N° d'ordre : 106/2019-C/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de L'enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediène

Faculté de Mathématiques

THÈSE DE DOCTORAT
Présentée pour l'obtention du grade de Docteur

En : MATHEMATIQUES

Spécialité : Recherche Opérationnelle et Mathématiques Discrètes.

Par : MOHABEDDINE Amine

Sujet

ORDONNANCEMENT EN PRÉSENCE DE TÂCHES CONCORDANTES

Soutenue publiquement le 18/12/2019, devant le jury composé de :

M. M. E. CHERGUI	Professeur à l'USTHB	Président
M. M. BOUDHAR	Professeur à l'USTHB	Directeur de thèse
Mme F. AFFIF CHAOUCHE	Maitre de conference A à l'USTHB	Examinatrice
M. M.BENDRAOUCHE	Maitre de conference A à l'USDBlida	Examinateur
Mme Z. BENMEZIANE	Maitre de conference A à l'USTHB	Examinatrice
Mlle N. H.TELLACHE	Docteur à Sonatrach	Invitée

RÉSUMÉ

Dans cette thèse nous nous sommes intéressés au problème d'ordonnancement en présence d'un graphe de concordance sur machines parallèles. Etant donné un ensemble J de n tâches et un ensemble de m machines parallèles. Le problème consiste à ordonnancer l'ensemble des tâches de J sur l'ensemble des machines tout en respectant les contraintes de concordance. Ces contraintes permettent à certaines tâches spécifiques de s'ordonnancer simultanément sur différentes machines. Deux tâches pouvant s'ordonnancer simultanément sur deux machines différentes sont appelées tâches concordantes. Cette relation de concordance entre les tâches est modélisée par un graphe appelé graphe de concordance où chaque paire de tâches de J est concordante si et seulement si les deux tâches sont adjacentes dans le graphe de concordance. Les résultats de complexité étant fermés pour le cas d'un graphe de concordance arbitraire, nous avons considéré des cas particuliers où le problème reste ouvert. Nous avons prouvé que le problème est NP-difficile sur deux machines identiques lorsque le graphe de concordance est un arbre. Nous avons ensuite proposé un algorithme en $O(n)$ pour résoudre le problème sur deux machines identiques lorsque le graphe de concordance est un graphe caterpillar. Cet algorithme a été généralisé pour résoudre en $O(n^2)$ le problème sur deux machines identiques lorsque le graphe de concordance est un cycle. Un autre problème auquel nous nous sommes intéressés est le problème d'ordonnancement en présence d'un graphe de concordance sur deux machines uniformes pour des temps de traitement unitaires. Le problème a été considéré sous le nom de problème d'ordonnancement avec contraintes de ressources. Le problème a été prouvé NP-difficile lorsque la vitesse d'une des deux machines uniformes vaut le double de l'autre. Nous avons pu généraliser ce résultat en prouvant que le problème resterait NP-difficile au sens fort lorsque les vitesses des deux machines sont arbitraires. Nous avons ensuite résolu le problème en proposant des méthodes exactes : un modèle mathématique que nous allons résoudre et une méthode Branch & Bound. Nous avons également proposé des méthodes de résolution approchées en utilisant la metaheuristique Simulated Annealing. Des bornes inférieures ont également été proposées pour permettre l'évaluation des méthodes de résolution en calculant l'écart entre les solutions obtenues et la meilleure borne inférieure. Nous avons expérimenté et évalué les méthodes de résolution sur un nombre conséquent d'instances en croisant plusieurs paramètres : temps de traitement, densité du graphe de concordance et vitesses des machines. Nous avons également comparé les différentes méthodes en analysant plusieurs critères : écart avec la borne inférieure, temps d'exécution, ou encore le nombre de solutions optimales.

ABSTRACT

In this work, we consider the problem of scheduling with agreements on parallel machines. Given a set J of n jobs and a set of m parallel machines, the problem consists in scheduling the jobs of J on m parallel machines with the respect of the agreement constraints. Agreement constraints allow specific jobs to be scheduled concurrently on different machines. These constraints between jobs are modeled by a graph called agreement graph. In the agreement graph, two jobs of J can be scheduled concurrently on different machines if and only if these jobs are connected in the agreement graph. The complexity results of the problem for a general agreement graph are closed. So, we get interested for the cases of particular agreement graphs for which the problem is still opened. We have proved the NP-hardness of the problem for the case of tree graphs. We also proposed an $O(n)$ algorithm to solve the problem for caterpillar agreement graphs. We generalized this algorithm to solve in $O(n^2)$ the problem for cycle graphs. We also addressed the problem for two uniform machines and unit processing times. The problem was introduced under the name of ressource constraints problem and has been proved to be NP-hard in strong sense when the speed of one machine is double the speed of the other one. We generalized this result and proved that the problem is still NP-hard in strong sense for arbitrary speeds. Then, we solved the problem with approached methods using the simulated annealing algorithm. Exact methods were presented as well using a mathematical model that we are going to solve and a branch and bound algorithm. Lower bounds were also proposed, they allow us to evaluate the solving methods by calculating the gap between the solutions provided by the methods and the best performing lower bound. Finally, we experimented and evaluated the solving methods on a large number of instances by varying the three parameters : processing times, density of the agreement graph, and the speeds of the machines. The methods were compared by analyzing multiple criteria : the gap with the lower bound, the execution time, or the number of optimal solutions.

REMERCIEMENTS

Je voudrais tout d'abord remercier mon directeur de thèse Pr. Mourad Boudhar pour son support continu, sa patience, sa motivation, son écoute et sa disponibilité, il était présent à chaque fois que je rencontrais un problème ou lorsque j'avais besoin de conseils pour ma rédaction. C'est grâce à ses précieux conseils que nous avons pu mener nos recherches à bon port.

Je voudrais également remercier Dr. Mohamed Bendraouche de m'avoir apporté son support lors de la rédaction de notre article. Ses connaissances sur le thème étudié et en langue anglaise ont grandement contribué à l'amélioration de la qualité du papier. Je tiens également à exprimer toute ma reconnaissance à mes amis, mes collègues ainsi qu'à mes enseignants qui ont su trouver les bons mots pour m'encourager et me donner la force de continuer pendant les moments les plus durs.

J'adresse mes remerciements au Pr. CHERGUI Mohamed El-Amine d'avoir accepté de présider mon jury de thèse. Je tiens également à remercier Dr. AFFIF CHAOUCHE Fatima, Dr. BENDRAOUCHE Mohamed, Dr. BENMEZIANE Zineb et Dr. TELLACHE Nour El Houda de m'avoir fait l'honneur d'être examinateurs de cette thèse. Je les remercie particulièrement pour l'intérêt qu'ils ont bien voulu porter à notre travail ainsi que le temps accordé à la lecture et l'évaluation de cette thèse.

Mes derniers remerciements, mais pas des moindre, vont tout particulièrement à mes parents ainsi qu'à mes sœurs qui m'ont accompagné tout au long de mon doctorat, avec leur soutien et encouragements, sans eux ce travail n'aurait pas pu aboutir.

Table des matières

Liste des tableaux

Table des figures

Liste des symboles

Δ degré maximum dans un graphe

$\overline{I_p}(G)$ poids d'un SPM dans G

C_1 date de fin de traitement sur la première machine

C_2 date de fin de traitement sur la deuxième machine

c_j date de fin de traitement de la tâche $j \in J$

$C_{max}(\sigma)$ date de fin de traitement d'un ordonnancement σ

$d(j)$ degré de la tâche j dans le graphe de concordance

$DSWA$ problème de décision associé au problème SWA

$I_p(G)$ un SPM, p est une fonction sur les tâches représentant les temps de traitement des tâches

$Lv(j)$ ensemble des feuilles de j

m nombre de machines

n nombre de tâches

$N(j)$ ensemble des voisins de la tâche $j \in J$. Cette notation peut être généralisée pour un ensemble de tâches J par $N(J) = \cup_{j \in J} N(j)$

p_j temps de traitement de la tâche $j \in J$

r_j date de disponibilité de la tâche $j \in J$

SPM stable de poids maximum

SWA problème d'ordonnancement avec graphe de concordance (Schefuling With Agreements)

SWC problème d'ordonnancement avec graphe de conflit (Schefuling With Conflicts)

$t(J)$ défini par $t(J) = min_{k \in J}\{t_k\}$

t_j date de début de traitement de la tâche $j \in J$

Introduction

Le problème d'ordonnancement avec graphe de concordance (SWA en abrégé : Scheduling With Agreements) sur deux machines parallèles consiste à ordonnancer un ensemble de n tâches non préemptibles sur deux machines parallèles. L'ordonnancement des tâches se fait compte tenu de contraintes de concordance entre les tâches. Ces contraintes expriment le fait que seules des tâches spécifiques peuvent s'ordonnancer simultanément sur différentes machines. Les contraintes de concordance sont modélisées par un graphe appelé graphe de concordance dans lequel deux tâches j et k sont reliées par une arête si et seulement si j et k peuvent s'ordonnancer simultanément sur différentes machines. L'objectif du problème consiste à minimiser la date de fin de traitement de l'ordonnancement (C_{max}).

L'intérêt aux problèmes d'ordonnancement s'est vu accroitre à la fin des années 50 où le concept d'ordonnancement de tâches non-préemptives sur un ensemble de machines parallèles a vu le jour. Plusieurs travaux sur deux machines ont suivi. Pour n'en citer qu'eux, M. H. Rothkopf [25] a proposé son célèbre programme dynamique, ou encore J. K. Lenstra et al. [21] avec les premiers résultats sur la complexité des problèmes d'ordonnancement. Le problème étudié n'est pas le premier problème d'ordonnancement avec un graphe pour modéliser des contraintes entre les tâches. Les premiers travaux avec un graphe orienté pour modéliser des contraintes de précédence entre les tâches ont été publié par T.C. Hu [18]. C'est en 1996, que les premiers résultats sur des machines identiques et un graphe de conflit (complémentaire du graphe de concordance) ont été publié par B. S. Baker et E. G. Coffman [1]. D'autres travaux avec graphe de concordance (respectivement graphe de conflit) ont suivi. G. Even et. al. [12] ont publié les premiers travaux avec graphe de conflit et des temps de traitement arbitraire. M. Bendraouche et M. Boudhar [2, 4, 3] sont les premiers à avoir utilisé la notion de "concordance", les auteurs ont repris le problème défini par G. Even et. al. [12] en apportant Beaucoup de nouveaux résultats notamment sur la complexité du problème.

Cette thèse est organisée comme suit. Dans le chapitre 1, nous donnerons les définitions et notions de base sur la théorie de l'ordonnancement, la théorie des graphes et l'optimisa-

tion combinatoire pour une meilleure assimilation des résultats présentés dans les chapitres qui suivent. Nous présenterons également un état de l'art des principaux travaux réalisés autour du problème d'ordonnancement avec graphe de concordance (ou graphe de conflit). Nous finirons ce chapitre en donnant quelques domaines d'application. Le chapitre 2 sera dédié aux résultats trouvés lorsque les machines sont identiques. Nous allons prouver que le problème est NP-difficile lorsque le graphe de concordance est un arbre. Nous allons également proposer deux algorithmes polynomiaux pour résoudre le problème lorsque le graphe de concordance est un graphe caterpillar ou un cycle. Dans le chapitre 3, nous montrerons que le problème est NP-difficile au sens fort lorsque les machines sont uniformes, peu importe leurs vitesses. Le problème est polynomial pour certains cas que nous présentons aussi. Des bornes inférieures, une méthode métaheuristique, un modèle mathématique, ainsi qu'une méthode branch and bound vont être proposés pour résoudre le problème. Nous finirons ce chapitre par une expérimentation des méthodes proposées ainsi que la résolution du modèle. La conclusion représente la dernière section.

Chapitre 1

Préliminaires et état de l'art

Le but de ce chapitre est de fournir un état de l'art sur le sujet. Afin d'avoir une meilleure compréhension du travail, nous donnerons les concepts de base sur la théorie de l'ordonnancement, la théorie des graphes et de l'optimisation combinatoire. Nous présenterons des travaux antérieurs du problème d'ordonnancement étudié et de quelques problèmes en relation avec le sujet traité. Quelques domaines d'application seront présentées également.

1.1 Préliminaires

1.1.1 Théorie de l'ordonnancement

Un problème d'ordonnancement peut être modélisé par un ensemble de tâches qui doit être ordonnancé sur un système de machines. L'objectif du problème est d'ordonnancer l'ensemble des tâches sur l'ensemble des machines. L'ordonnancement des tâches peut être soumis à certaines contraintes (contraintes de concordance dans notre cas) de sorte à optimiser un ou plusieurs critères de performance (dans notre cas minimiser la date de fin de traitement de l'ordonnancement). Il existe deux systèmes de machines. Des machines parallèles qui ont la même fonction, et des machines spécialisées où chaque machine est dédiée à l'exécution d'un certain type de tâches ou d'opérations.

Machines parallèles

Comme discuté plus haut, les machines parallèles ont la même fonction. En d'autres termes, les machines parallèles exécutent le même type de tâches. Aussi, chaque machine parallèle ne peut exécuter qu'une seule tâche à la fois (deux tâches ne peuvent s'exécuter sur la même machine durant le même intervalle de temps). Il existe trois types de machines parallèles :

Identiques, *uniformes* et *générales*. On peut distinguer entre ces différents types de machines par leurs vitesses d'exécution. Les *machines identiques* sont des machines qui possèdent la même vitesse d'exécution. Lorsque les vitesses d'exécutions diffèrent d'une machine à l'autre (indépendamment des tâches à exécuter) alors les machines sont appelées *machines uniformes*. Enfin, si les vitesses d'exécution des différentes machines varient en fonction de la tâche à traiter (une même machine a une vitesse pour l'execution d'une tâche j et une vitesse différente pour l'execution d'une autre tâche k) alors ces machines sont appelées *machines générales*.

Ensemble des tâches

Toutes les tâches de l'ensemble des tâches sont caractérisées par des paramètres ou par des contraintes qui sont décrits comme suit :

- La date de début de traitement et la date de fin de traitement d'une tâche j représentent respectivement la date à laquelle la tâche j débute son traitement et la date à laquelle la tâche j termine son traitement. La date de début de traitement ainsi que la date de fin de traitement de la tâche j sont respectivement notées t_j et c_j.

- Le temps de traitement d'une tâche j noté p_j, représente la durée nécessaire au traitement de la tâche j sur une machine de l'ensemble des machines. Lorsque les tâches doivent être ordonnancées sur des machines uniformes, la durée de traitement d'une tâche j sur une machine uniforme est $\frac{p_j}{v}$, v étant la vitesse de la machine. Dans les chapitres qui suivent, on peut trouver deux types de temps de traitement : arbitraires ou unitaires.

- La date de disponibilité d'une tâche j noté r_j est la date à partir de laquelle la tâche j peut-être ordonnancée.

- La préemption des tâches représente le fait de pouvoir arrêter l'exécution d'une tâche et de reprendre son exécution plus tard dans le temps. Dans ce travail, nous allons considérer uniquement des problèmes d'ordonnancement avec des tâches non-préemptibles i.e. dont l'exécution se fait sans interruption.

Contraintes et objectifs

L'ordonnancement des tâches se fait généralement compte tenu de contraintes sur les tâches. Un ordonnancement du problème considéré, se fait en présence de contraintes appelées contraintes de concordance. Les contraintes de concordance permettent à deux tâches de s'ordonnancer simultanément (dans le même intervalle de temps) sur deux machines différentes si et seulement

si les deux tâches sont concordantes. Ces contraintes de concordance entre les tâches sont modélisées par un graphe appelé graphe de concordance. Un graphe de concordance $G = (J, E)$ est composé d'un ensemble de tâches J et d'un ensemble d'arêtes E où chaque paire de tâches $j \in J$ et $k \in J$ est relié par une arête si et seulement si les tâches j et k sont concordantes.

L'ordonnancement des tâches sur l'ensemble des machines doit se faire en optimisant un ou plusieurs objectifs. L'unique objectif à optimiser dans le problème considéré est de minimiser la date de fin de traitement de l'ordonnancement. i.e. minimiser la date de fin de traitement de la tâche qui termine son exécution en dernier.

1.1.2 Théorie des graphes

Graphe simple. Un graphe Simple $G = (V, E)$ est défini par l'ensemble des sommets V et l'ensemble des arêtes E. Une arête $e \in E$ est définie par deux sommets reliés par e, ces deux sommets sont appelés extrémités de e. Un graphe simple ne peut contenir ni d'arêtes multiples [1] ni de boucle [2]. Dans ce travail nous allons considérer un graphe de concordance. Un graphe de concordance est un graphe simple non orienté où l'ensemble des sommets représente l'ensemble des tâches.

Adjacence et voisinage. Si deux sommets u et v sont reliés par une arête alors on dit que u et v sont adjacents. L'ensemble de tous les sommets adjacents à un sommet v est appelé le voisinage de v, et les sommets du voisinage de v sont appelés voisins de v. On note le voisinage d'un sommet v par $N(v)$.

Sous-graphe et sous-graphe induit. Un graphe $H = (V_H, E_H)$ est un sous-graphe de $G = (V, E)$ si $V_H \subset V$ et $E_H \subset E$. H est un sous-graphe induit de G si $V_H \subset V$ et chaque deux sommets $u, v \in V_H$ sont adjacents dans H si et seulement si u et v sont adjacents dans G.

Graphs isomorphiques. Soient deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$. Les graphe G et H sont isomorphiques si il existe une permutation $p : V_G \to V_H$ tel que $\{u, v\} \in E$ si et seulement si $\{p(v), p(u)\} \in E_H$.

Graphe pondéré. Un graphe pondéré de ses sommets est un graphe où à chaque sommet v est associé un poids $p(v)$. Dans un graphe de concordance, le temps de traitement p_j représentent

1. Lorsque deux sommets sont reliés par plusieurs arêtes
2. Une arête possédant deux extrémités identiques

le poids associé à la tâche j.

Degré. Le degré d'un sommet v noté $d(v)$ représente le nombre de voisins du sommet v.

Feuilles. Les feuilles d'un graphe $G = (V, E)$ sont tous les sommets $v \in V$ tels que $d(v) = 1$.

Stable maximum et stable de poids maximum. Soit $G = (V, E)$ un graphe simple, non orienté et pondéré :

- Un stable de G noté $I(G)$ est un sous-ensemble de sommet de V dont les sommets sont deux à deux non adjacents.

- Un stable maximum $I^*(G)$ est un stable de G tel que pour tout stable $I(G)$ de G, $|I^*(G)| \geq |I(G)|$.

- Un stable de poids maximum $I_p(G)$ de G est un stable de G tel que pour tout $I(G)$ de G, $\sum_{v \in I_p(G)} p(v) \geq \sum_{v \in I(G)} p(v)$. Le poids du stable de poids maximum est noté $\overline{I_p}(G) = \sum_{v \in I_p(G)} p(v)$.

Quelques classes de graphes

- *Les chaines* sont des graphes non-orientés dont les sommets peuvent être listés $v_1, v_2, \ldots, v_n$ tels que les arêtes sont $\{v_i, v_{i+1}\}$, $i = 1, \ldots, n-1$.

- *Les cycles élémentaires* sont des graphes non-orientés qui sont une suite d'arrêtes consécutives (chaines) dont les deux sommets extrémités sont identiques.

- *Les arbres* sont des graphs non-orientés dans lesquels chaque paire de tâches est reliée par exactement une seule chaine.

- Etant donné une chaine appelé chaine centrale, *les graphes caterpillars* sont des arbres pour lesquels chaque sommet est à une distance 1 de la chaine centrale.

Toutes les définitions et notations présentées dans cette section reste valide pour le cas d'un graphe de concordance où les sommets sont des tâches.

1.1.3 Optimisation combinatoire

Problème d'optimisation [15]

Un problème d'optimisation peut être vu comme une réponse à une question, généralement traitant plusieurs paramètres. Un problème est décrit en donnant : une description de tous ses paramètres, et une détermination des propriétés que la réponse, ou la solution, doit satisfaire. En spécifiant des valeurs pour tous les paramètres du problème on obtient une instance du problème.

Si on prend pour exemple le problème SWA sur machines parallèles, les paramètres de ce problème consistent en un ensemble fini de tâches non préemptibles J, un ensemble de machines parallèles, des temps de traitement p_j, $\forall j \in J$, et des relations de concordance entre chaque paire de tâches. Une solution est de trouver un ordonnancement des tâches de J sur l'ensemble des machines qui minimise la date de fin de traitement de l'ordonnancement.

Problèmes d'optimisation et Problèmes de décision

Une solution à un problème d'optimisation est associée à une valeur à optimiser. Résoudre un problème d'optimisation revient à déterminer la solution qui fournit la meilleure valeur possible. Un problème de décision associé à un problème d'optimisation fournit une réponse "oui" ou "non" à la question suivante : étant donné un entier K, existe-il une solution réalisable au problème d'optimisation tel que la valeur associée à cette solution est inférieure ou égale à K (pour un problème de minimisation). En prenant toujours le SWA sur machine parallèle comme exemple, le problème de décision associé au problème d'optimisation du problème SWA est décrit comme suit : Etant donné un entier K, un ensemble fini de tâches non préemptibles J, des temps de traitement p_j, $\forall j \in J$, et des relations de concordance entre chaque paire de tâches. Existe-il un ordonnancement σ des tâches de J sur un ensemble de machines parallèles tel que $C_{max}(\sigma) \leq K$?

Complexité des algorithmes [15]

Généralement, nous cherchons à trouver l'algorithme le plus "efficace" dans le but de résoudre un problème. Dans un sens large, l'efficacité implique toutes les différentes ressources nécessaires à l'exécution d'un algorithme. Cependant, par algorithme le "plus efficace" on exprime généralement le fait qu'il soit le plus rapide. Puisque le temps requis est le facteur le plus important qui détermine si un algorithme est assez efficace pour être utilisé en pratique.

Le temps requis à l'exécution d'un algorithme est communément exprimé en termes d'une seule variable, la "taille" d'une instance d'un problème, censé refléter la quantité de données

nécessaires à la description du problème. Pour le cas des problèmes d'ordonnancement, par exemple, le nombre de tâches est communément utilisé pour cette fin.

La complexité d'un algorithme est le nombre d'étapes requises à la résolution d'un problème de taille n. L'objectif dans la détermination de la complexité d'un algorithme n'est pas d'obtenir le nombre exact d'opérations mais d'obtenir une borne asymptotique sur le nombre d'étapes. La notation Grand-O fait usage d'analyses asymptotiques.

Notation Grand-O : un algorithme a une complexité $f(n) = O(g(n))$ s'il existe des constantes positives n_0 et c tel que $\forall n > n_0$, $f(n) \leq c.g(n)$ (la fonction $g(n)$ est une borne supérieure pour la fonction $f(n)$).

Algorithmes polynomiaux : Un algorithme est polynomial si sa complexité est $O(p(n))$, où $p(n)$ est une fonction polynomiale.

Algorithmes pseudo-polynomiaux : Un algorithme est pseudo-polynomial si sa complexité est $O(p)$, où p est une fonction polynomiale en fonction de la taille de l'instance mais aussi des valeurs des autres paramètres du problème.

Algorithmes exponentielles : Un algorithme est exponentielle si sa complexité est $O(c^n)$, où c est une constante réelle strictement supérieure à 1.

Complexité des problèmes [15]

La complexité d'un problème est équivalente à la complexité du meilleur algorithme permettant de résoudre le problème. Un problème est facile à résoudre s'il existe un algorithme polynomial qui permet de résoudre le problème. Un problème est irrésolvable ou difficile à résoudre s'il ne connait aucun algorithme polynomial pour résoudre le problème.

Un aspect important de la théorie de la complexité est de catégoriser les problèmes en classes de complexité. Une classe de complexité représente l'ensemble de tous les problèmes qui peuvent être résolus en utilisant une certaine quantité de ressources temporelles. Il existe deux importantes classes de problèmes : P et NP.

- *La classe P* représente l'ensemble de tous les problèmes de décision qui peuvent être résolus par un algorithme (déterministe) en temps polynomial.

– *La classe NP* représente l'ensemble de tous les problèmes de décision qui peuvent être résolus par un algorithme non déterministe en temps polynomial. Un algorithme non déterministe contient un ou plusieurs choix dans lesquels plusieurs scénarios sont possibles sans aucune connaissance du choix qui sera pris.

Réduction polynomiale : Un problème de décision A se réduit polynomialement au problème de décision B si, pour toute instance I_A de A, une instance I_B du problème B peut être construite en temps polynomial, tel que, I_A admet une solution si et seulement si I_B admet une solution.

Problème NP-complet et problème NP-difficile : Un problème de décision $A \in NP$ est NP-complet si tous les autres problèmes de la classe NP peuvent être réduit au problème A. Les problèmes NP-difficiles sont des problèmes d'optimisation dont les problèmes de décision associés sont NP-complets.

Problème NP-complet au sens fort : Si un problème A est NP-difficile au sens fort alors A ne peut être résolu par aucun algorithme pseudo-polynomial.

Méthodes d'optimisation

Un problème d'optimisation peut être résolu par une méthode exacte ou une méthode approchée. Les méthodes de résolution exactes obtiennent des solutions optimales avec garentie de leurs optimalités. Cependant, pour des problèmes NP-difficiles, les méthodes de résolution exactes ne sont pas polynomiales et lorsque la taille des instances commence à augmenter ces méthodes sont souvent pas très efficace. Pour un usage plus pratique, les méthodes approchées fournissent des solutions réalisables, sans garantie d'obtention d'une solution optimale, en un temps raisonnable.

Méthodes de résolution exactes : Il existe plusieurs méthodes de résolution exactes : programmation dynamique, la famille d'algorithme branch and X (branch and bound, branch and price, branch and cut), programmation par contrainte, ect. Etant donné que nous allons utiliser uniquement la méthode branch and bound, cela sera donc la seule méthode exacte que nous allons détailler.

L'algorithme *branch and bound* est basé sur une énumération implicite de toutes les solutions du problème d'optimisation considéré. L'espace de recherche est exploré en construisant de manière dynamique un arbre de recherche dont la racine représente l'espace de recherche associé. Les feuilles représentent les solutions potentielles et les nœuds intérieurs sont des sous-

problèmes de tout l'espace de solutions. Le taillage de l'arbre de recherche est basé sur une fonction permettant le calcul d'une borne inférieure sur un nœud intérieur qui permet de tailler les sous-arbres qui ne contiennent pas de solutions optimales.

Algorithmes approchés : Il existe deux types d'algorithmes approchés : des algorithmes d'approximation et des algorithmes heuristiques. Contrairement aux méthodes heuristiques, qui fournissent une solution réalisable en un temps raisonnable sans aucune garantie sur la qualité de cette solution, les algorithmes d'approximation procurent des solutions avec des garanties de leurs qualités.

Les algorithmes heuristiques procurent des solutions réalisables sur des instances de problèmes de grandes tailles. Ils permettent d'obtenir des solutions avec des performances acceptables à un coût acceptable pour un large éventail de problèmes. En général, les heuristiques n'ont pas de garantie d'approximation sur les solutions obtenues. Elles peuvent être classées en deux familles : des heuristiques spécifiques et les métaheuristiques.

