biologie computationnelle (voir (Chen et al., 2005)). Étant donnée une séquence d'acides aminés (la seule machine) et ce que l'on appelle des segments (ceux-ci correspondent aux tâches), et un bénéfice pour chaque affectation possible d'un segment à une position dans la séquence d'acides aminés. Le problème consiste à trouver une affectation tel qu'aucun segment ne se chevauche et que le poids total soit maximisé.

2.3 Problème d'ordonnancement dans le cloud

Dans cette section, nous rappelons quelques travaux sur les clouds tenant compte principalement de la consommation énergétique, introduite dans la section 1.4.1.

2.3.1 Ordonnancement dans le cloud orienté-marché

— **Ordonnancement niveau-service :** Le concept de cloud computing orienté-marché est appelé *utility computing* (informatique utilitaire, voir section 1.2.4). (Chun and Culler, 2002; Irwin et al., 2004) sont parmi les premiers à avoir proposé un modèle d'utility computing dans le domaine de l'informatique. L'économie est le critère commun à toutes les approches qui résultent de ce modèle. Les travaux de (Lee et al.,

2010) ont proposé deux algorithmes basés sur un modèle de tarification. Les deux algorithmes utilisent le partage des processeurs afin de trouver un compromis entre des objectifs contradictoires : le profit et le temps de réponse. (Burge et al., 2007) décrivent une méthode pour des machines hétérogènes qui maximise le profit en affectant les demandes aux machines en fonction de leur coût énergétique. D'autres approches basées sur des algorithmes génétiques qui maximisent le profit sont présentées dans (Yu and Buyya, 2006) et (Garg et al., 2011a). Garg et al. (2011a) ont proposé une matheuristique, un algorithme génétique basé sur la programmation linéaire. L'objectif de cet algorithme est d'obtenir le meilleur méta-ordonnanceur qui minimise le coût combiné associé à tous les utilisateurs de manière coordonnée. Tous ces travaux font référence au modèle d'utility computing. Dans ce type de modèle, l'interaction se fait directement entre le client et le fournisseur. Cela forme un modèle à deux niveaux, c'est-à-dire le client et le prestataire.

Cependant, la commercialisation du cloud computing est de plus en plus complexe et soulève un certain nombre de questions. En effet, lorsque l'on parle de performance ou de qualité de service, il faut entendre par là une dépense (des coûts), principalement lorsqu'il s'agit de modèles de cloud computing où le fournisseur a un contrôle total sur les prix. Pour faire face à un tel problème, une évolution du modèle a été introduite : un troisième niveau apparaît et joue le rôle d'un intermédiaire pour trouver le compromis entre les besoins des clients et les bénéfices des fournisseurs. Une série de travaux ont été menés sur ce nouveau modèle. Chaisiri et al. (2009) proposent un algorithme optimal de placement des machines virtuelles. L'objectif est de minimiser les coûts des fournisseurs tout en affectant les machines virtuelles dans un environnement de cloud multiple. Pour cela, ils utilisent une stratégie qui évite les deux extrêmes (sur et sous-affectations). L'approche répond aux besoins en ressources appropriées en ajustant la tarification en fonction de la charge d'arrivée des machines virtuelles (VMs).

D'autres travaux comme (Tordsson et al., 2012) et (Lucas-Simarro et al., 2013) illustrent le courtage de cloud appliqué à l'ordonnancement au niveau-service. Dans (Tordsson et al., 2012), les auteurs présentent une étude dans laquelle ils comparent les mécanismes de placement des VMs sur un cloud multi-fournisseurs et multi-sites. Ils prouvent qu'un déploiement multi-cloud utilisant un courtier a un effet bénéfique sur les performances et réduit les coûts des services déployés. Il en va de même pour (Lucas-Simarro et al., 2013) qui se focalisent sur différentes stratégies d'ordonnancement pour un déploiement optimal des services virtuels sur multi-cloud en traitant de manière agrégée, par exemple, le budget, la performance, les types d'instance, le placement, etc.

Dans (Elmroth et al., 2009), les auteurs décrivent des scénarios d'utilisation et un ensemble d'exigences pour une fédération des infrastructures en cloud basée sur la notion de RESERVOIR (*Resources and Services Virtualization without Barriers*[2]). Ils proposent également une architecture de comptabilité et de facturation entre les consommateurs de ressources et les fournisseurs d'infrastructures à utiliser dans le cadre de RESERVOIR. L'objectif est de faire face à la migration des machines

2. Permet le déploiement à grande échelle et de la gestion des services IT complexes dans différents domaines administratifs, plateformes et pays

virtuelles en gérant les systèmes de paiement post-payés et prépayés en fonction des besoins des utilisateurs.
Tous les auteurs cités jusqu'ici prennent en compte le profit et/ou d'autres objectifs dans leurs études, mais ils ne considèrent pas la relation entre les critères traités. Ils ne prêtent pas non plus attention à la consommation d'énergie de leur architecture. Le travail présenté par (Garg et al., 2011b) traite ces points, en proposant un nouveau modèle énergétique pour l'ordonnancement au niveau-service qui inclut les émissions de gaz et la tarification. Plusieurs heuristiques sont proposées pour trouver un bon compromis entre les objectifs. Cependant, cette approche est une hiérarchisation des objectifs, c'est-à-dire qu'elle ne peut optimiser qu'un seul objectif à la fois.
Nous remarquons à travers tous les travaux présentés sur l'ordonnancement au niveau-service deux problèmes majeurs. Le premier est le manque de prise en compte de l'énergie dans de nombreux travaux, malgré l'importance cruciale que peut avoir l'énergie sur une architecture distribuée. Le second point concerne l'approche multi-objectifs utilisée dans ces travaux. Il s'agit toujours d'une approche pseudo-multi-objectifs, c'est-à-dire que les objectifs sont agrégés ou classés par ordre décroissant du plus important au moins important. Toutefois, cet ordre est subjectif et change en fonction des besoins. En outre, le fait de changer l'ordre des objectifs peut modifier totalement les résultats et compliquer l'analyse des résultats.

— **Ordonnancement niveau-tâche** Comme indiqué précédemment, la deuxième partie de cloud computing orientée-marché, outre l'ordonnancement au niveau-service, est l'ordonnancement au niveau-tâche. Contrairement à l'ordonnancement au niveau-service, l'ordonnancement au niveau-tâche a pour but de traiter l'affectation des tâches au sein de chaque cloud. Le travail présenté par (Chen et al., 2011) propose une théorie de l'utilité basée sur l'économie pour développer un nouveau modèle qui intègre la satisfaction du client dans un cloud. Ils ont proposé deux algorithmes d'ordonnancement pour trouver un bon compromis entre la satisfaction du client et le profit du prestataire pendant l'affectation de la tâche. Toutefois, cette approche est une agrégation d'objectifs (c'est-à-dire qu'elle combine plusieurs objectifs pour en créer un nouveau). (Wu et al., 2013) proposent un ensemble de mesures pour l'ordonnancement hiérarchique dans un cloud distribué orienté-marché. L'approche traite hiérarchiquement, à la fois de l'ordonnancement au niveau-service et de l'ordonnancement au niveau-tâche, où les instances individuelles de flux de travail sont mises en correspondance avec les services cloud dans un cloud multi-fournisseurs. Les objectifs de cette approche consistent à minimiser le makespan, le coût et le temps de CPU. Pour cela un algorithme randomisé pour l'ordonnancement au niveau-service et trois méta-heuristiques (algorithme génétique, colonies de fourmis et essaims particulaires) pour l'ordonnancement au niveau-tâche ont été proposés. Tous les travaux étudiés ci-dessus présentent différentes approches de courtage (cf. section 2.3.1) pour gérer l'ordonnancement des tâches. Toutefois, aucune de ces études ne tient compte de la relation entre le prix facturé et la performance fournie. En ce qui concerne l'ordonnancement des services, les auteurs ne prêtent pas attention à la manière dont chacun de ces critères peut affecter les autres, et aucune solution proposée ne

les aborde simultanément.

2.3.2 Ordonnancement dans le cloud non orienté-marché

Après une course à la performance et au profit, les paradigmes de l'utility et du cloud computing sont confrontés à un problème énergétique. Pour réduire la consommation d'énergie, diverses questions telles que la gestion des ressources dans les couches logicielles et matérielles doivent être abordées. Les approches logicielles sont principalement basées sur la virtualisation ou la consolidation des tâches. La technique de consolidation vise à maximiser l'utilisation des ressources en minimisant la consommation d'énergie. En effet, une stratégie d'allocation des ressources qui prend en compte l'utilisation des ressources devrait conduire à une meilleure efficacité énergétique. Les approches matérielles utilisent l'opportunité offerte par les fabricants de processeurs modernes pour ajuster la tension et la fréquence. Cet ajustement peut faire varier les performances du processeur et donc sa consommation d'énergie. C'est pourquoi plusieurs travaux ont été proposés dans le domaine de l'informatique en tenant compte de l'énergie. Ces approches sont basées sur la couche logicielle ou matérielle du système (c'est-à-dire au niveau du système).

Avant la généralisation de la virtualisation, la plupart des approches liées à l'énergie au niveau du système n'étaient pas génériques, s'attaquant à ce problème uniquement en se référant et en se concentrant sur l'ordonnancement d'applications (tâches) dédiées sur une architecture spécifique. Dans (Lee and Zomaya, 2009; Rizvandi et al., 2010), par exemple, une technique matérielle est proposée. Elle consiste à faire varier dynamiquement la fréquence du CPU afin de minimiser la consommation d'énergie. Dans (Guzek et al., 2012), les auteurs étudient l'influence d'une technique matérielle en utilisant les algorithmes évolutifs (NSGAII) pour l'ordonnancement bi-objectif (consommation d'énergie et makespan) pour un graphe orienté acycliques de tâches sur une plate-forme multiprocesseur hétérogène. L'inconvénient de ce type de méthodes est l'hypothèse de l'existence d'un lien direct entre les tâches et les ressources. Pour résoudre ce problème, d'autres méthodes utilisant la consolidation apparaissent au niveau du système. Dans (Lee and Zomaya, 2012), les auteurs présentent deux heuristiques (ECTC et MaxUtil) de consolidation de tâches considérant l'énergie. Ces deux heuristiques visent à maximiser l'utilisation des ressources. Ils prennent explicitement en compte à la fois le calcul et la consommation d'énergie au repos. Alors, pour l'ordonnancement d'une tâche, les heuristiques proposées affectent chaque tâche à la ressource sur laquelle la consommation d'énergie est explicitement ou implicitement minimisée sans dégradation de la performance de cette tâche. Cependant, la consolidation a également des problèmes. Les travaux de (Srikantaiah et al.) exposent certains de ces problèmes, principalement en ce qui concerne les objectifs d'optimisation de la puissance et proposent quelques orientations de recherche pour relever les défis impliqués. De plus, une autre façon de réduire l'empreinte énergétique du cloud computing est proposée dans (Tesauro et al., 2007). Les auteurs présentent une approche de l'apprentissage par renforcement pour traiter l'optimisation de deux critères principaux, la performance et la consommation d'énergie.

Cependant, en raison de l'évolution des architectures due à la généralisation du concept de virtualisation, l'ordonnancement au niveau-service se restreint exclusivement à un niveau-VM (niveau-machine virtuelle). C'est pourquoi certaines des méthodes précédentes, comme

la consolidation, ont évolué de l'ordonnancement des tâches à l'ordonnancement des VMs.

Dans (Mezmaz et al., 2011), il est proposé une étude préliminaire pour l'étape de transition de l'ordonnancement au niveau du système à l'ordonnancement au niveau-VM. Dans ce travail, le problème de l'ordonnancement des applications parallèles avec des contraintes de précédences sur des systèmes informatiques hétérogènes (HCS) comme les infrastructures de cloud computing est étudié. Ce type d'application a ensuite été étudié et utilisé dans de nombreux travaux de recherche.

— **Ordonnancement niveau-VM** (Von Laszewski et al., 2009) ont proposé l'une des évolutions les plus importantes entre l'ordonnancement niveau-VM et le niveau-système. Dans ces travaux, l'algorithme d'ordonnancement des VM est basé sur la technique DVFS (*Dynamic Voltage and Frequency Scaling*, dite aussi technique matérielle) pour réduire la consommation d'énergie d'un seul cluster virtualisé OpenNebula (exemple d'un gestionnaire cloud). L'idée de ce travail est de réduire la fréquence d'horloge du cluster aussi bas que possible pour répondre exactement aux demandes des machines virtuelles. Le principal défaut de l'approche proposée dans (Von Laszewski et al., 2009) est qu'elle ne traite qu'un seul cluster et non un grand nombre de machines qui composent en général un cloud. En outre, cette étude fait une hypothèse sur la configuration matérielle des machines qui composent le cloud, en supposant qu'elles sont équipées de DVFS.
(Andreolini et al., 2009) ont proposé un algorithme visant à réaffecter des machines virtuelles dans un cloud à grande échelle. L'approche consiste à identifier uniquement les instances critiques en utilisant l'évolution de la charge au lieu des seuils pour prendre des décisions afin d'équilibrer la charge des clouds. La contribution de ce travail est une amélioration de la sélectivité et de la robustesse du processus de migration. Cependant, aucun intérêt n'est accordé au critère énergétique.
Dans (Guérout et al., 2017), les auteurs s'intéressent à l'analyse du niveau de qualité de service (QoS) dans un environnement de Cloud Computing puisqu'un fournisseur de Cloud doit prendre en compte de nombreux objectifs de QoS, ainsi que la manière de les optimiser pendant le processus d'allocation des machines virtuelles pour trouver un bon compromis. Les auteurs proposent donc une optimisation multiobjectif de quatre objectifs pertinents de qualité de service du nuage, en utilisant deux méthodes d'optimisation différentes : un algorithme génétique (AG) et une approche de programmation linéaire mixte en nombres entiers (MILP). La complexité du problème d'allocation des machines virtuelles est accrue par la modélisation de la mise à l'échelle dynamique de la tension et de la fréquence (DVFS) pour économiser l'énergie sur les hôtes. Une formulation globale de programmation non linéaire en nombres mixtes est présentée et une formulation MILP est dérivée par linéarisation. Une méthode de décomposition heuristique, qui utilise le MILP pour optimiser les objectifs intermédiaires, est proposée. De nombreux résultats expérimentaux montrent la complémentarité des deux heuristiques pour obtenir divers compromis entre les différents objectifs de qualité de service.
D'autres travaux dans la littérature modélisent le problème d'ordonnancement des VM comme un problème de bin-packing où les VM sont les objets à empaqueter et les machines sont les bins. On peut citer les travaux de Verma et al. (2008) qui proposent une architecture appelée pMapper qui permet d'affecter les applications en tenant

compte à la fois du coût de la puissance et de la performance : un algorithme basé sur First Fit Decreasing (FFD). (Borgetto et al., 2012b) présentent une approche utilisant la migration et la reconfiguration des machines virtuelles, et la gestion de la puissance physique des machines à des fins de réduction de la consommation d'énergie tout en offrant un SLA (Service Level Agreement) correct. L'approche est basée sur une boucle de gestion autonome utilisant différentes heuristiques, dont une basée sur FFD.
Une étude comparative de (Mills et al., 2011) a démontré que le FFD et ses variantes sont très performants comparé aux autres heuristiques pour l'affectation des VMs. Cependant, le problème majeur de ce type d'algorithmes est le manque de diversité. En effet, ils ne trouvent l'optimum global (meilleure solution) que pour des fonctions ayant un seul optimum local. Néanmoins, le problème d'ordonnancement change avec l'arrivée des différentes VM où des différents optima locaux sont possibles. Il est alors nécessaire d'explorer d'autres optima locaux. C'est le principe justifiant les méta-heuristiques.
Une autre étude portant à la fois sur la consommation d'énergie et les performances professionnelles, c'est-à-dire la meilleure allocation des jobs services sur les infrastructures. Borgetto et al. (2012a) proposent trois formulations différentes du problème et en utilisant pour chaque formalisation des heuristiques pour trouver le meilleur compromis entre les objectifs. Le principal problème d'une telle formalisation est que le problème multi-objectif est abordé soit par l'approche lexicographique, soit par agrégation des critères.
Une autre méthode consiste à proposer un gestionnaire de cloud natif en considérant l'aspect énergétique. Snooze par exemple est le premier gestionnaire de clouds qui inclut un critère énergétique (Feller et al., 2012). Les machines virtuelles qui composent le cloud ont la possibilité de s'éteindre lorsqu'elles sont inactives. Cette technique est l'évolution des travaux proposé dans (Orgerie et al., 2008). Les auteurs de ces travaux étudient la réduction de la consommation d'énergie dans les grilles de calcul à grande échelle comme Grid'5000 en désactivant les noeuds inactifs de manière intelligente. On peut voir l'impact de la virtualisation sur le concept de remplacement des noeuds par les VM. Le principal inconvénient de ces techniques est le surcoût de consommation d'énergie causé par une mauvaise anticipation lors de l'arrêt des machines/VM.
De plus, il est rare de trouver des approches multi-objectifs qui abordent l'ordonnancement des VMs comme les travaux de (Xu and Fortes, 2010). En effet, ces travaux proposent une politique de placement des VM utilisant un algorithme génétique reposant sur une évaluation floue multi-objectifs pour minimiser simultanément le gaspillage total de ressources, la consommation d'énergie et le coût de dissipation thermique. Le principal inconvénient est que, contrairement à une véritable approche de Pareto qui offre une diversité dans les solutions résultantes en proposant un large choix de solutions non-dominées, l'utilisation de la technique floue multi-objectifs proposée fournit une solution finale unique.

2.4 Autres problèmes d'optimisation proches

2.4.1 Sac-à-dos multidimensionnel

Le problème du sac-à-dos multidimensionnel "MKP" est un problème d'optimisation combinatoire intéressant et difficile à résoudre (Croce and Grosso, 2012; Varnamkhasti, 2012). Intéressant car il modélise un certain nombre d'applications issues du mode réel tels que le problème de chargement (Hifi, 2009), le stock de découpe (Gilmore and Gomory, 1966), l'affectation des tâches et l'ordonnancement multiprocesseur (Labbé et al., 2003), la sélection de projets (Petersen, 1967), etc. Il ne s'agit pas dans cette section de réaliser un état-de-l'art sur le MKP mais nous allons rappeler la définition du problème et certains résultats de la littérature que nous avons étudiés du cas MKP déterministe.

En effet, dans le MKP, un ensemble d'items est donné, chacun ayant une taille et une valeur, qui doivent être placés dans un sac-à-dos qui a un certain nombre de dimensions ayant chacune une capacité limitée. L'objectif est de trouver un sous-ensemble d'items conduisant au profit total maximum tout en respectant les contraintes de capacité. Il s'agit d'une généralisation du problème de sac-à-dos classique qui a connu et connaît toujours une étude approfondie (Croce et al., 2019; Croce and Scatamacchia, 2020). Un aperçu complet des résultats pratiques et théoriques du MKP peut être trouvé dans la monographie sur les problèmes de sac-à-dos dans (Kellerer et al., 2004). Une revue de la littérature sur le MKP a été donnée par (Fréville, 2004). Une littérature élaborée sur le MKP et ses relations avec différents problèmes est publiée également dans (Lai et al., 2014). En outre, une étude des algorithmes les plus populaires qui ont été utilisés pour résoudre le MKP, y compris les méthodes exactes et heuristiques, peut être trouvée dans (Varnamkhasti, 2012). Nous mentionnons également (Puchinger et al., 2010) pour une étude récente des structures et des algorithmes du MKP. D'autre part, dans (Lust and Teghem, 2012) les auteurs s'intéressent aux MKP multiobjectifs. Ils classifient et discutent brièvement les approches de résolution existantes sur ce sujet, en particulier les métaheuristiques. De la même manière, dans (Lienland and Zeng, 2015), les auteurs ont focalisé leurs travaux uniquement sur les algorithmes génétiques. Ils passent en revue et comparent onze variantes de cette approche pour résoudre le MKP. Ainsi, des approches heuristiques et métaheuristiques ont été proposées afin d'obtenir une solution approximative en un temps raisonnable, sans garantir l'optimalité. D'autres méthodes hybrides ont été développées en combinant une heuristique/métaheuristique avec une autre heuristique/métaheuristique (Alonso et al., 2006), ou en combinant une méthode exacte avec une autre méthode exacte (Dyer et al., 1995), ou encore en combinant une heuristique/métaheuristique avec une méthode exacte (Gallardo et al., 2005; Jourdan et al., 2009).

2.4.2 Problèmes de gestion de projet à contraintes de ressources

Les problèmes d'ordonnancement de projets sous contrainte de ressources (RCPSP) impliquent l'affectation de travaux ou de tâches à une ressource ou à un ensemble de ressources dont la capacité est limitée afin d'atteindre un objectif prédéfini. Le plus courant des objectifs considérés est la minimisation du makespan, c'est-à-dire le temps minimum pour achever l'ensemble du projet.

Le RCPSP est devenu un problème standard bien connu dans le contexte de l'ordonnancement de projet, ce qui a motivé de nombreux chercheurs pour le développement des algorithmes d'ordonnancement exacts (Koné et al., 2011; Artigues, 2017) et des heuristiques (Kolisch and Hartmann, 1999; Kolisch and Hartmann).

Cependant, le RCPSP est un modèle plutôt basique dont les hypothèses sont trop restrictives pour de nombreuses applications pratiques. Par conséquent, diverses extensions du RCPSP ont été développées. Dans (Hartmann and Briskorn, 2010), les auteurs donnent un aperçu de ces extensions, classées en fonction de la structure du RCPSP selon : des contraintes temporelles, des relations de précédence et des contraintes de ressources. Des objectifs et approches alternatifs pour l'ordonnancement de projets multiples sont également présentés. Des variantes et extensions ont été introduites et ont fait l'objet de différentes études telles que les décalages temporels minimaux et maximaux (Artigues and Briand, 2009).

Selon les données du problème, il existe deux modes de problèmes RCPSP : les problèmes d'ordonnancement monomode et les problèmes multimodes. Les problèmes à mode unique impliquent que chaque activité a un mode d'exécution unique : la durée de l'activité et ses exigences pour un ensemble de ressources sont supposées fixes. Dans les problèmes multimodes, chaque activité doit être traitée dans l'un des modes possibles. Chaque mode implique une option différente en termes de coût (éventuel) et de durée d'activité pour réaliser l'activité considérée.

Comme l'objectif de cette section n'est pas de développer un état-de-l'art exhaustif des études menées sur le RCPSP et ses extensions mais plutôt de situer notre problème par rapport aux différents modèles existants, nous allons rappeler succinctement le RCPSP de base monomode et les problèmes de RCPSP liés au bin-packing. Pour aller plus loin sur l'ordonnancement de projets sous contraintes de ressources, nous recommandons au lecteur intéressé les travaux de Kolisch and Hartmann (1999); Yang et al. (2001); Kolisch and Hartmann; Hartmann and Briskorn (2010); Artigues et al. (2013).

1. RCPSP de base monomode : Comme nous l'avons noté, un seul mode d'exécution est disponible pour chaque activité. Les RCPSPs à mode unique contiennent généralement quatre sous-classes. En général, un projet est représenté comme un réseau d'activités sous la forme d'un graphe $G(N, A)$, où les noeuds du graphe correspondent aux activités, et les arcs du graphe spécifient les relations de précédence : si l'arc (i, j) apparaît dans le graphe, alors l'activité i doit être achevée avant d'effectuer l'activité j. Selon les contraintes de précédence, nous avons deux types de conditions : les cas dans lesquels l'activité j peut commencer à tout moment après l'achèvement de l'activité i, ou les cas dans lesquels l'activité j doit commencer dans une certaine fenêtre de temps après l'achèvement de l'activité i (pour le cas General Precedence Relationships). La disponibilité d'une ressource donnée peut être la même pour toutes les périodes, ou elle peut varier d'une période à l'autre. De plus, si une tâche est assignée à une ressource dans plusieurs périodes, la tâche peut nécessiter la même quantité de temps de traitement dans toutes les périodes, ou la consommation de la ressource peut varier dans les différentes périodes. Même si le critère le plus considéré est le makespan, un large éventail d'autres objectifs est possible.

2. Problèmes RCPSP liés au bin-packing : Selon (Yang et al., 2001), un RCPSP lié au bin-packing fait référence à un type particulier de problème d'ordonnancement de projet sous contrainte de ressources qui peut être considéré comme analogue à un problème de bin-packing. Une instance de problème de mise en bins est spécifiée par une taille du bin standard et un ensemble d'articles, chacun d'une taille (éventuellement) unique. L'objectif est d'emballer chaque article de l'ensemble dans un bin tout en minimisant le nombre total de bins utilisés. De nombreux problèmes d'ordonnancement sous contrainte de ressources peuvent être considérés comme des généralisations (ou même des cas particuliers) du problème de base de l'emballage des bins. L'analogie entre les problèmes de bin-packing et de RCPSP peut être expliquée comme suit. La capacité des ressources représente la taille du bin, tandis que les exigences de consommation des ressources d'une tâche représentent la taille d'un élément. Dans le contexte du RCPSP, nous pouvons considérer chaque période de temps comme un bin dans lequel nous pouvons placer différentes tâches (bien sûr, dans beaucoup de ces problèmes, nous pouvons placer un élément dans des bins consécutifs, ou de manière équivalente, programmer des tâches sur des jours consécutifs). Si nous supposons, cependant, que chaque tâche prend moins d'une période de consommation de ressources et que chaque tâche doit être entièrement achevée en un seul jour, alors minimiser le makespan de ce RCPSP est équivalent à minimiser le nombre de bins utilisés dans un problème équivalent de bin-packing. Nous pouvons adapter au RCPSP des heuristiques extrêmement efficaces utilisées pour résoudre des problèmes de mise en bacs avec des modifications relativement mineures (Yang et al., 2001). Une des variantes proche de notre cas d'étude est le BP-RCPSP (BP multidimensionnel) sans relations de précédence. Dans ce problème, chaque tâche requiert une unité de temps pour un sous-ensemble de ressources $k = 1, 2, \ldots, K$. Les besoins en ressources des travaux $j = 1, 2, \ldots, n$ sont définis par les vecteurs $V_1, V_2, \ldots, V_n \in [0, 1]^K$. v_{kj} est le ième élément de V_j tel que : $v_{kj} = 1$ si le travail j nécessite la ressource k ; 0 sinon. La ressource k a une capacité r_k par période. Traiter le travail j dans une période de temps signifie mettre le vecteur V_j dans un bin.L'objectif est alors de placer tous les vecteurs dans un nombre minimal de bins (nombre de périodes de temps) de telle sorte que dans chaque bin l et pour chaque coordonnée k, le vecteur v_j soit placé dans un bin l et pour chaque coordonnée $k, k = 1, \ldots, K$, les restrictions de capacité pour chaque ressource soient satisfaites. Pour $k = 1$, le problème n'est autre qu'un problème classique de bin packing unidimensionnel, qui a été largement étudié. Garey et al. (1976) ont adapté les heuristiques First Fit (FF) et First Fit Decreasing (FFD) au cas du bin packing multidimensionnel. Notons que pour résoudre ce problème, il existe un algorithme d'approximation en temps polynomial dont le rapport asymptotique au pire cas est égal à 1 Coffman et al. (1984).

2.4.3 Ordonnancement dans les systèmes distribués hétérogènes

L'évolution de la conception des plateformes parallèles modernes conduit à revisiter l'ordonnancement des tâches sur des ressources distribuées. Récemment, Beaumont et al. (2020) publient un état-de-l'art assez complet présentant les principaux algorithmes pro-

posés pour l'ordonnancement des tâches soumises par les utilisateurs sur des ressources hétérogènes distribuées. Ils font état du manque d'études traitant cette problématique, contrairement au contexte d'une plateforme homogène composée de ressources identiques. Dans ce dernier cas, des algorithmes efficaces et des résultats d'approximation théoriques ont été développés (Drozdowski, 2009; Bleuse et al., 2015, 2017) avec des implémentations dans des ordonnanceurs batch réels, tels que SLURM ou Torque. Cependant, ces dernières années, une vaste littérature s'intéresse à l'évolution des architectures, évolution qui est en général mal comprise (Canon et al., 2017, 2020).

Dans (Beaumont et al., 2020), les auteurs mettent en particulier l'accent sur les difficultés intrinsèques introduites par l'hétérogénéité. En effet, il suffit d'analyser et de comparer l'efficacité des heuristiques de liste lorsqu'elles sont utilisées pour les plateformes homogènes et hétérogènes, en suivant l'analyse de Graham réalisée en 1969.

Il est à noter cependant qu'il n'est pas facile d'adapter les algorithmes de *List Scheduling* dans le contexte de ressources hétérogènes pour obtenir des algorithmes à faible coût dont les performances peuvent être évaluées théoriquement. Cependant, il est possible de concevoir des variantes de ce type d'algorithmes, en veillant sur une bonne allocation de l'ensemble de ressources.

Deux configurations sont considérées dans la littérature : hors ligne où l'ensemble des tâches est connu à l'avance (Kedad-Sidhoum et al., 2014, 2018) et en ligne où les tâches arrivent une par une, et il n'y a pas de connaissance à priori des tâches à venir (Chen et al., 2014; Imreh, 2003). Le cadre en ligne est plus difficile, mais il est d'un intérêt particulier dans le contexte des systèmes hétérogènes à grande échelle. En effet, dans le contexte des plateformes hétérogènes, les programmeurs d'applications parallèles s'appuient sur des systèmes dynamiques d'exécution. Ces ordonnanceurs, tels que Quark, Par-SeC, StarSs et StarPU, prennent toutes leurs décisions d'allocation et d'ordonnancement au moment de l'exécution. Ces décisions en ligne sont basées sur l'état de la plateforme, l'ensemble des tâches disponibles (prêtes) et éventuellement sur des stratégies de pré-affectation statiques et des priorités de travaux qui ont été calculées hors ligne.

Mittal and Vetter (2015) proposent une étude générale sur l'informatique hétérogène CPU-GPU qui couvre les langages de programmation, les cadres de travail et les outils de développement, mais aussi l'équilibrage des charges et plus généralement, les problèmes d'ordonnancement. La complexité des problèmes d'ordonnancement n'est pas abordée dans leur travail, mais une classification très étendue des solutions qui ont été proposées pour ordonnancer les applications sur des ressources hétérogènes CPU-GPU est proposée. Cette analyse de complexité des problèmes d'ordonnancement sur des ressources informatiques hétérogènes a été considérée dans (Beaumont et al., 2020). Lorsque le critère à optimiser est le makespan, des bornes inférieures sont proposées dans la littérature pour le cas hors ligne (citons par exemple les travaux de (Kedad-Sidhoum et al., 2015)) et pour le cas en ligne (citons par exemple les travaux de Canon et al. (2020)). En terme d'algorithmes de résolution à garantie de performance pour le cas hors ligne, un algorithme de liste basé sur une technique d'approximation duale de complexité $O(n \log n)$ est proposé par Kedad-Sidhoum et al. (2014). Deux familles d'algorithmes à garantie de performance qui combinent deux techniques, à savoir l'approximation duale et la programmation dynamique sont développées dans Kedad-Sidhoum et al. (2018).

Pour le problème d'ordonnancement en ligne et lorsque les travaux sont indépendants, un travail est programmé immédiatement et irrévocablement dès son arrivée. En 2003, Imreh (2003) a proposé deux heuristiques simples et un algorithme compétitif $(4-2/m)$ pour ce problème, lorsque deux types de ressources sont considérés : m (resp. k) ressources de type 1 (resp. type 2). Il s'agit de ressources de type CPU et GPU. La première heuristique, appelée Post Greedy, ordonnance chaque travail sur la machine où elle sera terminée le plus tôt. La complexité temporelle de cette opération est de $O(\log m)$. La deuxième heuristique, appelée Load Greedy, affecte chaque travail au type de ressource sur lequel le rapport de son temps de traitement divisé par le nombre de processeurs sur ce type de ressource est le plus petit. Avec cette méthode, les travaux sont assignés à un GPU uniquement s'ils sont suffisamment accélérés, de sorte que le débit des GPU est plus élevé pour de nombreux travaux identiques. Imreh prouve que le rapport concurrentiel est en fait égal à $(2+(m-1)/k)$. La complexité temporelle de cet algorithme est de $O(\log m)$.

2.5 Problème d'ordonnancement multiagent

L'ordonnancement multiagent fait partie du domaine de l'optimisation combinatoire multicritère. Dans la classe des problèmes d'ordonnancement multicritère classique (T'kindt and Billaut, 2006), un ensemble de critères imposé par un décideur est appliqué à la totalité des travaux. Ces modèles ne sont pas toujours représentatifs des problèmes réels. En effet, certains critères peuvent parfois être imposés par un autre décideur et concerner uniquement un sous-ensemble de travaux : commandes d'un client, articles périssables, etc. Il s'agit alors de problèmes d'ordonnancement multiagent. L'idée de base de ces problèmes est la considération de plusieurs preneurs de décision qui doivent se partager des ressources pour la réalisation de leurs travaux respectifs. En effet, la spécificité des problèmes d'ordonnancement multiagent est l'existence de plusieurs agents, chacun intéressé uniquement par un sous-ensemble de travaux. Chaque agent a sa propre mesure de performance qui dépend de l'ordonnancement de ses propres travaux. Cependant, tous les travaux doivent partager des ressources communes. Le problème consiste à trouver une stratégie efficace de partage des ressources pour exécuter les travaux des agents.

L'état-de-l'art sur les problèmes d'ordonnancement multiagents sont disponibles dans les travaux de (Agnetis et al., 2014) et (Perez-Gonzalez and Framinan, 2014).

Nous évoquons ci-après un certain nombre de notions s'inspirant de l'optimisation multicritère qui sont nécessaires dans la suite de ce manuscrit.

2.5.1 Notations

La notation la plus couramment utilisée pour décrire un problème d'ordonnancement a été introduite par (Graham et al., 1979). Leur proposition se résume en trois champs $\alpha|\beta|\gamma$, d'où l'appellation de *« notation à trois champs »*. Le champ α spécifie l'environnement de travail, β décrit l'ensemble des contraintes et γ indique le critère à optimiser. Selon le problème étudié, le champ α prend les formes suivantes : $1|\beta|\gamma$ machine unique, $Pm|\beta|\gamma$ machines parallèles identiques, $Qm|\beta|\gamma$ machines uniformes et $Rm|\beta|\gamma$ machines non reliées.

Dans l'état-de-l'art proposé par (T'kindt and Billaut, 2006), on trouve une extension de la notation des travaux d'ordonnancement proposée par (Graham et al., 1979) au cas multicritère. Les auteurs présentent des résumés détaillés d'une large palette de problèmes. Selon le problème étudié, le champ γ prend les formes suivantes :

- $\alpha|\beta|z$: si le problème étudié consiste à maximiser une seule fonction objectif (problème mono-objectif).
- $\alpha|\beta|z^1, \ldots, z^K$: dans le cas où plusieurs fonctions objectifs sont à maximiser et K est le nombre de fonctions.
- $\alpha|\beta|\varepsilon(z^1/z^2, \ldots, z^K)$: quand l'approche ε-contrainte est appliquée ; on cherche à maximiser le critère z^1, sous contrainte d'une borne inférieure sur chacune des autres fonctions objectif. On y associe alors un vecteur $Q = (Q_2, \ldots, Q_K)$, tel que chaque composante Q_k est la borne minimale du critère z^k, $k = 2, \ldots, K$.
- $\alpha|\beta|F_\ell(z^1, z^2, \ldots, z^K)$: exprime la combinaison linéaire de tous les critères. La maximisation d'un tel critère prend sens quand les critères sont commensurables, dans le cas contraire, d'autres approches multicritères sont plus favorables.
- $\alpha|\beta|Lex(z^1, z^2, \ldots, z^K)$: notation de la méthode lexicographique : on maximise z^1 puis à $z^1 = z^{1*}$ on maximise z^2 , ...
- $\alpha|\beta|P(z^1, z^2, \ldots, z^K)$: cette notation correspond au problème d'énumération totale du front de Pareto.
- $\alpha|\beta|\#(z^1, z^2, \ldots, z^K)$: étudie du nombre de solutions de Pareto admissibles pour le problème $\alpha|\beta|(z^1, z^2, \ldots, z^K)$.

2.5.2 Classe des problèmes d'ordonnancement multiagents

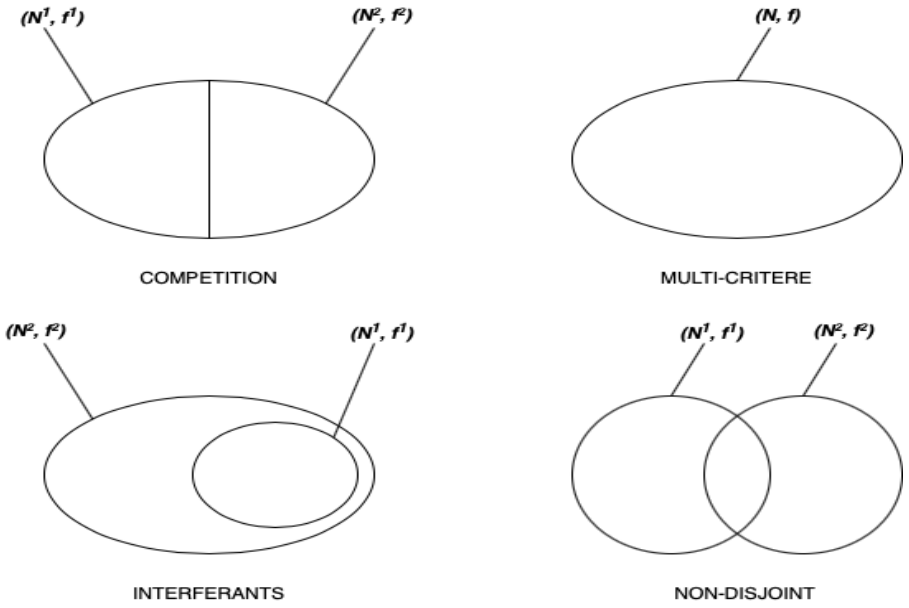
Selon Agnetis et al. (2014), un "agent" est une entité associée à un sous-ensemble de travaux inclus dans l'ensemble $\mathcal{N}$ des n travaux. Cette entité peut être associée à un autre décideur qui intervient dans le choix de la solution finale. Chaque agent vise à maximiser un critère qui lui est propre, car il ne dépend que de ses propres travaux. Ces agents sont en concurrence, puisqu'ils se partagent les mêmes ressources. Toutefois, ils peuvent aussi partager des travaux, et selon cette relation entre les agents, on définit différents scénarios correspondant à des classes de problèmes.

Il est important de noter que la notion d'agent est déjà considérée dans d'autres communautés à l'instar de l'Intelligence Artificielle, au demeurant, on ne trouve pas de définition commune dans cette communauté. En effet, (Moyaux et al., 2004) consacre une partie de son état de l'art à cette question et nous renvoyons vers sa thèse pour plus de détails. Nous retenons simplement, que dans ce domaine, un agent est un objet doté de certaines capacités telles que : l'autonomie et la réactivité ou bien même la capacité à poursuivre un but et à négocier de manière intentionnelle, le cas échéant, avec d'autres agents afin d'atteindre son but. Il semblerait que la notion d'agent que nous considérons dans nos travaux soit plus proche de celle utilisée en théorie des jeux (Moyaux et al., 2004). Les acteurs d'un jeu sont aussi appelés agents, ce peut être un objet, un humain, un programme, etc. On leur associe une fonction de gain, et à partir d'un ensemble de règles ils choisissent des stratégies qui peuvent les mener au meilleur des gains.

Si K agents sont considérés, chaque agent est associé à sous-ensemble $\mathcal{N}^k$, et vise à maximiser une fonction z_k. La figure 2.1 représente les quatre classes définies par (Agnetis

et al., 2014). On associe à chaque agent la notation $(\mathcal{N}^k, z^k)$, telle que $\mathcal{N}^k$ représente le sous-ensemble de travaux et z_k le critère à maximiser par l'agent k.

FIGURE 2.1 – Scénarios possibles pour les problèmes d'ordonnancement multiagent pour $K = 2$ agents.



1. **Multicritère classique** : C'est le cas classique de l'ordonnancement multicritères, il est désigné par (Soukhal, 2012) comme la classe de problèmes « symétriques » car les sous-ensembles de travaux N^K sont tous égaux à l'ensemble $N : N = N^1 = \ldots = N^K$. Toutefois, les critères ne sont par forcément identiques. Si un seul agent est considéré, dans ce cas le problème est mono-critère.

2. **Compétition (CO)** : Les agents n'ont aucun travail en commun, les sous-ensembles de travaux qui leur sont associés sont donc disjoints deux à deux : $N^h \cap N^l = \emptyset, \forall h, l \; tel \; que \; h \neq l$. On dit encore que tous les travaux du même sous-ensemble N^k appartiennent exclusivement au seul agent k. Initialement désignée par une classe de problèmes « multiagents », elle a été introduite par (Agnetis et al., 2000, 2014). C'est le scénario qui nous intéresse dans la suite de nos travaux.

3. **Interférants (IN)** : Ce scénario concerne les problèmes admettant $K - 1$ agents et un agent K associé à la totalité des travaux. Ce dernier sera parfois appelé « agent global », et « agents locaux » pour les autres. Sans perte de généralité, nous considérons la relation d'ensemble suivante : $N \supseteq N^1 \supseteq \ldots \supseteq N^{k-1}$, telle définie par (Agnetis et al., 2014). Les agents locaux sont en concurrence comme dans le cas « Compétition », ils sont aussi en conflit avec l'agent global, même dans le cas où leurs critères sont de même nature.

4. **Non-disjoint (ND)** C'est le cas le plus général, et les relations qui lient les agents entre eux peuvent être différentes de celles des autres scénarios, voir la figure 2.1.

2.5.3 Optimisation multicritère

La formulation générale des problèmes multicritères est la suivante :
— Soient $z^1(.), z^2(.), \ldots, z^K(.)$ des fonctions objectifs à maximiser.

$$\begin{cases} \text{Maximiser} : F(x) = (z^1(x), z^2(x), \ldots, z^K(x)) \\ x \in \Omega \end{cases}$$

— $x = (x_1, \ldots, x_n)$ représente le vecteur des variables de décision, nous précisons que sa dimension n n'est pas forcément égale au nombre de critères K.
— L'espace Ω représente l'ensemble des solutions réalisables, il peut être défini par un ensemble de contraintes vérifiées par chacune des solutions.

La méthode d'optimisation à adopter est un choix crucial. Selon la difficulté du problème, il peut exister un large panel de méthodes à appliquer. Afin d'orienter ce choix, certains critères comme la qualité des solutions obtenues ou le temps de calcul sont pris en compte.

Dans le cadre de problème d'optimisation multicritère la notion d'*optimalité* perd toute signification. En effet, les objectifs sont en général contradictoires au sens où l'augmentation de la valeur d'un critère implique la dégradation de la valeur d'au moins un autre critère. C'est pourquoi, une solution optimale $x^* = (x_1^*, \ldots, x_K^*)$ telle que $z_k(x_k^*) = max_{y \in \Omega}\{z^k(y_k)\}$ et $k = 1, \ldots, K$ est généralement rare. Ainsi est introduite la notion de "compromis" ou encore de "non-dominance". Plus de détails sont proposés dans la section ci-après.

2.5.4 Concept de dominance

La notion de dominance a été introduite par (Pareto, 1897) et initialement utilisée en économie. Elle est désormais utilisée dans d'autres domaines à l'instar de l'optimisation multicritère. Cette notion exprime une idée assez intuitive : voulant choisir entre deux vecteurs v_1 et v_2, nous choisirons celui qui est meilleur ou égal sur toutes les composantes tout en étant strictement meilleur sur au moins une d'entre elles. Si un tel choix ne peut être fait alors les vecteurs seront considérés non-dominés entre eux. Ainsi, partant d'un ensemble de décisions X, l'objectif est d'introduire un ordre total [3] entre les solutions de manière à pouvoir faire un choix. Une solution non-dominée est aussi désignée par *solution de Pareto* ou bien *solution efficace.*

Definition 1. *Soient deux solutions réalisables x et y pour un problème multicritère donné, $x = (x_1, \ldots, x_n)$ et $y = (y_1, ..., y_n)$: y est dit dominé par x, ou bien x domine y si et seulement si les conditions suivantes sont vérifiées : $z^1(x) \geq z^1(y), \ldots, z^k(x) \geq z^k(y)$ et $\exists k \in \{1, \ldots, K\}$ tel que $z^k(x_k) > z^k(y_k)$*

Ainsi, une solution qui n'est dominée par aucune autre solution est dite Pareto optimale, et inversement, si une solution est dominée par au moins une seule autre solution, alors elle ne peut être considérée Pareto optimale, d'où la définition suivante :

3. On appelle relation d'ordre total sur un ensemble E toute relation d'ordre telle que tout élément de E soit comparable avec tout autre élément de E.

Definition 2. *une solution* $x^* = (x_1^*, \ldots, x_K^*) \in \Omega$ *est Pareto optimale si et seulement si :* $\nexists y \in \Omega, \forall k \in \{1, \ldots, K\}$ *tel que* $z^k(y) \geq z^k(x^*)$ *avec* $z^h(y) > z^h(x^*), h \in \{1, \ldots, K\}$

2.5.5 Degré de dominance

Definition 3 (Dominance faible de Pareto)**.** *Soient deux solutions* x *et* y*, on dit que* x *domine faiblement (au sens faible)* y*, si et seulement si,* $\forall k \in \{1, \ldots, K\}$ *on a :* $z^k(x) \geq z^k(y)$ *tel que* $\exists k \in \{1, \ldots, K\}$ *et tel que* $z^k(x) > z^k(y)$

Definition 4 (Dominance stricte de Pareto)**.** *Soient deux solutions* x *et* y*, on dit que* x *domine strictement (au sens strict)* y*, si et seulement si,* $\forall k \in \{1, \ldots, K\} z^k(x) > z^k(y)$

Par abus de langage, la dominance stricte est souvent désignée par dominance de Pareto. L'ensemble de solutions optimales de Pareto est aussi désigné par *front de Pareto*, on parle de *cône de Pareto* si les solutions sont toutes strictement non-dominées.

2.5.6 Front Optimal

La résolution d'un problème multicritère consiste à déterminer le front optimal de Pareto, ce dernier est ensuite proposé au décideur à qui revient la tâche de choisir la ou les solutions qui lui conviennent le plus. Une question se pose alors sur la possibilité d'identifier toutes ces solutions de Pareto. On distingue deux situations (Dhaenens, 2005) :

1. *Front optimal complet* : le problème admet un nombre exponentiel de solutions de Pareto, ou bien un nombre très important.
2. *Front minimal complet* : le problème admet un nombre fini et maîtrisable de solutions de Pareto.

Dans le premier cas, on privilégie souvent l'obtention d'un sous-ensemble représentatif du front total. Des propriétés comme la diversité, les distances entre les solutions sont alors nécessaires pour évaluer la qualité du front (distance Euclidienne, hypervolume, nombre de solutions, etc.). Dans le second cas, il est possible de présenter toutes les solutions afin que le décideur puisse choisir.

2.5.7 Points de référence

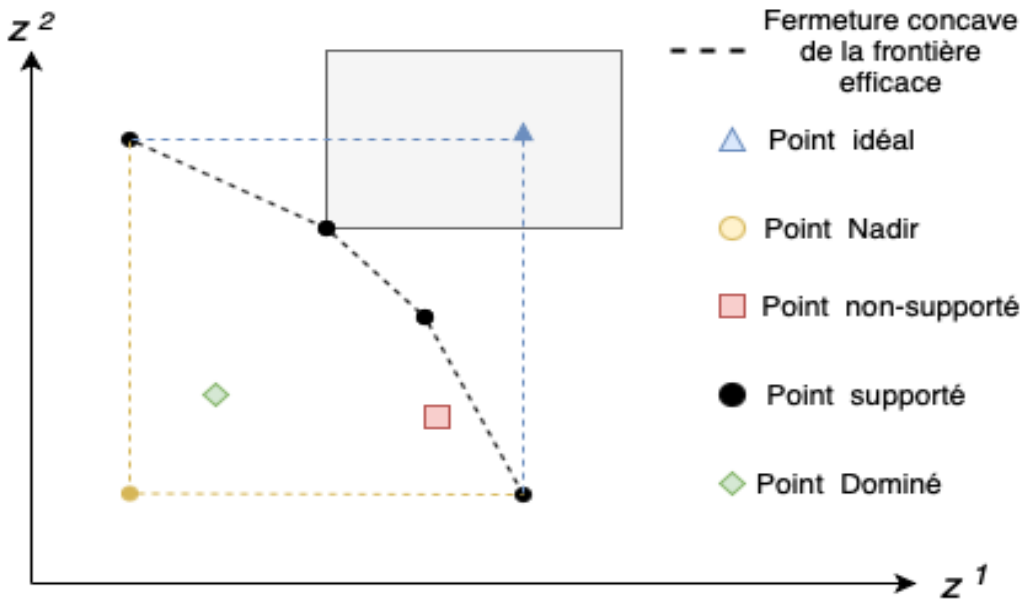
On associe à un problème multicritère, deux espaces :

1. Espace des objectifs.
2. Espace des solutions décisionnelles.

Un point dans l'espace des objectifs peut correspondre à plus d'une solution dans l'espace des décisions, mais une décision ne peut mener qu'à un seul résultat (valeurs des objectifs).

