Mémoire présenté pour l'obtention
de l'habilitation à diriger des recherches

De l'approximation standard vers l'approximation multicritère et multijoueur

Eric Angel

soutenue le 04/12/2006 devant le jury

Philippe Chrétienne	Rapporteur	Université Paris VI
Pierre Fraigniaud	Rapporteur	Université Paris XI
Claire Kenyon	Rapporteur	Brown University
Evripidis Bampis		Université d'Evry Val d'Essonne
Dominique de Werra		École Polytechnique Fédérale de Lausanne
Vassilis Zissimopoulos		Université d'Athène

Université d'Evry Val d'Essonne
IBISC, CNRS FRE 2873
523 Place des Terrasses, 91000 Evry

Prologue

L'optimisation combinatoire constitue un sujet de recherche très vaste, de par le type de problèmes et le contexte dans lequel ils peuvent être abordés, et de par le type de résultats recherchés. Ce mémoire qui décrit les travaux de recherche que j'ai menés dans l'équipe OPAL au sein du LaMI, depuis septembre 1999, en est une illustration.

Nous avons considéré des problèmes d'optimisation issus de domaines variés : théorie des graphes ; problèmes d'ordonnancement ; problèmes issus d'applications "réelles" ; etc. Les contextes dans lesquels ils ont été étudiés le sont tout autant : optimisation dans un cadre monocritère, multicritère, ou multijoueur. Le contexte a déterminé en partie quelle était l'approche utilisée : obtention d'algorithmes efficaces, avec ou sans garantie de performance, ou de mécanismes possédant certaines propriétés. Enfin les méthodes que nous avons utilisées sont multiples : utilisation d'algorithmes déterministes ou probabilistes ; programmation linéaire ; algorithmes de types gloutons ou de liste ; programmation dynamique ; recherche locale et méthodes hybrides ; réductions entres problèmes.

La partie I de ce mémoire constitue une introduction aux travaux présentés par la suite. Dans le chapitre 1 nous présentons les différents contextes et approches que nous avons utilisés par la suite. Le chapitre 2 rassemble les problèmes d'optimisation combinatoire que nous avons considérés.

La partie II traite des algorithmes approchés sans garantie de performance. Nous présentons dans le chapitre 3 des travaux relatifs aux voisinages de grande taille pour les algorithmes à base de recherche locale.

La partie III traite de l'approche avec garantie de performance. Nous présentons dans le chapitre 4 des résultats que nous avons obtenus dans le cadre de l'optimisation monocritère. La suite de cette troisième partie est consacrée à l'optimisation multiobjectif. Nous présentons dans le chapitre 5 les résultats que nous avons obtenus avec l'approche point idéal. Les chapitres 6 et 7 sont quant à eux consacrés à l'approche courbe de Pareto.

La partie IV est consacrée à l'approche théorie des jeux. Le chapitre 8 traite de la recherche de solutions stables, i.e. des équilibres de Nash, qui soient de bonne qualité, pour un problème d'ordonnancement. Dans le chapitre 9 on s'intéresse à la problématique de construire des mécanismes avec véracité garantie. Dans le chapitre 10 on considère la recherche de solutions équitables vis à vis de chacun des agents.

Enfin nous dressons dans le chapitre 11 quelques axes de recherche susceptibles de constituer un prolongement de ces travaux.

Table des matières

Première partie

Introduction

Chapitre 1

Contextes et approches

Sommaire

Un même problème peut souvent être vu ou défini de différentes manières, selon le contexte dans lequel on se place et les hypothèses que l'on fait : L'instance est-elle définie dès le départ ? Les données sont-elles connues avec certitude ? Évoluent-elles au cours du temps ? Cherche-t'on à optimiser un seul critère ? Peut-on agir de manière autonome, ou faut-il tenir compte de la présence d'autres participants (agents) ? Ceux-ci peuvent-ils coopérer entre eux ? Peuvent-ils mentir ?

De plus différentes approches de résolution sont possibles, et ceci de manière parfois orthogonale au contexte choisi : Cherche-t'on un algorithme efficace et susceptible de donner de bons résultats en pratique ? Cherche-t'on un algorithme pour lequel on soit capable de fournir des garanties de performance théoriques ? Cherche-t'on des algorithmes susceptibles de fournir des solutions qui ne soient pas remises en cause par les agents ?

Nous présentons dans ce chapitre les différents contextes ou points de vue que nous avons considéré lors de nos travaux, ainsi que les différentes approches suivies.

1.1 Optimisation monocritère

Classiquement dans un problème d'optimisation combinatoire, on dispose d'un ensemble fini de solutions réalisables $\mathcal{S}$, et d'une fonction objectif $f : \mathcal{S} \to \mathbb{Q}$ que l'on cherche à maximiser ou à minimiser.

Depuis les travaux initiés par Karp dans les années 70 [117], la majorité des problèmes en optimisation combinatoire ont été montrés **NP**-difficiles [86]. Il est donc vraisemblablement impossible, de trouver des algorithmes polynomiaux capables de les résoudre de manière exacte. Cependant, en relâchant la contrainte d'optimalité, c'est-à-dire en cherchant à produire des solutions approchées, il est souvent possible d'obtenir de manière efficace des solutions "presque" optimales, souvent satisfaisantes en pratique.

Deux voies de recherche sont alors possibles. On peut chercher des algorithmes qui vont donner, en pratique, de bons résultats expérimentaux. Les métaheuristiques constituent alors une méthode

Étape 1 :	Soit s une solution réalisable quelconque
Étape 2 :	**Tant qu'** il existe $s' \in \mathcal{N}(s)$ préférée à s **faire**
	$s \leftarrow s'$
	Fin Tant que
Étape 3 :	Retourner s

TAB. 1.1 – Algorithme standard de recherche locale. Une solution courante s est remplacée par une solution s' de son voisinage seulement si s' est préférée à s. La procédure termine lorsqu'un *optimum local* est atteint.

de choix. Celles-ci se basent souvent sur la recherche locale. Un inconvénient de ces méthodes est qu'il n'est pas possible de déterminer de manière théorique la qualité des solutions qu'elles permettent d'obtenir. L'autre voie de recherche concerne donc l'obtention d'algorithmes avec garantie de performance.

1.1.1 Métaheuristiques et recherche locale

Rappelons que les *métaheuristiques* sont des méthodes générales, qui permettent de concevoir des algorithmes fonctionnant très biens en pratique pour une très grande variété de problèmes [73, 104]. On peut citer entre autres les *algorithmes génétiques*, le *recuit simulé*, la *recherche tabou*, les *colonies de fourmis*.

Ces méthodes s'appuient très souvent sur la *recherche locale*, elle même basée sur la notion de *voisinage* : noté $\mathcal{N}(s) \subset \mathcal{S}$, le voisinage d'une solution s est un ensemble de solutions réalisables, qui en pratique sont obtenues en appliquant une transformation sur s. Dans un algorithme de recherche locale, à chaque étape, on cherche parmi les voisins de la solution courante, une solution de meilleure qualité (vis à vis de la fonction objectif f). Si une telle solution existe, alors elle se substitue à la solution courante. Le processus se poursuit jusqu'à ce qu'on ne puisse plus trouver de meilleure solution. La solution courante est alors un *optimum local*. Un tel algorithme est représenté dans le tableau 1.1.

Dans cette thématique on cherche à obtenir des algorithmes qui soient les plus performants possibles, aussi bien en terme de qualité des solutions obtenues, qu'en terme de temps de calcul pour les obtenir. Pour ce faire, il est intéressant d'utiliser des voisinages de grandes tailles, si possible de taille exponentielle par rapport à la taille de l'instance. Il faut alors s'assurer que ceux-ci peuvent être exploré de manière efficace, de manière à ce que l'algorithme de recherche locale demeure polynomial à chaque itération. Le chapitre 3 est consacré à cette problématique.

Une autre voie de recherche que nous avons abordée [25, 26] consiste à utiliser des méthodes hybrides, combinant par exemple des méthodes exactes, telle que la programmation par contraintes [30], et des métaheuristiques.

1.1.2 Algorithmes avec garantie de performance

La recherche d'algorithmes d'approximation avec garantie de performance est un domaine majeur de l'informatique [101, 170].

Définition 1 *Soit OPT le coût d'une solution optimale. Un algorithme est dit ρ-approché pour un problème de minimisation (respectivement maximisation) s'il produit toujours une solution dont le coût ne dépasse pas ρOPT, avec $\rho \geq 1$ (respectivement dépasse ρOPT, avec $\rho \leq 1$). Une telle solution est dite ρ-approchée.*

Définition 2 *Une famille d'algorithmes $(1+\varepsilon)$-approchés pour tout $\varepsilon > 0$, est appelée un schéma d'approximation polynomial si le temps d'exécution de chaque algorithme est polynomial en fonction de la taille de l'instance. Si le temps d'exécution est polynomial également en fonction de $1/\varepsilon$, alors on parle de schéma d'approximation totalement polynomial.*

Les algorithmes probabilistes permettent quelquefois d'obtenir de meilleurs résultats d'approximation en moyenne que ce qu'il serait possible d'obtenir si on se limitait à des algorithmes déterministes.

Définition 3 *Un algorithme probabiliste, pour un problème monocritère de minimisation (respectivement maximisation), est dit ρ-approché, si l'espérance du coût X de la solution qu'il retourne, noté $E[X]$, est au plus (respectivement au moins) ρ fois le coût d'une solution optimale : $E[X] \leq \rho OPT$ (respectivement $E[X] \geq \rho OPT$).*

Comme nous le verrons par la suite, il est parfois possible de dérandomiser un algorithme probabiliste pour obtenir un algorithme déterministe ayant la même garantie de performance.

Dans cette thématique, on recherche bien évidemment des algorithmes dont le rapport d'approximation est le plus proche possible de 1. Parallèlement à ce travail, il est intéressant d'obtenir également des résultats d'inapproximabilité. Le chapitre 4 expose les résultats que nous avons obtenus pour plusieurs problèmes d'optimisation combinatoire.

1.2 Optimisation multicritère

Dans un problème d'optimisation multicritère, on dispose d'un ensemble fini de solutions réalisables $\mathcal{S}$, et de $k \geq 2$ fonctions objectifs à optimiser : $f_1, \ldots, f_k$, avec $f_i : \mathcal{S} \to \mathbb{Q}$ pour $i = 1, \ldots, k$. Pour simplifier les notations, nous faisons dans cette section l'hypothèse que toutes les fonctions objectifs sont à minimiser, sans que cela remette en cause la généralité des notions exposées. Les notions de critère et de fonction objectif se confondent. Ainsi, on parle indifféremment d'optimisation multicritère ou d'optimisation multiobjectif.

Une des difficultés de ces problèmes provient du fait que les fonctions objectifs sont souvent en conflit : une solution optimale pour un critère ne l'est pas forcément pour un autre, par conséquence l'existence d'une solution réalisable optimisant simultanément chacune des fonctions objectifs n'est souvent pas garantie.

Les problèmes d'optimisation combinatoires multicritères qui généralisent un problème monocritère **NP**-difficile, sont de fait **NP**-difficiles. Par contre, certains problèmes multicritères sont **NP**-difficiles malgré le fait qu'ils généralisent des problèmes monocritères polynomiaux. C'est le cas par exemple du problème de l'arbre couvrant de poids minimum bicritère [52], du problème de la plus courte chaîne bicritère, ou du problème du couplage maximal de poids minimum bicritère [159].

Les approches qui ont été suivies pour résoudre les problèmes d'optimisation multicritère peuvent être classées en 4 grandes catégories. La première, et sans doute la plus évidente, consiste à considérer une combinaison linéaire des différents critères sous la forme d'une fonction objectif unique [66]. La deuxième approche consiste à chercher une solution satisfaisant simultanément, au mieux, tous les critères [164]. Une troisième approche, qualifiée d'approche budget par la suite, consiste à optimiser un critère alors que les autres sont contraints de ne pas dépasser des bornes fixées à l'avance [112, 162, 168]. Dans la dernière approche, contrairement à toutes les autres, on ne recherche pas une, mais un ensemble de solutions réalisant des compromis sur les différents critères. On parle alors de *courbe de Pareto* [16, 56]. Nous passons brièvement en revue les trois dernières approches.

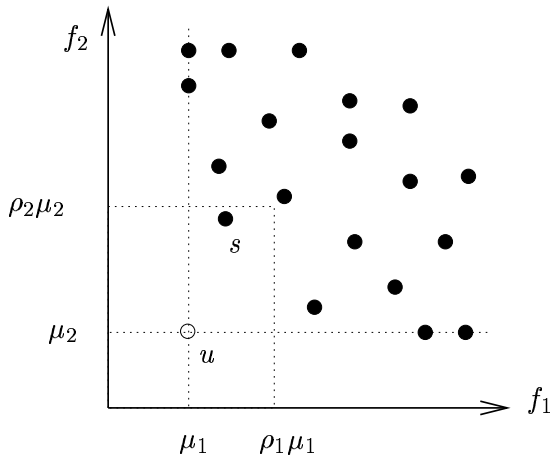


FIG. 1.1 – Le point u représente le point idéal. Cette valeur est rarement atteinte par une solution réalisable. La solution s est (ρ_1, ρ_2)-approchée de u.

1.2.1 Approche point idéal

Comme nous l'avons mentionné plus haut, le fait de considérer plusieurs fonctions objectifs en conflit écarte souvent l'existence d'une solution réalisable satisfaisant au mieux tous les critères. Cependant, la valeur d'une telle solution, appelée *point idéal* peut toujours servir de référence pour apprécier la qualité d'une solution réalisable.

Définition 4 *Le* point idéal, *pour un problème d'optimisation à k critères, est un vecteur $\vec{\mu} = (\mu_1, \ldots, \mu_k)$ tel que :*

$$\mu_i = \min_{s \in \mathcal{S}} \{f_i(s)\}, \text{ pour } i = 1, \ldots, k.$$

L'approche simultanée, ou *approche point idéal* consiste à produire une solution réalisable dont l'image soit la plus proche possible de $\vec{\mu}$ sur chacun des critères.

Définition 5 *Une solution $s \in \mathcal{S}$ est dite $(\rho_1, \ldots, \rho_k)$-simultanée-approchée pour un problème de minimisation multicritère si $f_i(s) \leq \rho_i \mu_i$ pour $i = 1, \ldots, k$.*

La figure 1.1 illustre cette définition.

Définition 6 *Un algorithme est dit $(\rho_1, \ldots, \rho_k)$-simultanée-approché pour un problème Π à k critères s'il produit une solution réalisable $(\rho_1, \ldots, \rho_k)$-simultanée-approchée pour toute instance de Π.*

Il est important de noter qu'il n'existe pas toujours de solution $\vec{\rho}$-simultanée-approchée pour certaines valeurs de $\vec{\rho}$. Par conséquent, en amont de ce travail d'approximation, il est intéressant de considérer des résultats d'inexistence.

Nous présentons dans le chapitre 5 les résultats que nous avons obtenus avec cette approche, pour un problème d'ordonnancement bicritère ainsi que pour le problème de la coupe maximale bicritère.

1.2.2 Approche budget

L'approche budget consiste à optimiser un critère alors que les autres sont contraints de ne pas dépasser des bornes fixées. Pour le cas d'un problème général à k critères, on fixe des bornes $B_2, \ldots, B_k$ pour les critères 2 jusqu'à k, et le problème qu'on cherche à résoudre s'écrit ainsi :

$$\begin{array}{rl} & \min f_1(s) \\ \text{sous les contraintes :} & f_2(s) \leq B_2 \\ & \vdots \\ & f_k(s) \leq B_k \\ & s \in \mathcal{S} \end{array}$$

Dans le cadre de l'approximation avec garantie de performance, on peut chercher à retourner une solution s vérifiant toutes les contraintes et qui soit ρ-approchée sur f_1 :

$$f_1(s) \leq \rho\Big(\min_{s' \in \mathcal{S}}\{f_1(s') \mid f_i(s') \leq B_i \text{ pour } i = 2, \ldots, k\}\Big).$$

Ainsi, seul un scalaire ρ est nécessaire pour qualifier la qualité de l'approximation. Cependant, une définition plus générale de l'approximation peut être donnée en supposant que les contraintes ne sont pas strictes et que les bornes peuvent être dépassées.

Définition 7 *Pour $(B_2, \ldots, B_k)$ fixé et $B_1 = min_{s \in \mathcal{S}}\{f_1(s) \mid f_i(s) \leq B_i$ pour $i = 2, \ldots, k\}$, une solution $s \in \mathcal{S}$ est dite $(\rho_1, \ldots, \rho_k)$-budget-approchée si $f_i(s) \leq \rho_i B_i$ pour $i = 1, \ldots, k$.*

Ce qui était une solution ρ-approchée est, pour la nouvelle définition, une solution $(\rho, 1, \ldots, 1)$-budget-approchée.

Définition 8 *Un algorithme est dit $(\rho_1, \ldots, \rho_k)$-budget-approché pour un problème Π à k critères s'il produit une solution réalisable $(\rho_1, \ldots, \rho_k)$-budget-approchée pour toute instance de Π et tout vecteur budget $(B_2, \ldots, B_k)$ réalisable.*

Cette approche est par exemple utilisée dans le chapitre 6 dans la section 6.2.

1.2.3 Approche courbe de Pareto

La courbe de Pareto est basée sur la notion de *dominance*. De manière informelle, une solution s domine une autre solution s' si elle est au moins "aussi bonne" sur tous les critères et strictement meilleure sur au moins un critère.

Définition 9 *Une solution s domine une solution s', noté $s < s'$, si :*
- *$f_i(s) \leq f_i(s')$ pour $i = 1, \ldots, k$, et*
- *il existe au moins un critère i^* tel que $f_{i^*}(s) < f_{i^*}(s')$.*

La dominance définit un ordre qui est seulement partiel car il peut exister deux solutions s et s', dites *incomparables*, telles que s ne domine pas s' et s' ne domine pas s.

Définition 10 *Une solution $s \in \mathcal{S}$ est dite* Pareto optimale, *s'il n'existe pas de solution $s' \in \mathcal{S}$ qui la domine. Pour une instance d'un problème multicritère,* l'ensemble de Pareto, *noté $\mathcal{S}_{Par} \subseteq \mathcal{S}$, est l'ensemble des solutions Pareto optimales. La* courbe de Pareto, *notée P, est l'ensemble des images des solutions Pareto optimales : $P = \{\vec{f}(s) \mid s \in \mathcal{S}_{Par}\}$.*

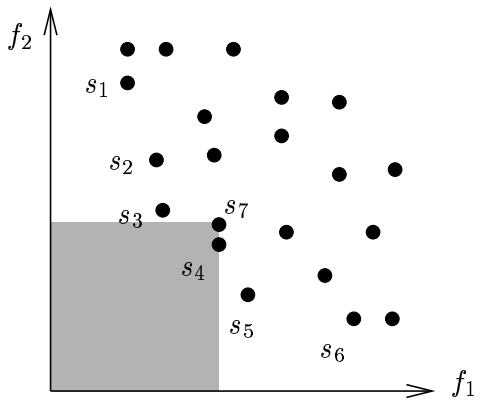


FIG. 1.2 – Les solutions s_1 à s_6 sont Pareto optimales : $\mathcal{S}_{\text{Par}} = \{s_1, s_2, s_3, s_4, s_5, s_6\}$ et $P = \{\vec{f}(s_1), \vec{f}(s_2), \vec{f}(s_3), \vec{f}(s_4), \vec{f}(s_5), \vec{f}(s_6)\}$.