- Les heuristiques spécifiques sont faites sur-mesure et conçues pour un problème spécifique.
- Les métaheuristiques sont des algorithmes généraux qui peuvent être appliqués pour résoudre pratiquement n'importe quel problème d'optimisation.

Quelques problèmes NP-complets

- *Problème de partition :* Instance : un ensemble fini de n éléments A, $s(a_i)$ une taille pour chaque élément $a_i \in A$, $i = 1, \ldots, n$, et un entier positif $B = \frac{\sum_{a_i \in A} s(a_i)}{2}$. Question : Existe-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a_i \in A'} s(a_i) = B$? ce problème est NP-complet [15].

- *Problème de partition d'un graphe en sous-graphes isomorphiques :* Instance : Un entier positif q et deux graphes $G = (V, E)$ et $H = (V', E')$ avec $|V'| \geq 3$ et $|V| = q|V'|$. Question : Existe-il une partition des sommets du graphe G en q ensembles disjoints V_1,..., V_q tel que le sous-graphe induit par les sommets de V_i ($i = 1, \ldots, q$) a le même nombre de sommets que H et contient un graphe partiel isomorphique à H ? Ce problème est NP-complet (Garey et Johnson [15]).

- *Problème de stable maximum :* Instance : Un graphe $G = (V, E)$ et un entier positif K. Question : Existe-il un sous-ensemble de sommets $I \in V$ où les sommets de I sont

deux à deux non adjacent dans G tel que $|I| \geq K$? Ce problème est NP-complet mais le problème peut devenir polynomial pour des classes particulières de graphes.

– *Problème de stable de poids maximum :* Instance : Un graphe $G = (V, E)$, un poids $p(v)$ pour chaque sommet $v \in V$ et un entier positif K. Question : Existe-il un un stable $I_p \in V$ tel que $\sum_{v \in I_p} p(v) \geq K$? Ce problème est NP-complet mais le problème peut devenir polynomial pour des classes particulières de graphes.

1.2 Etat de l'art

1.2.1 Machines identiques

Problème SWA

Le problème SWA a été introduit par Even et al. [12] sous le nom de problème d'ordonnancement avec graphe de conflit [3] (en abrégé SWC : Scheduling With Conflict). Les auteurs ont proposé dans ce papier un algorithme polynomial pour résoudre le problème SWA sur deux machines identiques pour des temps de traitement $p_j \in \{1, 2\}$. Ils ont également proposé un $\frac{4}{3}$-approximation algorithme pour résoudre le problème SWA sur deux machines identiques et des temps de traitement $p_j \in \{1, 2, 3\}$. Les auteurs ont aussi prouvé que le problème SWA sur deux machines identiques avec un graphe de concordance biparti et des temps de traitement $p_j \in \{1, 2, 3, 4\}$ est un problème NP-difficile. La deuxième partie de leurs travail est dédiée à la version online du problème.

Bendraouche et Boudhar [2] ont prouvé la NP-difficulté du problème SWA sur deux machines identiques avec un graphe de concordance biparti et des temps de traitement $p_j \in \{1, 2, 3\}$. Toujours pour un graphe de concordance biparti et deux machines identiques, les auteurs ont prouvé la NP-difficulté du problème SWA lorsque les temps de traitement $p_j \in \{1, 2\}$ et les temps de disponibilité $r_j \in \{0, r\}$. Les auteurs ont aussi proposé un algorithme polynomial pour résoudre le problème sur deux machines identiques avec un graphe de concordance biparti dans lequel une des deux parties des tâches du graphe de concordance ont des temps de traitement unitaires. Un algorithme de liste a également été proposé par les auteurs permettant une résolution approchée du problème sur m machines identiques et un graphe de concordance arbitraire.

3. Graphe complémentaire du graphe de concordance représentant le conflit entre les tâches

Bendraouche et Boudhar [4] ont prouvé la NP-difficulté du problème SWA avec graphe de concordance biparti sur deux machines identiques et deux temps de traitement différents : $p_j = a$ ou $p_j = 2a + b$ avec $b \neq 0$. Dans la deuxième partie de leurs travail, les auteurs ont prouvé l'équivalence entre le problème SWA et le problème d'ordonnancement avec contraintes de ressources lorsque la disponibilité de chaque ressource vaut 1 et la demande de chaque tâche pour n'importe quelle ressource vaut 0 ou 1. Les mêmes auteurs ont proposé dans un autre papier (Bendraouche et Boudhar [3]) une preuve de la NP-difficulté du problème SWA sur deux machines identiques avec un graphe de concordance sous la forme $G = (A \cup B, E)$ où le sous-graphe induit par les sommets de A forme un stable et celui induit par les sommets de B forme un graphe général. Ils ont également proposé quelques algorithmes polynomiaux lorsque le graphe de concordance est un complément de graphes biparti ou un graphe scindé.

TABLE 1.1: Complexité du problème SWA sur deux machines identiques.

Types de graphe	Temps de traitement	Temps de disponibilité	Complexité	Références
Arbitraire	$p_j \in \{2a, a\}$	-	Polynomial	[4]
Arbitraire	$p_j \in \{a, 2a + b\}$	-	NP-difficile	[4]
Biparti	$p_j \in \{1, 2, 3\}$	-	NP-difficile	[2]
Biparti	$p_j \in \{1, 2, 3, 4\}$	-	NP-difficile	[12]
Biparti	$p_j \in \{1, 2\}$	$r_j \in \{0, r\}$	NP-difficile	[2]

Problème d'exclusion mutuelle

Pour le cas de temps de traitement unitaires, Baker et Coffman [1] ont introduit le problème sous le nom de problème d'exclusion mutuelle pour lequel ils ont considéré un graphe de conflit au lieu d'un graphe de concordance. Les auteurs ont proposé des algorithmes polynomiaux pour résoudre le problème sur deux machines identiques lorsque le graphe de concordance est une forêt. Les auteurs ont également prouvé que le problème pour un nombre de machine $m \geq 3$ est NP-difficile.

Beaucoup de travaux ont été réalisé sur la complexité du problème d'exclusion mutuelle, ces résultats sont résumés comme suit : le problème d'exclusion mutuelle est polynomial pour des graphes de concordance : bipartis et co-graphes pour un nombre de tâches fixes [6], des graphes d'intervalles lorsque $m \geq 4$ [6], des graphes de permutation pour $m \geq 6$ [19], et des graphes de tolérances bornées pour $m \geq 3$ [13]. Le problème d'exclusion mutuelle est NP-difficile pour le

cas de : Graphes scindés pour $m \geq 3$ [13], graphes bipartis et forêts pour $m \geq 3$ [6], collection de cliques pour $m \geq 3$ ([10]), co-graphes [6], et des graphes circulaire-arc propre pour $m \geq 3$ [13].

1.2.2 Machines uniformes

Sur machines uniformes, il n'existe pas beaucoup de travaux. Le résultat le plus important a été présenté par J. Blazewicz et al. [5]. Les auteurs ont prouvé que le problème SWA sur deux machines uniformes est NP-difficile lorsque une des deux machines est deux fois plus rapide que l'autre.

1.2.3 Machines spécialisées

Pour le cas de machines dédiées ou spécialisées, Tellache et Boudhar [27] ont étudié le problème Open Shop avec graphe de conflit (OSC en abrégé). Ils ont montré que le problème OSC sur deux machines avec des temps de traitement $p_j \in \{1, 2, 3\}$ est NP-difficile même pour un graphe complémentaire du graphe biparti. Le même résultat est valable pour le problème OSC sur trois machines avec des temps de traitement unitaires et un graphe de conflit arbitraire. Des algorithmes efficaces ont également été proposés pour le problème OSC sur deux machines avec des temps de traitement $p_j = \{0, 1, 2\}$, et pour le problème OSC sur trois machines, des temps de traitement unitaires et un graphe de conflit ayant la structure d'un graphe sans triangle. Ils ont prouvé aussi que le problème OSC devient polynomial lorsque la préemption des tâches est permise. D'autre part, ils ont trouvé une équivalence entre le problème OSC est un cas particulier du problème de contraintes de ressources sur deux machines uniformes et ainsi de nouveaux résultats sur la complexité du deuxième problème cité ont été établis. Ils ont présenté une heuristique en deux phases et des bornes inférieures pour le problème OSC sur m machines. Les mêmes auteurs [29] ont considéré le problème Flow Shop avec graphe de Conflit (FSC en abrégé). Ils ont prouvé que le problème FSC est NP-difficile au sens fort même pour des graphes complémentaires du graphe scindé complet. Ils ont également montré que le problème reste NP-difficile pour un graphe complémentaire du graphe biparti complet. Ce dernier résultat est maintenu même en permettant la préemption des tâches. Ils ont ensuite considéré le cas d'opérations avec des temps de traitement unitaires et ont prouvé que le problème est NP-difficile au sens fort. D'autre part, un algorithme en $O(n^{2,5})$ a été proposé pour le problème FSC sur deux machines et des temps de traitement unitaires, et les classes de graphes pour lesquelles ce problème est polynomial ont été proposé. De plus, des heuristiques et des bornes inférieures ont été présenté. Tellache et Boudhar [28] ont étudié le problème FSC sur deux machines avec des temps de traitement d'opérations unitaires. Ils ont développé un modèle mathématique et

un algorithme branch and bound. Pour le même problème, Cai Y. et al. [7] ont présenté un $\frac{4}{3}$-approximation algorithme pour un graphe de conflit arbitraire. Pour des temps de traitement arbitraires et un graphe de concordance complémentaire du graphe biparti complet, ils ont mis en place un simple $\frac{3}{2}-$approximation algorithme. Le problème FSC sur deux machines a également été étudié dans [26], les auteurs ont prouvé que les ordonnancements de permutation ne sont pas dominants même pour deux machines et ils ont présenté un cas spécial pour lequel un ordonnancement optimal peut être obtenu par un ordonnancement de permutation. Quatre modèles mathématiques (MILP) ont été présenté avec une expérimentation de leurs résolutions.

Applications

Le problème SWA apparait généralement dans la littérature comme un problème d'ordonnancement avec contraintes de ressources lorsque les ressources ne peuvent pas être partagées. Deux tâches conflictuelles (non concordantes) représentent une paire de tâches qui utilise la même ressource [14]. Dans la littérature d'autres applications peuvent être trouvées, Bendraouche et Boudhar [2] ont présenté une application, dérivée du problème d'ordonnancement avec contraintes de ressources, pour la gestion de salles lors de l'organisation de Workshops. Baker et Coffman [1] ont proposé une application pour balancer la charge entre les différents processeurs dans la programmation parallèle. M. Halldórsson et al. [16] ont proposé d'autres applications dans le contrôle du trafic d'intersection, assignement de fréquence pour des réseaux cellulaires ou encore dans la gestion de sessions dans un réseau local.

Chapitre 2

Machines identiques

2.1 Introduction

Dans ce chapitre, nous considérons le problème d'ordonnancement avec graphe de concordance sur deux machines identiques. Nous allons prouver, en premier lieu, que le problème est NP-difficile lorsque le graphe de concordance est un arbre. La preuve consiste à réduire polynomialement le problème de décision associé au problème SWA noté DSWA au problème de partition. Nous proposons ensuite un algorithme polynomial, basé sur le calcul du stable de poids maximum, pour résoudre le problème lorsque le graphe de concordance est un graphe caterpillar. Nous présenterons un autre algorithme polynomial basé sur l'algorithme précédent, afin de résoudre le problème SWA lorsque le graphe de concordance est un cycle (nous considérons seulement le cas de cycles élémentaires).

2.2 Arbres

Afin de prouver que le problème est NP-difficile lorsque le graphe de concordance est un arbre, nous allons réduire polynomialement le problème de partition au problème de décision dérivé du problème SWA noté DSWA.

Définition du problème DSWA

Instance : un ensemble de tâches J, des temps de traitement p_j, $\forall j \in J$, un graphe de concordance $G = (J, E)$, et un entier positif K. Question : Existe-il un ordonnancement réalisable σ du problème SWA sur deux machines identiques avec le graphe de concordance G tel que $C_{max}(\sigma) \leq K$?

Définition du problème de partition

Soient A un ensemble fini de n éléments, $s(a_i)$ une taille pour chaque élément $a_i \in A$, $i = 1, \ldots, n$, et $B = \frac{\sum_{a_i \in A} s(a_i)}{2}$. Question : Existe-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a_i \in A'} s(a_i) = B$? ce problème est NP-complet [15].

Théorème 1. *([23, 24]) Le problème SWA sur deux machines identiques en présence d'un arbre de concordance est NP-difficile.*

Démonstration. On considère le problème de partition défini précédemment. Nous allons prouver que le problème de partition peut être réduit polynomialement au problème DSWA en présence d'un arbre de concordance.

Soit une instance du problème de partition, l'instance du DSWA qui lui correspond est défini comme suit : On considère un ensemble J de $2n + 5$ tâches et un arbre de concordance $T = (J, E)$. L'ensemble J est composé de deux sous-ensembles de n tâches X et Y ainsi que de cinq tâches u, v_1, v_2, w_1 et w_2. Les temps de traitement de ces tâches sont donnés par la Table 2.1. Les tâches qui correspondent aux éléments de A sont les tâches de X et de Y, en effet, chacune des tâches x_i et y_j correspond à l'élément a_i, $i = 1, \ldots, n$. On peut voir les tâches de X comme les "vrais" représentants des éléments de A, et les tâches de Y comme des représentants "fictifs". L'arbre de concordance T est défini comme suit (voir Figure 2.1) : La tâche u est adjacente à toutes les tâches de X ainsi qu'aux tâches v_1 et v_2. La tâche x_j est adjacente à la tâche y_j ($i = 1, \ldots, n$). Enfin la tâche v_i est adjacente à la tâche w_i ($i \in \{1, 2\}$).

TABLE 2.1: Temps de traitement des tâches de l'arbre de concordance T.

Sous-ensembles	-	-	-	X	Y
Tâches	u	v_i	w_i	x_i	y_i
p_j	$2B + 2$	2	1	$2.s(a_i)$	$s(a_i)$
i	-	$i \in \{1, 2\}$	$i \in \{1, 2\}$	$i = 1, \ldots, n$	$i = 1, \ldots, n$

Afin de prouver que le problème DSWA en présence d'un arbre de concordance est NP-difficile, nous devons prouver qu'il existe une partition A' de A si et seulement s'il existe un ordonnancement réalisable σ du problème SWA sur deux machines identiques en présence de l'arbre de concordance T tel que $C_{max}(\sigma) \leq 5B + 4$.

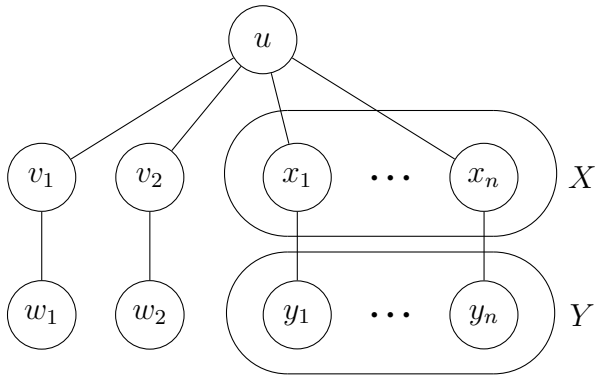


FIGURE 2.1: L'arbre de concordance T

D'abord, on suppose que $A' \subseteq A$ est une partition de A tel que $\sum_{a_i \in A'} = B$. L'ordonnancement σ est décrit comme suit : La tâche v_i est ordonnancée simultanément aux tâches u et w_i, $i = \{1, 2\}$. Les tâches de $X' \subseteq X$ qui correspondent aux éléments de A' sont ordonnancées entre les tâches v_1 et v_2 simultanément à la tâche u. Le reste des tâches de X noté X' sont ordonnancées simultanément aux tâches qui leurs sont adjacentes de $Y' \subseteq Y$. Les tâches restantes de Y sont ordonnancées en fin de l'ordonnancement. L'ordonnancement σ est montré par la Figure 2.2. Dans les illustrations qui suivent de l'ordonnancement σ, les tâches de Y'' doivent être séparées par des temps morts, mais pour faciliter la lisibilité des diagrammes, elles sont représentées sous forme d'un seul bloque de tâches, ce qui n'a aucune incidence sur la preuve. Le C_{max} de l'ordonnancement σ est le suivant :

$$\begin{aligned} C_{max}(\sigma) &= p_{v_1} + \sum_{j \in X'} p_j + p_{v_2} + \sum_{j \in X''} p_j + \sum_{j \in Y'} p_j \\ &= 2 + (2B + 2) + 2B + B \\ &= 5B + 4 \end{aligned}$$

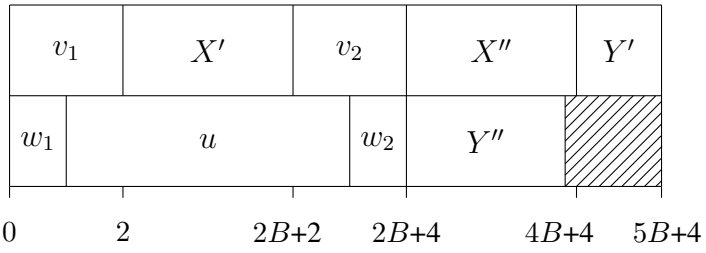


FIGURE 2.2: L'ordonnancement σ (les tâches de Y'' sont séparées par des temps morts

Inversement, on suppose qu'il existe un ordonnancement réalisable σ du problème SWA sur deux machines identiques en présence de l'arbre de concordance T tel que $C_{max}(\sigma) \leq 5B + 4$.

On note par $X' \subseteq X$ le sous-ensemble de tâches qui sont ordonnancées simultanément à la tâche u, l'ensemble du reste des tâches de X est noté X''. Le sous-ensemble de tâches qui sont ordonnancées simultanément aux tâches de X' est noté X, l'ensemble des tâches restantes de Y est noté Y''.

En premier lieu, on suppose que la tâche w_i est entièrement ordonnancée en parallèle à la tâche v_i, et v_i est ordonnancée le plus possible en parallèle à la tâche u ($i \in \{1, 2\}$). La date de fin de traitement de l'ordonnancement σ dépend donc de la somme des temps de traitement des tâches de X'. Nous distinguons trois cas :

1. $\sum_{j \in X'} p_j = 2B + \epsilon$ ($\epsilon > 0$) : L'ordonnancement σ est décrit par la Figure 2.3 et à la date de fin de traitement suivante :

$$\begin{aligned} C_{max}(\sigma) &= p_{v_1} + \sum_{j \in X'} p_j + p_{v_2} + \sum_{j \in X''} p_j + \sum_{j \in Y'} p_j \\ &= 2 + (2B + \epsilon) + 2 + (2B - \epsilon) + (B + \frac{\epsilon}{2}) = 5B + 4 + \frac{\epsilon}{2} \end{aligned}$$

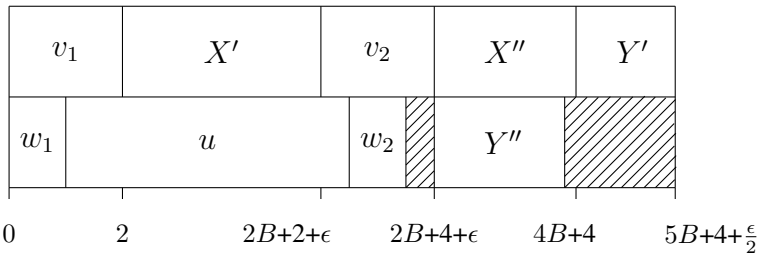


FIGURE 2.3: L'ordonnancement σ dans le premier cas (les tâches de Y'' sont séparées par des temps morts).

2. $\sum_{j \in X'} p_j = 2B - \epsilon$ ($\epsilon > 0$) : L'ordonnancement σ est décrit par la Figure 2.4 et à la date de fin de traitement suivante :

$$\begin{aligned} C_{max}(\sigma) &= p_{w_1} + p_u + p_{w_2} + \sum_{j \in X''} p_j + \sum_{j \in Y'} p_j \\ &= 1 + (2B + 2) + 1 + (2B + \epsilon) + (B - \frac{\epsilon}{2}) = 5B + 4 + \frac{\epsilon}{2} \end{aligned}$$

3. $\sum_{j \in X'} p_j = 2B$: L'ordonnancement σ est décrit par la Figure 2.2 et à la date de fin de traitement suivante :

$$\begin{aligned} C_{max}(\sigma) &= p_{v_1} + \sum_{j \in X'} p_j + p_{v_2} + \sum_{j \in X''} p_j + \sum_{j \in Y'} p_j \\ &= 2 + 2B + 2 + 2B + B = 5B + 4 \end{aligned}$$

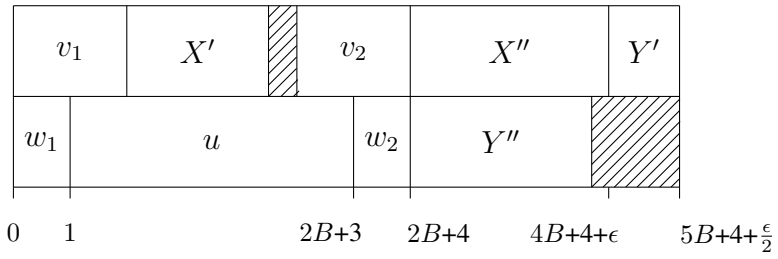


FIGURE 2.4: L'ordonnancement σ dans le deuxième cas (les tâches de Y'' sont séparées par des temps morts).

Nous allons à présent discuter le cas où les hypothèses sur w_i et v_i ($i \in \{1, 2\}$) ne sont pas vérifiées. Si w_i n'est pas ordonnancée en parallèle à v_i, l'intervalle de temps opposé à w_i restera vacant et aucune autre tâche ne pourrait être ordonnancée sur cet intervalle, et c'est le même cas pour v_j. En effet, si v_i n'est pas ordonnancée en parallèle aux tâches w_i et u un temps mort sera généré, ce temps mort ne pourra accueillir aucune autre tâche. Donc, la date de fin de traitement de l'ordonnancement ne décroitra pas si une de ces situations se produit. Pour les deux premiers cas, nous avons $C_{max}(\sigma) > 5B + 4$, donc, peu importe la position de w_i et v_i, la date de fin de traitement de σ restera supérieure à $5B + 4$. Ainsi, $\sum_{j \in X'} p_j = 2B$ (cas 3) est le seul cas pour avoir $C_{max}(\sigma) \leq 5B + 4$.

Donc, l'ensemble A' composé des éléments en correspondance avec les tâches de X' est un sous-ensemble de A tel que $\sum_{a_i \in A'} s(a_i) = B$. □

2.3 Graphes caterpillars

Dans cette section, nous allons présenter un algorithme polynomial pour résoudre le problème SWA sur deux machines identiques lorsque le graphe de concordance est un graphe caterpillar. L'algorithme est basé sur le calcul d'un stable de poids maximum, le stable permettra de répartir les tâches sur les deux machines. Le reste de l'algorithme permet d'affecter les dates de début de traitement aux tâches afin d'obtenir un ordonnancement optimal. La preuve d'optimalité de cet algorithme est explicite puisque le stable de poids maximum constitue une borne inférieure pour l'objectif du problème considéré.

2.3.1 Algorithme d'ordonnancement des graphe caterpillars [23]

Soit un graphe de concordance caterpillar $CAT = (J, E)$, un ordonnancement optimal σ du problème SWA sur deux machines identiques en présence du graphe de concordance CAT est

construit par l'algorithme 1.

Algorithme 1: Algorithme d'ordonnancement des graphes caterpillars

entées : $CAT = (J, E)$, des temps de traitement p_j, $\forall j \in J$;

sortie : Un ordonnancement optimal σ ;

début

  Calcul d'un stable de poids maximum S^* dans le graphe CAT ;

  Suppression de toutes les arêtes qui ne sont pas incidentes à S^* ;

  k parties connexes sont obtenues $CAT_i = (J_i, E_i)$ $(i = 1, \ldots, k)$;

  Pour $i = 1$ **à** k **faire** *Construction de l'ordonnancement σ_i pour CAT_i*

    $S_i^* = J_i \cap S^*$;

    Ordonner les tâches de J_i dans une liste L_i suivant l'ordre de la chaine centrale [†] ;

    Suivant l'ordre induit par L_i, ordonnancer les tâches de S_i^* successivement sur la première machine ;

    Ordonnancer les tâches de $J_i \setminus S_i^*$ sur la deuxième machine comme suit :

      $t(\alpha_0)$=0 [‡] ;

      si α et β appartiennent à $J_i \setminus S_i^*$, et sont consécutives dans cet ordre dans le respect de L_i**alors**

        $t_\beta = max\{t_\alpha + p_\alpha, t(N(\beta))\}$;

      fin si

  fait

  Construire un ordonnancement optimal σ en concaténant tous les ordonnancements σ_i, $i = 1, \ldots, k$;

fin.

Calcul de S^*

Afin de calculer S^*, nous utilisons le programme dynamique présenté par G. H. Chen et al. [9] qui calcule un SPM pour le graphe CAT en $O(n)$. Cette méthode a été conçue pour le calcul d'un SPM pour les arbres, par conséquant la méthode est valide pour les graphes caterpillars.

Ordonner les tâches

Pour arranger les tâches, nous prenons la séquence de tâche de la chaine centrale (qui est naturellement défini pour les graphes caterpillars). Nous ajoutons ensuite chaque tâche de cette

†. La chaine centrale est naturellemet défini pour les graphes caterpiillars.

‡. α_0 est la première tâche de $J_i \setminus S_i^*$ dans la liste L_i.

séquence (dans l'ordre de la chaine centrale) à la liste suivie par ses feuilles (les feuilles sont ajoutées dans n'importe quel ordre).

2.3.2 Exemple

On considère un ensemble $J = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o\}$ de 15 tâches et un graphe de concordance caterpillar $CAT = (J, E)$. Les temps de traitement des tâches sont donnés par la Table 2.2 et le graphe de concordance CAT est décrit par la Figure 2.5. Nous allons résoudre le problème SWA en utilisant *l'algorithme d'ordonnancement des graphes caterpillars* :

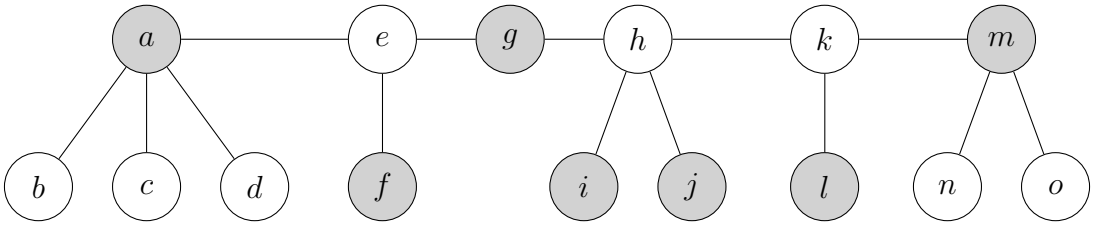


FIGURE 2.5: Graphe de concordance CAT (les sommets gris représentent les tâches de S^*).

TABLE 2.2: Temps de traitement des tâches.