Les meilleures décisions correspondent aux solutions Pareto optimales. Dans le cas bicritère, on représente l'ensemble des solutions par un plan dans $\mathbb{R}^2$ où : l'axe des abscisses est associé aux valeurs du premier critère (z^1) et l'axe des ordonnées à celles du second

FIGURE 2.2 – Points caractéristiques d'un problème de maximisation biobjectif.



critère (z^2) (voir figure 2.2). Le front de Pareto est l'ensemble des solutions non-dominées, généralement représenté par une fonction linéaire par morceaux.

Les points de Pareto sur le schéma sont représentés en noir. Pour vérifier qu'un point est sur le front de Pareto, on trace le cône de dominance (en gris), et on vérifie que cet espace ne contient aucune autre solution réalisable, autrement dit, que ce point n'est pas dominé par une autre solution.

Une solution de Pareto est « supportée » si et seulement si elle appartient à l'enveloppe concave [4] des solutions non-dominées.

Le point "Nadir", est un point qui souvent ne correspond pas à une solution réelle où tous les objectifs atteignent leur pire valeur. Si on note $\eta = (\eta_1, \ldots, \eta_K)$ le point Nadir, alors

$$\eta_k = min_{x \in \Omega} z^k(x), \forall k = 1, \ldots, K$$

Inversement, un point "idéal" est un point η^* où tous les objectifs atteignent leur meilleure valeur :

$$\eta_k^* = max_{x \in \Omega} z^k(x), \forall k = 1, \ldots, K$$

4. Une enveloppe concave de $\mathcal{S}$ est le plus petit espace concave qui le contient. Les éléments de l'enveloppe concave sont exactement les points x de $\mathcal{S}$ qu'on peut écrire sous la forme $x = \sum_{i=1}^{l} \lambda_i a_i$, expression pour laquelle les a_i sont aussi dans $\mathcal{S}$, l est un entier, et les λ_i sont des quantités réelles positives.

2.5.8 Approches de résolutions

Les problèmes d'ordonnancement multiagent peuvent être résolus dans un contexte multicritère. Dans le cas où on recherche une solution optimale de Pareto, nous intéressons généralement qu'aux solutions de Pareto stricts. L'ensemble des solutions de Pareto stricts est obtenu en trouvant les solutions de Pareto stricts une par une et de manière itérative. Enfin, en comptant le nombre de solutions optimales, l'objectif est de compter le nombre de solutions non dominées ou de donner une approximation de leur nombre.

Il existe plusieurs approches pour trouver une ou toutes les solutions Pareto optimales. Ces approches sont décrites en détail dans (T'kindt and Billaut, 2006). Nous présentons ici les approches les plus classiques utilisées dans la littérature sur l'ordonnancement multicritère. Les approches sont illustrées dans le cas de deux critères $K = 2$, mais elles peuvent facilement être étendues au cas général.

2.5.8.1 Combinaison linéaire des critères

Cette approche a été initiée par (Geoffrion, 1968), l'idée consiste à résoudre le problème maximisant la somme pondérée des critères :

$$max \sum_{k=1}^{K} w_k z^k(x) : w \in \Omega$$

Les solutions qui peuvent être obtenues par cette approche constituent un sous-ensemble de l'ensemble des solutions optimales strictes de Pareto. En d'autres termes, il peut être impossible de fixer des poids aux critères de manière à obtenir toutes les solutions optimales de Pareto. Cela est dû à la forme de la courbe de compromis, qui peut être non convexe, donc les solutions optimales de Pareto non suportées ne peuvent pas être retournées par une telle approche.

2.5.8.2 Approche ε-contrainte

Cette approche est souvent utilisée dans la littérature. Par exemple, dans le cas de deux fonctions objectifs, la première est maximisée et l'autre est bornée. Dans le cas de plus de deux objectifs, le problème à résoudre est de : trouver x, tel que $z^1(x)$ est maximisée et $z^2(x) \geq Q_2, \ldots, z^k(x) \geq Q_k$. Cette approche conduit à une solution optimale de Pareto faible. Pour obtenir une solution Pareto optimale stricte, il faut résoudre un problème symétrique. Dans le cas de deux agents, nous notons z^A le critère pour l'agent A à maximiser sous contrainte de l'agent B noté $z^B \leq Q$. En modifiant Q de manière itérative, il est possible d'obtenir l'ensemble des solutions Pareto optimalesstrictes. Notez que plusieurs algorithmes ont été proposés dans la littérature pour améliorer l'implémentation d'un tel processus (Laumanns et al., 2006; Mavrotas, 2009). L'algorithme 1 illustre cette approche dans le cas de K fonctions objectives. Les δ_k sont le pas à chaque itération qui dégrade Q_k, lb_k et ub_k sont les bornes inférieures et supérieures de la fonction objectif z^k et la fonction lex-max(f) retourne la solution avec une valeur lexicographique maximale, i.e. avec un maximum z^1 puis un maximum z^2, etc.

Algorithm 1 Approche ε-contrainte

$\mathcal{R} = \emptyset$
for $Q_2 = ub_2$ à lb_2 par pas de δ_2 **do**
  for $Q_3 = ub_3$ à lb_3 par pas de δ_3 **do**
    $\vdots$
    for $Q_k = ub_k$ à lb_k par pas de δ_k **do**
      $x = lex - max\{z : z^k(x) \geq Q_k, 2 \leq k \leq K\}$
      if $\nexists x^{'} \in \mathcal{R}$ tel que $z(x^{'}) \geq z(x)$ **then**
        $\mathcal{R} = \mathcal{R} \cup \{x\}$
      end if
    end for
    $\vdots$
  end for
end for
return $\mathcal{R}$ =0

L'algorithme 1 a une complexité temporelle de $O(ub^K)$ avec $ub = max_{1 \leq k \leq K}(ub_k - lb_k)$.

2.5.8.3 Approche lexicographique

Cette méthode consiste à minimiser les fonctions objectif les unes après les autres en complétant les contraintes du problème au fur et à mesure. Ainsi, pour résoudre le problème $max(z^1(x), \ldots, z^k(x)) : x \in \Omega$ la procédure est la suivante :

1. On commence par résoudre le problème mono-critère $maxz^1(x) : x \in \Omega$, on note z^{1*} la valeur optimale retournée, on résout ensuite le problème $maxz^2(x) : x \in \Omega \; et \; z^1(x) = z^{1*}(x)$, telle que la valeur du premier critère reste constante. A la dernière étape on résout le problème $maxz^k(x) : x \in \Omega \; et \; z^1(x) = z^{1*}(x), \ldots, z^{k-1}(x) = z^{k-1*}(x)$.
2. Là on demande au décideur de faire un choix préférentiel entre les critères, une sorte de classement, et la solution renvoyée par la méthode est tributaire de ce classement. Ceci rend l'approche fragile, et parfois difficile à appliquer, notamment quand le décideur n'arrive pas à choisir entre deux critères.

2.6 Conclusions du chapitre

Dans ce chapitre, nous avons présenté quelques travaux connexes sur les problèmes d'ordonnancement d'intervalles fixes. Par la suite, nous avons introduit différentes notions nécessaires à la compréhension des problématiques d'ordonnancement multiagent. Enfin, nous avons illustré les problèmes d'ordonnancement dans un contexte cloud. Dans la suite du manuscrit, nous allons nous focaliser sur le cas monocritère (par définition, un seul agent). Puis dans une seconde partie, nous étudions le cas de plusieurs agents. Afin d'illustrer nos différentes approches de résolution, les études menées sont pour le cas de deux agents. Nous indiquons lorsqu'il faut si nos approches sont généralisables et comment au cas de plusieurs agents.

Chapitre 3

Méthodes exactes pour l'ordonnancement monocritère sur machines parallèles multiressources

3.1 Introduction

Dans ce chapitre, nous étudions un problème d'ordonnancement de n travaux fixes, indépendants non-préemptifs sur m machines parallèles identiques. Différents types de ressources additionnelles et renouvelables sont nécessaires pour traiter chaque travail. Chaque machine peut traiter plusieurs travaux à la fois. L'objectif est de maximiser le nombre pondéré des travaux exécutés (équivalent à, minimiser le coût total de rejets des travaux non ordonnancés).

L'objectif de ce chapitre est de développer des méthodes exactes pour la résolution du problème d'ordonnancement monocritère. Nous présentons d'abord quelques définitions accompagnées des notations utilisées dans la suite du document. Nous analysons la complexité des problèmes étudiés. Pour la résolution exacte, nous développons dans la section 3.4 trois programmes linéaires en nombres entiers $PLNE$s. Un modèle basé sur la programmation par contrainte est introduit dans la section 3.5 permettant d'obtenir rapidement une bonne solution. On l'utilise ensuite comme solution initiale réalisable pour les $PLNE$s. Nous proposons par la suite une méthode par décomposition de type Branch & Price, décrite dans la section 3.6.2. Nous terminons ce chapitre par une analyse expérimentale des performances des méthodes exactes développées et une conclusion.

3.2 Définitions et notations

Dans un contexte de réseau informatique distribué, nous nous intéressons à l'ordonnancement de n travaux sur m machines parallèles identiques. Chaque machine dispose de $\mathcal{K}$ types de ressources additionnelles renouvelables. On note par $R_k, k = 1 \dots \mathcal{K}$ la quantité disponible de ressource k. On note par $r_{j,k}$ la quantité de ressource k requise pour exécuter le travail $j \in \mathcal{N}, j = 1, \dots, n$. Nous supposons que $r_{j,k} \leq R_k$ pour tout $j = 1, \dots, n$ et

$k = 1, \ldots, \mathcal{K}$. Pour chaque travail j, sa date de début s_j et sa date de fin f_j sont fixées et le travail doit être exécuté sur toute la durée de l'intervalle $[s_j, f_j]$. Ainsi, sa durée opératoire est donnée par : $p_j = f_j - s_j$. Les travaux n'ont pas tous la même priorité. w_j est le poids du travail j. A un instant t donné, chaque machine peut traiter plusieurs travaux à la fois, sous contrainte de la disponibilité des quantités de ressources requises. Ces machines sont disponibles en permanence durant l'horizon du temps considéré $[0, T]$. Toutes les données sont supposées entières et positives. Soit $x_{i,j}$ une variable de décision binaire telle que : $x_{i,j} = 1$ si le travail j est exécuté sur la machine i ; $x_{i,j} = 0$ sinon. L'objectif est de maximiser le nombre pondéré de jobs exécutés, donné par : $\max \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{i,j}$.

Selon la notation des problèmes d'ordonnancement en trois champs $\alpha|\beta|\gamma$, le problème étudié est noté : $Pm|s_j, f_j, r_{j,k}| \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{N}} w_j x_{i,j}$.

Nous rappelons ci-dessous quelques définitions importantes pour la suite de ce document.

Definition 5. *Un ensemble d'intervalles est dit propre, si aucun intervalle n'est inclus dans un autre ensemble d'intervalles.*

Definition 6. *Un graphe d'intervalles est le graphe d'intersection d'intervalles sur une droite. Donc, étant donné un ensemble $E = E_1, \ldots, E_n$ d'intervalles sur une droite, on lui associe le graphe d'intervalles $G = (V, U)$ où $V = 1, \ldots, n$ et deux sommets v_1 et v_2 sont reliés par une arête si et seulement si $E_{v_1} \cap E_{v_2} \neq \emptyset$. Une représentation du graphe G par de tels intervalles est appelée représentation d'intervalle. Un graphe d'intervalles propre est un graphe d'intervalles possédant une représentation d'intervalles dans laquelle aucun intervalle n'est inclus dans l'autre.*

3.2.1 Construction de l'ensemble de cliques maximales

Pour une instance I donnée du problème que nous traitons, nous construisons un graphe d'intervalles $G = (V, U)$ de la manière suivante : pour chaque travail j, un sommet v_j correspondant, deux sommets v_i et v_j sont reliés par une arête, si les intervalles de temps des travaux i et j se chevauchent. Si un sous-ensemble de travaux de I a la propriété définie dans la définition 5, alors le graphe G est évidemment un graphe d'intervalles propres.

La figure 3.1 représente la construction d'un graphe d'intervalles à partir d'une instance (exemple de l'instance 1.2) de huit travaux à ordonnancer.

Pour formuler le problème d'allocation des ressources en termes de graphe d'intervalle, nous donnons la définition suivante.

Definition 7. *Une clique d'un graphe G est un sous-graphe complet de G. Une clique maximale est une clique qui n'est pas proprement contenue dans une autre clique.*

Les cliques maximales d'un graphe d'intervalles peuvent être déterminées efficacement en $O(n^2)$ (voir (Booth and Lueker, 1976; Habib et al., 2000)). On note par $\mathcal{L} = \bigcup_h L_h$ (L_h est la notation de la $h$ème clique maximale). Soit $|\mathcal{L}| = H$, le nombre de cliques maximales dans un graphe d'intervalles. La figure 3.2 présente un exemple des cliques maximales obtenues à partir d'une instances de huit travaux.

FIGURE 3.1 – Graphe d'intervalle correspondant à huit travaux.

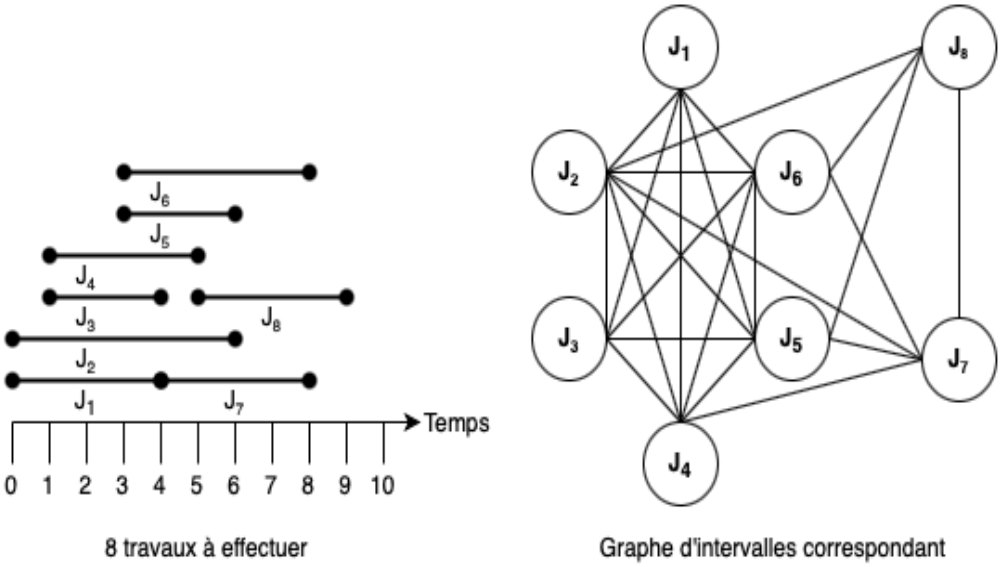
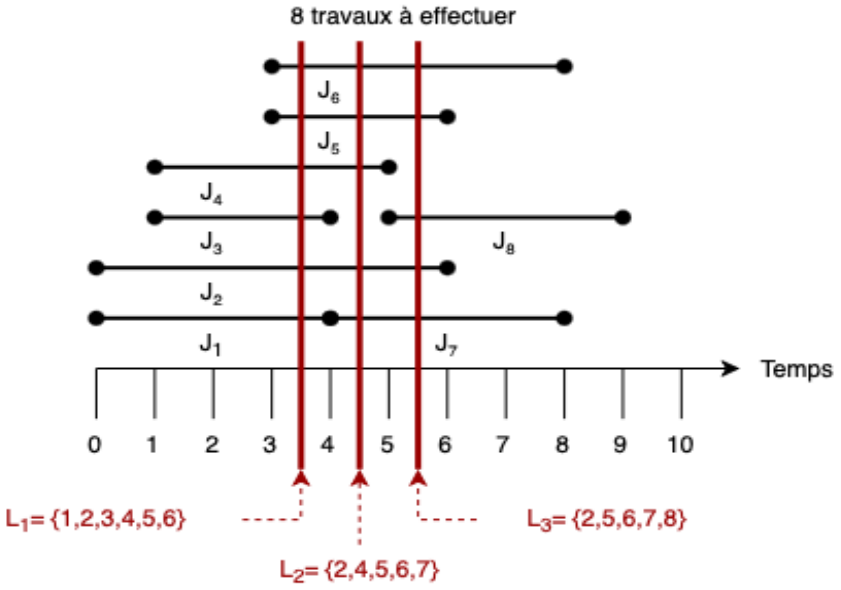


FIGURE 3.2 – Cliques maximales sur une instance de huit travaux.



3.3 Analyse de complexité

Le problème d'ordonnancement d'intervalles fixes sur machines parallèles multiressources est déjà difficile pour des cas particuliers. Même lorsqu'on considère une seule ressource additionnelle avec une capacité totale $R = 2$, le problème est prouvé NP-difficile au sens fort (Angelelli and Filippi, 2011).

Dans le cas d'une seule machine, nous pouvons montrer que lorsque tous les travaux ont la même date de début et de fin, le problème se réduit au cas de sac à dos multi-dimensionnel (MKP) ; sinon le problème se réduit au cas d'allocation de ressources (RAP). Ces deux problèmes sont connus comme des problèmes d'optimisation combinatoire NP-difficiles.

3.4 Programmation linéaire en nombres entiers PLNE

Cette section est dédiée au développement de trois formulations mathématiques en nombres entiers. Soient, $s_{min} = min_{1 \leq j \leq n}(s_j)$ et $f_{max} = max_{1 \leq j \leq n}(f_j)$.

3.4.1 PLNE-1

Le premier programme linéaire en nombres entiers proposé est noté $PLNE$-1. Dans $PLNE$-1 les variables sont de type "indexé-machine". Pour les trois $PLNE$, nous utilisons les variables de décision binaires suivantes :

$$x_{i,j} = \begin{cases} 1, & \text{Si le travail } j \text{ est exécuté sur la machine } M_i; \\ 0, & \text{Sinon.} \end{cases}$$

Notons que pour un travail j si $\forall i \in \mathcal{M}$, $x_{i,j} = 0$ alors j est rejeté.

La formulation générale est donnée comme suit :

$$\text{Maximize : } \sum_{i=1}^{m} \sum_{j=1}^{n} w_j \; x_{i,j} \tag{3.1}$$

$$\sum_{j : s_j \leq t < f_j} r_{j,k} x_{i,j} \leq R_k \; (k = 1, \ldots, K; i = 1, \ldots, m; t = s_{min}, \ldots, f_{max}) \tag{3.2}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \; (j = 1, \ldots, n) \tag{3.3}$$

$$x_{i,j} \in \{0, 1\} \; (i = 1, \ldots, m; j = 1, \ldots, n) \tag{3.4}$$

Les contraintes 3.2 sont des contraintes de disponibilité des ressources à tout instant $s_{min} \leq t < f_{max}$, i.e. à chaque instant t l'exécution des travaux sur la machine M_i ne doit pas dépasser la capacité totale de chaque ressource R_k disponible. Les contraintes 3.3 assurent qu'un travail j est affecté au plus à une machine.

Le tableau 3.1 résume les contraintes, leur nombre et le nombre de variables binaires.

TABLE 3.1 – Caractéristiques du modèle $PLNE$-1.

Problème	Z	#Var. bin.	Contraintes
$Pm\|s_j, f_j, r_{j,k}\| \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{i,j}$	$\sum_{i=1}^{m} \sum_{i=1}^{n} w_j \; x_{i,j}$	mn	$n + mK(f_{max} - s_{min})$

Remarque 1. *Les dates de début et de fin des travaux ne sont pas explicitement requises dans le modèle (PLNE-1). Néanmoins, nous devons vérifier à chaque instant durant l'exécution d'un travail j la disponibilité des ressources de la machine sur laquelle il est traité. Cet intervalle de temps est borné par la longueur $f_{max} - s_{min}$.*

3.4.2 PLNE-2

Pour gérer les disponibilités des ressources à tout instant t jusqu'au prochain événement défini par la date de fin du travail, nous proposons un deuxième $PLNE$ indexé sur le temps avec les variables de décision binaires suivantes :

$$y_{i,j,t} = \begin{cases} 1, & \text{Si le travail } j \text{ est exécuté sur la machine } M_i \text{ à l'instant } t; \\ 0, & \text{Sinon.} \end{cases}$$

$$\text{Maximize} : \sum_{i=1}^{m}\sum_{j=1}^{n} w_j \; x_{i,j} \tag{3.5}$$

$$\sum_{j=1}^{n} r_{j,k} \; y_{i,j,t} \leq R_k \; (k = 1, \ldots, K; i = 1, \ldots, m; t = s_{min}, \ldots, f_{max}) \tag{3.6}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \; (j = 1, \ldots, n) \tag{3.7}$$

$$\sum_{t=s_j}^{f_j - 1} y_{i,j,t} = (f_j - s_j) \; x_{i,j} \; (i = 1, \ldots, m; j = 1, \ldots, n) \tag{3.8}$$

$$x_{i,j} \; ; \; y_{i,j,t} \in \{0, 1\} \; (i = 1, \ldots, m; j = 1, \ldots, n; t = s_{min}, \ldots, f_{max}) \tag{3.9}$$

Les contraintes 3.6 assurent le respect de la disponibilité des ressources R_k à tout instant t durant l'horizon de planification. Les contraintes 3.7 assurent qu'un travail j est affecté au plus à une machine. Les contraintes 3.8 indiquent que si un travail j n'est pas rejeté alors il est exécuté durant tout son intervalle de temps.

Le tableau 3.2 résume les contraintes, leur nombre et le nombre de variables binaires.

TABLE 3.2 – Caractéristiques du modèle $PLNE$-2 indexé temps.

Problème	Z	#Var. bin.	Contraintes
$Pm\|s_j, f_j, r_{j,k}\| \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{i,j}$	$\sum_{i=1}^{m}\sum_{i=1}^{n} w_j \; x_{i,j}$	$mn(1 + f_{max} - s_{min})$	$n + mk(f_{max} - s_{min}) + mn$

Remarque 2. *Les résultats expérimentaux présentés dans la section 3.7.2 montrent les limites de ce deuxième modèle mathématique pour résoudre des problèmes de taille importante. Il dépend fortement de l'horizon de planification discrétisé.*

Notre idée est de développer un nouveau modèle qui vérifie à des moments clés la disponibilité des ressources. Ce troisième programme linéaire en nombres entiers ($PLNE$-3) se base sur le graphe d'intervalles.

3.4.3 PLNE-3

La troisième formulation mathématique utilise la notion de cliques maximales. Ainsi, nous pouvons associer aléatoirement à chaque clique maximale L_h un instant t_h, date commune à tous les travaux de l'ensemble L_h ($t_h \in \cap_{j \in L_h}[s_j, f_j)$). Nous pouvons donc définir un ordre total des cliques maximales selon t_h. Suivant cet ordre, lorsque nous passons d'une clique maximale à la suivante, au moins un travail se termine et au moins un nouveau travail débute son traitement. En conséquence, nous avons au plus n cliques maximales.

$$\text{Maximize} : \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{i,j} \tag{3.10}$$

$$\sum_{j \in L_h} r_{j,k} x_{i,j} \leq R_k \qquad i = 1, \ldots, m;\ k = 1, \ldots, \mathcal{K};\ h = 1, \ldots, H \tag{3.11}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \qquad j = 1, \ldots, n \tag{3.12}$$

$$x_{i,j} \in \{0, 1\} \qquad j = 1, \ldots, n;\ i = 1, \ldots, m \tag{3.13}$$

Les contraintes 3.11 permettent de respecter la limitation des ressources lors de traitement des travaux appartenant à la même clique maximale. Les contraintes 3.12 permettent l'affectation d'un travail à au plus une seule machine.

Le tableau 3.3 résume les contraintes, leur nombre et le nombre de variables binaires.

TABLE 3.3 – Caractéristiques du modèle $PLNE$-3 contraintes de ressources.

Problème	Z	#Var. bin.	Contraintes
$Pm\|s_j, f_j, r_{j,k}\| \sum_{i=1}^{m}\sum_{j=1}^{n} w_j x_{i,j}$	$\sum_{i=1}^{m}\sum_{i=1}^{n} w_j\ x_{i,j}$	mn	$n + mKH$

Le $PLNE$-3 est un modèle indexé-machine à $n + mkH$ contraintes, où $1 \leq H \leq n$. Si $H = n$ alors chaque clique maximale contient un travail. Dans ce dernier cas, le problème est trivial à résoudre. D'un autre côté, si $H = 1$ alors tous les travaux se chevauchent et le problème se réduit à un problème de sac à dos multidimensionnel MKP (Martello, 1990).

Remarque 3. *L'analyse expérimentale des performances des trois PLNE (cf section 3.7.2), montre que pour des instances importantes, ces PLNE n'arrivent pas à trouver une solution réalisable au bout d'une heure de calcul. Afin d'augmenter les performances de ces PLNEs, notre idée est de développer une méthode efficace permettant de générer des solutions initiales réalisables pour ces PLNEs. Notre choix s'est porté alors sur la programmation par contraintes.*

3.5 Programmation par contraintes

La programmation par contraintes (PPC) est une technique qui facilite la modélisation pour résoudre des problèmes combinatoires complexes en utilisant une description déclarative ; elle provient de la programmation logique et de l'intelligence artificielle. L'idée est de

séparer la déclaration des contraintes à l'aide d'un langage de contraintes riche en processus de recherche de solutions basé sur une utilisation active des contraintes pour réduire l'espace de recherche (propagation des contraintes). La PPC est basée sur des problèmes de satisfaction de contraintes (CSP), dans lesquels il faut satisfaire un groupe de contraintes, en attribuant des valeurs aux variables (Rossi et al., 2006).

Ce langage de modélisation déclaratif est un atout pour la programmation par contraintes où elle accepte tout type de relation pour formuler les contraintes d'un CSP, comprenant les inégalités linéaires, mais aussi les contraintes logiques (par exemple, sous forme de clause $x = 5 \vee y = 2 \vee z \neq 3$ ou d'implication $x = 0 \implies y \geq 1 \wedge y + Z < 2$), ou encore les contraintes globales qui sont un moyen de formuler succinctement un groupe de contraintes, comme par exemple la contrainte globale *cumulative* qui permet de modéliser l'occupation d'une ressource dans n'importe quel problème d'ordonnancement. En effet, ce type de modélisation, avec en particulier les contraintes globales, induit une décomposition du problème sur laquelle est basée le mode de résolution de la PPC. A chaque contrainte globale, est associée un ou plusieurs algorithmes de filtrage spécifiques.

Dans cette section nous proposons de modéliser notre problème par la PPC à l'aide des contraintes globale du CPOptimizer (CPO), nous formulons notre problème comme un problème d'ordonnancement à ressource cumulative (CuSP pour Cumulative Scheduling Problem). Les travaux sont représentés par des variables d'intervalles, notés $ITer_{ij}$. Une variable d'intervalle est une variable de décision dont la valeur est un intervalle de nombres entiers. Ainsi, la valeur d'une variable d'intervalle est caractérisée par deux nombres entiers :

— le premier nombre entier indique le début de l'intervalle : il s'agit donc des s_j,
— le deuxième nombre entier indique la fin de l'intervalle : il s'agit des f_j.

La longueur de l'intervalle doit être égale à la durée opératoire du travail $p_j = f_j - s_j$. Une caractéristique supplémentaire importante des variables d'intervalle est le fait qu'elles peuvent être optionnelles, c'est-à-dire que l'on peut décider de ne pas les prendre en compte dans la solution. Avec la fonction membre *setOptional*() de CPO, lorsqu'un intervalle est pris en compte dans la solution, l'intervalle est dit présent, sinon il est absent. Les autres fonctions de CPO que nous avons utilisées dans notre modèle PPC sont :

— $IloPresenceOf()$: est une contrainte logique sur un intervalle. Cette fonction renvoie une contrainte indiquant que la variable d'intervalle est présente.
— $IloCumulFunctionExpr()$: est l'expression de fonction cumulée dont la valeur dans une solution est la somme des contributions individuelles des intervalles, cette fonction est représentée dans la formulation ci-dessous par $CumF$.
— $IloPulse()$: est une fonction d'impulsion élémentaire définie par une variable d'intervalle (ou un intervalle fixe) dont la valeur est égale à 0 en dehors de l'intervalle et égale à une constante non négative sur l'intervalle. Cette valeur est appelée la hauteur de la fonction d'impulsion.

La formulation générale par CPO du problème que nous traitons est comme suit :

$$\text{Maximize} : \sum_{i=1}^{m}\sum_{j=1}^{n} w_j * IloPresenceOf(ITer_{i,j}) \tag{3.14}$$

$$CumF_{i,k} = \sum_{k \in K} IloPulse(ITer_{i,j}, r_{j,k})\ k = \overline{1,K};\ i = \overline{1,m};\ j = \overline{1,n} \tag{3.15}$$

$$CumF_{i,k} \leq R_k \qquad i = \overline{1,m};\ k = \overline{1,K} \tag{3.16}$$

$$\sum_{i=1}^{m} IloPresenceOf(ITer_{i,j}) \leq 1 \qquad j = \overline{1,n} \tag{3.17}$$

$$setOptional(ITer_{i,j}) \quad j = \overline{1,n};\ i = \overline{1,m} \tag{3.18}$$

Le $CumF_{i,k}$ dans 3.15 et 3.16 représente la fonction cumulatif indiquant l'utilisation de la ressource k sur la machine i au fil du temps. $IloPulse(ITer_{i,j}, r_{j,k})$ est une expression de la fonction cumulative élémentaire, elle couvre l'utilisation d'une ressource k sur une machine i, à la date de début s_j d'un travail, la fonction cumulatif augmente et à la date de fin f_j du même travail, les ressources sont libérées, donc la fonction cumulatif diminue. Les contrainte 3.16 représentent les capacités totales à ne pas dépasser par la fonction cumulatif. Les contraintes 3.17 permettent d'affecter une variable d'intervalle $ITer_{i,j}$ à une seule machine.

A l'aide de la formulation ci-dessus, nous développons une approche basique d'hybridation $PPCPLNE$ (Hajian et al. (1998)) où la PPC sert d'heuristique pour déterminer rapidement une solution réalisable de départ, à la racine uniquement d'un $Branch\&Bound$.

L'analyse expérimentale présentée en détail dans la section 3.7.3, montre que le $PLNE$-3 permet en effet de retourner maintenant une solution réalisable au bout de 30 minutes de calcul pour les instances de très grandes tailles. Cependant, aucune instance n'a pu être résolue à l'optimum.

3.6 Méthodes par décomposition

Les méthodes dites de décomposition en Recherche Opérationnelle sont des techniques efficaces pour résoudre certains problèmes d'optimisation combinatoire difficiles. Le principe de ces méthodes se résume en deux étapes importantes : séparation en plusieurs sous-problèmes qui peuvent être résolus efficacement, développement de techniques itératives de recomposition pour déterminer une solution du problème initial. On peut citer par exemple, la relaxation continue et génération de coupes, ou bien la décomposition de Danzig-Wolfe et la génération de colonnes.

Dans notre étude, la relaxation continue et la génération de coupes ne nous a pas permis d'avoir des résultats concluants. La convergence de cette procédure est trop lente. La relaxation linéaire d'un modèle comme le $PLNE$-3 n'est pas bonne. Pour cette raison, nous proposons une décomposition de Dantzig-Wolfe sur ce modèle conduisant à une approche de génération de colonnes, ainsi qu'une méthode Branch & Price, que nous détaillons dans la suite de cette section.

3.6.1 Décomposition Dantzig-Wolfe

Le modèle $PLNE$-3 est compact, mais il contient beaucoup de symétries. Soit une solution réalisable $X = [x_{i,j}]$, toute permutation des lignes de X (i.e toute permutation des indices des machines) aboutit à une solution réalisable différente avec la même valeur de la fonction objectif.

Considérons maintenant le problème RAP obtenu en fusionnant toutes les machines m dans une seule avec des unités de ressources mR_k disponibles. Nous proposons la formulation mathématique suivante, notée $PLNE$-4 :

$$y_j = \begin{cases} 1, & \text{Si le travail } j \text{ est exécuté ;} \\ 0, & \text{Sinon.} \end{cases}$$

$$\text{Maximize : } \sum_{i=1}^{n} w_j y_j \tag{3.19}$$

$$\sum_{j \in L_h} r_{j,k} y_j \leq mR_k \; (k = 1, \ldots, K; h = 1, \ldots, H) \tag{3.20}$$

$$y_j \in \{0, 1\} \; (j = 1, \ldots, n) \tag{3.21}$$

Proposition 1. *Les relaxations linéaires des modèles $PLNE$-3 et $PLNE$-4 sont équivalentes.*

Preuve. *Soit $\overline{x} = [\overline{x_{i,j}}]$ une solution réalisable de la relaxation linéaire du modèle $PLNE$-3, et soit $\overline{y_j} = \sum_{i=1}^{m} \overline{x_{i,j}}$ pour tout j. Alors $\overline{y} = [\overline{y_j}]$ est une solution réalisable de la relaxation linéaire de $PLNE$-4 avec la même valeur de la fonction objectif.*

Inversement, soit $\hat{y} = [\hat{y_j}]$ une solution réalisable de la relaxation linéaire de $PLNE$-4, et soit $\hat{x_{i,j}} = \hat{y_j}/m$ pour tout i et j. Alors, $\hat{x} = [\hat{x_{i,j}}]$ est une solution réalisable de la relaxation linéaire de $PLNE$-3, avec la même valeur de la fonction objectif.

Notons qu'à ce stade, nous ne nous attendons pas à ce que la relaxation linéaire de $PLNE$-3 soit "bonne", car elle néglige le fait que les unités de ressources sont réparties entre les différentes machines.

Nous proposons d'appliquer la décomposition de Dantzig-Wolfe sur le modèle $PLNE$-3 et une approche de génération de colonnes pour le résoudre selon la même démarche proposée dans (Caprara et al., 2013) pour résoudre le problème de sac-à-dos temporel TKP.

Soit $\mathcal{P}$ l'ensemble des solutions faisables (ordonnancements possibles) sur une seule machine, défini par l'équation suivante :

$$\mathcal{P} = \{Y \in \{0, 1\}^n : \sum_{j \in L_h} r_{j,k} y_j \leq R_k \; (k = 1, \ldots, K; h = 1, \ldots, H)\} \tag{3.22}$$

Nous pouvons ainsi définir le problème maître explicite PM-1 suivant :

$$\text{Maximize : } \sum_{i=1}^{m} \sum_{j=1}^{n} w_j x_{i,j} \tag{3.23}$$

$$x_{i,j} = \sum_{Y^p \in \mathcal{P}} y_j^p \lambda_i^p \; (i = 1, \ldots, m; j = 1, \ldots, n) \tag{3.24}$$

$$\sum_{Y^p \in \mathcal{P}} \lambda_i^p \leq 1 \; (i = 1, \ldots, m) \tag{3.25}$$

$$\sum_{i=1}^{m} x_{i,j} \leq 1 \; (j = 1, \ldots, n) \tag{3.26}$$

$$\lambda_i^p \geq 0 \; (i = 1, \ldots, m; Y^p \in \mathcal{P}) \tag{3.27}$$

$$x_{i,j} \in \{0, 1\} \; (i = 1, \ldots, m; j = 1, \ldots, n) \tag{3.28}$$

Les variables λ dans PM-1 sont les coefficients de combinaisons convexes des vecteurs Y^p. Cependant, l'intégralité des variables x ainsi que les contraintes 3.26 impliquent que, dans toute solution réalisable, il faut sélectionner exactement un vecteur dans $\mathcal{P}$, pour chaque machine i. Il en découle que dans toute solution réalisable, les variables λ sont des entiers. Donc, nous pouvons supprimer les variables x et les remplacer par les variables z définies par $z_p = \sum_{i=1}^{m} \lambda_i^p$. Les z_p sont des variables binaires avec $z_p = 1$ si le vecteur est choisi dans la solution ; 0 sinon. Soit $c_p = \sum_{j=1}^{n} w_j y_j^p$. Pour chaque vecteur p, les variables binaires y_j^p sont définies comme suit : $y_j^p = 1$ si le travail j est ordonnancé ; 0 sinon. Nous déduisons donc le nouveau problème maître implicite PM-2 suivant :

$$\text{Maximize} : \sum_{Y^p \in \mathcal{P}} c_p z_p \tag{3.29}$$

$$\sum_{Y^p \in \mathcal{P}} y_j^p z_p \leq 1 \; (j = 1, \ldots, n) \tag{3.30}$$

$$\sum_{Y^p \in \mathcal{P}} z_p \leq m \tag{3.31}$$

$$z_p \in \{0, 1\} \; (Y^p \in \mathcal{P}) \tag{3.32}$$

Notez que les deux problèmes maîtres PM-1 et PM-2 sont équivalents. Il existe alors une correspondance entre les solutions réalisables du problème maître explicite et du maître implicite, et cette correspondance préserve la valeur de la fonction objectif. Nous préférons utiliser la forme implicite PM-2 car elle a peu de variables et elle présente une structure d'un problème dit de *set packing problem* (Vemuganti, 1998).

Cependant, la formulation mathématique PM-2 peut avoir un grand nombre de colonnes, donc nous proposons une approche de type Branch&Price pour le résoudre. Afin de définir le *pricing problème*, noté PP, considérons la relaxation linéaire du modèle PM-2, noté LPM-2. Soit $\pi_j (j = 1, \ldots, n)$ les variables duales associées aux contraintes 3.30 et soit ρ la variable duale associée à la contrainte 3.31. Soit une réduction du PM-2 (noté RPM-2), définie sur un sous-ensemble des colonnes $\overline{\mathcal{P}} \subset \mathcal{P}$, et soit $(\pi^\star, \rho^\star)$, la solution optimale du dual de la relaxation linéaire restreinte, noté $RLPM$-2. Soit $Y^p \in \mathcal{P} \backslash \overline{P}$ un sous-ensemble de colonnes donné (c-à-d. un ensemble de solutions initiales). Les coefficients des coûts réduits de z_y sont :

$$c_p - \sum_{j=1}^{n} \pi_j^\star y_j^p - \rho^\star \tag{3.33}$$

Ainsi, le problème de pricing PP correspondant est le suivant :

$$\text{Maximize} : \sum_{j=1}^{n}(w_j - \pi_j^\star)y_j \tag{3.34}$$

$$\sum_{j \in L_h} r_{j,k} y_j \leq R_k \; (k = 1, \ldots, K; h = 1, \ldots, H) \tag{3.35}$$

$$y_j \in \{0, 1\} \; (j = 1, \ldots, n) \tag{3.36}$$

Soit $\hat{Y}^p$ une solution réalisable du problème PP, et soit $\hat{\varphi}$ la valeur de la fonction objectif correspondante. Si $\hat{\varphi} > \rho_i^\star$ alors la colonne $\hat{Y}^p$ a un coût réduit positif, et peut donc être incluse dans $\overline{\mathcal{P}}$. D'autre part, si la valeur optimale de PP n'est pas supérieure à $\rho^\star$, alors la solution relâchée courante de MP-2 est optimale. Notons que si $w_j - \pi_j^\star \leq 0$ alors la variable y_j peut être supprimée du PP, donc nous pouvons supposer que les coefficients de la fonction objectif du PP sont positifs. Ainsi, le *pricing problem* PP n'est autre que le problème RAP, qui est NP-difficile.

3.6.2 Branch&Price

L'algorithme de Branch&Price est une technique de résolution bien connue pour traiter des programmes linéaires en nombres entiers avec un grand nombre de variables (colonnes). Cet algorithme est essentiellement de type Branch&Bound où, à chaque noeud de l'arbre de recherche, les variables (colonnes) du problème sont générées en appliquant l'approche de génération de colonnes pour traiter la relaxation linéaire du problème, en y ajoutant éventuellement des contraintes de branchement. Pour plus de détails sur la méthode de génération de colonnes, voir (Desaulniers et al., 2006).

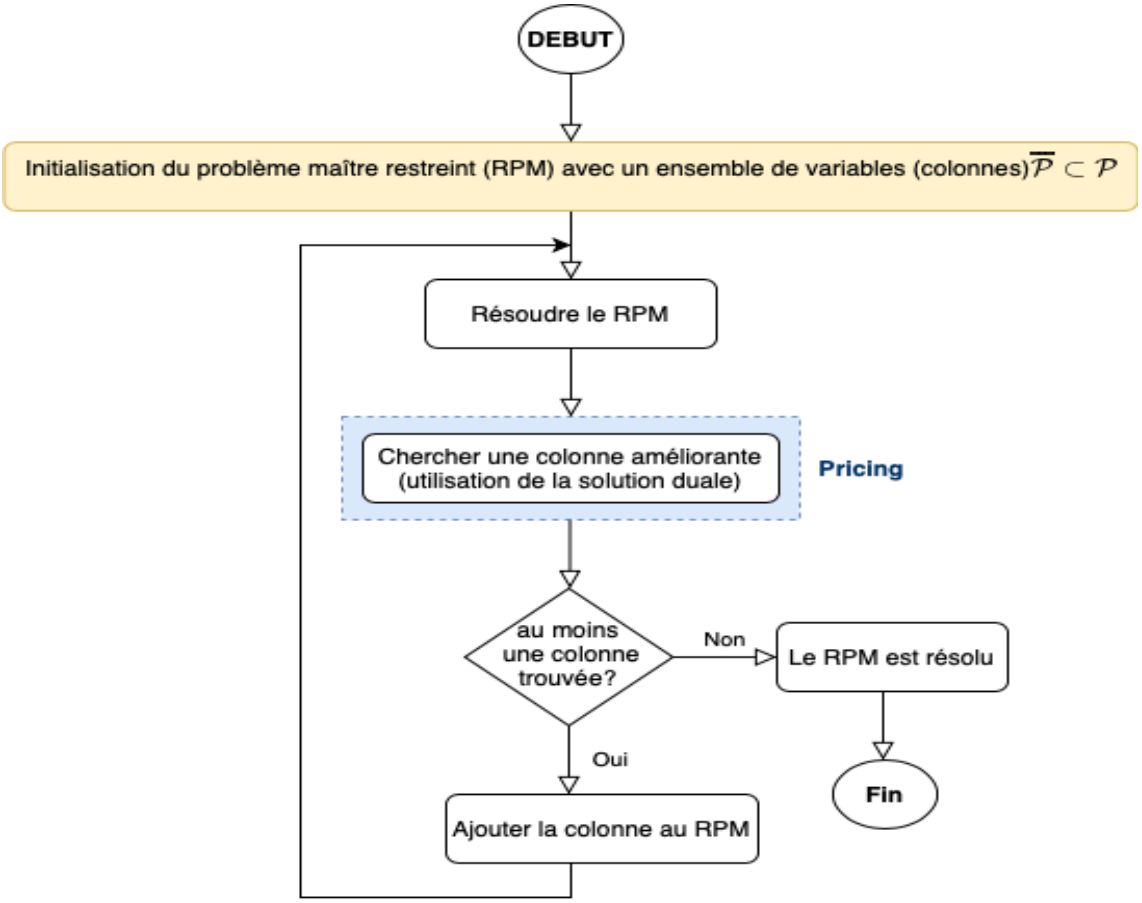
3.6.2.1 Génération de colonnes

La figure 3.3 représente l'enchaînement des différentes étapes d'une résolution par la génération de colonnes.

Le $RLMP$-2 est initialisé avec un ensemble $\overline{\mathcal{P}} \subset \mathcal{P}$ défini comme suit. Le noeud racine de l'arbre de Branch&Bound $\overline{\mathcal{P}}$ comprend une colonne fictive à coût élevé, par laquelle toutes les contraintes PM-2 sont satisfaites. Cette colonne est conservée dans le $RLMP$-2 jusqu'à ce que la faisabilité soit atteinte. Dans chaque noeud de l'arbre de recherche, $\overline{\mathcal{P}}$ doit inclure la colonne fictive et les colonnes réalisables générées jusqu'à présent, tout en respectant les contraintes de branchement.

Ensuite, de manière itérative, le $RLMP$-2 est résolu à l'optimum, et les PP sont résolus afin de générer de nouvelles colonnes de coûts réduits positifs. Les PP sont toujours résolus à l'optimum. Toutes les colonnes de coûts réduits positifs trouvées sont renvoyées au PM-2. Si aucune colonne positive de coûts réduits n'est trouvée, cela signifie que le $RLMP$-2 actuel est résolu.

FIGURE 3.3 – Schéma de résolution par génération de colonnes.



3.6.2.2 Schéma de branchement

Dans le cas où la solution optimale de la LMP-2 est fractionnaire, nous appliquons hiérarchiquement les règles de branchement suivantes.

Soit z^*, la solution fractionnaire optimale de la LMP-2 au noeud courant et soit $\mathcal{P}^*$ l'ensemble de colonnes correspondant à la fin du processus de résolution.

- Si $\eta = \sum_{\mathbf{Y^p} \in \mathcal{P}^*} z^*_{\mathbf{p}}$ est fractionnaire, nous effectuons des branchements sur le nombre total de machines à utiliser. Nous imposons :
  - $\sum_{\mathbf{Y^p} \in \mathcal{P}} z_{\mathbf{p}} \leq \lfloor \eta \rfloor$ sur une branche.
  - $\sum_{\mathbf{Y^p} \in \mathcal{P}} z_{\mathbf{p}} \geq \lceil \eta \rceil$ sur l'autre branche.
- Si η est un nombre entier, alors nous cherchons un travail ordonnancé de manière fractionnée. Étant donné le travail j' associé à la plus grande valeur fractionnaire $\sum_{\mathbf{Y^p} \in \mathcal{P}^*} y_{j'}\ z^*_{\mathbf{p}}$, on impose :
  - $\sum_{\mathbf{Y^p} \in \mathcal{P}} y_{j'}\ z_{\mathbf{p}} = 0$ sur une branche.

— $\sum_{\mathbf{Y^p} \in \mathcal{P}} y_{j'} \; z_{\mathbf{p}} = 1$ sur l'autre branche.

— Si aucun travail n'est ordonnancé de manière fractionnée, cela signifie qu'il doit exister deux travaux j' et j'' tels que $0 < \sum_{\mathbf{Y^p} \in \mathcal{P}^* | y^p_{j'} = y^p_{j''} = 1} z^*_p < 1$, dans ce cas, la règle de branchement de Ryan et Foster est appliquée (Ryan and Foster, 1981). Nous imposons :

  — Dans une branche, les travaux j' et j'' doivent être dans des ordonnancements différents, ce qui correspond à rajouter la contrainte $y^p_{j'} + y^p_{j''} \leq 1$ dans le *pricing problem*).

  — Dans l'autre branche, les travaux j' et j'' doivent se trouver dans le même ordonnancement, ce qui correspond à rajouter la contrainte $y^p_{j'} = y^p_{j''}$ dans le *pricing problem*).

Lorsque deux travaux ne sont pas affectés à la même machine, alors on parle d'ordonnancements différents. Au contraire, s'ils sont sur la même machine, alors ils sont dans le même ordonnancement.

La paire j', j'' est choisie comme suit. Pour chaque paire de travaux j', j'', nous calculons $V_{j',j''} = \sum_{\mathbf{Y^p} \in \mathcal{P}^* | y^p_{j'} = y^p_{j''} = 1} z^*_p$, et la paire j', j'' associée a la plus grande valeur fractionnaire.

TABLE 3.4 – Récapitulatif des différentes notations des modèles proposés

Modèles	Description
PLNE-1	Modèle linéaire en nombres entiers indéxé machine
PLNE-2	Modèle linéaire en nombres entiers indéxé sur le temps
PLNE-3	Modèle linéaire en nombres entiers indéxé machine basé sur les cliques max
PPC	Modèle de programmation par contraintes
PPCPLNE-3	Modèle hybride basique entre la PPC et le PLNE-3
PLNE-4	Modèle linéaire en nombres entiers en fusionnant toutes les machines
PM-1	Problème maître explicite obtenu par la décomposition Danzing-Wolfe
PM-2	Problème maître implicite obtenu par un changement de variable de PM-1
LPM-2	Relaxation linéaire du problème maître-2
RLPM-2	Relaxation linéaire du problème maître-2 restreint
PP	Le sous problème (*pricing problem*)

3.7 Résultats expérimentaux des méthodes exactes

Dans cette section nous analysons les performances des méthodes exactes développées. Pour tester et comparer nos méthodes, nous avons implémenté les modèles mathématiques introduits dans la section 3.4 ainsi que l'approche par PPC de la section 3.5 et le Branch& Price [1] présenté dans la section 3.6 en $C++$.

1. Je remercie chaleureusement M. Boris Detienne (et toute l'équipe IMB "Institut de Mathématiques de Bordeaux" pour l'accueil chaleureux pendant mon séjour) pour l'aide précieuse qu'il m'a apportée à la prise en main d'un environnement spécifique permettent d'implémenter le Branch&Price.

Tous nos tests ont été effectués sur une machine de 2,2 GHz Intel Core i7 et 16 Go de mémoire.

3.7.1 Jeux de données

Quatre types de jeux de données sont définis et utilisés :

Type 1 : Les premiers jeux de données sont les benchmarks générés par Angelelli et al. (2014) un seul type de ressource additionnelle est considéré ($\mathcal{K} = 1$). Le jeu de données disponible en ligne [2] comporte 165 instances définies comme suit :

- Le nombre de machines m est fixé dans l'ensemble $\{4, 7, 10, 15, 20\}$,
- La capacité totale R de la ressource varie dans l'ensemble $\{2, 4, 6\}$,
- Le nombre de travaux est de $n = 100, 110, \ldots, 200$ (par pas de 10),
- Pour chaque triplet (m, R, n), s_j, p_j et r_j sont générés respectivement suivant une loi uniforme dans $[0, 10n - 1], [1, 5n]$ et $[1, R]$. Ainsi, la date de fin d'un travail j est donc $f_j = s_j + p_j$,
- Les poids des travaux sont définis par $w_j = \left\lfloor r_j \times p_j \times (\frac{1}{2} + \mu) \right\rfloor$ où μ est généré suivant une loi uniforme $U[0, 1]$.