La figure 1.2 illustre ces notions pour un problème bicritère.

Déterminer si une solution fait partie de la courbe de Pareto est souvent un problème **NP**-complet. C'est le cas par exemple pour les problèmes de couplage, arbre couvrant de poids minimum, et plus court chemin, dans leur version bicritère [76]. De plus, la courbe de Pareto peut contenir un nombre exponentiel de points. C'est le cas par exemple pour le problème de la plus courte chaîne bicritère entre deux sommets [98]. Tout algorithme visant à produire explicitement l'ensemble des solutions Pareto optimales est alors nécessairement exponentiel.

Définition 11 *Un problème multicritère est dit* indocile, *si le cardinal de la courbe de Pareto peut être exponentiel par rapport à la taille d'une instance.*

Face à ces difficultés, on peut chercher à déterminer une courbe de Pareto approchée. En effet, Papadimitriou et Yannakakis [135] ont montré que pour tout problème multicritère il existait toujours une courbe de Pareto $\vec{\varepsilon}$-approchée (voir la définition ci-dessous) comportant un nombre de solutions polynomial en la taille de l'instance et $1/\varepsilon$ (le nombre de critères étant une constante).

De nombreuses métaheuristiques ont été adaptées afin de pouvoir retourner des courbes de Pareto approchées (sans garantie de performance). On peut citer ainsi le recuit simulé [70, 160], la recherche tabou [85, 99], les algorithmes génétiques [61, 174, 176] et les colonies de fourmis [84]. Nous exposons dans le chapitre 3, dans les sections 3.4.1 et 3.4, les travaux que nous avons menés concernant l'utilisation d'un voisinage de taille exponentielle pour le problème bicritère du voyageur de commerce.

On peut également chercher à produire des courbes de Pareto approchées avec garantie de performance.

Définition 12 *Étant donné un vecteur $\vec{\varepsilon} = (\varepsilon_1, \ldots, \varepsilon_k)$, une solution $s \in \mathcal{S}$ $\vec{\varepsilon}$-domine une solution s' si $f_i(s) \leq (1 + \varepsilon_i)\, f_i(s')$ pour $i = 1, \ldots, k$.*

Définition 13 *Pour un problème à k critères, un ensemble de Pareto $\vec{\varepsilon}$-approché, noté $\mathcal{S}_{\vec{\varepsilon}}$, est un ensemble de solutions réalisables tel que pour toute solution $s' \in \mathcal{S}$, il existe une solution $s \in \mathcal{S}_{\vec{\varepsilon}}$ qui $\vec{\varepsilon}$-domine s'. Une courbe de Pareto $\vec{\varepsilon}$-approchée, notée $P_{\vec{\varepsilon}}$, est l'ensemble des images des solutions d'un ensemble de Pareto $\vec{\varepsilon}$-approché.*

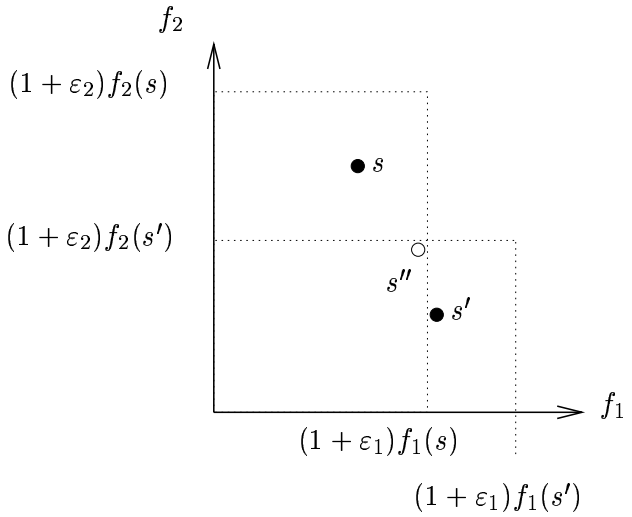


FIG. 1.3 – La solution s'' $\vec{\varepsilon}$-domine les solutions s et s'.

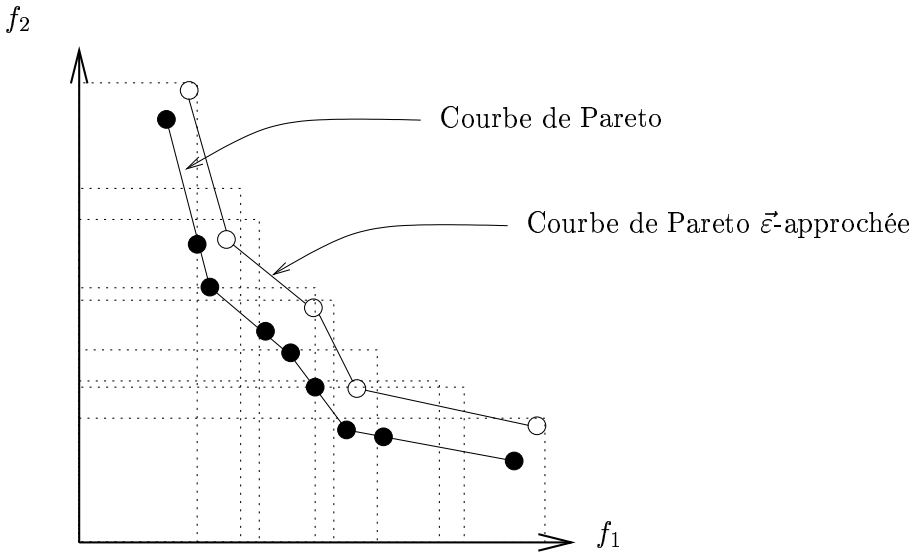


FIG. 1.4 – Illustration d'une courbe de Pareto $\vec{\varepsilon}$-approchée.

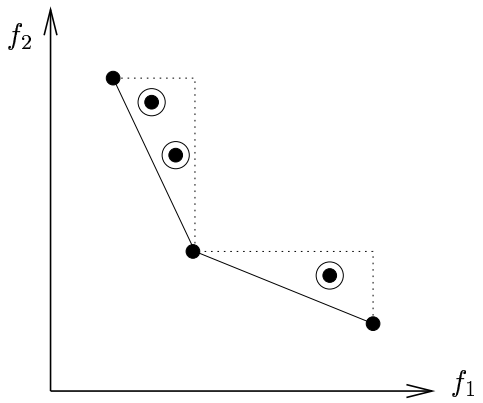


FIG. 1.5 – Les solutions représentées sont toutes Pareto optimales mais seulement trois sont supportées. Les solutions non supportées, entourées sur la figure, appartiennent aux triangles de convexité.

Les figures 1.3 et 1.4 illustrent les définitions précédentes.

Il est possible de distinguer le rapport d'approximation sur chacun des critères. En particulier, dans le cas bicritère, nous considérerons par la suite des courbes de Pareto $(\varepsilon, 0)$-approchées.

Nous exposons dans les chapitres 7 et 6 les résultats que nous avons obtenu avec cette approche.

1.2.4 Liens entre ces approches

L'approche courbe de Pareto englobe en quelque sorte chacune des autres approches mentionnées plus haut. En effet, si on connaît une courbe de Pareto exacte, alors on peut déterminer une solution optimale vis à vis d'une pondération des critères quelconque, on peut déterminer une solution approchant le point idéal, et on peut déterminer une solution optimale vis à vis de l'approche budget.

Comme nous le verrons plus loin, au chapitre 6, l'approche budget peut permettre de générer une courbe de Pareto approchée.

La pondération des fonctions objectifs permet aussi de générer des solutions Pareto optimales à l'aide d'algorithmes monocritères exacts.

Théorème 1 *[76] Toute solution réalisable s qui minimise $\tilde{f} = \sum_{i=1}^{k} \lambda_i f_i$, avec $\lambda_i \in \mathbb{R}^{+*}$ pour $i = 1, \ldots, k$, est Pareto optimale.*

On peut utiliser plusieurs fois la méthode de pondération des critères avec des pondérations différentes afin de générer un ensemble de solutions Pareto optimales. Le très grand nombre de ces pondérations interdit cependant toute approche exhaustive et se pose alors la difficulté de faire un choix judicieux d'un sous-ensemble de pondérations.

De plus, du fait de la *non convexité* de la courbe de Pareto, cette technique ne permet d'obtenir qu'une certaine catégorie de solutions Pareto optimales : les solutions *supportées* (voir la figure 1.5), c'est-à-dire celles qui appartiennent à l'enveloppe convexe de la courbe de Pareto.

1.2.5 Comparaison de courbes de Pareto approchées

Il n'existe pas de mesure unique, totalement satisfaisante, permettant de comparer deux courbes de Pareto approchées (voir [97, 114] pour une discussion sur le sujet). Par la suite nous avons utilisé

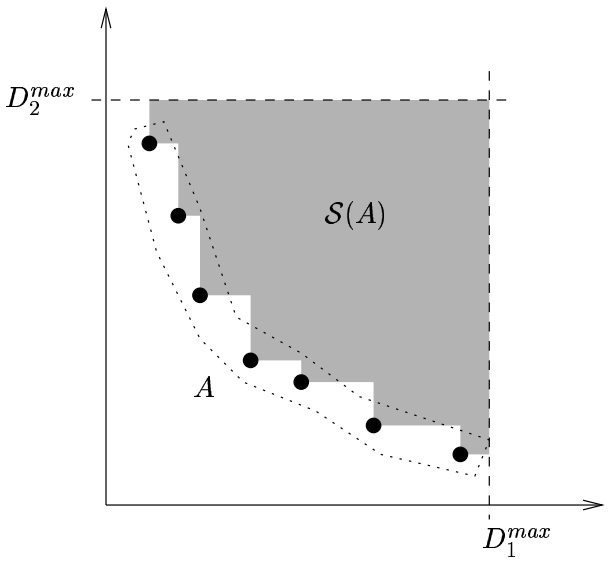


FIG. 1.6 – La surface $\mathcal{S}(A)$ est représentée en gris.

trois méthodes : la *différence de couverture de deux ensembles* introduite par Zitzler [174], *l'espérance sur les fonctions d'utilité de Tchebycheff pondérées* proposée par Hansen et Jaszkiewicz [97] et la mesure $\mathcal{C}$ utilisée par exemple dans [175]. Ces trois mesures sont brièvement décrites pour un problème bicritère où chaque critère est à minimiser.

Différence de couverture de deux ensembles
Soient D_1^{max} (respectivement D_2^{max}) la valeur maximale que peut prendre une solution sur le premier (respectivement second) critère pour une instance donnée du problème. Étant donné un ensemble de solutions incomparables A constituant une courbe de Pareto, on peut déterminer une surface $\mathcal{S}(A)$ comme montré en figure 1.6. La quantité $\mathcal{S}(A)$ mesure la portion de l'espace objectif dominée par l'ensemble de points A.

Étant donnés A et B, deux ensembles de solutions incomparables, constituant deux courbes de Pareto, la fonction $\mathcal{D}$ (différence de couverture de deux ensembles) est définie par $\mathcal{D}(A,B) = \mathcal{S}(A+B) - \mathcal{S}(B)$ et $\mathcal{D}(B,A) = \mathcal{S}(A+B) - \mathcal{S}(A)$ (voir la figure 1.7). La valeur $\mathcal{D}(A,B)$ mesure la portion de l'espace objectif dominée par A mais pas par B. De manière symétrique $\mathcal{D}(B,A)$ mesure la portion de l'espace objectif dominée par B mais pas par A. Intuitivement, si $\mathcal{D}(A,B) \geq \mathcal{D}(B,A)$ alors A est meilleur que B.

Espérance sur les fonctions d'utilité de Tchebycheff pondérées
Soient σ une solution et σ^* la solution idéale (qui n'existe pas forcément) telle que $\vec{D}_1(\sigma^*) = D_1^{min}$ et $\vec{D}_2(\sigma^*) = D_2^{min}$ (D_1^{min} et D_2^{min} sont respectivement les valeurs minimales que peut atteindre une solution pour les deux critères). La fonction d'utilité de Tchebycheff pondérée est définie par $u_\infty(\sigma,\sigma^*,\Lambda) = -\max\{\Lambda_1(\vec{D}_1(\sigma) - \vec{D}_1(\sigma^*)), \Lambda_2(\vec{D}_2(\sigma) - \vec{D}_2(\sigma^*))\}$ où $\Lambda = (\Lambda_1, \Lambda_2)$ est un vecteur de poids. Étant donné un ensemble de solutions non dominées A, $u_\infty^*(A,\Lambda) = \max_{\sigma \in A} u_\infty(\sigma,\sigma^*,\Lambda)$ est la meilleure utilité atteinte par la fonction $u_\infty(\sigma,\sigma^*,\Lambda)$ pour A. La qualité de A peut être évaluée par l'espérance sur la valeur de la fonction d'utilité pondérée de Tchebycheff pour l'ensemble des vecteurs de poids normalisés :

$$E[u_\infty^*(A,\Lambda)] = \int_{\Lambda \in \Psi} u_\infty^*(A,\Lambda) p(\Lambda) d\Lambda,$$

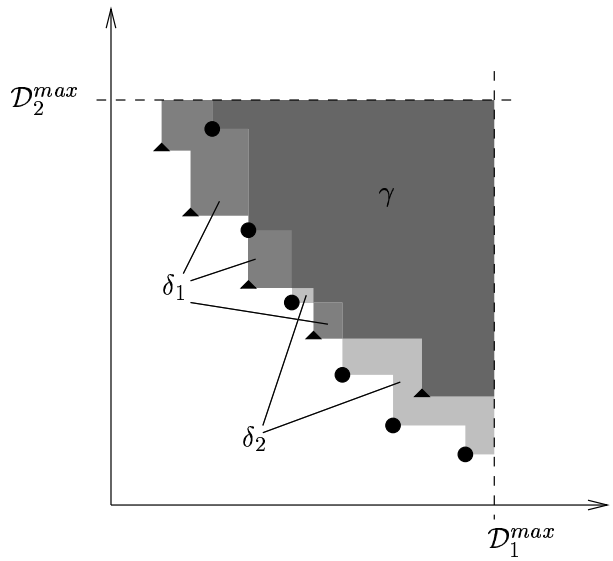


FIG. 1.7 – Soient A et B deux ensembles de solutions incomparables représentés respectivement par des triangles et des cercles. On a $\mathcal{D}(A,B) = \delta_1$ et $\mathcal{D}(B,A) = \delta_2$. La portion γ est dominée par les deux ensembles.

où $\Psi = \{\Lambda \in \mathbb{R}^2 \mid \Lambda_1 + \Lambda_2 = 1, \Lambda_1, \Lambda_2 \geq 0\}$ désigne l'ensemble de tous les vecteurs de poids normalisés et $p(\Lambda)$ est une fonction de probabilité d'intensification. Par la suite, $p(\Lambda)$ est supposée uniforme sur Ψ. Ainsi, si $E[u^*_\infty(A,\Lambda)] > E[u^*_\infty(B,\Lambda)]$ alors on conclut que A est meilleur que B pour cette mesure.

La mesure $\mathcal{C}$
Une troisième façon d'évaluer la qualité entre deux ensembles de solutions A et B est de considérer le rapport suivant :

$$\mathcal{C}(A,B) = \frac{|\{\sigma \in A \mid \nexists \sigma' \in B \text{ t.q. } \sigma' < \sigma\}|}{|A|}.$$

C'est le rapport entre le nombre de solutions appartenant à A qui ne sont pas dominées par une solution de B et le cardinal de A. Comme en général l'égalité $\mathcal{C}(A,B) + \mathcal{C}(B,A) = 1$ n'est pas respectée, il est nécessaire de calculer à la fois $\mathcal{C}(A,B)$ et $\mathcal{C}(B,A)$. Si on a $\mathcal{C}(A,B) > \mathcal{C}(B,A)$, on en conclut que A est meilleur que B pour ce critère.

1.3 Théorie des jeux

Jusqu'à présent on s'est placé dans le contexte d'un preneur de décision isolé, qui n'est pas en interaction avec d'autres agents, et qui peut calculer avec certitude les effets des décisions qu'il prend. Il est nécessaire de changer de contexte lorsqu'on s'attaque à des problèmes qui se posent dans un environnement distribué, dans lequel plusieurs agents interagissent.

Par exemple, de nos jours, la compréhension d'internet est devenue l'un des enjeux majeurs en informatique théorique. Or l'internet peut être vu comme un système complexe distribué dans lequel un grand nombre d'entités agissent de manière indépendante, chacune voulant maximiser son profit. Des protocoles organisent ce réseau, dans le but d'arriver à une situation qui soit la plus bénéfique pour tous. Plus précisément, on fait donc l'hypothèse que chaque agent possède une

fonction objectif chargée de mesurer son profit, tandis qu'il existe une fonction objectif globale chargée de mesurer la bonne utilisation des ressources du système. Cette manière de voir les choses rend possible l'utilisation de nombreuses notions de la théorie des jeux non coopératifs.

Vers quel état un tel système va-t'il converger ? Est-il possible de concevoir des protocoles de manière à obtenir des solutions qui soient stables, c'est-à-dire des solutions qui ne soient pas remises en cause par les agents, et qui soient en même temps satisfaisantes d'un point de vue global quant à l'utilisation des ressources du système ? Les agents peuvent-ils manipuler le protocole à leur avantage en lui fournissant de fausses informations ? Dans d'autres applications, les agents peuvent choisir de se concerter afin d'arriver à un but commun. On entre alors dans le domaine des jeux coopératifs. Quelle est alors une solution qui soit la plus équitable pour chacun des participants ?

1.3.1 Stabilité

Un jeu non coopératif est constitué d'un ensemble d'agents N, chacun ayant un ensemble fini de stratégies (actions). Pour un agent $i \in N$, on note A_i l'ensemble des stratégies à sa disposition. Chaque agent $i \in N$ doit donc choisir une stratégie $a_i \in A_i$. On note $a = (a_i)_{i \in N}$, ou plus simplement $a = (a_i)$ lorsque aucune confusion n'est possible, l'ensemble des stratégies choisies par les agents. Pour un agent $i \in N$, on note a_{-i} la collection $(a_j)_{j \in N \setminus i}$, c'est-à-dire l'ensemble des stratégies choisies par tous les agents excepté l'agent i. Étant donné $a_{-i} = (a_j)_{j \in N \setminus i}$ et une stratégie $a_i \in A_i$ on note (a_{-i}, a_i) la collection $(a_i)_{i \in N}$. Chaque agent $i \in N$, possède une fonction objectif p_i qu'il cherche à maximiser. Étant donné $a = (a_i)_{i \in N}$, la quantité $p_i(a)$ indique le profit de l'agent i dans cette situation.