Tâches	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Temps de traitement	5	1	2	1	3	4	2	4	3	2	1	5	4	2	1

1. $S^* = \{a, f, g, i, j, l, m\}$.
2. La seule arête qui relie les tâches de $J \setminus S^*$ est hk que nous allons retirer.
3. Les deux Graphes résultats sont $CAT_1 = (J_1, E_1)$, et $CAT_2 = (J_2, E_2)$ décrits respectivement par les Figures. 2.6 et 2.7.
4. Construction de l'ordonnancement σ_1 qui correspond à CAT_1 :
   (a) $L = (a, b, c, d, e, f, g, h, i, j)$ (La séquence de tâches de la chaine centrale considérée est : (a, e, g, h)).
   (b) $S_1^* = \{a, f, g, i, j\}$.
   (c) Les tâches a, f, g, i et j sont successivement ordonnancées dans cet ordre sur la première machine aux dates de début de traitement respectives $t_a = 0$, $t_f = 5$, $t_g = 9$, $t_i = 11$, et $t_j = 14$.

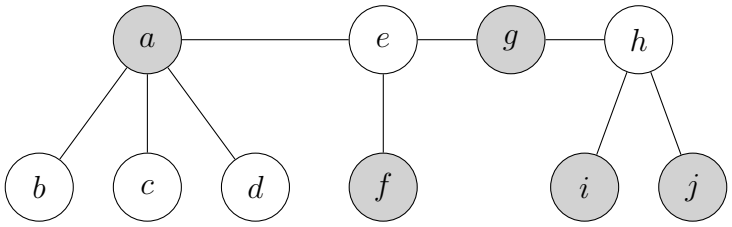


FIGURE 2.6: Graphe de concordance CAT_1 (les sommets en gris représentent les tâche de S_1^*).

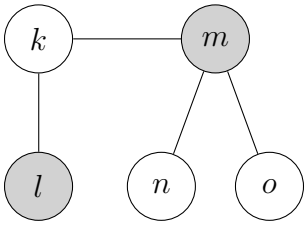


FIGURE 2.7: Graphe de concordance CAT_2 (les sommets en gris représentent les tâches de S_2^*).

(d) Les tâches b, c, d, e et h sont ordonnancées dans cet ordre sur la deuxième machine aux dates de début de traitement respectives $t_b = 0$, $t_c = 1$, $t_d = 3$, $t_e = 4$ et $t_h = 9$.

(e) L'ordonnancement obtenu σ_1 est montré par la Figure 2.8 avec $C_{max}(\sigma_1) = 16$.

5. Construction de l'ordonnancement σ_2 qui correspond à CAT_2 :

   (a) $L = (k, l, m, n, o)$ (La séquence de tâches de la chaine centrale considérée est : (k, m)).

   (b) $S_2^* = \{l, m\}$.

   (c) Les tâches l et m sont ordonnancées successivement dans cet ordre sur la première machine aux dates de début de traitement respectives $t_l = 0$ et $t_m = 5$.

   (d) Les tâches k, n et o sont ordonnancées dans cet ordre dans la deuxième machine aux dates de début de traitement respectives $t_k = 0$, $t_n = 5$ et $t_o = 7$.

   (e) L'ordonnancement obtenu σ_2 est montré par la Figure 2.9 avec $C_{max}(\sigma_2) = 9$.

6. La concaténation des deux ordonnancements σ_1 et σ_2 nous donne l'ordonnancement optimal σ montré par la Figure 2.10 avec $C_{max}(\sigma) = 25$.

2.3.3 Preuves de la polynomialité et de l'optimalité de l'algorithme

Afin de prouver que *l'algorithme d'ordonnancement des graphes caterpillars* construit un ordonnancement optimal en temps polynomial, nous présentons les résultats qui suivent. Etant donné le graphe de concordance CAT_i et l'ordonnancement σ_i ($i = 1, \ldots, k$), nous avons :

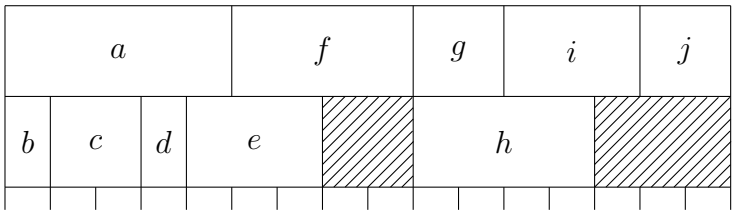


FIGURE 2.8: Ordonnancement σ_1.

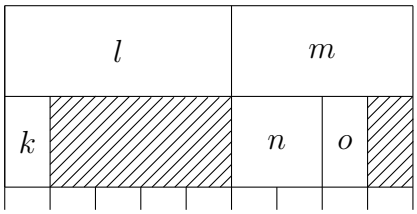


FIGURE 2.9: Ordonnancement σ_2.

Claim 1. *$S_i^* = J_i \cap S^*$ est un SPM pour le graphe de concordance CAT_i.*

Démonstration. On suppose qu'il existe un SPM $I_p(CAT_i)$ tel que $\overline{I_p}(CAT_i) > \sum_{j \in S_i^*} p_j$, on considère également l'ensemble $S' = (S^* \setminus S_i^*) \cup I_p(CAT_i)$. Les ensembles $S^* \setminus S_i^*$ et $I_p(CAT_i)$ n'ont aucune connexion dans CAT. En effet, $(S^* \setminus S_i^*) \cap J_i = \emptyset$, donc, les seules arêtes qui pourraient relier $S^* \setminus S_i^*$ et $I_p(CAT_i)$ sont les arêtes retirées à la deuxième étape de *l'algorithme d'ordonnancement des graphes caterpillars*. Ces arêtes ne sont pas incidentes aux tâches de S^*. Donc, S' est un stable dans CAT. S' a également un poids plus grand que celui de S^*, en effet, puisque $\overline{I_p}(CAT_i) > \sum_{j \in S_i^*} p_j$ nous avons :

$$\sum_{j \in S'} p_j = \overline{I_p}(CAT) - \sum_{j \in S_i^*} p_j + \overline{I_p}(CAT_i) > \overline{I_p}(CAT)$$

S' est un stable de poids supérieur à celui d'un SPM, ce qui constitue une contradiction. □

Claim 2. *$\forall W \subseteq J_i \setminus S_i^*, \sum_{j \in W} p_j \leq \sum_{j \in N(W)} p_j$.*

Démonstration. On suppose qu'il existe un sous-ensemble $W \subseteq J_i \setminus S_i^*$ tel que $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$. On considère aussi un ensemble $S' = (S_i^* \setminus N(W)) \cup W$. Nous avons $N(W) \subseteq S_i^*$ (voir la deuxième étape de *l'algorithme d'ordonnancement des graphes caterpillars*). Donc, S' est un stable dans CAT_i, S' a également un poids supérieur à celui de S_i^*, en effet, puisque $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$ nous avons :

$$\sum_{j \in S'} p_j = \sum_{j \in S_i^*} p_j - \sum_{j \in N(W)} p_j + \sum_{j \in W} p_j > \sum_{j \in S_i^*} p_j$$

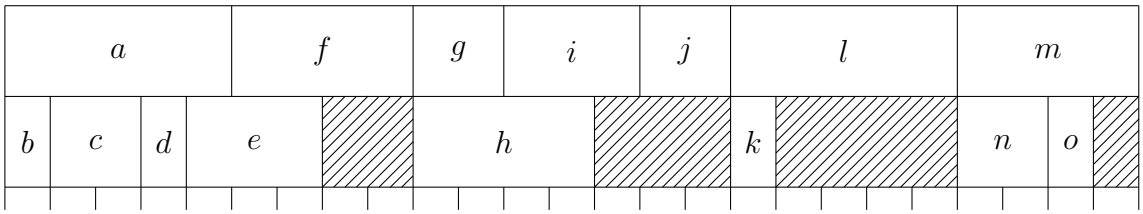


FIGURE 2.10: Ordonnancement optimal σ.

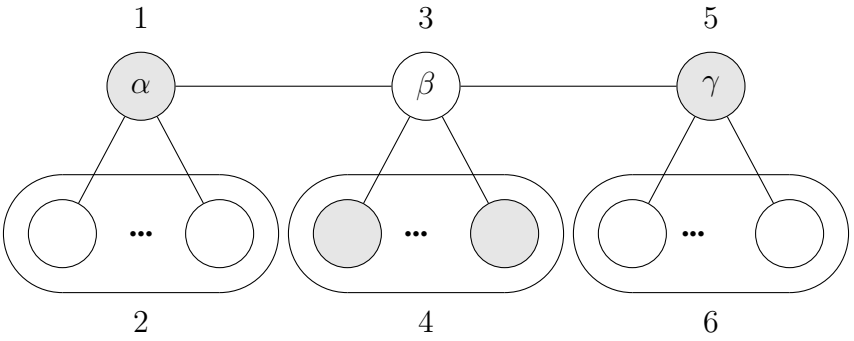


FIGURE 2.11: Illustration des voisins de β et leurs feuilles avec l'ordre dans lequel les tâches entrent dans la liste.

S' est un stable avec un poids supérieur à celui de S_i^*, ce qui constitue une contradiction (Claim 1). □

Claim 3. *$\forall \beta \in J_i \setminus S_i^*$, les voisins de β sont ordonnancées successivement sur la première machine dans l'intervalle de temps $I_\beta = [t(N(\beta)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

Démonstration. Une tâche $\beta \in J_i \setminus S_i^*$ pourrait être une feuille ou non. Si β est une feuille : β aurait un unique voisin α, donc, α est ordonnancée dans l'intervalle de temps $[t_\alpha, t_\alpha + p_\alpha] = I_\beta$.

Si β n'est pas une feuille : β pourrait avoir plusieurs feuilles et un maximum de deux voisins α et γ qui ne sont pas des feuilles. Les tâches entrent la liste L_i dans cet ordre : α (on aurait aussi bien pu prendre γ mais sans perte de généralité, nous avons pris α) et ses feuilles puis β et ses feuilles, et enfin γ et ses feuilles, comme montré sur la Figure 2.11.

Tous les voisins de β appartiennent à S_i^*, donc, ces tâches sont ordonnancées successivement sur la première machine avec l'ordre induit par L_i. Ainsi, les tâches de $N(\beta)$ sont toutes ordonnancées successivement dans l'intervalle de temps $[t(\alpha), t(\alpha) + p_\alpha + \sum_{j \in Lv(\beta)} p_j + p_\gamma] = I_\beta$. □

Claim 4. *$\forall \alpha, \beta \in J_i \setminus S_i^*$ si α et β sont ordonnancées successivement sur la deuxième machine alors α et β ont un voisin en commun.*

Démonstration. Sans perte de généralité, on suppose que α et β sont ordonnancées dans cet ordre. Tous les cas possibles pour α et β sont donnés comme suit :

1. α n'est pas une feuille : Soit γ une tâche adjacente à α qui n'est pas une feuille.
   a. β est la première feuille ordonnancée de $Lv(\gamma)$ (voir Figure 2.12a).
   b. Si $Lv(\gamma) = \emptyset$: β est une autre tâche adjacente à γ (autre que la tâche α) (voir Figure 2.12b).
2. α est une feuille : Soit γ une tâche adjacente à α qui n'est pas une feuille.
   a. β est une feuille voisine à γ qui est ordonnancée juste après α (voir Figure 2.12c) ;
   b. Si α est la dernière tâche ordonnancée parmi les tâches de $Lv(\gamma)$: β est le prochain voisin de γ qui n'est pas une feuille (voir Figure 2.12d).

Dans tous les cas possibles, α et β ont γ comme voisin en commun. □

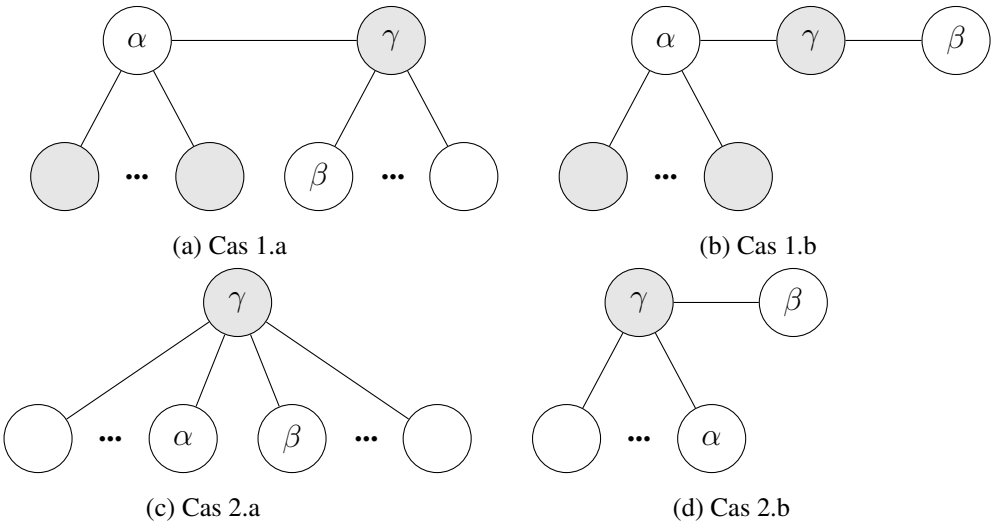


FIGURE 2.12: Tous les cas possibles pour deux tâches successives α et β ordonnancées sur la deuxième machine (les tâches en gris représentent les tâches de S_i^*).

Claim 5. *Si $W = \{\alpha, \ldots, \beta\}$ est un ensemble de tâches successives de $J_i \setminus S_i^*$ alors les tâches de $N(W)$ sont ordonnancées sur la deuxième machine dans l'intervalle de temps $I_W = [t(N(\alpha)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

Démonstration. Conséquence des Claim 3 et Claim 4. □

Lemme 1. *Chaque tâche $\beta \in J_i \setminus S_i^*$ est ordonnancée dans l'intervalle de temps $I_\beta = [t(N(\beta)), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$.*

Démonstration. On suppose qu'il existe une tâche $\beta \in J_i \setminus S_i^*$ qui n'est pas ordonnancée dans l'intervalle I_β. β n'est pas ordonnancée dans cet intervalle implique que : $t_\beta < t(N(\beta))$ ou $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$. Soit β_0 la tâche qui précède β sur la deuxième machine.

1. $t_\beta < t(N(\beta))$: β à une date de début de traitement $t_\beta = max\{t_{\beta_0} + p_{\beta_0}, t(N(\beta))\}$, donc, nous ne pouvons pas avoir $t_\beta < t(N(\beta))$.
2. $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$: Etant donné que $t_\beta = max\{t_{\beta_0} + p_{\beta_0}, t(N(\beta))\}$ on distingue deux cas :
   (a) $t_\beta = t_{\beta_0} + p_{\beta_0}$: Soit α la dernière tâche ordonnancée sur la deuxième machine tel que α précède β et $t_\alpha = t(N(\alpha))$. On considère également $W = \{\alpha, \ldots, \beta\}$ l'ensemble de tâches consécutivement ordonnancées sur la deuxième machine de α à β. il n'y a pas de temps mort entre les tâches de W, donc, les tâches de W sont ordonnancées sur l'intervalle de temps $[t_\alpha, t_\beta + p_\beta]$. les tâches de $N(W)$ sont ordonnancées sur l'intervalle de temps $[t(N(\alpha), t(N(\beta)) + \sum_{j \in N(\beta)} p_j]$ (Claim 5), comme montré par la Figure 2.13. Etant donné que $t_\alpha = t(N(\alpha))$ et $t_\beta + p_\beta > t(N(\beta)) + \sum_{j \in N(\beta)} p_j$, nous avons $\sum_{j \in W} p_j > \sum_{j \in N(W)} p_j$ ce qui constitue une contradiction selon Claim 2.
   (b) $t(\beta) = t(N(\beta))$: ce cas se rapporte au cas 2.a où $\alpha = \beta$ et $W = \{\beta\}$.

□

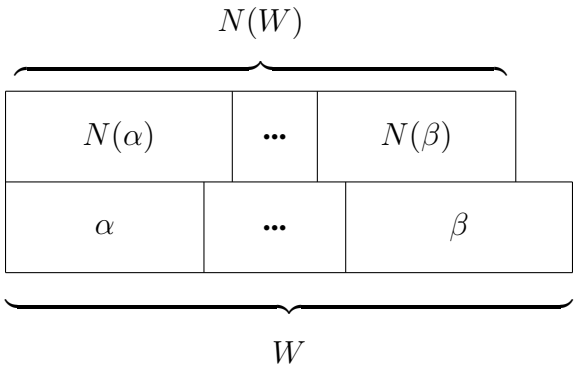


FIGURE 2.13: Ordonnancement des tâches de W et $N(W)$.

Théorème 2. *([23, 24]) l'algorithme d'ordonnancement des graphes caterpillars procure un ordonnancement optimal σ pour le problème SWA sur deux machines identiques en présence du graphe de concordance CAT en $O(n)$ avec $C_{max}(\sigma) = \overline{I_p}(CAT)$.*

Démonstration. Le calcul d'un SPM pour un graphe caterpillar (étape 1) se fait en $O(n)$ (voir [9]) et la détermination des parties connexes d'un graphe caterpillar (étape 3) se fait en $O(n)$

(voir [17]). La complexité de *l'algorithme d'ordonnancement des graphes caterpillars*est donc trivialement $O(n)$.

Pour $i = 1, \ldots, k$, l'ordonnancement σ_i est réalisable et $C_{max}(\sigma_i) \leq \overline{I_p}(CAT_i)$ (conséquence directe du Lemme 1). Etant donné que l'ordonnancement σ est obtenu par la concaténation des ordonnancements σ_i, $i = 1, \ldots, k$, nous avons :

$$C_{max}(\sigma) = \sum_{i=1}^{k} \sigma_i \leq \sum_{i=1}^{k} \sum_{j \in S_i^*} p_j \leq \sum_{j \in S^*} p_j = \overline{I_p}(CAT)$$

$\overline{I_p}(CAT)$ est une borne inférieure sur la date de fin de traitement de l'ordonnancement optimal. Donc, *l'algorithme d'ordonnancement des graphes caterpillars* procure un ordonnancement optimal σ, en $O(n)$, pour le problème SWA sur deux machines en présence d'un graphe de concordance caterpillar CAT tel que $C_{max}(\sigma) = \overline{I_p}(CAT)$. □

Corolaire 1. *([23, 24]) Le problème SWA sur deux machines identiques lorsque le graphe de concordance est une chaine ou une étoile peut être résolu en temps linéaire en utilisant l'algorithme d'ordonnancement des graphes caterpillars.*

Démonstration. Conséquence du Théorème 2. En effet, les chaines et les étoiles sont des sous-classes des graphes caterpillars. □

2.4 Cycles

Etant donné un ensemble de tâches J et un cycle élémentaire $C = (J, E)$, un ordonnancement optimal pour le problème SWA en présence du graphe de concordance C sur deux machines identiques se construit comme décrit dans *l'algorithme d'ordonnancement de cycles*.

Claim 6. *Dans l'ordonnancement σ, il existe au moins une tâche j tel que j n'est pas ordonnancée simultanément à ses deux voisins.*

Démonstration. On suppose que chaque tâche $j \in J$ est ordonnancée à l'opposé de ses deux tâches voisines. Ceci implique que les tâches sont ordonnancées sur un axe circulaire ce qui est impossible. □

Corolaire 2. *([23, 24]) l'algorithme d'ordonnancement de cycles construit un ordonnancement optimal pour le problème SWA sur deux machines identiques lorsque le graphe de concordance est un cycle en $O(n^2)$.*

Algorithme 2: Algorithme d'ordonnancement de cycles

entrées : $C = (J, E)$, temps de traitement p_j, $\forall j \in J$;

sortie : Ordonnancement optimal σ ;

1 **début**

2 **pour chaque** $e \in E$ **faire**

3 Construire un ordonnancement optimal pour la chaine $P = (J, E - e)$ en utilisant *l'algorithme d'ordonnancement des graphes Caterpillars* ;

4 **fait**

5 σ est l'ordonnancement avec la meilleure date de fin de traitement ;

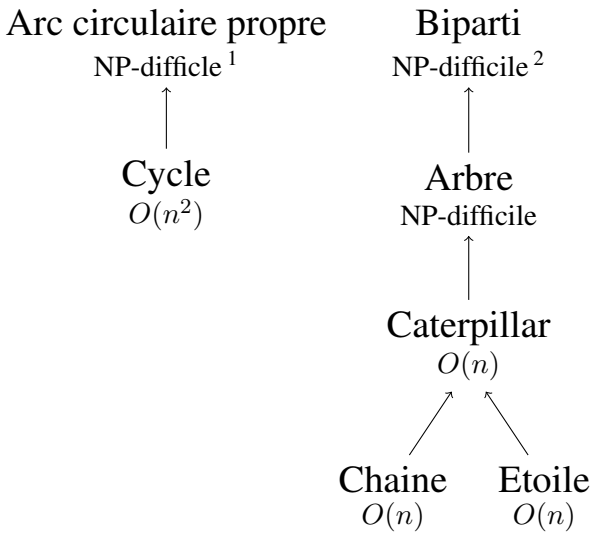
6 **fin.**

Démonstration. Afin de construire un ordonnancement réalisable, il est impossible d'utiliser toutes les relations de concordance du cycle (Selon Claim 6). Donc, résoudre le problème SWA lorsque le graphe de concordance est un cycle équivaut à résoudre le problème en retirant l'arête "non utilisée" du cycle. Retirer à chaque fois une arête différente [1], et résoudre le problème SWA pour chacun des chaines résultants, procure un ordonnancement optimal σ. *L'algorithme d'ordonnancement de cycles* utilise n fois *l'algorithme d'ordonnancement des graphes caterpillars*, la complexité de l'algorithme est donc $O(n^2)$. □

2.5 Conclusion

Dans ce chapitre, nous avons considéré le problème SWA sur deux machines identiques. Nous avons montré que le problème est NP-difficile lorsque le graphe de concordance est un arbre. Des algorithmes polynomiaux ont été proposé afin de résoudre le problème lorsque le graphe de concordance est soit un graphe caterpillar ou un cycle. La cartographie dans la Figure 2.14 récapitule les résultats présentés dans ce chapitre.

1. Enumérer tous les cas possibles.



[1] [15] ont prouvé La NP-difficulté d'un problème équivalent au problème SWA lorsque le graphe de concordance est un graphe complet ce qui est un sous-problème du problème lorsque le graphe de concordance est un graphe arc circulaire propre.

[2] [12] et [2] ont prouvé son NP-difficulté.

FIGURE 2.14: Cartographie de la complexité des cas étudiés

Chapitre 3

Machines uniformes

3.1 Introduction

On considère le problème d'ordonnancement sur deux machines uniformes. Le problème consiste à ordonnancer un ensemble J de n tâches non préemptibles sur deux machines uniformes tout en respectant les contraintes de concordance. Etant donné un nombre rationnel α, $0 < \alpha < 1$, une tâche j a un temps de traitement :

$$p_j = \begin{cases} 1 & \text{sur la première machine} \\ \alpha & \text{sur la deuxième machine} \end{cases}$$

Dans ce chapitre, nous allons tout d'abord prouver que le problème sur deux machines uniformes est NP-difficile au sens fort pour toutes les valeurs de α (le problème a déjà été prouvé NP-difficile lorsque $\alpha = \frac{1}{2}$ [5]). La preuve se fera en réduisant le problème DSWA (Problème de décision associé au problème SWA) sur deux machines uniformes au problème de partition d'un graphe en sous-graphes isomorphiques. Nous allons présenter un modèle mathématique, un algorithme Simulated Annealing, et une méthode Branch and Bound pour la résolution exacte du problème. Ce chapitre se conclura par une partie expérimentale.

3.2 NP-Difficulté du problème

Afin de prouver que le problème SWA sur deux machines uniformes est NP-difficile, nous allons proposer quelques définitions et notations. Etant donné deux entiers a et b premiers entre eux tel que $\alpha = \frac{a}{b}$ $(b > a)$, et J un ensemble de $a + b$ tâches, nous avons :

– σ_{ab} est un ordonnancement des tâches de J. L'ordonnancement est construit comme suit : a tâches sont ordonnancées successivement sur la première machine et les b tâches restantes sont ordonnancées successivement sur la deuxième machine. La date de fin de traitement de l'ordonnancement est $C_{max}(\sigma_{ab}) = a$ (Figure 3.1).

– Le graphe de concordance $G_{ab} = (J', E')$ est défini comme suit : $J' = J$ et deux tâches j et k sont connectées dans G_{ab} si et seulement si ces tâches sont ordonnancées simultanément dans l'ordonnancement σ_{ab}.

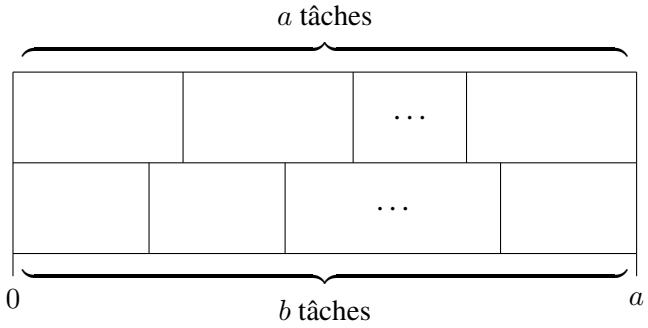


FIGURE 3.1: Ordonnancement σ_{ab}

Claim 1. *Il est possible de passer de l'ordonnancement σ_{ab} au graphe de concordance G_{ab} et inversement en temps polynômial.*

Démonstration. À partir de l'ordonnancement σ_{ab}, le graphe G_{ab} se construit en reliant chaque pair de tâches ordonnancée simultanément dans σ_{ab}.

Avant de montrer le passage du graphe G_{ab} à l'ordonnancement σ_{ab}, nous présentons les trois propositions suivantes :

1. Une tâche avec un degré unitaire est une tâche qui est ordonnancée sur la deuxième machine. En effet, comme l'ordonnancement σ_{ab} ne contient de temps mort et que les temps de traitement sur la première machine sont supérieurs aux temps de traitement sur la deuxième machine, il est impossible à une tâche de la première d'être ordonnancée simultanément à une seule tâche.
2. Deux tâche j et k ordonnancée successivement sur la première machine sont reliées à une même tâche l ordonnancée sur la deuxième machine simultanément aux tâches j et k. Car, si la tâche l n'existe pas alors la tâche j serait ordonnancée simultanément à une tâche l_1 avec $c_j = c_{l_1}$ et la tâche k serait ordonnancée simultanément à une tâche l_2 avec $t_k = t_{l_2}$. On considère les a' premières tâches de la première machine jusqu'à la tâche

j et les b' première tâches de la deuxième machine jusqu'à la tâche l_2. Cette séquance de tâches termine son traitement à la date a'. Les temps de traitement sur la deuxième machine sont :

$$\alpha b' = a' \Rightarrow \alpha = \frac{a'}{b'}$$

Ce qui représente une conclusion puisque a et b sont premiers entre eux on ne devrait donc pas avoir $\frac{a}{b} = \frac{a'}{b'}$ avec $a' < a$ et $b' < b$.