Remarque 4. *Selon les instances de **Type 1** la variation des poids des travaux est de l'ordre de 9000%. Or, les priorités des travaux dans un contexte Cloud, sont définies dans l'intervalle* $[1, 11]$ *(Minet et al., 2018). C'est pourquoi nous avons décidé de définir d'autres jeux de données.*

Type 2 : Les instances du deuxième jeu de données sont générées aléatoirement suivant un schéma de la littérature définissant l'ordonnancement des travaux dans les Clouds (Minet et al., 2018), nous avons donc :

- Le nombre de machines m est fixé dans l'ensemble $\{4, 7, 10, 15\}$,
- Les capacités totales disponibles des $R_{k;\ k=1,\ldots,\mathcal{K}}$ ressources sont normalisées à 100,
- Le nombre de travaux est de $n = 20, 60, 100, 150$,
- Pour chaque couple (n, m), 10 instances sont générées pour un total de $4 \times 1 \times 4 \times 10 = 160$ instances,
- Pour chaque quintuplet $(m, R_1, R_2, \ldots, R_k, \ldots, R_{\mathcal{K}}, n)$, s_j, f_j et $r_{j,k}$ sont générés respectivement suivant une loi uniforme dans $[0, 1437]$, $[s_j + 1, 1439]$ (par pas d'une minute sur une journée) et $[1, 100]$.
- Les poids des travaux w_j sont générés suivant une loi uniforme dans $[1, 11]$. En fonction de son poids, un travail appartient à l'une des cinq catégories représentées dans le tableau 3.5.

Type 3 : Sur la base de l'étude présentée dans (Minet et al., 2018), nous générons des instances de grande taille. Pour $\mathcal{K} = 3$, nous avons :

- Le nombre de machines m est fixé dans l'ensemble $\{10, 15, 20\}$,
- Les capacités totales disponibles des $R_{k;\ k=1,\ldots,\mathcal{K}}$ ressources sont normalisées à 100,

2. `http://or-brescia.unibs.it.`

TABLE 3.5 – Pourcentage de travaux par catégorie.

Catégories	Poids w_j	Pourcentages
1	11	30%
2	10	50%
3	9	10%
4	$[3, 8]$	5%
5	$[1, 2]$	5%

— Le nombre de travaux est de $n = 500, 1000, 1500$,
— Pour chaque couple (n, m), 10 instances sont générées pour un total de $3 \times 1 \times 3 \times 10 = 90$ instances,
— Le reste des données est généré sur le même principe que pour le Type 2.

TABLE 3.6 – Récapitulatif des jeux de données

Jeux de données	Description
Type 1	Benchmarks avec un seul type de ressource
Type 2	Instances Cloud, génération aléatoirement avec trois types de ressources
Type 3	Instances de grande traille Cloud, génération aléatoirement avec trois types de ressources

3.7.2 Performances des $PLNE$

Afin d'analyser la performance des $PLNEs$ ($PLNE1$ vs $PLNE2$ vs $PLNE3$), nous avons effectué des expérimentations sur les jeux de données de Type1 et Type2. Pour les trois $PLNEs$, le temps de calcul est fixé à une 1/2 heure (30 minutes). Nous avons utilisé IBM ILOG CPLEX Optimization Studio version 12.8 avec 1 thread.

3.7.2.1 Résultats avec le jeu de données Type1

Le tableau 3.7 et la figure 3.4 résument les performances moyennes des trois $PLNEs$ selon le nombre de machines (la colonne m) et la disponibilité totale de la ressource additionnelle par machine (la colonne R). Les colonnes $CPU(s)$ représentent le temps de calcul moyen en secondes et les colonnes $Gap(\%)$ représentent le gap moyen obtenu entre la meilleure solution réalisable (Z_i^*) et la meilleure borne supérieure trouvée par le Solveur (UB_i) pour chaque modèle $PLNE$.

$$Gap_i(\%) = \frac{UB_i - Z_i^*}{UB_i}$$

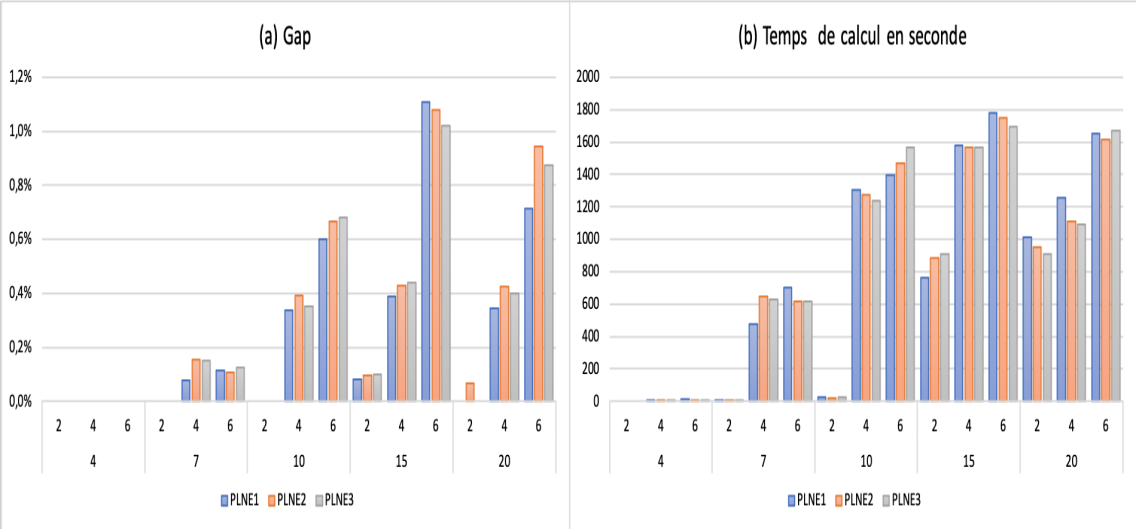
Ainsi, un gap positif indique que tout le temps de calcul autorisé a été consommé pour au moins une instance, tandis qu'un gap nul indique que toutes les instances ont été résolues à l'optimum.

Sur les 165 instances, les trois $PLNEs$ ont quasiment le même taux d'instances résolues à l'optimum, 61% pour les $PLNE1$ et $PLNE3$, 62% pour $PLNE2$. Pour les instances où les

TABLE 3.7 – Résultats ($PLNE1, PLNE2, PLNE3$) sur le jeu de données Type1 avec $m \in \{4, 7, 10, 15, 20\}$ et $R \in \{2, 4, 6\}$

Instances Type 1		PLNE1		PLNE2		PLNE3	
m	R	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
4	2	0	0,0%	0	0,0%	0	0,0%
	4	2	0,0%	2	0,0%	2	0,0%
	6	12	0,0%	8	0,0%	9	0,0%
7	2	2	0,0%	3	0,0%	2	0,0%
	4	474	0,1%	644	0,2%	630	0,1%
	6	701	0,1%	618	0,1%	617	0,1%
10	2	28	0,0%	22	0,0%	25	0,0%
	4	1305	0,3%	1274	0,4%	1237	0,4%
	6	1395	0,6%	1467	0,7%	1565	0,7%
15	2	761	0,1%	884	0,1%	906	0,1%
	4	1578	0,4%	1568	0,4%	1565	0,4%
	6	1779	1,1%	1747	1,1%	1692	1,0%
20	2	1012	0,0%	953	0,1%	909	0,0%
	4	1256	0,3%	1112	0,4%	1092	0,4%
	6	1651	0,7%	1618	0,9%	1670	0,9%
Moy Globale		797	0,3%	795	0,3%	795	0,3%
Taux d'instances résolues à l'optimum		61%		62%		61%	

FIGURE 3.4 – PLNE1 vs PLNE2 vs PLNE3 avec le jeu de donnée Type1



trois $PLNEs$ ne trouvent pas de solutions optimales durant le temps alloué (30 minutes), les $PLNEs$ produisent un faible gap qui ne dépasse pas les 1,1%.

Dans la figure 3.4 (a) et (b), nous observons que pour les trois $PLNEs$ le gap moyen

et le temps de calcul moyen augmentent avec le nombre de machines et la capacité totale d'une seule ressource additionnelle. Le gap moyen global produit par les trois $PLNEs$ est identique 0,3%. Pour le temps de calcul moyen global, les trois $PLNEs$ on quasiment le même temps, 795 secondes pour le $PLNE2$ et $PLNE3$, 797 secondes pour le $PLNE1$.

3.7.2.2 Résultats avec le jeu de données Type2

Le tableau 3.8 et la figure 3.5 résument les performances moyennes des trois $PLNEs$ en faisant varier le nombre de travaux (la colonne n) et le nombre de machines (la colonne m) pour trois types de ressources additionnelles par machine avec des capacités totales normalisées à 100 pour chaque type de ressource.

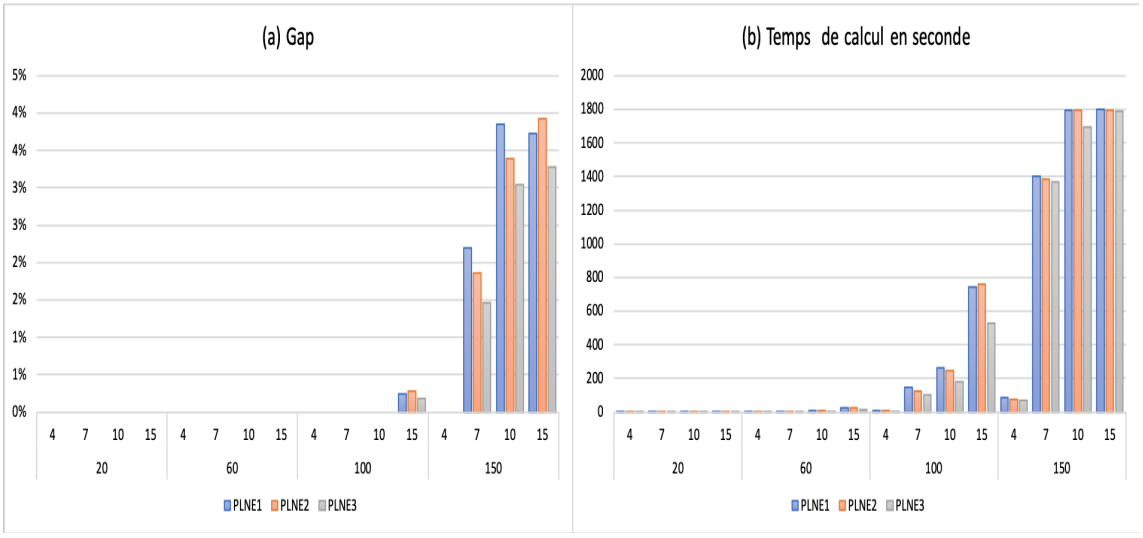
TABLE 3.8 – PLNE1 vs PLNE2 vs PLNE3 en faisant varier le nombre de machines (4-15) avec une capacité total normalisée à 100 pour les trois types de ressources

Instances Type 2		PLNE1		PLNE2		PLNE3	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	0	0%	0	0%	0	0%
	7	0	0%	1	0%	0	0%
	10	0	0%	1	0%	0	0%
	15	0	0%	1	0%	0	0%
60	4	1	0%	2	0%	1	0%
	7	4	0%	4	0%	2	0%
	10	9	0%	9	0%	4	0%
	15	24	0%	28	0%	14	0%
100	4	7	0%	9	0%	5	0%
	7	146	0%	123	0%	102	0%
	10	265	0%	245	0%	180	0%
	15	746	0%	763	0%	529	0%
150	4	84	0%	75	0%	70	0%
	7	1404	2%	1386	2%	1371	1%
	10	1796	4%	1796	3%	1697	3%
	15	1798	4%	1796	4%	1790	3%
Moy Globale		393	0,6%	390	0,6%	360	0,5%
Taux d'instances résolues à l'optimum		79%		78%		79%	

Sur 160 instances, les trois $PLNEs$ ont quasiment le même pourcentage d'instances résolues à l'optimum, 79% pour les $PLNE1$ et $PLNE3$, 78% pour $PLNE2$. Pour les instances où les trois $PLNEs$ ne trouvent pas de solution optimale durant le temps alloué (30 minutes), le $PLNE3$ produit un gap max de 3% légèrement meilleur que les $PLNE1$ et $PLNE2$ qui produisent un gap max de 4%.

Dans la figure 3.5 (a) et (b), nous observons que pour les trois $PLNEs$ le gap moyen et le temps de calcul moyen augmentent avec le nombre de travaux et le nombre de machines. Nous observons aussi que le temps de calcul moyen et le gap moyen de $PLNE3$ sont

FIGURE 3.5 – PLNE1 vs PLNE2 vs PLNE3 avec le jeu de donnée Type2



toujours inférieurs à ceux de $PLNE1$ et $PLNE2$. Le gap moyen global produit par le $PLNE3$ est de 0,5% et 0,6% pour les $PLNE1$ et $PLNE2$. Pour le temps de calcul moyen global, le $PLNE3$ domine les $PLNE1$ et $PLNE2$ avec 360 secondes pour le $PLNE3$, 390 pour $PLNE2$ et 393 secondes pour le $PLNE1$.

3.7.2.3 Conclusion

Le jeu de données Type1 ne nous a pas permis de tirer des conclusions sur les trois $PLNEs$ qui ont quasiment les mêmes performances en termes de temps de calcul et de qualité des solutions, contrairement au jeu de données de Type2 où le $PLNE3$ domine les $PLNE1$ et $PLNE2$.

Le $PLNE3$ présente des limites, notamment pour le jeu de donnée de Type3, i.e le $PLNE3$ n'est pas en mesure de trouver une solution réalisable après plus d'une heure de temps de calcul. Ainsi, nous avons proposé la programmation par contraintes qui nous a permis de trouver des solutions réalisables qui serviront de solutions initiales pour le $PLNE3$.

3.7.3 Performances du PLNE3 vs PPC vs PPCPLNE3

Afin d'analyser la performance des $PLNE3$, PPC et $PPCPLNE3$, nous avons effectué des expérimentations sur les jeux de données de Type2 et Type3, le temps de calcul est limité à 600 secondes (10 minutes). Nous avons utilisé IBM ILOG CPLEX Optimization Studio version 12.8 avec 1 thread pour résoudre le $PLNE3$ et le concept de variables d'intervalles et en utilisant les contraintes globales d'ordonnancement déjà définies dans IBM CP Optimizer (CPO) pour résoudre le modèle PPC.

3.7.3.1 Résultats avec le jeu de données Type2

Le tableau 3.9 et la figure 3.6 résument les performances moyennes des $PLNE3$, PPC et $PPCPLNE3$ en faisant varier le nombre de travaux (la colonne n) et le nombre de machines (la colonne m) pour trois types de ressources additionnelles par machine avec des capacités totales normalisées à 100 pour chaque type de ressource.

TABLE 3.9 – PLNE3 vs PPC vs PPCPLNE3 en faisant varier le nombre de machines (4-15) avec une capacité totale normalisée à 100 pour les trois types de ressources

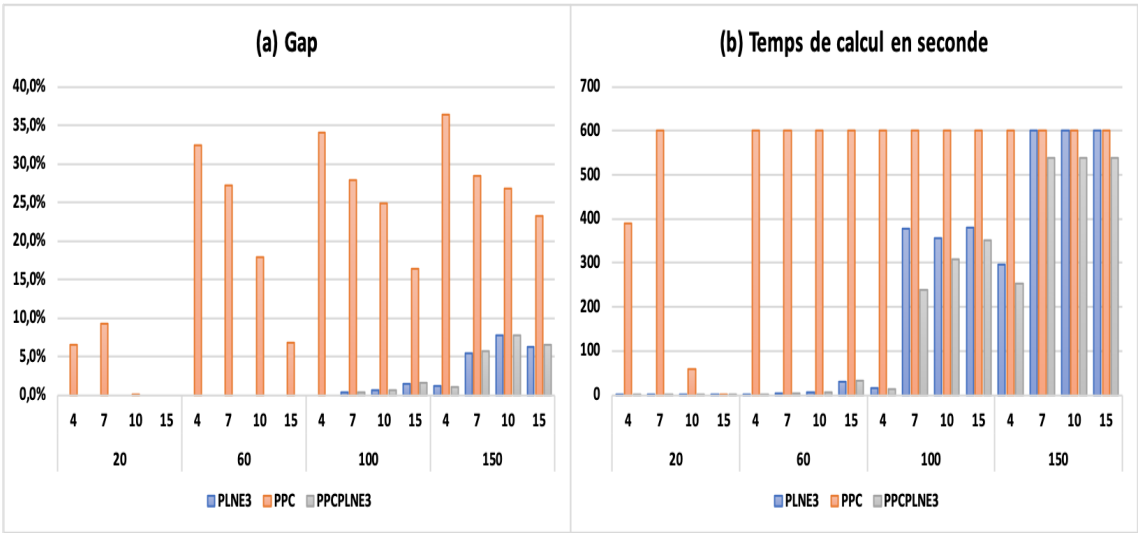
Instances Type 2		PLNE3		PPC		PPCPLNE3	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	0	0,0%	391	6,5%	0	0,0%
	7	0	0,0%	600	9,2%	0	0,0%
	10	0	0,0%	60	0,1%	0	0,0%
	15	0	0,0%	0	0,0%	0	0,0%
60	4	1	0,0%	600	32,5%	1	0,0%
	7	4	0,0%	600	27,3%	5	0,0%
	10	6	0,0%	600	17,9%	6	0,0%
	15	29	0,0%	600	6,7%	32	0,0%
100	4	15	0,0%	600	34,1%	14	0,0%
	7	379	0,4%	600	27,8%	239	0,3%
	10	357	0,6%	600	24,9%	309	0,7%
	15	380	1,4%	600	16,3%	350	1,6%
150	4	297	1,1%	600	36,3%	253	1,0%
	7	600	5,5%	600	28,5%	539	5,7%
	10	600	7,8%	600	26,8%	539	7,8%
	15	600	6,3%	600	23,3%	538	6,5%
Moy Globale		204	1,4%	516	19,9%	177	1,5%
Taux d'instances résolues à l'optimum		71%		15%		73%	

Sur 160 instances, concernant le taux d'instances résolues à l'optimum, le modèle $PPCPLNE3$ a résolu 73% d'instances, dominant ainsi le $PLNE3$ avec 71% et la PPC avec 15%. Pour les instances où les $PPCPLNE3$, $PLNE3$ et PPC ne trouvent pas de solutions optimales durant le temps alloué (10 minutes), $PPCPLNE3$ et $PLNE3$ produisent un gap maximum de 7,8% et PPC un gap maximum de 36,3%.

Dans la figure3.6(a) et (b), nous observons que le temps de calcul moyen et le gap moyen de $PPCPLNE3$ est globalement toujours inférieur à celui de $PLNE3$ et PPC. Le gap moyen global produit par le $PPCPLNE3$ et $PLNE3$ est quasiment identique 1,5% et 1,4%, respectivement et 19,9% pour la PPC. Il est normal que le gap fourni par la PPC soit important car *CpOptimizer* ne donne pas de borne inférieure. Néanmoins, les solutions réalisables trouvées par *CpOptimizer* sont de bonne qualité (les solutions réalisables des instances de 20 et 60 travaux sont optimales).

Pour le temps de calcul moyen global, le $PPCPLNE3$ domine les $PLNE3$ et PPC

FIGURE 3.6 – PLNE3 vs PPC vs PPCPLNE3 avec le jeu de donnée Type2



avec 177 secondes pour le $PPCPLNE3$, 204 pour $PLNE3$ et 516 secondes pour la PPC, donc, le modèle $PPCPLNE3$ converge plus rapidement que le $PLNE3$.

3.7.3.2 Résultats avec le jeu de données Type3

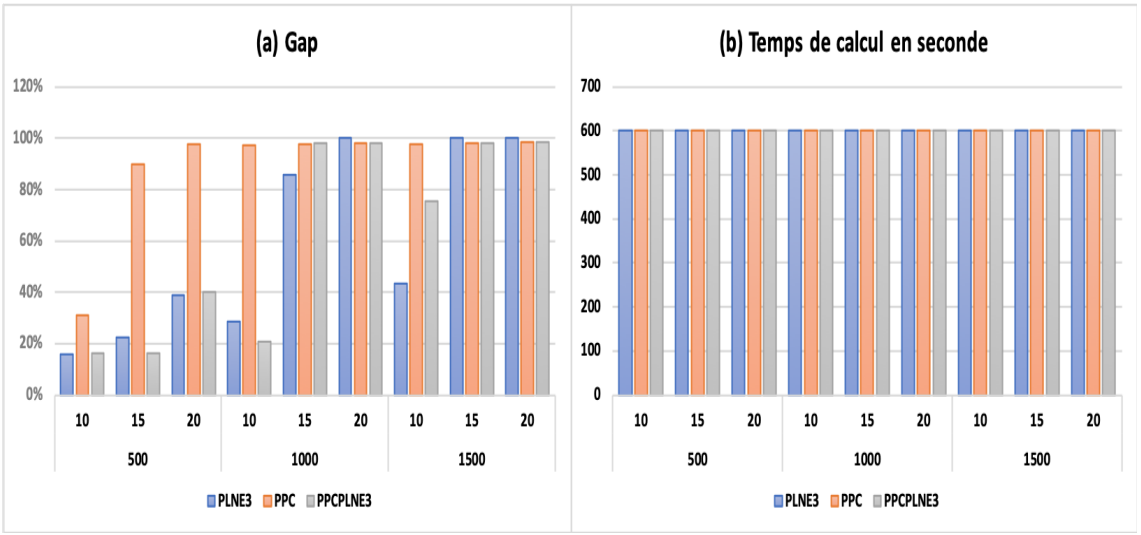
Le tableau 3.10 et la figure 3.7 résument les performances moyennes des $PLNE3$, PPC et $PPCPLNE3$ en faisant varier le nombre de travaux (la colonne n) et le nombre de machines (la colonne m) pour trois types de ressources additionnelles par machine avec des capacités totales normalisées à 100 pour chaque type de ressource.

TABLE 3.10 – PLNE3 vs PPC vs PPCPLNE3 en faisant varier le nombre de machine (10-20) avec une capacité total normalisé à 100 de trois types de ressources

Instances Type 3		PLNE3		PPC		PPCPLNE3	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
500	10	600	16%	600	31%	600	16%
	15	600	23%	600	90%	600	16%
	20	600	39%	600	97%	600	40%
1000	10	600	28%	600	97%	600	21%
	15	600	86%	600	98%	600	98%
	20	600	100%	600	98%	600	98%
1500	10	600	43%	600	98%	600	75%
	15	600	100%	600	98%	600	98%
	20	600	100%	600	98%	600	98%
Taux d'instances résolues à l'optimum		0%		0%		0%	

Sur 90 instances, les $PPCPLNE3$, $PLNE3$ et $PPCPLNE3$ ne trouvent pas de so-

FIGURE 3.7 – PLNE3 vs PPC vs PPCPLNE3 avec le jeu de donnée Type3



lutions optimales durant le temps alloué. En effet, sur l'ensemble des tests effectués, nous avons constaté qu'après 10 minutes de temps de calcul jusqu'à une heure, nous obtenons quasiment les mêmes performances : seulement 1% d'écart entre les solutions trouvées à 10 minutes et 1h de temps de calcul. Nous avons donc fixé le temps de calcul à 10 minutes.

Nous remarquons que là où le $PLNE3$ échoue à trouver des solutions réalisables, par exemple, pour les instances de (n=1000, m=20), (n=1500, m=15) et (n=1500, m=20) où le gap moyen est de 100%, le $PPCPLNE3$ et la PPC arrivent à trouver des solutions réalisables. Le $PPCPLNE3$ trouve des meilleurs gaps moyens que le $PLNE3$ et PPC, sauf, pour les instances (n=1000, m=15) et (n=1500, m=10) le $PLNE3$ trouve des meilleures solutions réalisables que le $PPCPLNE3$ et la PPC.

3.7.3.3 Conclusion

La PPC trouve des bonnes solutions initiales mais a du mal à converger, c'est pour cette raison nous l'avons utilisée pour trouver des solutions initiales pour le $PLNE3$, d'où le modèle $PPCPLNE3$.

Le modèle $PPCPLNE3$ (l'hybridation entre la programmation par contrainte et la programmation linéaire en nombres entiers) a permis d'améliorer les performances du modèle $PLNE3$. Par contre, un tel modèle ($PPCPLNE3$) a ses limites sur les jeux de données de Type3 (instances de grande taille). C'est pour cette raison que nous avons proposé une autre méthode exacte Branch&Price. Dans ce qui suit on reprend les résultats expérimentaux.

3.7.4 Analyse de performance du Branch&Price

Afin d'analyser la performance des Branch&Price ($B\&P$) et $PPCPLNE3$, nous avons effectué des expérimentations sur les jeux de données de Type2 et Type3, le temps de calcul

est limité à 600 secondes (10 minutes). Nous avons utilisé IBM ILOG CPLEX Optimization Studio version 12.8 avec 1 thread pour résoudre le $PLNE3$, le problème maître $PM2$ et le pricing problème PP. Pour résoudre le modèle PPC nous avons utilisé le concept de variables d'intervalle et aussi les contraintes globales d'ordonnancement déjà définies dans IBM CP Optimizer (CPO).

3.7.4.1 Résultats avec le jeu de données Type2

Le tableau 3.11 et la figure 3.8 résument les performances moyennes des $B\&P$ et $PPCPLNE3$ en faisant varier le nombre de travaux (la colonne n) et le nombre de machines (la colonne m) pour trois types de ressources additionnelles par machine avec des capacités totales normalisées à 100 pour chaque type de ressource.

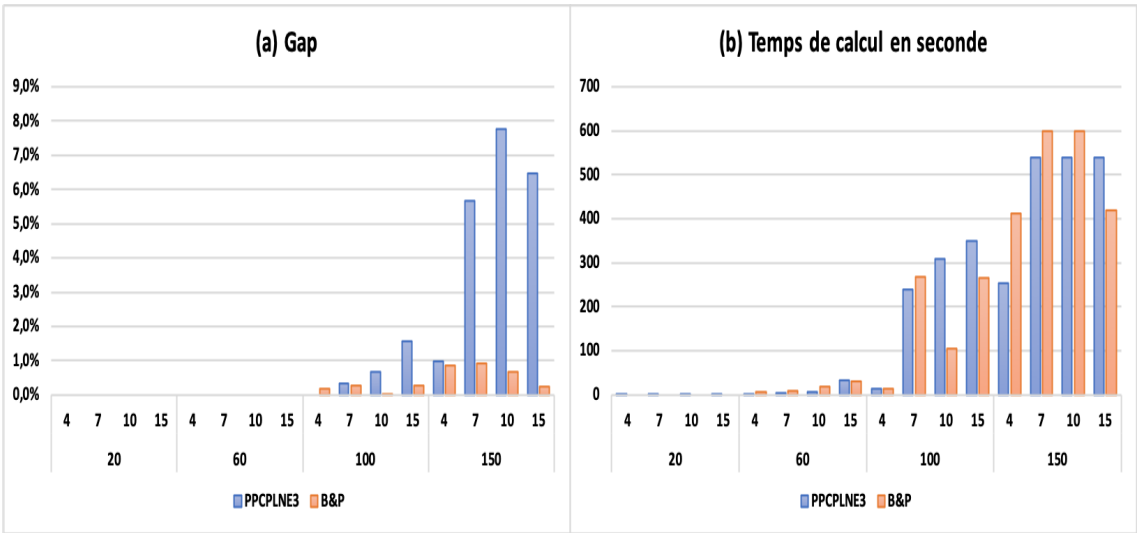
TABLE 3.11 – PPCPLNE3 vs B&P en faisant varier le nombre de machines (4-15) avec une capacité totale normalisée à 100 de trois types de ressources

Instances Type 2		PPCPLNE3		B&P	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	0	0,0%	0	0,0%
	7	0	0,0%	0	0,0%
	10	0	0,0%	0	0,0%
	15	0	0,0%	0	0,0%
60	4	1	0,0%	6	0,0%
	7	5	0,0%	10	0,0%
	10	6	0,0%	18	0,0%
	15	32	0,0%	31	0,0%
100	4	14	0,0%	15	0,2%
	7	239	0,3%	268	0,3%
	10	309	0,7%	105	0,0%
	15	350	1,6%	266	0,3%
150	4	253	1,0%	413	0,9%
	7	539	5,7%	600	0,9%
	10	539	7,8%	600	0,7%
	15	538	6,5%	418	0,2%
Moy Globale		177	1,5%	172	0,2%
Taux d'instances résolues à l'optimum		73%		74%	

Sur 160 instances, le modèle $B\&P$ a résolu 74% d'instances dominant légèrement le $PPCPLNE3$ avec 73%. Pour les instances où les $B\&P$ et $PPCPLNE3$ ne trouvent pas de solutions optimales durant le temps alloué (10 minutes), sur toutes les instances, $B\&P$ produit un gap maximum de 0,9% nettement meilleur que le gap maximum de $PPCPLNE3$ qui est de 7,8%.

Dans la figure3.8(a) et (b), sur toutes les instances, nous observons que le gap moyen de $B\&P$ est toujours inférieur à celui de $PPCPLNE3$. Le gap moyen global produit par

FIGURE 3.8 – PPCPLNE3 vs Branch&Price avec le jeu de donnée Type2



le $B\&P$ est de 0,2% et de 1,5% pour $PPCPLNE3$. Nous observons aussi que le temps de calcul moyen de $B\&P$ est globalement toujours inférieur à celui de $PPCPLNE3$, sauf pour les instances (n=100, m=7) et (n=150, m={4,7,10}). Le $PPCPLNE3$ produit des temps de calcul inférieurs à ceux de $B\&P$. Le temps de calcul moyen global produit par le $B\&P$ est de 172 secondes dominant celui de $PPCPLNE3$ qui est de 177 secondes, donc le modèle $B\&P$ converge plus rapidement que le $PPCPLNE3$.

3.7.4.2 Résultats avec le jeu de données Type3

Le tableau 3.12 et la figure 3.9 résument les performances moyennes des $B\&P$ et $PPCPLNE3$ en faisant varier le nombre de travaux (la colonne n) et le nombre de machines (la colonne m) pour trois types de ressources additionnelles par machine avec des capacités totales normalisées à 100 pour chaque type de ressource.

Sur 90 instances, les $B\&P$ et $PPCPLNE3$ ne trouvent pas de solutions optimales durant le temps alloué (10 minutes), les gaps moyens sont donc tous positifs. Sur les instances (n=1000, m={15,20}) et (n=1500, m={15,20}), le $B\&P$ trouve des gaps moyens inférieurs à celui de $PPCPLNE3$, inversement, sur les instances (n=500, m={10,15,20}), (n=1000, m=10) et (n=1500, m=10) le $PPCPLNE3$ produit des gaps moyens inférieurs à ceux de $B\&P$.

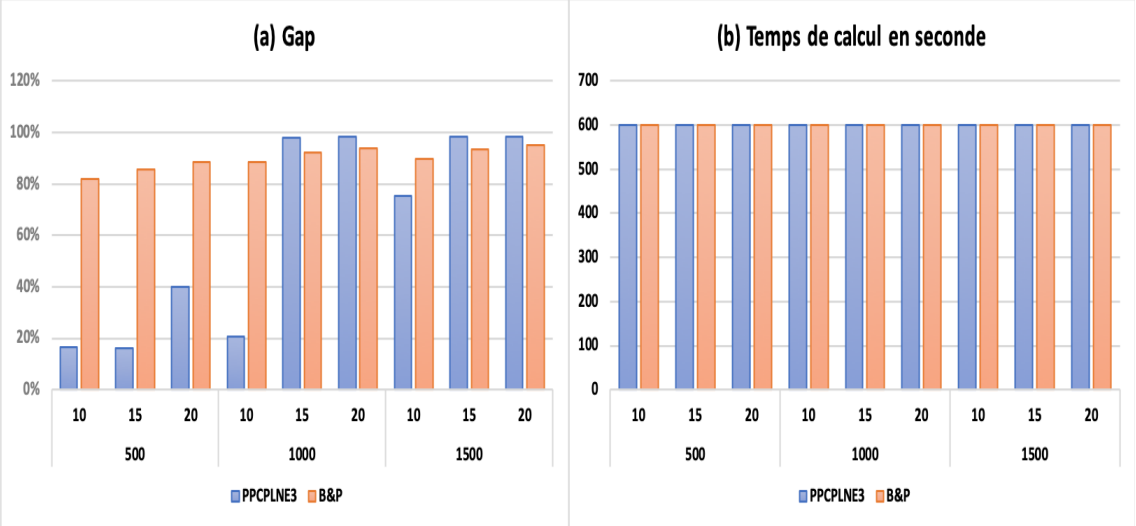
3.7.4.3 Conclusion

Sur le jeu de données Type2, le *Branch&Price* domine nettement le $PPCPLNE3$. En revanche, les deux modèles montrent leurs limites lors du passage à l'échelle avec le jeu de données de Type3. Globalement le *Branch&Price* et le $PPCPLNE3$ sont difficiles à départager sur le temps alloué de dix minutes.

TABLE 3.12 – PPCPLNE3 vs Branch&Price en faisant varier le nombre de machines (10-20) avec une capacité totale normalisée à 100 de trois types de ressources

Instances Type 3		PPCPLNE3		B&P	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)
500	10	600	16%	600	82%
	15	600	16%	600	86%
	20	600	40%	600	88%
1000	10	600	21%	600	88%
	15	600	98%	600	92%
	20	600	98%	600	94%
1500	10	600	75%	600	90%
	15	600	98%	600	93%
	20	600	98%	600	95%
Taux d'instances résolues à l'optimum		0%		0%	

FIGURE 3.9 – PPCPLNE3 vs Branch&Price avec le jeu de donnée Type3



3.8 Conclusions du chapitre

Nous avons considéré une classe de problèmes d'ordonnancement de travaux multiressources à intervalles fixes, qui généralise les problèmes étudiés dans la littérature lorsque le chevauchement vertical n'est pas autorisé, i.e. une machine ne peut exécuter qu'un seul travail à un instant t donné. Nous avons proposé un panel de méthodes exactes pour calculer une solution optimale. Ces méthodes exactes permettent aussi d'évaluer les performances des méthodes approchées proposées dans le chapitre suivant. Lorsque ces méthodes exactes échouent, les gaps sont calculés par rapport aux meilleures bornes supérieures trouvées par ces méthodes exactes. Les meilleures bornes retournées par les méthodes exactes seront donc des valeurs de références pour les méthodes approchées.

Nous avons proposé trois programmes linéaires en nombres entiers. Notons que le $PLNE1$ admet un nombre pseudo-polynomial de contraintes tandis que le $PLNE2$ présente un nombre pseudo-polynomial de variables et de contraintes. De plus, les résultats expérimentaux montrent une légère dominance de $PLNE3$ sur les autres modèles mathématiques. Cependant, le $PLNE3$ échoue sur des instances de grandes tailles, notées $Type3$. Pour améliorer ses performances, nous avons utilisé la programmation par contraintes pour générer une première solution. Il s'agit d'une approche basique d'hybridation $PPCPLNE3$ où la PPC sert d'heuristique pour déterminer rapidement une solution réalisable de départ. Les résultats expérimentaux sur les instances de grande taille montrent l'intérêt de cette hybridation.

Comme la relaxation linéaire du $PLNE3$ ne donne pas de bons résultats, nous avons proposé une décomposition de Dantzig-Wolfe conduisant à une approche de génération de colonnes, ainsi qu'un algorithme de $Branch\&Price$. En général, le $Branch\&Price$ est nettement plus performant que les autres modèles et en particulier le $PPCPLNE3$, même dans le cas où la solution retournée n'a pas été prouvée optimale.

Une des principales perspectives des travaux considérés dans ce chapitre concerne la convergence de l'algorithme $Branch\&Price$. Pour accélérer la convergence, il serait bien entendu intéressant de développer des bornes primales pour s'assurer que le problème maître est toujours réalisable. Nous pouvons résoudre le dual et donc déterminer la nouvelle colonne candidate donnée par la résolution du *pricing problem*. Nous pensons aussi développer une méthode efficace permettant d'identifier de manière appropriée l'ensemble des colonnes que le problème maître doit résoudre et ainsi d'éviter l'infaisabilité des solutions. En effet, le sous-problème PP (*pricing problem*) est bien un problème de sac-à-dos temporel multidimensionnel. Le développement d'un programme dynamique ou même d'une approche hybride faisant appel à la PPC permettant de résoudre le PP est aussi envisageable.

Chapitre 4

Méthodes approchées pour l'ordonnancement monocritère sur machines parallèles multiressources

4.1 Introduction

Nous avons pu analyser dans le chapitre précédent les limites des méthodes exactes proposées pour la résolution du problème d'ordonnancement des travaux à intervalles fixes, nécessitant des ressources non-consommables autres que la machine hôte, pour leur exécution, puisque le problème d'ordonnancement étudié est $\mathcal{NP}$-difficile. Certes, nous avons pu constater que le Branch & Price présente des performances intéressantes pour résoudre des instances de taille raisonnable, mais malheureusement ces méthodes ne peuvent pas passer à l'échelle lorsqu'on s'intéresse au contexte des réseaux informatiques. Le recours à des heuristiques rapides et efficaces est essentiel dans ce cas d'application. Ce chapitre est donc dédié aux méthodes approchées (heuristiques). Notre préoccupation majeure est que les heuristiques proposées doivent présenter le meilleur rapport possible (qualité de la solution)/(temps de calcul). Elles reposent sur des stratégies variées dont le principe est basé sur des arguments intuitifs ou sur des paradigmes de phénomènes naturels. Nous verrons que ce choix est justifié par les résultats qu'elles offrent. Dans la section 4.2, nous proposons des algorithmes gloutons basés sur des règles d'affectation et d'ordonnancement classiques ou basées sur les données du problème à résoudre. Nous introduisons dans la section 4.3 une heuristique constructive, dite PILOT qui combine deux ou plusieurs règles d'ordonnancement et d'affectation pour construire une solution au problème. La section 4.4 est dédiée à un algorithme génétique qui a également pour but d'améliorer les solutions retournées par les heuristiques, puisqu'elles sont utilisées pour générer la population initiale. Dans la section 4.5, nous analysons les performances des méthodes approchées développées dans ce chapitre.

4.2 Heuristiques : algorithmes gloutons

Nous proposons d'utiliser un schéma de génération d'ordonnancement des travaux en série avec des règles de priorité. Soit une instance $\mathcal{I}$ contenant les travaux à ordonnancer triés selon une règle de priorité, nous prenons un par un les travaux dans I et effectuons leur ordonnancement et leur affectation selon une stratégie d'affectation de manière séquentielle. Pour chaque travail $j \in \mathcal{I}$, nous vérifions la possibilité de l'ordonnancer durant son intervalle de temps $[s_j, f_j]$, en tenant compte des contraintes de disponibilité des ressources. Ce travail n'est pas ordonnancé (le travail est rejeté) si son affectation entraîne une solution irréalisable. Cette procédure est répétée jusqu'au dernier travail.

L'algorithme 2 de complexité $O(n \log n)$, décrit l'heuristique de liste proposée.

Algorithm 2 Schéma général des algorithmes de liste

Initialisation :
Soit $\mathcal{I}$ l'ensemble des travaux à ordonnancer triés selon une règle de priorité 4.2.1
$\mathcal{S}^* = \emptyset$. $\mathcal{S}^*$ est la solution (i.e sur chaque machine i, l'ensemble des travaux ordonnancés)
Itération :
Sélectionner un travail j dans la liste $\mathcal{I}$
Selon une stratégie d'affectation 4.2.2
for $i = 1$ **to** m **step** 1 **do**
    Vérifier la possibilité d'ordonnancer le travail j durant sont intervalle de temps $[s_j, f_j]$ sur la machine i, en tenant compte des contraintes de disponibilité des ressources.
    if Contraintes de ressources respectées **then**
        Affecter le travail j sur la machine i
    else
        Rejeter le travail j
    end if
end for
Répéter les étapes jusqu'au dernier travail dans $\mathcal{I}$
Évaluation :
return $\mathcal{S}^* = \{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$ =0

Nous avons étudié plusieurs critères de tri. Dans ce manuscrit nous citons quatre critères qui ont donné des bonnes performances. Nous obtenons donc quatre variantes d'heuristiques de liste, notées **Heur-JS** où les **JS** sont les critères de tri définis dans la section suivante 4.2.1

4.2.1 Règles de priorité

Les travaux sont triés selon une liste de clés dérivées de leurs propres attributs. Si toutes les clés sont égales, l'ordre lexicographique est pris en compte.

Chaque clé peut correspondre soit à un attribut de base (poids w_j, durée opératoire $p_j = f_j - s_j$, quantité de ressource r_{jk}), soit à une expression impliquant deux ou plusieurs

attributs de base.

JS1 : Les travaux sont triés selon l'ordre décroissant de leur poids $\mathbf{w_j}$, en cas d'égalité, selon l'ordre croissant du $\mathbf{max(r_{j,k})}$ des quantités de ressources pour chaque travail, et en cas d'égalité selon l'ordre croissant des $\mathbf{p_j}$. Ce critère permet de favoriser les travaux avec des poids importants.

JS2 : Les travaux sont triés selon l'ordre croissant de $\frac{\mathbf{p_j}}{\mathbf{w_j}}$, en cas d'égalité, selon l'ordre croissant du $\mathbf{max(r_{j,k})}$ des quantités de ressources pour chaque travail, et en cas d'égalité selon l'ordre croissant des $\mathbf{p_j}$. Ce critère permet de favoriser les travaux qui libèrent les ressources le plus rapidement possible (i.e les travaux qui durent moins longtemps sur la machine).

JS3 : Les travaux sont triés selon l'ordre croissant des $\frac{\mathbf{max(r_{j,k})}}{\mathbf{w_j}}$, en cas d'égalité, selon l'ordre croissant du $\mathbf{max(r_{j,k})}$ des quantités de ressources pour chaque travail, et en cas d'égalité selon l'ordre croissant des $\mathbf{p_j}$. Ce critère permet de favoriser les travaux qui occupent moins d'espace sur la machine.

JS4 : Les travaux sont triés selon l'ordre croissant des $\frac{\mathbf{max(r_{j,k})*p_j}}{\mathbf{w_j}}$, en cas d'égalité, selon l'ordre croissant du $\mathbf{max(r_{j,k})}$ des quantités de ressources pour chaque travail, et en cas d'égalité selon l'ordre croissant des $\mathbf{p_j}$. Ce critère permet de favoriser les travaux qui occupent moins d'espace et durent moins longtemps sur la machine.

4.2.2 Stratégies d'affectation

Après avoir ordonné les travaux selon un critère de tri fixe, nous affectons ces travaux aux machines, selon une des stratégies suivantes :

Str1 : chaque travail est affecté à la première machine sur laquelle il peut être ordonnancé de façon réalisable (i.e en respectant les contraintes de ressources). Les travaux qui sont attribués à la première machine sont supprimés de la liste des travaux à ordonnancer et la procédure est répétée sur les autres machines. Cette stratégie est dite **affectation machine par machine**.

Str2 : chaque travail est affecté à la machine la moins chargée. Pour trouver une machine qui est moins chargée, nous calculons le minimum d'utilisation moyenne des ressources sur chaque machine.

$$min_{1\leq i\leq m}(\sum_{j\in\mathcal{S}^{(i)}}\sum_{k=1}^{\mathcal{K}}\frac{r_{jk}}{\mathcal{K}})$$

($\mathcal{S}^{(i)}$ est l'ensemble des travaux déjà ordonnancés sur la machine i).

Les travaux qui sont attribués à la machine moins chargée sont supprimés de la liste des travaux à ordonnancer et la procédure est répétée jusqu'à ce que les contraintes de ressources ne soient pas respectées. Cette stratégie est dite **affectation machine par équilibrage de charge**.

L'analyse expérimentale a montré que les performances en termes de temps de calcul et qualité des solutions de la **Str1** sont légèrement meilleures que la **Str2** ou équivalentes.

Dans le reste de ce manuscrit, l'affectation des travaux choisie est **machine par machine Str1**.

Ainsi, par chaque instance $\mathcal{I}$, nous avons quatre heuristiques de tri plus une cinquième heuristique définie comme la meilleure des quatre premières :

$\mathbf{BestHeur-JS}(\mathcal{I}) = max\ \{\mathbf{Heur-JS1}(\mathcal{I})\ , \mathbf{Heur-JS2}(\mathcal{I})\ , \mathbf{Heur-JS3}(\mathcal{I})\ , \mathbf{Heur-JS4}(\mathcal{I})\}$

4.3 Méthode PILOT pour l'ordonnancement

Il arrive que les heuristiques gloutonnes basées sur des listes de priorités échouent car elles sont basées sur des choix myopes. Pour pallier cet inconvénient, nous proposons le développement d'heuristiques de type PILOT introduites pour la résolution du problème de voyageur de commerce TSP. En effet, La méthode PILOT est une technique itérative d'anticipation préférée, proposée par (Voßs et al., 2005). Elle fait partie des heuristiques dites constructives. Cette méthode tente d'éviter la myopie des méthodes purement gloutonnes en estimant les coûts totaux à chaque étape de la construction de la solution. Dans le cas du problème TSP, à chaque itération permettant de déterminer le prochain arrêt parmi les villes non encore visitées, l'heuristique considère non pas une seule ville mais plutôt un sous-ensemble de villes. Pour chaque prochain arrêt potentiel, nous calculons une estimation du coût total de la tournée complète du TSP, mesurée à partir de la dernière ville visitée (dernière ville choisie à l'itération précédente). La prochaine ville à visiter offrant le meilleur coût total est ainsi sélectionnée. L'estimation du coût total peut-être obtenue par l'application d'une heuristique de construction offrant de bonnes performances, telle que *Plus proche ville d'abord*.

Selon Voßs et al. (2005), PILOT est une méta-heuristique dont l'idée principale est d'effectuer une répétition en utilisant une ou plusieurs heuristiques de construction gloutonne pour anticiper des solutions améliorantes. A chaque itération de PILOT, nous calculons donc provisoirement pour chaque choix (ou mouvement) possible une solution dite "pilote", en enregistrant les meilleurs résultats afin d'obtenir à la fin de l'itération une solution dite "maître" avec le mouvement correspondant. Sur ce principe, la méthode PILOT que nous proposons fait appel aux heuristiques proposées dans la section 4.2, que nous notons **PILOTEHeur-JS**. L'algorithme 3 décrit notre démarche globale de l'élaboration d'une telle méthode avec $H1$ et $H2$ deux heuristiques de liste. Considérons $H1$ l'heuristique PILOT permettant d'estimer le coût total des rejets des travaux.

Sur les quatre règles de priorité définies dans la section 4.2.1, nous avons douze combinaisons de ces règles, ainsi, pour chaque instance de $\mathcal{I}$, nous avons douze algorithmes PILOT plus un treizième défini comme le meilleur des douze premiers :

$\mathbf{BestPILOT}(\mathcal{I}) = max\ \{\mathbf{PILOT-JS1\ JS2}(\mathcal{I})\ , \mathbf{PILOT-JS1\ JS3}(\mathcal{I})\ , \mathbf{PILOT-JS1\ JS4}(\mathcal{I})\ , \mathbf{PILOT-JS2\ JS1}(\mathcal{I}), \mathbf{PILOT-JS2\ JS3}(\mathcal{I}), \mathbf{PILOT-JS2\ JS4}(\mathcal{I}), \mathbf{PILOT-JS3\ JS1}(\mathcal{I}), \mathbf{PILOT-JS3\ JS2}(\mathcal{I}), \mathbf{PILOT-JS3\ JS4}(\mathcal{I}), \mathbf{PILOT-JS4\ JS1}(\mathcal{I}), \mathbf{PILOT-JS4\ JS2}(\mathcal{I}), \mathbf{PILOT-JS4\ JS3}(\mathcal{I})\}$

Algorithm 3 Schéma général d'algorithme PILOT

Initialisation

— Soit $\mathcal{S}_1$ l'ensemble des travaux triés selon une règle de priorité 4.2.1, par exemple le critère **JS4**.

— Soit $\mathcal{S}_2$ l'ensemble des travaux triés selon une autre règle de priorité 4.2.1, par exemple le critère **JS3**.

— Soit **JS4** le critère **Master**

— Soit $\mathcal{S}$-**Master** $= \{\mathcal{S}_1[1]\}$ la solution finale, $\mathcal{S}_1[1]$ est le premier travail de la liste $\mathcal{S}_1$.

Itération

A partir de $\mathcal{S}$-**Master**, nous construisons deux listes temporaires $\mathcal{S}_{\mathbf{temp1}}$ et $\mathcal{S}_{\mathbf{temp2}}$, la figure 4.1 représente la première itération.