Définition 14 *Un* équilibre de Nash *(pur)* $a^* = (a_i^*)_{i \in N}$ *est une situation dans laquelle, pour chaque agent* $i \in N$, $p_i(a_{-i}^*, a_i) \leq p_i(a_{-i}^*, a_i^*)$, *pour chaque stratégie* $a_i \in A_i$ *[129].*

Un équilibre de Nash est donc une situation du système dans laquelle aucun agent n'a intérêt à changer de stratégie, compte tenu des stratégies des autres agents. Un tel équilibre n'existe pas toujours.

Étant donné que les agents agissent indépendamment afin d'optimiser leur fonction objectif individuelle, la valeur de la fonction objectif globale peut ne pas être aussi bonne que celle d'une solution choisie par une autorité centrale. Il est donc important de mesurer et d'être capable de borner la perte en qualité des solutions. Autrement dit, vis à vis de la fonction objectif globale, quelle est la qualité d'un équilibre de Nash par rapport à la qualité d'un optimum construit de manière centralisée ? C'est dans ce but que Koutsoupias et Papadimitriou [119] ont introduit le *prix de l'anarchie*.

Définition 15 *Le* prix de l'anarchie *pour une instance, est égal au rapport entre la valeur de la fonction objectif globale dans le* pire *équilibre de Nash pour cette instance, et sa valeur dans une solution optimale (qui n'est pas forcément un équilibre de Nash) pour cette instance. Le prix de l'anarchie pour un problème de minimisation est égal au maximum du prix de l'anarchie sur chacune de ses instances.*

Est-il possible de faire baisser ce prix de l'anarchie ? Christodoulou et al. [57] ont étudié des *mécanismes de coordination*, dont le but est de minimiser le prix de l'anarchie. Ces mécanismes, qui ne doivent pas altérer la nature distribuée du système, forcent les agents à "coopérer" spontanément, s'ils veulent maximiser leur fonction objectif.
Par exemple, considérons un problème d'ordonnancement sur deux machines dans lequel les tâches

(agents) cherchent à minimiser leur date de fin d'exécution. Un mécanisme de coordination consiste par exemple à spécifier que la première (respectivement seconde) machine utilise une politique d'ordonnancement SPT (respectivement LPT). Cela signifie que les tâches qui sont ordonnancées sur la première (respectivement seconde) machine le seront selon l'ordre croissant (respectivement décroissant) de leur durée d'exécution. La tâche la plus petite (respectivement longue) a ainsi intérêt à aller sur la première (respectivement seconde) machine puisqu'elle sait qu'elle sera la première tâche a être ordonnancée sur cette machine. Plus généralement chaque tâche peut déterminer sur quelle machine elle a intérêt à se placer (une tâche ira sur la première machine si et seulement si la somme des longueurs des tâches plus petites qu'elle est inférieure à la somme des longueurs des tâches plus grandes qu'elle). Avec ce mécanisme de coordination, l'ordonnancement obtenu est $\frac{4}{3}$-approché vis à vis du *makespan*. Le rapport de coordination de ce mécanisme est donc égal à 4/3. Christodoulou et al. [57] ont proposé d'autres mécanismes de coordination avec un meilleur rapport d'approximation.

Dans une autre direction, un point de vue plus centralisé a été introduit dans [29]. En effet, dans de nombreuses applications liées aux réseaux, les agents ne sont pas totalement autonomes : en fait ils interagissent avec un protocole qui propose une solution collective aux participants. Chaque participant peut soit accepter la solution proposée, soit la refuser. Clairement, le concepteur du protocole doit proposer le meilleur équilibre de Nash, afin que la solution soit bonne pour la fonction globale, et stable (aucun agent n'a intérêt à unilatéralement changer de stratégie dans un équilibre de Nash). Cela motive la définition suivante introduite dans [29].

Définition 16 *Le* prix de la stabilité *pour une instance, est égal au rapport entre la valeur de la fonction objectif globale dans le* meilleur *équilibre de Nash pour cette instance, et sa valeur dans une solution optimale (qui n'est pas forcément un équilibre de Nash) pour cette instance. Le prix de la stabilité pour un problème de minimisation est égal au maximum du prix de la stabilité de chacune de ses instances.*

Récemment des questions de ce type ont été posées pour plusieurs problèmes, parmi lesquels des problèmes d'ordonnancement [57, 68, 69, 119, 148], des problèmes de routage [64, 149, 150], et des problèmes d'emplacement de ressources [171].

Nous avons cherché à faire baisser le prix de l'anarchie en relaxant la définition d'un équilibre de Nash.

Définition 17 *Nous disons qu'une solution est un équilibre de Nash α-approché* ($\alpha \geq 1$) si, lorsque un agent décide de changer de manière unilatérale sa stratégie, son nouveau profit est inférieur ou égal à son profit actuel multiplié par α.*

Lorsque $\alpha = 1$ on retrouve la notion classique d'équilibre de Nash. Les solutions que nous considérons *stables* sont celles qui correspondent à des équilibres de Nash approchés. Cela revient à prendre en compte une certaine inertie dans le changement : un agent ne décide de changer sa stratégie que si la différence entre son nouveau profit et son profit actuel est suffisamment grande.

De la même manière, nous avons défini le prix de la stabilité α-approchée.

Définition 18 *Le* prix de la stabilité α-approchée *est le rapport entre la valeur de la fonction objectif globale dans le* meilleur *équilibre de Nash α-approché, et sa valeur dans une solution optimale (qui n'est pas forcément un équilibre de Nash).*

*Lipton et al. [126] ont introduit une définition similaire. Une solution est un équilibre ε-Nash approché si, lorsque un agent décide de changer de manière unilatérale sa stratégie, son nouveau profit est inférieur ou égal à son profit actuel *plus* ε.

Dans le chapitre 8 nous considérons un problème d'ordonnancement pour lequel nous proposons un algorithme capable de retourner des solutions stables et de bonne qualité.

1.3.2 Véracité

L'hypothèse qui est généralement faite est que les agents sur lesquels les protocoles s'appliquent sont "honnêtes". Cette supposition peut s'avérer non réaliste dans certains cas, en particulier si les agents se rendent compte qu'il est possible de manipuler le protocole à leur avantage en lui fournissant de fausses informations.

La construction de mécanismes (Mechanism Design) est un domaine de la théorie des jeux, dans lequel on cherche à concevoir des protocoles dans lesquels les agents ont intérêt à dire la vérité, et qui fournissent des solutions de bonne qualité vis à vis de la collectivité. La technique la plus classique, et de loin la plus connue, pour la construction de tels mécanismes avec véracité garantie, est connue sous le nom de mécanisme Vickrey-Clarke-Groves (VCG) [59, 93, 172]. L'idée principale consiste à rémunérer les agents de telle manière à ce que le bénéfice de chaque agent (le profit qu'il retire de la solution retournée par le mécanisme plus le paiement qu'il reçoit) est maximisé lorsqu'il dit la vérité. Cependant, lorsqu'il est appliqué à des problèmes d'optimisation combinatoire, ce mécanisme ne fonctionne que lorsque la fonction objectif globale, c'est-à-dire celle qui mesure la qualité de la solution vis à vis de la collectivité, est de type *utilitaire*, ce qui signifie qu'elle doit s'exprimer sous la forme d'une somme des fonctions utilités de chacun des agents. De plus le mécanisme doit être en mesure de calculer une solution optimale du problème d'optimisation, ce qui implique donc que ce dernier ne soit pas **NP**-difficile. Récemment Archer et Tardos [31] ont introduit un mécanisme avec véracité garantie qui s'applique lorsque la fonction objectif globale n'est pas de type utilitaire, mais qui nécessite certaines hypothèses supplémentaires.

Dans le chapitre 9, nous avons proposé un mécanisme avec véracité garantie ad hoc pour un problème d'ordonnancement qui ne pouvait pas être traité avec les techniques générales de construction de mécanismes précédemment connues. Dans un deuxième temps, nous avons également proposé un mécanisme de coordination avec véracité garantie.

1.3.3 Équité

La théorie des jeux coopératifs s'applique dans des situations où plusieurs agents décident d'entreprendre en commun un projet dans le but de réduire leurs coûts. Ainsi les participants ou agents ont deux problèmes à résoudre. D'une part ils doivent exécuter leur projet de manière optimale. D'autre part ils doivent se répartir, de manière équitable, le coût global du projet. On parle alors d'*allocation*. Le premier problème est du domaine de la recherche opérationnelle tandis que le second fait appel à la théorie des jeux.

Définition 19 *Un* jeu de coalition *avec coûts transférables* $\langle V, c\rangle$ *consiste en un nombre fini de joueurs et une fonction* c *qui associe à tout sous-ensemble non vide* S *de* V *(une coalition) un nombre réel* $c(S)$ *qui est le coût contracté par la coalition* S*, c'est-à-dire le prix que les joueurs de la coalition* S *doivent payer collectivement pour avoir accès au service.*

Soit x_i, pour $i \in V$ le coût que le joueur i doit payer. Une question centrale est de savoir comment allouer le coût $c(V)$ parmi tous les joueurs.

Celle allocation doit être stable vis à vis de toutes les coalitions possibles qui pourraient se former. On veut en effet éviter la situation dans laquelle un groupe de participants se rend compte qu'il peut faire mieux en travaillant sans les autres. Ceci n'est pas possible lorsque l'allocation des coûts

appartient au *cœur* du jeu. Une solution $x = (x_i)_{i \in V}$ appartient au *cœur*, noté $\mathcal{C}$, si aucune coalition ne peut obtenir pour tous ses membres un meilleur coût que l'affectation courante $(x_i)_{i \in V}$ [131].

Définition 20 *Le cœur $\mathcal{C}$ du jeu de coalition $\langle V, c \rangle$ est l'ensemble des vecteurs coûts $(x_i)_{i \in V}$ tels que $\sum_{i \in V} x_i = c(V)$ et pour tout $S \subseteq V$ on ait $\sum_{i \in S} x_i \leq c(S)$. Une définition équivalente est de dire que le cœur est l'ensemble des vecteurs coûts $(x_i)_{i \in V}$ pour lesquels il n'existe pas de coalition S et de vecteur coût $(y_i)_{i \in V}$ pour lequel $\sum_{i \in S} y_i = c(S)$ et $y_i < x_i$ pour tout $i \in S$.*

Ainsi, étant donnée une solution du cœur, il n'y a aucune incitation pour un agent de quitter la grande coalition V.

Comme nous l'avons dit, une voie importante en théorie des jeux consiste à chercher une allocation qui appartiennent au cœur. Cependant, si pour certains jeux la situation est problématique en raison d'un cœur vide, pour d'autres jeux la situation est problématique en raison du très grand nombre d'allocations qui appartiennent au cœur. Dans ce dernier cas, un participant, même s'il est incité à participer, peut être insatisfait par le prix qu'il lui est demandé comparé à un meilleur prix (du point de vue individuel) qu'il pourrait payer dans une allocation différente appartenant également au cœur. Nous avons voulu prendre en compte cette situation que l'on appelle par la suite le *mécontentement* des participants. Nous avons ainsi introduit un nouveau critère, à savoir le *rapport de mécontentement*.

Définition 21 *Soit le jeu de coalition $\langle V, c \rangle$, ayant pour cœur $\mathcal{C}$. Étant donnée une solution $(x_i)_{i \in V}$, le* mécontentement *de l'agent i, noté $\Delta_i(x, \mathcal{C})$, est défini par :*

$$\Delta_i(x, \mathcal{C}) = \frac{x_i}{\min_{y \in \mathcal{C}} y_i}.$$

Nous avons cherché à étudier le problème qui consistait à déterminer une solution qui à la fois appartient au cœur, et qui minimise ce rapport. Ceci donne lieu à deux problèmes d'optimisation :

- MPM (Minimiser le Pire Mécontentement), *i.e.* déterminer une allocation $x = (x_i)_{i \in V} \in \mathcal{C}$ qui minimise $\max_{i \in V} \Delta_i(x, \mathcal{C})$;
- MMM (Minimiser le Mécontentement Moyen), *i.e.* déterminer une allocation $x = (x_i)_{i \in V} \in \mathcal{C}$ qui minimise $\sum_{i \in V} \Delta_i(x, \mathcal{C})$.

Cette approche est illustrée dans le chapitre 10 pour le problème classique du jeu de l'arbre couvrant de poids minimum ainsi qu'un problème d'ordonnancement.

Chapitre 2

Problèmes abordés

Sommaire

2.1 Problèmes issus de la théorie des graphes

2.1.1 Le problème du voyageur de commerce

Le problème du voyageur de commerce, noté TSP (pour *travelling salesman problem*) par la suite, fait partie des problèmes classiques **NP**-difficiles en optimisation combinatoire. Rappelons qu'étant donné un graphe G, non orienté, complet où chaque arête e possède une longueur d_e, le but est de déterminer un tour ou cycle hamiltonien (cycle passant exactement une fois par chaque sommet du graphe) de longueur minimum.

Si aucune hypothèse n'est faite sur les distances alors le problème n'est pas approximable [133]. Cependant, si les distances respectent l'inégalité triangulaire, c'est-à-dire que pour tout triplet i, j, k de sommets de G, la distance entre i et j est nécessairement plus petite que la distance entre i et k plus la distance entre k et j, alors il existe un algorithme polynomial 3/2-approché [58].

Nous nous sommes intéressés à un cas particulier de ce problème, noté TSP$(1,2)$ par la suite, dans lequel la longueur de chaque arête est soit 1, soit 2. Ce problème peut être vu comme une généralisation du problème **NP**-complet CYCLE HAMILTONIEN ce qui prouve son caractère **NP**-difficile [133]. Papadimitriou et Yannakakis [134] ont proposé un algorithme 7/6-approché pour le TSP$(1,2)$ basé sur une technique d'assemblage de sous-cycles. Récemment ce rapport d'approximation a été amélioré. Ainsi Bläser et Ram [44] ont obtenu un rapport d'approximation égal à 65/56, tandis que Berman et Karpinski [42] obtenaient un rapport égal à 8/7. Actuellement la meilleure borne d'inapproximabilité connue est de 741/740 [78]

Nous avons considéré le problème TSP$(1,2)$ dans un cadre multicritère. Dans un cadre bicritère, chaque arête $e \in E$ possède un vecteur de distances $\vec{d}(e) \in \{1,2\}^2$ tel que $\vec{d}_1(e)$ (respectivement $\vec{d}_2(e)$) désigne la première coordonnée du vecteur distance (respectivement la seconde coordonnée du vecteur distance) $\vec{d}(e)$. Une solution réalisable est un cycle hamiltonien (ou tour). La distance totale sur le premier critère d'un tour t, notée $\vec{D}_1(t)$ vaut $\sum_{e\in t} \vec{d}_1(e)$. La distance totale sur le second critère d'un tour t, notée $\vec{D}_2(t)$ vaut $\sum_{e\in t} \vec{d}_2(e)$. On cherche à minimiser la distance sur chacun des critères. Ce problème bicritère est **NP**-difficile car il l'est déjà dans sa version monocritère. Nous

avons obtenu des résultats d'approximation avec l'approche courbe de Pareto, ainsi que des résultats d'inapproximabilité, qui sont présentés dans le chapitre 6.

2.1.2 Le problème de tournées de véhicules

Les problèmes de tournées de véhicules figurent parmi les problèmes les plus étudiés en recherche opérationnelle [167]. Il en existe de nombreuses variantes.

Le problème de tournées de véhicules que nous avons considéré, noté par la suite VRP (pour *vehicle routing problem*), est le suivant : n clients doivent être servis par un dépôt unique. Chaque client demande un bien, et un véhicule de capacité C est disponible pour délivrer les biens. Comme la capacité du véhicule est limitée, le véhicule doit périodiquement retourner au dépôt pour se recharger. Il n'est pas possible de livrer un client en plusieurs fois. Ainsi, une solution du VRP est une suite de tours dans lesquels chaque client est servi une seule fois, et la quantité de biens livrés ne doit pas excéder C (le nombre de clients servis par tour doit donc être d'au plus C). A chaque couple (i, j), où i et j sont des clients (ou le dépôt), est associée une valeur $d_{i,j}$ représentant la distance (ou temps de parcours) de i à j. Le but est de trouver un ensemble de tours minimisant la distance totale parcourue. Chaque tour commence et termine au dépôt.

Ce problème qui généralise le problème du voyageur de commerce est **NP**-difficile, et la recherche locale reste l'un des meilleurs outils pour trouver des solutions presque optimales pour de grandes instances du VRP [87]. Nous avons proposé, pour ce problème, un voisinage de taille exponentielle pouvant néanmoins être exploré de manière efficace, c'est-à-dire en temps polynomial. Ces travaux sont exposés dans le chapitre 3.

2.1.3 Le problème de la coupe maximum

Soit $G(V, E)$ un graphe non-orienté tel que toute arête $[i, j] \in E$ possède un poids w_{ij} positif. Une coupe est une partition de V en deux sous-ensembles S et $\overline{S}$. Le poids de la coupe $(S, \overline{S})$, noté $W(S, \overline{S})$, est défini par :

$$W(S, \overline{S}) = \sum_{i \in S,\, j \in \overline{S}} w_{ij}.$$

Dans le problème de la coupe maximum, noté MAX-CUT par la suite, on cherche à déterminer une coupe de poids maximum. Il s'agit d'un problème **NP**-difficile [117] qui a des applications dans de nombreux domaines incluant l'élaboration de circuits VLSI et la physique statistique [40]. Concernant ce problème, Sahni et Gonzalez [153] ont analysé un algorithme simple issu de [115] et ont montré qu'il était 1/2-approché. En utilisant la programmation semi-définie, Goemans et Williamson [88, 89] ont obtenu un algorithme 0.879-approché.

Nous nous sommes intéressés à une version bicritère de ce problème. Dans cette version chaque arête $e \in E$ possède maintenant un poids $w_e > 0$ et une longueur $l_e > 0$. Les fonctions objectifs suivantes, à savoir le *poids* et la *longueur*, sont prises en considération : $W(S, \overline{S}) = \sum_{e \in (S, \overline{S})} w_e$ et $L(S, \overline{S}) = \sum_{e \in (S, \overline{S})} l_e$. Nous avons proposé des algorithmes afin d'approcher le point idéal. Ces travaux sont exposés dans le chapitre 5.

Peu d'articles traitent de problèmes de coupe dans un cadre bicritère. Cependant, on recense un article récent de Bruglieri et al. [49] où il est question de coupe minimale sujette à une contrainte de cardinal. Papadimitriou et Yannakakis [135] ont proposé un théorème concernant la non existence d'un schéma d'approximation totalement polynomial pour la construction d'une courbe de Pareto $\vec{\varepsilon}$-approchée pour le problème $s-t$ MIN-CUT bicritère. Dans cette version du problème, deux sommets s et t donnés doivent être impérativement séparés par la coupe.

2.1.4 Le problème de l'arbre couvrant de poids minimum

Supposons qu'un service, issu d'un sommet appelé la source, doit être apporté à un ensemble de clients V à travers un réseau représenté à l'aide d'un graphe $G = (V, E)$. Chaque arête $e \in E$ a un coût positif. Le service peut être apporté directement à un client s'il existe une arête entre la source et ce dernier où via d'autres clients. La structure minimale pouvant être utilisée est un arbre $T \subseteq E$ contenant la source et couvrant tous les clients. Cet arbre possède un coût $\sum_{e \in T} c_e$ que les clients doivent se partager.