Maintenant nous allons montrer comment construire l'ordonnancement σ_{ab} à partir de du graphe G_{ab} :

- Lorsque $a = 1$: le graphe G_{ab} est une étoile : une tâche j reliée à b tâches deux à deux indépendantes. L'ordonnancement σ_{ab} est construit en ordonnançant la tâche j sur la première machine et les b tâches restantes sur la deuxième.
- Lorsque $a > 1$: le graphe G_{ab} est soit une chaine ou un graphe caterpillar.
  - Si G_{ab} est une chaine : les deux extrémités avec un degré unitaire sont la première et la dernière tâche ordonnancées sur la deuxième machine (voir 1). Les tâches sont prises dans l'ordre de la chaine en ordonnancement l'extrémité sur la deuxième machine les tâches restantes sont ordonnancées en alternance entre la première et deuxième machine (voir 2).
  - Si G_{ab} est un graphe caterpillar : la chaine centrale est composée des tâches ayant un degré supérieur ou égal à 2. Les tâches de la chaine centrale sont prises dans l'ordre. La première tâche de la chaine centrales est ordonnancée sur la première machine ses feuilles voisines sont ensuite ordonnancées sur la deuxième (voir 1). Les tâches de la chaine centrale sont ordonnancées en alternance entre la première et deuxième machine (voir 2). Si une la tâche de la chaine centrale est ordonnancée sur la première machine ordonnancer ces feuilles voisines (si elles existent) sur la deuxième machine.

□

Problème de partition d'un graphe en sous-graphes isomorphiques
Instance : un entier positif q et deux graphes $G = (V, E)$ et $H = (V', E')$ avec $|V'| \geq 3$ et $|V| = q|V'|$.
Question : Existe-il une partition des sommets du graphe G en q ensembles disjoints V_1,..., V_q tel que le sous-graphe induit par les sommets de V_i ($i = 1, \ldots, q$) a le même nombre de sommets que H et contient un graphe isomorphique à H ? Ce problème est NP-complet (Garey et Johnson [15]).

Théorème 3. *([22]) Le problème d'ordonnancement avec graphe de concordance sur deux machines uniformes est NP-difficile au sens fort.*

Démonstration. On considère le problème de partition d'un graphe en sous-graphes isomorphiques définit plus haut. En prenant $H = G_{ab}$, on peut prouver que ce problème peut être réduit polynomialement au problème (P) suivant. Instance : Un ensemble J de $q(a+b)$ tâches, deux machines uniformes et un graphe de concordance $A = (J, E'')$. Chaque tâche $j \in J$ correspond à un sommet $v \in V$, et deux tâches sont adjacentes dans A si et seulement si les sommets qui leurs correspondent sont adjacents dans G. Les temps de traitement des tâches de J valent 1 sur la première machine et $\frac{a}{b}$ sur la deuxième. Question : Existe-il un ordonnancement réalisable σ avec $C_{max}(\sigma) = qa$?

D'abord, nous supposons que le problème de la partition d'un graphe en sous-graphes isomorphiques admet une solution. Soit G_i le sous-graphe de G induit par les sommets de V_i $(i = 1, \ldots, q)$. On considère A_i un sous-graphe de A tel que A_i correspond au graphe isomorphique de G_{ab} contenu dans G_i $(i = 1, \ldots, q)$ i.e. de la même manière que le graphe de concordance A correspond au graphe G. A partir du graphe A_i, il est possible de construire l'ordonnancement σ_i, similaire à l'ordonnancement σ_{ab} (voir Claim 1) avec $C_{max}(\sigma_i) = a$ $(i = 1, \ldots, q)$. Si nous faisons la concaténation des ordonnancements $\sigma_1, \ldots, \sigma_q$ on obtient l'ordonnancement σ (voir Figure 3.1), avec $C_{max}(\sigma) = qa$, ce qui représente une solution au problème (P).

Inversement, on suppose que le problème (P) admet une solution. Dans l'ordonnancement σ (solution du problème (P)) il n'y a aucun temps mort, en effet, un ordonnancement sans temps mort aurait une date de fin de traitement égale à $\frac{q(a+b)}{1+\frac{b}{a}} = qa$ ce qui représente la date de fin de traitement de σ. Les a premières tâches ordonnancées sur la première machine et les b premières tâches ordonnancées sur la deuxième machine terminent leur traitement à la date a. Cette séquence de $a+b$ tâches, telle qu'elle est ordonnancée dans σ, forme un ordonnancement similaire à σ_{ab}. Cette séquence de tâche se répète q fois chaque a unité de temps (voir Figure 3.2). Nous pouvons construire les ordonnancements $\sigma_1, \ldots, \sigma_q$ à partir de chacune de ces séquences de $a + b$ tâches, et à partir de chaque ordonnancement σ_i nous pouvons construire le graphe de concordance A_i qui lui correspond (voir Claim 1), isomorphique à G_{ab}. On considère les sous-graphes $G_1 = (V_1, E_1), \ldots, G_q = (J_q, E_q)$ (sous-graphe de G) tel que G_i correspond au graphe de concordance A_i. Les ensembles $V_1, \ldots, V_q$ forment une partition du graphe G tel que le sous-graphe induit par V_i contient un graphe isomorphique à G_{ab} (puisque G_i correspond à A_i, et A_i est isomorphique à G_{ab}), ce qui représente une solution au problème de partition du graphe G en sous-graphes isomorphiques. □

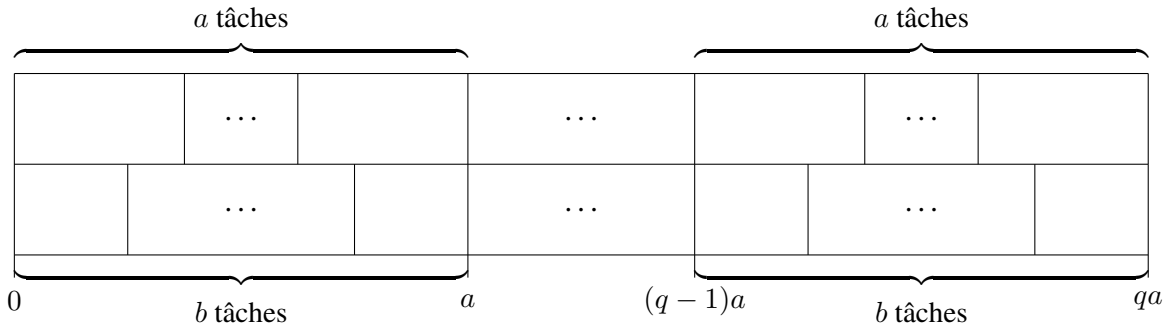


FIGURE 3.2: Ordonnancement σ

3.3 Quelques cas polynomiaux

Propriété 1. *Soient un ensemble de tâches J avec $|J| \leq \lfloor \frac{1}{\alpha} \rfloor$, ainsi que deux ordonnancements des tâches de J : σ_1 et σ_2.*

- *L'ordonnancement σ_1 contient une seule tâche $j \in J$ ordonnancée sur la première machine simultanément à un maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$ tâches.*

- *Dans l'ordonnancement σ_2 toutes les tâches sont ordonnancées sur la deuxième machine.*

Pour les deux ordonnancements σ_1 et σ_2 nous avons :

$$C_{max}(\sigma_2) \leq C_{max}(\sigma_1)$$

Démonstration. La date de fin de traitement de l'ordonnancement σ_1 vaut 1, en effet, le temps de traitement des tâches ordonnancées sur la deuxième machine vaut $\alpha(\lfloor \frac{1}{\alpha} \rfloor - 1) < 1$. Tandis que la date de fin de traitement de l'ordonnancement σ_b ne dépasse pas $\alpha \lfloor \frac{1}{\alpha} \rfloor \leq 1$. Ce qui prouve que $C_{max}(\sigma_2) \leq C_{max}(\sigma_1)$. □

Claim 2. *Le problème SWA sur deux machines uniformes peut être résolu en temps polynomial lorsque $\Delta < \lfloor \frac{1}{\alpha} \rfloor$ ou $n < \lfloor \frac{1}{\alpha} \rfloor + 1$.*

Démonstration. Une tâche ordonnancée sur la première machine peut être ordonnancée simultanément à un maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$ autres tâches. Et selon la Propriété 1, ordonnancer toutes les tâches sur la deuxième machine procure un ordonnancement optimal. □

Claim 3. *Le problème SWA sur deux machines uniformes peut être résolu en temps polynomial lorsqu'il existe une unique tâche j avec $d(j) \geq \lfloor \frac{1}{\alpha} \rfloor$.*

Démonstration. Soit J l'ensemble des tâches à ordonnancer. A l'exception de la tâche $j \in J$, les autres tâches peuvent s'ordonnancer simultanément à un maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$ tâches (puisque $\forall k \in J,\ k \neq j : d(k) < \lfloor \frac{1}{\alpha} \rfloor$), et selon la propriété 1 une telle configuration procurerait un ordonnancement avec un temps de traitement plus important. Donc, mis à part la tâche j les autres tâches doivent être ordonnancées sur la deuxième machine.

On distingue deux ordonnancements possibles : σ_1 lorsque j est ordonnancée sur la deuxième machine et σ_2 lorsque j est ordonnancée sur la première machine :

1. σ_1 : toutes les tâches seront ordonnancées sur la deuxième machine, ce qui nous donne une date de fin de traitement : $C_{max}(\sigma_1) = \alpha n$.
2. σ_2 : lorsque la tâche j est ordonnancée sur la première machine on peut distinguer deux autres cas :
   (a) $d(j) = \lfloor \frac{1}{\alpha} \rfloor$: la tâche j sera ordonnancée simultanément à $\lfloor \frac{1}{\alpha} \rfloor$ tâches, car sinon (si la tâche j est ordonnancée à moins de $\lfloor \frac{1}{\alpha} \rfloor$ tâches) l'ordonnancement engendré aurait une date de fin de traitement plus importante. La date de fin de traitement de cet ordonnancement noté $\sigma_{2.a}$ est :

$$\begin{aligned} C_{max}(\sigma_{2.a}) &= \alpha(n - \lfloor \tfrac{1}{\alpha} \rfloor - 1) + max\{\alpha \lfloor \tfrac{1}{\alpha} \rfloor, 1\} \\ &= (\alpha n - \alpha \lfloor \tfrac{1}{\alpha} \rfloor - \alpha) + 1 \\ &= \alpha n + 1 - (\alpha \lfloor \tfrac{1}{\alpha} \rfloor + \alpha) \leq \alpha n = C_{max}(\sigma_1) \end{aligned}$$

   En effet, nous avons :

$$\begin{aligned} \lfloor \tfrac{1}{\alpha} \rfloor \geq \tfrac{1}{\alpha} - 1 &\Rightarrow \alpha \lfloor \tfrac{1}{\alpha} \rfloor \geq 1 - \alpha \\ &\Rightarrow \alpha \lfloor \tfrac{1}{\alpha} \rfloor + \alpha \geq 1 \end{aligned}$$

   (b) $d(j) \geq \lfloor \frac{1}{\alpha} \rfloor$: la tâche j sera ordonnancée simultanément à $\lceil \frac{1}{\alpha} \rceil$ tâches et la date de fin de traitement de cet ordonnancement noté $\sigma_{2.b}$ est :

$$\begin{aligned} C_{max}(\sigma_{2.b}) &= \alpha(n - \lceil \tfrac{1}{\alpha} \rceil - 1) + max\{\alpha \lceil \tfrac{1}{\alpha} \rceil, 1\} \\ &= (\alpha n - \alpha \lceil \tfrac{1}{\alpha} \rceil - \alpha) + \alpha \lceil \tfrac{1}{\alpha} \rceil \\ &= \alpha n - \alpha < C_{max}(\sigma_{2.a}) \leq \alpha n \end{aligned}$$

   En effet, nous avons $C_{max}(\sigma_{2.b}) < C_{max}(\sigma_{2.a})$ puisque :

$$\begin{aligned} C_{max}(\sigma_{2.a}) &= \alpha n - \alpha \lfloor \tfrac{1}{\alpha} \rfloor - \alpha + 1 \\ &= \alpha n - \alpha + (1 - \alpha \lfloor \tfrac{1}{\alpha} \rfloor) \\ &> \alpha n - \alpha = C_{max}(\sigma_{2.b}) \end{aligned}$$

Donc, ordonnancer la tâche j sur la première machine simultanément à un maximum de tâches possibles, et le reste des tâches sur la deuxième machine nous procure un ordonnancement optimal. □

Claim 4. *Le problème SWA sur deux machines uniformes peut être résolu en temps polynomial lorsque* $n \leq \lfloor \frac{2}{\alpha} \rfloor + 1$.

Démonstration. Lorsque $n < \lfloor \frac{1}{\alpha} \rfloor + 1$, toutes les tâches sont ordonnancées sur la deuxième machine, puisqu'une tâche ordonnancée sur la première machine peut s'ordonnancer simultanément à un maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$ tâches (voir Claim 2). Lorsque le nombre de tâches est supérieur ou égale à $\lfloor \frac{1}{\alpha} \rfloor + 1$, on distingue trois cas :

1. $\Delta < \lfloor \frac{1}{\alpha} \rfloor$: dans ce cas toutes les tâches sont ordonnancées sur la deuxième machine (voir Claim 2)
2. $\Delta = \lfloor \frac{1}{\alpha} \rfloor$: une tâche j avec $d(j) = \lfloor \frac{1}{\alpha} \rfloor$ sera ordonnancée sur la première machine simultanément à ses $\lfloor \frac{1}{\alpha} \rfloor$ tâches voisines. Les tâches restantes sont au nombre maximum de $\lfloor \frac{1}{\alpha} \rfloor$.Si une tâche parmi les tâches restantes est ordonnancée sur la première machine, celle-ci sera au mieux ordonnancée simultanément à $\lfloor \frac{1}{\alpha} \rfloor - 1$ tâches, or ordonnancer les tâches restantes sur la deuxième machine résulterait un ordonnancement avec une date de fin de traitement plus petite (Propriété 1).
3. $\Delta \geq \lceil \frac{1}{\alpha} \rceil$:
   (a) Si $\frac{1}{\alpha} \in N$: une tâche j avec $d(j) \geq \frac{1}{\alpha}$ est ordonnancée sur la première machine simultanément à ses $\frac{1}{\alpha}$ tâches voisines. Les tâches restantes (au nombre maximum de $\frac{1}{\alpha} - 1$) sont ordonnancée sur la deuxième machine (Claim 2).
   (b) Si $\frac{1}{\alpha} \notin N$: la tâche j avec $d(j) = \lceil \frac{1}{\alpha} \rceil$ est ordonnancée sur la première machine simultanément à ses $\lceil \frac{1}{\alpha} \rceil$ tâches voisines. Les tâches restantes sont au nombre maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$. La dernière tâche ordonnancée sur la deuxième machine est ordonnancée partiellement à l'opposé de la tâche j. Même en comptabilisant cette tâche, si une tâche, parmi les tâches restantes, serait ordonnancée sur la première machine, celle-ci pourra être ordonnancée simultanément à un maximum de $\lfloor \frac{1}{\alpha} \rfloor - 1$ tâches. Or, ordonnancer les tâches restantes sur la deuxième machine donnerait un ordonnancement optimal (voir Claim 2).

□

Claim 5. *Le problème SWA sur deux machines uniformes peut être résolu en temps polynomial lorsque le graphe de concordance est une clique.*

Démonstration. Lorsque le graphe de concordance est une clique, le problème devient équivalent au problème d'ordonnancement sur deux machines uniformes qui peut être résolu en $O(n)$ [11]. □

3.4 Modèle mathématique

Dans cette section, nous présentons un modèle mathématique [22] pour modéliser le problème SWA sur deux machines uniformes. L'idée vient du fait de pouvoir considérer n'importe quel ordonnancement réalisable et de pouvoir le diviser en "sous-ordonnancements" ou compartiments. Dans chaque compartiment il existe deux contraintes à respecter :

1. Aucun temps mort ne sépare deux tâches ordonnancées sur la même machine du même compartiment.

2. Chaque compartiment possède a positions pouvant accueillir jusqu'à a tâches sur la première machine et b positions pouvant accueillir un maximum de b tâches sur la deuxième machine.

Ainsi, au lieu de construire un seul ordonnancement pour l'ensemble des tâches, nous allons construire n' sous-ordonnancements avec aucun temps mort entre les tâches. L'ordonnancement global sera obtenu en concaténant les n' sous-ordonnancements.

Exemple

On considère un ensemble $J = \{a, b, c, d, e, f, g, h\}$ de 8 tâches, ainsi que deux machines uniformes. Le temps de traitement sur la deuxième machine $\alpha = 0,5$. a et b sont les deux nombres premiers entre eux tel que $\frac{a}{b} = \alpha = 0,5 = \frac{1}{2}$, les valeurs de a et b sont donc respectivement 1 et 2. Soit σ un ordonnancement réalisable des tâches de J illustré par la Figure 3.3.

L'ordonnancement σ peut être divisé en 3 compartiments. Le premier compartiment représente le sous-ordonnancement contenant les tâches a et b. Le deuxième compartiment représente le sous-ordonnancement des tâches c, d et e (ce compartiment ne peut pas accueillir plus de tâches puisque le compartiment a atteint le nombre maximum de tâches permis $a = 1$ tâche sur la première machine et $b = 2$ sur la deuxième). Le troisième et dernier compartiment est le sous-ordonnancement des tâches f, g et h.

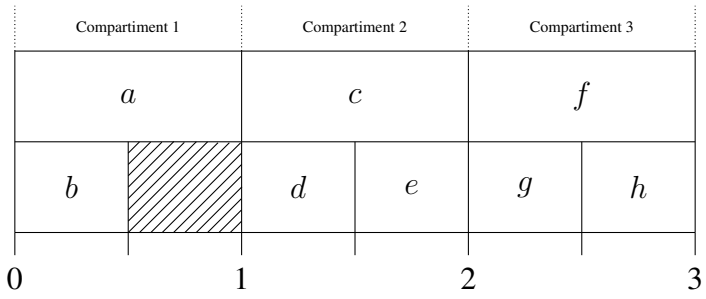


FIGURE 3.3: Exemple de la subdivision d'ordonnancements en compartiments.

Dans la Section 3.4.1 nous allons présenter toutes les variables qui vont nous permettre de modéliser le problème sous forme d'un modèle mathématique. Un modèle mathématique consiste en une fonction objectif et des contraintes que nous allons présenter dans les sections 3.4.2 et 3.4.3.

3.4.1 Variables

On note le nombre de compartiments par n'. Pour $i = 1, \ldots, n$, $j = 1, \ldots, a$ et $k = 1, \ldots, n'$, on considère les variables suivantes :

$$x_{ijk} = \begin{cases} 1 & \text{si la tâche } i \text{ est ordonnancée sur la machine 1 à la position } j \\ & \text{du compartiment } k \\ 0 & \text{sinon} \end{cases}$$

Pour $i = 1, \ldots, n$, $j = 1, \ldots, b$ et $k = 1, \ldots, n'$ nous avons :

$$y_{ijk} = \begin{cases} 1 & \text{si la tâche } i \text{ est ordonnancée sur la machine 2 à la position } j \\ & \text{du compartiment } k \\ 0 & \text{sinon} \end{cases}$$

c_k : est la date du fin de traitement du compartiment k, $k = 1 \ldots, n'$.

3.4.2 Fonction Objectif

L'objectif du problème est de minimiser la date de fin de traitement de l'ordonnancement. Etant donné que l'ordonnancement global est obtenu en concaténant les différents compartiments que nous allons construire, minimiser la date de fin de traitement de l'ordonnancement revient à minimiser la somme des dates de fin de traitement de tous les compartiments, et cela est exprimé par la fonction objectif qui suit :

$$min \sum_{k=1}^{n'} c_k$$

3.4.3 Contraintes

1. Ces premières contraintes garantissent que toutes les tâches soient ordonnancées, tout en s'assurant que chaque tâche soit ordonnancée sur une seule machine, une seule position d'un unique compartiment, afin d'éviter d'avoir une même tâche ordonnancée sur deux emplacements différents.

$$\sum_{j=1}^{a} \sum_{k=1}^{n'} x_{ijk} + \sum_{j=1}^{b} \sum_{k=1}^{n'} y_{ijk} = 1 \qquad i = 1, \ldots, n$$

2. Afin d'éviter tout chevauchement entre des tâches ordonnancées sur une même machine, les contraintes suivantes assurent que des tâches différentes ne soient pas affectées à une même position d'un même compartiment.

$$\begin{aligned} \sum_{i=1}^{n} x_{ijk} \leq 1 \qquad & j = 1, \ldots, a \\ & k = 1, \ldots, n' \\ \sum_{i=1}^{n} y_{ijk} \leq 1 \qquad & j = 1, \ldots, b \\ & k = 1, \ldots, n' \end{aligned}$$

3. Comme décrit plus haut, une des contraintes présentes dans chaque compartiment est qu'il n'y ait pas de temps mort entre les tâches. Ces contraintes peuvent être exprimées de la manière suivante : chaque position avec aucune tâche assignée (position vide ou temps mort) ne peut être suivie par une position avec une tâche assignée. Cela permettra, non seulement, d'éviter d'avoir des temps mort entre les tâches de chaque compartiment, mais aussi, d'éliminer certains ordonnancements "inefficaces" ou redondants. i.e. ordonnancer une séquence de tâches à la position 1 donnera toujours un meilleur ordonnancement que l'ordonnancement de la même séquence de tâches après une ou plusieurs positions vides.

$$\begin{aligned} \sum_{i=1}^{n} x_{ijk} \geq \sum_{i=1}^{n} x_{ij+1k} \qquad & j = 1, \ldots, a-1 \\ & k = 1, \ldots, n' \\ \sum_{i=1}^{n} y_{ijk} \geq \sum_{i=1}^{n} y_{ij+1k} \qquad & j = 1, \ldots, b-1 \\ & k = 1, \ldots, n' \end{aligned}$$

4. Etant donné qu'il n'y a pas de temps mort avant une tâche assignée (voir les contraintes précédentes), il est possible de déterminer la position des tâches qui sont ordonnancées à

l'opposé de n'importe quelle tâche ordonnancée sur la première machine. Dans un même compartiment, la tâche ordonnancée sur la première machine à la position 1 est ordonnancée simultanément aux tâches ordonnancées sur la deuxième machine entre les positions 1 et $\lceil \frac{b}{a} \rceil$. Les contraintes suivantes assurent que la tâche ordonnancée sur la première machine à la position 1 soit bien ordonnancée simultanément à des tâches qui lui sont concordantes.

$$\begin{aligned} x_{i1k} + y_{i'jk} \leq 1 + a_{ii'} \qquad & i = 1, \ldots, n \\ & i' = 1, \ldots, n \\ & j = 1, \ldots, \lceil \tfrac{b}{a} \rceil \\ & k = 1, \ldots, n' \end{aligned}$$

5. Les contraintes suivantes assurent que chaque tâche ordonnancée sur la première machine à une position $j \geq 2$ d'un compartiment k soit ordonnancée simultanément à des tâches qui lui sont concordantes. En effet, dans un même compartiment chaque tâche ordonnancée sur la première machine à une position $j \geq 2$ est ordonnancée simultanément aux tâches ordonnancées sur la deuxième machine entre les positions $\lceil \frac{(j-1)b}{a} \rceil$ et $\lceil \frac{jb}{a} \rceil$.

$$\begin{aligned} x_{ijk} + y_{i'j'k} \leq 1 + a_{ii'} \qquad & i = 1, \ldots, n \\ & i' = 1, \ldots, n \\ & j = 2, \ldots, b \\ & j' = \lceil \tfrac{(j-1)b}{a} \rceil, \ldots, \lceil \tfrac{jb}{a} \rceil \\ & k = 1, \ldots, n' \end{aligned}$$

6. La date de fin de traitement du compartiment k est égale au maximum entre le temps de traitement total sur la première et la deuxième machine.

$$\begin{aligned} \sum_{i=1}^{n} \sum_{j=1}^{a} a x_{ijk} \leq c_k \qquad & k = 1, \ldots, n' \\ \sum_{i=1}^{n} \sum_{j=1}^{b} b y_{ijk} \leq c_k \qquad & k = 1, \ldots, n' \end{aligned}$$

3.5 Bornes inférieures

Dans cette section, nous allons proposer 5 bornes inférieures pour le problème SWA sur deux machines uniformes décrites comme suit :

1. Une borne inférieure naturelle notée LB_1 serait la valeur de la date de fin de traitement de l'ordonnancement obtenu en ignorant le conflit entre les tâches, le problème devient un simple problème d'ordonnancements sur deux machines uniformes qui peut être résolu

en temps polynomial par l'algorithme décrit plus bas. Etant donné, C_1 et C_2 les dates de fin de traitement respectives sur la première et deuxième machine. L'algorithme de calcul de la borne LB_1 est défini comme suit :

Algorithme 3: Calcul de la borne inférieure LB_1

entrée : Ensemble de tâches J ;
sortie : Borne inférieure LB_1 ;

```
1  début
2      pour chaque j ∈ J Faire
3          si C2 + α ≤ C1 + 1 alors
4              Ordonnancer j sur la deuxième machine à la date cj = C2 + α ;
5              C2 = cj ;
6          sinon
7              Ordonnancer j sur la première machine à la date cj = C1 + 1 ;
8              C1 = cj ;
9          fin si
10     fait
11     LB1 = max{C1; C2} ;
12 fin.
```

2. Une deuxième borne inférieure notée LB_2 peut être obtenue simplement en calculant LB_1 sur chaque partie connexe du graphe de concordance. Cette borne permet d'améliorer la borne inférieure spécialement pour les graphes de concordance avec peu de concordance entre les tâches.

3. Lorsque α prend des valeurs significativement petites chaque tâche ordonnancée sur la première machine va être ordonnancée simultanément à un nombre important de tâches. Une tâche avec autant de tâches qui lui sont concordantes n'existe pas tout le temps, particulièrement, en présence d'un graphe de concordance comportant un taux important de tâches conflictuelles. Le calcul de la borne LB_3 est basé sur le calcul de la borne LB_2. Les tâches sont ordonnancées en ignorant le conflit entre les tâches, cependant, on s'assure que chaque tâche ordonnancée sur la première machine ait un degré supérieur ou égale à $\lfloor \frac{1}{\alpha} \rfloor$.

Algorithme 4: Calcul de la borne inférieure LB_3

entrée : Ensemble de tâches J ;
sortie : Une borne inférieure LB_1 ;

1 **début**
2 $S = J$;
3 Ordonner les tâches de S par ordre décroissant de leurs degrés ;
4 **pour chaque** $j \in J$
5 **si** $C_2 + \alpha \leq C_1 + 1$ **alors**
6 Ordonnancer j sur la deuxième machine à la date $c_j = C_2 + \alpha$;
7 $C_2 = c_j$;
8 **sinon**
9 **si** $\exists k \in S, d(k) \geq \lfloor \frac{1}{\alpha} \rfloor$ **alors**
10 $S = S - k$;
11 Ordonnancer j sur la première machine à la date $c_j = C_1 + 1$;
12 $C_1 = c_j$;
13 **sinon**
14 Ordonnancer les tâches restantes sur la deuxième machine ;
15 Aller à la ligne 19 de l'algorithme ;
16 **fin si**
17 **fin si**
18 **fait**
19 $LB_1 = \max\{C_1; C_2\}$;
20 **fin.**

4. Une autre borne inférieure peut être obtenue en calculant le temps de traitement d'un ensemble de tâches indépendantes ordonnancées sur la deuxième machine . En effet, si on considère des tâches indépendantes entre elles (en conflit entre elles), celles-ci ne pourront jamais être ordonnancées simultanément. Cependant, dans un ordonnancement optimal nous ignorons sur quelle machine ces tâches seraient ordonnancées, nous allons donc supposer que ces tâches sont ordonnancées sur la deuxième machine ce qui nous permet d'obtenir une borne inférieure par la formule $\alpha|I|$, où I est un ensemble de tâches indépendantes entre elles. Pour le calcul de cette borne, les tâches d'un stable maximum procureraient la meilleure borne possible, mais étant donné que le problème est NP-difficile, nous allons utiliser une heuristique pour le calcul approché du stable maximum. Nous utilisons donc l'algorithme suivant pour le calcul du stable maximal :

Algorithme 5: Calcul d'un stable maximal

entrées : Ensemble de tâches J, Graphe de concordance $G = (J, E)$;
sortie : Stable I ;
début
  $I = \emptyset$;
  tant que $J \neq \emptyset$ **faire**
    Sélectionner une tâche $j \in J$;
    Ajouter j à l'ensemble I ;
    Retirer de l'ensemble J la tâche j et ses voisins $N(j)$;
  fait
fin.