— $\mathcal{S}_{\mathbf{temp1}} = \{\mathcal{S} - \mathbf{Master}, \mathcal{S}_1[j]_{2 \leq j \leq n}\}$

— $\mathcal{S}_{\mathbf{temp2}} = \{\mathcal{S} - \mathbf{Master}, \mathcal{S}_2[j^{'}], \mathcal{S}_1[j]_{2 \leq j \leq n}\}\ \forall j^{'} = 1 \ldots n$

— $\mathcal{Z}_{\mathcal{S}_{\mathbf{temp1}}}$ est la valeur de la fonction objectif obtenue par l'heuristique **Heur-JS**$(\mathcal{S}_{\mathbf{temp1}})$

— $\mathcal{Z}_{\mathcal{S}_{\mathbf{temp2}}}$ est la valeur de la fonction objectif obtenue par l'heuristique **Heur-JS**$(\mathcal{S}_{\mathbf{temp2}})$

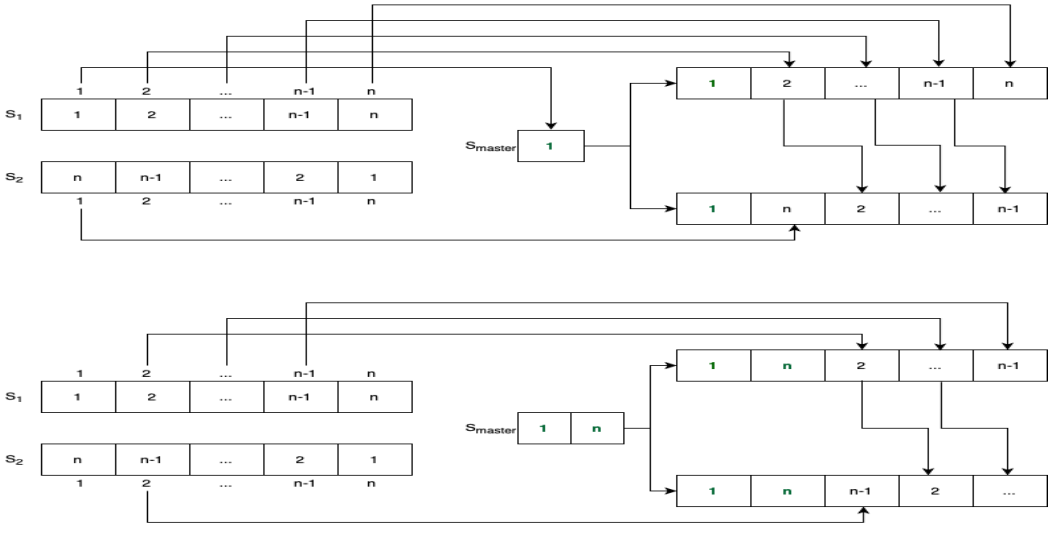
if $\mathcal{Z}_{\mathcal{S}_{\mathbf{temp1}}} \geq \mathcal{Z}_{\mathcal{S}_{\mathbf{temp2}}}$ **then**
  $\mathcal{S}$-**Master** $= \{\mathcal{S}_1[1], \mathcal{S}_1[j]\}$
else
  $\mathcal{S}$-**Master** $= \{\mathcal{S}_1[1], \mathcal{S}_2[j^{'}]\}$
end if

Répéter les étapes jusqu'aux derniers travaux de $\mathcal{S}_1$ et $\mathcal{S}_2$

Évaluation

return $\mathcal{S}$-**Master**

FIGURE 4.1 – Première itération de la méthode PILOT.



4.4 Méthodes évolutionnaires

Les méthodes évolutionnaires font partie de la classe des métaheuristiques et sont aussi appelées méthodes évolutives, méthodes évolutionnistes ou méthodes à base de population. Leur principe est inspiré de la théorie de l'évolution naturelle introduite par Charles Darwin pour expliquer l'adaptation plus ou moins optimale des organismes à leur milieu. Leur origine remonte aux années 50, et depuis plusieurs méthodes évolutionnaires sont proposées dans la littérature, la programmation génétique, les algorithmes génétiques, NSGA ou encore NSGA-II (Jarboui et al., 2013).

Dans cette section 4.4, nous traitons principalement des algorithmes génétiques (voir algorithme 4). Ces types d'algorithmes sont stochastiques, c'est-à-dire qu'ils utilisent des opérateurs pour l'évolution qui conduisent à des résultats différents d'un cycle à l'autre. La population contient un certain nombre d'individus codés, où chacun représente une solution potentielle. La première population (population initiale) est généralement générée de manière aléatoire ou bien par d'autres méthodes comme des heuristiques. Chaque itération de l'algorithme est appelée une génération. Au cours d'une génération, un ensemble de solutions est sélectionné. Ces solutions sélectionnées sont diversifiées en utilisant les opérateurs génétiques, croisement et mutation pour fournir de nouvelles solutions. Les nouvelles solutions remplacent les pires solutions de la population précédente par le processus d'intensification. Pour chaque solution on évalue ensuite leur critère de qualité. La méthode itère de cette façon jusqu'à atteindre le critère d'arrêt, ce dernier peut être défini en fonction du nombre d'itérations, du temps d'exécution ou bien de la qualité des solutions trouvées.

Algorithm 4 Le modèle général d'un algorithme évolutionnaire

Initialisation :
Population (P_0)
t=0
repeat
    $P^{'}(t)$ = Selection($P(t)$)
    $P^{'}(t)$ = Croisement($P^{'}(t)$) ▷ Diversification
    $P^{'}(t)$ = Mutation($P^{'}(t)$)
    $P(t+1)$ = Remplacer($P(t), P^{'}(t)$) ▷ Intensification
    t = t+1
until critère d'arrêt

4.4.1 Choix du codage

Nous avons choisi un codage appelé *codage par permutation*. Un individu est un vecteur de taille n où chaque élément j représente le travail j . Considérons un exemple avec $n = 8$ travaux et $m = 2$ machines. Pour chaque travail j, la date de début, la date de fin, le poids de priorité et les besoins en ressources $r_{j,k}; k = 1, 2, 3$ sont donnés dans le tableau 1.2.

Le décodage d'un individu se fait à chaque itération de l'algorithme. Le décodage se fait suivant l'algorithme 5 en $O(mn|R|)$.

Algorithm 5 Décodage d'un individu

Initialisation :

Soit $\mathcal{I}$ l'ensemble des travaux à ordonnancer d'un individu.

$\mathcal{S}^* = \emptyset$. $\mathcal{S}^*$ est la solution (i.e sur chaque machine i, l'ensemble des travaux ordonnancés)

Itération :

for $i = 1$ **to** m **step** 1 **do**

    Sélectionner un travail j dans la liste $\mathcal{I}$

    Vérifier la possibilité d'ordonnancer le travail j durant sont intervalle de temps $[s_j, f_j]$ sur la machine i, en tenant compte des contraintes de disponibilité des ressources.

    if Contraintes de ressources respectées **then**

        Affecter le travail j sur la machine i

    else

        Rejeter le travail j

    end if

    Répéter les étapes jusqu'au dernier travail dans $\mathcal{I}$

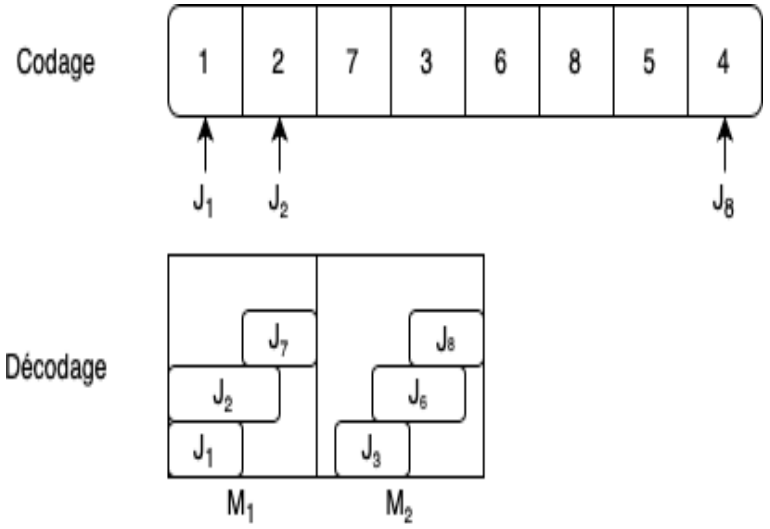
end for

Évaluation :

return $\mathcal{S}^* = \{\mathcal{S}^{(1)}, \ldots, \mathcal{S}^{(m)}\}$

La figure 4.2 montre le codage et le décodage d'un individu.

FIGURE 4.2 – Codage et décodage d'un individu.



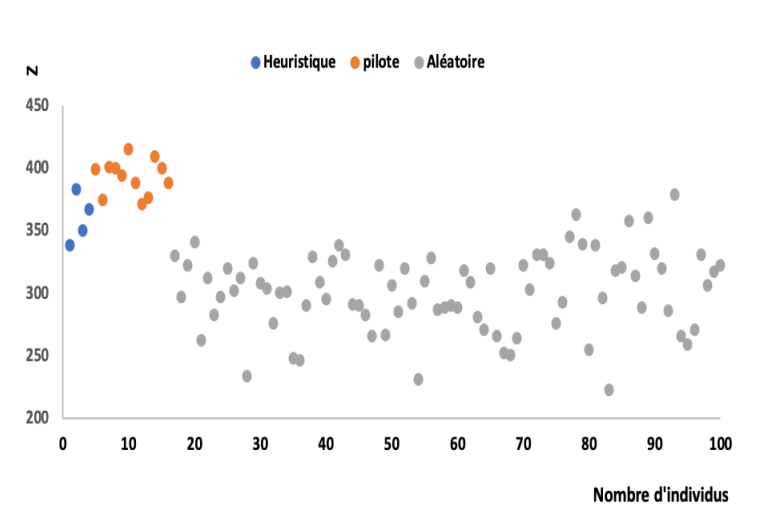
4.4.2 Génération de la population initiale

La population initiale peut-être générée aléatoirement, à base de recherche locale ou encore via l'utilisation d'heuristiques dédiées. Utiliser différentes méthodes pour la génération de la population initiale a pour but de diversifier les solutions initiales afin d'élargir

la recherche dans l'espace des solutions faisables.

Pour résoudre notre problème d'ordonnancement, nous optons pour une génération de la population initiale selon deux approches. Une première partie des individus est générée en utilisant les heuristiques présentées dans les sections 4.2 et 4.3. Selon le choix de la règle 4.2.1 utilisée, les travaux sont ordonnancés dans l'ordre défini par 4.2.1 où l'affectation des travaux aux machines est faite selon la stratégie **Str1** (**affectation machine par machine**) 4.2.2. Une seconde partie de la population est générée suivant un schéma totalement aléatoire. Nous n'avons pas autorisé le clonage des individus. Notons par $N_{\mathcal{P}^0}$ la taille de la population initiale $\mathcal{P}^0$. Nous avons fixé cette taille $N_{\mathcal{P}^0} = 100$ qui a donné le meilleur compromis entre la qualité des solutions et le temps de calcul. Nous avons 4 individus générés par les heuristiques de liste, 12 individus générés par PILOT et 84 individus générés aléatoirement. La figure 4.3 représente un exemple de populations générée à l'aide des techniques discutées précédemment.

FIGURE 4.3 – Population initiale sur une instance de n=100 et m=4.



4.4.2.1 Croisement

Conformément au mécanisme de croisement inter-chromosomique naturel, l'opérateur de croisement permet le brassage génétique des parents. Chaque individu possède deux gènes différents hérités des deux chromosomes des parents. Le patrimoine génétique du nouvel individu est composé aléatoirement d'une partie du patrimoine de chacun de ses deux parents pour permettre au matériel génétique de se répandre au sein de l'espèce.

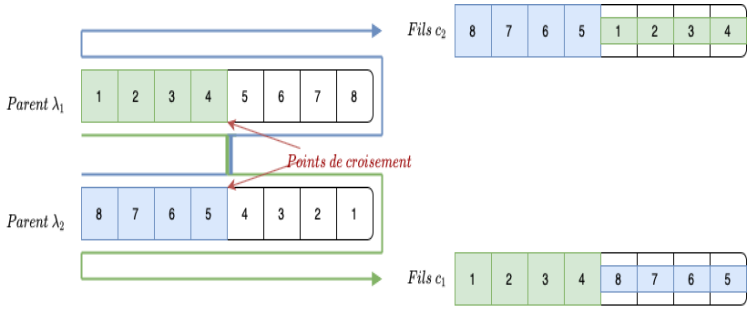
Cet opérateur permet de générer de nouveaux individus potentiellement meilleurs que les individus de la population courante. Le croisement que nous avons choisi opère sur deux individus d'une population et génère deux enfants. Durant la procédure de croisement, deux parents λ_1 et λ_2 sont sélectionnés parmi les individus de la population courante $\mathcal{P}^{q-1}$. Une application de la procédure de croisement à X-points génère deux nouveaux individus fils. Soit C^q l'ensemble des individus générés par l'opérateur de croisement de taille N_{C^q}. Pour

obtenir une solution enfant c_1, soit r ($r \leq n-1$) le nombre de croisements fixés au début de la méthode. c_1 est composé de $r+1$ parties notées $\beta_1//\beta_2//\ldots//\beta_{r+1}$ où β_l est la $l$ième partie ($l = 1, \ldots, r+1$). L'individu c_1 hérite β_1, β_3, β_5, etc. de λ_1 et les autres parties sont héritées du parent λ_2. Le second individu fils est noté c_2, il hérite β_2, β_4, etc. du parent λ_1 et le reste du parent λ_2.

Remarque 5. *Il faut vérifier que les éléments des parties héritées du parent λ_2 (resp. parent λ_1) par l'individu c_1 (resp. l'individu c_2) n'existe pas dans les parties héritées du parent λ_1 (resp. parent λ_2). Si c'est le cas alors il faut chercher à copier les éléments de β_{l-1} etc.*

La figure 4.4 présente un exemple avec un point de croisement.

FIGURE 4.4 – Exemple de croisement avec 8 travaux avec un point de croisement.



4.4.2.2 Mutation

L'opérateur de mutation introduit des modifications aléatoires dans un chromosome en modifiant un élément ou plus. Le but de la mutation est de permettre la diversification de la recherche par l'exploration d'autres points de l'espace des solutions.

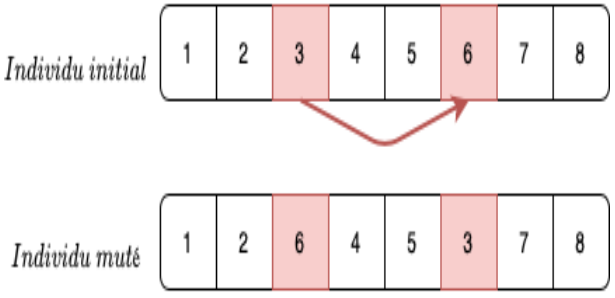
Soit M^q l'ensemble des individus généré par l'opérateur de mutation de taille N_{M^q}.

Une valeur ρ est générée aléatoirement entre 0 et 1. Si $\rho \leq \rho_{mut}$, l'enfant c_1 mute et l'enfant obtenu $c_1^{'}$ est ajouté à la population M^q. Ainsi, un individu c mute si une probabilité choisie au hasard est inférieure ou égale au paramètre ρ_{mut}. Notre opérateur de mutation est défini comme suit : deux gènes (travaux) sont choisis au hasard et leurs caractéristiques sont échangées (voir la figure 4.5).

4.4.2.3 Choix des paramètres

Des paramètres comme la taille de la population, la probabilité de mutation ou bien le nombre de points de croisement sont souvent difficiles à fixer, or le succès de la méthode en dépend fortement. Nous avons appliqué l'algorithme génétique avec différentes combinaisons de paramètres (voir table 4.1). Pour chacun des choix des paramètres et à chaque exécution de la méthode sur le jeu de données Type2, nous récupérons la valeur de la fonction objectif aux itérations $\{5, 10, 15 \ldots, 100\}$. Nous avons comparé les gaps moyens obtenus entre la solution approchée et la meilleur borne supérieure retournée par

FIGURE 4.5 – Exemple d'opérateur de mutation à deux points avec 8 travaux



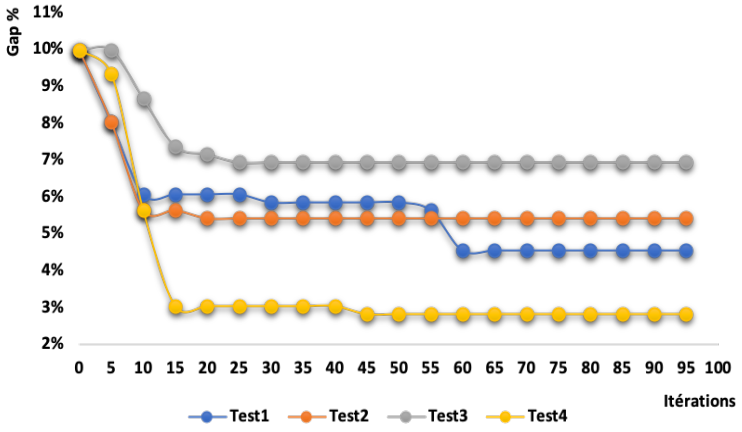
les méthodes exactes. Quatre combinaisons de paramètres sont définies où varient seule-

TABLE 4.1 – Combinaisons de paramètres

Test	$\|\mathcal{P}\|$	$\|\mathcal{P}\|max$	nb pts croisement	$\rho_{croissement}$	$\rho_{mutation}$
Test1	100	200	1	20%	10%
Test2	100	200	1	50%	25%
Test3	100	200	1	70%	35%
Test4	100	200	1	100%	100%

ment la probabilité de croisement et de mutation. Ce choix est dû aux meilleurs résultats expérimentaux obtenus.

FIGURE 4.6 – Choix des paramètres



La figure 4.6 nous montre globalement que le $Test4$ est celui qui produit les plus petits gaps entre la valeur obtenue par l'algorithme génétique et la meilleure borne supérieure trouvée par les méthodes exactes. On observe aussi une même tendance sur les quatre tests, avec une stabilité souvent atteinte aux alentours de 20 itérations.

4.5 Résultats expérimentaux

Dans cette section nous analysons les performances des méthodes approchées développées. Pour tester et comparer nos méthodes, nous avons implémenté en $C++$ les algorithmes gloutons présentés dans la section 4.2, la méthode PILOT de la section 4.3 ainsi que la méthode évolutionnaire de la section 4.4.

Tous nos tests ont été effectués sur une machine de 2,2 GHz Intel Core i7 et 16 Go de mémoire.

4.5.1 Algorithmes gloutons

Afin d'analyser les heuristiques gloutonnes, nous avons effectué des expérimentations sur les jeux de données de Type1, de Type2 et de Type3 (voir la section 3.7.1)

4.5.1.1 Résultats avec le jeu de données Type1

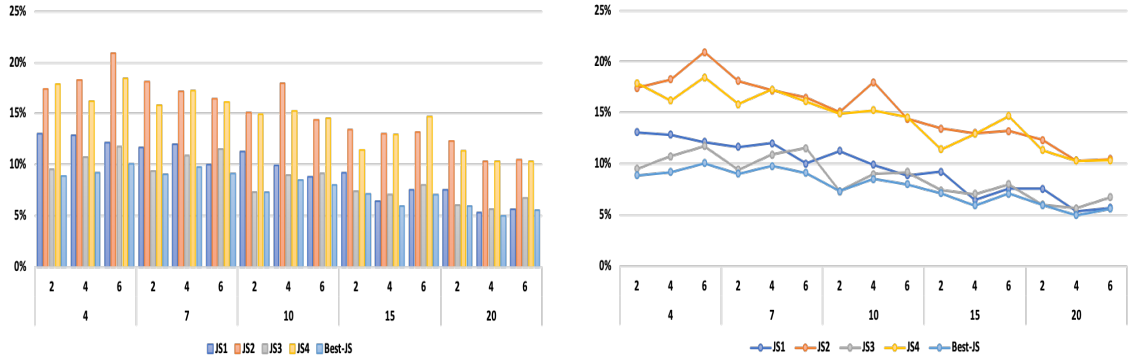
Le tableau 4.2 et la figure 4.7 représentent les gaps moyens obtenus entre les solutions trouvées par les algorithmes gloutons basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes.

TABLE 4.2 – Gap des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type1.

Type 1		JS1		JS2		JS3		JS4		BestHeur-JS	
m	R	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
4	2	0	13%	0	17%	0	10%	0	18%	0	9%
	4	0	13%	0	18%	0	11%	0	16%	0	9%
	6	0	12%	0	21%	0	12%	0	18%	0	10%
7	2	0	12%	0	18%	0	9%	0	16%	0	9%
	4	0	12%	0	17%	0	11%	0	17%	0	10%
	6	0	10%	0	16%	0	12%	0	16%	0	9%
10	2	0	11%	0	15%	0	7%	0	15%	0	7%
	4	0	10%	0	18%	0	9%	0	15%	0	8%
	6	0	9%	0	14%	0	9%	0	15%	0	8%
15	2	0	9%	0	13%	0	7%	0	11%	0	7%
	4	0	6%	0	13%	0	7%	0	13%	0	6%
	6	0	8%	0	13%	0	8%	0	15%	0	7%
20	2	0	8%	0	12%	0	6%	0	11%	0	6%
	4	0	5%	0	10%	0	6%	0	10%	0	5%
	6	0	6%	0	10%	0	7%	0	10%	0	6%
Moy Globale		0	10%	0	15%	0	9%	0	14%	0	8%
Taux d'instances résolues à l'optimum		1%		0%		0%		0%		1%	

Nous observons sur la figure 4.7 que tous les algorithmes produisent des gaps moyens plus faibles lorsque le nombre de machines augmente. Cela s'explique par l'introduction de plus de ressources conduisant à placer et exécuter plus facilement les travaux de poids plus forts. Les algorithmes $JS2$ et $JS4$ sont systématiquement moins bons que les deux autres. De plus, $JS1$ obtient des mauvais gaps avec la capacité totale d'une seule ressource additionnelle (par exemple, $R = 2$), mais il tend à s'améliorer lorsque la capacité totale de la ressource augmente, dans le cas où $R = 6$, $JS1$ est plus performant que $JS3$, sinon $JS3$ domine les autres algorithmes. L'algorithme $BestHeur - JS$ a un net avantage (ou au pire, les mêmes performances) sur les différents algorithmes. Les temps de calcul sont de l'ordre

FIGURE 4.7 – Gap des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type1



de quelques millisecondes pour chaque algorithme. $BestHeur - JS$ prend la somme des 4 temps de calcul et reste dans le même ordre de grandeur.

4.5.1.2 Résultats avec le jeu de données Type2

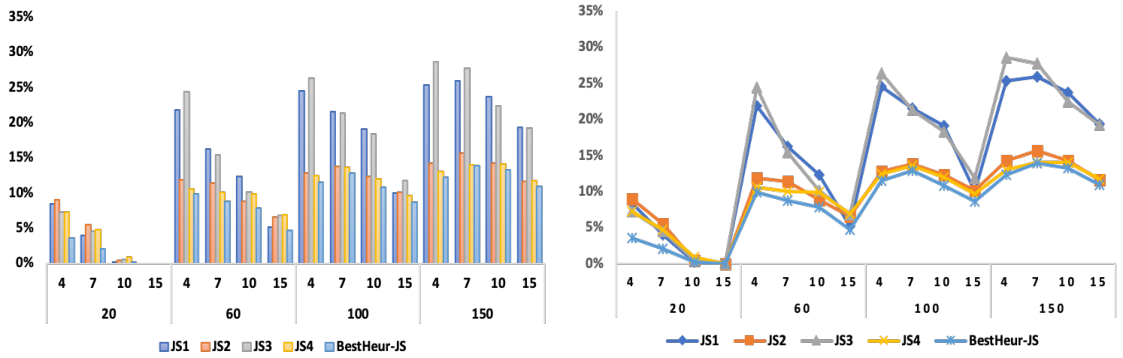
Le tableau 4.3 et la figure 4.8 représentent les gaps moyens obtenus entre les solutions trouvées par les algorithmes gloutons basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes.

TABLE 4.3 – Gap des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type2.

Type 2		JS1		JS2		JS3		JS4		BestHeur-JS	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	0	8%	0	9%	0	7%	0	7%	0	4%
	7	0	4%	0	5%	0	5%	0	5%	0	2%
	10	0	0%	0	0%	0	0%	0	1%	0	0%
	15	0	0%	0	0%	0	0%	0	0%	0	0%
60	4	0	22%	0	12%	0	24%	0	11%	0	10%
	7	0	16%	0	11%	0	15%	0	10%	0	9%
	10	0	12%	0	9%	0	10%	0	10%	0	8%
	15	0	5%	0	7%	0	7%	0	7%	0	5%
100	4	0	25%	0	13%	0	26%	0	12%	0	12%
	7	0	22%	0	14%	0	21%	0	14%	0	13%
	10	0	19%	0	12%	0	18%	0	12%	0	11%
	15	0	10%	0	10%	0	12%	0	10%	0	9%
150	4	0	25%	0	14%	0	29%	0	13%	0	12%
	7	0	26%	0	16%	0	28%	0	14%	0	14%
	10	0	24%	0	14%	0	22%	0	14%	0	13%
	15	0	19%	0	12%	0	19%	0	12%	0	11%
Moy Globale		0	15%	0	10%	0	15%	0	9%	0	8%
Taux d'instances résolues à l'optimum		14%		12%		12%		12%		15%	

Nous observons sur la figure 4.8 que tous les algorithmes produisent des gaps moyens plus faibles lorsque le nombre de machines augmente (i.e. augmentation des ressources) et le nombre de travaux augmente. A titre d'exemple, avec 60 travaux et 15 machines 80% des instances sont résolues à l'optimum. Pour les instances où le nombre de travaux $n = 20$ les quatre algorithmes ont quasiment les mêmes performances. À quelques exceptions

FIGURE 4.8 – Gap des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type2.



près, à savoir les instances ($n = \{60, 100, 150\}, m = 15$) les algorithmes $JS1$ et $JS3$ sont systématiquement moins bons que les deux autres. L'algorithme $JS4$ a un léger avantage par rapport à $JS2$, sauf pour les instances (n=60, m=10) où $JS2$ est plus performant que $JS4$. L'algorithme $BestHeur - JS$ a un net avantage ou bien les mêmes performances sur les algorithmes $JS4$ et $JS2$. Les temps de calcul sont de l'ordre de quelques millisecondes pour chaque algorithme. $BestHeur - JS$ prend la somme des 4 temps de calcul et reste dans le même ordre de grandeur.

4.5.1.3 Résultats avec le jeu de données Type3

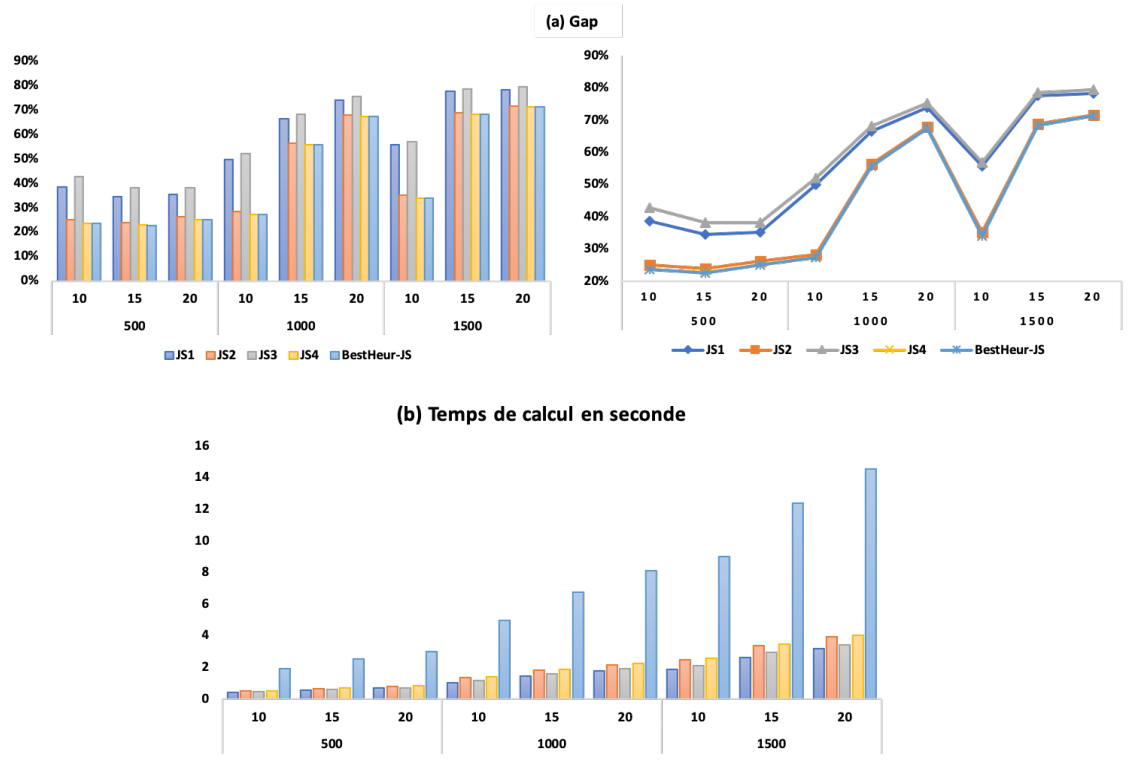
Le tableau 4.4 et la figure 4.9 représentent les gaps moyens obtenus entre les solutions trouvées par les algorithmes gloutons basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes.

TABLE 4.4 – Gap des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type3.

Type 3		JS1		JS2		JS3		JS4		BestHeur-JS	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
500	10	0	39%	1	25%	0	43%	1	24%	2	24%
	15	1	35%	1	24%	1	38%	1	23%	3	23%
	20	1	35%	1	26%	1	38%	1	25%	3	25%
1000	10	1	50%	1	28%	1	52%	1	27%	5	27%
	15	1	66%	2	56%	2	68%	2	56%	7	56%
	20	2	74%	2	68%	2	75%	2	67%	8	67%
1500	10	2	56%	2	35%	2	57%	3	34%	9	34%
	15	3	78%	3	69%	3	78%	3	68%	12	68%
	20	3	78%	4	72%	3	79%	4	71%	15	71%
Moy Globale		2	57%	2	45%	2	59%	2	44%	7	44%
Taux d'instances résolues à l'optimum		0%		0%		0%		0%		0%	

Nous observons sur la figure4.9 que tous les algorithmes produisent des gaps moyens plus importants, et en particulier lorsque les nombres de machines et de travaux augmentent. Les algorithmes $JS1$ et $JS3$ sont systématiquement moins bons que les deux autres. L'algorithme $JS4$ a un léger avantage par rapport à $JS2$, néanmoins nous sommes

FIGURE 4.9 – Performances des heuristiques gloutonnes basées sur des règles de tri sur le jeu de données Type3.



à 44% en moyenne par rapport à la meilleure borne connue, sur l'ensemble des instances, ce qui n'est pas négligeable. L'algorithme $BestHeur - JS$ présente lui aussi les (quasiment) mêmes performances que l'algorithme $JS4$ et $JS2$. On observe aussi que les temps de calcul moyens sont de l'ordre de quelques secondes pour chaque algorithme, que les temps de calcul moyens augmentent lorsque le nombre de machines augmente et le nombre de travaux augmente pour atteindre 15 secondes avec des instances de 1500 travaux et 20 machines.

4.5.1.4 Conclusion

Sur le jeu de données Type1 globalement l'heuristique $JS3$ domine les autres. Il s'agit donc d'un résultat attendu puisque les variations des poids des travaux pour ce jeu de données est de l'ordre de 9000%. Par contre avec les jeux de données Type2 et Type3, l'heuristique $JS4$ domine les autres heuristiques. Néanmoins, la performance de ces heuristiques se détériore lorsque nous considérons les jeux de données Type3. A titre d'exemple, avec $JS4$ le gap moyen global est de l'ordre de 9% pour le Type2 par rapport à la solution optimale et passe à 44% pour le Type3 par rapport aux meilleures bornes connues retournées par les méthodes exactes. $BestHeur - JS$ a un léger avantage sur $JS4$ en terme de qualité de solution.

4.5.2 Algorithmes PILOT

Afin d'analyser les algorithmes PILOT, nous avons effectué des expérimentations sur les jeux de données de Type1, Type2 et Type3 (voir la section 3.7.1)

4.5.2.1 Résultats avec le jeu de données Type1

Le tableau 4.5 et les figures 4.10 (A) et (B) représentent les gaps moyens et les temps de calculs moyens obtenus entre les solutions trouvées par les algorithmes PILOT basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes.

Nous observons sur la figure 4.10 (A) que tous les algorithmes produisent des gaps moyens plus faibles lorsque le nombre de machines augmente et sur la figure4.10(B) les temps de calculs moyens sont de l'ordre 0, 1 ou 2 secondes. Cela s'explique par le fait que l'augmentation de l'espace qui permet de placer plus facilement les travaux les plus rentables. Les algorithmes $PILOT - JS2JS1$, $PILOT - JS2JS3$, $PILOT - JS2JS4$, $PILOT - JS4JS1$, $PILOT - JS4JS2$ et $PILOT - JS4JS3$ sont systématiquement moins bons que les deux autres. Nous observons aussi qu'il n'y a pas de dominance claire entre $PILOT - JS1JS2$, $PILOT - JS1JS3$, $PILOT - JS1JS4$, $PILOT - JS3JS1$, $PILOT - JS3JS2$, $PILOT - JS3JS4$. $BestPILOT$ a un net avantage sur les différents algorithmes. $BestPILOT$ prend la somme des 4 temps, le temps de calcul moyen global est de 12 secondes, le max pour toutes les instances est de 22 secondes. Le gap moyen global est de 5%, le max est de 7%.

4.5.2.2 Résultats avec le jeu de données Type2

Le tableau 4.6 et la figure 4.11 (A) et (B) représentent les gaps moyens et les temps de calculs moyens obtenus entre les solutions trouvées par les algorithmes PILOT basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exacts.

Nous observons sur la figure 4.11 (A) que tous les algorithmes produisent des gaps moyens plus faibles lorsqu'on a plus de machines et plus de travaux mais avec un temps qui augmente raisonnablement comme illustré dans la figure 4.10(B). Pour les instances où le nombre de travaux est égal à 20, les quatre algorithmes ont quasiment les mêmes performances. À quelques exceptions près, à savoir les instances ($n = \{60, 100, 150\}, m = 15$) les algorithmes $PILOT - JS1JS2$, $PILOt - JS1JS3$, $PILOT - JS1JS4$, $PILOT - JS3JS1$, $PILOT - JS3JS2$ et $PILOT - JS3JS4$ sont systématiquement moins bons que les deux autres. Nous observons aussi qu'il n'y a pas de dominance claire entre $PILOT - JS2JS1$, $PILOT - JS2JS3$, $PILOT - JS2JS4$, $PILOT - JS4JS1$, $PILOT - JS4JS2$, $PILOT - JS4JS3$. $BestPILOT$ a un net avantage sur les différents algorithmes. $BestPILOT$ prend la somme des 4 temps, le temps de calcul moyen global est de 74 secondes, le max pour toutes les instances est de 309 secondes. Le gap moyen global est de 5%, le max est de 10%.

TABLE 4.5 – Gap des PILOTs basés sur des règles de tri sur le jeu de données Type1.

Type 1		PILOT-JS1 JS2		PILOT-JS1 JS3		PILOT-JS1 JS4		PILOT-JS2 JS1		PILOT-JS2 JS3		PILOT-JS2 JS4		PILOT-JS3 JS1		PILOT-JS3 JS2		PILOT-JS3 JS4		PILOT-JS4 JS1		PILOT-JS4 JS2		PILOT-JS4 JS3		BestPILOT	
m	R	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
4	2	0	10%	0	7%	0	9%	0	12%	0	11%	0	12%	0	7%	0	7%	0	7%	0	11%	0	13%	0	12%	4	6%
	4	0	10%	0	8%	0	9%	0	13%	0	12%	0	12%	0	7%	0	8%	0	8%	0	10%	0	11%	0	12%	4	5%
	6	0	9%	0	8%	0	9%	0	12%	0	14%	0	15%	0	10%	0	8%	0	9%	0	11%	0	13%	0	13%	5	7%
7	2	1	9%	1	8%	1	9%	1	13%	1	13%	1	13%	1	7%	1	7%	1	7%	1	11%	1	12%	1	11%	9	6%
	4	1	10%	1	8%	1	10%	1	13%	1	12%	1	12%	1	7%	1	9%	1	9%	1	12%	1	12%	1	12%	9	7%
	6	1	8%	1	7%	1	8%	1	12%	1	11%	1	13%	1	8%	1	8%	1	8%	1	11%	1	12%	1	12%	8	5%
10	2	1	9%	1	7%	1	9%	1	11%	1	10%	1	12%	1	5%	1	6%	1	6%	1	10%	1	11%	1	11%	10	5%
	4	1	8%	1	7%	1	8%	1	13%	1	12%	1	12%	1	6%	1	7%	1	8%	1	11%	1	12%	1	12%	11	6%
	6	1	7%	1	6%	1	8%	1	11%	1	10%	1	11%	1	6%	1	6%	1	7%	1	10%	1	11%	1	11%	10	5%
15	2	1	8%	1	6%	1	7%	1	10%	1	10%	1	10%	1	5%	1	5%	1	6%	1	8%	1	8%	1	8%	15	5%
	4	2	5%	1	5%	2	5%	2	10%	2	9%	1	9%	1	5%	2	6%	2	5%	2	9%	1	9%	2	9%	18	4%
	6	1	6%	1	5%	1	6%	1	10%	1	10%	1	10%	1	6%	2	7%	2	7%	2	11%	2	11%	2	10%	18	5%
20	2	2	6%	2	5%	2	6%	2	8%	2	8%	2	10%	2	4%	2	4%	2	4%	2	8%	2	8%	2	8%	22	4%
	4	2	4%	2	4%	2	4%	1	7%	2	7%	1	8%	2	4%	2	5%	2	5%	2	8%	1	7%	2	8%	18	3%
	6	2	4%	2	4%	2	4%	1	7%	2	7%	1	7%	1	5%	2	6%	2	5%	2	7%	1	8%	2	8%	18	3%
Moy Globale		1	8%	1	6%	1	7%	1	11%	1	11%	1	11%	1	6%	1	7%	1	7%	1	10%	1	11%	1	10%	12	5%
Taux d'instances résolues à l'optimum			1%		1%		1%		1%		1%		1%		1%		1%		1%		1%		1%		1%		1%

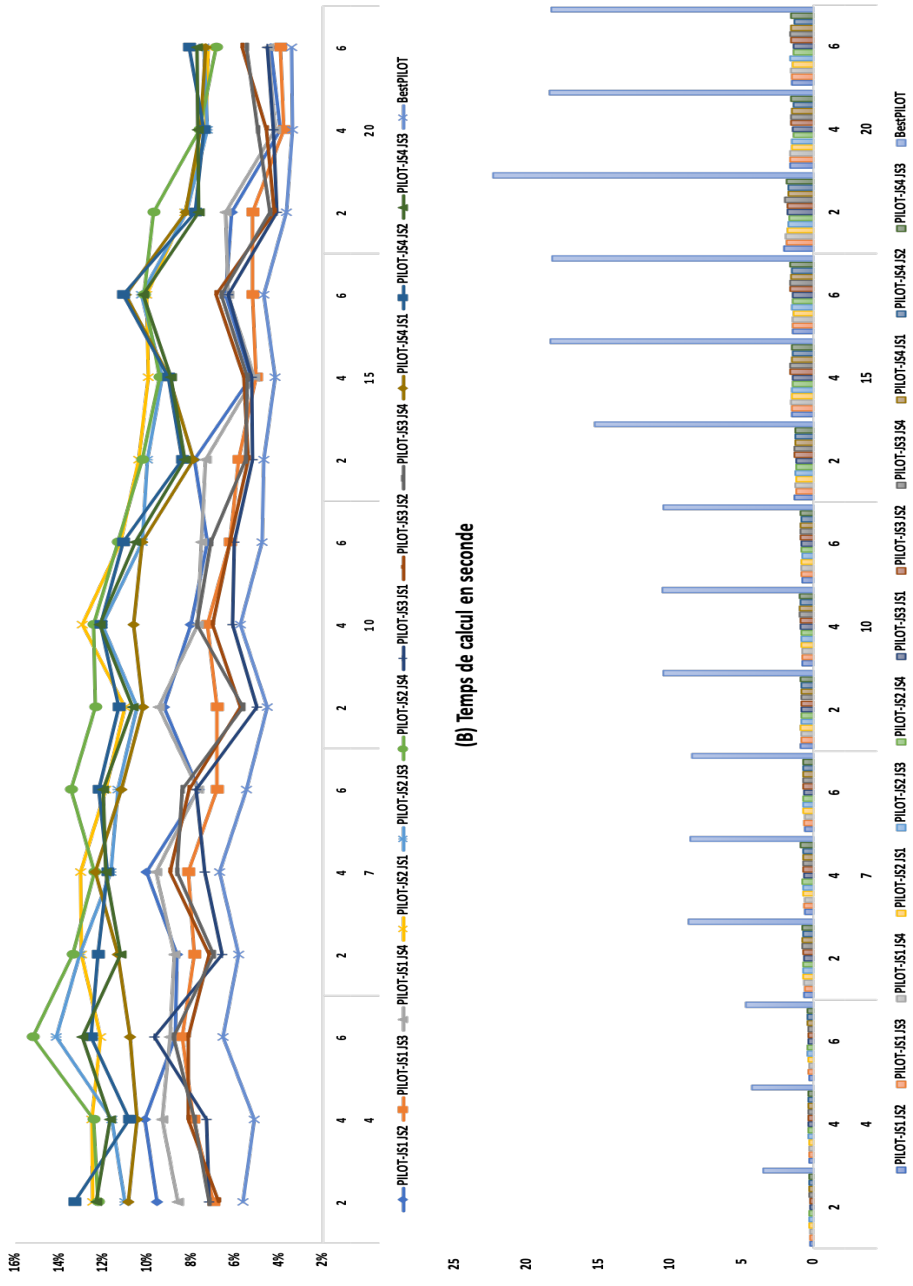


FIGURE 4.10 – Gap des PILOTs basés sur des règles de tri sur le jeu de donné Type1.

TABLE 4.6 – Gap des PILOTs basés sur des règles de tri sur le jeu de données Type2.

Type 2		PILOT-JS1 JS2		PILOT-JS1 JS3		PILOT-JS1 JS4		PILOT-JS2 JS1		PILOT-JS2 JS3		PILOT-JS2 JS4		PILOT-JS3 JS1		PILOT-JS3 JS2		PILOT-JS3 JS4		PILOT-JS4 JS1		PILOT-JS4 JS2		PILOT-JS4 JS3		BestPILOT	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	0	4%	0	1%	0	5%	0	4%	0	5%	0	5%	0	4%	0	4%	0,065	4%	0	3%	0	4%	0	4%	1	1%
	7	0	1%	0	1%	0	2%	0	2%	0	3%	0	3%	0	2%	0	1%	0,097	2%	0	2%	0	4%	0	3%	1	0%
	10	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0,078	0%	0	0%	0	0%	0	0%	1	0%
	15	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0	0%	0,071	0%	0	0%	0	0%	0	0%	1	0%
60	4	1	13%	1	17%	1	14%	1	8%	1	9%	1	8%	1	16%	1	13%	0,921	15%	1	8%	1	8%	1	8%	12	6%
	7	1	11%	1	11%	1	9%	2	7%	2	7%	2	6%	1	12%	2	10%	1,531	10%	2	7%	2	7%	2	8%	19	4%
	10	2	7%	2	8%	2	8%	2	7%	2	6%	2	6%	2	7%	2	7%	1,786	7%	2	6%	2	6%	2	6%	23	4%
	15	2	4%	2	3%	2	3%	3	4%	3	4%	3	4%	2	4%	2	4%	2,342	4%	3	4%	3	3%	3	4%	30	2%
100	4	3	17%	3	18%	3	17%	4	9%	4	10%	3	9%	3	17%	3	16%	3,397	12%	4	8%	4	8%	4	10%	41	7%
	7	5	14%	5	15%	5	13%	6	11%	6	10%	5	10%	5	16%	5	12%	5,319	15%	6	10%	6	9%	6	10%	66	8%
	10	6	13%	6	13%	6	14%	8	10%	8	9%	7	8%	6	14%	6	13%	6,286	13%	8	9%	7	8%	8	9%	83	7%
	15	8	8%	8	8%	8	7%	10	7%	10	6%	9	7%	8	8%	8	9%	8,497	8%	10	6%	9	6%	10	7%	106	5%
150	4	8	18%	7	21%	8	15%	8	12%	9	10%	8	8%	8	22%	8	19%	8,187	18%	9	10%	8	9%	9	9%	99	7%
	7	13	19%	12	21%	13	18%	16	11%	16	12%	15	9%	13	19%	15	18%	13,94	17%	16	11%	16	10%	17	11%	177	9%
	10	16	20%	16	19%	16	19%	19	13%	20	12%	18	11%	17	18%	17	17%	17,8	17%	20	12%	19	11%	21	11%	216	10%
	15	23	15%	22	15%	23	15%	30	10%	30	10%	26	9%	23	14%	24	15%	23,24	14%	30	9%	26	8%	28	10%	309	8%
Moy Globale		6	10%	5	11%	5	10%	7	7%	7	7%	6	6%	6	11%	6	10%	6	10%	7	7%	6	6%	7	7%	74	5%
Taux d'instances résolues à l'optimum		18%		19%		16%		14%		13%		12%		16%		16%		15%		15%		13%		13%		23%	

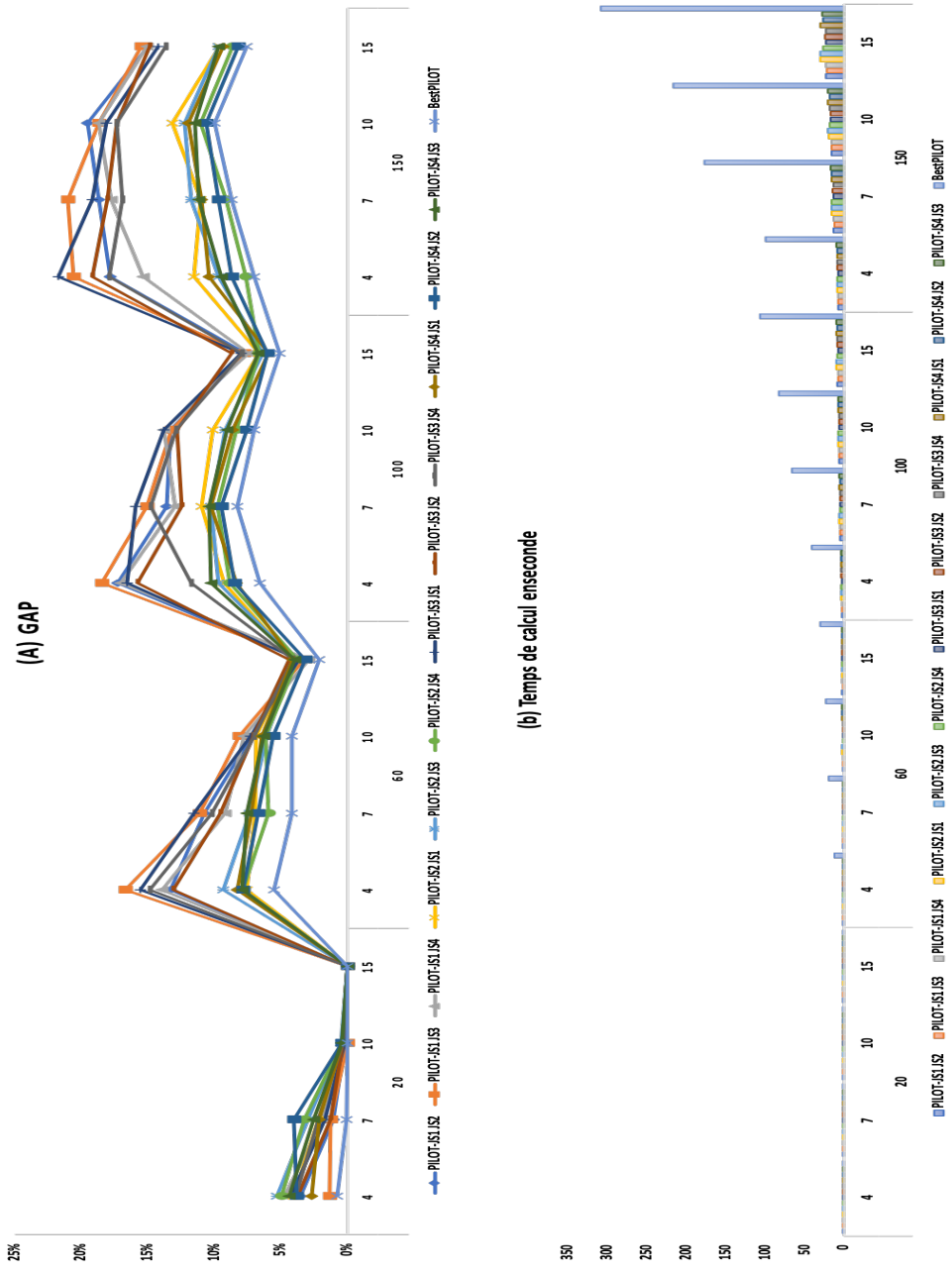


FIGURE 4.11 – Gap des PILOTs basées sur des règles de tri sur le jeu de donné Type2.