Le problème consistant à trouver un arbre couvrant de poids minimum dans un graphe G est un problème polynomial, qui peut être résolu par exemple à l'aide des algorithmes de Prim et de Kruskal [133]. Nous avons étudié ce problème dans le contexte de la théorie des jeux coopératifs. Ces travaux sont exposés dans le chapitre 10.

2.2 Problèmes d'ordonnancement

Les problèmes d'ordonnancement sont extrêmement nombreux et variés. Le lecteur peut se référer aux livres [45, 101], ainsi qu'à l'article [55], pour avoir une vue d'ensemble de quelques résultats fondamentaux du domaine.

De manière générale, on dispose d'un ensemble de tâches que l'on cherche à ordonnancer sur une ou plusieurs machines, de manière à optimiser une ou plusieurs fonctions objectifs. Les problèmes d'ordonnancement peuvent être classés selon les caractéristiques des machines et les différentes fonctions objectifs que l'on cherche à optimiser.

2.2.1 Caractéristiques des machines

On fait souvent l'hypothèse qu'à un instant donné, une machine ne peut traiter qu'une seule tâche. Dans ce contexte :

- Les machines peuvent être *identiques*, ce qui signifie que le temps d'exécution d'une tâche est le même sur toutes les machines. Dans la notation standard de Graham et al. [91], on note P pour signifier que l'on dispose de m machines identiques. Si m est une constante on note Pm.
- Les machines peuvent être *reliées*. Cela signifie que chaque machine a une certaine vitesse, et que le temps d'exécution d'une tâche est proportionnel à la vitesse de la machine sur laquelle elle s'exécute. Dans la notation de Graham et al. on note Q pour signifier que l'on dispose de m machines reliées.
- Les machines peuvent être *non reliées*, ce qui signifie que le temps d'exécution d'une tâche dépend de la machine sur laquelle elle s'exécute (et ceci de manière arbitraire). Dans la notation de Graham et al. on note R pour signifier que l'on dispose de m machines non reliées.

Par contre, dans le modèle à base de lots (*batch*), chaque machine peut traiter simultanément $b \geq 1$ tâches. Lorsque $b = 1$ le modèle est donc le même que celui évoqué plus haut. Les tâches qui sont traitées simultanément forment un lot.

2.2.2 Caractéristiques des tâches et fonctions objectifs

Dans les problèmes que nous avons considérés, la préemption n'est pas autorisée, autrement dit une tâche ne peut jamais être interrompue entre le début et la fin de son exécution. De plus, par la suite on ne considère que les ordonnancements qui ne laissent aucun temps d'inactivité entre l'exécution de deux tâches.

Soit $J = \{1, 2, \ldots, n\}$ un ensemble de tâches. La caractéristique principale d'une tâche est bien sûr sa durée d'exécution. Dans la suite, pour une tâche $j \in J$, on note, dans le cas de problèmes avec une seule machine, p_j sa durée d'exécution. Pour les problèmes avec plusieurs machines, on note p_{ij} la durée d'exécution de la tâche j lorsqu'elle est exécutée sur la machine i. Étant donné un ordonnancement des tâches, on note C_j le *temps de complétude* de la tâche j, c'est-à-dire la date à laquelle elle se termine.

2.2.3 Problèmes d'ordonnancement sur 1 machine

Tout ordonnancement de tâches sur une seule machine peut être représenté à l'aide d'une permutation π des tâches $\{1, \ldots, n\}$, de sorte que $\pi(i)$, avec $1 \leq i \leq n$, indique la i-ème tâche dans l'ordonnancement. De manière symétrique, on note $\pi^{-1}(j)$ la position de la tâche j dans la permutation π. Pour une permutation π des tâches, la date C_j à laquelle se termine la tâche $j = \pi(i)$ est égale à $\sum_{k=1}^{i} p_{\pi(k)}$.

Le problème $1||\sum w_j C_j$
Dans le problème $1||\sum w_j C_j$, on associe un poids w_j à chaque tâche j, et on cherche un ordonnancement des tâches sur la machine, de manière à minimiser la somme pondérée des temps de complétude des tâches. Nous appelons *poids total* $w(\pi) = \sum_{1 \leq j \leq n} w_j C_j$ cette fonction objectif à minimiser.

Ce problème d'optimisation peut être résolu en temps polynomial. En effet, Smith [163] a prouvé qu'il suffit de placer les tâches selon leurs rapports w_j/p_j décroissants pour obtenir un ordonnancement optimal. Plus précisément, on prend une à une chaque tâche selon l'ordre de Smith, et on la place sur la machine la moins chargée (celle dont la dernière tâche se termine le plus tôt). Un tel algorithme glouton est appelé un algorithme de liste. Réciproquement, tout ordonnancement optimal est tel que les tâches sont placées selon leurs rapports w_j/p_j décroissants. Bien entendu, lorsque tous les poids sont égaux à 1, on est en présence d'un cas particulier, noté $1||\sum_j C_j$, qui peut selon la règle de Smith être résolu de manière optimale en ordonnançant chacune des tâches selon l'ordre SPT (*shortest processing time*), c'est-à-dire en ordonnançant les tâches selon l'ordre croissant de leur durée.

Nous avons étudié une version bicritère de ce problème décrite ci-dessous.

Le problème $1||\sum w_j C_j, \sum c_j C_j$
Dans le problème bicritère
$1||\sum w_j C_j, \sum c_j C_j$ on s'intéresse au problème d'ordonnancement bicritère issu de $1||\sum w_j C_j$ auquel on ajoute pour chaque tâche j un coût c_j ainsi qu'une seconde fonction objectif : $c(\pi) = \sum_{1 \leq j \leq n} c_j C_j$. Cette fonction objectif, appelée *coût total* par la suite, est de même nature que la fonction de poids total.

Bien que polynomial dans sa version monocritère comme nous l'avons vu à l'aide de la règle de Smith, le problème $1||\sum w_j C_j, \sum c_j C_j$ est **NP**-difficile. En effet, Hoogeveen a montré à l'aide d'une réduction du problème PARTITION que la version de décision de ce problème est **NP**-complète [102]. Nous avons montré qu'il était possible d'approcher le point idéal. Ces travaux sont exposés dans le chapitre 5.

Le problème $1\,|\,p_j^t, idleness\,|\,\sum_j w_j T_j$
Dans la théorie de l'ordonnancement, il y a un intérêt croissant, durant ces dernières années, pour les problèmes dont les durées d'exécution des tâches ne sont pas constantes, mais varient au cours du temps [2]. Nous avons considéré une version dynamique du problème $1||\sum_j w_j T_j$. Dans ce dernier problème, chaque tâche j possède une date de fin d'achèvement d_j. On note $T_j = \max\{C_j - d_j, 0\}$

le retard de la tâche j, et on cherche à obtenir un ordonnancement qui minimise la somme pondérée des retards de toutes les tâches : $\sum_{j=1}^{n} w_j T_j$.

Dans la version dynamique que nous avons considérée, le temps d'exécution $f_j(t)$ de chaque tâche j dépend de la date t à laquelle elle commence à s'exécuter, et est donné par une fonction f_j. Par la suite on note $f_j(t)$ par p_j^t. Ainsi, si une tâche j commence à s'exécuter immédiatement après une tâche i, sa durée est $p_j^{C_i}$, avec C_i la date d'achèvement de la tâche i. En utilisant la notation standard de Graham et al. [91] ce problème peut être noté par $1\,|\,p_j^t, idleness\,|\,\sum_j w_j T_j$.

Ce problème est fortement **NP**-difficile puisqu'il s'agit d'une généralisation du problème $1\,||\,\sum_j w_j T_j$ [122]. Ce dernier problème peut être résolu à l'aide de la programmation dynamique en temps $\mathcal{O}(n^3 \sum_{j=1}^{n} p_j)$, mais uniquement lorsque les poids vérifient $p_j \geq p_k \Rightarrow w_j \leq w_k$ pour toutes les tâches j et k [122]. Celui-ci a également été traité avec un algorithme *branch and bound* [141], mais il ne peut être utilisé en pratique sur des instances de plus de 50 tâches [62]. Il existe un schéma d'approximation totalement polynomial dû à Lawler [123] et amélioré par Kovalyov [120] en temps $\mathcal{O}(n^6 \log n + n^6/\varepsilon)$. Comme le problème que nous considérons $1\,|\,p_j^t, idleness\,|\,\sum_j w_j T_j$ constitue une généralisation du problème $1\,||\,\sum_j w_j T_j$, il est peu vraisemblable qu'il existe un schéma d'approximation totalement polynomial pour lui.

Nous avons proposé pour ce problème un voisinage exponentiel explorable en temps polynomial, et nous avons mené des expérimentations afin de mesurer l'intérêt pratique d'un tel voisinage. Ces travaux sont exposés dans le chapitre 3.

Le problème $1(batch, C_{batch}, p_{max})||\Sigma w_j C_j, C_{max}$

On se place dans le modèle à base de lots évoqué dans la section 2.2.1. On dispose d'une machine, qui peut traiter un nombre quelconque de tâches simultanément. Par conséquent un ordonnancement est une suite de lots $(\mathcal{B}_1, \ldots, \mathcal{B}_r)$, où chaque lot $\mathcal{B}_i$ $(i = 1, \ldots, r)$ est constitué d'un ensemble de tâches. Dans le modèle *unbounded burn-in* que nous avons considéré, le temps d'exécution d'un lot est égal à la durée maximum des tâches qui le composent. Autrement dit, le temps d'exécution d'un lot $\mathcal{B}_i$ est $p(\mathcal{B}_i) = max_{j \in \mathcal{B}_i} p_j$, et sa date de fin de complétude est égale à $C(\mathcal{B}_i) = \sum_{k=1}^{i} p(\mathcal{B}_k)$. Pour chaque tâche $j \in \mathcal{B}_i$ son temps de complétude C_j est égal à $C(\mathcal{B}_i)$.

Un critère est dit *régulier* si il est non décroissant en fonction du temps de complétude des tâches. Lorsqu'on considère de tels critères pour la fonction objectif à optimiser, il existe toujours une solution optimale dans laquelle les lots sont exécutés les uns à la suite des autres, sans temps d'inactivité. Brucker et al. [46] ont étudié dans un cadre monocritère différentes fonctions objectifs régulières.

Dans le problème que nous avons considéré, on cherche à minimiser simultanément les deux critères réguliers suivants : le *makespan* de l'ordonnancement $C_{max} = \max_{1 \leq j \leq n} C_j$, et la somme pondérée des temps de complétude des tâches $\sum_j w_j C_j$.

2.2.4 Problèmes d'ordonnancement sur $m \geq 2$ machines

Le problème $P||\sum_j C_j$

Dans ce problème, on dispose de m machines identiques et on cherche à minimiser la somme des temps de complétude des tâches.

On peut résoudre ce problème de manière optimale en temps polynomial à l'aide d'un algorithme de liste SPT. Plus précisément, il est possible de caractériser tous les ordonnancements optimaux pour ce problème. On peut les définir à l'aide de la notion de *rang* [63]. On dit d'une tâche qu'elle se situe au i-ième rang si sa durée est inférieure ou égale à celle des tâches qui sont de rang plus grand que i, et si sa durée est supérieure ou égale à celle des tâches qui sont de rang plus petit que i. Plus précisément, en supposant que les tâches 1 à n ont été numérotées de manière à vérifier

$p_1 \leq p_2 \leq \cdots \leq p_n$. On définit des sous-ensembles de tâches de la manière suivante : $\pi_k = \{n, n-1, \ldots, n-m+1\}$, $\pi_{k-1} = \{n-m, \ldots, n-2m+1\}$, $\ldots$, $\pi_1 = \{n-(k-1)m, \ldots, 1\}$, avec $k = \lceil \frac{n}{m} \rceil$. Les tâches de rang i sont celles qui font partie du sous-ensemble π_i. Les ordonnancements optimaux sont obtenues en ordonnançant les tâches selon leur rang dans l'ordre $\pi_1, \pi_2, \ldots, \pi_k$, de telle manière à ce que deux tâches ayant le même rang ne soit pas ordonnancées sur la même machine. Les tâches de même rang peuvent être placées selon toutes les permutations possibles sur chacune des m machines. On fait référence à cette famille d'ordonnancements lorsque plus tard on parle de la famille des ordonnancements SPT.

Le problème $P||C_{max}$
Dans ce problème, on dispose de m machines identiques et on cherche à minimiser le *makespan*, c'est-à-dire la date à laquelle la dernière tâche a fini de s'exécuter.

Nous aurons l'occasion de rencontrer ce problème dans le chapitre 8 concernant la recherche de solutions stables ainsi que dans le chapitre 9 concernant les mécanismes avec véracité garantie.

Nous rappelons quelques résultats classiques qui seront utilisés par la suite. L'algorithme de liste SPT possède un rapport d'approximation égal à $(2 - 1/m)$, où m est le nombre de machines. L'algorithme de liste LPT possède un rapport d'approximation égal à $4/3 - 1/(3m)$ [90]. Enfin signalons qu'il existe un schéma d'approximation polynomial pour ce problème, appelé l'algorithme de Graham [90].

Le problème $P2||\sum w_j C_j, C_{max}$
Dans ce problème bicritère, on dispose de 2 machines identiques, et on cherche un ordonnancement qui minimise la somme pondérée des temps de complétude et le *makespan*. Chacun des critères pris isolément donne lieu à un problème monocritère NP-complet (au sens faible) [51, 117], mais il existe un schéma d'approximation totalement polynomial pour chacun des 2 problèmes monocritères [173]. Ce problème est étudié dans la section 7.3.

Le problème $Rm||C_{max}, \sum_j c_j$
On dispose de m machines non reliées. L'exécution de la tâche j sur la machine i nécessite un temps égal à p_{ij} et induit un coût (de placement) égal à c_{ij}. On cherche un ordonnancement qui minimise à la fois le *makespan* et la somme du coût de placement des tâches.

Lin et Vitter [125] ont proposé un algorithme polynomial qui, étant donné T, C et $\varepsilon > 0$, trouve une solution de *makespan* $(2+\frac{1}{\varepsilon})T$ et de coût total $(1+\varepsilon)C$, si il existe un ordonnancement ayant un *makespan* égal à T et de coût total C. Ce résultat s'appuie sur la programmation linéaire. Shmoys et Tardos [162] ont amélioré ce résultat en présentant un algorithme polynomial qui, étant donnés T et C, trouve un ordonnancement de coût au plus C et avec un *makespan* au plus égal à $2T$, si il existe un ordonnancement de coût C et avec un *makespan* égal à T. Ce résultat ne peut pas être amélioré de manière substantielle dans le cas où le nombre de machines n'est pas une constante, puisque Lenstra et al. [124] ont prouvé qu'il n'existe pas d'algorithme $(1+\varepsilon)$-approché avec $\varepsilon < 1/2$, à moins que **P=NP**, pour le problème monocritère où l'on cherche à minimiser le *makespan*.

Pour le problème bicritère que nous considérons ici, Jansen et Porkolab [112] ont proposé un schéma d'approximation totalement polynomial qui étant données des valeurs pour T et C, retourne pour tout $\varepsilon > 0$, un ordonnancement en temps $O(n(m/\varepsilon)^{\mathcal{O}(m)})$ ayant un *makespan* d'au plus $(1+\varepsilon)T$ et un coût total d'au plus $(1+\varepsilon)C$, si il existe un ordonnancement ayant un *makespan* égal à T et un coût total égal à C. Ce résultat s'appuie sur la programmation linéaire et utilise des méthodes assez sophistiquées.

Nous avons proposé un algorithme qui étant donné T, retourne pour tout $\varepsilon > 0$ un ordonnancement avec un *makespan* au plus égal $(1+\varepsilon)T$ et de coût au plus $C_{opt}(T)$, et ceci en temps

$\mathcal{O}(n(n/\varepsilon)^m)$. De plus, nous avons adapté notre algorithme afin d'obtenir une courbe de Pareto $(\varepsilon, 0)$-approchée. Ces travaux sont exposés dans le chapitre 7.

2.2.5 Problèmes avec communications hiérarchiques

Pendant de nombreuses années, le modèle standard de communication pour l'ordonnancement de tâches d'un programme parallèle a été le modèle homogène introduit par Rayward-Smith [145]. On dispose d'un ensemble de processeurs identiques qui sont capables de communiquer entre eux et d'un ensemble de tâches qui sont sujettes à des contraintes de précédence.

Nous avons considéré le modèle de communication hiérarchique [37, 39] dans lequel nous supposons que les délais de communication ne sont plus homogènes. Les processeurs sont regroupés sous forme de *clusters* et les communications à l'intérieur d'un même cluster sont beaucoup plus rapides que les communications entre des processeurs faisant partie de clusters différents. Ce modèle est plus proche de l'architecture actuelle des ordinateurs parallèles (le lecteur peut consulter [37, 53] pour des modélisations apparentées).

On dispose d'un ensemble de machines multiprocesseurs (où clusters) qui sont utilisées pour traiter n tâches possédant des contraintes de précédence. On note p_i la durée de la tâche i. Chaque machine (cluster) est composée de plusieurs processeurs identiques. Chaque arc (i, j) du graphe de précédence est valué par un couple (c_{ij}, ϵ_{ij}) indiquant des temps de communication. La quantité c_{ij} (respectivement ϵ_{ij}) est appelée le temps de communication inter-cluster (respectivement inter-processeur), et on suppose que $c_{ij} \geq \epsilon_{ij}$. Si les tâches i et j sont exécutées sur des machines différentes, alors j doit être exécutée au moins c_{ij} unités de temps après la date de complétude de la tâche i. Si les tâches i et j sont exécutées sur la même machine, mais sur des processeurs différents, alors la tâche j ne peut commencer à s'exécuter au minimum après ϵ_{ij} unités de temps après la date de complétude de la tâche i. Cependant, si i et j sont exécutées sur le même processeur, alors j peut commencer à s'exécuter immédiatement après que la tâche i ait finie de s'exécuter.

Dans [38], il a été montré qu'il n'existe pas d'algorithme ρ-approché avec $\rho < 5/4$, même pour le cas simple UET-UCT ($p_i = 1; (c_{ij}, \epsilon_{ij}) = (1, 0)$) avec un nombre non borné de machines bi-processeurs, noté par la suite $\bar{P}(P2)|prec; (c_{ij}, \epsilon_{ij}) = (1, 0); p_i = 1|C_{\max}$, à moins que **P**=**NP**. Dans le cas où chaque machine contient m processeurs, avec m une constante ($\bar{P}(Pm)|prec; (c_{ij}, \epsilon_{ij}) = (1, 0); p_i = 1|C_{\max}$), il existe un algorithme $\frac{4m}{2m+1}$-approché [37]. Dans le cas $c_{ij} = \epsilon_{ij}$, c'est-à-dire le modèle classique avec des délais de communication, et sous l'hypothèse $\Phi = \frac{\min p_i,\ i \in V}{\max c_{kj},\ (k,j) \in E} \geq 1$, Hanen et Munier [128] ont proposé un algorithme $\frac{2(1+\Phi)}{2\Phi+1}$-approché pour le problème comportant un nombre non borné de machines.