En considérant les tâches par ordre croissant et décroissant de leurs degrés lors du calcul du stable par l'algorithme ci-dessus, nous obtenons les bornes inférieures respectives $LB_4 = \alpha|I_{inc}|$ et $LB_5 = \alpha|I_{dec}|$, où I_{inc} et I_{dec} représentent respectivement les stables obtenus par l'algorithme décrit plus haut lorsque les tâches sont considérées par ordre croissant et décroissant.

3.6 Métaheuristique : Simulated Annealing

La métaheuristique Simulated Annealing (recuit simulé) appliqué aux problèmes d'optimisation a émergé des travaux de S. Krikpatrick et al. [20] et V. Cerny [8]. Dans ces premiers travaux, la métaheuristique a été appliqué pour un problème de partion de graphes et dans l'intégration à très grande échelle (une technologie de circuits intégrés). L'algorithme Simulated Annealing est basé sur le principe de la mécanique statistique pour laquelle le processus du recuit nécessite de chauffer à une température très élevée puis refroidir lentement une substance afin d'obtenir une structure cristalline forte.

3.6.1 Représentation d'une solution

Un ordonnancement réalisable est représenté par une liste ordonnée de tâches. Résoudre le problème original revient à trouver une permutation des tâches qui fournit un ordonnancement optimal. Obtenir un ordonnancement réalisable à partir d'une liste de tâches peut se faire selon les deux stratégies suivantes :

– *Stratégie S_1 :* La stratégie S_1 consiste à ordonnancer la première tâche non ordonnancée j de la liste à la dernière position de la première ou de la deuxième machine de sorte à

minimiser c_j. La stratégie S_1 est décrite par l'algorithme 6.

– *Stratégie S_2 :* La stratégie est similaire à la stratégie S_1. Elle consiste à ordonnancer la première tâche non ordonnancée j de la liste à la dernière position de la première ou de la deuxième machine de sorte à minimiser t_j (au lieu de c_j pour la stratégie S_1), avec pratiquement le même algorithme (remplacer la condition de la ligne 4 par $C_2 \leq C_1$).

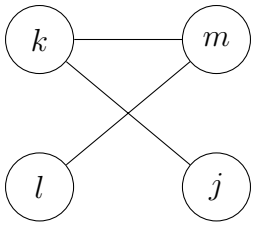


FIGURE 3.4: Graphe de concordance G.

Exemple.

Soit un ensemble de tâches $J = \{j, k, l, m\}$, $\alpha = 0.4$, une liste $L = (j, k, l, m)$ et un graphe de concordance $G = (J, E)$ décrit par la Figure 3.4. Les ordonnancements résultants des stratégies S_1 et S_2 sont décrits comme suit et illustrés par la Figure 3.5.

L'ordonnancement des tâches de la liste L selon la stratégie S_1 se fait de la manière suivante :

1. itération 1 : tâche à ordonnancer : j
   (a) $C_1 = 0$ et $C_2 = 0$
   (b) Nous avons $C_2 + 0,4 \leq C_1 + 1$ $(0,4 \leq 1)$:
      – tâche j peut s'ordonnancer sur la machine 2 à la date $t_j = C_2$:
        – Ordonnancer la tâche j sur la machine 2 à la date $t_j = C_2 = 0$
        – $C_2 = 0,4$
2. itération 2 : tâche à ordonnancer : k
   (a) $C_1 = 0$ et $C_2 = 0,4$
   (b) Nous avons $C_2 + 0,4 \leq C_1 + 1$ $(0,8 \leq 1)$:
      – tâche k peut s'ordonnancer sur la machine 2 à la date $t_k = C_2$:
        – Ordonnancer la tâche k sur la machine 2 à la date $t_k = C_2 = 0,4$
        – $C_2 = 0,8$
3. itération 3 : tâche à ordonnancer : l

Algorithme 6: Evaluation d'une liste de priorité avec la stratégie S_1

H

entrée : Une liste de priorité L ;

sortie : Un ordonnancement réalisable des tâches de L ;

début
  j = première tâche de la liste L ;
  répéter
    si $C_2 + \alpha \leq C_1 + 1$ **alors**
      si j peut s'ordonnancer sur la machine 2 à la date $t_j = C_2$ **alors**
        ordonnancer j sur la machine 2 à la date $t_j = C_2$;
        $C_2 = C_2 + \alpha$;
      sinon
        $C_2 = C_1$;
        ordonnancer j sur la machine 2 à la date $t_j = C_2$;
        $C_2 = C_2 + \alpha$;
      fin si
    sinon
      si j peut s'ordonnancer sur la machine 1 à la date $t_j = C_1$ **alors**
        ordonnancer j sur la machine 1 à la date $t_j = C_1$;
        $C_1 = C_1 + 1$;
      sinon
        $C_1 = C_2$;
        ordonnancer j sur la machine 2 avec $c_j = C_2 + \alpha$;
        $C_2 = C_2 + \alpha$;
      fin si
    fin si
    j = la prochaine tâche de L ;
  jusqu'à toutes les tâches de L sont ordonnancées ;
fin.

(a) $C_1 = 0$ et $C_2 = 0,8$

(b) Nous avons $C_1 + 1 < C_2 + 0,4$ $(1 < 1,2)$:

- tâche l ne peut pas s'ordonnancer sur la machine 1 à la date $t_l = C_1$ puisque la tâche l devrait s'ordonnancer simultanément à des tâches conflictuelles (tâches j et k) :
  - $C_1 = C_2 = 0,8$
  - Ordonnancer la tâche k sur la machine 2 à la date $t_l = C_2 = 0,8$
  - $C_2 = 1,2$

4. itération 4 : tâche à ordonnancer : m

(a) $C_1 = 0,8$ et $C_2 = 1,2$

(b) Nous avons $C_2 + 0,4 \leq C_1 + 1$ $(1,6 \leq 1,8)$:

- tâche m peut s'ordonnancer sur la machine 2 à la date $t_m = C_2$:
  - Ordonnancer la tâche m sur la machine 2 à la date $t_m = C_2 = 1,2$
  - $C_2 = 1,6$

L'ordonnancement des tâches de la liste L se fait comme suit suivant la stratégie S_2 :

1. itération 1 : tâche à ordonnancer : j

(a) $C_1 = 0$ et $C_2 = 0$

(b) Nous avons $C_2 \leq C_1$ $(0 \leq 0)$:

- tâche j peut s'ordonnancer sur la machine 2 à la date $t_j = C_2$:
  - Ordonnancer la tâche j sur la machine 2 à la date $t_j = C_2 = 0$
  - $C_2 = 0,4$

2. itération 2 : tâche à ordonnancer : k

(a) $C_1 = 0$ et $C_2 = 0,4$

(b) Nous avons $C_1 < C_2$ $(0 < 0,4)$:

- tâche k peut s'ordonnancer sur la machine 1 à la date $t_k = C_2$ (puisque j et k sont des tâches concordantes) :
  - Ordonnancer la tâche k sur la machine 1 à la date $t_k = C_1 = 0$
  - $C_1 = 1$

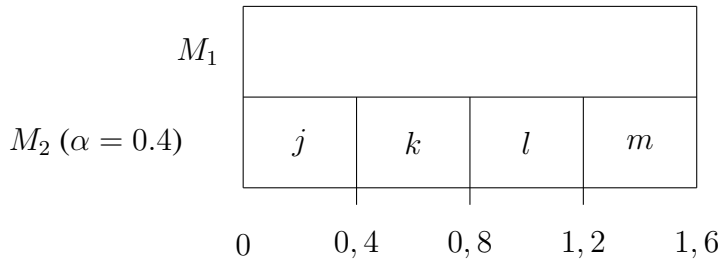
3. itération 3 : tâche à ordonnancer : l

(a) $C_1 = 1$ et $C_2 = 0,4$

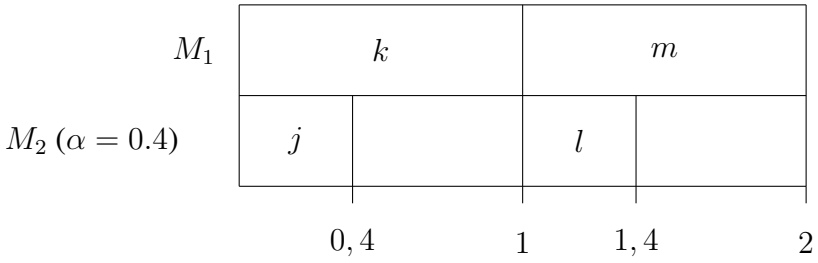
(b) Nous avons $C_2 \leq C_1$ $(0,4 \leq 1)$:

– tâche l ne peut pas s'ordonnancer sur la machine 2 à la date $t_l = C_2$ puisque la tâche l et la tâche k ne sont pas concordantes :
  – $C_2 = C_1 = 1$
  – Ordonnancer la tâche k sur la machine 2 à la date $t_l = C_2 = 1$
  – $C_2 = 1,4$

4. itération 4 : tâche à ordonnancer : m
   (a) $C_1 = 1$ et $C_2 = 1,4$
   (b) Nous avons $C_1 < C_2$ $(1 < 1,4)$:
   – tâche m peut s'ordonnancer sur la machine 1 à la date $t_m = C_1$ puisque les tâche l et m sont concordantes :
     – Ordonnancer la tâche m sur la machine 1 à la date $t_m = C_1 = 1$
     – $C_1 = 2$



(a) Ordonnancement des tâches selon S_1.



(b) Ordonnancement des tâches selon S_2.

FIGURE 3.5: Ordonnancement des tâches selon les stratégies S_1 et S_2

3.6.2 Solution initiale

La solution initiale est l'ordonnancement obtenu à partir d'une liste L_0 générée aléatoirement.

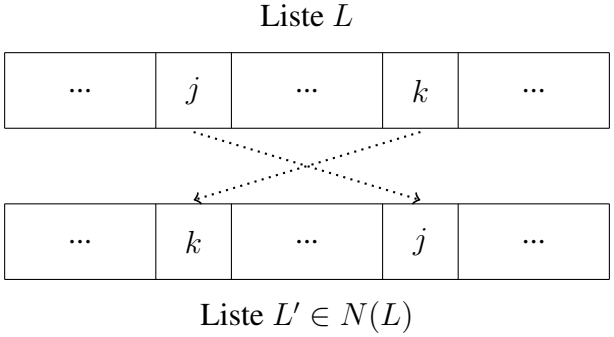


FIGURE 3.6: Illustration d'un liste voisine L' à partir d'une liste L.

3.6.3 Voisinage

Etant donné une liste L de tâches, le voisinage de L est défini par l'ensemble des listes qui sont obtenues en inversant la position de deux tâches de L (voir Figure 3.6).

3.6.4 Probabilité d'acceptance

La probabilité d'accepter une solution voisine qui n'améliore pas la valeur de la fonction objectif est :

$$e^{\frac{-\Delta E}{T}}$$

- ΔE est la différence entre la valeur de la fonction objectif de la solution courante et la solution voisine
- T est la température actuelle.

3.6.5 Schéma de refroidissement

A partir d'une température initiale T_{max}, la température décroit avec la formule suivante :

$$T = \beta T$$

β est une constante tel que $0 < \beta < 1$.

3.6.6 Critère d'arrêt

L'algorithme s'arrête si l'une des deux propositions suivantes se vérifie :

- La température terminale T_{min} est atteinte.

– La meilleure solution rencontrée atteint la borne inférieure.

3.6.7 Algorithme Simulated Annealing

Algorithme 7: Algorithme Recuit Simulé

entrée : Liste initiale L_0 ;
sortie : Liste qui fournit le meilleur ordonnancement rencontré ;
début
  $L = L_0$; // Générer une liste aléatoire
  $T = T_{max}$; // Initialiser la température
  répéter
    répéter// A une température fixe
      Générer aléatoirement une liste voisine L' ;
      $\Delta E = C_{max}(L') - C_{max}(L)$;
      si $\Delta E \leq 0$ **alors**
        $L = L'$; // Accepter L'
      sinon
        Accepter L' avec une probabilité $e^{\frac{-\Delta E}{T}}$;
      fin si
    jusqu'à un certain nombre d'itérations Nbr_{iter}
    $T = \beta T$; // Mise à jour de la température
  jusqu'à Critère d'arrêt
fin.

– T_{max} et T_{min} sont respectivement les températures initiales et terminales.
– β est un nombre réel tel que $0 < \beta < 1$.

3.7 Algorithme Branch and Bound

Dans cette section, nous proposons une méthode Branch and Bound [22] pour résoudre le problème étudié. Nous discutrons en premier lieu de la stratégie à utiliser pour l'évaluation d'une liste de priorité. Nous définirons ensuite, les règles de séparation ainsi que la stratégie d'exploration pour le parcours de l'arbre de recherche. Enfin, nous présenterons une borne inférieure sur chaque nœud de l'arbre de recherche.

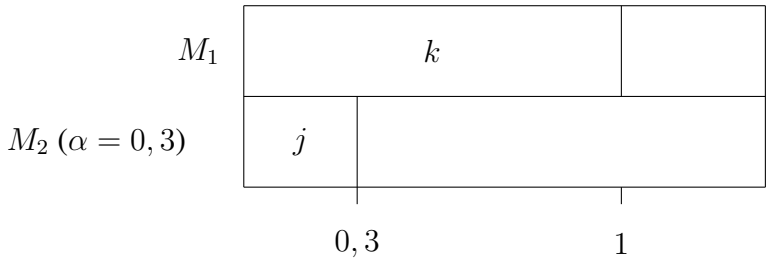


FIGURE 3.7: Ordonnancement des tâches j et k selon S_2.

3.7.1 Représentation d'une solution

Un ordonnancement réalisable sera représenté par une liste de tâches comme décrit dans la section 3.6.1. Cependant, la méthode branch & bound est une méthode de résolution exacte basée sur le balayage de "toutes" les solutions réalisables. Il faudrait donc que chaque solution réalisable puisse être représentée par une permutation des tâches (une liste).

A partir de chaque liste de tâches un ordonnancement réalisable est obtenu en utilisant la stratégie S_1 décrite dans la section 3.6.1, pour rappel S_1 consiste à ordonnancer la prochaine tâche j de la liste dans la dernière position de la première ou de la deuxième machine de sorte à minimiser c_j.

La stratégie S_2 décrite dans la section $3.6.1$ ne peut être utilisée ici pour construire un ordonnancement à partir d'une liste. En effet, si on considère A l'ensemble des ordonnancements construits par S_2 à partir de toutes les permutations d'une liste de tâches, certains ordonnancements du problème n'appartiendront pas à A. Pour rappel S_2 consiste à ordonnancer la prochaine tâche j de la liste dans la dernière position de la première ou de la deuxième machine de sorte à minimiser t_j.

Exemple. On considère deux tâches concordantes j et k qui doivent être ordonnancées sur deux machines uniformes avec $\alpha = 0,3$. L'ordonnancement obtenu en utilisant la stratégie S_2 procure l'ordonnancement décrit par la Figure 3.7 (Sans perte de généralité la tâche j est ordonnancée sur la deuxième machine et la tâche k sur la première). L'ordonnancement optimal consiste à ordonnancer successivement les deux tâches sur la deuxième machine, or, avec la stratégie S_2 il est impossible d'obtenir un tel ordonnancement. Ce qui est problématique puisque la méthode branch and bound doit trouver une solution optimale, alors que celle-ci ne pourra éventuellement pas pouvoir être trouvée en utilisant la stratégie S_2 pour évaluer une liste.

3.7.2 Calcul d'une borne supérieure initiale

Calculer une borne supérieure initiale permet d'avoir une solution initiale à laquelle l'évaluation de chaque nœud de l'arbre de recherche va être comparée. Plus la borne inférieure est serrée, plus tôt la stérilisation des nœuds se fera.

Dans notre cas, nous allons considérer deux listes de tâches L_{inc} et L_{dec}. Les tâches dans L_{inc} sont ordonnées par ordre *croissant* de leurs degrés dans le graphe de concordance, et les tâches dans L_{dec} sont ordonnées par ordre *décroissant* de leurs degrés. Les deux listes seront évaluées en utilisant les stratégies S_1 et S_2. Nous retiendrons la solution avec une date de fin de traitement minimale.

3.7.3 Règle de séparation

Pour chaque tâche il existe n positions possibles (pour assigner une tâche dans la liste). Un nœud au niveau k de l'arbre de recherche représente une liste partielle dans laquelle les k premières tâches sont fixées dans la liste. La séparation de n'importe quel nœud au niveau k consiste à déterminer la $k^{eme} + 1$ tâche qui doit être ajoutée à la liste dans la position $k + 1$. Ceci implique que séparer un nœud parent au niveau k de l'arbre de recherche va générer $n - k$ enfants.

3.7.4 Stratégie pour sélectionner le prochain nœud

Nous utiliserons une stratégie de recherche en profondeur. La recherche commence de la racine et consiste à explorer chaque branche de l'arbre de recherche en profondeur jusqu'aux feuilles avant de revenir au premier nœud qui contient encore des enfants non visités. Plus précisément lorsqu'on visite un nœud au niveau k la sélection d'un nœud du niveau $k + 1$ ou la sélection de la tâche à ajouter à la liste partielle se fait selon l'ordre décroissant du degré dans le graphe de concordance.

3.7.5 Borne inférieure sur une liste partielle

Une borne inférieure sur une liste partielle d'un nœud au niveau i est calculée par l'algorithme suivant :

Algorithme 8: Borne inférieure à un nœud donné

entrées : Une liste partielle L d'un nœud S au niveau i de l'arbre de recherche ;

sortie : Une borne inférieure $LB(S)$ pour le nœud S ;

début

  Ordonnancer les i premières tâches fixées de L selon la stratégie S_1 ;

  si aucune tâche restante ne peut s'ordonnancer à la position P [†] **alors**

    Combler l'écart entre C_1 et C_2 avec un temps mort ;

  fin si

  Ordonnancer les tâches restantes selon l'algorithme de calcul de LB_3 [‡] ;

  $LB = C_{max}$;

fin.

[†]P est la position qui minimise la date de fin de traitement de la tâche ordonnancée (entre la dernière position sur la première ou sur la deuxième machine)

[‡]la borne LB_3 est la borne avec les meilleures performances (voir Section 3.8.1)

3.8 Expérimentations

Dans ce chapitre, nous allons procéder à l'expérimentation numérique des méthodes décrites plus haut : heuristiques pour le calcul des bornes inférieures, la méthode métaheuristique Simulated Annealing pour la résolution approchée du problème sur machines uniformes, la méthode branch and bound pour la résolution exacte du problème toujours sur machines uniformes, ainsi que la résolution du modèle mathématique en utilisant le solveur CPLEX.

Toutes les méthodes ont été codées en C++, en utilisant Visual Studio 2012, sur un ordinateur ayant un processeur i3 avec 8GB de RAM. Pour chaque méthode plus de 3600 instances sont testées. Les instances sont construites de la manière suivante : Les graphes de concordance sont générés en utilisant la méthode $G(n, p)$ de Erdos Rényi, ce qui est la méthode la plus intuitive et la plus utilisée pour générer un graphe de façon aléatoire. Etant donné un ensemble de n tâches, la méthode consiste à générer un graphe de concordance où chaque arête est présente dans le graphe de concordance avec une probabilité p. La densité d'un graphe représente le pourcentage d'arêtes présentes dans un graphe de concordance $G = (J, E)$ par rapport au nombre maximal possible d'arêtes. Donc, plus la probabilité p est élevée plus la densité du graphe de concordance augmente et inversement.

Algorithme 9: Génération aléatoire d'un graphe de concordance

entrées : Un nombre de tâches n, une probabilité $p \in [0, 1]$;
sortie : Une matrice d'adjacence A ;
début
  Soit A une matrice d'adjacence de taille n ;
  pour $i = 1$ **à** n **faire**
    pour $j = 1$ **à** i **faire**
      si Rand() [†]<p **alors**
        $A[i][j] = 1$;
      sinon
        $A[i][j] = 0$;
      fin si
    fait
  fait
fin.

[†]Rand est une fonction qui génère un réel de façon aléatoire dans l'intervalle $[0, 1]$

Afin de générer une instance du problème SWA sur deux machines uniformes nous avons besoin de trois paramètres : le nombre de tâches n, la densité du graphe de concordance (directement liée à la probabilité p) et le temps de traitement α des tâches sur la deuxième machine. Pour chaque $n \in \{20; 35; 50; 75; 100; 150; 200; 350; 500\}$ et $\alpha, p \in \{0,1; 0,2; 0,3; 0,4; 0,5; 0,6; 0,7; 0,8; 0,9\}$ nous allons construire 5 graphes de concordances. On considère que les graphes générés avec une probabilité p sont : des graphes de faible densité lorsque $p \in \{0,1; 0,2; 0,3\}$, des graphes de moyenne densité lorsque $p \in \{0,4; 0,5; 0,6\}$ et des graphes de haute densité lorsque $p \in \{0,7; 0,8; 0,9\}$. On considère également que les valeurs de α sont faibles, moyennes et hautes si les valeurs de α sont respectivement $\alpha \in \{0,1; 0,2; 0,3\}$, $\alpha \in \{0,4; 0,5; 0,6\}$ et $\alpha \in \{0,7; 0,8; 0,9\}$. On dira aussi que les valeurs de n sont faibles, moyennes et hautes si les valeurs de n sont respectivement $n \in \{20; 35; 50\}$, $n \in \{75; 100; 150\}$ et $n \in \{200; 350; 500\}$ Chacune des méthodes est évaluée en utilisant les critères suivants :

– GAP(%) : Représente le pourcentage de déviation moyenne de la date de fin de traitement de l'ordonnancement obtenue par la méthode évaluée par rapport à la meilleure borne inférieure. Le calcul du pourcentage de la déviation moyenne se fait par la formule suivante $\frac{C_{max}-LB}{LB} \times 100$ où C_{max} est la valeur de la date de fin de traitement de l'ordonnancement obtenue par la méthode évaluée et LB la meilleure borne inférieure.

– NbOpt : représente le nombre de fois qu'une méthode donne une solution optimale. Une solution optimale peut être identifiée soit par la meilleure solution obtenue par une méthode de résolution exacte, soit lorsque la valeur de la date de fin de traitement donnée par la méthode évaluée atteint la borne inférieure dans le cas des méthodes approchées.

– CPU : temps d'exécution moyen en secondes.

L'expérimentation des méthodes de résolution exactes : méthode branch and bound ainsi que la résolution du modèle mathématique va se faire comme suit : les méthodes de résolution exactes vont s'exécuter pendant un maximum de 3600 secondes. Si aucune solution optimale n'est trouvée durant ce temps imparti, l'exécution de la méthode est interrompue et nous dirons qu'aucune solution optimale n'a pu être trouvée.

3.8.1 Bornes inférieures

Avant de pouvoir expérimenter les différentes méthodes de résolution, nous allons d'abord comparer les bornes inférieures définies dans la Section 3.5. La borne LB_3 est la borne la plus efficace entre LB_1 et LB_2, puisque LB_3 est une amélioration de la borne LB_2 qui a son tour est une version améliorée de LB_1. Nous allons malgré tout quantifier cette amélioration en utilisant une version modifiée du critère GAP(%) précédemment défini. Nous comparerons également les bornes LB_4 et LB_5 à la borne LB_3. L'amélioration moyenne (en pourcentage) de la borne LB_a par rapport à la borne LB_b (LB_a vs LB_b) est donnée par la formule $\frac{LB_a - LB_b}{LB_b} \times 100$.

Pour des graphes de concordance à faible densité, comme le montre le Tableau 3.1, la borne LB_2 améliore en moyenne de 6.1% la borne LB_1 lorsque $n = 20$ et $p = 0,1$. Lorsque le nombre de tâches commence à augmenter ou que la densité du graphe augmente l'amélioration devient moins notable. En effet, lorsque $p = 0,1$ l'amélioration est de 1,9% en moyenne pour $n = 50$ et 0,3% pour $n = 75$. Lorsque $n > 75$ il n'existe pas de différence entre LB_1 et LB_2. De même lorsque $p = 0,2$, l'amélioration est de $0,6\%$ en moyenne pour $n = 20$, mais pour des valeurs plus élevées de n aucune amélioration n'est notable. Pour $p \geq 0,3$ il n'y a également plus d'amélioration.

La borne LB_3 améliore la borne LB_2 pour des graphes de concordance à faible densité seulement lorsque la vitesse entre les deux machines est très importante ($\alpha = 0,1$), comme montré par le Tableau 3.1. Lorsque $p = 0,1$, l'amélioration est présente pour $n \leq 50$ avec une amélioration moyenne de 6,3%. Lorsque $p = 0,2$ l'amélioration se fait pour $n \leq 35$ avec une amélioration moyenne de 1,9%. Enfin, lorsque $p = 0,3$, l'amélioration est de 3,2% en moyenne

pour $n = 20$.

La borne LB_3 surperforme les bornes LB_4 et LB_5. LB_3 est meilleure que LB_4 et LB_5 pour toutes les valeurs de p, α et n, avec une amélioration moyenne de plus de 1400% par rapport à LB_4 et 1800% par rapport à LB_5.

La borne LB_3 est la meilleure borne. En ignorant le conflit entre les tâches, la borne LB_3 donne de bonnes performances lorsque le conflit est moins présent entre les tâches (pour les graphes de concordance à moyenne et forte densité). Elle permet également, d'apporter quelques améliorations (par rapport à LB_1 et LB_2) lorsqu'il existe un fort pourcentage de tâches conflictuelles (pour des graphes de concordance à faible densité), ce qui permet à LB_3 d'avoir de solides performances.