4.5.2.3 Résultats avec le jeu de données Type3

Le tableau 4.7 et la figure 4.12 représentent les gaps moyens obtenus entre les solutions trouvées par les algorithmes PILOTs basés sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes.

Les gaps moyens des solutions calculées par les heuristiques par rapport aux bornes connues sont indiqués dans la figure 4.12 (A). La figure 4.10(b) indique les temps de calculs moyens qui bien entendu augmentent lorsque le nombre de machines et de travaux augmentent. Nous rappelons ici que les déviations sont calculées par rapport à la meilleure borne du modèle relâché. Le tableau 3.12 introduit dans le chapitre précédent indique lui aussi des déviations importantes entre la solution faisable (lorsqu'elle est trouvée) et ces bornes. Même si nous ne pouvons conclure avec certitude sur les performances en termes de gap de nos algorithmes pour ce troisième jeu de données à cause de l'absence de références, comme des bornes supérieures efficaces (problème de maximisation). Cependant, nous constatons que les algorithmes $PILOT - JS1JS2$, $PILOT - JS1JS3$, $PILOT - JS1JS4$, $PILOT - JS3JS1$, $PILOT - JS3JS2$ et $PILOT - JS3JS4$ sont systématiquement moins bons que les deux autres. Nous observons aussi qu'il n'y a pas de dominance claire entre $PILOT - JS2JS1$, $PILOT - JS2JS3$, $PILOT - JS2JS4$, $PILOT - JS4JS1$, $PILOT - JS4JS2$, $PILOT - JS4JS3$. $BestPILOT$ a un léger avantage sur les différents algorithmes, notamment sur les instances $(n = 500, m = \{10, 15\})$. A noter tout de même que $BestPILOT$ prend la somme des 4 temps, le temps de calcul moyen global est proche de 6 heures et le max constaté pour l'exécution de toutes les instances est quasiment de 18 heures.

4.5.2.4 Conclusion

Sur les jeux de données Type1 et Type2 globalement, il n'y a pas de dominance claire entre les différents algorithmes PILOTs en terme de qualité de la solution et de temps de calcul, même si nous constatons que $BestPILOT$ a un léger avantage sur les différents algorithmes. Les performances des PILOTs se détériorent avec le jeu de données Type3 où le temps de calcul rend cette heuristique non praticable.

TABLE 4.7 – Gap des PILOTs basés sur des règles de tri sur le jeu de données Type3.

Type 3		PILOT-JS1 JS2		PILOT-JS1 JS3		PILOT-JS1 JS4		PILOT-JS2 JS1		PILOT-JS2 JS3		PILOT-JS2 JS4		PILOT-JS3 JS1		PILOT-JS3 JS2		PILOT-JS3 JS4		PILOT-JS4 JS1		PILOT-JS4 JS2		PILOT-JS4 JS3		BestPILOT	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
500	10	240	31%	213	35%	238	30%	243	23%	254	23%	307	18%	173	36%	283	37%	263	36%	289	22%	227	22%	302	22%	3033	18%
	15	347	31%	366	32%	334	26%	401	21%	440	21%	374	20%	378	32%	383	33%	397	33%	416	21%	401	19%	376	21%	4612	19%
	20	442	31%	481	31%	618	31%	835	23%	695	24%	618	21%	541	30%	680	32%	611	31%	650	20%	527	20%	635	21%	7332	20%
1000	10	895	46%	867	47%	921	45%	1101	28%	1136	28%	1252	26%	1095	46%	990	47%	1444	42%	1412	26%	1074	25%	1459	25%	13646	25%
	15	1485	71%	1859	70%	1362	71%	1746	62%	1658	62%	1599	62%	1662	71%	1775	71%	1752	70%	1721	61%	1675	61%	1757	61%	20652	61%
	20	2170	73%	2284	74%	1907	73%	2546	69%	2399	69%	2412	68%	2305	75%	2008	74%	2702	75%	2747	68%	2430	68%	2866	68%	29437	68%
1500	10	2706	47%	1434	52%	2423	47%	2427	28%	2544	28%	3798	26%	2078	48%	2442	46%	2748	47%	2781	27%	2575	26%	2678	26%	30632	26%
	15	3810	73%	2584	75%	3312	75%	3986	68%	4214	68%	4616	67%	3516	76%	4875	75%	4169	75%	3818	67%	4678	67%	4214	67%	47791	67%
	20	4394	76%	3783	78%	5253	76%	5879	72%	5461	72%	4922	72%	5683	79%	6587	80%	5896	78%	5682	72%	5180	71%	6120	72%	64841	71%
Moy. Globale		1832	53%	1541	55%	1819	53%	2129	44%	2089	44%	2211	42%	1937	55%	2291	55%	2227	54%	2168	43%	2085	42%	2267	43%	24597	42%
Taux d'instances résolues à l'optimum		0%		0%		0%		0%		13%		0%		0%		0%		0%		15%		0%		0%		0%	

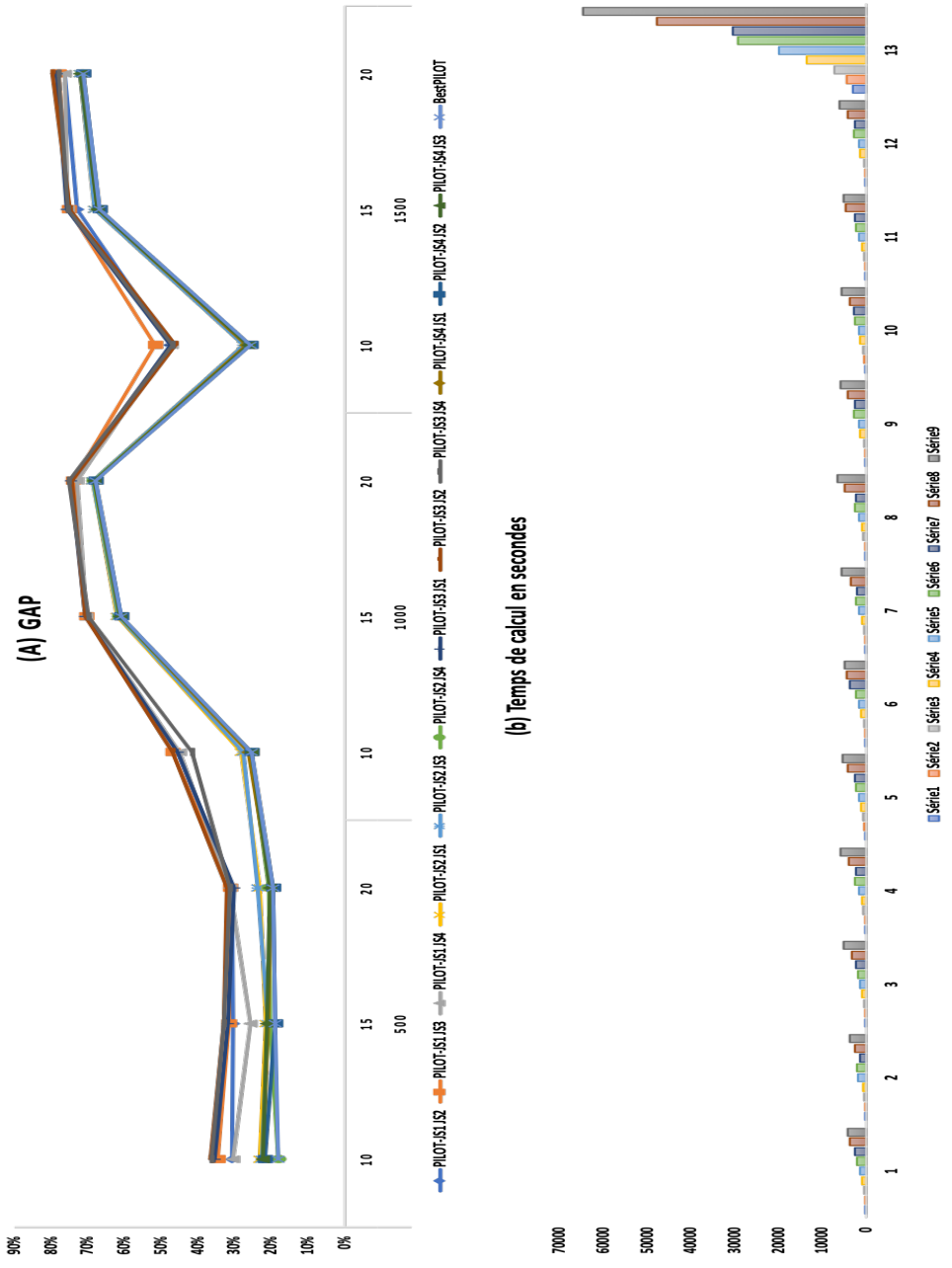


FIGURE 4.12 – Gap des PILOTs basés sur des règles de tri sur le jeu de données Type2.

4.5.3 Algorithmes génétique

Afin d'analyser l'algorithme génétique, nous avons effectué des expérimentations sur les jeux de données de Type1, Type2 et Type3 (voir la section 3.7.1).

4.5.3.1 Résultats avec le jeu de données Type1

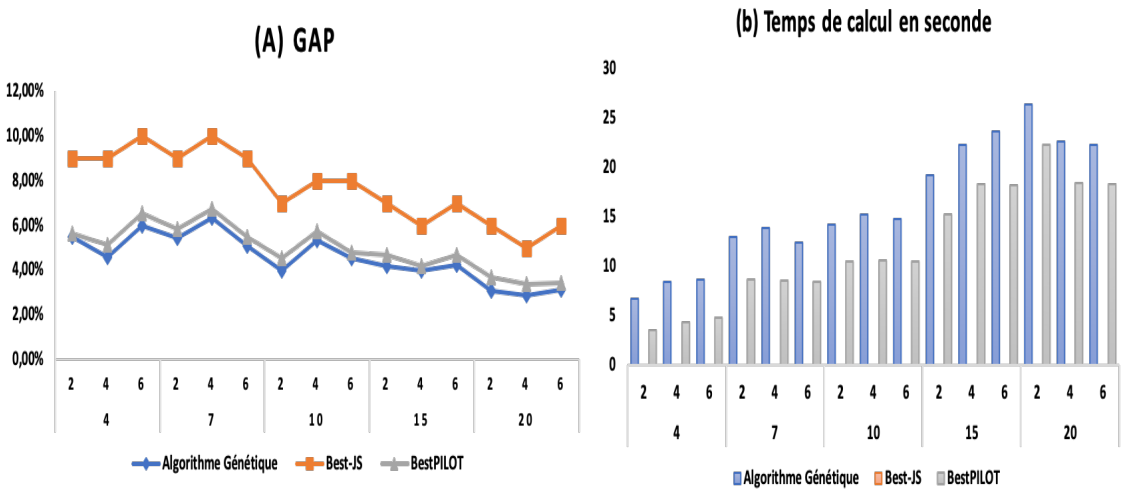
Le tableau 4.8 et les figures 4.13 (A) et (b) représentent les gaps moyens et les temps de calculs moyens obtenus entre les solutions trouvées par l'algorithme génétique basé sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes, les résultats obtenus par l'algorithme génétique sont comparés à ceux trouvés par $Best - JS$ et $BestPILOT$.

TABLE 4.8 – Génétique vs Best-JS vs BestPilot basés sur des règles de tri sur le jeu de données Type1.

Type 1		Algorithme Génétique		Best-JS		BestPILOT	
m	R	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
4	2	7	5,50%	0	9%	4	5,64%
	4	8	4,56%	0	9%	4	5,13%
	6	9	5,97%	0	10%	5	6,56%
7	2	13	5,41%	0	9%	9	5,84%
	4	14	6,34%	0	10%	9	6,72%
	6	12	5,09%	0	9%	8	5,49%
10	2	14	3,98%	0	7%	10	4,52%
	4	15	5,31%	0	8%	11	5,74%
	6	15	4,52%	0	8%	10	4,78%
15	2	19	4,18%	0	7%	15	4,70%
	4	22	3,97%	0	6%	18	4,19%
	6	24	4,25%	0	7%	18	4,70%
20	2	26	3,06%	0	6%	22	3,66%
	4	23	2,90%	0	5%	18	3,38%
	6	22	3,15%	0	6%	18	3,41%
Moy Globale		16	4,55%	0	7,73%	12	4,96%
Taux d'instances résolues à l'optimum		1%		1%		1%	

Nous retrouvons sur la figure 4.13(A) les déviations relatives moyennes de l'algorithme génétique par rapport aux bornes. La figure 4.13(B) reprend les temps de calcul. En terme de qualité de solution, l'algorithme $Best - JS$ est moins performant que les algorithmes $BestPILOT$ et l'algorithme génétique. L'algorithme génétique a un léger avantage sur $BestPILOT$. Le gap moyen global est de 4,55% pour l'algorithme génétique et de 4,96% pour $BestPILOT$. Le gap maximal sur toutes les instances est de 6,34% pour l'algorithme génétique et de 6,72% pour $BestPILOT$. Notons que très peu d'instances ont été résolues à l'optimum par ces méthodes (de l'ordre de 1% pour l'ensemble des instances). Quant au temps de calcul, $BestPILOT$ a un léger avantage avec un temps moyen global de 12

FIGURE 4.13 – Génétique vs Best-JS vs BestPilot basées sur des règles de tri sur le jeu de données Type1.



secondes contre 16 secondes pour l'algorithme génétique.

4.5.3.2 Résultats avec le jeu de données Type2

Le tableau 4.9 et la figure 4.14 (A) et (b) représentent les gaps moyens et les temps de calculs moyens obtenus entre les solutions trouvées par l'algorithme génétique basé sur les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes, les résultats obtenus par l'algorithme génétique sont comparés à ceux trouvés par $Best - JS$ et $BestPILOT$.

La figure 4.11 (A) (resp. 4.10(b)) présente les gaps (resp. temps de calcul) moyens entre les solutions retournées par l'algorithme génétique et les solutions optimales ou à défaut, les meilleures bornes trouvées par le solveur lorsque la solution optimale n'a pas pu être trouvée durant le temps imparti. Pour les instances où le nombre de travaux est égal à 20, à quelques exceptions près, à savoir pour $m = 4$ l'algorithme génétique et $BestPILOT$ ont les mêmes performances : les deux algorithmes trouvent des solutions optimales. L'algorithme $Best - JS$ est moins performant que les algorithmes $BestPILOT$ et l'algorithme génétique en termes de qualité de solution. Le gap moyen global est de 3,51% pour l'algorithme génétique et de 4,79% pour $BestPILOT$. Le gap maximal sur toutes les instances est de 8% pour l'algorithme génétique et de 10% pour $BestPILOT$. $BestPILOT$ a un avantage en termes de temps de calcul avec un temps moyen global de 74 secondes contre 121 secondes pour l'algorithme génétique.

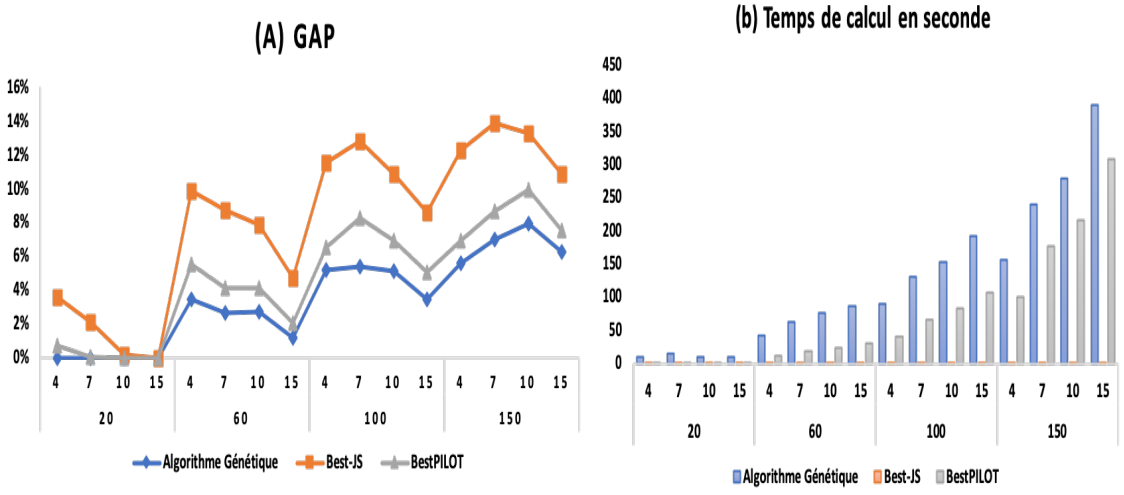
4.5.3.3 Résultats avec le jeu de données Type3

Le tableau 4.10 et la figure 4.15 (A) et (b) représentent les gaps moyens et les temps de calculs moyens obtenus entre les solutions trouvées par l'algorithme génétique basé sur

TABLE 4.9 – Génétique vs Best-JS vs BestPilot basés sur des règles de tri sur le jeu de données Type2.

Type 2		Algorithme Génétique		Best-JS		BestPILOT	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
20	4	10	0%	0	4%	1	1%
	7	14	0%	0	2%	1	0%
	10	10	0%	0	0%	1	0%
	15	10	0%	0	0%	1	0%
60	4	42	3%	0	10%	12	6%
	7	63	3%	0	9%	19	4%
	10	75	3%	0	8%	23	4%
	15	86	1%	0	5%	30	2%
100	4	90	5%	0	12%	41	7%
	7	131	5%	0	13%	66	8%
	10	153	5%	0	11%	83	7%
	15	192	3%	0	9%	106	5%
150	4	157	6%	0	12%	99	7%
	7	239	7%	0	14%	177	9%
	10	279	8%	0	13%	216	10%
	15	390	6%	0	11%	309	8%
Moy Globale		121	3,51%	0	8%	74	4,79%
Taux d'instances résolues à l'optimum		29%		15%		23%	

FIGURE 4.14 – Génétique vs Best-JS vs BestPilot basés sur des règles de tri sur le jeu de données Type2.



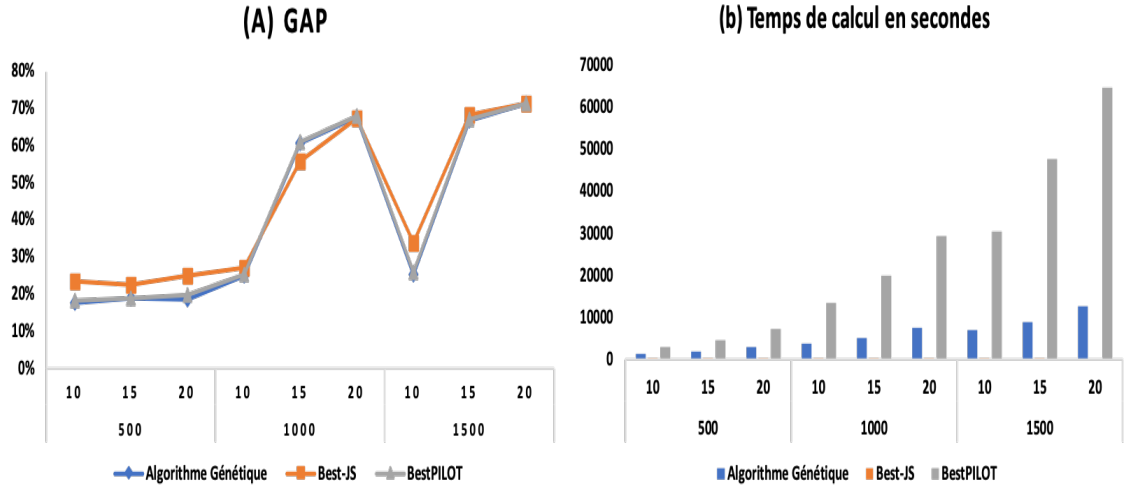
les règles de tri et les meilleures bornes supérieures trouvées par les méthodes exactes, les résultats obtenus par l'algorithme génétique sont comparés à ceux trouvés par $Best - JS$

et $BestPILOT$.

TABLE 4.10 – Génétique vs Best-JS vs BestPilot basés sur des règles de tri sur le jeu de données Type3.

Type 3		Algorithme Génétique		Best-JS		BestPILOT	
n	m	CPU (s)	Gap (%)	CPU (s)	Gap (%)	CPU (s)	Gap (%)
500	10	1505	18%	2	24%	3033	18%
	15	1942	19%	3	23%	4612	19%
	20	2999	19%	3	25%	7332	20%
1000	10	3932	25%	5	27%	13646	25%
	15	5125	61%	7	56%	20052	61%
	20	7716	68%	8	67%	29437	68%
1500	10	7004	25%	9	34%	30632	26%
	15	9084	67%	12	68%	47791	67%
	20	12780	71%	15	71%	64841	71%
Moy Globale		5787	41%	7	44%	24597	42%
Taux d'instances résolues à l'optimum		0%		0%		0%	

FIGURE 4.15 – Génétique vs Best-JS vs BestPilot basés sur des règles de tri sur le jeu de données Type3.



Nous observons sur la figure 4.12 (A) que tous les algorithmes produisent des gaps moyens plus importants lorsque le nombre de machines et le nombre de travaux augmentent. Sur la figure 4.10(b) nous constatons le même effet pour les temps de calculs moyens. A quelques exceptions près, à savoir pour les instances (n=1000, m = {15,20}), l'algorithme $Best-JS$ obtient un léger avantage par rapport à l'algorithme génétique et à $BestPILOT$, qui ont les mêmes performances. Globalement, les trois algorithmes algorithme génétique, $BestPILOT$ et $Best-JS$ ont quasiment les mêmes performances en termes de qualité de

solution. A défaut d'avoir une valeur de référence (la valeur optimale ou une "bonne" borne supérieure), pour ce troisième jeu d'instances, nous remarquons effectivement des déviations relatives importantes (comme pour le gap entre la meilleure solution faisable trouvée par le solveur et sa borne supérieure). Considérant les bornes du solveur comme valeurs de référence, le gap moyen global est de 41% pour l'algorithme génétique et $BestPILOT$, de 44% pour $Best-JS$. Le gap maximal sur toutes les instances est de 71% pour l'algorithme génétique, $BestPILOT$ et $Best-JS$. Concernant la performance en termes de temps de calcul $Best-JS$ domine largement l'algorithme génétique et $BestPILOT$, le temps de calcul moyen global est de 7 secondes pour $Best-JS$, 5787 secondes pour l'algorithme génétique et de 24597 secondes pour $BestPILOT$.

4.5.3.4 Conclusion

Pour les jeux de données Type1 et Type2 et à quelques exceptions près, l'algorithme génétique domine les algorithmes $BestPILOT$ et $Best-JS$ en termes de qualité de solution où le temps de calcul reste raisonnable.

Pour le jeu de données Type3, il est difficile de se prononcer quant à la qualité des solutions trouvées. Cependant, $Best-JS$ offre les meilleurs temps de calcul pour un gap similaire.

4.6 Conclusions du chapitre

Dans ce chapitre, nous nous somme intéressés à la résolution approchée du problème d'ordonnancement par intervalles fixes. Nous avons proposé un panel de méthodes approchées, des algorithmes gloutons basés sur des listes de priorité ou règles de tri, des algorithmes dits PILOT en combinant ces règles de tri deux à deux et un algorithme génétique.

Afin d'analyser les performances des algorithmes gloutons, PILOT et génétique, nous avons repris les jeux de données Type1, Type2 et Type3. Considérant les instances Type1 et Type2 pour lesquelles les méthodes exactes ont pu résoudre à l'optimum certaines instances ou du moins retourner des bornes supérieures correctes, les études expérimentales menées nous ont permis de conclure sur les performances des heuristiques et méthaheuristique proposées dans ce chapitre. Cette étude révèle que l'algorithme génétique est efficace puisqu'il offre des gaps moyens plus faibles par rapport à ceux produits par les algorithmes gloutons ou PILOTs. Les résultats de l'étude expérimentale menée sur les instances Type3 ne sont pas complètement concluants par faute d'avoir de "bonnes" bornes supérieures permettant de juger correctement les performances des heuristiques proposées. Ceci dit, en considérant les bornes supérieures du solveur, nous constatons que l'algorithme génétique offre globalement les mêmes performances que les algorithmes gloutons en termes de qualité de solution, moyennant un temps de calcul un peu plus important.

Une des perspectives de ce chapitre qui retient notre attention concerne le *calcul de bornes supérieures efficaces*, peut-être en exploitant au mieux la structure du problème en utilisant l'ensemble des cliques maximales.

Il serait aussi intéressant de proposer un autre codage de la solution pour l'algorithme

génétique proposé et ainsi appliquer l'opérateur de croisement que nous proposons afin d'améliorer ses performances.

D'autre part, l'étude expérimentale menée montre que la règle d'affectation des travaux aux machines "Affectation machine par machine" **Str1** est efficace et permet d'obtenir une solution optimale à partir d'une permutation des travaux *adéquate*. C'est pourquoi, nous pensons qu'il serait intéressant de développer une autre méthode itérative telle que *Iterated Local Search* ILS.

Chapitre 5

Méthodes exactes pour l'ordonnancement multiagent sur machines parallèles multiressources

5.1 Introduction

Dans les chapitres précédents, nous avons abordé un modèle traditionnel d'ordonnancement où la qualité de l'ordonnancement est évaluée selon une seule mesure définie par une fonction objectif, appliquée sur tous les travaux sans aucune distinction ou presque. La distinction entre les travaux est faite en associant à chaque travail un coefficient de pondération, appelé poids du travail. Ces poids permettent de donner plus ou moins d'importance à un travail. Cette distinction peut ne pas être suffisante (Agnetis et al., 2014).

Lorsqu'un ensemble de critères imposé par un décideur est appliqué sur la totalité des travaux, on parle de problèmes d'ordonnancement multicritère classique. Cependant, il arrive qu'un critère soit pertinent pour un sous-ensemble de travaux et non pas pour la totalité des travaux. Dans les réseaux informatiques par exemple, la minimisation des retards dans le traitement des données multimédias est pertinent mais ne l'est pas pour toutes les applications (Peha, 1995).

Plus précisément, nous nous intéressons à la classe des problèmes d'ordonnancement de travaux en compétition, cette classe est notée (CO). Une définition détaillée est donnée dans la section 2.5. Différents agents associés à des sous-ensembles de travaux disjoints sont considérés. Chaque agent souhaite maximiser le gain total de l'ordonnancement de ses travaux (ou équivalent, cherche à minimiser le coût de rejet total de ses propres travaux).

L'objectif de ce chapitre est donc de généraliser l'étude précédente au cas du problème d'ordonnancement multiagent. Nous présentons d'abord quelques définitions accompagnées des notations utilisées dans la suite du document. Nous analysons la complexité des problèmes étudiés, et des cas particuliers polynomiaux sont identifiés. Pour la résolution exacte du problème d'ordonnancement multiagent, nous proposons la généralisation des méthodes exactes développées pour l'ordonnancement monocritère au cas multiagent. Nous développons dans la section 5.5.1 un programme linéaire en nombres entiers $PLNE$. Un modèle

basé sur la programmation par contraintes est introduit dans la section 5.5.2 permettant de générer une solution initiale réalisable pour le $PLNE$. Nous proposons par la suite une méthode par décomposition de type Branch & Prince, décrite dans la section 5.5.3. A la fin ce chapitre, nous présentons une analyse expérimentale des performances des méthodes exactes développées.

5.2 Problèmes étudiés

Nous nous focalisons sur l'étude de l'ordonnancement des travaux fixes, indépendants et multiressources. Les travaux sont répartis en O sous-ensembles disjoints. Le scénario COMPETITION, noté CO est considéré (Agnetis et al., 2014). La fonction objectif de l'agent o est définie par : $z^o = \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{N}^o} w_j x_{ij}$ avec $o = A, B$.

5.3 Analyse de complexité

Dans cette section, nous analysons la complexité du problème d'ordonnancement multiagent sur machines parallèles multiressources. Nous nous concentrons sur les trois approches multicritères suivantes : l'approche ε-contrainte, la combinaison linéaire des critères et l'énumération du front de Pareto.

5.3.1 Combinaison linéaire des critères : $F_\ell(z^A, z^B)$

Lorsque les fonctions objectifs de tous les agents sont de même nature et de type $min - sum$ ou $max - sum$, les problèmes d'ordonnancement multiagent sont équivalents aux cas monocritère si l'approche utilisée est la combinaison linéaire des critères. La preuve est donnée par la réécriture de la fonction objectif. C'est le cas de notre étude, puisque l'objectif pour chaque sous-ensemble de travaux (agent) est de même nature, c'est-à-dire maximiser le poids total des travaux ordonnancés (équivalent à minimiser le coût total pondéré des travaux rejetés).

Proposition 2. *Le problème d'ordonnancement $Pm|CO, s_j, f_j, r_j|F_\ell(z^A, z^B)$ est $\mathcal{NP}$-difficile.*

Preuve. *Soit à ordonnancer les travaux de deux agents A et B. Lorsque c'est l'approche combinaison linéaire des critères qui est considérée, le problème est équivalent au cas monocritère $\mathcal{NP}$-difficile $Pm|s_j, f_j, r_j| \sum_{i \in \mathcal{M}} \sum_{j \in \mathcal{N}} w'_j x_{ij}$ avec :*

- $w'_j = \alpha w_j$ *si* $j \in \mathcal{N}^A$
- $w'_j = (1 - \alpha) w_j$ *si* $j \in \mathcal{N}^B$, *avec* $\alpha \in [0, 1]$

5.3.2 Approche ε-contrainte : $\varepsilon(z^B/z^A)$

Lorsqu'une fonction objectif est à minimiser et que la deuxième fonction objectif est bornée (dans le cas de deux critères), on parle de l'approche ε-contrainte. Nous écrivons : $\min z^B$, s.t. $z^A \leq Q_A$. Cette méthode est souvent utilisée dans la littérature et conduit à une solution de Pareto faible. Pour obtenir une solution optimale au sens de Pareto, le problème symétrique doit être résolu.

Proposition 3. *Le problème d'ordonnancement* $Pm|CO, s_j, f_j, r_j|\varepsilon(z^B/z^A)$ *est* $\mathcal{NP}$*-difficile.*

Preuve. *Soit à ordonnancer les travaux de deux agents* A *et* B*. Si l'objectif est de minimiser le coût total de rejet des travaux de* $\mathcal{N}^B$ *sous contrainte que le coût total de rejet des travaux de* $\mathcal{N}^A$ *est borné par* Q_A*, le problème est équivalent au cas monocritère* $Pm|s_j, f_j, r_j|\sum_{i\in\mathcal{M}}\sum_{j\in\mathcal{N}} w'_j x_{ij}$ *en fixant* Q_A *à une valeur suffisamment grande.*

5.3.2.1 Cliques maximales disjointes

Dans cette section, nous allons considérer un cas particulier pour l'ordonnancement des travaux des deux agents avec des durées opératoires et des poids quelconques sur m machines. Contrairement au cas général, une machine ne peut exécuter qu'un seul travail à la fois. Nous supposons que l'ensemble des cliques maximales forme H cliques disjointes, c'est-à-dire $L_h \cap L_{h+1} = \emptyset, h = 1, \ldots, H-1$. Selon la notation des problèmes d'ordonnancement, nous introduisons dans le champ β le terme : $L_h \cap L_{h+1} = \emptyset$. Nous pouvons ainsi noter le problème étudié dans cette section par : $Pm|CO, s_j, f_j, L_h \cap L_{h+1} = \emptyset|\varepsilon(z^A, z^B)$, $h = 1, \ldots, H-1$.

Sans perte de généralité, le nombre de travaux de l'agent A (resp. B) est égal au nombre de machines m (sinon compléter l'ensemble des travaux des agents par des travaux fictifs de poids nul). Il est clair qu'au maximum m travaux parmi les $n_k, k = A, B$ pourront être ordonnancés. Pour minimiser les coûts de rejet total des travaux de chaque agent, seuls les m premiers travaux de poids les plus importants peuvent être considérés.

Proposition 4. *Le problème* $Pm|CO, s_j, f_j, L_h \cap L_{h+1} = \emptyset|\varepsilon(z^A, z^B)$ *est* $\mathcal{NP}$*-difficile, même si* $m = 1$*.*

Preuve. *Considérons le cas d'une seule machine. Le problème de minimisation des coûts de rejet* $1|CO, s_j, f_j, L_h \cap L_{h+1} = \emptyset, \sum_{j\in\mathcal{N}^A} w_j x_j \leq Q_A|\sum_{j\in\mathcal{N}^B} w_j x_j$ *est* $\mathcal{NP}$*-difficile.*

Soit PW^AW^B *le problème de décision associé à* $1|CO, s_j, f_j, L_h \cap L_{h+1} = \emptyset|\varepsilon(\sum_{j\in\mathcal{N}^B} w_j x_j \,/\, \sum_{j\in\mathcal{N}^A} w_j x_j)$*. Ce problème est défini par :*

PW^AW^B

Données *: Un ensemble* $\mathcal{N}^A$ *de* n *travaux, un ensemble* $\mathcal{N}^B$ *de* n *travaux, date de début* s_j *et date de fin* f_j *et un poids* w_j *pour chaque travail* j*,* $1 \leq j \leq 2n$*, deux valeurs entières* Q_A *et* Y^B*.* $\forall j = 1, \ldots, n$*, le* $(n+j)$*ème travail de* $\mathcal{N}^A$ *a les mêmes caractéristiques que le* j*ème travail de* $\mathcal{N}^B$*, c'est-à-dire les mêmes intervalles de temps, durées opératoires et poids. Chaque clique maximale est définie par :* $L_h = \{j, n+j\},\ h = 1, \ldots, n$

Question *: Existe-t-il un ordonnancement* σ *pour* $\mathcal{N}^A$ *tel que*

$$\sum_{j\in\mathcal{N}^A} w_j x_j \leq Q_A \text{ et } \sum_{j\in\mathcal{N}^B} w_j x_j \leq Y^B \text{ ?}$$

Nous montrons que le problème de PARTITION se réduit à $1|CO, s_j, f_j, L_h \cap L_{h+1} = \emptyset|\varepsilon(\sum_{j\in\mathcal{N}^B} w_j x_j \,/\, \sum_{j\in\mathcal{N}^A} w_j x_j)$ *(*$PARTITION \propto PW^AW^B$*). Rappelons que le problème de* $PWES$ *(Garey and Johnson, 1979) est défini comme suit :*

Données *: Un ensemble* $\mathcal{I}$ *de* n *éléments avec des valeurs entières notées* $a_1, a_2, \ldots, a_n$*,* $\sum_{i=1}^{n} a_i = 2\mathcal{B}$*.*

Question *: Existe-t-il un sous-ensemble* $\mathcal{I}_1$ *des indices tels que*

$$\sum_{i\in\mathcal{I}_1} a_i = \sum_{i\in\mathcal{I}\setminus\mathcal{I}_1} a_i = \mathcal{B} \ ?$$

À partir de l'instance de $PARTITION$, on définit une instance du problème PW^AW^B comme suit :

- $\mathcal{N} = \{1, 2, \ldots, 2n\}$; $\mathcal{N}^A = \{1, 2, \ldots, n\}$; $\mathcal{N}^B = \{n+1, \ldots, 2n\}$,
- *Pour les travaux de $\mathcal{N}^A$, nous avons : $w_j = a_j$, $s_j = j$ et $f_j = j+1$, $\forall j \in \{1, \ldots, n\}$. Ainsi : $\sum_{j\in\mathcal{N}^A} a_i = 2\mathcal{B}$.*
- *Pour les travaux de $\mathcal{N}^B$, nous avons : $w_j = a_j$, $s_j = j - n$ et $f_j = j + 1 - n$, $\forall j \in \{n+1, \ldots, 2n\}$. Ainsi : $\sum_{j\in\mathcal{N}^B} a_i = 2\mathcal{B}$.*
- *$Q_A = \mathcal{B}$ et $Y^B = \mathcal{B}$.*

Comme il s'agit de l'ordonnancement des travaux des deux agents sur une seule machine qui ne peut traiter qu'un seul travail à la fois, selon la construction de l'instance, si un travail j de l'agent A est ordonnancé (resp. rejeté), systématiquement le travail $n + j$ de l'agent B est rejeté (resp. ordonnancé). Par construction de l'instance, soit l'ensemble $\mathcal{N}_1$ les indices des travaux de l'agent A ordonnancés. Ainsi, l'ensemble $\mathcal{N}_1$ détermine l'ensemble des travaux de B rejetés, i.e. si $j \in \mathcal{N}_1 \Rightarrow (n + j)$ est rejeté.

Soit la variable de décision binaire x_j, nous avons : $x_j = 1$ si j est rejeté ; 0 sinon.

Pour $\mathcal{N}_1$ donné, nous pouvons ainsi déduire la relation des coûts de rejet entre les travaux des deux agents comme suit :

$$\sum_{(n+j)\in\mathcal{N}^B : j\in\mathcal{N}_1} w_j x_j = \sum_{j\in\mathcal{N}^A : j\in\mathcal{N}^A\setminus\mathcal{N}_1} w_j x_j \tag{5.1}$$

Montrons maintenons que $PARTITION \propto PW^AW^B$.

($\Rightarrow$) Supposons que la réponse au problème $PARTITION$ est 'oui' et soit $\mathcal{I}_1 \subset \mathcal{I}$ la solution. On définit une solution pour PW^AW^B comme suit :

- *Solution de l'agent A : le travail j tel que $j \in \mathcal{I}_1$ est rejeté (ou équivalent, les indices des travaux dans $\mathcal{I}\setminus\mathcal{I}_1$ sont ordonnancés). Ainsi nous avons $\sum_{j\in I_1} w_j x_j = Q_A = \mathcal{B}$.*
- *Solution de l'agent B : le travail $(n+j)$ tel que $j \in \mathcal{I}\setminus\mathcal{I}_1$ est rejeté (ou équivalent, les indices des travaux dans $\mathcal{I}_1$ sont ordonnancés). Ainsi nous avons $\sum_{j\in\mathcal{I}\setminus\mathcal{I}_1} w_{n+j}x_{n+j} = Y^B = \mathcal{B}$.*

Les coûts de rejet total de chaque agent vérifient donc l'équation suivante :

$$\sum_{j\in\mathcal{N}^A : j\in\mathcal{I}_1} w_j x_j = \sum_{(n+j)\in\mathcal{N}^B : j\in\mathcal{I}\setminus\mathcal{I}_1} w_j x_j = \mathcal{B}$$

Cet ordonnancement est une solution réalisable et la réponse au problème de PW^AW^B est donc 'oui'.

($\Leftarrow$) S'il existe une solution réalisable pour le problème PW^AW^B, alors soit $\mathcal{N}_1$ l'ensemble des indices des travaux de A rejetés.

$$\sum_{j\in\mathcal{N}^A} w_j x_j = \Big(\sum_{j\in\mathcal{N}_1} w_j x_j + \sum_{j\in\mathcal{N}^A\setminus\mathcal{N}_1} w_j x_j\Big) \leq Q_A = \mathcal{B} \tag{5.2}$$

Nous avons $\sum_{j\in\mathcal{N}^A\setminus\mathcal{N}_1} w_j x_j = 0$*, car il s'agit de l'ensemble des travaux de l'agent A ordonnancés, donc* $x_j = 0$*.*

$$(5.2) \Rightarrow \sum_{j\in\mathcal{N}_1} w_j x_j \leq Q_A = \mathcal{B}$$

Supposons :

$$\sum_{j\in\mathcal{N}_1} w_j x_j < Q_A \Rightarrow \sum_{j\in\mathcal{N}^A\setminus\mathcal{N}_1} w_j(1-x_j) > Q_A \tag{5.3}$$

Puisque la machine exécute un seul travail à la fois, si le travail j *de* $\mathcal{N}^A$ *est ordonnancé, alors le travail* $(n+j)$ *est rejeté. Dans ce cas, par définition* j *appartient à l'ensemble* $\mathcal{N}^A \setminus \mathcal{N}_1$*.*

$$(5.3) \Rightarrow \sum_{j\in\mathcal{N}^B} w_j x_j = \sum_{j\in\mathcal{N}^A\setminus\mathcal{N}_1} w_{n+j} x_{n+j} > Q_A = \mathcal{B} = Y^{\mathcal{B}}$$

Contradiction avec la solution réalisable.

Nous déduisons que $\sum_{j\in\mathcal{N}_1} a_j = \mathcal{B}$ *et* $\sum_{j\in\mathcal{N}^A\setminus\mathcal{N}_1} a_j = \mathcal{B}$ *et donc la réponse à PARTITION est 'oui'.* □

Nous pouvons ainsi déduire le corollaire suivant dans le cas où les cliques maximales sont quelconques, i.e. non forcément disjointes.

Corollaire 1. *Le problème* $Pm|CO, s_j, f_j|\varepsilon(z^A, z^B)$ *est* $\mathcal{NP}$*-difficile, même si* $m = 1$*.*

5.3.3 Enumération de l'ensemble des optima de Pareto : $\mathcal{P}(z^A, z^B)$

Lorsqu'il s'agit de l'énumération des optima de Pareto stricts, le problème est noté $\alpha|\beta|\mathcal{P}(z^A, z^B)$.

Notons que plusieurs algorithmes ont été proposés dans la littérature pour améliorer la mise en oeuvre d'un processus pour déterminer le front de Pareto par l'approche ε-contrainte (T'kindt et al., 2007; Mavrotas, 2009; Agnetis et al., 2014). Pour déterminer une solution non-dominée, nous résolvons le problème symétrique, c'est-à-dire nous minimisons z^A sous la contrainte $z^B \leq Q_B^*$.

Contrairement à l'approche combinaison linéaire des critères, en modifiant le vecteur Q itérativement, tout le front de Pareto est accessible. Néanmoins, elle présente les inconvénients suivants :

- Il s'agit d'un problème d'optimisation sous contrainte,
- Il faut choisir les Q_k. Ainsi, nous avons autant de calculs que de valeurs possibles pour les Q_k.

Remarque 6. *Lorsque le nombre de solutions de Pareto stricts* $\mathcal{P}$ *est polynomial, la détermination d'une solution optimale par combinaison linéaire peut être obtenue en temps polynomial. Il suffit de chercher* x *parmi les solutions de l'ensemble* $\mathcal{P}$ *qui minimise la fonction :* $\alpha z^A(x) + (1-\alpha)z^B(x)$*. Certes, cette procédure peut s'avérer coûteuse en temps et donc dépend fortement de la complexité de l'algorithme d'élaboration de l'ensemble* $\mathcal{P}$*.*

Considérons le cas de deux agents. Pour l'ordonnancement des travaux fixes multiressources avec poids unitaires, nous avons la proposition suivante.

Proposition 5. *Le problème d'ordonnancement des travaux fixes avec poids unitaires, noté* $Pm|CO, s_j, f_j, r_j|\mathcal{P}(z^A, z^B)$ *admet un nombre de solutions de Pareto borné par* $O(nb)$ *avec* $nb = \min(n_A, n_B)$.

Preuve. *Il est évident que pour le cas de deux agents et avec des travaux de poids unitaires, le problème admet au plus* $\min(n_A, n_B)$ *solutions : allant de 0 travail rejeté pour un agent, au cas où tous ses travaux sont rejetés.*

Considérons maintenant le cas où les poids des travaux sont quelconques et avec deux agents.

Proposition 6. *Le problème d'ordonnancement des travaux fixes avec poids quelconque, noté* $Pm|CO, s_j, f_j, r_j|\mathcal{P}(z^A, z^B)$ *admet un nombre de solutions de Pareto pseudo-polynomial borné par* $O(W)$ *avec* $W = \min(\sum_{j\in\mathcal{N}^A} w_j, \sum_{j\in\mathcal{N}^B} w_j)$.

Preuve. *Il est évident que pour le cas de deux agents avec des travaux de poids quelconques, le problème admet au plus* $\min(\sum_{j\in\mathcal{N}^A} w_j, \sum_{j\in\mathcal{N}^B} w_j)$ *solutions : allant de 0 coût de rejet des travaux d'un agent à un coût total de rejet correspondant au rejet de la totalité de ses travaux.*

5.4 Cas polynomiaux

Dans cette section, nous étudions une variante du problème étudié : une machine ne peut exécuter qu'un seul travail à la fois. Nous parlons de *Chevauchement vertical interdit*. Pour illustrer notre démarche, seulement deux agents sont considérés. Néanmoins, nous indiquons les résultats qui se généralisent au cas de O agents.

Rappelons que pour le cas d'un seul agent (cas monocritère), l'ordonnancement des travaux à intervalles fixes sans contrainte de ressources additionnelles, noté $Pm|s_j, f_j|\sum_i \sum_j w_j x_{ij}$, est polynomial (Arkin and Silverberg (1987)). Cependant, nous avons montré que dans le cas d'un problème d'ordonnancement multiagent $Pm|CO, s_j, f_j|\varepsilon(z^B/z^A)$ le problème devient $\mathcal{NP}$-difficile (cf Corollaire 1).

Les résultats obtenus dans cette section sont résumés dans le Tableau 5.1.

Le problème $Pm|CO, s_j, f_j|\varepsilon(z^B/z^A)$ est $\mathcal{NP}$-difficile. Lorsque la combinaison linéaire des critères est considérée, le problème est polynomial (premier résultat de la Table 5.1). Les trois derniers problèmes présentés dans cette table sont des cas particuliers du $Pm|CO, s_j, f_j|\varepsilon(z^B/z^A)$.

5.4.1 Combinaison linéaire des critères

Soit le problème d'ordonnancement $Pm|CO, s_j, f_j|F_\ell(z^A, z^B)$ avec deux agents. Si l'approche combinaison linéaire des critères est considérée, nous avons la proposition suivante.

Proposition 7. *Le problème d'ordonnancement* $Pm|CO, s_j, f_j|F_\ell(z^A, z^B)$ *est Polynomial. Une solution optimale au sens de Pareto peut-être calculée en* $O(n^2 logn)$.

TABLE 5.1 – Résultats cas polynomiaux.

Problème	Complexité	Section
$Pm\|CO, s_j, f_j\|F_\ell(z^A, z^B)$	$O(n^2 logn)$	Sec. 5.4.1
$Pm\|CO, s_j, f_j, \mathcal{N}_1^A \subseteq \mathcal{N}^A\|\varepsilon(z^B/z^A)$	$O(n^2 logn)$	Sec. 5.4.2.1
$Pm\|CO, s_j, f_j, s_i \leq s_j \Rightarrow f_i \leq f_j\|\varepsilon(z^A, z^B)$	$O(mnn_An_B)$	Sec. 5.4.2.2
$Pm\|CO, s_j, f_j, s_i \leq s_j \Rightarrow f_i \leq f_j\|\mathcal{P}(z^A, z^B)$	$O(mnn_An_B)$	Sec. 5.4.2.2

Preuve. *La preuve est identique à celle de la Proposition 5.3.1. Le problème est ainsi équivalent au cas monocritère $Pm|s_j, f_j|\sum_i \sum_j w_j x_{ij}$ (Arkin and Silverberg (1987)).*

Remarque 7. *Pour le cas de O agents, le problème demeure polynomial et une solution de Pareto optimale peut-être calculée en $O(n^2 logn)$.*

5.4.2 Approche ε-contrainte

Le problème $Pm|CO, s_j, f_j|\varepsilon(z^A/z^B)$ avec ressources disjonctives reste ouvert. Dans cette section, nous montrons que quelques cas particuliers sont polynomiaux.

5.4.2.1 Ordonnancement d'un sous-ensemble fixé de travaux

Le problème $Pm|CO, s_j, f_j|\varepsilon(z^A/z^B)$ reste ouvert. Dans cette section, nous analysons le cas où un agent spécifie l'ensemble des travaux à ordonnancer. C'est-à-dire, nous cherchons à minimiser les coûts de rejet des travaux de l'agent B connaissant le sous-ensemble des travaux de A qui seront ordonnancés, noté $\mathcal{N}_1^A \subseteq \mathcal{N}^A$. Pour la notation du problème, nous introduisons dans le champ β le terme $\mathcal{N}_1^A \subseteq \mathcal{N}^A$.

Proposition 8. *Le problème $Pm|CO, s_j, f_j, \mathcal{N}_1^A \subseteq \mathcal{N}^A|\varepsilon(z^B/z^A)$ est Polynomial. Une solution optimale au sens de Pareto si elle existe, peut-être calculée en $O(n_An_B logn)$.*

Preuve. *: $\mathcal{N}_1^A \subseteq \mathcal{N}^A$ est le sous-ensemble des travaux de l'agent A à ordonnancer sur m machines. Sans perte de généralité, s'il existe une solution pour laquelle aucun travail de $\mathcal{N}_1^A$ n'est rejeté, alors on peut le vérifier en temps polynomial.*

Pour résoudre le problème, nous adaptons la démarche proposée par Arkin and Silverberg (1987) pour résoudre le cas monocritère $Pm|s_j, f_j|\sum_i \sum_j w_j x_j$, par la recherche d'un flot à coût minimum.

D'abord, nous modifions les poids des travaux de $\mathcal{N}_1^A$ comme suit :
— *$\forall j \in \mathcal{N}_1^A$, $w_j = W_B + 1$, avec $W_B = \sum_{j \in \mathcal{N}^B} w_j$.*

Pour déterminer une solution optimale, nous formulons le problème comme une instance de flot à coût minimum sur l'ensemble des travaux $\mathcal{N}_1^A \cup \mathcal{N}^B$ en considérant les nouveaux poids. Nous déterminons par la suite, toutes les cliques maximales $L_1, \ldots, L_H$ du graphe d'intervalle. Les cliques maximales sont rangées selon l'ordre non-décroissant des dates de début du premier travail de chaque clique. Par construction des cliques maximales, chaque travail appartient soit à une seule clique soit à un groupe de cliques maximales consécutives.