Nous avons considéré pour la première fois le cas où le nombre de clusters était borné et plus précisément le problème dans lequel on a 2 clusters, chacun étant constitué de processeurs identiques : $(P2(P)|prec; (c_{ij}, \epsilon_{ij}) = (1, 0); p_i = 1|C_{max})$. Nous avons prouvé que le problème qui consiste à déterminer s'il existe un ordonnancement de durée 3 était **NP**-complet, et ce même pour des graphes de précédence bipartis. Ce résultat implique qu'il n'existe pas d'algorithme polynomial approché avec un rapport d'approximation meilleur que 4/3 (à moins que **P**=**NP**). Nous avons proposé un algorithme polynomial pour le problème de décision qui consiste à déterminer s'il existe un ordonnancement de longueur 2, dans le cas où le nombre de clusters est constant et le nombre de processeurs par cluster arbitraire.

2.3 Problèmes issus d'applications

2.3.1 Requêtes multiples dans une base de données

Nous avons considéré un problème issu de l'optimisation de requêtes dans les systèmes de base de données. Chaque requête est susceptible d'être résolue à l'aide de différents plans, et chacun de ces plans est composé de différentes tâches. Afin de réduire le temps d'exécution total de toutes les requêtes, il est souvent possible d'exploiter le fait que plusieurs plans partagent un même sous-ensemble de tâches [151, 157]. Ces tâches peuvent ainsi être traitées une seule fois, et leurs résultats servir à de multiples reprises lors de l'exécution de différentes requêtes. De manière informelle, le problème MQO (pour *Multiple-Query Optimization*) consiste, étant donné un ensemble de requêtes, et pour chacune d'entres elles, un ensemble de plans alternatifs susceptibles de la résoudre, à sélectionner exactement un plan par requête de manière à minimiser le temps total d'exécution de toutes les requêtes.

Plus précisément ce problème, qui est **NP**-difficile [157, 158] peut s'énoncer ainsi. Soit $Q = \{q_1, q_2, \ldots, q_k\}$ un ensemble de k requêtes, et soit $T = \{t_1, t_2, \ldots, t_r\}$ un ensemble de r tâches. Un plan p est un sous-ensemble de T et une requête q_i peut être résolue à l'aide de $n(i)$ plans distincts $P_i = \{p_i^1, p_i^2, \ldots, p_i^{n(i)}\}$. Autrement dit, chaque plan de P_i constitue une alternative pour résoudre la requête q_i. Nous notons par $P = \cup_{i=1}^{k} P_i$ l'ensemble de tous les plans. Chaque tâche t_j a un coût (ou durée d'exécution) $c_j \in \mathbb{Q}^+$. On cherche à sélectionner un plan par requête, par conséquent une solution peut être vue comme un vecteur $s = (s_1\, s_2\, \ldots\, s_k)$ où $s_i \in \{1, \ldots, n(i)\}$ ($p_i^{s_i}$ est sélectionné pour q_i). Étant donnée une solution s, son coût total est défini par

$$C(s) = \sum_{\{j | t_j \in \bigcup_{i=1}^{k} p_i^{s_i}\}} c_j.$$

Le coût d'une tâche qui appartient à plusieurs plans n'est compté qu'une seule fois.

Exemple 1 *Considérons l'instance suivante représentée à la figure 2.1 :*
- $Q = \{q_1, q_2, q_3\}$
- $P_1 = \{p_1^1, p_1^2\}$, $P_2 = \{p_2^1, p_2^2, p_2^3\}$, $P_3 = \{p_3^1, p_3^2\}$
- $p_1^1 = \{t_1, t_3, t_4\}$, $p_1^2 = \{t_1, t_2\}$, $p_2^1 = \{t_2, t_4\}$, $p_2^2 = \{t_5\}$, $p_2^3 = \{t_1, t_2, t_3\}$, $p_1^3 = \{t_1, t_3\}$, $p_3^2 = \{t_4\}$
- $c_1 = c_2 = 3$, $c_3 = c_4 = 1$, $c_5 = 2$

La solution (1 3 1) *est de coût total* 8, *tandis que les solutions* (1 2 1) *et* (2 1 2) *sont chacune optimales, avec un coût total égal à* 7.

Un algorithme de type *branch-and-bound* a été proposé dans [157, 158], tandis qu'un algorithme à base de programmation dynamique a été utilisé dans [137, 166]. Roy et al. [151] ont proposé plusieurs heuristiques dont l'efficacité a été testée expérimentalement. Dans le domaine de l'approximation avec garantie de performance, d'autres problèmes issus de l'optimisation de requêtes avaient fait l'objet de travaux (voir entre autres [127, 132]), cependant, à notre connaissance, le problème MQO n'avait pas encore été étudié sous cet angle. Nous avons proposé différents algorithmes approchés pour ce problème. Ces travaux sont exposés dans le chapitre 4.

2.3.2 Groupage de trafic dans un réseau WDM en étoile

Récemment de nouveaux modèles afin d'utiliser au mieux la bande passante disponible dans les réseaux optiques ont vu le jour. L'introduction de convertisseurs optique/électronique permet

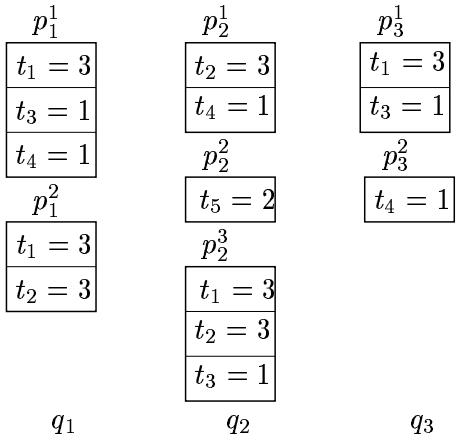


FIG. 2.1 – Un exemple provenant de [166].

de diminuer de manière significative la bande passante nécessaire pour router un trafic donné. L'inconvénient de cette approche est le coût souvent élevé de tels équipements. Ces observations ont poussé Dutta et Rouskas [74] à étudier le problème de groupage de trafic afin de trouver un bon équilibre entre le coût de l'infrastructure d'un réseau et ses performances. Il est en effet possible, depuis quelques années, de grouper plusieurs paquets dans une même longueur d'onde.

Nous considérons un réseau sous la forme d'une étoile avec $N + 1$ sommets. Il existe un unique sommet le *commutateur* qui est connecté à tous les autres par un lien physique. Tous les sommets, à l'exception du commutateur, sont divisés en deux groupes V_1 et V_2 : les sommets de V_1 sont les émetteurs, et les sommets de V_2 sont les récepteurs. Le commutateur est numéroté 0 et les N autres sommets sont numérotés de 1 à N dans un ordre arbitraire. Chaque lien physique consiste en une fibre optique, et chaque fibre peut transporter W longueurs d'ondes. Chaque longueur d'onde a une capacité C. Le trafic entre les sommets est modélisé sous la forme d'une matrice de trafic $T = [t_{ij}]$, où l'entier t_{ij} indique le nombre d'unités de trafic depuis le sommet $i \in V_1$ vers le sommet $j \in V_2$. Nous n'autorisons pas à ce que la demande de trafic entre deux sommets soit supérieure à la capacité d'une longueur d'onde, i.e. pour tout $(i, j), 0 \leq t_{ij} \leq C$.

Le commutateur possède des capacités optiques et électroniques : il peut laisser passer certaines requêtes de manière transparente à travers lui, mais il peut également arrêter certaines requêtes, les traiter, puis les envoyer vers leur destinataire. Une requête t_{ij} doit bénéficier d'une longueur d'onde à elle seule de i au commutateur et du commutateur à j pour être routée optiquement, alors qu'elle peut partager une longueur d'onde avec d'autres requêtes si elle est commutée électroniquement. Nous disons dans ce dernier cas que la requête est routée électroniquement. Le commutateur doit alors séparer plusieurs requêtes groupées dans la même longueur d'onde et les router vers leur destinataire (en les regroupant éventuellement avec d'autres requêtes). Ceci a un coût et le problème que nous considérons ici consiste donc à minimiser la quantité totale de trafic routé électroniquement. Ce problème est communément désigné dans la littérature sous le terme : problème de groupage de trafic (*traffic grooming problem*).

Il existe en fait deux versions du problème de groupage de trafic : soit on souhaite minimiser la quantité totale de trafic routé électroniquement, soit on souhaite maximiser la quantité totale de trafic routé optiquement. Ces deux versions sont équivalentes, une solution optimale pour l'une l'étant également pour l'autre.

Exemple 2 *Nous avons un émetteur i, qui a des demandes de trafic vers trois récepteurs j, k et*

l. Ces demandes sont les suivantes : $t_{ij} = t_{ik} = C/3$ *et* $t_{il} = C/2$*, avec* C *la capacité de chaque longueur d'onde. Le nombre de longueurs d'ondes par fibre est 2 (*$W = 2$*). Puisqu'il n'y a que 2 longueurs d'ondes à partir de* i*, les trois demandes de trafic ne peuvent pas être toutes routées optiquement. Si la demande de trafic* t_{ij} *est routée optiquement, elle occupe une longueur d'onde à elle seule entre* i *et le commutateur. Les demandes de trafic* t_{ik} *et* t_{il} *doivent alors partager la longueur d'onde restante vers le commutateur, et sont donc routées électroniquement : la quantité de trafic électroniquement routé est alors égale à* $C/3 + C/2 = 5C/6$*. Dans une solution optimale,* t_{il} *est routée optiquement, alors que* t_{ij} *et* t_{ik} *sont routées électroniquement : la quantité de trafic routé électroniquement est alors de* $2C/3$*.*

L'intérêt d'étudier des réseaux en étoile, en plus de leur simplicité, ce qui permet d'obtenir des algorithmes avec garantie de performance, est motivé par leur utilisation courante dans l'interconnexion de réseaux locaux ou de moyennes distances (LAN ou MAN) avec l'épine dorsale (*backbone*) d'un réseau étendu (WAN).

Dutta et Rouskas ont considéré dans [74] le problème de groupage de trafic pour plusieurs topologies, dont l'étoile. Cependant, il existe des différences entre le modèle qu'ils ont considéré et le nôtre : dans [74], chaque sommet du réseau, le commutateur inclus, peut être un émetteur et un récepteur, et la demande de trafic entre deux sommets peut être supérieure à la capacité maximum qu'il est possible de router à l'aide d'une seule longueur d'onde (i.e. il est possible d'avoir $t_{ij} > C$ entre deux sommets i et j). Afin de distinguer les deux problèmes, nous appelons leur problème le problème de groupage de trafic dans une étoile active, alors que notre problème sera appelé le problème de groupage de trafic dans une étoile passive. Remarquons qu'une fois que nous savons quelles requêtes sont routées optiquement, l'affectation des requêtes aux différentes longueurs d'onde est un problème facile.

Nous avons montré que le problème de groupage de trafic dans une étoile passive était **NP**-difficile. De plus nous avons proposé des algorithmes avec garantie de performance pour ce problème. Ces travaux sont exposés dans le chapitre 4.

Deuxième partie

Algorithmes approchés sans garantie de performance

Chapitre 3

Recherche locale et voisinages exponentiels

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [5, 11, 23].

3.1 Introduction

Les algorithmes de recherche locale, et de manière plus générale les métaheuristiques qui s'en inspirent telles que le recuit simulé et la recherche tabou, sont souvent utilisés pour obtenir des solutions de bonne qualité pour un grand nombre de problèmes d'optimisation combinatoire **NP**-difficiles [27, 28, 116, 146].

Rappelons que ce type de méthodes repose sur la notion de voisinage : noté $\mathcal{N}(s)$, le voisinage d'une solution s est l'ensemble des solutions réalisables pouvant être obtenues à partir d'une transformation, ou mouvement, de s. Dans un algorithme élémentaire de recherche locale, à chaque étape, on cherche parmi les voisins de la solution courante, une solution de meilleure qualité. Si une telle solution existe, alors elle se substitue à la solution courante. Le processus se poursuit jusqu'à ce qu'on ne puisse plus trouver de meilleure solution. La solution courante est alors un *optimum local*. Un tel algorithme est représenté dans le tableau 1.1, page 12.

La qualité des solutions retournées, i.e. les minima locaux, de même que le temps d'exécution de ces algorithmes, dépend de manière cruciale du voisinage. Une des caractéristiques essentielle d'un voisinage est sa taille, c'est-à-dire le nombre de solutions voisines qu'il contient. De manière intuitive, plus le voisinage est grand, moins il y a de minima locaux, et meilleur est leur qualité. Cependant, plus le voisinage est grand, plus le temps requis pour l'explorer à chaque itération est important. Par conséquent, un voisinage plus grand, n'implique pas nécessairement un algorithme plus efficace. L'idéal serait d'avoir des voisinages de taille exponentielle, pour lesquels il soit néanmoins possible de déterminer en temps polynomial la meilleure solution voisine d'une solution quelconque.

Récemment plusieurs voisinages de taille exponentielle, examinables néanmoins en temps polynomial, ont été proposés pour plusieurs problèmes d'optimisation combinatoire : le problème du voyageur de commerce [54, 71, 94, 95, 140] (voir aussi les travaux précurseurs [154] et [155]), le problème de l'affectation quadratique [71] le problème de tournées de véhicule [80], ainsi que des problèmes d'ordonnancement [62, 107, 108]. Le lecteur peut consulter [1] pour une vue d'ensemble de ces résultats.

Ces méthodes d'exploration de voisinages exponentiel s'appuient soit sur la programmation dynamique, soit utilisent des réductions vers des problèmes de la théorie des graphes résolvables en temps polynomial tels que : la recherche d'un plus court chemin dans un graphe ou la recherche d'un couplage de poids minimum.

Nous avons cherché à étendre ce genre de résultats pour une généralisation du problème du voyageur de commerce, à savoir le problème de tournée de véhicules. Nous avons introduit un voisinage exponentiel et montré qu'il pouvait être exploré en temps polynomial à l'aide d'une réduction vers un problème de couplage contraint pouvant être résolu en temps polynomial. Ce travail est exposé dans la section 3.2.

Comme nous l'avons indiqué plus haut, beaucoup de travaux utilisent la programmation dynamique afin d'explorer de manière efficace des voisinages exponentiels. Nous avons cherché à savoir si cette approche, pouvait être étendue et si elle était intéressante du point de vue expérimental, dans le cas d'un problème d'ordonnancement dynamique en optimisation monocritère. Ce travail fait l'objet de la section section 3.3. Enfin, nous avons cherché à étendre l'utilisation de voisinages de taille exponentielle aux problèmes multicritère. La section 3.4.1 expose tout d'abord les adaptations qu'il est nécessaire d'effectuer à l'algorithme de recherche locale, afin qu'il puisse traiter des problèmes multicritères. Comme problème d'étude nous avons choisi le problème du voyageur de commerce bicritère, et la section 3.4 expose notre approche pour ce problème. Dans la section 3.5 nous décrivons brièvement les résultats expérimentaux que nous avons obtenus.

3.2 Le problème de tournée de véhicules

Le problème a été défini dans la section 2.1.2, page 26. Rappelons que chacun des clients demande la même quantité du bien (il s'agit de demandes unitaires).

Nous avons proposé un nouveau voisinage exponentiel pour ce problème, et nous avons montré qu'il pouvait être exploré en temps polynomial en se ramenant à un problème de couplage contraint de poids minimum dans un graphe biparti qui pouvait lui même être résolu à l'aide de la recherche d'un flot de coût minimum dans un graphe.

3.2.1 Un problème de couplage contraint

Soit $G(V_1, V_2, E)$ un graphe biparti. Rappelons qu'un ensemble d'arêtes $M \subseteq E$ est un *couplage* si aucun sommet de V n'est incident à plus d'une arête de M. La taille d'un couplage est son nombre d'arêtes. Si $|V_1| \leq |V_2|$ et chaque sommet $x \in V_1$ est incident avec une arête de M, alors le couplage est dit *complet*. Soit $E_1, E_2, \ldots, E_k$ des sous-ensembles de E, et $r_1, r_2, \ldots, r_k$ des entiers positifs. Le problème du couplage complet *restreint*, noté RCM (*Restricted Complete Matching*) par la suite, consiste à déterminer s'il existe, pour le graphe G, un couplage complet M qui satisfait les contraintes supplémentaires suivantes :

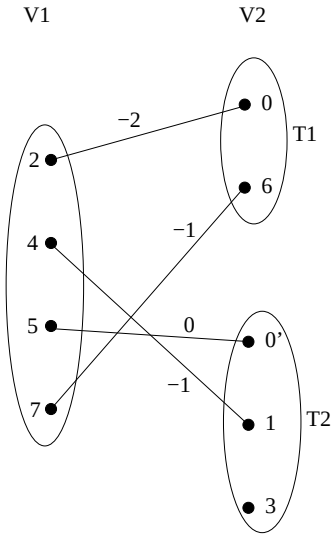
$$|M \cap E_j| \leq r_j \ \forall j \in \{1, \ldots, k\}.$$

Il a été montré que ce problème était **NP**-complet [111].

Nous nous sommes intéressé à un cas particulier de ce problème dans lequel les sous-ensembles $E_1, E_2, \ldots, E_k$ ne sont pas arbitraires mais correspondent à une partition de l'ensemble $V_2 : T_1, T_2, \ldots, T_k$. Plus précisément, pour chaque $j \in \{1, \ldots, k\}$, E_j est l'ensemble des arêtes qui ont une extrémité dans l'ensemble T_j. Le problème noté PRCM (pour *Particular Restricted Complete Matching*) consiste à déterminer s'il existe, pour le graphe G, un couplage complet M qui satisfait les contraintes indiquées plus haut :

$$|M \cap E_j| \leq r_j \ \forall j \in \{1, \ldots, k\}.$$

Dans la version pondérée, notée MIN WPRCM par la suite, chaque arête $[v, w]$ a un poids c_{vw}, et on recherche un couplage satisfaisant les contraintes et qui soit de poids minimum. La figure 3.1 illustre ce problème.



	0	6	0'	1	3
2	-2	4	3	1	7
4	4	1	6	-1	1
5	0	4	0	4	7
7	1	-1	3	4	4

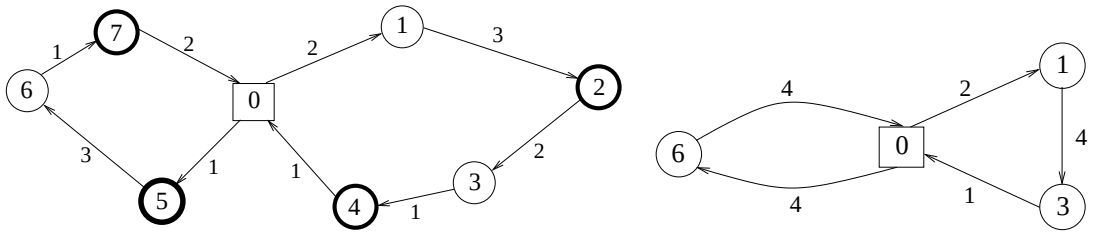
FIG. 3.1 – Une instance du problème MIN WPRCM, avec $V_1 = \{2, 4, 5, 7\}$, $V_2 = \{0, 0', 1, 3, 6\}$, $T_1 = \{0, 6\}$, $T_2 = \{0', 1, 3\}$ et $r_1 = r_2 = 2$. *Gauche* : Une solution optimale de coût -4. *Droite* : Matrice des distances.