TABLE 3.1: GAP (%) : LB_2 vs LB_1 et LB_3 vs LB_2 pour des graphes de concordance à faible densité

		LB_2 vs LB_1										LB_3 vs LB_2									
	α	n	20	35	50	75	100	150	200	350	500	n	20	35	50	75	100	150	200	350	500
$p = 0,1$	0,1		**1,1**	**1,3**	0,0	0,0	0,0	0,0	0,0	0,0	0,0		**4,2**	**7,4**	**7,4**	0,0	0,0	0,0	0,0	0,0	0,0
	0,2		**7,1**	**0,7**	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,3		**5,0**	**2,0**	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,4		**4,0**	**1,6**	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,5		**7,1**	**1,7**	**0,6**	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,6		**7,2**	**3,0**	**0,2**	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,7		**6,4**	**1,9**	**0,4**	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,8		**9,8**	**3,0**	**0,5**	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,9		**7,5**	**2,0**	**0,7**	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
$p = 0,2$	0,1		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		**3,2**	**0,6**	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,3		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,4		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,5		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,6		**2,1**	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,7		**0,7**	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,8		**2,7**	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,9		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
$p = 0,3$	0,1		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		**3,2**	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,3		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,4		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,5		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,6		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,7		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,8		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	0,9		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0

3.8.2 Métaheuristiques

Dans cette section, nous allons évaluer l'algorithme Simulated Annealing présenté dans la Section 3.6. Nous allons comparer les deux variantes de l'algorithme notées SA_1 et SA_2 dans lesquelles l'évaluation d'une liste de priorité se fait en utilisant respectivement les stratégies S_1 et S_2. Une troisième variante de l'algorithme notée SA_{mix} est obtenue en mixant les deux stratégies S_1 et S_2. En d'autres termes, une liste de priorité est évaluée en utilisant les deux stratégies S_1 et S_2 et la solution retenue est celle ayant une date de fin de traitement minimale.

Afin de pouvoir évaluer SA_1, SA_2 et SA_{mix}, nous devons définir les paramètres suivants de l'algorithme obtenus par des expérimentations préliminaires :

- Température initiale T_{max} : $T_{max} = 0,85 \times LB^{0,64}$
- Température terminale T_{min} : $T_{min} = 5,26 \times 10^{-3} \times LB^{0,58}$
- Nombre d'itérations NB_{iter} à une température fixe : $NB_{iter} = \frac{79,2 \times 10^3}{n^{1,33}}$, étant donné que dans l'algorithme SA_{mix} une liste de priorité est évaluée en utilisant S_1 et S_2 pour de déterminer le minimum, nous allons donc diviser par 2 le nombre d'itérations dans SA_{mix} afin d'avoir une comparaison équitable avec SA_1 et SA_2.
- Coefficient de refroidissement β : nous allons opter pour un refroidissement très long avec $\beta = 0,99$

Limites de la stratégie S_2

Avant d'entamer l'expérimentation des méthodes, nous voulons mettre en avant deux limitaions constatés lors de l'utilisation de la stratégie S_2 pour l'évaluation d'une liste de priorité.

1. La première limitation apparait lorsque les valeurs de p, α et de n sont les plus basses. En effet, lors de l'ordonnancement d'une tâche j sur la deuxième machine, si la tâche suivante dans la liste, qu'on appelera tâhe k, est concordante à la tâche j, celle-ci sera ordonnancée sur la première machine. Etant donné que le taux de conflit entre les tâches est très important (faible valeurs de p), le nombre de tâches qui sont concordantes à la tâche k sont peu nombreuses plus particulièrement lorsque le nombre de tâches est réduit. La possibilité d'avoir des tâches concordantes à k qui viennent juste après k dans la liste est très faible. La tâche k sera donc ordonnancée simultanément à très peu de tâches (voir

qu'à la tâche j) laissant ainsi un temps mort (à l'opposé de la tâche k) qui ne pourra être comblé. Et plus les valeurs de α sont réduites plus grand sera le temps mort. Or, si ces tâches auraient été ordonnancées sur la deuxième machine le temps de traitement global serait moins important.

2. Afin de constater la deuxième limitation rencontré lors de l'utilisation de la stratégie S_2 pour l'évaluation d'une liste, nous allons nous focaliser sur le cas le plus probant, c'est à dire lorsque $\alpha = 0,1$. Comme mentionné dans la section 3.4, un ordonnancement peut être splitté en compartiments. Lorsque la concordance entre les tâches est importante, les compartiments contiennent peu (ou pas) de temps mort. Dans notre cas, chaque compartiment contient 11 tâches (1 sur la première et 10 sur la deuxième machine). Soit r le nombre de tâches ordonnancées dans le dernier compartiment. Si $r < 11$, la solution optimale serait obtenue en ordonnançant les r tâches sur la deuxième machine. Afin d'ordonnancer ces tâches sur la deuxième machine en utilisant la stratégie S_2, il est nécessaire que chaque paire de tâches successives soient en conflit, car, si deux tâches successives sont concordantes la stratégie S_2 ordonnancera la deuxième tâche sur la première machine obtenant ainsi une solution avec une date de fin de traitement plus importante. Les valeurs de p et de r jouent un rôle important, car plus ces deux valeurs augmentent, plus il est difficile de trouver une liste avec r tâches qui se suivent où chaque paire de tâches successives sont en conflit. Le nombre de tâches des instances testées sont 20, 35, 50, 75, 100, 150, 200, 350 et 500 ce qui génère des valeurs de r respectives de 9, 2, 6, 9, 1, 7, 2, 9 et 5 (en considérant que chaque autre compartiment contient 11 tâches). On pourrait s'attendre à une difficulté à déterminer des solutions optimales pour les instances avec les nombres de tâches qui génèrent des valeurs de r les plus élevées, et la difficulté augmente lorsque les valeurs de p augmentent également.

Nous allons à présent passer à l'expérimentation des trois méthodes. Il est clair que nombre de tâches n, les valeurs de α et les valeurs de p influent directement sur le temps de traitement, le GAP et le nombre de solutions optimales.

Pour des valeurs fixes de p et de α, le temps de traitement des trois méthodes augmente lorsque le nombre de tâches augmente. Cependant, le temps de traitement diminue lorsque les valeurs de p et de α augmentent (voir Figure 3.8). Pour ce qui est des solutions optimales, leur pourcentage augmente lorsque les valeurs de p est de α augmentent pour les trois méthodes. Lorsque n augmente le pourcentage de solutions optimales augmente pour SA_1 mais semble avoir un comportement différent avec SA_2 et SA_{mix} que nous allons expliquer plus bas. Quant

au GAP moyen (voir Figure 3.9), les méthodes ont des comportements différents lorsque n, p et α varient. Pour des valeurs fixes de p et de α, le GAP pour la méthode SA_1 augmente lorsque le nombre de tâches augmente. Le GAP pour les méthodes SA_2 et SA_{mix} quant à lui diminue lorsque le nombre de tâches augmente. Lorsque les valeurs de p et de α augmentent le GAP diminue pour les trois méthodes jusqu'à même disparaitre.

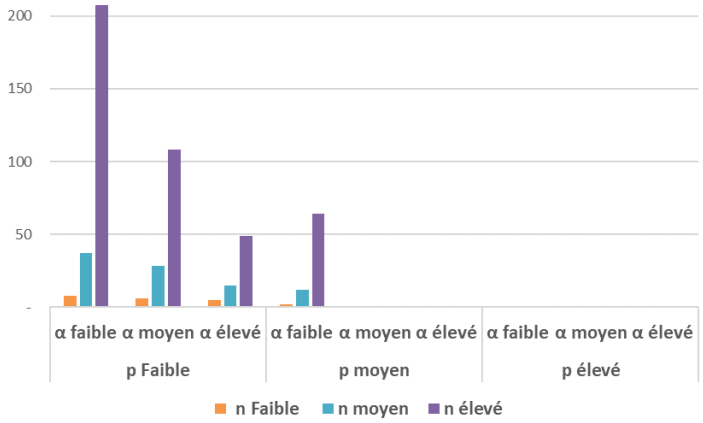
Nous allons maintenant évaluer les performances des trois méthodes et les comparer entre elles :

- Graphes à faible densité : La méthode SA_1 a globalement les performances les moins bonnes que ce soit pour le GAP moyen avec 2,8% ou le pourcentage de solutions optimales avec 52%. La méthode SA_{mix} est la plus performante avec un GAP de 0,7% et un pourcentage de solutions optimales de 80,2%. La méthode SA_2 quant à elle vient en deuxième place mais avec des résultats contrastés. SA_2 a un GAP moyen de 2.4% et un pourcentage de solutions optimales de 79,8%. Cependant, pour des valeurs faibles de α et de n, SA_2 a les moins bonnes performances des trois méthodes avec un GAP de 17,6% et un pourcentage de solutions optimales de 45,9%. Alors que la méthode SA_2 à les meilleures performances pour des valeurs faibles de α et des valeurs moyennes et élevées de n avec un GAP de 2,7% (16,6% pour SA_1 et 2,8% pour SA_{mix}) et un pourcentage de solutions optimales de 62,2% (5,9% pour SA_1 et 56,7% pour SA_{mix}). Les résultats de SA_2 sont encore moins bons pour de faibles valeurs de n et $\alpha = 0,1$ avec un GAP de 44,8% et 22,8% de solutions optimales seulement. Ces mauvais résultats sont dus à la deuxième limitation de l'utilisation de la stratégie S_2 lors de l'évaluation d'une liste.

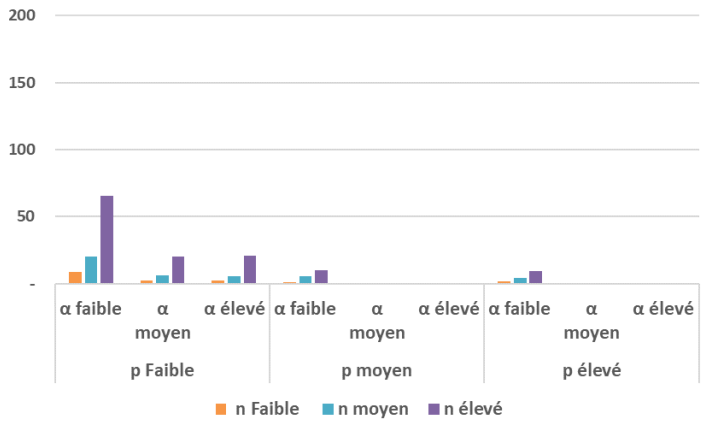
- Graphes à moyenne densité : Pour des valeurs de α moyennes et élevés les trois méthodes ont 100% de solutions optimales. Pour les valeurs basses de α, SA_{mix} reste la plus performante avec un GAP de $0,16\%$ et $87,4\%$ de solutions optimales. SA_2 arrive juste derrière SA_{mix} avec un GAP de $0,56\%$ et $85,2\%$ de solutions optimales. En dernier, SA_1 a les moins bonnes performances avec un GAP moyen de $0,9\%$ et $74,1\%$ de solutions optimales. En retirant le cas où $\alpha = 0,1$, SA_2 et SA_{mix} ont 100% de solutions optimales.

- Graphes à haute densité : La méthode SA_1 a les meilleures performances avec 100% de solutions optimales. SA_2 et SA_{mix} ont également 100% de solutions optimales à l'exception de $\alpha = 0,1$ où le GAP est de 2,1% pour SA_2 et de 0,5% pour SA_{mix}. Le pourcentage de solutions optimales est de 33,3% pour SA_2 et 62,2% pour SA_{mix}.

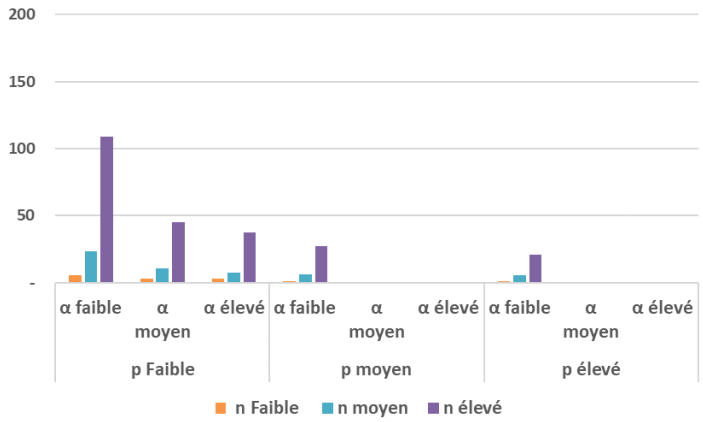
FIGURE 3.8: CPU (S) : Comparatif entre les méthodes SA_1, SA_2 et SA_{mix}



(a) Méthode SA_1

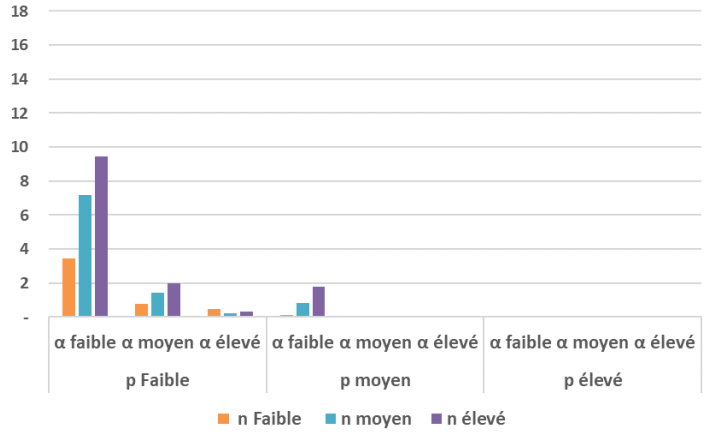


(b) Méthode SA_2

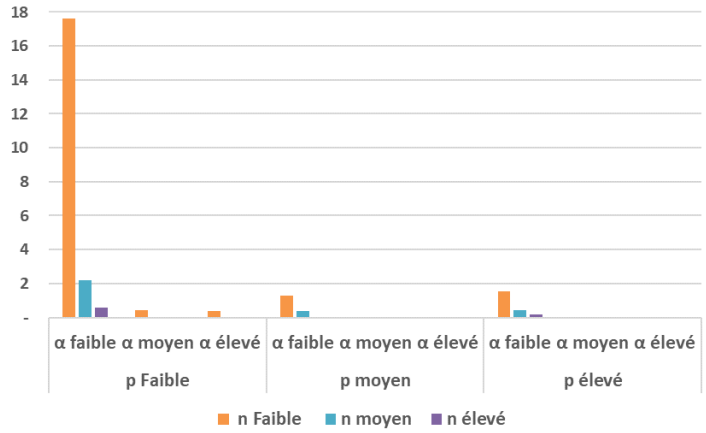


(c) Méthode SA_{mix}

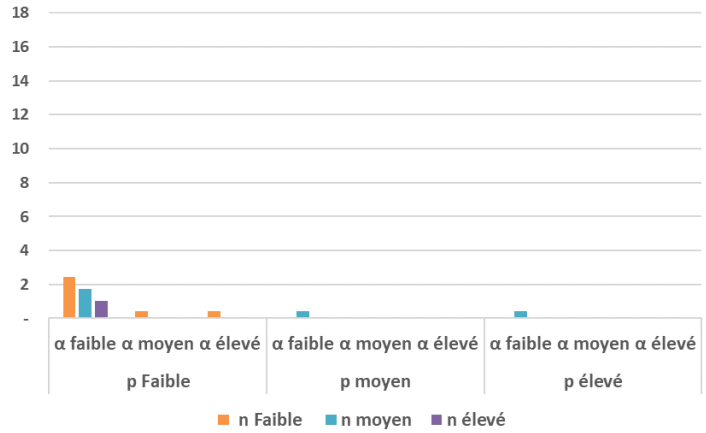
FIGURE 3.9: GAP (%) : Comparatif entre les méthodes SA_1, SA_2 et SA_{mix}



(a) Méthode SA_1

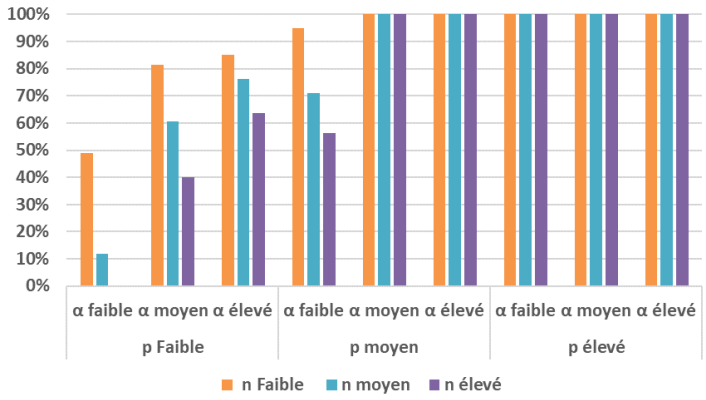


(b) Méthode SA_2

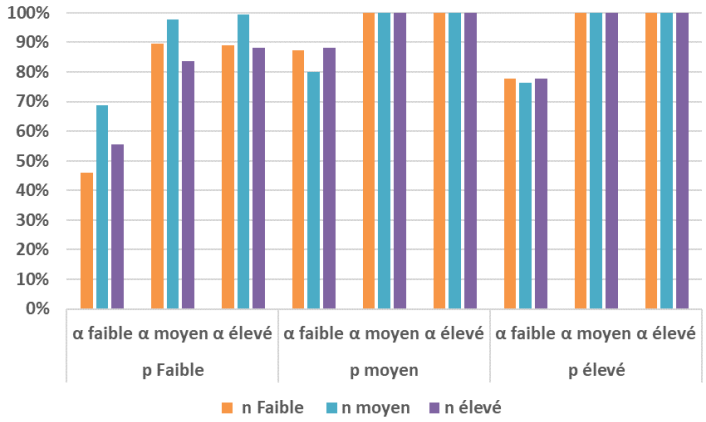


(c) Méthode SA_{mix}

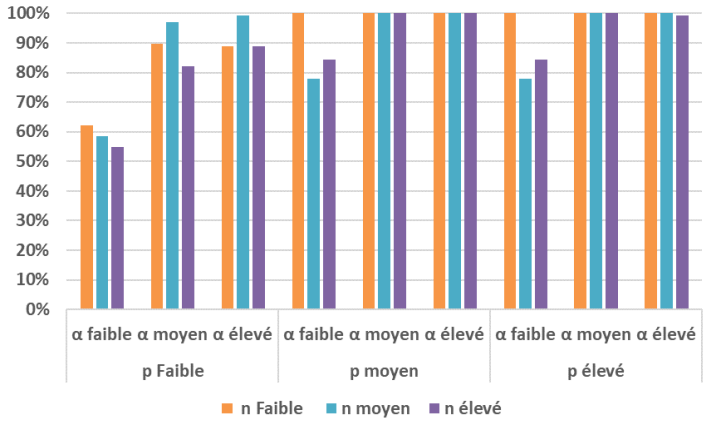
FIGURE 3.10: OPT (%) : Comparatif entre les méthodes SA_1, SA_2 et SA_{mix}



(a) Méthode SA_1



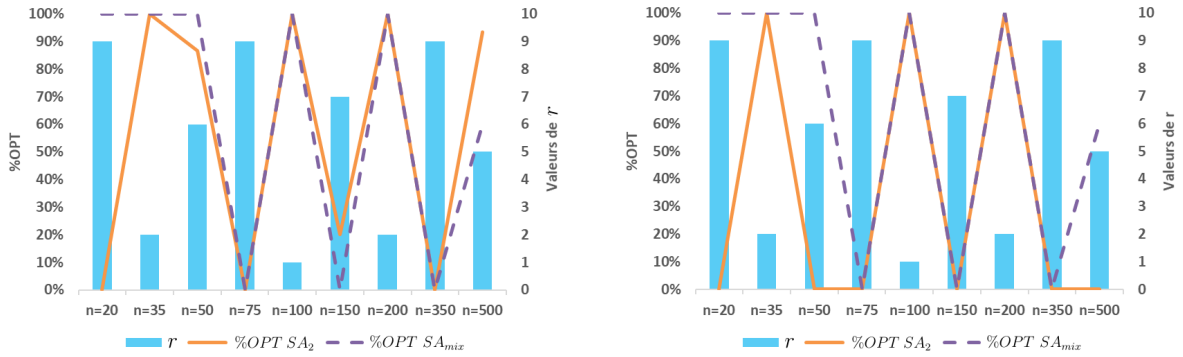
(b) Méthode SA_2



(c) Méthode SA_{mix}

Pour des graphes de moyenne et de forte densité les méthodes SA_2 et SA_{mix} n'obtiennent que des solutions optimales sauf pour le cas de $\alpha = 0,1$. En effet, lorsque les valeurs de α sont les plus basses et lorsque la densité du graphe est assez importantes l'utilisation de S_2 pour l'évaluation d'une liste peut devenir problématique pour certaines valeurs de n (voir deuxième limitation de S_2). Comme le montre la Figure 3.11, lorsque les valeurs de r (défini lors de description de la deuxième limitation de $S2$) sont élevées, les deux méthodes ont des difficultés à déterminer des solutions optimales. On peut également constater que la méthode SA_{mix} a de meilleures performances par rapport à SA_2 puisque SA_{mix} utilise également la stratégie S_1 pour l'évaluation d'une liste, ce qui va lui permettre de passer outre les limitations de S_2 dans certains cas.

FIGURE 3.11: Relation entre les valeurs de r et $\%OPT$ pour les méthodes SA_2 et SA_{mix}



(a) Graphes de moyenne densité

(b) Graphes de haute densité

La méthode SA_{mix} est supérieure à la méthode SA_2 dans pratiquement tous les aspects. Comparée à SA_1, la méthode SA_{mix} a globalement les meilleures performances, à l'exception des instances avec des graphes de concordance de forte densité et des valeurs de $\alpha = 0,1$, là ou SA_1 atteint les performances maximales. Les temps de traitement, les GAP ainsi que le nombre de solutions optimales des instances testées pour les trois méthodes sont détaillés dans les Tableaux 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9 et 3.10.

TABLE 3.2: GAP (%) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à faible densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,1$	20		0,0	6,7	7,5	4,5	2,6	2,0	2,5	2,9	4,6			97,0	27,5	13,1	5,2	2,6	1,5	1,6	2,9	4,6			0,0	6,7	7,5	4,5	2,6	1,5	1,6	2,9	4,6	
	35		0,6	9,3	6,0	4,4	0,0	0,9	0,5	0,5	0,2			77,5	12,6	4,8	2,4	0,0	0,0	0,0	0,2	0,2			0,6	10,0	5,1	2,4	0,0	0,0	0,0	0,2	0,1	
	50		1,2	13,3	7,0	5,3	0,0	0,4	0,0	0,0	0,4			43,7	5,7	1,5	0,0	0,0	0,0	0,0	0,0	0,3			1,2	6,2	2,1	0,0	0,0	0,0	0,0	0,0	0,4	
	75		8,7	15,9	9,8	5,6	1,6	2,0	1,5	0,5	0,0			29,9	0,0	0,5	0,0	0,0	0,0	0,0	0,0	0,0			8,7	0,6	0,8	0,0	0,0	0,0	0,0	0,0	0,0	
	100		9,9	16,0	10,4	6,4	2,4	2,0	1,3	0,5	0,0	5,2		15,2	0,0	0,6	0,0	0,0	0,0	0,0	0,0	0,0	4,6		9,9	0,0	0,8	0,0	0,0	0,0	0,0	0,0	0,0	1,5
	150		9,2	17,8	11,4	8,8	4,4	3,0	1,3	0,4	0,3			5,8	0,8	0,2	0,3	0,0	0,0	0,0	0,0	0,0			9,5	0,6	0,4	0,3	0,0	0,1	0,0	0,0	0,0	
	200		9,7	18,1	13,8	8,7	3,6	3,1	1,5	1,0	0,2			5,5	0,5	0,7	0,4	0,0	0,2	0,0	0,0	0,0			9,9	0,6	0,7	0,4	0,0	0,3	0,0	0,0	0,0	
	350		9,4	18,8	14,6	10,0	3,8	2,6	1,7	0,9	0,3			2,6	0,8	0,2	0,4	0,0	0,1	0,1	0,0	0,0			7,9	0,4	0,3	0,4	0,0	0,1	0,2	0,0	0,0	
	500		9,6	19,0	15,4	10,2	3,4	1,9	1,2	0,9	0,4			2,5	0,5	0,3	0,2	0,0	0,0	0,1	0,1	0,0			4,6	0,4	0,3	0,2	0,0	0,0	0,1	0,1	0,0	
$p = 0,2$	20		2,1	4,7	0,4	0,0	0,0	0,5	1,4	0,0	0,0			83,1	7,1	0,4	0,0	0,0	0,5	0,9	0,0	0,0			0,0	4,7	0,4	0,0	0,0	0,5	0,9	0,0	0,0	
	35		8,1	1,3	0,5	0,0	0,0	0,0	0,0	0,0	0,0			33,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		8,3	3,3	0,5	0,0	0,0	0,0	0,0	0,0	0,0			18,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		8,1	4,8	2,0	0,0	0,0	0,0	0,0	0,0	0,0			2,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		9,2	7,6	2,3	0,0	0,0	0,0	0,0	0,0	0,0	2,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,8		5,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,5
	150		8,9	10,1	2,9	1,4	0,4	0,1	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		9,2	11,7	5,5	1,6	0,0	0,2	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		9,5	13,4	4,8	1,7	0,1	0,2	0,1	0,0	0,0			0,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		9,6	13,6	4,9	1,6	0,1	0,1	0,0	0,0	0,0			0,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
$p = 0,3$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			42,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		6,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			6,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			3,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		6,1	0,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		7,2	0,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		8,1	0,7	0,3	0,0	0,0	0,0	0,0	0,0	0,0	1,1		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,7		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		8,5	3,0	0,3	0,0	0,0	0,0	0,0	0,0	0,0			1,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		9,0	4,2	0,7	0,1	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		9,1	5,2	0,7	0,3	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		9,4	4,9	0,8	0,1	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	

TABLE 3.3: GAP (%) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à moyenne densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,4$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			15,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		2,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		1,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		3,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,6		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0
	150		6,1	0,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		6,9	0,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		8,0	1,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		8,0	1,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
$p = 0,5$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			10,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		1,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		2,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		3,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		3,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		4,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
$p = 0,6$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			3,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		1,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		1,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		3,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		3,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		4,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	

TABLE 3.4: GAP (%) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à haute densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,7$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
$p = 0,8$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
$p = 0,9$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1
	150		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			2,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	350		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	500		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			1,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	

TABLE 3.5: CPU (S) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à faible densité