Pour le calcul de flot, nous construisons le graphe orienté G comme suit :

- *Créer les sommets $L_0, \ldots, L_H$ et des arcs pour $h = 1, \ldots, H, (v_i, v_{i-1})$. Les coûts sur ces arcs sont égaux à 0 et leurs capacités sont infinies. Chaque arc (v_h, v_{h-1}) représente la clique L_h.*
- *Pour chaque travail, si J_j est dans les cliques $L_h, \ldots, L_{h+l}$, on ajoute un arc (v_{h-1}, v_{h+l}) d'un coût w_j et de capacité 1.*
- *Pour chaque clique h qui n'est pas de taille maximale, nous ajoutons un arc (v_{h-1}, v_h) d'un coût 0 et de capacité égale à la taille maximale d'une clique moins la taille de la clique h. Il s'agit donc d'introduire autant de travaux fictifs pour que toutes les cliques contiennent le même nombre de travaux.*

Soit $TailleMax = \max_{1 \leq h \leq H}(|L_h|)$. Une solution au problème d'ordonnancement est donnée par la recherche d'un flot de taille $(TailleMax - m)$ à coût minimum. En effet, à un instant donné, au maximum m travaux sont ordonnancés. Ainsi, dans le graphe G, nous cherchons les travaux à rejeter minimisant le coût total de rejet. Si un arc (v_{h-1}, v_{h+l}) (correspond à un travail J_j) a un flux non nul, ce travail J_j est alors rejeté.

Il est clair que si le coût du flot dépasse W_B alors il n'existe pas de solution réalisable puisque au moins un des travaux de $\mathcal{N}_1^A$ est rejeté.

Exemple : Reprenons l'exemple 1.3 avec $m = 2$. Comme illustré dans la Figure 5.1, nous avons trois cliques maximales, L_1, L_2 et L_3. La taille maximale est $|L_1| = 6$. Nous augmentons chacune des deux autres cliques L_2 et L_3 par un travail fictif de poids nul. Comme le nombre de machines est de deux, pour chaque clique, le nombre maximum de travaux que nous pouvons ordonnancer est de deux. Nous cherchons donc un flot de taille (6-2=4) à coût minimum.

5.4.2.2 $\mathbf{s_i \leq s_j \Rightarrow f_i \leq f_j}$

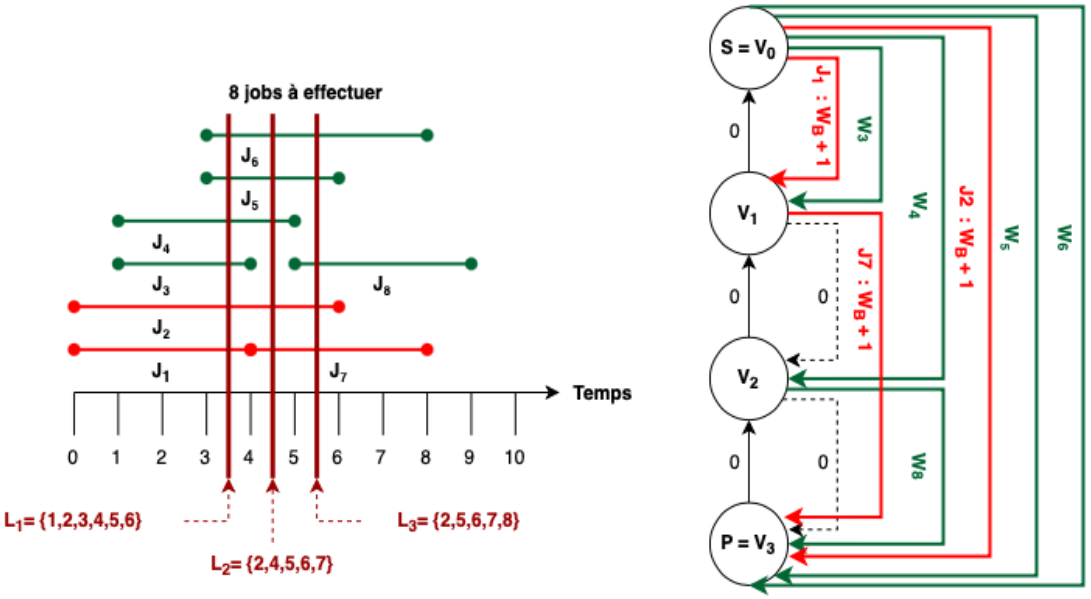
Dans cette section, nous cherchons à ordonnancer des travaux avec des durées opératoires et des poids quelconques sur machines parallèles. Une machine ne peut exécuter qu'un seul travail à la fois. Nous supposons que les intervalles de temps des travaux vérifient la condition suivante : $\forall J_i, J_j \in \mathcal{N} = \mathcal{N}^A \cup \mathcal{N}^B : s_{j_1} \leq s_{j_2} \leq f_{j_1} \leq f_{j_2}$.

Pour résoudre le cas de m machines, étudions d'abord le cas d'une seule machine.

Cas d'une seule machine

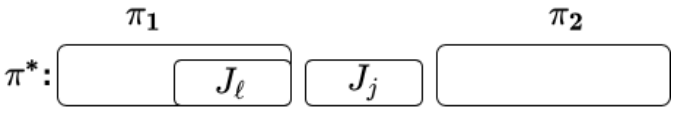
Comme les dates de début et de fin des travaux vérifient la condition $s_{j_1} \leq s_{j_2} \leq f_{j_1} \leq f_{j_2}$, trier les travaux selon l'ordre non-décroissant des s_i donne le même ordre que selon le tri non-décroissant des f_j. Les travaux sont alors indexés selon l'ordre non-décroissant des s_j. Soit π^* une solution optimale et soit J_i (resp. J_j) le premier travail rejeté (resp.

FIGURE 5.1 – Flot à coût min.



ordonnancé). J_i et J_j sont deux travaux d'un même agent avec $s_i \leq s_j$. La solution π^* peut être décrite par : $\pi^* = \pi_1//J_j//\pi_2$ avec J_l le dernier travail ordonnancé dans π_1. La notation $a//b$ représente la concaténation de a et b. La Figure 5.2 illustre la solution π^*.

FIGURE 5.2 – Structure de la solution π^*



Nous avons les lemmes suivants.

Lemma 1. *Soit la solution optimale π^*. Si le travail J_i est dans la même clique maximale que les travaux J_l et J_j (cf Figure 5.3.(a)). Remplacer J_j par J_i augmente alors le nombre total de travaux rejetés d'au moins 2 travaux.*

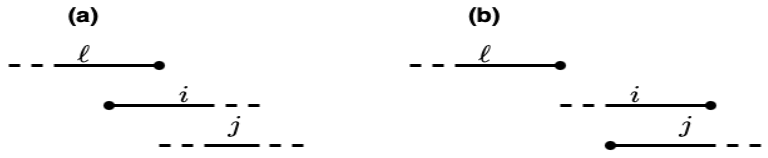
Preuve. *Si les travaux J_l, J_i et J_j appartiennent tous au même agent, la solution pour l'agent est alors dégradée de 2. Sinon, au moins un agent voit sa solution dégradée de 1.*

Lemma 2. *Considérons une solution optimale π^*. Supposant que le travail J_i se chevauche uniquement avec J_j (cf Figure 5.3.(b)). Remplacer J_j par J_i donne : $F(\pi) = F(\pi^*)$.*

Preuve. *Le nombre de travaux ordonnancés dans les deux solutions est le même. Cependant, selon la solution π la machine est libérée plus tôt.*

Lemma 3. *L'ordonnancement des travaux selon l'ordre non-décroissant des s_i en libérant la machine au plus tôt est dominant.*

FIGURE 5.3 – Etude de cas selon l'intervalle du temps de J_i



Preuve. *Conséquence directe des Lemmes 1 et 2.*

Pour calculer une solution optimale au sens de Pareto, nous proposons le programme dynamique suivant.

Programme dynamique

— Ré-indexer les travaux des deux agents confondus selon l'ordre croissant des s_i (identique selon f_i)

— Soit $C(j, U^A, U^B)$ date de fin du dernier travail ordonnancé en considérant les j premiers travaux, avec U^A (resp. U^B) nombre de travaux rejetés de l'agent A (resp. B)

1. **Valeurs initiales** :
   — $C(0, U^A, U^B) = 0 \,; \forall \,; U^A \geq 0 \,; U^B \geq 0$
   — $C(j, U^A, U^B) = +\infty \,; si \,; U^A < 0 \,; U^B < 0$
2. **Fonction de récurrence** :

$$C(j, U^A, U^B) = \begin{cases} min(f_j + F(j, U^A, U^B); C(j-1, U^A - 1, U^B)) & si\ j \in \mathcal{N}^A \\ \\ min(f_j + F(j, U^A, U^B); C(j-1, U^A, U^B - 1)) & si\ j \in \mathcal{N}^B \end{cases}$$

Avec

$$F(j, U^A, U^B) = \begin{cases} +\infty & si\ C(j-1, U^A, U^B) > s_j \\ 0 & sinon; \end{cases}$$

3. **Solution optimale** :
   Pour déterminer une solution optimale avec Q_A donné, selon l'état final du programme dynamique, nous cherchons la valeur minimale de U^B pour laquelle $C(n, Q_A, U^B) < +\infty$. Nous pouvons ainsi écrire : $U^{*B} = min(U^B : C(n, Q_A, U^B) < +\infty)$.

Selon ce programme dynamique, pour un travail j donné, nous avons deux décisions : soit le travail est rejeté et donc la date de fin du dernier travail ordonnancé est celle qui correspond à la décision prise à l'état $j-1$, sinon la date de fin est f_j si la machine est disponible à la date s_j.

Nous avons donc la proposition suivante :

Proposition 9. *Le problème* $1|CO, s_j, f_j, \sum_{j\in\mathcal{N}_A} x_j \leq Q_A| \sum_{j\in\mathcal{N}_B} x_j$ *tel que* $\forall J_i, J_j \in \mathcal{N} = \mathcal{N}^A \cup \mathcal{N}^B : s_i \leq s_j \leq f_i \leq f_j$ *est Polynomial. Une solution optimale au sens de Pareto peut-être calculée en* $O(nn_An_B)$.

Remarque 8. *Le programme dynamique permet de générer l'ensemble du front de Pareto, dont le front optimal avec une complexité de* $O(nn_An_B)$*. Il est clair que le nombre de solutions de Pareto non-dominées est borné par* $O(n_k)$ *tel que* $n_k = min(n_A, n_B)$*.*

Remarque 9. *Le programme dynamique peut être généralisé au cas de* O *agents pour déterminer le front optimal avec une complexité de* $O(n\prod_{k=1}^{O} n_k)$

Cas plusieurs machines

Le programme dynamique présenté dans la section précédente cherche à placer un nouveau travail sur la machine libre au plus tôt (minimiser la date de fin du dernier travail ordonnancé). Comme les travaux sont pris dans l'ordre non-décroissant de leur date de début s_j, si un conflit entre deux travaux existe, la priorité est donnée à celui qui a la plus petite date de fin f_j, conformément au Lemme 3. Nous introduisons maintenant l'algorithme 6 permettant de déterminer une solution non-dominée du problème $Pm|CO, s_j, f_j|\sum_{j\in\mathcal{N}_A} x_j, \sum_{j\in\mathcal{N}_B} x_j$ tel que $\forall J_i, J_j \in \mathcal{N} = \mathcal{N}^A \cup \mathcal{N}^B : s_i \leq s_j \leq f_i \leq f_j$.

Algorithm 6 Ordre total sur dates de début et sur dates de fin des travaux identiques

$MC_{i,U^A,U^B} = 0, i = 1\ldots, m$ //date de disponibilité de M_i avec U^A et U^B travaux rejetés de A et B
$Prec(j) = 0, j = 1\ldots, n$ // indice de la machine sélectionnée pour ordonnancer j
for $j = 1$ **to** n **step** 1 **do**
  for $U^A = 1$ **to** n_A **step** 1 **do**
    for $U^B = 1$ **to** n_B **step** 1 **do**
      $k = argmin\{i : min_{1\leq i\leq m}(MC_{i,U^A,U^B})\}$ //La machine libre au plus tôt
      $l = Prec^{-1}(k)$
      Calculer MC_{k,U^A,U^B} par la formule de récurrence 5.4
    end for
  end for
  $Prec(j) = k$
end for
return une solution strictement non-dominée respectant Q_A

Formule de récurrence :

$$C(j, U^A, U^B) = \begin{cases} min(f_j + F(j, U^A, U^B); C(l, U^A - 1, U^B)) & si\ j \in \mathcal{N}^A \\ min(f_j + F(j, U^A, U^B); C(l, U^A, U^B - 1)) & si\ i \in \mathcal{N}^B \end{cases} \tag{5.4}$$

Avec :

$$F(j, U^A, U^B) = \begin{cases} +\infty & si\ C(l, U^A, U^B) > s_i \\ 0 & sinon; \end{cases}$$

Selon l'algorithme 6, pour chaque travail J_j pris dans l'ordre non-décroissant des dates de début, deux décisions sont à considérer : ordonnancer le travail ou le rejeter. Un travail est rejeté si et seulement si la date de disponibilité de la machine M_k libre au plus tôt (MC_{k,U^A,U^B}) est supérieure ou égal à s_j. Notons que si le travail j est rejeté sur M_k, alors toute autre machine le rejettera. Contrairement au cas d'une seule machine, l'état précédent correspond à la décision prise pour l'ordonnancement du dernier travail J_l sur M_k avec (U^A, U^B) travaux rejetés de A et B. Selon l'algorithme 6, le travail J_l est déterminé par

$Prec^{-1}(k)$. Il s'agit du dernier travail appelant le programme dynamique avec la machine M_k. Si la décision était d'ordonnancer le travail J_l alors $MC_{l,U^A,U^B} = f_l$.

Pour déterminer une solution optimale au sens de Pareto avec $\sum_{j\in\mathcal{N}_A} x_j \leq Q_A$, nous cherchons dans la table finale $C_(n, U^A, U^B)$ la valeur minimale de U^B pour laquelle $U^A \leq Q_A$ et $C(n, U^A, U^B) < +\infty$, c'est-à-dire une solution réalisable.

Proposition 10. *Le problème $Pm|CO, s_j, f_j, s_i \leq s_j \Rightarrow f_i \leq f_j|\varepsilon(z^A, z^B)$ est Polynomial. Une solution de Pareto strict peut-être calculée en $O(m(nn_An_B))$.*

Comme pour le cas d'une seule machine, nous avons les deux remarques suivantes.

Remarque 10. *Le programme dynamique permet de générer l'ensemble du front de Pareto optimal avec une complexité de $O(mnn_An_B)$. Il est clair que le nombre de solutions de Pareto non-dominées est borné par $O(n_k)$ tel que $n_k = min(n_A, n_B)$.*

Remarque 11. *Ce programme dynamique peut être généralisé au cas O agents pour déterminer le front optimal avec une complexité de $O(mn\prod_{k=1}^{O} n_k)$*

5.5 Méthodes exactes

Dans cette section, nous allons généraliser les méthodes exactes que nous avons proposées dans le chapitre 3, notamment pour $PLNE-3$, PPC et Branch&Price au cas multiagents. Pour trouver une solution de Pareto nous allons utilisé l'approche ε-contrainte où la fonction objectif de l'agent A est bornée. Pour énumérer l'ensemble des solutions strictes de Pareto, nous résolvons le problème $Pm|CO, s_j, f_j, r_j, Q_A \leq Z^A|Z^B$ avec différentes valeurs de Q_A. La première étape consiste à calculer la borne inférieure LB pour l'agent A, on pose $Q_A = LB$ et $I = 1$, la solution trouvée est notée $(\hat{Z}^A, \hat{Z}^B)$. Ensuite, on résout le problème inverse $Pm|CO, s_j, f_j, r_j, \hat{Z}^B \leq Z^B|Z^A$. La solution obtenue est une solution de Pareto stricte, notée $(\hat{Z}^{A'}, \hat{Z}^B)$, et on ajoute cette solution à l'ensemble des solutions de Pareto strictes $\mathcal{S}$. On met à jour la borne Q_A comme suit : $Q_A = \hat{Z}^{A'} + 1$, on incrémente la valeur de I, et on réitère le procédé. Si aucune solution réalisable n'est obtenue, alors on arrête le procédé. $\mathcal{S}$ est alors le front de Pareto strict. L'algorithme 7 décrit la procédure pour le cas de deux agents.

Algorithm 7 Enumération du front de Pareto optimal par l'approche ε-contrainte

```
1: Trouver Z^B en résolvant Pm|CO, s_j, f_j, r_j, Q_A ≤ Z^A|Z^B ; σ^1 = (Ẑ^A, Ẑ^B) ; Q_A = Ẑ^A = LB
2: I = 1 ; S = σ^1
3: while Q_A ≤ UB do
4:     Résoudre le problème inverse Pm|CO, s_j, f_j, r_j, Ẑ^B ≤ Z^B|Z^A
5:     On note σ^I = ((Ẑ^A)^I, (Ẑ^B)^I) la solution trouvée
6:     if (Ẑ^B)^I = (Ẑ^B)^{I-1} then
7:         S = (S \ {σ^{I-1}}) ∪ {σ^I}
8:     else
9:         S = S ∪ {σ^I}
10:    end if
11:    Q_A = (Ẑ^A)^I + 1 ; I++
12: end while
13: return S
```

5.5.1 Programmation linéaire en nombres entiers

Une formulation de programmation linéaire en nombres entiers $PLNE3Agent$ peut être donnée comme suit :

$$\text{Maximize} : \sum_{i=1}^{m}\sum_{j=1}^{n_B} w_j x_{ij} \tag{5.5}$$

$$\sum_{j\in L_h} r_{jk}x_{ij} \leq R_k \qquad i=1,\ldots,m;\ k=1,\ldots,\mathcal{K};\ h=1,\ldots,H \tag{5.6}$$

$$\sum_{i=1}^{m} x_{ij} \leq 1 \qquad j=1,\ldots,n \tag{5.7}$$

$$Q_A \leq \sum_{i=1}^{m}\sum_{j=1}^{n_A} w_j x_{ij} \tag{5.8}$$

$$x_{ij} \in \{0,1\} \qquad j=1,\ldots,n;\ i=1,\ldots,m \tag{5.9}$$

5.5.2 Programmation par contraintes

La formulation par CPO, noté $PPCAgent$ est comme suit :

$$\text{Maximize} : \sum_{i=1}^{m}\sum_{j=1}^{n_B} w_j * IloPresenceOf(ITer_{ij}) \tag{5.10}$$

$$CumF_{ik} = \sum_{k\in K} pulse(ITer_{ij}, r_{jk})\ k=\overline{1,K};\ i=\overline{1,m};\ j=\overline{1,n} \tag{5.11}$$

$$CumF_{ik} \leq R_k \qquad i=\overline{1,m};\ k=\overline{1,K} \tag{5.12}$$

$$\sum_{i=1}^{m} IloPresenceOf(ITer_{ij}) \leq 1 \qquad j=\overline{1,n} \tag{5.13}$$

$$Q_A \leq \sum_{i=1}^{m}\sum_{j=1}^{n_A} w_j * IloPresenceOf(ITer_{ij}) \tag{5.14}$$

$$setOptional(ITer_{ij}) \quad j=\overline{1,n};\ i=\overline{1,m} \tag{5.15}$$

5.5.3 Branch&Price pour l'ordonnancement multiagent

Les étapes de décomposition appliquées au cas monocritère sont les mêmes pour le cas multiagent, nous rajoutons au problème maître implicite obtenu $PM-2$ la contrainte de l'approche ε-contrainte appliquée au critère de l'agent B. Le problème maître implicite $PM-2-Agent$ est comme suit :

$$\text{Maximize} : \sum_{Y^p\in\mathcal{P}} c_p^B\ z_p \tag{5.16}$$

$$\sum_{Y^p \in \mathcal{P}} y_j^p z_p \leq 1 \; (j = 1, \ldots, n) \tag{5.17}$$

$$\sum_{Y^p \in \mathcal{P}} z_p \leq m \tag{5.18}$$

$$Q_A \leq \sum_{Y^p \in \mathcal{P}} c_p^A \; z_p \tag{5.19}$$

$$z_p \in \{0, 1\} \; (Y^p \in \mathcal{P}) \tag{5.20}$$

Les variables z_p sont des variables binaires avec $z_p = 1$ si le vecteur est choisi dans la solution ; 0 sinon. Soit $c_p^A = \sum_{j=1}^{n_A} w_j y_j^p$ (resp. $c_p^B = \sum_{j=1}^{n_B} w_j y_j^p$). Pour chaque vecteur p, les variables binaires y_j^p sont définies comme suit : $y_j^p = 1$ si le travail j est ordonnancé ; 0 sinon.

Ainsi, le pricing problème PP correspondant est le suivant :

$$\text{Maximize} : \sum_{j=1}^{n} (w_j - \pi_j^\star) y_j \tag{5.21}$$

$$\sum_{j \in L_h} r_{jk} y_j \leq R_k \; (k = 1, \ldots, K; h = 1, \ldots, H) \tag{5.22}$$

$$y_j \in \{0, 1\} \; (j = 1, \ldots, n) \tag{5.23}$$

La génération de colonnes et le schéma de branchement resteront les mêmes (voir section 3.6.2)

5.6 Résultats expérimentaux des méthodes exactes

Dans cette section nous analysons les performances des méthodes exactes développées pour le cas mutiagent. Pour tester et comparer nos méthodes, nous avons implémenté les modèles mathématiques introduits dans la section 5.5.1 ainsi que l'approche par PPC de la section 5.5.2 et le Branch&PriceAgent présenté dans la section 5.5.3 en $C++$.

Tous nos tests ont été effectués sur une machine de 2,2 GHz Intel Core i7 et 16 Go de mémoire.

Afin d'analyser la performance des méthodes exactes proposées, nous avons limité le temps de calcul à 600 secondes (10 minutes). Nous avons utilisé IBM ILOG CPLEX Optimization Studio version 12.8 avec 1 thread pour résoudre le modèle linéaire en nombres entiers et le concept de variables d'intervalle et en utilisant les contraintes globales d'ordonnancement déjà définies dans IBM CP Optimizer (CPO) pour résoudre le modèle de programmation par contraintes.

5.6.1 Jeux de données

Dans cette section, l'ensemble des instances considéré est le jeu de données Type 2 (voir 4.1) avec n travaux au total.

Concernant les travaux de chaque agent, deux configurations sont considérées :

1. $(AgentA, AgentB) = (30\%, 70\%) \times n$
2. $(AgentA, AgentB) = (50\%, 50\%) \times n$

5.6.1.1 Calcul de point de Pareto strictement non-dominé

Par l'approche ε-contrainte, nous nous sommes intéressés à calculer une solution de Pareto strictement non-dominée. Afin d'évaluer les performances de nos approches de résolution, pour chaque instance de données, nous cherchons trois points strictement non dominés obtenus selon la valeur fixée de Q^A. En effet, les trois valeurs de Q^A ont été fixées selon la règle suivante : $Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$ et $W^A = \sum_{j \in \mathcal{N}^A} w_j$.

Rappelons que le problème étudié est symétrique et qu'une solution optimale au sens de Pareto est obtenue par la résolution de chaque instance deux fois : $Max\ Z^B(I)\ s.t.\ Z^A(I) \geq Q^A$ pour obtenir le point $(Z^A(I), Z^{*B}(I))$; Puis résoudre à l'optimum : $Max\ Z^A(I)\ s.t.\ Z^B(I) \geq Z^{*B}(I)$.

1. (Agent A, Agent B)=(30%, 70%)
   $Q^A = 30\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)
   $Q^A = 50\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)
   $Q^A = 80\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)
2. (Agent A, Agent B)=(50%, 50%)
   $Q^A = 30\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)
   $Q^A = 50\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)
   $Q^A = 80\%W^A$ Tableaux (PLNE3, PPC-PLNE3, B&P-Agent)

5.6.2 Résultats $(n_A, n_B) = (30\%, 70\%)$

Le tableau 5.2 et la figure 5.4 résument les performances des PLNE3-Agent, PPCPLNE3-Agent et B&P-Agent en faisant varier le nombre de travaux de l'agent A (la colonne $30\%n$) et le nombre de machines (la colonne m). Pour chaque n_A, une valeur Q_A est fixée (la colonne $Q_A = \alpha W^A$). Les colonnes CPU (s) représentent le temps de calcul moyen en secondes. Les colonnes $\%S^*$ (point de Pareto strictement non-dominé) représentent le pourcentage des instances (10 instances par m) résolues à l'optimum pour les deux agents (les deux critères) et les colonnes $\%WS^*$ (point de Pareto faiblement dominé) représentent le pourcentage des instances (10 instances par m) résolues à l'optimum pour l'un des agents (i.e un critère est résolu à l'optimum et l'autre non).

TABLE 5.2 – Pourcentage de points de Pareto strictement non-dominés et faiblement dominés

Type2			PLNE3-Agent			ppcPLNE3-Agent			B&P-Agent		
$Q_A = \alpha W^A$	$30\%n$	m	%S*	%WS*	CPU(S)	%S*	%WS*	CPU(S)	%S*	%WS*	CPU(S)
$\alpha = 30\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	100%	0%	3	100%	0%	5	100%	0%	43
		7	100%	0%	9	100%	0%	16	100%	0%	21
		10	100%	0%	8	100%	0%	8	100%	0%	10
		15	100%	0%	26	100%	0%	22	100%	0%	18
	100	4	100%	0%	26	100%	0%	26	**100%**	**0%**	237
		7	90%	10%	229	100%	0%	233	**100%**	**0%**	175
		10	80%	20%	354	80%	10%	315	**100%**	**0%**	91
		15	70%	30%	416	60%	40%	404	**100%**	**0%**	32
	150	4	**50%**	20%	659	40%	20%	620	40%	20%	605
		7	0%	10%	1175	**10%**	0%	1105	**10%**	0%	1080
		10	0%	0%	1197	0%	10%	1145	**10%**	0%	1105
		15	0%	**20%**	1181	0%	10%	1155	0%	10%	1140
$\alpha = 50\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	**100%**	0%	3	**100%**	0%	4	90%	10%	79
		7	100%	0%	14	100%	0%	23	100%	0%	26
		10	100%	0%	11	100%	0%	9	100%	0%	5
		15	100%	0%	23	100%	0%	21	100%	0%	18
	100	4	100%	0%	35	100%	0%	37	100%	0%	221
		7	50%	20%	680	**60%**	**30%**	546	**60%**	**30%**	242
		10	**60%**	20%	675	40%	40%	598	**60%**	**40%**	259
		15	60%	**20%**	583	60%	**20%**	478	**90%**	0%	143
	150	4	**80%**	10%	358	70%	20%	308	50%	**40%**	399
		7	0%	0%	1193	0%	0%	1193	0%	**10%**	1096
		10	0%	0%	1197	0%	0%	1197	0%	0%	1190
		15	0%	0%	1193	0%	0%	1198	0%	0%	1193
$\alpha = 80\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	80%	0%	1	80%	0%	1	80%	0%	42
		7	90%	0%	10	90%	0%	19	90%	0%	62
		10	100%	0%	19	100%	0%	21	100%	0%	31
		15	100%	0%	34	100%	0%	32	100%	0%	7
	100	4	-	-	-	-	-	-	-	-	-
		7	80%	20%	293	80%	20%	261	80%	20%	248
		10	30%	**30%**	795	40%	20%	631	**70%**	20%	280
		15	30%	20%	855	40%	10%	721	**60%**	0%	254
	150	4	-	-	-	-	-	-	-	-	-
		7	0%	**20%**	1030	0%	10%	1120	**10%**	0%	1090
		10	0%	0%	1197	0%	0%	1195	**10%**	**10%**	1100
		15	0%	0%	1196	0%	0%	1194	**30%**	0%	840

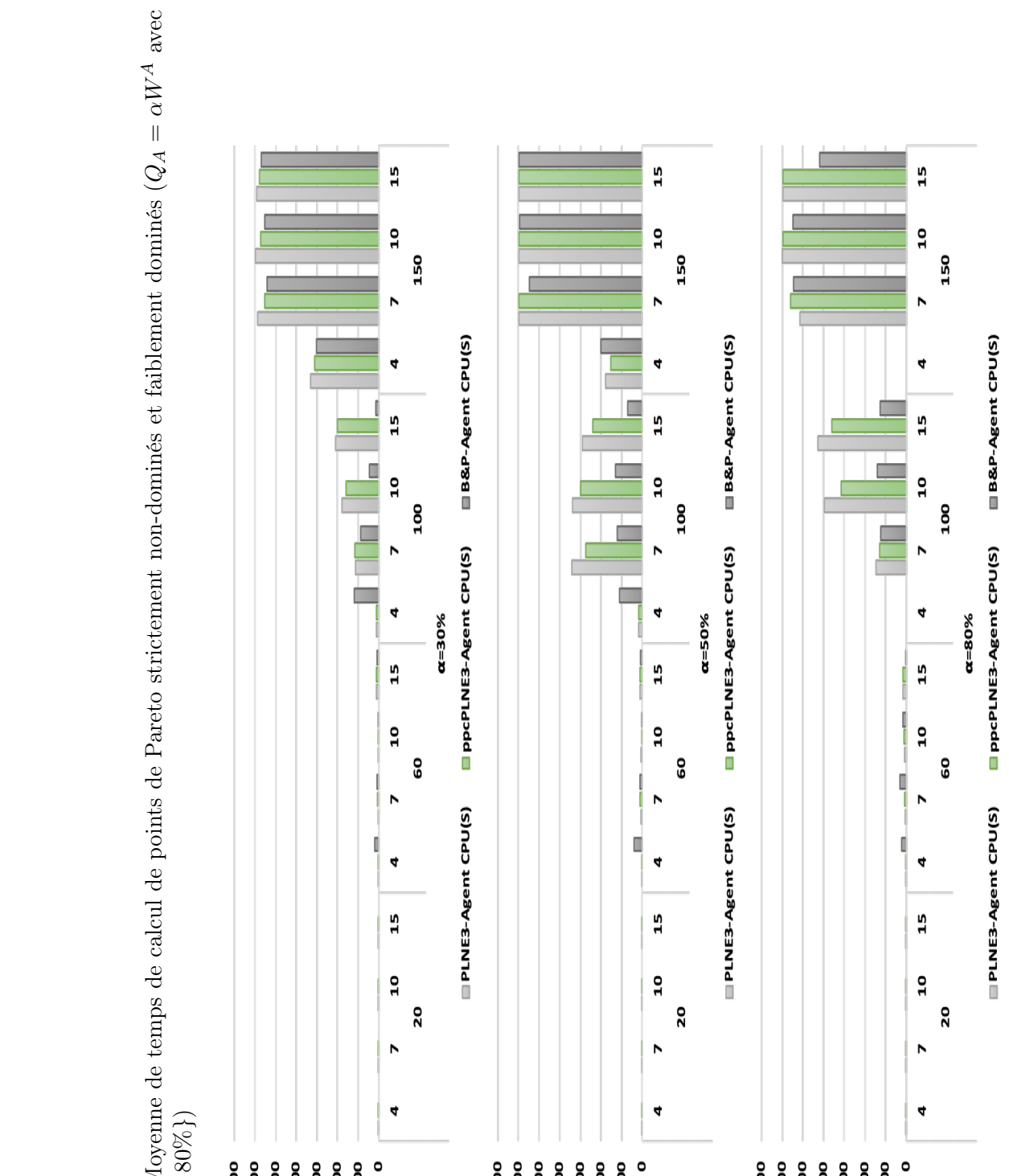


FIGURE 5.4 – Moyenne de temps de calcul de points de Pareto strictement non-dominés et faiblement dominés ($Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$)

Dans la figure 5.4, nous observons que $\forall\alpha \in \{30\%, 50\%, 80\%\}$ le temps de calcul moyen augmente avec l'augmentation du nombre de travaux et nombre de machines pour les trois modèles PLNE3-Agent, PPCPLNE3-Agent et B&P-Agent. Nous observons aussi, pour les instances (n=20, m=$\{4,7,10,15\}$) que les trois modèles convergent instantanément (le temps de calcul moyen est quasiment nul). Pour les instances (n=60, m=$\{4,7,10,15\}$) les trois modèles ont quasiment les mêmes performances en termes de temps de calcul. Pour les instances (n=$\{100, 150\}$, m=$\{7,10,15\}$) le B&P-Agent produit des temps de calcul inférieurs à ceux des modèles PLNE3-Agent et PPCPLNE3-Agent.

Dans le tableau 5.2, nous observons que $\forall\alpha \in \{30\%, 50\%, 80\%\}$ le pourcentage des instances résolues à l'optimum au sens de Pareto (points de Pareto strict) diminue avec l'augmentation du nombre de travaux et du nombre de machines pour les trois modèles PLNE3-Agent, PPCPLNE3-Agent et B&P-Agent.

Lorsque l'un des modèles ne trouve pas de solution optimale, soit il échoue ou soit il trouve une solution de Pareto faible. Rappelons que pour Q_A donné, chaque instance est résolue deux fois durant le temps d'exécution alloué. Un pourcentage dans la colonne $\%WS*$ indique que le modèle a pu trouver une solution optimisant le critère de l'agent B tout en satisfaisant la contrainte sur Q_A, mais n'a pas pu résoudre le problème symétrique dans les temps impartis. Ainsi, si nous regardons les résultats pour 150 travaux avec 10 machines pour $\alpha = 80\%$ (l'avant dernière ligne de la Table 5.2), là où les autres méthodes échouent, le B&P-Agent a permis de résoudre une seule instance à l'optimum. Nous remarquons également que pour une seule instance le B&P-Agent a trouvé une solution faible : le B&P-Agent a eu suffisamment du temps pour minimiser $Z^B(I)$ s.t. $Z^A(I) \leq Q_A(I)$, mais pas assez pour résoudre le problème symétrique, i.e. minimiser $Z^A(I)$ s.t. $Z^B(I) \leq Z^{*B}(I)$.

Nous observons aussi, pour les instances (n=$\{20, 60\}$, m=$\{4,7,10,15\}$) que les trois modèles trouvent les points de Pareto strictement non-dominés.

Globalement les trois modèles produisent les mêmes performances en termes de pourcentage d'instances résolues à l'optimum et arrivent à trouver des points de Pareto stricts ou Pareto faibles, sauf pour les instances avec $\alpha = 50\%$ et (n=150, m=$\{10,15\}$).

Les pourcentages en **gras** dans le tableau 5.2 sont les cas où l'un des modèles produit les meilleures performances, par exemple, pour les instances avec $\alpha = 30\%$ et (n=100, m=$\{4,7,10,15\}$) le B&P-Agent domine $PLNE3 - Agent$ et $PPCPLNE3 - Agent$ où il trouve 100% de points de Pareto strict. Pour les instances avec $\alpha = 80\%$ et (n=150, m=$\{7,10,15\}$) le B&P-Agent arrive à trouver des points de Pareto stricts, contrairement aux $PLNE3 - Agent$ et $PPCPLNE3 - Agent$.

Notons tout de même que le PLNE3-Agent offre de bonnes performances pour les instances avec un nombre de machines égal à 4 quelque soit la taille de l'instance.

Sur l'ensemble des tests réalisés, nous constatons que les instances résolues (nous avons obtenu des solutions strictes ou faibles) par la B&P-Agent ne sont pas forcément celles résolues par les deux autres méthodes. Ce constat est encore plus accentué lorsque nous nous focalisons uniquement sur les deux premières approches PLNE3-Agent et ppcPLNE3-Agent. Sur l'ensemble des instances, à ce stade nous pouvons conclure que ces deux méthodes sont assez complémentaires, c'est-à-dire pour certaines instances PLNE3-Agent domine la ppcPLNE3-Agent et pour d'autres instances l'effet inverse se produit.

5.6.3 Résultats $(n_A, n_B) = (50\%, 50\%)$

TABLE 5.3 – Pourcentage de points de Pareto strictement non-dominés et faiblement dominés

Type2			PLNE3-Agent			ppcPLNE3-Agent			B&P-Agent		
$Q_A = \alpha W^A$	50%n	m	%S*	%WS*	CPU(S)	%S*	%WS*	CPU(S)	%S*	%WS*	CPU(S)
$\alpha = 30\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	100%	0%	2	100%	0%	3	100%	0%	47
		7	100%	0%	3	100%	0%	4	100%	0%	29
		10	100%	0%	4	100%	0%	3	100%	0%	8
		15	100%	0%	20	100%	0%	17	100%	0%	10
	100	4	**100%**	0%	16	**100%**	0%	14	80%	0%	270
		7	**100%**	0%	125	90%	10%	156	70%	10%	436
		10	**100%**	0%	107	**100%**	0%	115	90%	10%	138
		15	80%	20%	273	90%	10%	203	**100%**	0%	95
	150	4	**60%**	10%	631	50%	30%	567	20%	40%	1114
		7	**30%**	10%	1007	20%	20%	819	10%	30%	1085
		10	0%	20%	1112	0%	20%	898	**10%**	**20%**	1098
		15	0%	60%	1053	0%	60%	842	**50%**	**20%**	570
$\alpha = 50\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	100%	0%	5	100%	0%	5	100%	0%	99
		7	100%	0%	14	100%	0%	16	100%	0%	91
		10	100%	0%	4	100%	0%	5	100%	0%	3
		15	100%	0%	24	100%	0%	22	100%	0%	10
	100	4	**100%**	0%	106	**100%**	0%	99	50%	0%	695
		7	80%	10%	463	80%	0%	405	**90%**	0%	365
		10	**60%**	10%	657	50%	**30%**	557	50%	20%	627
		15	70%	30%	289	70%	30%	274	**90%**	**10%**	106
	150	4	0%	0%	1196	0%	0%	1078	0%	0%	1195
		7	0%	0%	1195	0%	0%	1078	0%	0%	1195
		10	0%	0%	1198	0%	0%	1075	**10%**	0%	1090
		15	0%	0%	1181	0%	0%	1078	0%	0%	1195
$\alpha = 80\%$	20	4	100%	0%	0	100%	0%	0	100%	0%	0
		7	100%	0%	0	100%	0%	0	100%	0%	0
		10	100%	0%	0	100%	0%	0	100%	0%	0
		15	100%	0%	0	100%	0%	0	100%	0%	0
	60	4	-	-	-	-	-	-	-	-	-
		7	90%	0%	21	90%	0%	23	90%	0%	214
		10	100%	0%	12	100%	0%	14	100%	0%	16
		15	100%	0%	34	100%	0%	29	100%	0%	11
	100	4	-	-	-	-	-	-	-	-	-
		7	10%	10%	711	10%	0%	607	**20%**	0%	494
		10	20%	50%	805	40%	10%	653	**70%**	0%	561
		15	30%	20%	907	30%	10%	721	**60%**	0%	557
	150	4	-	-	-	-	-	-	-	-	-
		7	-	-	-	-	-	-	-	-	-
		10	-	-	-	-	-	-	-	-	-
		15	0%	0%	1182	0%	0%	1077	**10%**	0%	1090

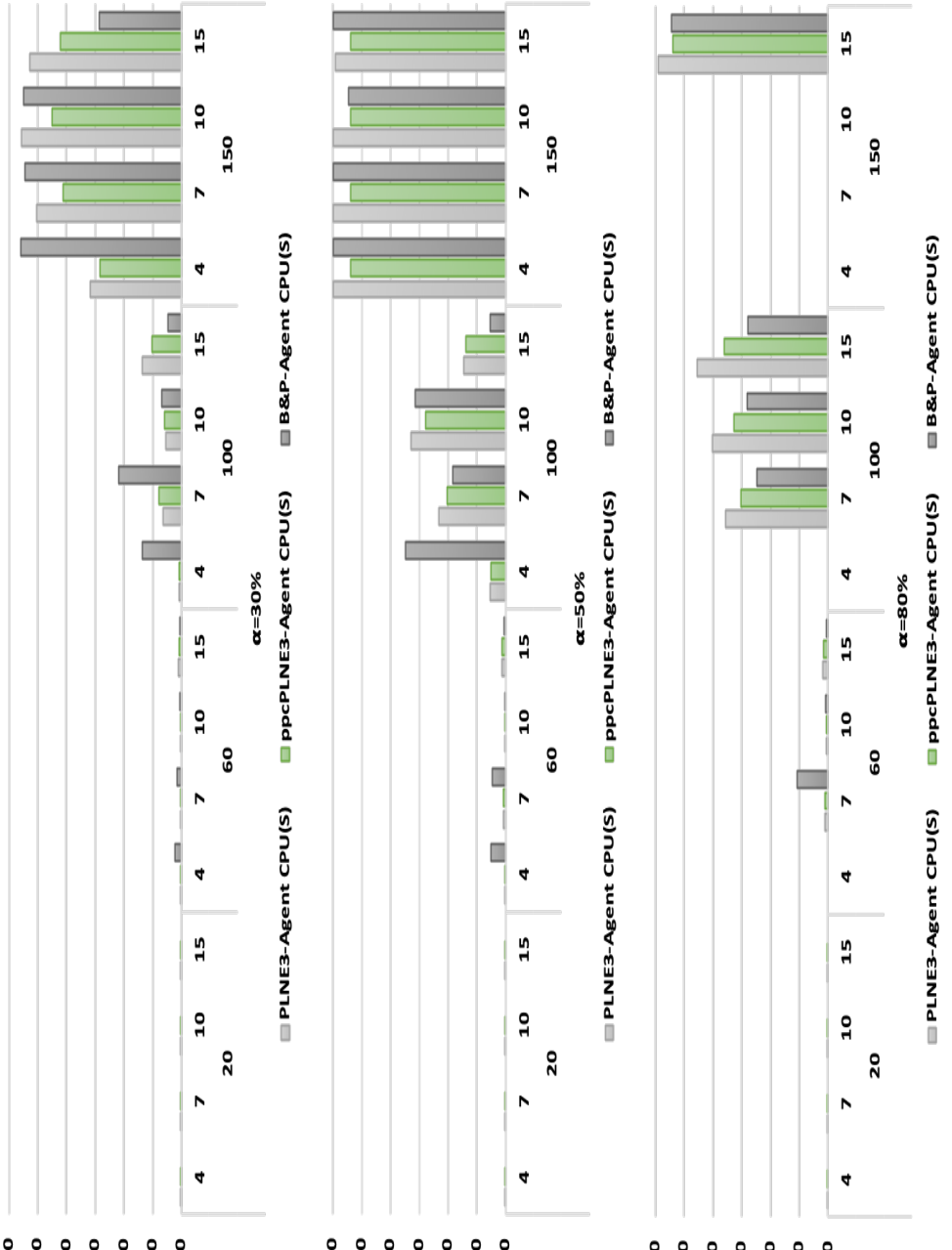


FIGURE 5.5 – Moyenne de temps de calcul de points de Pareto strictement non-dominés et faiblement dominés ($Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$)

Les résultats expérimentaux confirment les affirmations de la littérature traitant l'ordonnancement multiagent, à savoir lorsque les agents ont presque le même nombre de travaux, les instances à résoudre deviennent de plus en plus difficiles. Ainsi, les trois méthodes ne présentent pas des performances similaires au cas $(n_A, n_B) = (30\%n, 70\%n)$, où une tendance de la dominance de B&P-Agent par rapport aux deux autres approches se dégageait. Cependant, il est difficile de tirer une telle conclusion dans le cas $(n_A, n_B) = (50\%n, 50\%n)$. Néanmoins, nous constatons qu'en général le PLNE3-Agent présente de meilleures performances que le B&P-Agent (voire similaire) sur les instances de taille moyenne avec un nombre de machines limité (jusqu'à 7 machines). Cependant, en moyenne le B&P-Agent offre de meilleures performances pour les instances avec au moins 100 travaux et au moins 10 machines, avec des temps de calculs en moyenne meilleurs.

Notons tout de même que les trois approches de résolution offrent les mêmes performances pour les instances allant jusqu'à 60 travaux indépendamment du nombre de machines.

5.7 Conclusions du chapitre

L'étude des problèmes d'ordonnancement multiagents lorsque les sous-ensembles des travaux sont disjoints, s'avère utile dans le cas "d'ordonnancement distribué" dans le cas d'un système centralisé en présence d'un Ressource Manager, par exemple. Notre travail consiste donc à calculer un ordonnancement optimisant les objectifs locaux des différents agents.

Dans ce chapitre, deux agents concurrents sont considérés. Cela permet d'illustrer nos approches de résolution. Toutes ces approches sont bien entendu facilement généralisables aux cas de K agents.

Comme le problème est $\mathcal{NP}$-difficile, nous avons examiné certains cas de problèmes particuliers où nous avons obtenu de nouveaux résultats de complexité résumés dans la Table 5.1.

Pour la résolution du cas général, nous avons développé des modèles mathématiques, de la PPC et une méthode de décomposition de type Branch&Price en se basant sur l'étude menée dans le cas d'un seul agent (problème d'ordonnancement monocritère). L'étude expérimentale montre l'intérêt du Branch&Price pour la résolution exacte des problèmes d'ordonnancement multiagents des travaux multiressources à intervalles fixes lorsque le nombre de travaux et de machines deviennent de plus en plus importants (on réduit en moyenne d'au moins 50% les temps de calcul par rapport au $PLNE3$). Pour des problèmes d'ordonnancement dans systèmes de production, cette approche s'avère efficace mais pour le contexte du travail de cette thèse, nous devons développer des méthodes approchées efficaces permettant de déterminer une solution dans un temps réduit. Il s'agit donc de l'objectif du chapitre suivant.

Chapitre 6

Méthodes approchées pour l'ordonnancement multiagent sur machines parallèles multiressources

6.1 Introduction

Après avoir exposé dans le chapitre précédent les méthodes exactes proposées pour la résolution du problème d'ordonnancement des travaux fixes en compétition sur machines parallèles identiques multiressources, nous allons dans ce chapitre présenter des heuristiques gloutonnes basées sur des listes de priorité, pour résoudre efficacement ce problème $\mathcal{NP}$-difficile lorsque l'approche ε-contrainte est considérée. Il s'agit donc d'algorithmes rapides pour générer une seule bonne solution de compromis sans en garantir l'optimalité. Nous montrons comment utiliser ces heuristiques pour générer une solution de Pareto non-dominée. Pour générer le front de Pareto approché, des algorithmes évolutionnaires tels que NSGA-II sont proposés dans la littérature pour leur efficacité lorsque le problème d'optimisation à résoudre est multicritère. Ces méthodes sont basées sur des mécanismes d'intensification et de diversification de la recherche afin de trouver les solutions nondominées. Ainsi, un algorithme NSGA-II a été développé et est présenté vers la fin de ce chapitre. L'analyse des performances des méthodes proposées clôture ce chapitre.

6.2 Heuristiques de liste

Dans cette section, nous allons introduire les heuristiques de liste développées pour la détermination d'une solution de Pareto strictement non-dominée. Il s'agit d'algorithmes dédiés permettant de résoudre le problème $Pm|CO, s_j, f_j, r_{jk}|\varepsilon(Z^A/Z^B)$, c'est-à-dire lorsque l'approche ε-contrainte est considérée.

Sur la base de l'étude menée pour le cas monocritère présentée dans le chapitre 4, nous proposons la généralisation des heuristiques gloutonnes au cas multiagent. Le principe de ces heuristiques est basé sur des règles de tri avec une fonction d'affectation des travaux aux machines. La règle d'affectation retenue dans ce chapitre est une affectation machine

par machine, notée **Str1** (cf section 4.2.2). Pour déterminer l'ordre de priorité entre les travaux, nous reprenons les règles de tri présentées dans la section 4.2.1, qui sont :

JS1 : trier les travaux selon l'ordre décroissant des $\mathbf{w_j}$,
JS2 : trier les travaux selon l'ordre croissant des $\frac{\mathbf{p_j}}{\mathbf{w_j}}$,
JS3 : trier les travaux selon l'ordre croissant des $\frac{\mathbf{max(r_{j,k})}}{\mathbf{w_j}}$,
JS4 :trier les travaux selon l'ordre croissant des $\frac{\mathbf{max(r_{j,k})\times p_j}}{\mathbf{w_j}}$.

En cas d'égalité entre deux travaux, pour arbitrer, nous considérons dans un second temps l'ordre croissant des $\mathbf{max(r_{j,k})}$, sinon l'ordre croissant des $\mathbf{p_j}$.

L'algorithme 8 décrit le schéma global des heuristiques développées. Selon cet algorithme, nous utilisons les notations suivantes :

— π est l'ordonnancement obtenu, indiquant si le travail est ordonnancé (ou non) et sur quelle machine,
— *seq* est un booléen indiquant si le travail a pu être ordonnancé (vrai) ou rejeté (faux),
— $\mathcal{H}_{NoSeq}$ est la liste des travaux de l'agent B rejetés,
— $\mathcal{N}[1]$ et $\mathcal{H}_{NoSeq}[1]$ indiquent le premier travail de $\mathcal{N}$ et $\mathcal{H}_{NoSeq}$ respectivement.