3.2.2 Le voisinage exponentiel multi-insertion

Soit S une solution initiale arbitraire. Dans cette solution, on choisit de manière aléatoire un sous-ensemble de sommets qui sont considérés comme mobiles. Les autres sommets et le dépôt sont fixes. Dans le voisinage *multi-insertion* que nous avons proposé, une solution voisine de S est une solution S' dans laquelle chaque sommet mobile a été inséré entre deux sommets fixes de la solution initiale S (ces deux sommets peuvent être le dépôt si le seul sommet fixe d'un tour est le dépôt). Remarquons que comme il n'est possible d'insérer qu'au plus un sommet mobile entre deux sommets fixes, le nombre de sommets mobiles doit être inférieur ou égal au nombre de sommets fixes. Remarquons aussi qu'avec ce voisinage, le nombre de tours peut rester constant ou diminuer, mais ne peut pas augmenter. Ce voisinage peut être considéré comme étant une extension de celui proposé par Gutin [94] pour le problème du voyageur de commerce.

Exemple 3 *La figure 3.2 montre une solution initiale de coût 16 (*Gauche*) ainsi que la matrice des distances. Les sommets fixes forment un nouveau graphe (*Droite*), dans lequel il est possible d'insérer au plus un sommet mobile entre deux sommets (fixes). La figure 3.3 montre une solution*

*voisine de la solution initiale représentée à la figure 3.2 (*Gauche*). Le coût de cette dernière solution est 10. On peut remarquer que l'orientation de chaque tour est préservée lors de la construction d'une solution voisine.*



	0	1	2	3	4	5	6	7
0	0	2	1	4	4	1	4	2
1	4	0	3	4	1	4	4	4
2	4	4	0	2	4	4	1	4
3	1	4	4	0	1	4	4	4
4	1	4	4	2	0	4	4	4
5	4	1	4	4	4	0	3	4
6	4	4	4	4	4	4	0	1
7	2	4	4	4	4	4	4	0

FIG. 3.2 – *Gauche* : Une solution pour le problème VRP. Les sommets encerclés en gras sont les sommets mobiles. *Droite* : Le graphe obtenu lorsque les sommets mobiles ont été enlevés.

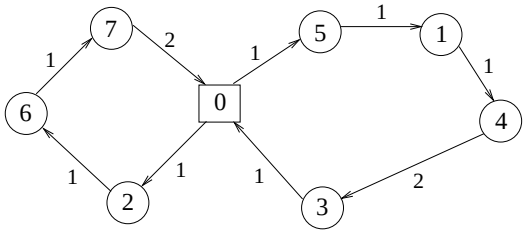


FIG. 3.3 – Une solution de coût 11 pour le problème de tournées de véhicules.

Remarquons que ce voisinage n'est pas de taille constante. Autrement dit, le nombre de solutions voisines d'une solution n'est pas une constante, mais dépend de la structure de la solution initiale (nombre de tournées, capacité des véhicules, choix des sommets mobiles). Cependant, en considérant une solution initiale formée d'un seul tour, et $n/2$ sommets mobiles sur un total de n sommets, il est facile de montrer le théorème suivant :

Théorème 2 *[23] Le voisinage multi-insertion est de taille exponentielle.*

Afin d'utiliser une recherche locale avec la méthode de la plus profonde descente, il est nécessaire de connaître à chaque étape de la recherche locale la meilleure solution voisine de la solution courante. Comme la taille du voisinage multi-insertion est exponentielle il est nécessaire de développer des techniques afin de l'explorer de manière efficace, c'est-à-dire en temps polynomial.

3.2.3 Réduction vers le problème MIN WPRCM

On note D la matrice des distances, avec $d_{i,j}$ la distance entre les clients i et j. La capacité du véhicule est notée C, la solution initiale S est représentée à l'aide d'un graphe $G(V, E)$ et les sommets mobiles ont été choisis. On note k le nombre de tournées dans la solution S. Dans S, chaque sommet fixe u a un successeur $s(u)$ qui est le prochain sommet fixe qui le suit dans le tour. On pose $s(u) = u$ si u est le dépôt et si u est le seul sommet fixe d'une tournée. Par exemple, pour la solution S représentée à la figure 3.2 *Gauche*, le successeur $s(1)$ du sommet 1 est le sommet 3. Il s'agit du successeur direct de 1 une fois que les sommets mobiles ont été enlevés (voir la figure 3.2 *Droite*).

La recherche d'une meilleure solution voisine de S peut se formuler à l'aide d'un problème de couplage de la manière suivante. Soit $G'(V', E')$ un graphe complet biparti tel que $V' = V_1 \cup V_2$, avec V_1 l'ensemble des sommets mobiles et V_2 l'ensemble des sommets fixes plus k copies du dépôt. Nous avons $E' = \{[u, v] \mid u \in V_1, v \in V_2\}$. Soit $u \in V_1$ et $v \in V_2$, le poids de l'arête $[u, v]$ est égal à $d_{v,u} + d_{u,s(v)} - d_{v,s(v)}$. Si une arête $[u, v]$ appartient au couplage recherché, cela signifie que le sommet u est inséré entre les sommets v et $s(v)$. Soit $\{T_1, T_2, \ldots, T_k\}$ une partition de V_2 telle que pour chaque $i \in \{1, \ldots, k\}$, les sommets de T_i sont les sommets fixes de la i-ième tournée de S plus une copie du dépôt. Soit $r_i = C - |T_i| + 1$. On recherche dans G' un couplage complet de poids minimum M tel que le nombre d'arêtes dans $M \cap T_i$ soit plus petit ou égal à r_i, pour chaque $i \in \{1, \ldots, k\}$.

La figure 3.1 montre l'instance correspondante du problème MIN WPRCM pour la solution représentée à la figure 3.2. La matrice des distances de G' est donnée à la figure 3.1 *Droite*. Dans la figure 3.1 *Gauche*, seules les 4 arêtes faisant partie du couplage optimal, de poids -4, sont indiquées. Grâce à ce couplage on peut obtenir la meilleure solution voisine de la solution initiale représentée à la figure 3.2. En partant de cette solution initiale, il suffit d'insérer dans la première tournée le sommet 2 entre les sommets 0 et 6 ainsi que le sommet 7 entre les sommets 6 et 0, et d'insérer dans la seconde tournée le sommet 4 entre les sommets 1 et 3 ainsi que le sommet 4 entre les sommets 1 et 3 ainsi que le sommet 5 entre le dépôt (0' est une copie du dépôt) et le sommet 1. On obtient ainsi la solution représentée à la figure 3.3.

Théorème 3 *[23] Soit S une solution réalisable pour une instance du problème de tournées de véhicules avec des demandes unitaires et soit ξ un sous-ensemble de sommets de S qualifiés de mobiles. Soit G' le graphe complet biparti défini plus haut, et soit M un couplage optimal pour le problème* MIN WPRCM *sur le graphe G'. Ce couplage permet d'obtenir, pour le voisinage multi-insertion, la meilleure solution voisine S' de la solution S. La solution S' est obtenue à partir de la solution S en enlevant les sommets mobiles et, à chaque fois qu'une arête $[u, v]$ appartient au couplage M, en insérant le sommet mobile u entre les sommets fixes v et $s(v)$.*

3.2.4 Polynomialité du problème MIN WPRCM

Le problème MIN WPRCM peut s'exprimer à l'aide du programme linéaire en nombres entiers suivant :

$$\text{Minimiser} \sum_{[v,w] \in E} c_{vw} x_{vw} \tag{3.1}$$

sous les contraintes :

$$\sum_{w \in V, [v,w] \in E} x_{vw} = 1, \quad \forall v \in V_1 \tag{3.2}$$

$$\sum_{w \in V, [v,w] \in E} x_{vw} \leq 1, \quad \forall v \in V_2 \tag{3.3}$$

$$\sum_{[v,w] \in E_i} x_{vw} \leq r_i, \quad \forall i \in \{1, \ldots, k\} \tag{3.4}$$

$$x_{vw} \in \{0, 1\}, \quad \forall [v, w] \in E. \tag{3.5}$$

On a $x_{vw} = 1$ si l'arête $[v, w]$ appartient au couplage recherché et $x_{vw} = 0$ sinon. Nous avons montré que la matrice des contraintes de ce programme linéaire en nombres entiers était totalement unimodulaire [24]. Par conséquent le problème MIN WPRCM peut être résolu en temps polynomial.

Il existe cependant une preuve plus directe du caractère polynomial de ce problème. Il peut en effet être résolu à l'aide du problème de la recherche d'un flot maximum de coût minimum dans un graphe. On construit un graphe orienté $G' = (V', E')$ dans lequel chaque arc (u, v) a un coût $w_{u,v}$ est une capacité $Cap_{u,v}$. Les sommets V' sont les suivants : la source, les sommets de V_1 et V_2, ainsi que k sommets $\{a_1, \ldots, a_k\}$, et le puits. La source a un arc sortant vers chaque sommet de V_1. Il existe un arc depuis un sommet $u \in V_1$ vers un sommet $v \in V_2$ si et seulement si il existe une arête $[u, v]$ dans E. Chaque sommet $u \in T_i$ a un arc sortant vers le sommet a_i, et il existe un arc entre chacun des sommets $\{a_1, \ldots, a_k\}$ et le puits. Le coût de chaque arc est 0, sauf pour les arcs entre V_1 et V_2 : pour $u \in V_1$ et $v \in V_2$, le coût de l'arc (u, v) est égal au coût de l'arête $[u, v]$ dans l'instance du problème MIN WPRCM, i.e. $w_{u,v} = c_{u,v}$. La capacité de chaque arc est égale à 1, sauf pour les arcs entre les sommets $\{a_1, \ldots, a_k\}$ et le puits : la capacité de l'arc entre a_i et le puits est égale à r_i. Enfin, $|V_1|$ unités de flot partent de la source.

Exemple 4 *La figure 3.4 montre l'instance du problème de flot maximum de coût minimum qui correspond à l'instance du problème* MIN WPRCM *représentée à la figure 3.1. La capacité de chaque arc est égale à Cap, où 1 quand Cap n'est pas indiqué, et le coût de chaque arc est égal à 0, sauf pour les arcs entre les sommets de V_1 et les sommets de V_2.*

Le problème du flot maximum de coût minimum peut être résolu en temps polynomial [133], et à partir d'une solution optimale il est possible de récupérer très simplement une solution optimale pour le problème MIN WPRCM. En effet, une arête entre un sommet de V_1 et un sommet de V_2 appartient au couplage restreint de poids minimum si et seulement si il y a une unité de flot qui traverse cette arête dans G'.

À partir de cette réduction, nous obtenons donc le corollaire suivant :

Corollaire 1 *[23] Pour le problème de tournées de véhicules avec des demandes unitaires et pour le voisinage multi-insertion, il est possible de trouver en temps polynomial la meilleure solution voisine d'une solution courante.*

On peut remarquer qu'il est possible d'utiliser le voisinage multi-insertion dans le cadre d'une recherche tabou, en donnant un coût important à certaines arêtes afin d'empêcher un sommet mobile d'être inséré à la même position que dans la solution initiale.

À l'aide d'une réduction qui utilise le problème PARTITION [86], il est possible de montrer que dans un cas plus général, le voisinage multi-insertion n'est plus explorable en temps polynomial.

Corollaire 2 *[23] Il est NP-difficile de trouver la meilleure solution du voisinage multi-insertion si les demandes des clients sont arbitraires, et ce même si les distances entre les clients sont toutes identiques.*

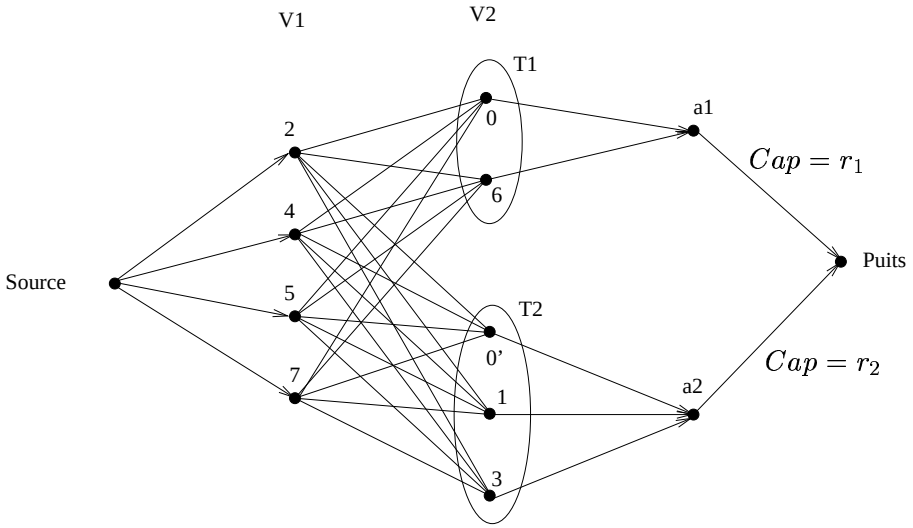


FIG. 3.4 – L'instance du problème de flot correspondant à l'instance du problème de couplage de la figure 3.1.

3.3 Le problème d'ordonnancement $1 \mid p_j^t, idleness \mid \sum_j w_j T_j$

Rappelons que ce problème a été défini dans la section 2.2.3. Congram et al. ont considéré le problème d'ordonnancement classique $1||\sum w_j T_j$ et ont proposé un voisinage de taille exponentielle, appelé *ds-swap* (pour *dynasearch-swap*), pour celui-ci [62]. Les résultats expérimentaux qu'ils ont obtenu furent mitigé. Nous avons cherché à savoir si cette approche pouvait être appliquée pour une version dynamique de ce problème d'ordonnancement. Nous pensions en effet que l'efficacité d'un tel voisinage exponentiel serait plus importante dans un cadre dynamique.

Le voisinage que nous avons utilisé est basé sur le voisinage *swap*. Ce choix fut motivé par le fait que le voisinage *swap* est connu pour donner les meilleurs résultats, par rapport à d'autres voisinages existants, pour le problème $1 \,||\, \sum_j w_j T_j$ [4, 65], et donc probablement aussi pour la généralisation que nous avons considérée.

Une solution (un ordonnancement) est représentée par une permutation $\sigma = (\sigma(1), \ldots, \sigma(n))$ de l'ensemble des éléments $\{1, 2, \ldots, n\}$. Cette permutation indique que la tâche $\sigma(j)$ est la j-ième tâche à être ordonnancée. Étant donnée une permutation $\sigma = (\sigma(1), \ldots, \sigma(i), \ldots, \sigma(j) \ldots \sigma(n))$ le voisinage *swap* consiste en toutes les permutations de la forme $\sigma' = (\sigma(1), \ldots, \sigma(j), \ldots, \sigma(i) \ldots \sigma(n))$, avec $1 \leq i < j \leq n$, qui peuvent être obtenues à partir de σ en échangeant deux tâches. Deux mouvements, l'un qui échange les tâches $\sigma(i)$ et $\sigma(j)$, et l'autre qui échange les tâches $\sigma(k)$ et $\sigma(l)$, sont dits *indépendants* si $\max\{i, j\} < \min\{k, l\}$ ou $\max\{k, l\} < \min\{i, j\}$. Le voisinage *ds-swap*, introduit dans [62], est constitué de toutes les solutions qui peuvent être obtenues en appliquant une suite de mouvements *swap* deux à deux indépendants.

Exemple 5 *Étant donnée la permutation* $\sigma = (1\ 3\ 4\ 6\ 2\ 5\ 8\ 10\ 9\ 7\ 12\ 14\ 13\ 11)$, *il est possible d'appliquer les 3 mouvements indépendants indiqués pour obtenir la permutation voisine* $\sigma' = (1\ 6\ 4\ 3\ 2\ 5\ 7\ 10\ 9\ 8\ 12\ 14\ 11\ 13)$.

Il est facile de voir que ce voisinage est de taille exponentielle, à savoir : $2^{n-1} - 1$. Afin d'explorer ce voisinage de manière efficace, c'est-à-dire trouver la meilleure solution voisine d'une permutation

Entrée :	un voisinage $\mathcal{N}$
Étape 1 :	Soit s_{ini} une solution réalisable quelconque
Étape 2 :	$Q = \{s_{ini}\}$
Étape 3 :	$S = \emptyset$
Étape 4 :	**Tant que** l'image de S est différente de l'image de Q, **faire**
	$S = Q$
	$R = \bigcup_{s \in Q} \mathcal{N}(s)$
	$Q = MinUndom(Q \cup R)$
	Fin Tant que
Étape 5 :	Retourner S

TAB. 3.1 – L'algorithme de recherche locale multicritère $\mathbf{BLS}_{\mathcal{N}}$.

parmi les $2^{n-1} - 1$ solutions candidates, nous avons utilisé la programmation dynamique.

La technique utilisée est similaire à celle exposée plus loin, dans la section 3.4, pour le problème du voyageur de commerce dans un cadre bicritère. Le lecteur peut se référer à l'article [5] concernant les détails de l'implantation. En supposant que les durées d'exécution des tâches sont bornées par une constante p_{max}, i.e. $0 \leq p^t_{\sigma(j)} \leq p_{max}$, $\forall j, t$, nous avons obtenu un algorithme de complexité $\mathcal{O}(n^4 p_{max})$ en temps, et de complexité $\mathcal{O}(n^2 p_{max})$ en mémoire.

3.4 Le problème du voyageur de commerce bicritère

Le problème a été défini dans la section 2.1.1, page 25.

3.4.1 Recherche locale multicritère

Moyennant quelques modifications, la recherche locale peut être adaptée aux problèmes multicritères [136, 165].

Pour un ensemble S de solutions, on note $Undom(S)$ le sous-ensemble de S composé des solutions non dominées : $Undom(S) = \{s \in S \mid \nexists s' \in S \text{ telle que } s' < s\}$. De plus, on note $MinUndom(S)$ tout sous-ensemble de $Undom(S)$ de cardinal maximal tel qu'aucune paire de solutions appartenant à $MinUndom(S)$ n'ait la même image dans l'espace des objectifs.

Le schéma type de la recherche locale multicritère est donné dans le tableau 3.1. Cet algorithme permet de construire un ensemble de solutions *localement Pareto optimal* [136], à savoir un ensemble S tel qu'aucune solution appartenant à $\bigcup_{s \in S} \mathcal{N}(s)$ ne puisse enrichir S. Graphiquement (voir la figure 3.5), on part d'une solution s_{ini} puis étape par étape on construit une courbe de Pareto approchée qui converge vers un ensemble de solutions localement Pareto optimal. Par la suite on appelle $\mathbf{BLS}_{\mathcal{N}}$ (**B**icriteria **L**ocal **S**earch) cet algorithme utilisé dans un cadre bicritère.