		SA_1											SA_2											SA_{mix}										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,1$	20		2,4	3,1	5,6	7,8	2,8	6,4	10,5	7,1	12,7			11,0	10,5	11,3	11,3	3,4	6,8	5,6	7,1	13,1			0,0	6,2	8,3	10,4	3,1	7,2	6,5	8,1	13,0	
	35		5,5	8,0	12,3	15,2	9,3	15,5	13,8	14,6	8,7			18,4	16,9	20,1	14,6	0,0	5,4	7,0	7,8	6,9			2,5	14,6	16,7	14,8	0,3	9,6	9,9	10,3	6,9	
	50		8,3	13,7	20,0	23,9	17,9	22,6	17,1	21,2	10,7			26,5	23,9	28,0	15,2	0,1	4,9	0,3	6,0	7,6			7,1	21,6	25,5	20,2	0,1	7,8	3,8	6,6	8,7	
	75		13,8	23,8	34,8	37,9	33,7	36,0	36,0	33,5	10,1			37,3	21,7	33,0	22,0	2,3	20,2	24,0	13,8	0,7			26,4	26,8	38,9	25,0	6,8	32,4	32,4	12,6	0,5	
	100		20,1	36,5	52,1	55,6	46,8	46,5	44,5	42,0	33,9	87,1		45,6	25,3	44,2	20,9	0,5	12,0	24,2	8,2	17,0	41,0		34,3	32,1	53,7	32,5	0,1	9,3	34,9	11,7	15,8	67,4
	150		36,9	67,5	95,7	89,9	74,0	71,2	64,9	55,9	53,9			67,0	58,3	59,2	41,9	8,5	38,2	29,3	2,2	32,5			53,7	74,3	89,6	91,7	19,1	70,4	46,1	11,3	37,4	
	200		56,5	104,6	147,8	130,7	101,9	93,5	85,9	72,8	68,7			84,5	74,9	78,2	66,3	1,2	52,6	49,9	33,6	4,1			76,2	104,4	131,7	129,7	0,2	115,6	66,8	37,1	5,4	
	350		145,0	256,1	348,7	274,0	192,9	172,4	149,5	123,5	108,2			165,0	124,6	123,2	99,8	2,3	84,8	99,0	14,1	28,8			172,7	214,4	282,1	236,9	0,8	166,0	193,8	0,9	12,6	
	500		245,1	459,0	608,9	460,5	295,6	249,1	214,9	177,5	151,7			259,6	174,2	165,6	119,4	7,7	94,4	130,1	113,5	71,3			300,8	366,7	461,8	362,0	1,1	183,5	298,7	222,6	148,2	
$p = 0,2$	20		3,4	4,5	3,2	0,4	0,2	2,4	5,1	0,1	0,0			10,5	7,7	2,1	0,0	0,0	2,3	5,3	0,0	0,0			0,0	6,0	1,8	0,1	0,0	2,3	5,5	0,5	0,0	
	35		5,9	8,5	11,5	10,9	0,7	3,1	0,3	0,0	0,0			12,9	0,1	2,0	0,0	0,0	0,0	0,0	0,0	0,0			10,4	0,1	1,0	0,1	0,0	0,1	0,1	0,2	0,0	
	50		8,5	14,0	16,3	14,9	2,1	3,8	0,1	0,3	0,0			17,3	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0			14,6	0,3	0,1	0,0	0,0	0,0	0,1	0,1	0,0	
	75		14,1	24,2	30,4	24,1	12,7	12,1	4,5	0,2	0,0			22,3	0,3	0,1	0,0	0,0	0,1	0,1	0,1	0,0			21,4	0,7	0,8	0,1	0,1	0,1	0,1	0,1	0,0	
	100		20,4	36,6	43,0	34,6	16,6	2,5	10,7	0,2	0,3	43,1		20,1	0,3	2,9	0,1	0,1	0,1	0,2	0,1	0,1	6,6		31,1	0,1	9,6	0,1	0,1	0,1	0,1	0,1	0,1	8,8
	150		36,8	67,3	71,8	56,3	38,8	35,8	5,3	0,2	0,2			28,6	22,5	0,4	0,5	0,1	0,2	0,2	0,1	0,3			43,4	26,3	1,1	0,2	0,1	0,2	0,2	0,1	0,2	
	200		55,9	104,9	102,7	74,7	15,0	38,7	23,0	1,2	0,3			46,7	3,2	7,8	0,5	0,1	0,8	0,2	0,3	0,3			57,2	2,1	19,1	0,5	0,1	0,2	0,3	0,2	0,2	
	350		133,8	254,5	201,9	135,6	41,4	75,0	51,6	1,5	0,9			111,8	4,0	1,4	4,1	0,4	0,5	1,7	0,6	0,9			154,1	13,4	1,0	5,7	0,5	1,2	0,5	0,5	0,4	
	500		244,2	446,4	316,2	198,7	102,1	70,1	63,7	18,7	1,8			160,1	10,1	4,9	3,7	0,9	2,1	2,1	4,3	2,2			240,9	10,1	6,6	2,8	0,9	1,4	11,6	1,7	1,0	
$p = 0,3$	20		2,1	3,3	0,1	0,0	0,4	0,0	0,0	0,0	0,0			8,4	0,1	0,1	0,0	0,0	0,0	0,1	0,0	0,0			0,3	0,1	0,0	0,1	0,0	0,1	0,0	0,0	0,0	
	35		6,0	5,9	3,7	2,3	0,0	0,0	0,0	0,0	0,0			6,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			6,0	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	
	50		8,6	12,2	8,3	0,6	0,0	0,0	0,0	0,0	0,0			1,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,5	0,1	0,1	0,0	0,0	0,1	0,1	0,1	0,1	
	75		14,4	21,4	19,3	4,7	0,1	0,0	0,0	0,0	0,0			18,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			18,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	
	100		20,9	32,1	28,4	3,2	0,1	0,1	0,1	0,0	0,1	23,9		4,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,2		15,6	0,1	0,2	0,1	0,1	0,2	0,1	0,1	0,1	5,1
	150		36,9	56,5	47,1	26,1	0,3	0,3	0,1	0,1	0,2			34,3	0,5	0,1	0,1	0,1	0,1	0,1	0,1	0,1			33,1	0,1	0,1	0,1	0,1	0,1	0,1	0,2	0,1	
	200		56,5	84,6	62,4	28,1	0,2	0,6	0,3	0,2	0,2			17,7	0,2	0,2	0,2	0,1	0,2	0,1	0,1	0,2			27,2	0,2	0,3	0,2	0,2	0,2	0,3	0,2	0,2	
	350		134,5	190,2	115,6	63,8	0,6	2,0	0,6	0,4	0,4			80,9	0,5	0,4	0,6	0,3	0,5	0,4	0,3	0,3			122,5	0,8	0,5	0,7	0,5	0,4	0,5	0,5	0,5	
	500		239,1	316,7	165,3	97,4	1,5	1,3	2,9	1,0	1,0			71,8	1,0	1,0	0,7	0,6	0,9	1,2	1,1	1,2			165,2	1,4	1,1	1,3	0,8	1,0	2,2	1,4	1,1	

TABLE 3.6: CPU (S) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à moyenne densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,4$	20		0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,9	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	
	35		6,2	0,9	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		9,0	8,3	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			3,3	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	
	75		14,6	15,9	0,3	0,0	0,0	0,0	0,0	0,0	0,0			17,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			16,5	0,0	0,0	0,0	0,1	0,1	0,0	0,0	0,0	
	100		21,4	23,1	1,4	0,1	0,0	0,0	0,1	0,1	0,0	13,0		0,4	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	1,9		0,8	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	4,1
	150		38,8	44,2	2,0	0,2	0,1	0,1	0,1	0,1	0,1			26,3	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1			40,6	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	
	200		61,0	62,5	7,4	0,3	0,1	0,2	0,2	0,1	0,1			0,7	0,1	0,2	0,2	0,1	0,2	0,2	0,2	0,1			1,1	0,2	0,3	0,2	0,2	0,2	0,2	0,2	0,2	
	350		140,8	120,5	1,2	1,0	0,4	0,5	0,4	0,4	0,4			70,7	0,5	0,4	0,7	0,4	0,4	0,4	0,5	0,5			123,2	0,6	0,5	1,2	0,5	0,5	0,5	0,4	0,6	
	500		244,8	177,1	42,9	1,8	0,7	0,8	1,5	0,8	0,8			18,1	0,9	1,0	0,9	0,7	0,8	1,4	1,3	0,8			123,7	1,1	1,2	1,8	0,8	0,9	1,3	1,0	1,0	
$p = 0,5$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			7,3	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	
	35		5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		6,5	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			3,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			6,9	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		13,9	0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0			16,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			17,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		21,5	0,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	6,8		0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,8		0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,7
	150		37,4	12,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0			29,6	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			37,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		55,9	3,8	0,2	0,0	0,0	0,0	0,0	0,0	0,0			0,2	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,3	0,2	0,1	0,1	0,0	0,0	0,0	0,0	0,0	
	350		127,7	18,6	0,1	0,3	0,0	0,2	0,3	0,0	0,1			56,8	0,4	0,1	0,4	0,1	0,3	0,2	0,1	0,1			109,2	0,5	0,1	0,1	0,1	0,1	0,3	0,1	0,1	
	500		215,3	28,3	0,5	0,8	0,1	0,3	0,6	0,7	0,1			27,5	0,8	0,4	0,6	0,2	0,6	0,4	0,7	0,1			121,5	1,0	0,4	0,3	0,2	0,2	0,2	0,2	0,2	
$p = 0,6$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			6,6	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		4,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		6,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			7,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		13,2	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			17,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			16,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		20,5	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	6,4		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,1		0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,9
	150		35,8	9,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0			32,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			37,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		53,8	2,3	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,1	0,1	0,1	0,1	0,0	0,1	0,0	0,0	
	350		121,8	5,5	0,1	0,4	0,0	0,3	0,3	0,0	0,0			58,0	0,1	0,1	0,4	0,1	0,3	0,1	0,1	0,1			107,6	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	
	500		206,9	32,4	0,2	1,1	0,1	0,2	1,0	0,9	0,1			39,3	0,2	0,4	0,6	0,2	0,4	0,4	0,9	0,2			145,1	0,4	0,5	0,3	0,3	0,2	0,3	0,3	0,2	

TABLE 3.7: CPU (S) : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à haute densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,7$	20		0,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,7	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			9,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,3	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0			12,4	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			16,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,2	0,1	0,0	0,0	0,1	0,1	0,1	0,0	0,0	0,2		0,0	0,8	0,0	0,1	0,1	0,0	0,0	0,0	0,0	1,9		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,1
	150		0,3	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1			20,7	0,1	0,3	0,1	0,1	0,1	0,1	0,1	0,1			37,9	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,8	0,1	0,2	0,2	0,1	0,1	0,2	0,1	0,2			0,1	0,2	0,2	0,2	0,1	0,1	0,2	0,2	0,1			0,2	0,1	0,1	0,1	0,0	0,0	0,1	0,1	0,1	
	350		0,8	0,5	0,4	0,5	0,4	0,5	0,4	0,4	0,4			39,7	0,5	0,4	0,5	0,4	0,5	0,4	0,4	0,5			109,0	0,2	0,1	0,1	0,1	0,1	0,1	0,1	0,1	
	500		3,0	1,1	0,8	1,0	0,8	0,8	0,8	0,9	0,8			52,9	1,0	1,5	0,9	0,8	1,2	0,9	1,0	0,9			72,7	0,7	1,0	0,3	0,3	0,2	0,2	0,2	0,2	
$p = 0,8$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,5	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			8,7	0,0	0,0	0,0	0,0	0,1	0,1	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			11,3	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			15,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1		0,1	6,8	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,6		0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	3,3
	150		0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0			17,9	0,1	0,4	0,0	0,0	0,0	0,0	0,0	0,0			33,6	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,2	0,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,2	0,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,1	0,0	0,0	0,1	0,0	0,0	0,1	0,0	0,0	
	350		0,8	0,5	0,1	0,2	0,1	0,1	0,2	0,1	0,1			33,5	0,6	0,1	0,2	0,1	0,2	0,2	0,1	0,1			95,0	0,2	0,1	0,1	0,1	0,2	0,1	0,1	0,1	
	500		1,0	0,9	0,2	0,4	0,2	0,3	0,4	0,5	0,2			41,9	1,5	0,5	0,3	0,2	0,9	0,5	0,7	0,2			106,0	0,4	0,4	0,2	0,3	0,3	0,3	0,2	0,3	
$p = 0,9$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			5,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	35		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1	
	50		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			7,9	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			6,7	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	75		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			10,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			15,2	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	100		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,1		0,0	5,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	1,6		0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	2,9
	150		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			18,9	0,0	0,2	0,0	0,0	0,0	0,0	0,0	0,0			33,0	0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	
	200		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,0	0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0			0,1	0,1	0,0	0,0	0,0	0,0	0,1	0,0	0,0	
	350		0,1	0,1	0,1	0,1	0,1	0,2	0,2	0,1	0,1			32,0	0,1	0,2	0,1	0,1	0,1	0,1	0,1	0,1			94,1	0,3	0,1	0,1	0,1	0,1	0,1	0,1	0,2	
	500		0,2	0,2	0,2	0,6	0,2	0,3	0,3	0,4	0,2			43,4	0,5	1,0	0,5	0,2	0,6	0,4	0,6	0,2			82,6	0,6	0,6	0,3	0,2	0,3	0,2	0,2	0,3	

TABLE 3.8: NbrOpt : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à faible densité

		SA_1											SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,1$	20		5	1	0	0	3	2	1	2	0			0	0	0	0	3	2	3	2	0			5	1	0	0	3	2	3	2	0	
	35		4	0	0	0	5	3	3	3	4			0	0	0	2	5	5	5	4	4			4	0	0	2	5	5	5	4	4	
	50		3	0	0	0	5	3	5	5	4			0	0	0	5	5	5	5	5	4			3	0	0	5	5	5	5	5	4	
	75		0	0	0	0	1	0	0	1	5			0	5	3	5	5	5	5	5	5			0	3	1	5	5	5	5	5	5	
	100		0	0	0	0	0	0	0	1	5	18,5		0	5	1	5	5	5	5	5	5	54,3		0	5	0	5	5	5	5	5	5	55,6
	150		0	0	0	0	0	0	0	1	0			0	0	3	2	5	5	5	5	4			0	1	1	2	5	4	5	5	4	
	200		0	0	0	0	0	0	0	0	0			0	1	0	0	5	2	4	5	5			0	0	0	0	5	1	4	5	5	
	350		0	0	0	0	0	0	0	0	0			0	0	0	0	5	2	0	5	5			0	1	0	0	5	2	0	5	5	
	500		0	0	0	0	0	0	0	0	0			0	0	0	1	5	3	0	1	4			0	0	0	0	5	3	0	1	5	
$p = 0,2$	20		3	1	4	5	5	4	3	5	5			0	1	4	5	5	4	3	5	5			5	1	4	5	5	4	3	5	5	
	35		0	3	4	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	50		0	0	4	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	75		0	0	0	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		0	0	0	5	5	5	5	5	5	60,7		5	5	5	5	5	5	5	5	5	90,6		0	5	5	5	5	5	5	5	5	89,6
	150		0	0	0	0	3	3	5	5	5			5	5	5	5	5	5	5	5	5			2	5	5	5	5	5	5	5	5	
	200		0	0	0	0	5	3	4	5	5			5	5	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5	
	350		0	0	0	0	4	1	3	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		0	0	0	0	3	2	4	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
$p = 0,3$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		0	5	5	5	5	5	5	5	5			2	5	5	5	5	5	5	5	5			2	5	5	5	5	5	5	5	5	
	50		0	4	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5	
	75		0	3	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		0	2	4	5	5	5	5	5	5	76,8		5	5	5	5	5	5	5	5	5	94,3		5	5	5	5	5	5	5	5	5	95,3
	150		0	0	2	5	5	5	5	5	5			1	5	5	5	5	5	5	5	5			1	5	5	5	5	5	5	5	5	
	200		0	0	0	4	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		0	0	0	1	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		0	0	0	1	5	5	5	5	5			4	5	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5	

TABLE 3.9: NbrOpt : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à moyenne densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,4$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		1	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		2	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		0	5	5	5	5	5	5	5	5	86,4		5	5	5	5	5	5	5	5	5	95,6		5	5	5	5	5	5	5	5	5	96,0
	150		0	1	5	5	5	5	5	5	5			2	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		0	1	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		0	0	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		0	0	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5	
$p = 0,5$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		0	5	5	5	5	5	5	5	5	93,8		5	5	5	5	5	5	5	5	5	95,3		5	5	5	5	5	5	5	5	5	95,8
	150		0	5	5	5	5	5	5	5	5			1	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			3	5	5	5	5	5	5	5	5	
$p = 0,6$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			3	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		0	5	5	5	5	5	5	5	5	93,8		5	5	5	5	5	5	5	5	5	94,3		5	5	5	5	5	5	5	5	5	95,6
	150		0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		0	5	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5			2	5	5	5	5	5	5	5	5	

TABLE 3.10: NbrOpt : méthodes SA_1, SA_2 et SA_{mix} pour des graphes de concordance à haute densité

		SA_1										SA_2										SA_{mix}									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,7$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	5	5	5	5	5	5	5	5	92,6		5	5	5	5	5	5	5	5	5	95,8
	150		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			3	5	5	5	5	5	5	5	5	
$p = 0,8$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	3	5	5	5	5	5	5	5	92,1		5	5	5	5	5	5	5	5	5	95,8
	150		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			3	5	5	5	5	5	5	5	5	
$p = 0,9$	20		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	5	5	5	5	5	5	5	5	92,6		5	5	5	5	5	5	5	5	5	95,6
	150		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	350		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5	
	500		5	5	5	5	5	5	5	5	5			0	5	5	5	5	5	5	5	5			3	5	5	5	5	5	5	5	4	

3.8.3 Méthodes Exactes

Dans cette section nous allons expérimenter la méthode branch and boud et la résolution du modèle mathématique. La résolution du modèle se fait en deux phases :

- Phase 1 : résoudre le modèle avec le nombre minimum de compartiments possibles $n' = \lceil \frac{n}{a+b} \rceil$.
- Phase 2 : si aucune solution optimale n'est trouvée à la phase 1, résoudre le modèle avec $n' = n$.

Les tableaux 3.11, 3.12, 3.13, 3.14, 3.15 et 3.16 représentent le nombre de solutions optimales et les temps d'exécution de la méthode branch and bound et de la résolution du modèle pour des graphes de faible, moyenne, et haute densité.

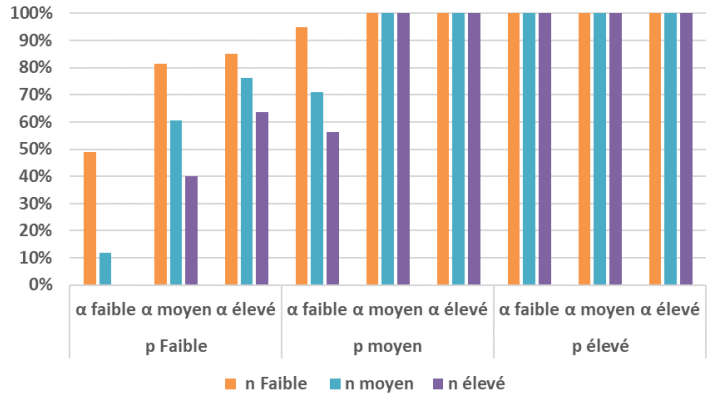
Etant donné que les méthodes métaheuristiques testées dans la section précédente, performent étonnamment bien quand il s'agit de nombre de solutions optimales. Nous allons inclure SA_1 et SA_{mix} lors de la comparaison des méthodes exactes (nous avons exclu SA_2, puisque SA_{mix} surpasse SA_2 dans tous les aspects). Les graphes comparant le pourcentage de solutions optimales sont données dans la Figure 3.12. Nous allons à présent comparer les 4 méthodes.

- Faibles valeurs de α :

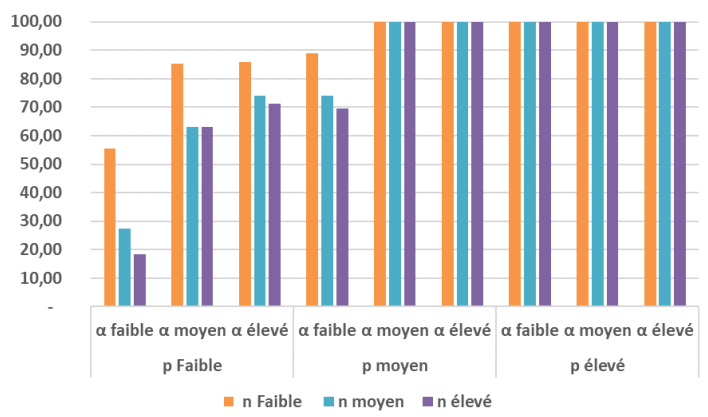
  - $n \leq 50$: la résolution du modèle, surclasse les autres méthodes avec 100% de solutions optimales (aussi bien que la méthode SA_{mix}) lorsque la densité du graphe est moyenne à forte. Lorsque la densité du graphe est faible, la résolution du modèle performe bien mieux que les autres méthodes avec 81,5% de solutions optimales (55,6% pour la méthode branch and bound, 48,9% pour SA_1 et $62,2\%$ pour SA_{mix}).

  - $75 \leq n \leq 150$: aucune méthode ne se détache des autres. Lorsque la densité du graphe est faible la méthode SA_{mix} a les meilleures performances avec 59% de solutions optimales. Avec des graphes de densité moyenne les trois méthodes ont des performances similaires avec 78% de solutions optimales pour SA_{mix}, 74% pour branch and bound, 73% pour la résolution du modèle et enfin 71% pour SA_1. Lorsque le graphe de concordance a une forte densité, les méthodes SA_1 et branch and bound ont des performances maximales avec 100% de solutions optimales, suivies par la résolution du modèle avec 95% de solutions optimales. La méthode SA_{mix} arrive en dernier avec 84% de solutions optimales.

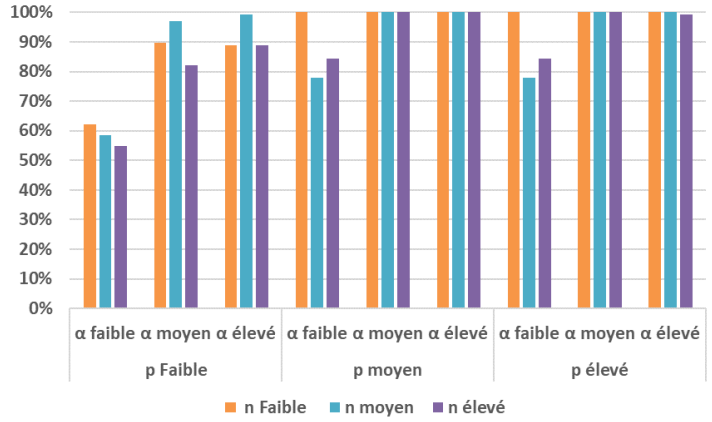
FIGURE 3.12: OPT (%) : Comparatif entre les méthode SA_1, SA_{mix}, Branch and Bound et la résolution du modèle



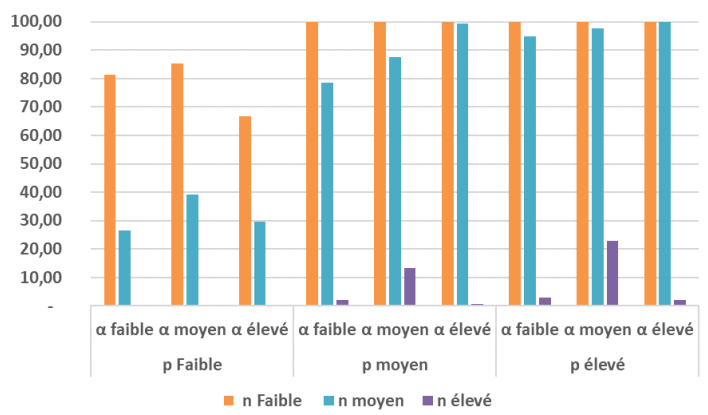
(a) Méthode SA_1



(b) Méthode Branch and Bound

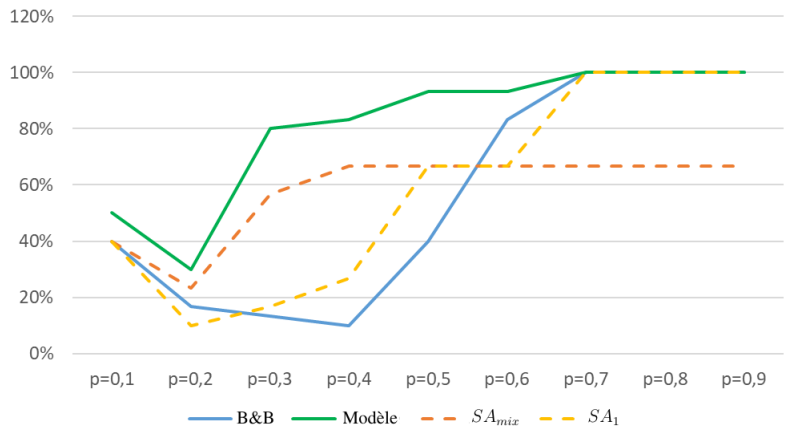


(c) Méthode SA_{mix}



(d) Résolution du modèle

FIGURE 3.13: OPT (%) : $\alpha = 0,1$ et $n \leq 150$

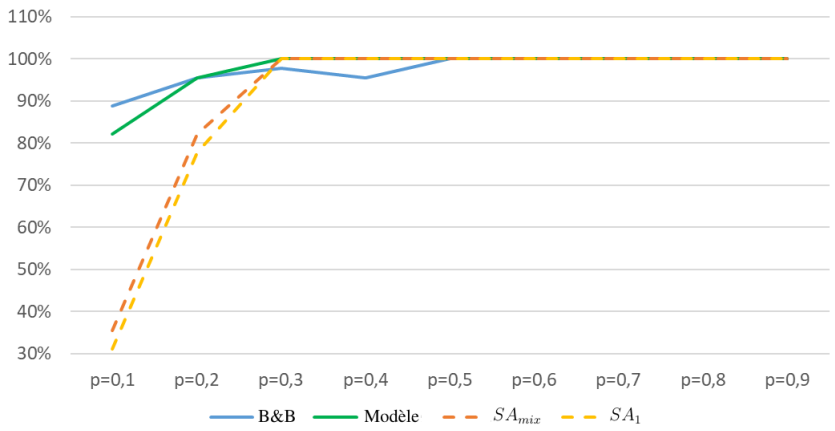


– $200 \leq n \leq 500$: avec un nombre de tâches importants, les performances de la résolution du modèle s'effondrent avec à peine 2% de solutions optimales. Lorsque le graphe de concordance a une faible à moyenne densité, la méthode SA_{mix} a les meilleures performances avec respectivement 55% et 70% de solutions optimales pour les graphes de faible et moyenne densité. Avec des graphes de concordance à forte densité les méthodes SA_1 et branch and bound dominent la méthode SA_{max} avec 100% de solutions optimales.

– Moyennes et fortes valeurs de α : La méthode SA_{mix} a les meilleures performances comparée aux autres méthodes avec 100% de solutions optimales lorsque $p \geq 0,3$. Lorsque $p \leq 0,2$ la méthode SA_{mix} garde généralement les meilleures performances avec 85% de solution optimales.

La méthode SA_1 a d'excellentes performances lorsqu'il s'agit de graphes à forte densité, cependant la méthode branch and bound surperforme SA_1 dans quasiment tous les aspects. Quant à la méthode SA_{mix}, celle-ci a de très bonnes performances lorsque le graphe de concordance a une faible et moyenne densité, cependant, SA_{mix} peine à trouver des solutions optimales pour les plus petites valeurs de n ($n = 20$) ou lorsque le graphe de concordance a une faible densité, plus particulièrement pour les petites valeurs de α (α=0,1). En effet, la résolution du modèle améliore grandement les performances des autres méthodes lorsque $\alpha = 0,1$, $n \leq 150$ pour des graphes de faible à moyenne densité comme le montre la Figure 3.13. Lorsque $n = 20$ et $p \leq 0,2$ la résolution du modèle procure un pourcentage maximal de solutions optimales pour des valeurs faibles de α, une nette amélioration comparée à SA_{mix} qui est à 53% de solutions optimales. Même lorsque les valeurs de α sont arbitraires la résolution du modèle et même la

FIGURE 3.14: OPT (%) : $\forall\alpha$ et $n = 20$



méthode branch and bound font mieux que la méthode SA_{mix} pour $n = 20$ et $p \leq 0,2$ (voir Figure 3.14). A noter que même si la résolution du modèle propose bon nombre d'améliorations par rapport aux autres méthodes, les performances de celle-ci s'effondrent pour des valeurs élevées de n.