L'algorithme 8 fait appel à deux procédures : i. *tryToSchedule()*, ii. *Schedule()*. La première permet de vérifier si un travail, pris dans l'ordre de priorité prédéfini, peut être ordonnancé et donc inséré dans la séquence π en cours de construction (la solution). La seconde a pour but de construire une solution faisable respectant la borne de l'ε-contrainte. L'algorithme 8 peut être résumé en deux étapes suivantes :

1. Considérer l'ensemble de tous les travaux, sans distinction entre agents, indexés selon le choix de la règle de tri $\mathcal{R}$. Pour chaque travail j, si les ressources additionnelles sont disponibles durant tout l'intervalle de son exécution, il est alors ordonnancé sur la première machine offrant des disponibilités de ressources nécessaires (cf la procédure *tryToSchedule*. Il s'agit de l'algorithme 9).

2. Dans le cas où tous les travaux ont été considérés et si la contrainte sur Q_B n'est pas satisfaite, alors une seconde procédure est appelée. La procédure *Schedule* 10 considère uniquement les travaux rejetés de l'agent B appartenant à la liste $\mathcal{H}_{NoSeq}$. Cette procédure cherche à placer les travaux de B rejetés, pris un par un dans l'ordre prédéfini par la règle de tri $\mathcal{R}$, en supprimant les travaux de A déjà placés, tout en veillant à minimiser l'impact de cette suppression sur la valeur de la fonction objectif de l'agent A. Aucun travail de B déjà ordonnancé ne doit être remis en cause. Ainsi, soit le travail de B considéré est placé et nous passons au suivant, soit ce travail est définitivement rejeté. Cette procédure est réitérée tant que la contrainte sur Q_B n'est pas respectée (cf lignes 35 à 47).

Ainsi, l'algorithme retourne une solution réalisable π, sinon il échoue et indique la non existence d'une solution réalisable avec la valeur Q_B choisie.

Notons que pour chaque instance $\mathcal{I}$, nous avons quatre heuristiques de tri.

Algorithm 8 Heuristique de liste basée sur les poids des travaux

Données :
$\mathcal{N}, \mathcal{N}^A, \mathcal{N}^B, Q_B$
Initialisation :
Numéroter les travaux de $\mathcal{N}$ selon l'ordre défini par la règle $\mathcal{R}$
$Z^A = 0$, $Z^B = 0$ // valeurs des objectifs de chaque agent
$seq = Faux$
$\mathcal{H}_{NoSeq} = \emptyset$
$\pi = \emptyset$ //séquence de l'ordonnancement
Traitement :
while $(Z^B < Q_B) \wedge (\mathcal{N} \neq \emptyset)$ **do**
    $seq = tryToSchedule(\pi, \mathcal{N}[1])$
    if $seq = Vrai$ **then**
        if $\mathcal{N}[1] \in \mathcal{N}^A$ **then**
            $Z^A = Z^A + w_{\mathcal{N}[1]}$
        else
            $Z^B = Z^B + w_{\mathcal{N}[1]}$
        end if
    else
        if $\mathcal{N}[1] \in \mathcal{N}^B$ **then**
            $\mathcal{H}_{NoSeq} = \mathcal{H}_{NoSeq} \cup \mathcal{N}[1]$
        end if
    end if
    $\mathcal{N} = \mathcal{N} \setminus \{\mathcal{N}[1]\}$
end while
if $Z^B \geq Q_B$ **then**
    Retirer de $\mathcal{N}$ les travaux de B
    while $\mathcal{N} \neq \emptyset$ **do**
        $seq = tryToSchedule(\pi, \mathcal{N}[1])$
        if $seq = Vrai$ **then**
            $Z^A = Z^A + w_{\mathcal{N}[1]}$
        end if
        $\mathcal{N} = \mathcal{N} \setminus \{\mathcal{N}[1]\}$
    end while
    return π
else
    while $(Z^B < Q_B \wedge \mathcal{H}_{NoSeq} \neq \emptyset)$ **do**
        $seq = Schedule(\pi, \mathcal{H}_{NoSeq}[1])$
        if $seq = Vrai$ **then**
            $Z^B = Z^B + w_{\mathcal{N}[1]}$
        end if
        $\mathcal{H}_{NoSeq} = \mathcal{H}_{NoSeq} \setminus \{\mathcal{H}_{NoSeq}[1]\}$
    end while
    if $Z^B \geq Q_B$ **then**
        return π
    else
        return Pas d'ordonnancement faisable
    end if
end ifreturn 0

6.2.1 Procédure *tryToSchedule*

L'algorithme 9 décrit la procédure d'insertion d'un travail si les ressources demandées sont disponibles. Nous introduisons les notations suivantes :

- π est la séquence de l'ordonnancement en cours de construction,
- *seq* est un booléen indiquant si le travail peut être ordonnancé (vrai) ou rejeté (faux),
- $\mathcal{L}$ est la liste des cliques maximales contenant le travail j,
- $R^h_{i,k}(\pi)$ est la quantité résiduelle de chaque ressource de la machine i selon l'ordonnancement π. La quantité de chaque ressource résiduelle dépend des travaux déjà placés. Nous devons donc définir pour chaque clique maximale h une quantité $R^h_{i,k}(\pi)$. Ces quantités initialisées à 100% (aucun travail n'est encore ordonnancé), doivent être mises à jour dès lors un travail j est ordonnancé, pour chaque h tel que $j \in L_h$.

Selon la règle de tri $\mathcal{R}$ choisie, à chaque itération, l'algorithme essaye de placer le premier travail j de la liste à ordonnancer (cf. la procédure *tryToSchedule* définie par l'algorithme 9). Nous devons d'abord vérifier s'il n'y a pas de conflit avec les travaux déjà placés en analysant $R^h_{i,k}(\pi)$ de chaque clique contenant j. Si j est ordonnancé, nous actualisons π ainsi que la valeur de la fonction objectif de l'agent concerné, puis nous passons au prochain travail de la liste. L'algorithme réitère jusqu'à la satisfaction de la contrainte sur Q_B. S'il reste des travaux à ordonnancer non encore considérés, l'algorithme continue seulement avec les travaux de l'agent A, puis retourne la solution trouvée (cf. les lignes 24 à 33).

Algorithm 9 *tryToSchedule*(π, J_j)

```
1: L = {h : J_j ∈ L_h}
2: seq = Faux
3: for i = 1 à m do
4:     for h = L[1] à L[|L|] do
5:         for k=1 à K do
6:             if r_jk ≤ R^h_{i,k}(π) then
7:                 π = π ∪ N[1]
8:                 seq = Vrai
9:             end if
10:        end for
11:    end for
12: end for
13: return seq
```

6.2.2 Procédure *Schedule*

L'algorithme 10 est appelé lorsque la solution construite n'est pas réalisable, c'est-à-dire si $Z^B < Q_B$. Lors de la deuxième phase, nous cherchons à placer des travaux rejetés de l'agent B, pris un par un dans l'ordre défini par la règle de tri $\mathcal{R}$. Par conséquent, certains travaux de l'agent A déjà ordonnancés seront remis en cause et donc seront rejetés afin

d'obtenir une solution réalisable, si elle existe. Pour l'ordonnancement d'un travail j de l'agent B, cette procédure utilise les nouvelles notations suivantes :

- π est l'ordonnancement obtenu par la première phase,
- S_h^A est l'ensemble des travaux de l'agent A déjà ordonnancés, appartenant à la même clique maximale L_h que le travail j,
- S^A est la liste de tous les travaux de l'agent A déjà ordonnancés, appartenant aux mêmes cliques maximales que le travail j,
- $\mathcal{J}_h^A$ (resp. $\mathcal{J}^A$) est la liste des travaux de l'agent A appartenant à la clique L_h (resp. qui se chevauchent avec j).

Comme nous le constatons, deux étapes principales définissent l'algorithme 10. Pour un ordonnancement π et un travail j donnés :

1. On cherche la première machine pouvant exécuter le travail j. Pour chaque clique L_h, nous devons vérifier si la suppression des travaux de l'agent A permet de libérer les ressources nécessaires pour exécuter j. Si c'est le cas, l'ensemble S^A de tous les travaux de l'agent A déjà ordonnancés seront remis en cause, quelle que soit la clique L_h et quelle que soit la machine M_i. j est ainsi ordonnancé (cf. ligne 37 de l'algorithme 10). Sinon, le travail J_j est rejeté et π n'est pas modifié (cf. ligne 49).
2. Si j est ordonnancé, l'algorithme *tryToSchedule* est appelé pour chaque travail de l'agent A appartenant aux mêmes ensembles de cliques maximales du travail j, c'est-à-dire les travaux de l'ensemble $\mathcal{J}^A$. A l'issue de cette étape, certains travaux de l'agent A décidés initialement rejetés (resp. ordonnancés) peuvent être ordonnancés (resp. rejetés).

6.2.3 Analyse de complexité des heuristiques de liste

L'algorithme 9 peut s'exécuter en $O(mKH)$, où H est le nombre total de cliques maximales. En effet, à chaque fois qu'un travail j est ordonnancé, nous mettons à jour les disponibilités des ressources pour chaque clique maximale contenant j.

Concernant l'algorithme 10, le temps d'exécution est borné par $O(mKHn^A)$.

La liste des travaux $\mathcal{N}$ est parcourue pour ordonnancer les travaux un à un, triés selon une règle $\mathcal{R}$. Le tri des travaux se fait en $O(n\log(n))$. Cependant, la complexité de l'algorithme 8 est bornée par $O(mKHn^An^B)$ qui correspond à la phase de la construction d'une solution réalisable. Notons de même que le nombre de cliques maximales H est borné par $O(n)$. Ainsi, dans le pire des cas, la complexité de l'algorithme 8 est de $O(mnn^An^B)$.

Algorithm 10 *Schedule*(π, j)

Données :
$I = \{h : J_j \in L_h\}$
$R^h_{i,k}$ pour tout $h \in I$
$S^A_h = \{l : J_l \in \mathcal{N}^A \wedge J_l \in \pi \wedge J_l \in L_h \wedge h \in I\}$
$S^A = \bigcup_{h \in I} S^A_h$
$\mathcal{J}^A_h = \{l : J_l \in \mathcal{N}^A \wedge J_l \in L_h \wedge h \in I\}$
$\mathcal{J}^A = \bigcup_{h \in I} \mathcal{J}^A_h$
Traitement :
$H = |I|$
$i = 1$, $k = 1$, $h = 1$
$seq^B = Vrai$
while $(i \leq m)$ **do**
  while $(h \leq I[H])$ **do**
    while $(k \leq K \wedge seq^B = Vrai)$ **do**
      if $r_{jk} > R^h_{i,k} + \sum_{l \in S^A_h} r_{lk}$ **then**
        $seq^B = Faux$
        $h = I[H] + 1$
      end if
      $k = k + 1$
    end while
    $h = h + 1$
  end while
  if $seq^B = Faux$ **then**
    $i = i + 1$
  else
    $indice = i$
    $i = m + 1$
  end if
end while
if $seq^B = Vrai$ **then**
  for $i = 1$ à m **do**
    for $h = I[1]$ à $I[H]$ **do**
      for $k = 1$ à K **do**
        $R^h_{i,k} = R^h_{i,k} - \sum_{l \in S^A_h} r_{lk}$
        if $(i = indice)$ **then**
          $R^h_{i,k} = R^h_{i,k} + r_{jk}$
        end if
      end for
      $\pi = (\pi \cup \{J_j\}) \setminus S^A_h$
    end for
  end for
  while $\mathcal{J}^A \neq \emptyset$ **do**
    $seq = tryToSchedule(\pi, \mathcal{J}^A[1])$
    if $seq = Vrai$ **then**
      $Z^A = Z^A + w_{\mathcal{J}^A[1]}$
    end if
    $\mathcal{J}^A = \mathcal{J}^A \setminus \{\mathcal{J}[1]\}$
  end while
  return $Vrai$
end if
return $Faux$

6.3 Algorithme NSGA-II pour le calcul du front de Pareto approché

Pour déterminer le front de Pareto approché, nous proposons un algorithme génétique de tri non dominé appelé NSGA-II (*non-dominated sorting genetic algorithm*) (Deb et al., 2002). Cet algorithme est souvent utilisé pour résoudre les problèmes d'optimisation multiobjectifs. Basé sur un algorithme génétique, NSGA-II utilise une méthode de sélection par classement pour mettre en valeur les solutions actuelles non dominées et une méthode de "nichage" pour maintenir la diversité de la population. Nous recommandons au lecteur intéressé le tutoriel sur les méthodes d'optimisation multicritère utilisant des algorithmes génétiques présenté dans (Konak et al., 2006).

Dans cette section, nous nous focalisons au cas où les travaux ont un poids unitaire, i.e. $w_j = 1, j = 1, \ldots, n$ (Zahout et al., 2019).

6.3.1 Principaux concepts et mise en oeuvre

Dans la suite, nous appelons les solutions des individus. Un ensemble d'individus forme une population qui correspond à l'ensemble des solutions réalisables de notre problème. Chaque individu est représenté par un chromosome, la représentation de ce chromosome est le codage d'une solution.

6.3.1.1 Choix du codage

La valeur de la fonction objectif d'une solution du problème $Pm|CO, s_j, f_j, r_{jk}|\mathcal{P}(Z^A, Z^B)$ est complètement déduite à partir d'une affectation des travaux aux machines, puisqu'il s'agit de résoudre un problème d'affectation des travaux aux machines. Par conséquent, le codage choisi est basé sur un schéma dit *affectation-machine*. Un individu est donc un vecteur de taille n où chaque élément j stocke le numéro de la machine qui effectue le travail J_j. Si le travail j est rejeté, l'élément j stocke le numéro 0.

Reprenons l'exemple précédent 1.3 avec $m = 2$ machines, $n = 8$ travaux où $n_B = 4$ ($n_A = 4$). Nous rappelons que les quatre premiers travaux appartiennent à $\mathcal{N}^A$. La figure 6.1 montre le codage et le décodage d'une solution.

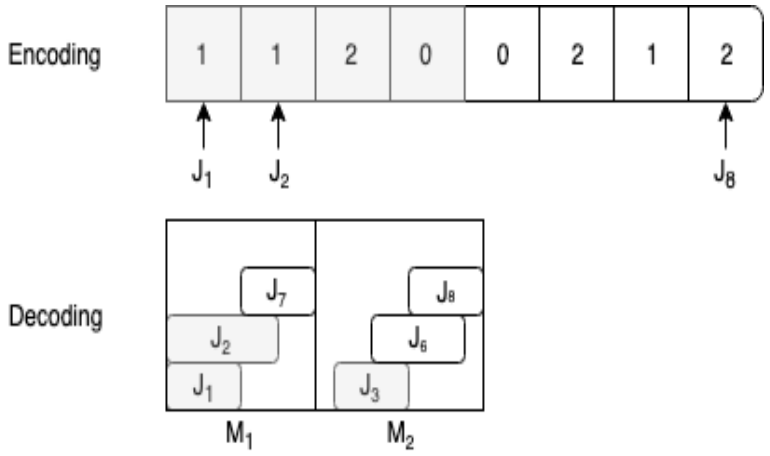
Le codage par permutation a été testé, mais ne conduit pas à de bons résultats, c'est pourquoi ce codage n'est pas détaillé ici.

Pour décoder un individu les travaux sont affectés aux machines tant que les contraintes de capacités sont respectées. Si un travail viole une contrainte de capacité, il est rejeté.

6.3.1.2 Génération de la population initiale

La population initiale peut-être générée aléatoirement, à base de recherche locale ou encore via l'utilisation d'heuristiques dédiées. Utiliser différentes méthodes pour la génération de la population initiale a pour but de diversifier les solutions initiales afin d'élargir la recherche dans l'espace des solutions faisables.

FIGURE 6.1 – Codage et décodage de la solution.



Pour résoudre notre problème d'ordonnancement avec deux agents en compétition, nous optons pour une génération de la population initiale selon deux approches. Une partie des individus est générée en utilisant les heuristiques de liste présentées dans les sections 6.2, l'autre partie est générée aléatoirement.

D'abord, nous initialisons la population initiale par les deux points extrêmes. Pour calculer ces deux points, nous faisons appel aux heuristiques introduites dans le Chapitre 4. Pour le premier point, les heuristiques considèrent seulement les travaux de l'agent A. A partir de la solution obtenue, nous cherchons à optimiser l'objectif de l'agent B en faisant appel à l'algorithme 9 sur l'ensemble des travaux $\mathcal{N}^B$ (cf. Algorithme 11). Pour le second point, nous opérons de la même façon : les heuristiques sont appelées sur l'ensemble $\mathcal{N}^B$, puis nous optimisons l'objectif de l'agent A en appliquant l'algorithme 9 sur l'ensemble des travaux $\mathcal{N}^A$ à partir de la solution retournée par les heuristiques (cf. Algorithme 12).

Algorithm 11 Point extrême maximisant l'objectif de l'agent A

1: π solution représentant un point du front de Pareto
2: $seq = Faux$, $Z^B = 0$
3: $\mathcal{N}^A, \mathcal{N}^B$
4: $\pi = PILOT(\mathcal{N}^A)$
5: **while** $(\mathcal{N}^B \neq \emptyset)$ **do**
6: $J_j = \mathcal{N}^B[1]$
7: $seq = tryToSchedule(\pi, J_j)$
8: **if** $(seq = Vrai)$ **then**
9: $Z^B = Z^B + w_j$
10: $\pi = \pi \cup \{J_j\}$
11: **end if**
12: $\mathcal{N}^B = \mathcal{N}^B \setminus \{J_j\}$
13: **end while**
14: **return** π

Algorithm 12 Point extrême maximisant l'objectif de l'agent B

1: π solution représentant un point du front de Pareto
2: $seq = Faux$, $Z^A = 0$
3: $\mathcal{N}^A$,$\mathcal{N}^B$
4: $\pi = PILOT(\mathcal{N}^B)$
5: **while** $(\mathcal{N}^A \neq \emptyset)$ **do**
6: $J_j = \mathcal{N}^A[1]$
7: $seq = tryToSchedule(\pi, J_j)$
8: **if** $(seq = Vrai)$ **then**
9: $Z^A = Z^A + w_j$
10: $\pi = \pi \cup \{J_j\}$
11: **end if**
12: $\mathcal{N}^A = \mathcal{N}^A \setminus \{J_j\}$
13: **end while**
14: **return** π

Pour compléter la partie de la population initiale générée par les heuristiques, nous devons déterminer une affectation aux machines. A partir d'une séquence, l'affectation aux machines est faite selon la stratégie **Str1** (**affectation machine par machine**) 4.2.2. Ainsi, nous obtenons cette première partie de population initiale que nous notons $\mathcal{P}_1^0$.

La population est complétée par des individus générés de manière aléatoire. La procédure est la suivante : considérons les $n = n_A + n_B$ travaux. Nous générons aléatoirement une séquence des travaux (un ordre complètement aléatoire de la totalité des travaux des deux agents confondus), nous appliquons par la suite la stratégie **Str1** pour affecter les travaux aux machines. Nous déduisons ainsi la valeur de l'objectif de chaque agent. Cette deuxième partie de population initiale est notée : $\mathcal{P}_2^0$

La population initiale obtenue est donc : $\mathcal{P}^0 = \mathcal{P}_1^0 \cup \mathcal{P}_2^0$.

Soit $N_{\mathcal{P}^0}$ la taille de la population $\mathcal{P}^0$. Cette taille est fixée expérimentalement à $N_{\mathcal{P}^0} = n + n/2$. Cette taille donne le meilleur compromis entre la qualité des solutions obtenues et le temps de calcul.

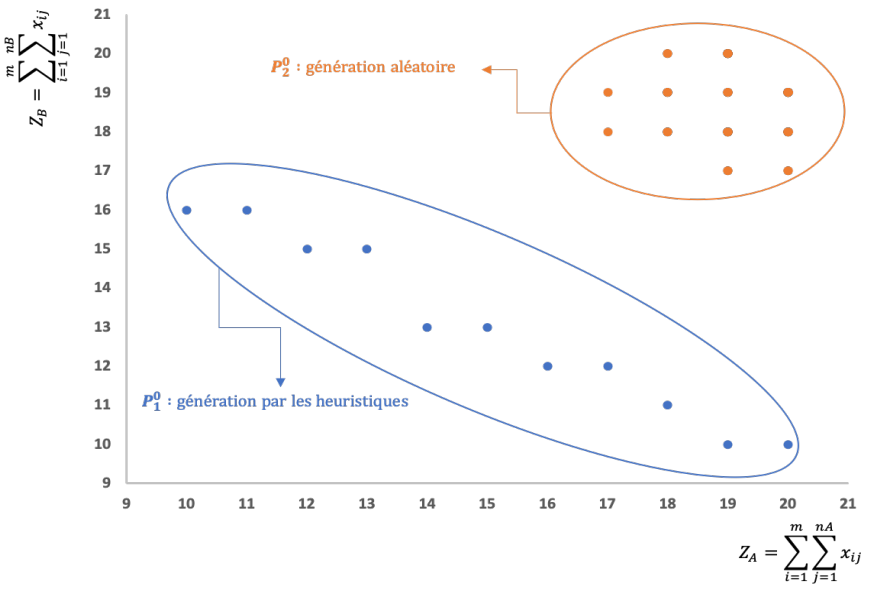
La figure 6.2 montre un nuage de points $\mathcal{P}^0$ résultant sur une première génération, en considérant une instance de taille $n_A = 20, n_B = 20$ et deux machines.

Selon la figure 6.2, le nuage de points $\mathcal{P}_2^0$ montre des solutions strictement dominées par les solutions appartenant à $\mathcal{P}_1^0$. Cependant, l'intérêt de $\mathcal{P}_2^0$ est l'introduction de la diversité à la population initiale.

6.3.1.3 Croisement

Conformément au mécanisme de croisement interchromosomique naturel, l'opérateur de croisement permet le brassage génétique des parents. Chaque individu possède deux gènes différents hérités des deux chromosomes des parents. Le patrimoine génétique du nouvel individu est composé aléatoirement d'une partie du patrimoine de chacun de ses deux parents pour permettre au matériel génétique de se répandre au sein de l'espèce.

FIGURE 6.2 – Instance avec $n_A = 20$, $n_B = 20$ et 2 machines : population initiale.



Cet opérateur permet de générer de nouveaux individus potentiellement meilleurs que les individus de la population courante. Le croisement que nous avons choisi opère sur deux individus d'une population et génèrent deux enfants. Il est appliqué avec une probabilité ρ_{cross} sur deux individus sélectionnés.

Soit C^q l'ensemble des individus générés par l'opérateur de croisement de taille N_{C^q}. Pour obtenir une solution enfant c_1, nous procédons comme suit :

1. De la population actuelle $\mathcal{P}^{q-1}$, sélectionner huit individus au hasard,

2. Parmi les huit individus, sélectionner deux individus parents λ_1 et λ_2 selon la sélection par tournoi 6.3.1.5,

3. Croiser λ_1 et λ_2 si $\sigma \leq \rho_{cross}$ avec σ une probabilité générée aléatoirement entre 0 et 1. La probabilité de croisement σ_{cross} est fixée à 0.8,

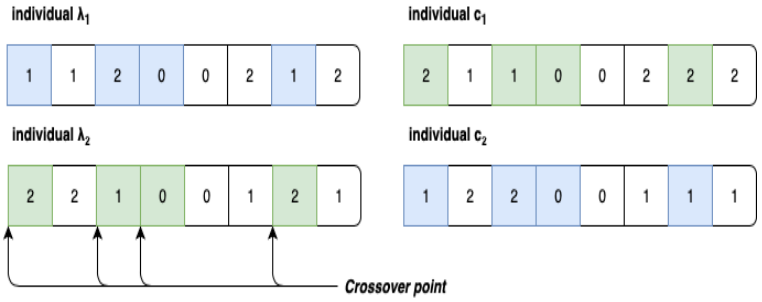
4. Ajouter à la population C^q les deux descendants c_1 et c_2.

Le premier (resp. le deuxième) enfant c_1 (resp. c_2) résultant du croisement des deux parents λ_1 et λ_2 est obtenu comme suit : Soit r le nombre de gènes que c_1 (resp. c_2) héritera de λ_2 (resp. λ_1), où r est généré aléatoirement dans $[1, \frac{4n-4}{n}]$; c_1 (resp. c_2) hérite donc des gènes de son premier (resp. deuxième) parent à l'exception des gènes r qui sont sélectionnés aléatoirement chez le deuxième (resp. premier) parent.

L'opérateur de croisement que nous proposons est original et est illustré par la figure 6.3 pour le cas de 8 travaux et 2 machines.

L'opérateur de croisement X-points a été testé, mais ne conduit pas à de bons résultats.

FIGURE 6.3 – Exemple de croisement avec 8 travaux.



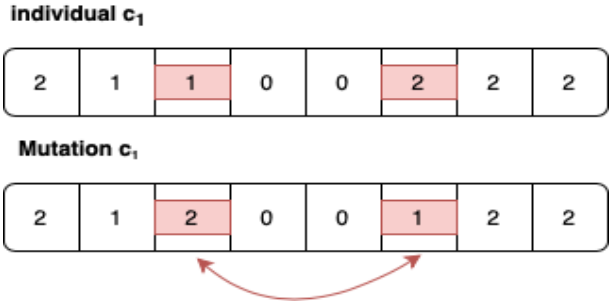
6.3.1.4 Mutation

L'opérateur de mutation introduit des modifications aléatoires dans un chromosome en modifiant un élément ou plusieurs. Le but de la mutation est de permettre la diversification de la recherche par l'exploration d'autres points de l'espace des solutions.

Soit M^q l'ensemble des individus générés par l'opérateur de mutation de taille N_{M^q}.

Une valeur ρ est générée aléatoirement entre 0 et 1. Si $\rho \leq \rho_{mut}$, l'enfant c_1 mute et l'enfant obtenu $c_1^{'}$ est ajouté à la population M^q. La probabilité de mutation ρ_{mut} est fixée à 0.8. Notre opérateur de mutation est défini comme suit : deux gènes (travaux) sont choisis au hasard et leurs caractéristiques sont échangées (voir la figure 6.4). Quant au choix des deux gènes, nous veillons à ce que leurs caractéristiques (i.e. indices des machines) soient différentes. Le nouvel individu est décodé par la procédure de décodage.

FIGURE 6.4 – Exemple d'opérateur de mutation avec 8 travaux.



6.3.1.5 Sélection

La population évolue d'une génération à l'autre tout en héritant des caractéristiques des populations précédentes. Les meilleurs individus résistent à la sélection naturelle, c'est-à-dire seuls ceux qui survivent et se reproduisent ont des descendants. Plusieurs mécanismes de sélection ont été introduits dans la littérature, qu'ils soient pour le choix des parents, pour le croisement ou pour la constitution de la future population. Le but est de transmettre au fil des générations des gènes avantageux et de faire disparaître les mauvais gènes. Nous avons opté pour la sélection dite "Sélection par rang" pour le choix de la population suivante

et la sélection dite "Sélection par tournoi" pour le choix des parents à croiser.

1. Sélection par rang : la sélection par rang trie d'abord la population par ordre non-croissant de la fitness (fonction objectif). Nous attribuons un rang à chaque chromosome en fonction de sa position dans le tri. Lors de cette étape, la population des individus est classée en rangs ou fronts successifs non dominés, appelés rang 1, rang 2, etc. Cela signifie que les solutions appartenant au rang rg sont non dominées entre elles et sont dominées par les solutions du rang $rg-1$. Pour trier les individus de la population, nous utilisons la procédure de classement proposée dans (**?**) qui est basée sur une recherche binaire.

   Sur la même idée que pour la "sélection par roulette", les rangs sont utilisés dans la définition d'une probabilité associée à chaque individu. Les meilleurs individus ont un rang égal à 1, et nous voulons leur attribuer la plus grande probabilité possible. L'idée est de promouvoir les meilleurs individus pour l'amélioration des prochaines populations.

   Soit Rg_λ le rang de l'individu λ et $Rg_{\max}$ le rang maximum. L'expression de la probabilité Pr_λ, de $\lambda \in \mathcal{P}$ est la suivante :

$$Pr_\lambda = \frac{Rg_{\max} - Rg_\lambda}{N_{\mathcal{P}} Rg_{max} - \sum_{p\in\mathcal{P}} Rg_p}$$

   avec $\sum_{\lambda\in\mathcal{P}} Pr_\lambda = 1$.

   Notons que seuls les Pop_{size} meilleurs individus différents sont alors sélectionnés pour constituer la nouvelle génération.

2. Sélection par tournoi : les meilleurs individus sont sélectionnés pour le croisement. Pour permettre le croisement des individus offrant une moins bonne qualité en termes de fitness, nous choisissons aléatoirement un sous-ensemble d'individus, puis les deux meilleurs parmi les individus retenus sont croisés.

6.3.1.6 Condition d'arrêt

L'algorithme s'arrête après un certain nombre d'itérations. Les résultats expérimentaux préliminaires nous ont conduit à fixer ce critère à $35n$ itérations.

6.4 Résultats expérimentaux

Dans cette section nous analysons les performances des méthodes approchées développées pour le cas multiagent. Pour tester et comparer nos méthodes, nous avons implémenté en $C++$ les heuristiques de liste présentés dans la section 6.2, l'algorithme NSGA-II de la section 6.3 a été implémenté en $Java$.

Tous nos tests ont été effectués sur une machine de 2,2 GHz Intel Core i7 et 16 Go de mémoire.

6.4.1 Heuristiques de liste

Dans cette section, l'ensemble des instances considérées est pris dans le jeu de données Type 2 (voir 4.1) avec n travaux au total.

Concernant les travaux de chaque agent, deux configurations sont considérées :

1. $(AgentA, AgentB) = (30\%, 70\%) \times n$
2. $(AgentA, AgentB) = (50\%, 50\%) \times n$

6.4.1.1 Calcul de points de Pareto

Afin d'évaluer les performances de nos heuristiques, pour chaque instance de données, nous cherchons trois points obtenus selon la valeur fixée de Q^A. Les trois valeurs de Q^A ont été fixées selon la règle suivante : $Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$ et $W^A = \sum_{j \in \mathcal{N}^A} w_j$.

1. (Agent A, Agent B)=(30%, 70%)
   $Q^A = 30\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)
   $Q^A = 50\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)
   $Q^A = 80\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)
2. (Agent A, Agent B)=(50%, 50%)
   $Q^A = 30\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)
   $Q^A = 50\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)
   $Q^A = 80\%W^A$ Tableaux (JS1-Agent, JS2-Agent, JS3-Agent, JS4-Agent, Best-JS-Agent)

6.4.2 Résultats $(n_A, n_B) = (30\%, 70\%)$

Le tableau 6.1 résume les performances des heuristiques de liste en variant le nombre de travaux de l'agent A (la colonne $30\%n$), le nombre de machines (la colonne m). Pour chaque n_A, une valeur Q_A est fixée (la colonne $Q_A = \alpha W^A$). Les colonnes CPU (s) représentent le temps de calcul moyen en secondes. Les colonnes GD représentent les moyennes de la distance euclidienne entre le point de Pareto obtenu par les heuristiques et le meilleur point de Pareto trouvé par les méthodes exactes dans le cas multiagent.

Dans le tableau 6.2, les colonnes $\%S$ (points de Pareto stricts) représentent le pourcentage des instances (10 instances par m) résolues à l'optimum par les heuristiques et les colonnes $\%WS$ (points de Pareto faibles) représentent le pourcentage des instances (10 instances par m) résolues à l'optimum par les heuristiques sur l'un des deux critères des agents (i.e un critère est résolu à l'optimum et l'autre non).

TABLE 6.1 – Distance euclidienne entre le point de Pareto approché et la solution trouvée par la méthode exacte

Type2			JS1-Agent		JS2-Agent		JS3-Agent		JS4-Agent		Best-JS-Agent	
$Q_A = \alpha W^A$	30%n	m	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)
$\alpha = 30\%$	20	4	10	0	15	0	24	0	10	0	10	0
		7	7	0	6	0	9	0	4	0	4	0
		10	19	0	20	0	29	0	18	0	18	0
		15	20	0	21	0	30	0	19	0	19	0
	60	4	150	0	142	0	145	0	148	0	142	0
		7	86	0	82	0	82	0	86	0	82	0
		10	50	0	46	0	46	0	50	0	46	0
		15	7	0	9	0	6	0	7	0	6	0
	100	4	311	0	318	0	327	0	319	0	311	0
		7	199	0	205	0	220	0	213	0	199	0
		10	129	0	136	0	151	0	146	0	129	0
		15	52	0	73	0	73	0	66	0	52	0
	150	4	557	0	560	0	560	0	583	0	557	0
		7	384	0	392	0	389	0	405	0	384	0
		10	278	0	289	0	285	0	299	0	278	0
		15	166	0	170	0	170	0	185	0	166	0
$\alpha = 50\%$	20	4	12	0	12	0	25	0	11	0	11	0
		7	1	0	1	0	10	0	3	0	1	0
		10	19	0	20	0	29	0	18	0	18	0
		15	20	0	21	0	30	0	19	0	19	0
	60	4	167	0	160	0	162	0	164	0	160	0
		7	95	0	90	0	90	0	95	0	90	0
		10	53	0	49	0	49	0	53	0	49	0
		15	7	0	9	0	6	0	7	0	6	0
	100	4	350	0	349	0	363	0	352	0	349	0
		7	221	0	230	0	243	0	238	0	221	0
		10	140	0	152	0	162	0	156	0	140	0
		15	57	0	65	0	76	0	71	0	57	0
	150	4	662	0	673	0	668	0	685	0	662	0
		7	441	0	446	0	441	0	461	0	441	0
		10	315	0	322	0	322	0	335	0	315	0
		15	186	0	192	0	192	0	210	0	186	0
$\alpha = 80\%$	20	4	22	0	23	0	40	0	21	0	21	0
		7	13	0	14	0	26	0	12	0	12	0
		10	19	0	20	0	34	0	18	0	18	0
		15	20	0	21	0	34	0	19	0	19	0
	60	4	249	0	240	0	244	0	246	0	240	0
		7	132	0	124	0	127	0	130	0	124	0
		10	73	0	60	0	65	0	70	0	60	0
		15	28	0	22	0	23	0	29	0	22	0
	100	4	-	-	-	-	-	-	-	-		
		7	343	0	348	0	364	0	358	0	343	0
		10	219	0	226	0	243	0	236	0	219	0
		15	99	0	106	0	125	0	116	0	99	0
	150	4	-	-	-	-	-	-	-	-		
		7	649	0	663	0	650	0	672	0	649	0
		10	467	0	479	0	474	0	488	0	467	0
		15	288	0	304	0	295	0	310	0	288	0

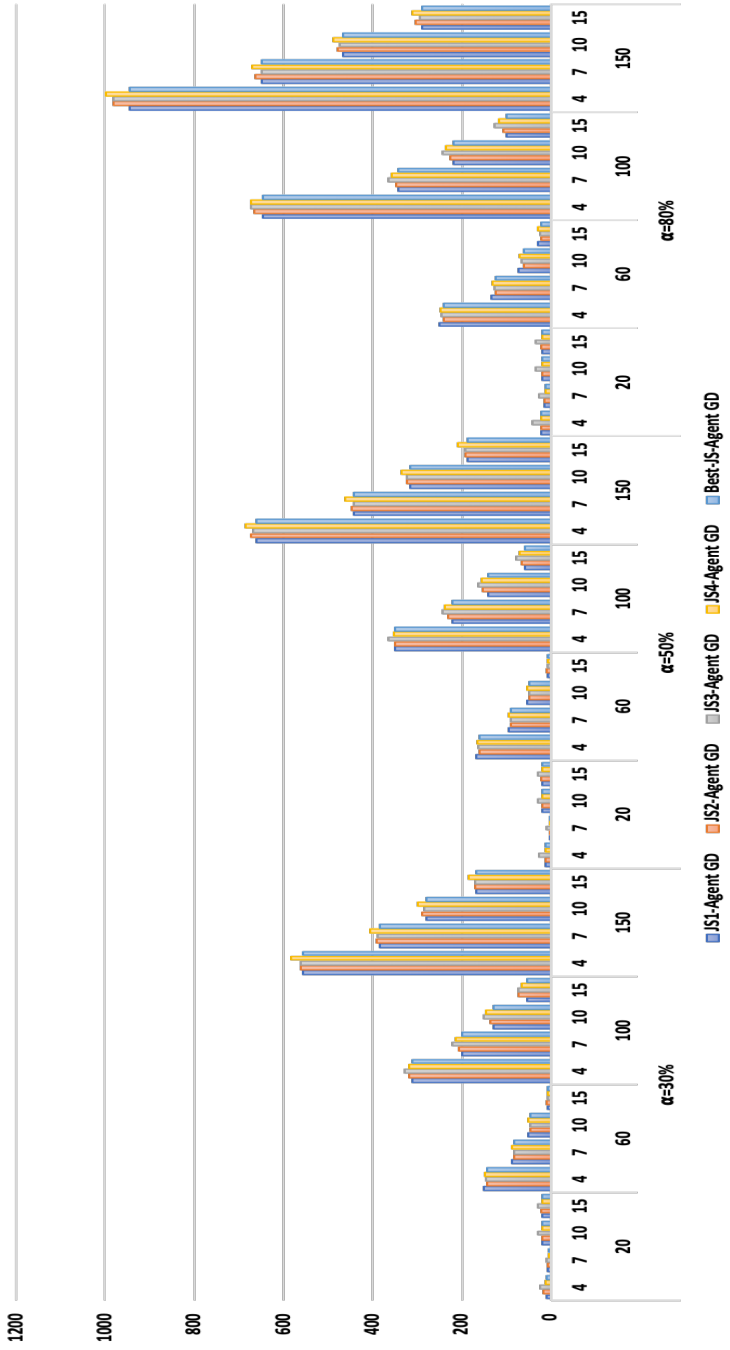


FIGURE 6.5 – Distance euclidienne entre le point de Pareto heuristique et la méthode exacte ($Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$)

Dans la figure 6.1, nous observons que $\forall \alpha \in \{30\%, 50\%, 80\%\}$ le temps de calcul moyen pour obtenir un point de Pareto par les heuristiques est nul. Nous observons aussi dans la figure 6.5 que la distance moyenne euclidienne (GD) augmente avec le nombre de travaux, avec un nombre de machines faible. Cependant, GD diminue avec l'augmentation de nombre de machines m quelle que soit la taille de l'instance n. Pour les instances $(n = \{20, 60\}, m = \{4, 7, 10, 15\})$, le GD calculé est relativement petit, i.e le point de Pareto trouvé par les heuristiques et le meilleur point trouvé par les méthodes exactes sont relativement adjacents. Pour les instances $(n = \{100, 150\}, m = \{4, 7, 10, 15\})$, le GD se détériore. La distance entre le point de Pareto trouvé par les heuristiques et le meilleur point trouvé par les méthodes exactes est importante. Globalement, les cinq heuristiques ont les mêmes performances : elles trouvent quasiment le même point de Pareto, sinon d'une déviation absolue de GD égale à 1.

Les pourcentages en **gras** dans le tableau 6.2 indiquent les cas où l'une des heuristiques trouve la meilleure performance. Nous observons que les heuristiques ne trouvent quasiment pas de point de Pareto strict ou faible, à quelques exceptions près, par exemple, pour les instances $\alpha = 50\%, n = 20, m = 7$ où l'heuristique $JS2 - Agent$ trouve 100% des points Pareto stricts et où les heuristiques $JS1 - Agent$ et $JS4 - Agent$ trouvent 100% de points de Pareto faibles.

6.4.3 Résultats $(n_A, n_B) = (50\%, 50\%)$

Il s'agit ici de résoudre par les heuristiques des instances encore plus difficiles que pour le cas $(n_A, n_B) = (30\%, 70\%)$. Les résultats montrent qu'aucune heuristique n'arrive à trouver une solution de Pareto faible. Lorsque $\alpha \in \{30\%, 50\%\}$, les distances euclidiennes calculées sont importantes et montrent que les solutions trouvées par les heuristiques sont loin d'être de bonnes solution de Pareto. Cependant, les résultats sont plutôt satisfaisants lorsque $\alpha = 80\%$.

TABLE 6.2 – Pourcentage de points de Pareto strictement et faiblement non-dominés obtenus par les heuristiques

Type2			JS1-Agent		JS2-Agent		JS3-Agent		JS4-Agent	
$Q_A = \alpha W^A$	30%n	m	%S	%WS	%S	%WS	%S	%WS	%S	%WS
$\alpha = 30\%$	20	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	**100%**	0%	**100%**	0%	0%	0%	0%
		15	0%	**100%**	0%	0%	0%	0%	0%	0%
	60	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
	100	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
	150	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
$\alpha = 50\%$	20	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	**100%**	**100%**	0%	0%	0%	0%	**100%**
		10	0%	**100%**	0%	**100%**	0%	0%	0%	0%
		15	0%	**100%**	0%	0%	0%	0%	0%	0%
	60	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	**100%**	0%	0%	0%	**100%**	0%	0%
	100	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
	150	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
$\alpha = 80\%$	20	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	**100%**	0%	**100%**	0%	0%	0%	0%
		15	0%	**100%**	0%	0%	0%	0%	0%	0%
	60	4	0%	0%	0%	0%	0%	0%	0%	0%
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	**100%**	0%	0%	0%	0%
	100	4	-	-	-	-	-	-	-	-
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%
	150	4	-	-	-	-	-	-	-	-
		7	0%	0%	0%	0%	0%	0%	0%	0%
		10	0%	0%	0%	0%	0%	0%	0%	0%
		15	0%	0%	0%	0%	0%	0%	0%	0%

TABLE 6.3 – Distance euclidienne entre le point de Pareto approché et la solution trouvée par la méthode exacte

Type2			JS1-Agent		JS2-Agent		JS3-Agent		JS4-Agent		Best-JS-Agent	
$Q_A = \alpha W^A$	50%n	m	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)	GD	CPU(s)
$\alpha = 30\%$	20	4	37	0	40	0	42	0	24	0	24	0
		7	32	0	35	0	35	0	19	0	19	0
		10	25	0	27	0	31	0	19	0	19	0
		15	26	0	28	0	30	0	18	0	18	0
	60	4	166	0	164	0	175	0	165	0	164	0
		7	153	0	146	0	162	0	151	0	146	0
		10	137	0	123	0	142	0	133	0	123	0
		15	64	0	53	0	73	0	65	0	53	0
	100	4	298	0	291	0	304	0	287	0	287	0
		7	296	0	287	0	307	0	290	0	287	0
		10	290	0	284	0	301	0	283	0	283	0
		15	209	0	199	0	227	0	203	0	199	0
	150	4	421	0	434	0	439	0	476	0	421	0
		7	450	0	444	0	445	0	483	0	444	0
		10	451	0	437	0	444	0	475	0	437	0
		15	418	0	410	0	409	0	448	0	409	0
$\alpha = 50\%$	20	4	30	0	33	0	36	0	18	0	18	0
		7	30	0	32	0	33	0	17	0	17	0
		10	25	0	27	0	31	0	19	0	19	0
		15	26	0	28	0	30	0	18	0	18	0
	60	4	117	0	117	0	126	0	116	0	116	0
		7	120	0	113	0	129	0	118	0	113	0
		10	124	0	110	0	130	0	119	0	110	0
		15	64	0	53	0	73	0	65	0	53	0
	100	4	203	0	214	0	221	0	212	0	203	0
		7	216	0	213	0	230	0	215	0	213	0
		10	224	0	218	0	234	0	217	0	217	0
		15	207	0	196	0	224	0	200	0	196	0
	150	4	283	0	329	0	329	0	333	0	283	0
		7	313	0	324	0	324	0	357	0	313	0
		10	320	0	308	0	314	0	344	0	308	0
		15	306	0	299	0	298	0	336	0	298	0
$\alpha = 80\%$	20	4	6	0	7	0	9	0	6	0	6	0
		7	17	0	19	0	20	0	7	0	7	0
		10	25	0	27	0	31	0	19	0	19	0
		15	26	0	28	0	30	0	18	0	18	0
	60	4	-	-	-	-	-	-	-	-	-	-
		7	37	0	36	0	51	0	43	0	36	0
		10	44	0	31	0	49	0	39	0	31	0
		15	60	0	50	0	69	0	61	0	50	0
	100	4	-	-	-	-	-	-	-	-	-	-
		7	70	0	75	0	93	0	78	0	70	0
		10	90	0	90	0	103	0	92	0	90	0
		15	91	0	80	0	109	0	84	0	80	0
	150	4	-	-	-	-	-	-	-	-	-	-
		7	-	-	-	-	-	-	-	-	-	-
		10	-	-	-	-	-	-	-	-	-	-
		15	117	0	114	0	117	0	139	0	114	0

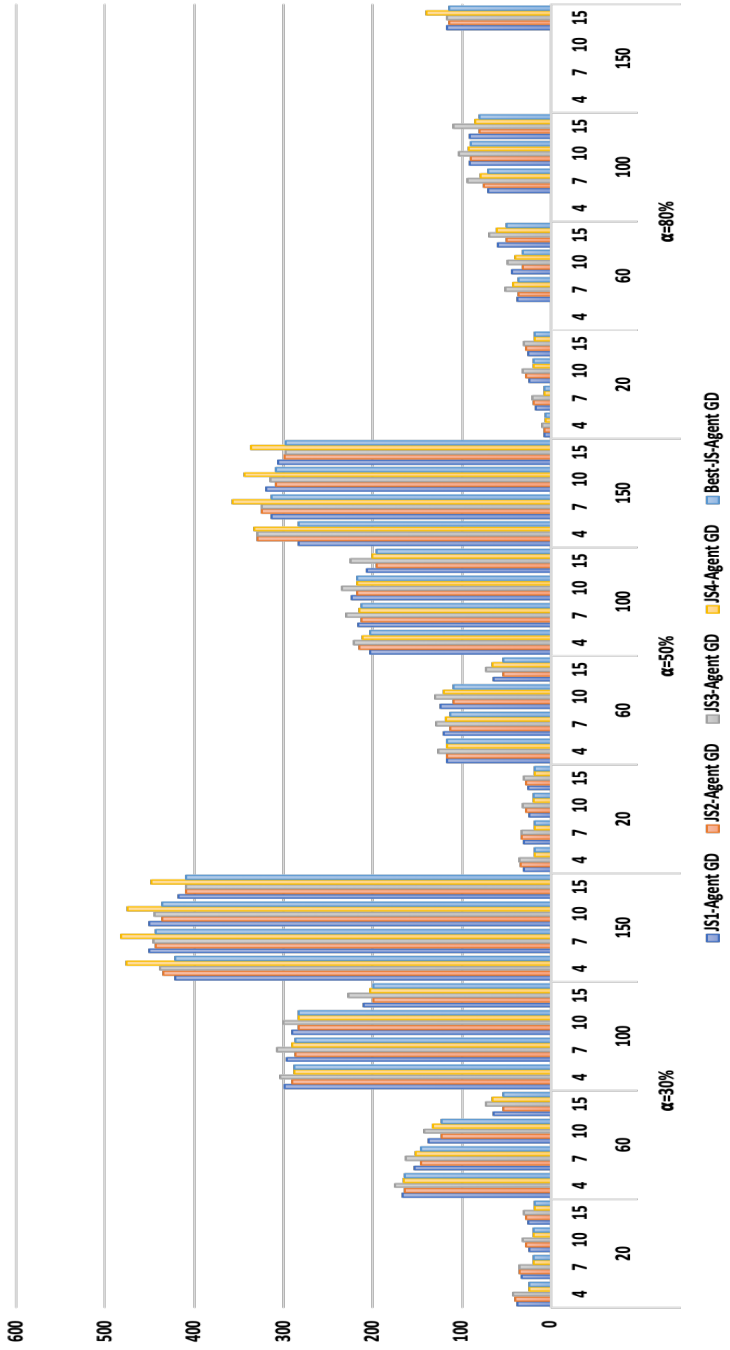


FIGURE 6.6 – Distance euclidienne entre le point de Pareto heuristique et la méthode exacte ($Q_A = \alpha W^A$ avec $\alpha \in \{30\%, 50\%, 80\%\}$)

6.4.4 NSGA-II

Afin d'analyser les performances de NSGA-II, nous avons généré 120 instances, avec $m \in \{2, 4, 6\}$ et $n \in \{20, 40, 60, 100\}$. 50% des travaux sont des jobs de l'agent A. 10 instances sont générées par valeur de n. En choisissant 50% des travaux pour chaque agent, on s'intéresse donc à la résolution des problèmes les plus difficiles.

Les dates de début des travaux ont été générées aléatoirement selon une distribution uniforme discrète entre 1 minute et 1438 minutes. De même, les dates de fin des travaux ont été générées aléatoirement entre $(s_j + 1)$ min et 1440 min. Trois types de ressources sont considérés. Sans perte de généralité, nous normalisons les unités de chaque ressource renouvelable à 1000. Ainsi, $R_k = 1000, k = 1, 2, 3$. $r_{j,k}$ est alors généré aléatoirement dans $[1, 1000]$, $j = 1, \ldots, n$ et $k = 1, 2, 3$.

A l'exception des poids unitaires, ces instances sont comparables aux instances de Type 2.