L'algorithme $\mathbf{BLS}_{\mathcal{N}}$ souffre cependant d'un défaut qu'on ne rencontre pas pour la recherche locale monocritère. L'ensemble Q des solutions incomparables peut grossir de façon exponentielle. Afin de conserver un algorithme polynomial, on peut utiliser un procédé qui restreint l'ensemble Q à un échantillon. Un moyen d'obtenir cet échantillon est d'utiliser une grille géométrique qui, si on restreint le nombre de solutions par boîte, permet de garantir qu'à tout moment le nombre de solutions est polynomial (voir la figure 3.6). Les valeurs atteignables sur chacun des critères sont divisées en intervalles géométriques de raison $(1+\varepsilon)$ où $\varepsilon > 0$ est un paramètre. Les valeurs $\alpha > 0$ et $\beta > 0$ sont des bornes inférieures sur le premier critère et le second critère respectivement. Si Q est

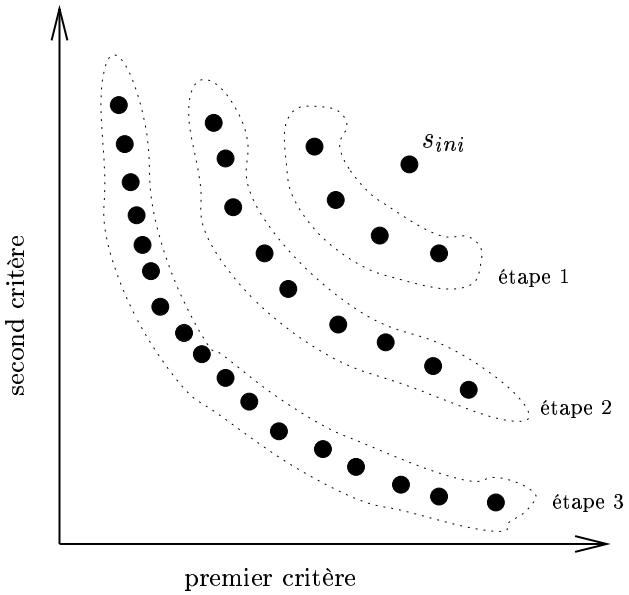


FIG. 3.5 – Évolution de la recherche locale bicritère où étape par étape on converge vers un ensemble localement Pareto optimal.

un ensemble de solutions incomparables, on note $[Q]_\varepsilon \subseteq Q$ un ensemble qui contient au maximum une solution par boîte de la grille. Par la suite, lorsque l'on évoque $[Q]_\varepsilon$, on suppose que l'on est muni d'une règle déterministe qui permet de choisir la solution sélectionnée dans une boîte. On peut alors remarquer que $[Q]_\varepsilon$ est une ε-approximation de Q car pour tout tour $\sigma \in Q$ il existe nécessairement dans $[Q]_\varepsilon$ un tour σ' tel que $\vec{D}(\sigma') \leq (1+\varepsilon)\vec{D}(\sigma)$. De plus, le nombre de solutions dans $[Q]_\varepsilon$ est toujours polynomial en n [135].

Nous avons donc considéré une seconde version de la recherche locale bicritère, notée $\mathbf{RBLS}_{\mathcal{N}}$ (**R** pour *rounded*) où une grille est utilisée afin de contrôler à chaque étape le nombre de solutions générées (voir le tableau 3.2).

3.4.2 Le voisinage *ds-2-opt*

Pour le TSP monocritère, Congram, Potts et van de Velde [62, 140] ont proposé la méthode *Dynasearch* où un programme dynamique est utilisé afin d'explorer de manière efficace un voisinage de taille exponentielle. Les résultats expérimentaux ont montré que malgré un temps d'exécution plus grand lors de chaque étape de la recherche locale, la qualité des résultats obtenus était dans son ensemble compétitive avec les voisinages classiques [5, 140]. Nous avons donc cherché à savoir s'il en était de même dans un cadre bicritère. Au delà de la construction de courbes de Pareto approchées, c'est ce point qui a motivé notre travail.

Le voisinage *ds-2-opt* (*ds* pour dynasearch) noté $\mathcal{N}_{dyna}(\sigma)$ et introduit dans [140] pour le TSP monocritère est l'ensemble des tours qui peuvent être obtenus à partir de σ par une série de mouvements 2-opt dits *indépendants*. Deux mouvements 2-opt sont dits *indépendants* si les arêtes impliquées ne se croisent pas (voir la figure 3.7). On peut facilement voir que la taille du voisinage ds-2-opt est exponentielle (au moins $2^{n/2}$, où n est le nombre de villes). Néanmoins, l'utilisation de la *programmation dynamique* comme dans les articles [62, 140] montre qu'il est possible d'explorer ce voisinage en temps polynomial.

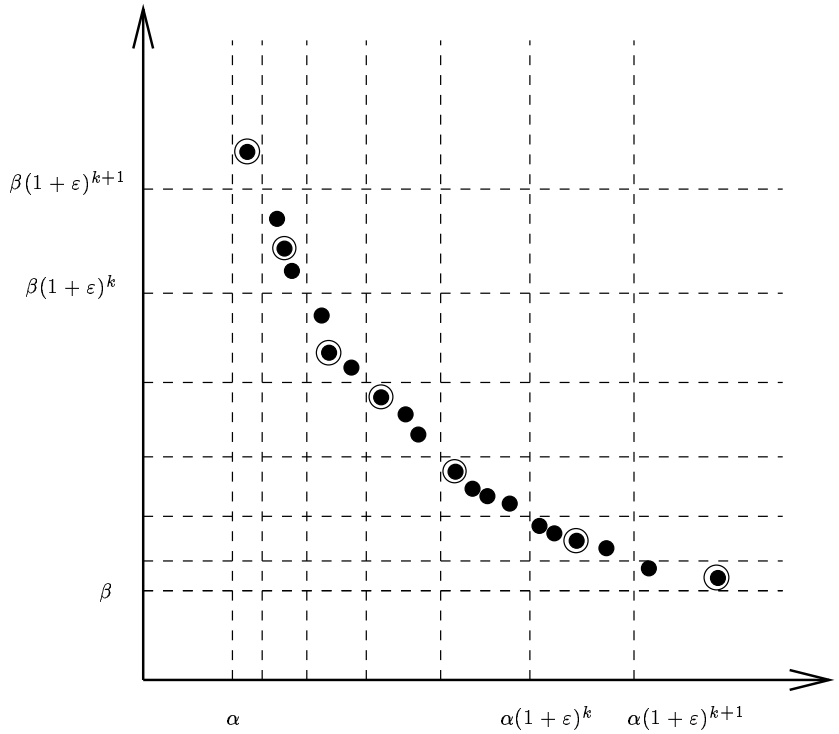
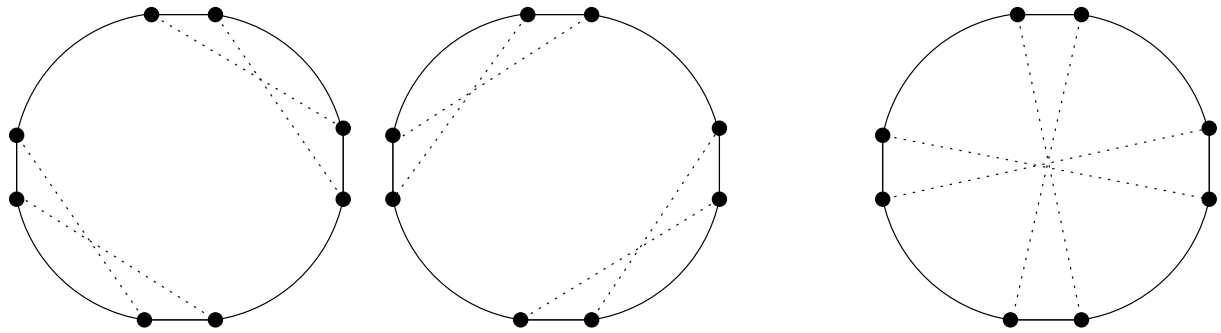


FIG. 3.6 – Subdivision géométrique de l'espace. On ne conserve qu'une seule solution par boîte (celles qui sont entourées par exemple).

Entrées :	Un voisinage $\mathcal{N}$ et $\varepsilon > 0$
Étape 1 :	Soit s_{ini} une solution réalisable quelconque
Étape 2 :	$Q = \{s_{ini}\}$
Étape 3 :	$S = \emptyset$
Étape 4 :	**Tant que** l'image de S est différente de l'image de Q, **faire**
	$S = Q$
	$R = \emptyset$
	Tant que $Q \neq \emptyset$, **faire**
	Prendre une solution $s \in Q$
	$Q = Q \backslash \{s\}$
	$R = [MinUndom(R \cup \{s\} \cup \mathcal{N}(s))]_{\varepsilon}$
	Fin Tant que
	$Q = R$
	Fin Tant que
Étape 5 :	Retourner S

TAB. 3.2 – **RBLS** : Recherche locale bicritère arrondie.



mouvements 2-opt indépendants mouvements 2-opt dépendants

FIG. 3.7 – Deux mouvements 2-opt sont dits indépendants si les nouvelles arêtes insérées ne se croisent pas.

Nous avons étendu cette approche au cas du TSP bicritère de la manière suivante. Soit σ un tour tel que $\sigma = \sigma(1)\sigma(2)\ldots\sigma(n)$. On suppose que $\sigma(n+1) = \sigma(1)$. On cherche à déterminer $MinUndom(\mathcal{N}_{dyna}(\sigma))$. On note $F(i,\sigma)$, pour $i \in \{1,\ldots,n+1\}$, l'ensemble des vecteurs distances des chemins partant de $\sigma(1)$, terminant en $\sigma(i)$ tout en passant par tous les sommets $\sigma(2)\ldots\sigma(i-1)$. L'initialisation de la programmation dynamique est obtenue en posant* :

$$\begin{aligned} F(1,\sigma) &= \{(0,0)\} \\ F(2,\sigma) &= \{\vec{d}([\sigma(1),\sigma(2)])\} \\ F(3,\sigma) &= \{\vec{d}([\sigma(1),\sigma(2)]) + \vec{d}([\sigma(2),\sigma(3)])\} \end{aligned}$$

On suppose à présent que l'on a déjà calculé $F(j,\sigma)$ pour $1 \leq j \leq i$ et que l'on se penche sur le $(i+1)$-ème sommet. Une première façon de construire un chemin appartenant à $F(i+1,\sigma)$ est d'ajouter l'arête $[\sigma(i),\sigma(i+1)]$ à chaque chemin de $F(i,\sigma)$ comme le montre la figure 3.8. On obtient un ensemble de chemins de vecteurs distances

$$A = \{(x,y) + \vec{d}([\sigma(i),\sigma(i+1)] \mid (x,y) \in F(j,\sigma)\}.$$

Une deuxième alternative est illustrée en figure 3.9. On obtient un ensemble de chemins de coûts

$$B = \bigcup_{j=1}^{i-2} \{(x,y) + \vec{d}([\sigma(j),\sigma(i)]) + \mathcal{D}_{j+1}^{i} + \vec{d}([\sigma(j+1),\sigma(i+1)]) \mid (x,y) \in F(i,\sigma)\}$$

où

$$\mathcal{D}_{j+1}^{i} = \sum_{k=j+1}^{i-1} \vec{d}([\sigma(k),\sigma(k+1)]).$$

Au final, on a $F(i+1,\sigma) = MinUndom(A \cup B)$ pour $i = 3,\ldots,n$ et $F(n+1,\sigma)$ contient l'ensemble des vecteurs distances des meilleurs compromis appartenant à $\mathcal{N}_{dyna}(\sigma)$.

Dans le cas monocritère, chaque $F(i,\sigma)$ contient exactement une valeur alors que dans le cas bicritère ce sont plusieurs vecteurs qui sont potentiellement compris. Par conséquent ce programme dynamique produit potentiellement un nombre exponentiel de solutions. On peut cependant remédier à ce problème en employant une grille, technique similaire à celle utilisée pour l'algorithme

*Les sommes entre vecteurs sont supposées être coordonnée par coordonnée : $(a,b) + (c,d) = (a+c, b+d)$.

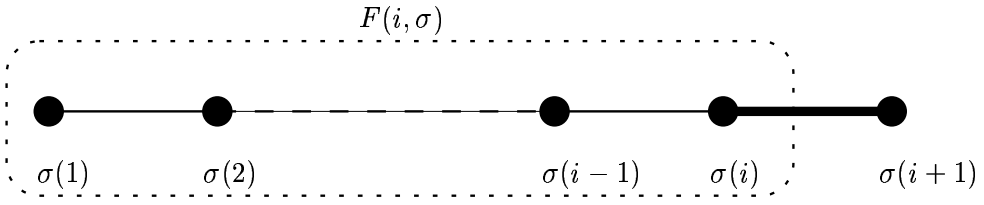


FIG. 3.8 – Première alternative : $F(i+1,\sigma) = F(i,\sigma) + \vec{d}([\sigma(i),\sigma(i+1)])$.

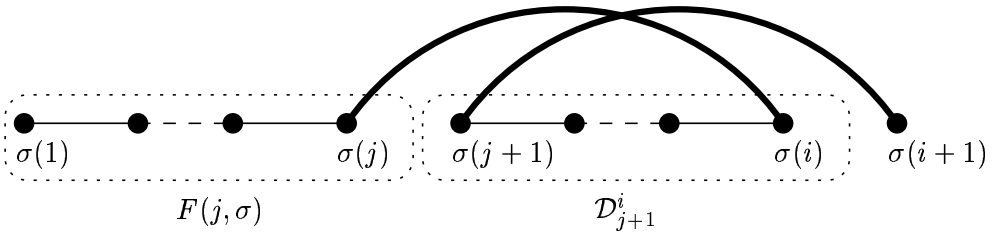


FIG. 3.9 – Seconde alternative : $F(i+1,\sigma) = F(j,\sigma) + \vec{d}([\sigma(j),\sigma(i)]) + \mathcal{D}^i_{j+1} + \vec{d}([\sigma(j+1),\sigma(i+1)])$

général de recherche local bicritère. L'initialisation de la programmation dynamique reste alors la même pour $F(1,\sigma)$, $F(2,\sigma)$ et $F(3,\sigma)$ mais $F(i+1,\sigma) = MinUndom([A \cup B]_\varepsilon)$ pour $i = 3,\ldots,n$.

3.5 Résultats expérimentaux

3.5.1 Le problème d'ordonnancement $1 \mid p_j^t, idleness \mid \sum_j w_j T_j$

Afin de comparer les deux voisinages *swap* et *ds-swap*, nous avons utilisé la recherche locale multi-départ. L'algorithme $MLS_{\mathcal{N}}(nb)$ (pour *multi-start local search*), effectue nb recherches locales avec le voisinage $\mathcal{N}$ et retourne, parmi les nb optima locaux trouvés, le meilleur. Nous avons ainsi comparé $MLS_{ds-swap}(10)$ avec $MLS_{swap}(x)$, où x est un entier tel que le temps d'exécution des deux algorithmes soit à peu près le même ou légèrement à l'avantage de $MLS_{swap}(x)$. Les deux algorithmes peuvent ainsi être comparés de manière équitable.

Nous avons généré des instances aléatoires possédant 30 ou 40 tâches, et de types variés selon le profil des durées d'exécution des tâches et leur date de fin d'achèvement. Ainsi la durée d'exécution p_j^t d'une tâche j peut par exemple être aléatoire, croissante ou décroissante en fonction de la date t. Les dates de fin d'achèvement ont été générées en suivant la procédure classique qui existe pour le problème $1||\sum_j w_j T_j$, ce qui permet de générer des instances plus ou moins difficiles et contraintes selon que la vaste majorité des tâches est ou n'est pas en retard dans toute solution (même celle optimale). Afin d'avoir des résultats en moyenne, chaque algorithme a été exécuté 10 fois sur chacune des instances d'un même profil. Les expériences ont été réalisées sur un Pentium III biprocesseur à 500Mhz.

Dans la grande majorité des cas, $MLS_{ds-swap}(10)$ donne de meilleurs résultats que $MLS_{swap}(x)$ alors même que x est souvent compris entre 200 et 400. Le gain apporté par le voisinage *ds-swap* est le plus fort pour les instances les moins structurées, c'est-à-dire celles pour lesquelles les durées d'exécution des tâches ne suivent aucune loi. On peut remarquer aussi que la différence de gain diminue pour les instances qui sont très peu contraintes, où qui sont au contraire très contraintes. Il

s'agit là en effet d'instances qui sont ou trop faciles, ou trop difficiles à résoudre, et pour lesquelles les algorithmes donnent des résultats sensiblement identiques. Pour plus de détails, le lecteur peut consulter l'article [5].

L'efficacité de l'algorithme $MLS_{ds-swap}$ par rapport à l'algorithme MLS_{swap} peut s'expliquer de la manière suivante : en partant d'une solution σ, l'échange d'une tâche $\sigma(i)$ avec la tâche $\sigma(j)$ $(i < j)$ peut conduire à une solution σ' dans laquelle l'échange de deux tâches $\sigma(k)$ et $\sigma(l)$ $(i < j < k < l$ ou $k < l < i < j)$ mène à une solution σ'' ayant un coût plus petit que σ, alors que si on effectuait en premier l'échange des tâches $\sigma(k)$ et $\sigma(l)$ sur la solution σ cela ferait augmenter le coût de la solution. Cette situation ne peut pas exister pour le problème d'ordonnancement statique $1||\sum w_j T_j$.

3.5.2 Le problème du voyageur de commerce bicritère

Nous avons testé plusieurs variantes des algorithmes de recherche locale, toutes désignées par le nom générique suivant : **[R]BLS**$_{vois}$. La présence du **R** placé devant signifie qu'une grille globale est utilisée. Le champs *vois* peut prendre les valeurs *2-opt* ou *ds-2-opt*, et indique le voisinage utilisé. Si *vois=ds-2-opt* alors on utilise une grille à l'intérieur de la programmation dynamique, comme cela est exposé à la fin de la section 3.4.2.

Les expériences ont été menées sur des instances bicritères du TSP construites à partir des instances géométriques de Krolak (`kroA100`, `kroB100`, *etc.*) extraites de la bibliothèque TSPLIB [169]. Nous avons considéré des instances avec 50, 75, 100, 150 et 200 villes. Afin de mettre autant que possible les algorithmes testés sur un même pieds d'égalité, ceux-ci ont été codés avec le même langage (`C`). Le compilateur utilisé est (`gcc`), et les tests ont été effectués sur un PC de type Pentium IV 2Ghz muni de 256 Mo de RAM. Comme dans la section précédente, dans le souci de donner une même durée d'exécution à deux algorithmes, nous avons utilisé la recherche locale multi-départ. Pour une heuristique de recherche locale quelconque **RL**, on note **RL**(k) la procédure qui consiste à exécuter k fois l'algorithme **RL** en partant de k solutions initiales générées aléatoirement puis à retourner la courbe de Pareto approchée issue de la fusion des k courbes approchées obtenues lors des k exécutions. On compare ainsi **RL**(k) à une autre heuristique **RL'**$(\lceil xk \rceil)$ avec un coefficient x tel que le temps d'exécution de ces deux algorithmes soit approximativement le même.