3.9 Conclusion

Dans ce chapitre, nous avons considéré le problème SWA sur deux machines uniformes. Nous avons prouvé que le problème est NP-difficile au sens fort. Nous avons ensuite mis en place des bornes inférieures, des méthodes de résolution approchées, et des méthodes de résolution exactes. Les méthodes approchées ont obtenu d'excellents résultats avec des méthodes atteignant un GAP moyen de moins de 1% (pour la méthode SA_{mix}). Les pourcentages de solutions optimales fournies par les méthodes approchées rivalisent avec ceux obtenus par les méthodes de résolutions exactes. Les méthodes de résolution exactes ont globalement des résultats satisfaisants avec de solides performances particulièrement pour les cas les plus difficiles à résoudre par les méthodes de résolution approchées (faible valeurs de p et de α).

TABLE 3.11: NbrOpt : Branch and bound et résolution du modèle pour des graphes de concordance à faible densité

		B&B										Modèle										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,1$	20		5	4	3	5	5	5	5	5	3			5	5	5	5	5	5	3	4	0	
	35		4	0	0	0	4	2	4	1	1			5	2	0	0	5	0	1	0	0	
	50		3	0	0	0	4	2	3	3	1			5	0	0	0	5	0	0	0	0	
	75		0	0	0	0	0	0	0	2	1			0	0	0	0	5	0	0	0	0	
	100		0	0	0	0	0	0	0	2	1	22,0		0	0	0	0	5	0	0	0	0	17,3
	150		0	0	0	0	0	0	0	2	2			0	0	0	0	0	0	0	0	0	
	200		0	0	0	0	0	0	0	2	2			0	0	0	0	0	0	0	0	0	
	350		0	0	0	0	0	0	0	2	0			0	0	0	0	0	0	0	0	0	
	500		0	0	0	0	0	0	0	0	1			0	0	0	0	0	0	0	0	0	
$p = 0,2$	20		5	3	5	5	5	5	5	5	5			5	5	5	5	5	5	4	5	4	
	35		0	1	5	4	5	5	5	5	5			4	5	5	5	5	5	5	5	4	
	50		0	1	3	4	5	5	5	5	5			0	5	5	5	5	5	5	5	0	
	75		0	2	4	5	5	4	5	5	5			0	5	0	4	5	0	1	0	0	
	100		0	0	2	5	5	5	5	5	5	72,8		0	2	0	0	5	0	0	0	0	36,3
	150		0	0	3	3	5	3	5	5	5			0	4	0	0	0	0	0	0	0	
	200		0	0	1	4	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	350		0	0	2	4	5	5	4	5	5			0	0	0	0	0	0	0	0	0	
	500		0	0	1	4	4	4	5	5	5			0	0	0	0	0	0	0	0	0	
$p = 0,3$	20		4	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		0	4	5	5	5	5	5	5	5			4	5	5	5	5	5	5	5	5	
	50		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		0	4	5	5	5	5	5	5	5	86,4		5	5	0	4	5	5	5	5	5	56,0
	150		0	2	5	5	5	5	5	5	5			0	0	0	0	0	0	3	1	5	
	200		0	4	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	350		0	3	4	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		0	0	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	

TABLE 3.12: NbrOpt : Branch and bound et résolution du modèle pour des graphes de concordance à moyenne densité

		B&B										Modèle										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,4$	20		3	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		0	5	5	5	5	5	5	5	5	89,6		5	5	5	5	5	5	5	5	5	62,0
	150		0	5	5	5	5	5	5	5	5			0	0	0	0	5	3	4	5	5	
	200		0	5	5	5	5	5	5	5	5			0	0	0	0	4	0	0	0	0	
	350		0	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		0	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
$p = 0,5$	20		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		4	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		3	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		0	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		0	5	5	5	5	5	5	5	5	91,9		5	5	5	5	5	5	5	5	5	66,7
	150		0	5	5	5	5	5	5	5	5			3	5	0	4	5	5	5	5	5	
	200		0	5	5	5	5	5	5	5	5			0	1	0	0	4	2	0	1	0	
	350		0	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		0	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
$p = 0,6$	20		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		3	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	96,0		5	5	5	5	5	5	5	5	5	65,2
	150		2	5	5	5	5	5	5	5	5			3	5	0	0	5	1	5	5	5	
	200		1	5	5	5	5	5	5	5	5			0	2	0	0	5	3	0	0	0	
	350		1	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		2	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	

TABLE 3.13: NbrOpt : Branch and bound et résolution du modèle pour des graphes de concordance à haute densité

		B&B											Modèle									
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	%
$p = 0,7$	20		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	5	5	5	5	5	5	5	5	66,7
	150		5	5	5	5	5	5	5	5	5			5	3	0	2	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			0	0	0	0	5	5	0	0	0	
	350		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
$p = 0,8$	20		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	5	5	5	5	5	5	5	5	69,9
	150		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			0	1	0	0	5	5	1	1	0	
	350		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
$p = 0,9$	20		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	35		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	50		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	75		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	100		5	5	5	5	5	5	5	5	5	100		5	5	5	5	5	5	5	5	5	70,4
	150		5	5	5	5	5	5	5	5	5			5	5	5	5	5	5	5	5	5	
	200		5	5	5	5	5	5	5	5	5			1	1	1	1	5	5	0	1	0	
	350		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	
	500		5	5	5	5	5	5	5	5	5			0	0	0	0	0	0	0	0	0	

TABLE 3.14: CPU (S) : Branch and bound et résolution du modèle pour des graphes de concordance à faible densité

		B&B										Modèle										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,1$	20		0,1	720,1	1560,9	44,1	97,1	6,7	44,1	132,1	2162,4			0,3	0,4	183,7	52,8	1,9	52,4	1571,7	824,6	>3600	
	35		720,1	>3600	>3600	>3600	926,2	2224,6	901,1	2885,4	2880,4			3,6	2537,2	>3600	>3600	3,7	>3600	2946,8	>3600	>3600	
	50		1440,1	>3600	>3600	>3600	810,9	2317,0	2368,2	1526,7	2880,0			79,1	>3600	>3600	>3600	11,8	>3600	>3600	>3600	>3600	
	75		>3600	>3600	>3600	>3600	>3600	>3600	>3600	2323,0	2880,0			>3600	>3600	>3600	>3600	283,9	>3600	>3600	>3600	>3600	
	100		>3600	>3600	>3600	>3600	>3600	>3600	>3600	2167,7	3372,9	2866,4		>3600	>3600	>3600	>3600	404,2	>3600	>3600	>3600	>3600	2999,5
	150		>3600	>3600	>3600	>3600	>3600	>3600	>3600	2515,4	2485,7			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	200		>3600	>3600	>3600	>3600	>3600	>3600	>3600	2360,6	2348,6			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	350		>3600	>3600	>3600	>3600	>3600	>3600	>3600	2173,8	>3600			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	2896,5			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,2$	20		0,1	1953,5	101,7	0,1	0,0	11,7	75,8	0,0	0,1			0,2	3,6	154,6	15,1	0,5	652,1	1117,7	709,4	785,0	
	35		>3600	2880,4	427,7	721,8	0,0	0,9	3,1	0,2	0,1			1211,3	13,0	327,9	52,3	3,2	53,0	731,1	44,2	871,2	
	50		>3600	2893,2	1511,8	724,7	0,2	0,1	0,1	0,1	0,1			>3600	93,9	552,5	244,8	7,0	447,9	304,2	336,9	>3600	
	75		>3600	2362,4	1422,9	23,6	0,5	720,1	0,2	0,2	24,5			>3600	485,1	>3600	2025,9	279,8	>3600	3492,7	>3600	>3600	
	100		>3600	>3600	2165,5	24,3	0,3	42,1	0,4	0,2	2,7	1081,5		>3600	2668,6	>3600	>3600	329,0	>3600	>3600	>3600	>3600	2441,4
	150		>3600	>3600	1914,0	2005,5	7,2	1455,1	0,6	15,3	13,8			>3600	3187,1	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	200		>3600	>3600	2883,8	986,2	13,2	1,6	2,8	3,1	38,2			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	350		>3600	>3600	2371,0	1105,4	41,5	561,4	748,8	60,2	1429,9			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		>3600	>3600	2884,1	973,5	842,3	746,3	764,3	71,1	765,5			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,3$	20		741,2	53,1	0,1	0,0	0,0	0,0	0,0	0,0	0,0			0,3	1,1	3,2	1,6	0,4	7,5	5,1	3,3	1702,8	
	35		>3600	720,9	0,1	0,1	0,0	0,1	0,0	0,0	0,0			1474,7	2,8	93,7	19,0	1,7	101,8	30,3	24,3	51,9	
	50		>3600	13,0	0,3	0,1	0,1	0,1	0,1	0,1	0,0			57,9	29,2	127,6	77,7	5,1	245,6	4,7	210,7	521,9	
	75		>3600	59,2	10,5	0,1	0,2	0,1	0,1	0,1	0,1			409,4	415,7	147,0	1321,3	204,3	1338,4	52,7	97,4	10,0	
	100		>3600	967,4	27,6	39,0	0,3	0,1	6,6	0,2	0,1	531,9		2825,7	2157,1	>3600	1163,3	659,7	110,6	184,3	180,9	821,8	1792,1
	150		>3600	3163,8	83,9	0,5	1,5	0,4	0,5	0,4	0,3			>3600	>3600	>3600	>3600	>3600	>3600	2446,0	2984,2	422,2	
	200		>3600	724,0	261,4	1,5	4,6	0,8	0,8	1,0	0,8			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	350		>3600	1491,4	765,2	375,4	42,6	5,4	8,0	5,0	5,9			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		>3600	>3600	26,8	24,1	151,6	24,0	38,9	32,8	791,6			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	

TABLE 3.15: CPU (S) : Branch and bound et résolution du modèle pour des graphes de concordance à moyenne densité

		B&B										Modèle										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,4$	20		1675,7	0,0	0,1	0,0	0,0	0,0	0,0	0,0	0,1			0,4	0,4	2,4	0,9	0,3	0,3	1,0	0,6	0,7	
	35		>3600	9,5	0,1	0,1	0,1	0,1	0,0	0,0	0,0			3,7	2,1	10,8	6,0	0,3	2,1	3,2	0,6	2,4	
	50		>3600	0,1	0,1	0,1	0,1	0,1	0,0	0,1	0,1			15,6	11,9	56,7	28,6	1,9	7,2	1,6	6,0	10,8	
	75		>3600	0,2	33,6	0,1	0,1	0,1	0,1	0,1	0,1			100,9	50,6	215,1	261,8	9,8	23,9	89,8	18,1	5,9	
	100		>3600	0,8	0,2	0,1	0,3	0,1	0,2	0,1	0,1	385,5		1186,4	790,5	1332,7	374,5	38,1	99,0	71,0	69,2	410,7	1485,8
	150		>3600	1,2	1,2	0,3	1,8	0,4	0,4	0,5	0,3			>3600	>3600	>3600	>3600	1992,0	1779,7	1125,5	881,8	291,0	
	200		>3600	22,6	0,8	26,8	3,7	3,5	0,9	0,8	0,9			>3600	>3600	>3600	>3600	949,1	>3600	>3600	>3600	>3600	
	350		>3600	44,5	5,6	4,9	34,1	6,3	6,5	4,8	7,6			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		>3600	208,1	22,0	35,6	168,4	18,6	28,9	18,4	20,8			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,5$	20		263,7	0,0	0,1	0,0	0,1	0,0	0,1	0,1	0,1			0,4	0,6	0,8	0,7	0,3	1,4	2,8	1,9	4,3	
	35		1464,3	0,1	0,1	0,0	0,1	0,1	0,0	0,0	0,1			3,0	1,6	6,4	6,8	0,9	20,4	6,7	3,9	15,2	
	50		1921,8	0,1	0,1	0,1	0,1	0,1	0,1	0,0	0,1			15,8	7,2	21,5	16,8	3,4	30,9	94,2	13,3	11,3	
	75		>3600	0,2	0,1	0,1	0,2	0,1	0,1	0,1	0,1			88,2	56,0	141,7	565,6	16,5	1198,1	17,4	15,5	4,1	
	100		>3600	0,3	0,1	0,2	0,4	0,1	0,1	0,1	0,1	319,9		1579,4	236,5	793,6	191,1	27,0	76,3	50,8	77,6	55,3	1402,4
	150		>3600	1,5	0,6	0,3	2,3	0,4	0,3	0,3	0,4			1797,0	1745,1	>3600	3479,2	835,7	1130,6	986,4	987,6	224,5	
	200		>3600	2,7	0,8	0,8	4,2	0,8	0,9	0,8	0,7			>3600	>3600	>3600	>3600	918,1	2301,5	>3600	>3600	>3600	
	350		>3600	42,7	3,0	4,4	45,6	4,7	4,9	3,9	5,3			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		>3600	259,3	21,1	20,2	153,5	19,1	17,2	14,6	19,0			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,6$	20		25,4	0,0	0,1	0,1	0,0	0,0	0,1	0,1	0,1			0,5	0,5	1,2	1,5	0,3	1,6	1,4	1,1	2,7	
	35		1226,7	0,0	0,0	0,0	0,0	0,1	0,0	0,3	0,0			2,4	2,0	16,7	4,1	1,1	12,0	7,3	5,5	12,3	
	50		133,7	0,1	0,0	0,0	0,1	0,1	0,1	0,1	0,1			10,3	8,2	23,9	24,2	2,5	109,9	29,9	10,8	32,3	
	75		2410,9	0,2	0,1	0,1	0,2	0,1	0,1	0,1	0,1			48,2	36,6	99,4	379,7	8,5	402,9	49,3	50,8	3,4	
	100		620,6	0,3	0,1	0,2	0,3	0,1	0,2	0,3	0,3	196,9		777,0	353,6	222,2	637,6	54,8	54,7	40,8	38,8	44,4	1370,7
	150		2381,4	1,0	0,3	0,4	1,8	0,3	0,2	0,2	0,3			1673,7	2722,3	>3600	>3600	146,5	3590,6	939,2	215,7	218,4	
	200		3051,1	2,6	0,8	0,8	5,4	0,7	0,6	0,5	0,7			>3600	2348,7	>3600	>3600	280,2	1632,3	>3600	>3600	>3600	
	350		2891,5	27,2	3,2	4,1	37,0	4,0	3,9	3,8	4,1			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		2759,1	100,9	26,8	18,8	149,2	13,5	11,9	10,3	12,5			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	

TABLE 3.16: CPU (S) : Branch and bound et résolution du modèle pour des graphes de concordance à haute densité

		B&B										Modèle										
	n	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr	α	.1	.2	.3	.4	.5	.6	.7	.8	.9	Avr
$p = 0,7$	20		1,8	0,0	0,0	0,0	0,0	0,0	0,0	0,1	0,0			0,2	0,2	0,7	0,4	0,2	0,6	0,7	1,4	7,1	
	35		0,1	0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,0			0,7	0,7	7,8	2,6	0,5	11,4	4,1	2,0	4,7	
	50		0,1	0,0	0,0	0,0	0,1	0,0	0,1	0,1	0,0			2,7	2,1	22,3	14,9	1,3	13,5	16,0	7,9	11,5	
	75		164,3	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1			22,2	9,8	29,4	34,7	6,6	213,4	9,1	5,9	2,3	
	100		0,2	0,2	0,1	0,1	0,3	0,1	0,1	0,1	0,1	7,9		125,0	36,4	68,7	410,9	27,4	37,7	28,7	31,5	24,9	1249,3
	150		0,9	0,7	0,2	0,3	1,1	0,3	0,3	0,3	0,2			202,1	2289,1	>3600	2581,4	34,1	283,9	201,7	194,3	135,5	
	200		1,6	2,2	0,6	0,6	3,6	0,6	0,7	0,6	0,6			>3600	>3600	>3600	>3600	208,6	135,0	>3600	>3600	>3600	
	350		13,2	21,5	2,7	3,1	32,5	4,9	4,7	3,3	3,7			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		82,2	66,7	13,7	13,1	134,5	14,6	12,2	11,5	14,0			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,8$	20		0,0	0,1	0,0	0,1	0,1	0,2	0,0	0,1	0,0			0,2	0,2	0,4	0,3	0,2	0,5	0,6	1,4	1,6	
	35		0,1	0,0	0,0	0,1	0,1	0,0	0,0	0,0	0,1			0,4	0,5	5,0	1,7	0,5	2,7	6,0	1,6	7,4	
	50		0,1	0,1	0,1	0,1	0,1	0,2	0,0	0,0	0,1			1,4	1,4	5,9	4,5	1,0	3,7	7,8	5,2	19,3	
	75		0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,2	0,1			4,6	3,7	15,9	34,4	3,9	38,9	25,8	43,3	1,9	
	100		0,2	0,2	0,1	0,1	0,2	0,1	0,1	0,1	0,1	4,4		106,8	15,9	165,5	15,0	10,5	9,0	9,9	9,5	7,6	1132,8
	150		0,4	0,6	0,2	0,2	1,2	0,3	0,2	0,3	0,2			889,6	174,2	1204,4	66,4	101,2	49,2	45,3	39,9	58,5	
	200		0,9	1,6	0,5	0,5	3,1	1,0	0,6	0,7	0,7			>3600	3250,0	>3600	>3600	79,4	66,8	2911,5	2912,3	>3600	
	350		10,1	16,1	2,6	2,9	27,4	3,4	3,1	3,1	3,1			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		30,7	56,5	8,8	10,4	112,2	11,4	11,3	16,3	11,1			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
$p = 0,9$	20		0,0	0,0	0,0	0,0	0,0	0,0	0,0	0,2	0,2			0,1	0,2	0,3	0,4	0,3	0,4	0,7	1,2	10,7	
	35		0,0	0,0	0,0	0,0	0,1	0,0	0,0	0,0	0,0			0,4	0,4	2,1	2,0	0,7	5,6	6,0	2,3	14,2	
	50		0,0	0,1	0,1	0,0	0,1	0,0	0,2	0,0	0,0			0,9	0,9	7,2	2,7	1,0	17,1	7,8	6,1	11,2	
	75		0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1	0,1			2,8	2,2	12,7	11,4	3,8	39,6	56,2	18,9	1,8	
	100		0,1	0,1	0,1	0,1	0,2	0,1	0,1	0,1	0,1	3,8		6,9	6,1	7,0	131,8	7,2	7,2	5,8	6,8	6,8	1080,5
	150		0,3	0,5	0,2	0,2	0,9	0,2	0,2	0,3	0,3			24,3	25,1	29,6	30,7	27,9	27,5	21,9	20,4	19,9	
	200		0,8	1,4	0,5	0,5	2,6	0,5	0,7	0,5	0,5			2941,7	2896,0	3123,1	2904,6	39,5	53,7	>3600	2894,9	>3600	
	350		6,5	12,9	2,4	2,6	23,7	3,1	3,1	3,2	2,9			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	
	500		26,6	48,3	8,2	10,6	96,1	10,5	12,4	10,3	12,2			>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	>3600	

Conclusion et perspectives

Dans cette thèse nous avons étudié le problème SWA sur deux machines parallèles. Le tableau 3.17 résume tous les résultats de complexité présentés.

TABLE 3.17: Résultats de complexité du problème d'ordonnancement avec graphe de concordance sur deux machines parallèles

Machines	Temps de traitement	Type de Graphes	Complexité	Référence
Identiques	$\forall p_j$	Arbre	NP-difficile	Théorème 1
Identiques	$\forall p_j$	Caterpillar	$O(n)$	Théorème 2
Identiques	$\forall p_j$	Chaine	$O(n)$	Corollaire 1
Identiques	$\forall p_j$	Etoile	$O(n)$	Corollaire 1
Identiques	$\forall p_j$	Cycle	$O(n^2)$	Corollaire 2
Uniformes	1	Arbitraire	NP-difficile au sens fort	Théorème 3

Comme le problème SWA sur deux machines uniformes est NP-difficile. Nous avons proposé un modèle mathématique, une méthode métaheuristique et une méthode branch and bound pour résoudre le problème. Nous avons opté pour la métaheuristique Simulated Annealing. Nous avons proposé trois variantes SA_1, SA_2, et SA_{mix}. Les trois variantes ont des performances très satisfaisantes. La méthode SA_{mix} a les meilleures performances pour des graphes de faible et moyenne densité avec des GAP respectifs de 0,7% et 0,1%. Lorsque la densité du graphe est haute, la méthode SA_1 a les meilleures performances avec aucun GAP.

Pour ce qui est du pourcentage de solutions optimales, la méthode SA_{mix} a globalement les meilleures performances et dépasse même les méthodes de résolution exactes pour des valeurs moyennes et élevées de α. Même si dans d'autres cas, les méthodes exactes ont les meilleures performances. En effet, pour de très faible valeurs de α et $n \leq 150$ la résolution du modèle reste la meilleure. De même, lorsque la densité du graphe est élevé, la méthode branch and bound fait mieux que la méthode SA_{mix} avec 100% de solutions optimales.

Concernant les perspectives de recherche, nous suggérons quelques axes d'amélioration :

– Etudier la complexité du problème SWA sur deux machines uniformes et des temps de traitement unitaires pour des classes de graphes particulières.

– Etudier la complexité du problème sur deux machines uniformes et des temps de traitement arbitraires pour tous les cas polynomiaux sur deux machines uniformes.

– Etant donné que le problème SWA sur deux machines uniformes lorsque le graphe de concordance est un arbre est NP-difficile au sens faible, il serait judicieux de développer des méthodes de résolution spécifiques à cette classe de graphes et éventuellement développer un algorithme pseudo-polynomial pour résoudre le problème.

– Développer des méthodes de résolution pour un nombre de machines $m \geq 3$ et les comparer aux adaptations des méthodes de résolution présentées dans cette thèse pour un nombre de machines $m \geq 3$.

– Considérer le problème pour d'autres objectifs.

Bibliographie

[1] Brenda S Baker and Edward G Coffman Jr. Mutual exclusion scheduling. *Theoretical Computer Science*, 162(2) :225–243, 1996.

[2] Mohamed Bendraouche and Mourad Boudhar. Scheduling jobs on identical machines with agreement graph. *Computers & Operations Research*, 39(2) :382–390, 2012.

[3] Mohamed Bendraouche and Mourad Boudhar. Scheduling with agreements : new results. *International Journal of Production Research*, 54(12) :3508–3522, 2016.

[4] Mohamed Bendraouche, Mourad Boudhar, and Ammar Oulamara. Scheduling : Agreement graph vs resource constraints. *European Journal of Operational Research*, 240(2) :355–360, 2015.

[5] Jacek Blazewicz, Jan Karel Lenstra, and AHG Rinnooy Kan. Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1) :11–24, 1983.

[6] Hans L Bodlaender and Klaus Jansen. Restrictions of graph partition problems. part i. *Theoretical Computer Science*, 148(1) :93–109, 1995.

[7] Yinhui Cai, Guangting Chen, Yong Chen, Randy Goebel, Guohui Lin, Longcheng Liu, and An Zhang. Approximation algorithms for two-machine flow-shop scheduling with a conflict graph. In *International Computing and Combinatorics Conference*, pages 205–217. Springer, 2018.

[8] Vladimír Černỳ. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1) :41–51, 1985.

[9] GH Chen, MT Kuo, and JP Sheu. An optimal time algorithm for finding a maximum weight independent set in a tree. *BIT Numerical Mathematics*, 28(2) :353–356, 1988.

[10] Dominique De Werra. Restricted coloring models for timetabling. *Discrete Mathematics*, 165 :161–170, 1997.

[11] Mohamed I Dessouky, Ben J Lageweg, Jan Karel Lenstra, and Steef L van de Velde. Scheduling identical jobs on uniform parallel machines. *Statistica Neerlandica*, 44(3) :115–123, 1990.

[12] Guy Even, Magnús M Halldórsson, Lotem Kaplan, and Dana Ron. Scheduling with conflicts : online and offline algorithms. *Journal of Scheduling*, 12(2) :199–224, 2009.

[13] Frédéric Gardi. Mutual exclusion scheduling with interval graphs or related classes, part i. *Discrete Applied Mathematics*, 157(1) :19–35, 2009.

[14] Michael R Garey and Ronald L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2) :187–200, 1975.

[15] Michael R Garey and David S Johnson. *Computers and intractability : A Guide to the Theory of NP-Completeness*. New York : Freeman, 1979.

[16] Magnús M Halldórsson, Guy Kortsarz, Andrzej Proskurowski, Ravit Salman, Hadas Shachnai, and Jan Arne Telle. Multi-coloring trees. In *International Computing and Combinatorics Conference*, pages 271–280. Springer, 1999.

[17] John Hopcroft and Robert Tarjan. Algorithm 447 : efficient algorithms for graph manipulation. *Communications of the ACM*, 16(6) :372–378, 1973.

[18] Te C Hu. Parallel sequencing and assembly line problems. *Operations research*, 9(6) :841–848, 1961.

[19] Klaus Jansen. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180(2) :71–81, 2003.

[20] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[21] Jan Karel Lenstra, AHG Rinnooy Kan, and Peter Brucker. Complexity of machine scheduling problems. In *Annals of discrete mathematics*, volume 1, pages 343–362. Elsevier, 1977.

[22] Amine MOHABEDDINE and Mourad BOUDHAR. Problème d'ordonnacement avec graphe de concordance sur machines uniformes : Complexité et résolution exacte. *Journées Scientifiques du Laboratoire RECITS, JSL'2017*, 4 2017.

[23] Amine Mohabeddine and Mourad Boudhar. New results in two identical machines scheduling with agreement graphs. *Theoretical Computer Science*, 779 :37–46, 2019.

[24] Amine MOHABEDDINE, Mourad BOUDHAR, and Ammar OULAMARA. Ordonnancement sur deux machines avec contraintes de concordance : machines identiques et uniformes. *6th Operational Research Practice in Africa Conference (ORPA'2015)*, 2015.

[25] Michael H Rothkopf. Scheduling independent tasks on parallel processors. *Management Science*, 12(5) :437–447, 1966.

[26] Nour El Houda Tellache and Mourad Boudhar. The two-machine flow shop problem with conflict graphs. *IFAC-PapersOnLine*, 49(12) :1026–1031, 2016.

[27] Nour El Houda Tellache and Mourad Boudhar. Open shop scheduling problems with conflict graphs. *Discrete Applied Mathematics*, 227 :103–120, 2017.

[28] Nour El Houda Tellache and Mourad Boudhar. Two-machine flow shop problem with unit-time operations and conflict graph. *International Journal of Production Research*, 55(6) :1664–1679, 2017.

[29] Nour El Houda Tellache and Mourad Boudhar. Flow shop scheduling problem with conflict graphs. *Annals of Operations Research*, 261(1-2) :339–363, 2018.