6.4.4.1 Mesures de performance

Différentes mesures de performances sont utilisées pour évaluer NSGA-II dans ce travail. Soit l'ensemble des solutions optimales de Pareto $S^* = \{a_1, \ldots, a_{|S^*|}\}$ et l'ensemble des solutions approchées $S = \{b_1, \ldots, b_{|S|}\}$. Nous classons les mesures de performance en trois catégories :

- *Mesures de cardinalité :* nous calculons la taille du front de Pareto exact $|S^*|$ et du front de Pareto approché $|S|$. Nous combinons ces mesures pour obtenir le pourcentage de solutions non dominées strictes générées par NSGA-II : $\%Sol = \frac{|S \cap S^*|}{|S^*|} \times 100$. Nous combinons également ces métriques pour obtenir le pourcentage de solutions non dominées faibles générées par les méthodes proposées, noté $\%wSol$.
- *Distance moyenne :* nous utilisons la moyenne de la distance euclidienne minimale : $GD = \frac{1}{|S|}(\sum_{i=1}^{|S|} d_i)$, avec d_i la distance euclidienne minimale entre l'élément $b_i \in S$ et l'élément le plus proche dans S^*.
- *Hypervolume :* lorsque la coque convexe du front de Pareto S est proche de la coque convexe du front de Pareto S^*, la distance moyenne peut être un mauvais indicateur de la qualité du front S. C'est pourquoi nous introduisons la métrique $HyperV$ pour hypervolume. Cette métrique calcule l'écart des surfaces entre le front S^* et le front S.

Les résultats sont présentés dans le tableau 6.4 et les figures 6.7 et 6.8. Pour chaque instance, nous comparons le front exact S^* (calculé par la méthode exacte) et le front de Pareto S (calculé par l'algorithme NSGA-II). La figure 6.7 indique le nombre de solutions de Pareto obtenues par la méthode exacte et l'algorithme NSGA-II en fonction du nombre de travaux et du nombre de machines. Ainsi, lorsque le nombre de travaux et de machines augmente, le nombre de solutions de Pareto augmente également.

Avec des instances de petite taille (20 travaux), nous remarquons que NSGA-II renvoie le front de Pareto exact (voir Fig. 6.8.(A)). Cette performance se détériore avec l'augmentation du nombre de travaux. Néanmoins, NSGA-II permet d'obtenir des solutions de Pareto faibles pour chaque instance (la Fig. 6.8.(B) montre cette tendance). Ces solutions faibles sont très proches des solutions optimales.

FIGURE 6.7 – Moyenne du nombre de solutions de Pareto.

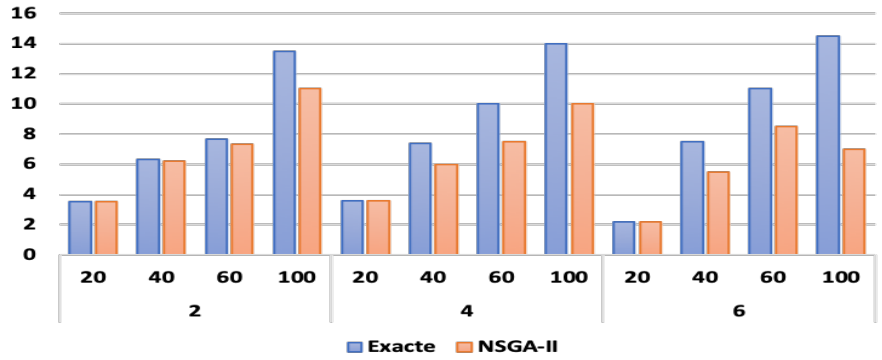
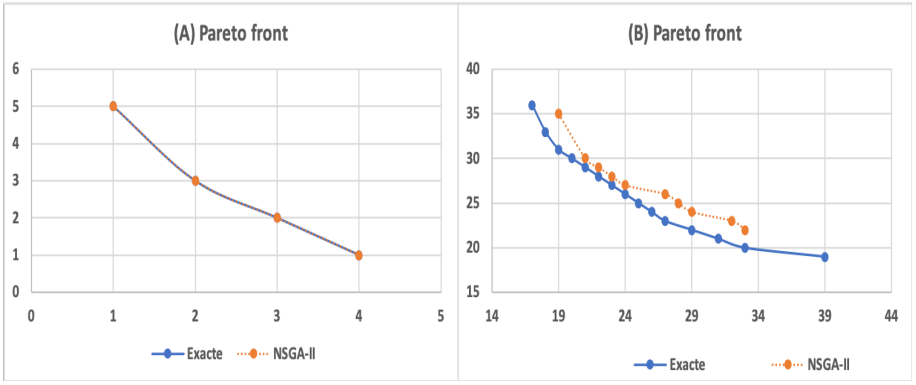


FIGURE 6.8 – Exemple de front de Pareto avec : (A) $n = 20$ et $m = 4$; (B) $n = 100$ et $m = 4$.



D'après le tableau 6.4, pour des instances avec $n = 100$ et $m = 6$, la méthode exacte a besoin de plus d'une heure pour obtenir l'ensemble des solutions de Pareto strictes. Néanmoins, elle reste assez efficace pour les petites instances (moins d'une minute pour les instances avec 40 jobs). En ce qui concerne la mesure de la distance moyenne euclidienne (GD), on peut conclure que la distance par rapport à l'ensemble des solutions optimales est très faible. Par exemple, la distance moyenne entre le front exact et le front approximatif obtenu par NSGA-II est inférieure à 2 pour les instances de 100 tâches et 6 machines. Pour les instances de 40 tâches et 6 machines, NSGA-II ne trouve pas de solution de Pareto stricte mais la plupart des solutions obtenues sont des solutions de Pareto faibles (entre 96% et 100%). Ces conclusions sont confirmées par la métrique Hypervolume. Nous pouvons conclure que les fronts de Pareto approchés obtenus par NSGA-II sont proches des fronts de Pareto exacts.

6.5 Conclusions du chapitre

Dans ce chapitre, nous nous sommes intéressés à la résolution approchée du problème d'ordonnancement multiagent à intervalles fixes. Nous adaptons les quatre heuristiques de

TABLE 6.4 – Résultats avec $n_A = 50\%n$

m	n	$Exacte$		$NSGA-II$					
		CPU_s	$\|S^*\|$	CPU_s	$\|S\|$	GD	$HyperV$	$\%S$	$\%wS$
2	20	2,29	3,50	1,50	3,50	0,00	0,00	100,00	0,00
	40	8,22	6,30	5,31	6,20	0,00	1,40	98,57	1,43
	60	15,08	7,67	10,91	7,33	0,30	16,00	71,30	25,00
	100	56,36	13,50	24,97	11,00	0,33	106,50	59,62	40,38
4	20	4,74	3,60	2,37	3,60	0,00	0,00	100,00	0,00
	40	21,38	7,40	8,99	6,00	0,35	5,20	55,26	43,08
	60	52,15	10,00	19,48	7,50	0,43	24,50	55,00	45,00
	100	205,23	14,00	140,04	10,00	1,53	31,00	0,00	100,00
6	20	5,35	2,20	2,80	2,20	0,00	0,00	100,00	0,00
	40	35,55	7,50	11,41	5,50	0,46	4,20	40,00	58,33
	60	95,83	11,00	65,99	8,50	1,12	17,50	4,55	95,45
	100	4510,21	14,50	213,03	7,00	1,98	66,00	0,00	100,00

listes proposées pour la résolution du cas monocritère. Il s'agit des algorithmes gloutons basés sur des listes de priorités ou règles de tri. Pour calculer le front de Pareto approché, dans le cas où tous les travaux ont un poids unitaire, nous proposons un algorithme génétique NSGA-II.

Afin d'analyser les performances des algorithmes gloutons et de NSGA-II, nous avons repris les jeux de données Type2. Considérons les instances Type2 lesquelles les méthodes exactes ont pu résoudre à l'optimum certaines instances ou du moins retourner des bornes supérieures correctes. Les études expérimentales menées nous ont permis de conclure sur les performances des heuristiques et de la métaheuristique proposées dans ce chapitre. Ces études révèlent que les algorithmes de listes n'offrent pas de bonnes performances en général et que les déviations mesurées selon la distance euclidienne pour chaque instance résolue sont importantes.

L'algorithme NSGA-II quant à lui est très efficace puisqu'il offre des gaps moyens plus faibles par rapport aux solutions optimales au sens de Pareto lorsque les travaux ont un poids unitaire.

Concernant les heuristiques gloutonnes, il serait intéressant de les reprendre pour compiler le front de Pareto et ainsi calculer les déviations des points trouvés par rapport aux optima de Pareto les plus proches.

Pour NSGA-II, les premiers résultats expérimentaux dans le cas où chaque travail a un poids w_j ne sont pas concluants. En effet, le codage choisi est mieux adapté aux problèmes résolus dans ce chapitre, i.e au cas des poids unitaires. C'est pourquoi nous pensons qu'il serait intéressant de proposer un autre codage pour le cas général et ainsi appliquer l'opérateur de croisement original proposé dans ce chapitre.

D'autres métaheuristiques telles que *Iterated Local Search* ILS et $GRASP$ méritent également d'être développées.

Conclusion générale et perspectives

La pertinence des modèles est une question qui se pose toujours aux ordonnanceurs soucieux de modéliser finement le problème d'ordonnancement présent dans leurs systèmes, avant de passer à la phase de résolution pour proposer des méthodes efficaces et aider ainsi le décideur à faire son choix.

Dans ce manuscrit, nous avons présenté une synthèse de nos principaux travaux de thèse de doctorat qui porte sur l'ordonnancement des travaux multiressources à intervalles fixes sur machines parallèles identiques. Les machines sont dotées de différents types de ressources renouvelables, présents en quantités limitées, nécessaires pour traiter un travail soumis par un utilisateur. Cette limitation du système contraint l'ordonnanceur à faire un choix sur l'acceptation des travaux à exécuter et le refus des travaux à rejeter. Son objectif est de réduire l'impact négatif des décisions de rejet.

Une des applications de ce travail pour laquelle nous nous sommes intéressés est l'ordonnancement des travaux dans des systèmes distribués, tels que le Cloud. Dans cette étude, nous nous sommes restreints à la résolution d'un problème d'ordonnancement complètement maîtrisé où toutes les données du problèmes sont connues : les travaux sont disponibles au début de l'ordonnancement, les dates de début et de fin des travaux sont données et donc les durées opératoires sont connues, les travaux exécutés doivent l'être sans interruption, les travaux sont pondérés et donc n'ont pas tous la même importance et les ressources renouvelables sont disponibles à tout instant de la planification. La difficulté du problème réside dans la détermination de l'ensemble des travaux à exécuter et dans l'affectation des ressources nécessaires pour leur traitement. Dans cette étude nous avons développé des algorithmes d'ordonnancement exacts et approchés pour maximiser le profit de l'administrateur du système, i.e. minimiser le coût total de rejet des travaux, dans une approche monocritère (la qualité de l'ordonnancement est mesurée par une seule fonction objectif appliquée sur l'ensemble des travaux). Par la suite, nous avons généralisé ces résultats et introduit de nouveaux algorithmes de résolution exacts et approchés lorsqu'il s'agit de chercher des solutions de meilleurs compromis pour le cas multicritère. Le problème d'ordonnancement multicritère considéré dans notre étude est différent de la définition classique puisque l'ordonnancement de chaque sous-ensemble disjoint de travaux est évalué en fonction d'un critère propre à lui, mais les travaux sont tous en concurrence pour l'utilisation des ressources. Il s'agit d'un problème d'ordonnancement où un nouveau type de compromis doit être obtenu. Nous parlons d'ordonnancement multiagent, aussi appelé ordonnancement avec agents en compétition. Dans ces problèmes d'ordonnancement, il y a plusieurs agents auxquels sont rattachés des sous-ensembles disjoints de travaux. Chaque agent a sa propre mesure de performance qui dépend uniquement de l'ordonnan-

cement de ses travaux. Il s'agit d'un nouveau modèle d'ordonnancement multicritère des travaux multiressources à intervalles fixes, non abordé jusqu'à présent dans la littérature. De nouveaux résultats de complexité sont proposés avec identification de cas particulier polynomiaux pour lesquels nous avons développé des méthodes exactes, principalement de type programmation dynamique.

Pour les systèmes distribués, il s'agit de résoudre des problèmes d'ordonnancement en ligne où le challenge demeure toujours dans le développement d'heuristiques extrêmement rapides mais pas au détriment de la qualité, permettant ainsi le passage à l'échelle. Analyser théoriquement les performances de ces heuristiques n'est pas facile. Un des intérêts de nos travaux est d'offrir un moyen de le faire a posteriori.

Comme cela a été signalé dans le deuxième chapitre "Ordonnancement : Travaux connexes", au delà du contexte de ce travail de thèse de doctorat, la résolution des problèmes d'ordonnancement des travaux multiressources à intervalles fixes demeure d'actualité et peut être adaptée à une grande diversité d'applications, comme dans certains systèmes de production de biens, par exemple.

Les perspectives de recherche qui découlent de ce travail sont très nombreuses. A court et à moyen terme, les pistes que nous envisagerions de suivre portent en particulier sur le scénario "agents en compétition". Nos perspectives sont de deux natures : théoriques et appliquées.

Les discussions et perspectives exposées tout au long de ce manuscrit étaient orientées d'un point de vue technique, précisant le travail restant à faire pour le cas monocritère et multiagent.

Différentes décompositions du modèle PLNE-3 pourraient être conçues, comme cela est suggéré dans (Caprara et al., 2013) pour le problème d'allocation de ressources. Nous avons testé une décomposition alternative de PLNE-3 où le sous-problème (*pricing problem*) s'avère être un problème de sac-à-dos multidimensionnel. Les résultats expérimentaux indiquent que cette décomposition n'est pas compétitive avec la méthode proposée dans (Caprara et al., 2013). D'autres méthodes de décomposition sont à explorer.

Pour le Branch& Price, développer des bornes primales, proposer des conditions de dominances ou encore développer des méthodes exactes ou approchées pour résoudre le *pricing problem* sont des pistes qui devraient permettre d'améliorer les performances de cette méthode exacte. En effet, le *pricing problem* n'est autre qu'un problème de sac-à-dos temporel (TKP). Nous pensons en premier lieu utiliser la PPC puis nous inspirer des résultats de la littérature en nous focalisant principalement sur les approches de type programmation dynamique. Nous pensons aussi proposer une méthode de type SAT : à une itération donnée, SAT est appellé pour déterminer parmi l'ensemble des colonnes celles qui seront retenues dans le problème maître restreint (avant la résolution du *pricing problem*) afin d'éviter l'infaisabilité. Il s'agit d'une piste de recherche initiée dans le cadre d'une collaboration avec le Dr. Valentin Montmirail [1] lors de son court séjour dans notre laboratoire en 2019. Notons qu'après avoir modélisé/réécrit nos contraintes dans le bon format, nous avons utilisé CEGAR (Lagniez et al., 2017) [2] comme boite noire développée par le labora-

1. Valentin Montmirail ancien doctorant du CRIL, ingénieur-chercheur chez Schneider Electric Nice, Provence-Alpes-Côte d'Azur, France

2. Solveur de modèles SAT

toire CRIL. Les premiers résultats obtenus pour le cas monocritère ne sont pas compétitifs avec le Branch& Price.

Même si elles sont efficaces pour résoudre le cas monocritère, les heuristiques de type *List Scheduling*, montrent leurs limites pour la résolution du problème d'ordonnancement multiagent avec l'approche ε-contrainte. Pour une borne $Q_A \in [0, W^A]$ donnée, il peut être intéressant d'utiliser les heuristiques proposées en ajoutant une recherche bidirectionnelle autour de la valeur Q_A maximisant ainsi le critère du second agent, moyennant un sur-coût de complexité raisonnable de l'ordre de $O(\log(W^A)$, où W^A est le poids total des travaux.

De même, NSGA-II n'a pas permis d'avoir des résultats concluants pour l'ordonnancement des travaux pondérés dans le cas de l'ordonnancement multiagent. Il serait donc intéressant de chercher un autre codage ou encore de penser à une hybridation de cette méthode avec une heuristique basée sur la recherche de flot dans un réseau, en se basant sur l'ensemble des cliques maximales. En effet, lorsqu'un seul type de ressources est considéré, la principale difficulté est la détermination de l'ensemble des travaux à ordonnancer. Aussi, les règles d'affectation des travaux aux machines à partir de la bonne permutation des travaux permettent d'obtenir une solution optimale. C'est pourquoi nous pensons à une méthode telle que Iterated Local Search (ILS) ou GRASP, qui peuvent être efficaces et moins gourmandes en temps d'exécution.

Pour le modèle étudié, il serait intéressant de considérer le cas où, au lieu que chaque client n'ait qu'une seule date de début et de fin fixe pour chaque application, un ensemble discret d'heures de début et de fin d'exécution de chaque application soit possible. Dans ce cas, selon les règles de tri, les algorithmes de listes proposés, peuvent être adaptés.

De plus, la contrainte sur la ressource énergétique a été considérée implicitement dans notre étude. Il serait intéressant de l'intégrer de façon explicite à notre problème d'ordonnancement des travaux multiressources à intervalles fixes puisqu'il s'agit d'une ressource partagée par tout le système. En effet, un des défis majeurs auxquels font face les fournisseurs de service cloud est la réduction du coût d'exploitation énergétique, ou de respecter au moins le *Power capping*. C'est un travail en cours et un programme linéaire en nombres entiers a été développé.

Dans cette thèse, nous nous sommes concentrés sur l'ordonnancement statique (hors ligne) des machines virtuelles. Il serait intéressant de tester les performances des méthodes approchées proposées dans le cas en ligne avec préemption des applications. Des critères d'optimisation tels que la minimisation des encours, l'équilibrage de charge ou encore la satisfaction du client (respect du contrat de service) peuvent être considérés.

Il serait intéressant aussi de prendre en compte l'aspect dynamique de l'environnement du cloud. Le problème de la reconfiguration dynamique d'applications arbitraires (demande des clients sporadique) impliquant éventuellement la migration des machines virtuelles dans le cloud devraient nécessiter des aménagements à nos algorithmes. La prise en compte de la dynamicité dans les modèles nécessite d'utiliser, en complément, les modèles stochastique, par exemple.

Dans le chapitre 1, nous avons identifié trois niveaux d'ordonnancement dans le cloud computing, une perspective plus générale liée à l'ensemble des contributions de cette thèse peut être proposée. Elle consiste à combiner tous les différents niveaux d'ordonnancement

en tant que parties d'un même flux d'ordonnancement. Le résultat sera une approche générale de l'ordonnancement, quelle que soit la granularité du service. On peut voir qu'un client peut envoyer une demande au cloud. En fonction des besoins, un méta-ordonnanceur au premier niveau (niveau tâche) commencera par trouver le cloud (data center) approprié parmi les clouds pour répondre à la demande du client ou bien elle sera directement transmise à l'ordonnanceur au deuxième niveau (niveau service). L'étape suivante sera l'affectation de la demande du client à la combinaison judicieuse des instances de machine virtuelle (VM) par l'ordonnanceur du deuxième niveau. Au dernier niveau, celui des VM, l'ordonnanceur du gestionnaire de cloud sera chargé de trouver pour chaque VM proposée par le niveau des tâches, la meilleure machine physique où elle pourrait être exécutée efficacement.

Bibliographie

A. Agnetis, P. B. Mirchandani, D. Pacciarelli, and A. Pacifici. Nondominated schedules for a job-shop with two competing users. *Computational & Mathematical Organization Theory*, 6(2) :191–217, 2000.

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

A. Agnetis, G. De Pascale, and M. Pranzo. Computing the nash solution for scheduling bargaining problems. *International Journal of Operational Research*, 6(1) :54–69, 2009b.

A. Agnetis, J.-C. Billaut, S. Gawiejnowicz, D. Pacciarelli, and A. Soukhal. Multiagent scheduling. *Berlin Heidelberg : Springer Berlin Heidelberg. doi*, 10(1007) :978–3, 2014.

C. Alonso, F. Caro, and J. Monta.a. A flipping local search genetic algorithm for the multidimensional 0-1 knapsack problem. *Lecture Notes in Computer Science*, 4177 : 21–30, 2006.

A. Amazon2012. Amazon web services llc. amazon elastic compute cloud (amazon ec2)`http://aws.amazon.com/ec2/`, 2012.

M. Andreolini, S. Casolari, M. Colajanni, and M. Messori. Dynamic load management of virtual machines in cloud architectures. In *International Conference on Cloud Computing*, pages 201–214. Springer, 2009.

E. Angelelli and C. Filippi. On the complexity of interval scheduling with a resource constraint. *Theoretical Computer Science*, 412(29) :3650–3657, 2011.

E. Angelelli, N. Bianchessi, and C. Filippi. Optimal interval scheduling with a resource constraint. *Computers & Operations Research*, 51 :268–281, 2014.

J. Anthonisse and J. K. Lenstra. Operational operations research at the mathematical centre. *European Journal of Operational Research*, 15(3) :293–296, 1984.

E. M. Arkin and E. B. Silverberg. Scheduling jobs with fixed start and end times. *Discrete Applied Mathematics*, 18(1) :1–8, 1987.

C. Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Oper. Res. Lett.*, 45(2) :154–159, 2017.

C. Artigues and C. Briand. The resource-constrained activity insertion problem with minimum and maximum time lags. *J. Sched.*, 12(5) :447–460, 2009.

C. Artigues, S. Demassey, and E. Neron. *Resource-Constrained Project Scheduling : Models, Algorithms, Extensions and Applications.* ISTE/Wiley, 2013.

A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. *SIAM journal on computing*, 28(5) :1806–1828, 1999.

A. Bar-Noy, S. Guha, J. Naor, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing*, 31(2) :331–352, 2001.

B. Baremetal2014. `http://baremetalcloud.com/index.php/en/`, 2014.

J. Beasley and B. Cao. A dynamic programming based algorithm for the crew scheduling problem. *Computers & Operations Research*, 25(7-8) :567–582, 1998.

O. Beaumont, L. Canon, L. Eyraud-Dubois, G. Lucarelli, L. Marchal, C. Mommessin, B. Simon, and D. Trystram. Scheduling on two types of resources : A survey. *ACM Comput. Surv.*, 53(3) :56 :1–56 :36, 2020.

P. Berman and B. DasGupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, 4(3) :307–323, 2000.

K. Bessai, S. Youcef, A. Oulamara, C. Godart, and S. Nurcan. Scheduling strategies for business process applications in cloud environments. *Int. J. Grid High Perform. Comput.*, 5(4) :65–78, 2013.

R. Bhatia, J. Chuzhoy, A. Freund, and J. S. Naor. Algorithmic aspects of bandwidth trading. In *International Colloquium on Automata, Languages, and Programming*, pages 751–766. Springer, 2003.

R. Bleuse, S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. Scheduling independent tasks on multi-cores with GPU accelerators. *Concurr. Comput. Pract. Exp.*, 27 (6) :1625–1638, 2015.

R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mounie, and D. Trystram. Scheduling independent moldable tasks on multi-cores with gpus. *IEEE Trans. Parallel Distributed Syst.*, 28(9) :2689–2702, 2017.

K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms. *Journal of computer and system sciences*, 13(3) :335–379, 1976.

D. Borgetto, H. Casanova, G. Da Costa, and J.-M. Pierson. Energy-aware service allocation. *Future Generation Computer Systems*, 28(5) :769–779, 2012a.

D. Borgetto, M. Maurer, G. Da-Costa, J.-M. Pierson, and I. Brandic. Energy-efficient and sla-aware management of iaas clouds. In *2012 Third International Conference on Future Systems : Where Energy, Computing and Communication Meet (e-Energy)*, pages 1–10. IEEE, 2012b.

K. I. Bouzina and H. Emmons. Interval scheduling on identical machines. *Journal of Global Optimization*, 9(3-4) :379–393, 1996.

M. Brehob, S. Wagner, E. Torng, and R. Enbody. Optimal replacement is np-hard for nonstandard caches. *IEEE Transactions on computers*, 53(1) :73–76, 2004.

D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture : Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20 (6) :26–44, 2000.

P. Brucker. *Scheduling algorithms (4. ed.)*. Springer, 2004. ISBN 978-3-540-20524-1.

P. Brucker and L. Nordmann. Thek-track assignment problem. *Computing*, 52(2) :97–122, 1994.

J. Burge, P. Ranganathan, and J. L. Wiener. Cost-aware scheduling for heterogeneous enterprise machines (cash'em). In *2007 IEEE international conference on cluster computing*, pages 481–487. IEEE, 2007.

R. H. Campbell, I. Gupta, M. T. Heath, S. Y. Ko, M. Kozuch, M. Kunze, T. T. Kwan, K. Lai, H. Y. Lee, M. Lyons, et al. Open cirrus^{TM} cloud computing testbed : Federated data centers for open source systems and services research. *HotCloud*, 9 :1–1, 2009.

R. Canetti and S. Irani. Bounding the power of preemption in randomized scheduling. *SIAM Journal on Computing*, 27(4) :993–1015, 1998.

L. Canon, L. Marchal, and F. Vivien. Low-cost approximation algorithms for scheduling independent tasks on hybrid platforms. In F. F. Rivera, T. F. Pena, and J. C. Cabaleiro, editors, *Euro-Par 2017 : Parallel Processing - 23rd International Conference on Parallel and Distributed Computing, Santiago de Compostela, Spain, August 28 - September 1, 2017, Proceedings*, volume 10417 of *Lecture Notes in Computer Science*, pages 232–244. Springer, 2017.

L. Canon, L. Marchal, B. Simon, and F. Vivien. Online scheduling of task graphs on heterogeneous platforms. *IEEE Trans. Parallel Distributed Syst.*, 31(3) :721–732, 2020.

A. Caprara, F. Furini, and E. Malaguti. Uncommon dantzig-wolfe reformulation for the temporal knapsack problem. *INFORMS Journal on Computing*, 25(3) :560–571, 2013.

M. C. Carlisle and E. L. Lloyd. On the k-coloring of intervals. *Discrete Applied Mathematics*, 59(3) :225–235, 1995.

M. W. Carter and C. A. Tovey. When is the classroom assignment problem hard ? *Operations Research*, 40(1-supplement-1) :S28–S39, 1992.

S. Chaisiri, B.-S. Lee, and D. Niyato. Optimal virtual machine placement across multiple cloud providers. In *2009 IEEE Asia-Pacific Services Computing Conference (APSCC)*, pages 103–110. IEEE, 2009.

J. Chen, C. Wang, B. B. Zhou, L. Sun, Y. C. Lee, and A. Y. Zomaya. Tradeoffs between profit and customer satisfaction for service provisioning in the cloud. In *Proceedings of the 20th international symposium on High performance distributed computing*, pages 229–238, 2011.

L. Chen, D. Ye, and G. Zhangm. Online scheduling of mixed cpu-gpu jobs. *Int. J. Found. Comput. Sci.*, 25(6) :745–761, 2014.

Z.-Z. Chen, G. Lin, R. Rizzi, J. Wen, D. Xu, Y. Xu, and T. Jiang. More reliable protein nmr peak assignment via improved 2-interval scheduling. *Journal of Computational Biology*, 12(2) :129–146, 2005.

T. E. Cheng, C. Ng, and J. Yuan. Multi-agent scheduling on a single machine to minimize total weighted number of tardy jobs. *Theoretical computer science*, 2006.

B. N. Chun and D. E. Culler. User-centric performance analysis of market-based cluster batch schedulers. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'02)*, pages 30–30. IEEE, 2002.

J. Chuzhoy, S. Guha, S. Khanna, and J. S. Naor. Machine minimization for scheduling jobs with interval constraints. In *45th annual IEEE symposium on foundations of computer science*, pages 81–90. IEEE, 2004.

Y. Coady, O. Hohlfeld, J. Kempf, R. McGeer, and S. Schmid. Distributed cloud computing : Applications, status quo, and challenges. *Computer Communication Review*, 45(2) :38–43, 2015. doi : 10.1145/2766330.2766337. URL `https://doi.org/10.1145/2766330.2766337`.

E. Coffman, M. Garey, and D. Johnson. Approximation algoritms for bin-packing – an updated survey. *in Algorithm Design for Computer Design, G. Ausiello, M. Lucertini, and P. Serafini (eds.), Springer-Verlag*, pages 49–106, 1984.

F. D. Croce and A. Grosso. Improved core problem based heuristics for the 0/1 multi-dimensional knapsack problem. *Comput. Oper. Res.*, 39(1) :27–31, 2012.

F. D. Croce and R. Scatamacchia. An exact approach for the bilevel knapsack problem with interdiction constraints and extensions. *Math. Program.*, 183(1) :249–281, 2020.

F. D. Croce, U. Pferschy, and R. Scatamacchia. New exact approaches and approximation results for the penalized knapsack problem. *Discret. Appl. Math.*, 253 :122–135, 2019.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm : Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2) :182–197, 2002.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

C. Dhaenens. *Optimisation Combinatoire Multi-Objectif : Apport des méthodes coopératives et contribution à l'extraction de connaissances.* PhD thesis, Université des Sciences et Technologie de Lille-Lille I, 2005.

V. R. Dondeti and H. Emmons. Algorithms for preemptive scheduling of different classes of processors to do jobs with fixed times. *European journal of operational research*, 70 (3) :316–326, 1993.

M. Drozdowski. *Scheduling for Parallel Processing*. Springer, 2009.

M. Dyer, W. Riha, and J. Walker. A hybrid dynamic programming/ branch-and-bound algorithm for the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics*, 58 :43–54, 1995.

E. Elmroth, F. G. Marquez, D. Henriksson, and D. P. Ferrera. Accounting and billing for federated cloud infrastructures. In *2009 Eighth International Conference on Grid and Cooperative Computing*, pages 268–275. IEEE, 2009.

E. Eucalyptus2014. Inc. eucalyptus systems. eucalyptus cloud `http://www.eucalyptus.com/eucalyptus-clou`, 2014.

U. Faigle and W. M. Nawijn. Note on scheduling intervals on-line. *Discrete Applied Mathematics*, 58(1) :13–17, 1995.

U. Faigle, W. Kern, and W. M. Nawijn. A greedy on-line algorithm for thek-track assignment problem. *Journal of Algorithms*, 31(1) :196–210, 1999.

E. Feller, L. Rilling, and C. Morin. Snooze : A scalable and autonomic virtual machine management framework for private clouds. In *2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 482–489. IEEE, 2012.

M. Fischetti, S. Martello, and P. Toth. The fixed job schedule problem with working-time constraints. *Operations Research*, 37(3) :395–403, 1989.

L. Ford and D. Fulkerson. Flows in networks. princeton university press, princeton, nj, 1962. 194 s. *Ledelse og Erhvervsøkonomi*, 1962.

A. Fréville. The multidimensional 0-1 knapsack problem : An overview. *Eur. J. Oper. Res.*, 155(1) :1–21, 2004.

V. Gabrel. Scheduling jobs within time windows on identical parallel machines : New model and algorithms. *European Journal of Operational Research*, 83(2) :320–329, 1995.

J. E. Gallardo, C. Cotta, and A. Fernndez. A hybrid dynamic programming/ branch-and-bound algorithm for the multiple-choice knapsack problem. *Solving the Multidimensional Knapsack Problem Using an Evolutionary Algorithm Hybridized with Branch and Bound*, 3562 :21–30, 2005.

d. G. R. Garey, M.R. a, D. Johnson, and C. Yao. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory*, 21 :257–298, 1976.

M. R. Garey and D. S. Johnson. Computers and intractability : a guide to the theory of np-completeness. 1979. *San Francisco, LA : Freeman*, 58, 1979.

S. K. Garg, P. Konugurthi, and R. Buyya. A linear programming-driven genetic algorithm for meta-scheduling on utility grids. *International Journal of Parallel, Emergent and Distributed Systems*, 26(6) :493–517, 2011a.

S. K. Garg, C. S. Yeo, A. Anandasivam, and R. Buyya. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, 71(6) :732–749, 2011b.

A. M. Geoffrion. Proper efficiency and the theory of vector maximization. *Journal of mathematical analysis and applications*, 22(3) :618–630, 1968.

C. Gilmore, P and E. Gomory, R. The theory and computation of knapsack functions. *Operations Research*, 14 :1045–1074, 1966.

G. Gogrid2014. `http://www.gogrid.com/products/cloud-hosting`, 2014.

S. A. Goldman, J. Parwatikar, and S. Suri. Online scheduling with hard deadlines. *Journal of Algorithms*, 34(2) :370–389, 2000.

M. C. Golumbic. Algorithmic graph theory and perfect graphs, acad. *Press, New York*, 1980.

R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan. Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of discrete mathematics*, volume 5, pages 287–326. Elsevier, 1979.

G. Grid2014. `http://www.grid5000.fr`, 2014.

T. Guérout, Y. Gaoua, C. Artigues, G. D. Costa, P. Lopez, and T. Monteil. Mixed integer linear programming for quality of service optimization in clouds. *Future Gener. Comput. Syst.*, 71 :1–17, 2017.

U. I. Gupta, D.-T. Lee, and J.-T. Leung. An optimal solution for the channel-assignment problem. *IEEE Transactions on Computers*, .(11) :807–810, 1979.

M. Guzek, C. O. Diaz, J. E. Pecero, P. Bouvry, and A. Y. Zomaya. Impact of voltage levels number for energy-aware bi-objective dag scheduling for multi-processors systems. In *International Conference on Advances in Information Technology*, pages 70–80. Springer, 2012.

M. Habib, R. McConnell, C. Paul, and L. Viennot. Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2) :59–84, 2000.

M. T. Hajian, H. El-Sakkout, M. Wallace, J. M. Lever, and B. Richards. Towards a closer integration of finite domainpropagation and simplex-based algorithms. *Annals of Operations Research*, 81 :421–433, 1998.

J. Hamilton. Cooperative expendable micro-slice servers (cems) : low cost, low power servers for internet-scale services. In *Conference on Innovative Data Systems Research (CIDR'09)(January 2009)*. Citeseer, 2009a.

J. R. Hamilton. An architecture for modular data centers. In *CIDR 2007, Third Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 7-10, 2007, Online Proceedings*, pages 306–313. www.cidrdb.org, 2007. URL `http://cidrdb.org/cidr2007/papers/cidr07p35.pdf`.

J. R. Hamilton. Internet-scale data center power efficiency. In *CIDR 2009, Fourth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 4-7, 2009, Online Proceedings*. www.cidrdb.org, 2009b. URL `http://www-db.cs.wisc.edu/cidr/cidr2009/JamesHamilton_CEMS.pdf`.

S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207 (1) :1–14, 2010. ISSN 0377-2217.

A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment within large apertures. In *Proceedings of the 8th Design Automation Workshop*, pages 155–169, 1971.

M. Hifi. Approximate algorithms for the container loading problem. *Operations Research*, 9 :747–774, 2009.

H. Hoogeveen. Multicriteria scheduling. *European Journal of operational research*, 167(3) : 592–623, 2005.

C. Imreh. Scheduling problems on two sets of identical machines. *Computing*, 70(4) : 277–294, 2003.

D. E. Irwin, L. E. Grit, and J. S. Chase. Balancing risk and reward in a market-based task service. In *Proceedings. 13th IEEE International Symposium on High performance Distributed Computing, 2004.*, pages 160–169. IEEE, 2004.

T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In A. Chandrakasan and S. Kiaei, editors, *Proceedings of the 1998 International Symposium on Low Power Electronics and Design, 1998, Monterey, California, USA, August 10-12, 1998*, pages 197–202. ACM, 1998. doi : 10.1145/280756.280894. URL `https://doi.org/10.1145/280756.280894`.

B. Jarboui, P. Siarry, and J. Teghem. Métaheuristiques pour l'ordonnancement multicritère et les problèmes de transport, 2013.

L. Jourdan, M. Basseur, and E. Talbi. Hybridizing exact methods and metaheuristics : A taxonomy. *Eur. J. Oper. Res.*, 199(3) :620–629, 2009.

S. Kedad-Sidhoum, F. M. Mendonca, F. Monna, G. Mounie, and D. Trystram. Fast biological sequence comparison on hybrid platforms. In *43rd International Conference on Parallel Processing, ICPP 2014, Minneapolis, MN, USA, September 9-12, 2014*, pages 501–509. IEEE Computer Society, 2014.

S. Kedad-Sidhoum, F. Monna, and D. Trystram. Scheduling tasks with precedence constraints on hybrid multi-core machines. In *2015 IEEE International Parallel and Distributed Processing Symposium Workshop, IPDPS 2015, Hyderabad, India, May 25-29, 2015*, pages 27–33. IEEE Computer Society, 2015.

S. Kedad-Sidhoum, F. Monna, G. Mounié, and D. Trystram. A family of scheduling algorithms for hybrid parallel platforms. *Int. J. Found. Comput. Sci.*, 29(1) :63–90, 2018.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems*. Springer, 2004.

A. W. Kolen and L. G. Kroon. On the computational complexity of (maximum) class scheduling. *European Journal of Operational Research*, 54(1) :23–38, 1991.

A. W. Kolen and L. G. Kroon. An analysis of shift class design problems. *European Journal of Operational Research*, 79(3) :417–430, 1994.

R. Kolisch and S. Hartmann. Experimental investigation of heuristics for resource- constrained project scheduling : An update. *European journal of operational research*, 174.

R. Kolisch and S. Hartmann. *Heuristic algorithms for the resource-constrained project scheduling problem : Classification and computational analysis*. In : Węglarz J. (eds) Project Scheduling. International Series in Operations Research & Management Science, Springer, Boston, MA, 1999.

A. Konak, D. W. Coit, and A. E. Smith. Multi-objective optimization using genetic algorithms : A tutorial. *Reliability engineering & system safety*, 91(9) :992–1007, 2006.

O. Koné, C. Artigues, P. Lopez, and M. Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Comput. Oper. Res.*, 38(1) :3–13, 2011.

J. G. Koomey et al. Estimating total power consumption by servers in the us and the world, 2007.

M. Y. Kovalyov, A. Oulamara, and A. Soukhal. Two-agent scheduling on an unbounded serial batching machine. In *International Symposium on Combinatorial Optimization*, pages 427–438. Springer, 2012.

L. G. Kroon, M. Salomon, and L. N. Van Wassenhove. Exact and approximation algorithms for the operational fixed interval scheduling problem. *European Journal of Operational Research*, 82(1) :190–205, 1995.

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

V. Kumar. Approximating circular arc colouring and bandwidth allocation in all-optical ring networks. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 147–158. Springer, 1998.

M. Labbé, G. Laporte, and S. Martello. Upper bounds and algorithms for the maximum cardinality bin packing problem. *Eur. J. Oper. Res.*, 149(3) :490–498, 2003.

J.-M. Lagniez, D. Le Berre, T. De Lima, and V. Montmirail. A recursive shortcut for cegar : Application to the modal logic k satisfiability problem. In *IJCAI*, pages 674–680, 2017.

G. Lai, D. Yuan, and S. Yang. A new hybrid combinatorial genetic algorithm for multidimensional knapsack problems. *Journal of Supercomputing*, 70 :930–945, 2014.

J. H. Laros III, K. T. Pedretti, S. M. Kelly, W. Shu, K. B. Ferreira, J. V. Dyke, and C. T. Vaughan. *Energy-Efficient High Performance Computing - Measurement and Tuning*. Springer Briefs in Computer Science. Springer, 2013. ISBN 978-1-4471-4491-5. doi : 10.1007/978-1-4471-4492-2. URL `https://doi.org/10.1007/978-1-4471-4492-2`.

M. Laumanns, L. Thiele, and E. Zitzler. An efficient, adaptive parameter variation scheme for metaheuristics based on the epsilon-constraint method. *European Journal of Operational Research*, 169(3) :932–942, 2006.

Y. C. Lee and A. Y. Zomaya. Minimizing energy consumption for precedence-constrained applications using dynamic voltage scaling. In *2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 92–99. IEEE, 2009.

Y. C. Lee and A. Y. Zomaya. Energy efficient utilization of resources in cloud computing systems. *The Journal of Supercomputing*, 60(2) :268–280, 2012.

Y. C. Lee, C. Wang, A. Y. Zomaya, and B. B. Zhou. Profit-driven service request scheduling in clouds. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 15–24. IEEE, 2010.

J. Y. Leung, editor. *Handbook of Scheduling - Algorithms, Models, and Performance Analysis*. Chapman and Hall/CRC, 2004. ISBN 978-1-58488-397-5.

B. Li, J. Li, J. Huai, T. Wo, Q. Li, and L. Zhong. Enacloud : An energy-saving application live placement approach for cloud computing environments. In *2009 IEEE International Conference on Cloud Computing*, pages 17–24. IEEE, 2009.

B. Lienland and L. Zeng. A review and comparison of genetic algorithms for the 0-1 multidimensional knapsack problem. *Int. J. Oper. Res. Inf. Syst.*, 6(2) :21–31, 2015.

R. J. Lipton and A. Tomkins. Online interval scheduling. In *SODA*, volume 94, pages 302–311, 1994.

F. Liu, J. Tong, J. Mao, R. Bohn, J. Messina, L. Badger, and D. Leaf. Nist cloud computing reference architecture. *NIST special publication*, 500(2011) :1–28, 2011.

J. L. Lucas-Simarro, R. Moreno-Vozmediano, R. S. Montero, and I. M. Llorente. Scheduling strategies for optimal service deployment across multiple clouds. *Future Generation Computer Systems*, 29(6) :1431–1441, 2013.

T. Lust and J. Teghem. The multiobjective multidimensional knapsack problem : a survey and a new approach. *Int. Trans. Oper. Res.*, 19(4) :495–520, 2012.

S. Martello. Knapsack problems : algorithms and computer implementations. *Wiley-Interscience series in discrete mathematics and optimiza tion*, 1990.

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

G. Mavrotas. Effective implementation of the ε-constraint method in multi-objective mathematical programming problems. *Applied mathematics and computation*, 213(2) :455–465, 2009.

P. Mell, T. Grance, et al. The nist definition of cloud computing. 2011.

M. Mezmaz, N. Melab, Y. Kessaci, Y. C. Lee, E.-G. Talbi, A. Y. Zomaya, and D. Tuyttens. A parallel bi-objective hybrid metaheuristic for energy-aware scheduling for cloud computing systems. *Journal of Parallel and Distributed Computing*, 71(11) :1497–1508, 2011.

M. Microsoft2014. Microsoft windows azure`http://www.windowsazure.com/en-us/`, 2014.

K. Mills, J. Filliben, and C. Dabrowski. Comparing vm-placement algorithms for on-demand clouds. In *2011 IEEE Third International Conference on Cloud Computing Technology and Science*, pages 91–98. IEEE, 2011.

P. Minet, E. Renault, I. Khoufi, and S. Boumerdassi. Analyzing traces from a google data center. In *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pages 1167–1172. IEEE, 2018.

A. Mingozzi, M. A. Boschetti, S. Ricciardelli, and L. Bianco. A set partitioning approach to the crew scheduling problem. *Operations Research*, 47(6) :873–888, 1999.

S. Mittal and J. S. Vetter. A survey of CPU-GPU heterogeneous computing techniques. *ACM Comput. Surv.*, 47(4) :69 :1–69 :35, 2015.

H. Miyazawa and T. Erlebach. An improved randomized on-line algorithm for a weighted interval selection problem. *Journal of Scheduling*, 7(4) :293–311, 2004.

T. Moyaux, B. Chaib-Draa, and S. D'Amours. Multi-agent simulation of collaborative strategies in a supply chain. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems, 2004. AAMAS 2004.*, pages 52–59. IEEE, 2004.

K. Nakajima and S. L. Hakimi. Complexity results for scheduling tasks with discrete starting times. *Journal of Algorithms*, 3(4) :344–361, 1982.

K. Nakajima, S. L. Hakimi, and J. K. Lenstra. Complexity results for scheduling tasks in fixed intervals on two types of machines. *SIAM Journal on Computing*, 11(3) :512–520, 1982.

N. Nimbus2014. Nimbus cloud `ttp://www.nimbusproject.org/`, 2014.

O. OpenNebula2014. Opennebula project (opennebula.org). opennebula open-source cloud`http://www.opennebula.org/`, 09 2012.

O. OpenStack2014. Openstack.org. openstack : the open source cloud operating system, 2014.

A.-C. Orgerie, L. Lefèvre, and J.-P. Gelas. Save watts in your grid : Green strategies for energy-aware framework in large scale distributed systems. In *2008 14th IEEE international conference on parallel and distributed systems*, pages 171–178. IEEE, 2008.

V. Pareto. The new theories of economics. *Journal of political economy*, 5(4) :485–502, 1897.

J. M. Peha. Heterogeneous-criteria scheduling : minimizing weighted number of tardy jobs and weighted completion time. *Computers & operations research*, 22(10) :1089–1100, 1995.

P. Perez-Gonzalez and J. M. Framinan. A common framework and taxonomy for multicriteria scheduling problems with interfering and competing jobs : Multi-agent scheduling problems. *European Journal of Operational Research*, 235(1) :1–16, 2014.

C. C. Petersen. Computational experience with variants of the balas algorithm applied to the selection of r&d projects. *Management Science*, 13(9) :736–750, 1967.

J. Puchinger, G. R. Raidl, and U. Pferschy. The multidimensional knapsack problem : Structure and algorithms. *INFORMS J. Comput.*, 22(2) :250–265, 2010.

R. Rackspace2014. Rackspace open cloud`http://www.rackspace.com/cloud/`, 2014.

B. P. Rimal, E. Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *2009 Fifth International Joint Conference on INC, IMS and IDC*, pages 44–51. Ieee, 2009.

N. B. Rizvandi, J. Taheri, A. Y. Zomaya, and Y. C. Lee. Linear combinations of dvfs-enabled processor frequencies to modify the energy-aware scheduling algorithms. In *2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, pages 388–397. IEEE, 2010.

F. Rossi, P. Van Beek, and T. Walsh. *Handbook of constraint programming*. Elsevier, 2006.

D. M. Ryan and B. A. Foster. An integer programming approach to scheduling. *Computer scheduling of public transport urban passenger vehicle and crew scheduling*, pages 269–280, 1981.

S. S. Seiden. Randomized online interval scheduling. *Operations Research Letters*, 22(4-5) : 171–177, 1998.

S. SoftLayer2014. `http://www.softlayer.com`, 2014.

A. Soukhal. *Modèles et Algorithmes pour l'ordonnancement des travaux Indépendants et concurrents*. PhD thesis, Habilitation à diriger des recherches, Université François-Rabelais de Tours., 2012.

S. Srikantaiah, A. Kansal, and F. Zhao. Energy aware consolidation for cloud computing. *In Proceedings of Workshop on Power Aware Computing and Systems (HotPower). USENIX*.

G. Tesauro, R. Das, H. Chan, J. Kephart, and D. Levine. Flr iii, and c. *Lefurgy,"Managing power consumption and performance of computing systems using reinforcement learning," in NIPS*, 2007.

V. T'kindt and J.-C. Billaut. *Multicriteria scheduling : theory, models and algorithms*. Springer Science & Business Media, 2006.

J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente. Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. *Future generation computer systems*, 28(2) :358–367, 2012.

N. H. Tuong, A. Soukhal, and J.-C. Billaut. Single-machine multi-agent scheduling problems with a global objective function. *Journal of Scheduling*, 15(3) :311–321, 2012.

V. T'kindt, F. D. Croce, and J.-L. Bouquard. Enumeration of pareto optima for a flowshop scheduling problem with two criteria. *INFORMS Journal on Computing*, 19(1) :64–72, 2007.

L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner. A break in the clouds : towards a cloud definition, 2008.

M. Varnamkhasti. Overview of the algorithms for solving the multidimensional knapsack problems. *Advanced Studies in Biology*, 4 :37–47, 2012.

N. Vasić, M. Barisits, V. Salzgeber, and D. Kostic. Making cluster applications energy-aware. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 37–42, 2009.

J. S. Vaughan-Nichols. Containers ou vm : comment faire le bon choix ?, 05 2016. URL `https://www.lemondeinformatique.fr/actualites/lire-containers-ou-vm-comment-faire-le-bon-choix-64793.html`.

R. Vemuganti. Applications of set covering, set packing and set partitioning models : A survey. In *Handbook of combinatorial optimization*, pages 573–746. Springer, 1998.

A. Verma, P. Ahuja, and A. Neogi. pmapper : power and migration cost aware application placement in virtualized systems. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 243–264. Springer, 2008.

G. Von Laszewski, L. Wang, A. J. Younge, and X. He. Power-aware scheduling of virtual machines in dvfs-enabled clusters. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–10. IEEE, 2009.

S. Voßs, A. Fink, and C. Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136(1) :285–302, 2005.

T. Wenhong and Z. Yong. *Optimized Cloud Resource Management and Scheduling*. Morgan Kaufmann, 2015.

G. J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1) :5–16, 1994.

Z. Wu, X. Liu, Z. Ni, D. Yuan, and Y. Yang. A market-oriented hierarchical scheduling strategy in cloud workflow systems. *The Journal of Supercomputing*, 63(1) :256–293, 2013.

J. Xu and J. A. Fortes. Multi-objective virtual machine placement in virtualized data center environments. In *2010 IEEE/ACM Int'l Conference on Green Computing and Communications & Int'l Conference on Cyber, Physical and Social Computing*, pages 179–188. IEEE, 2010.

B. Yang, J. Geunes, and W. J. O'brien. *Resource-constrained project scheduling : Past work and new directions*. PhD thesis, Department of Industrial and Systems Engineering, University of Florida, 2001.

J. Yu and R. Buyya. Scheduling scientific workflow applications with deadline and budget constraints using genetic algorithms. *Scientific Programming*, 14(3, 4) :217–230, 2006.

B. Zahout, A. Soukhal, and P. Martineau. Fixed jobs multi-agent scheduling problem with renewable resources. In *International Conference on Artificial Evolution (Evolution Artificielle)*, pages 162–176. Springer, 2019.

Q. Zhang, L. Cheng, and R. Boutaba. Cloud computing : state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1) :7–18, 2010.