L'utilisation d'un grille devient vite nécessaire, et ce même pour des instances avec 50 villes, car les heuristiques de recherche locale bicritère sans arrondi prennent énormément de temps avant de converger.

Nous avons comparé **RBLS**$_{dyna}$ et **RBLS**$_{2-opt}$ avec différentes tailles de grilles : ε pouvant prendre des valeurs parmi $\{0.01, 0.013333, 0.016666, 0.02\}$. Nous avons utilisé les trois mesures de qualité mentionnées dans la section 1.2.5 page 18. La procédure expérimentale employée est la suivante :

Entrée :	une instance I et un ε
Étape 1 :	Exécuter $\mathbf{RBLS}_{dyna}$ sur I à partir de 50 solutions tirées aléatoirement et mettre les courbes obtenues dans l'archive ARCH1
Étape 2 :	Exécuter $\mathbf{RBLS}_{2-opt}$ sur I à partir de 50 solutions tirées aléatoirement et mettre les courbes obtenues dans l'archive ARCH2
Étape 3 :	Répéter 100 fois : $Q_1 \leftarrow$ fusion de 10 courbes $\in ARCH1$ choisies aléatoirement $Q_2 \leftarrow$ fusion de X courbes $\in ARCH2$ choisies aléatoirement Calculer la valeur de Tchebycheff pour Q_1 et Q_2 Calculer $\mathcal{C}(Q_1, Q_2)$ et $\mathcal{C}(Q_2, Q_1)$ Calculer $\mathcal{D}(Q_1, Q_2)$ et $\mathcal{D}(Q_2, Q_1)$
Étape 4 :	Retourner le minimum, la moyenne, l'écart type et le maximum

Nous avons obtenu avec le voisinage *ds-2-opt* de bien meilleurs résultats qu'avec le voisinage *2-opt*. La supériorité du voisinage *ds-2-opt* est plus claire pour de plus grandes valeurs de ε comme on peut le voir sur la figure 3.10, présentant un exemple de courbes de Pareto approchées obtenues en utilisant les deux algorithmes. Pour de plus petites valeurs de de ε, la différence est moins visuelle mais celle-ci existe si on se réfère aux trois mesures de qualité utilisées. Les trois mesures abondent en effet dans le même sens. La mesure de Tchebycheff est toujours plus grande pour $\mathbf{RBLS}_{dyna}(10)$ que pour $\mathbf{RBLS}_{2-opt}(X)$. De plus on a : $\mathcal{C}(\mathbf{RBLS}_{dyna}(10), \mathbf{RBLS}_{2-opt}(X)) > \mathcal{C}(\mathbf{RBLS}_{2-opt}(X), \mathbf{RBLS}_{dyna}(10))$ et
$\mathcal{D}(\mathbf{RBLS}_{dyna}(10), \mathbf{RBLS}_{2-opt}(X)) > \mathcal{D}(\mathbf{RBLS}_{2-opt}(X), \mathbf{RBLS}_{dyna}(10))$. Cette conclusion reste vraie pour des instances de taille 150 et 200, mais aussi pour de plus grandes exécutions de $\mathbf{RBLS}_{dyna}$ comme par par exemple $\mathbf{RBLS}_{dyna}(20)$ opposé à $\mathbf{RBLS}_{2-opt}(X)$ sur l'instance `kroAB100`.

Étant donné que les mouvements effectués par $\mathbf{RBLS}_{2-opt}$ sont plus petits que les mouvements opérés par $\mathbf{RBLS}_{dyna}$, se pose la question de savoir si l'utilisation d'une même grille pour les deux algorithmes ne pénalise pas $\mathbf{RBLS}_{2-opt}$. En effet, les solutions générées par $\mathbf{RBLS}_{2-opt}$ pourraient être rapidement piégés dans les boîtes de la grille dès le début de l'exécution de l'algorithme. En fait il n'en n'est rien. Par exemple, pour l'instance `kroAB100`, le nombre d'évaluations de voisinage est toujours plus grand pour $\mathbf{RBLS}_{2-opt}$ que pour $\mathbf{RBLS}_{dyna}$. Les mêmes observations peuvent être faites pour les instances `kroAB150` et `kroAB200`.

Les courbes de Pareto approchées que nous avons obtenues sont moins bonnes que celles générées par l'algorithme génétique de Jaszkiewicz [113] pour les mêmes instances. Cependant, pour faire une comparaison équitable, il faudrait pouvoir donner un temps d'exécution comparable aux deux méthodes comme nous l'avons fait dans cette étude. Il est aussi important de noter que Paquete et al. [136], qui ont utilisé la recherche locale sans grille avec entre autres le voisinage *2-opt*, ont mis en évidence que les courbes de Pareto approchées obtenues étaient meilleures que celles précédemment fournies par Jaszkiewicz. Néanmoins, notre but principal n'était pas la construction de meilleures courbes approchées mais la comparaison entre deux voisinages, et particulièrement de déterminer si l'utilisation des voisinages de grande taille peut apporter des améliorations aux métaheuristiques multicritères.

3.6 Conclusion

Il y a certainement un équilibre à trouver entre le bénéfice qu'il y a à utiliser un large voisinage en terme de qualité des optima locaux obtenus, et le temps passé à l'explorer pendant chaque itération de la recherche locale. Nous avons cherché à étudier cette problématique pour différents problèmes

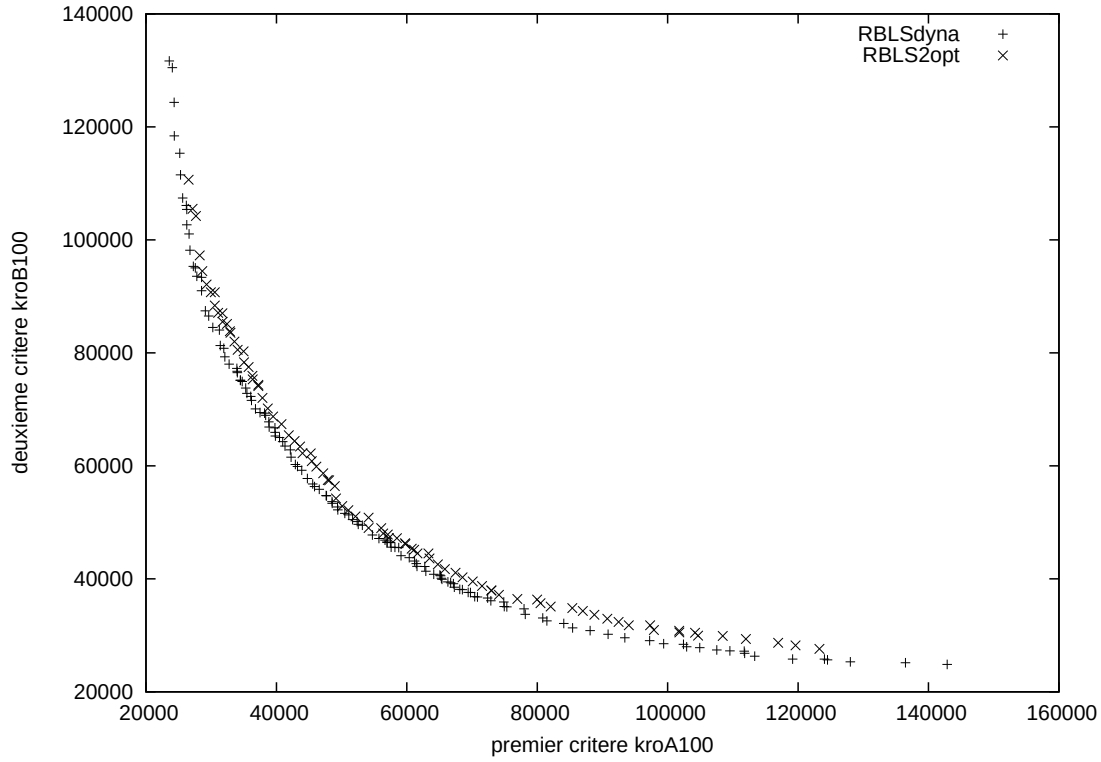


FIG. 3.10 – $RBLS_{dyna}(10)$ contre $RBLS_{2-opt}(9)$ avec $\varepsilon = 0.02$ pour l'instance `kroAB100`.

d'optimisation combinatoire.

Nous avons introduit le voisinage *multi-insertion* de taille exponentielle pour le problème de tournées de véhicules, et montré qu'il pouvait être exploré en temps polynomial en utilisant des techniques de flot. Ce voisinage est une généralisation de celui proposé par Gutin [94] pour le problème du voyageur de commerce. Nous avons montré que l'exploration du voisinage *multi-insertion* est polynomiale si chaque client demande la même quantité de bien (quelle que soit la matrice des distances). Par contre, pour des demandes quelconques, le problème devient **NP**-difficile (même si toutes les distances sont égales).

D'un point de vue pratique, une question intéressante serait d'étudier l'impact du choix des sommets mobiles/fixes sur la qualité de l'optimum local trouvé par la suite. De plus, récemment Ergun et al. [80] ont proposé d'autres voisinages exponentiels pour le problème de tournées de véhicules. La nature des mouvements n'est pas la même que dans le voisinage *multi-insertion*, et la technique utilisée est plutôt de nature heuristique et utilise la recherche de plus courts chemins dans un graphe auxiliaire. Il serait intéressant de comparer expérimentalement ces différents voisinages.

Concernant le problème d'ordonnancement dynamique $1\,|\,p_j^t, idleness\,|\,\sum_j w_j T_j$, nous avons montré que le voisinage exponentiel *ds-swap* pouvait être exploré en temps pseudo-polynomial à l'aide de la programmation dynamique. Les tests expérimentaux que nous avons menés sur différentes familles d'instances aléatoires ont montré la supériorité d'un tel voisinage par rapport au voisinage standard *swap*.

Enfin, nous avons considéré le problème du voyageur de commerce bicritère. Là aussi les résultats expérimentaux montrent que l'utilisation d'un voisinage de taille exponentielle associée à une méthode efficace d'exploration permet de construire des courbes de Pareto approchées de meilleure qualité que celles obtenues avec un voisinage classique.

Étant donné que le voisinage *3-opt* fournit en général de meilleures solutions que le voisinage *2-opt*, on peut se poser la question de la performance de $\mathbf{RBLS}_{dyna}$ face à $\mathbf{RBLS}_{3-opt}$. Dans cette optique, il serait intéressant d'étendre dynasearch afin d'inclure des mouvements différents tels que le $3-opt$, l'insertion ou l'échange. Un essai dans cette direction dans le cadre monocritère est présenté par Ergun et al. [80].

Troisième partie

Algorithmes approchés avec garantie de performance

Chapitre 4

Approche monocritère

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [8, 14, 18].

4.1 Introduction

Nous présentons dans ce chapitre les résultats d'approximation avec garantie de performance que nous avons obtenus pour différents problèmes d'optimisation combinatoire. Les algorithmes présentés seront soit déterministes, soit probabilistes. Nous traitons du problème des requêtes multiples, qui est un problème classique en base de données, dans la section 4.2. La section 4.3 est consacrée au problème de groupage de trafic dans un réseau WDM en étoile, enfin la section 4.4 est consacrée à un problème d'ordonnancement de tâches dans un modèle à base de communications hiérarchiques.

4.2 Problème des requêtes multiples

Ce problème, noté par la suite MQO, a été défini dans la section 2.3.1, page 32. Nous notons par la suite :

- N : le nombre maximum de plans par requête,
- M : le nombre maximum de requêtes dans lesquelles une tâche peut apparaître, et
- k le nombre de requêtes.

Ce problème possède des similitudes avec le problème classique de la couverture d'ensembles. En exploitant ces similitudes, il est possible d'obtenir des résultats d'inapproximabilité et d'obtenir des algorithmes avec garantie de performance.

4.2.1 Résultat d'inapproximabilité

En réduisant le problème de la couverture d'ensemble au problème MQO nous avons obtenu le résultat suivant :

Théorème 4 *[14] Soit $\varepsilon > 0$. S'il existe un algorithme polynomial $(1-\varepsilon)\ln k$-approché pour le problème* MQO, *alors* $\mathbf{NP} \subset TIME(n^{\mathcal{O}(\log\log k)})$.

Nous avons également proposé une réduction polynomiale, préservant le rapport d'approximation, de MQO vers le problème de la couverture d'ensembles, mais uniquement lorsque N est borné par une constante.

4.2.2 Algorithmes gloutons

Nous avons considéré un premier algorithme glouton, dans lequel pour chaque requête on sélectionne le plan de moindre coût, en ne tenant pas compte des tâches déjà sélectionnées. Il est facile de voir que cet algorithme est M-approché et que ce rapport est atteint.

Il est naturel de s'intéresser à une variante gloutonne dans laquelle on tient compte des tâches déjà sélectionnées. On définit ainsi le *coût estimé* d'un plan comme étant le rapport de la somme du coût des tâches le constituant, en ne tenant pas compte du coût des tâches qui font partie des plans qu'on a sélectionné auparavant, sur le nombre de requêtes que permettent de satisfaire ces nouvelles tâches à l'aide des tâches faisant partie des plans déjà sélectionnés. On obtient un algorithme similaire à celui déjà employé pour résoudre le problème de la couverture d'ensembles [170]. En fait il n'est pas difficile de s'apercevoir également que le rapport d'approximation de cet algorithme n'est pas meilleur que le précédent.

4.2.3 Algorithmes basés sur la programmation linéaire

Il est possible de formuler le problème MQO à l'aide d'un programme linéaire en nombres entiers :

$$\text{Minimiser} \quad \sum_{1\leq j\leq r} x_j\, c_j \tag{4.1}$$

$$\begin{aligned} \text{sous les contraintes} \quad & \sum_{n=1}^{n(i)} x_{i,n} = 1 && i = 1\ldots k \\ & \sum_{\{n|t_j\in p_i^n\}} x_{i,n} \leq x_j && \forall (i,j) \text{ t.q. } t_j \text{ apparaît dans } q_i,\ j=1,\ldots,r \text{ et} \\ & && i = 1\ldots k \qquad (4.2) \\ & x_{i,n} \in \{0,1\} && i = 1\ldots k \text{ et } n = 1\ldots n(i) \\ & x_j \in \{0,1\} && j = 1\ldots r \end{aligned}$$

Rappelons qu'on dit d'une tâche qu'elle apparaît dans une requête, si elle fait partie d'un des plans susceptibles de résoudre cette requête. Chaque tâche t_j est représentée par une variable x_j prenant la valeur 1 où 0, selon que la tâche t_j est exécutée où non. Chaque plan p_i^n est représenté par une variable $x_{i,n}$ prenant la valeur 1 où 0, selon que le plan est sélectionné où non. La première série de contraintes assure que pour chaque requête, parmi les plans qui peuvent la résoudre, exactement un seul va être sélectionné. La seconde série de contraintes assure que si un plan p contenant une tâche t est sélectionné, alors la tâche t doit être exécutée. Puisqu'on cherche à minimiser la fonction objectif, une tâche qui n'appartient à aucun plan sélectionné ne sera pas sélectionnée.

Nous considérons la relaxation linéaire de ce programme linéaire (LP) :

1 :	Résoudre LP
2 :	**Pour** $i = 1$ **jusqu'à** k **Faire**
2.1 :	$n' := argmax_{1 \leq n \leq n(i)}\{x_{i,n}\}$
2.2 :	$s_i := n'$
	Fin Pour
3 :	Exécuter s

TAB. 4.1 – L'algorithme D-ARRONDI.

1 :	Résoudre LP
2 :	**Pour** $i = 1$ **jusqu'à** k **Faire**
2.1 :	Sélectionner de manière aléatoire un plan parmi $\{p_i^1, \ldots, p_i^{n(i)}\}$ avec la distribution de probabilité $\{x_{i,1}, \ldots, x_{i,n(i)}\}$, soit p_i^n ce plan
2.2 :	$s_i := n$
	Fin Pour
3 :	Exécuter s

TAB. 4.2 – L'algorithme R-ARRONDI.

$$\text{Minimiser} \quad \sum_{1 \leq j \leq r} x_j \, c_j \tag{4.3}$$

$$\text{sous les contraintes} \quad \sum_{n=1}^{n(i)} x_{i,n} = 1 \qquad i = 1 \ldots k \tag{4.4}$$

$$\sum_{\{n | t_j \in p_i^n\}} x_{i,n} \leq x_j \qquad \forall (i,j) \text{ t.q. } t_j \text{ apparaît dans } q_i, \tag{4.5}$$

$$j = 1 \ldots r \text{ et } i = 1 \ldots k \tag{4.6}$$

$$0 \leq x_{i,n} \leq 1 \qquad i = 1 \ldots k \text{ et } n = 1 \ldots n(i) \tag{4.7}$$

$$0 \leq x_j \leq 1 \qquad j = 1 \ldots r \tag{4.8}$$

Nous avons considéré dans un premier temps un algorithme simple, noté D-ARRONDI (voir le tableau 4.1) inspiré d'un algorithme existant pour le problème de la couverture d'ensembles [100]. Si à l'étape 2.1, il y a plusieurs variables qui atteignent le maximum, alors on en choisit une de manière arbitraire. Toute solution retournée par cet algorithme est nécessairement réalisable, puisque pour chaque requête exactement un plan est sélectionné, et toutes les tâches qui font partie des plans sélectionnés sont exécutées.

Théorème 5 *[14] L'algorithme* D-ARRONDI *est N-approché.*

Il est possible d'améliorer ce rapport en utilisant un algorithme probabiliste. Une idée naturelle et classique afin d'arrondir des variables 0-1 fractionnaires, consiste à les interpréter comme des probabilités. Cela conduit à l'algorithme R-ARRONDI donné dans le tableau 4.2.

Théorème 6 *[14] L'algorithme* R-ARRONDI *est $N\big(1 - (1 - \frac{1}{N})^M\big)$-approché en moyenne.*

Proposition 1 *[14] Les rapports d'approximation mentionnés pour* D-ARRONDI *et* R-ARRONDI *sont atteints.*

1 :	Résoudre LP
2 :	$\mathbf{Pr}[s_i = n] := x_{i,n}$ pour $i = 1 \dots k$ et $n = 1 \dots n(i)$
3 :	**Pour** $i = 1$ **jusqu'à** k **Faire**
3.1 :	Trouver $n \in \{1, \dots, n(i)\}$ qui minimise $\mathbf{E}[C(s) \mid s_i = n]$
3.2 :	$\mathbf{Pr}[s_i = n] := 1$ (et $\mathbf{Pr}[s_i \neq n] := 0$)
	Fin Pour
4 :	Exécuter s

TAB. 4.3 – L'algorithme D2-ARRONDI. $\mathbf{Pr}[x]$ dénote la probabilité de l'évènement x.

Il est possible de dérandomiser l'algorithme R-ARRONDI à l'aide de la méthode classique des espérances conditionnelles [170]. Nous avons ainsi obtenu l'algorithme déterministe D2-ARRONDI représenté dans le tableau 4.3.

Théorème 7 *[14] L'algorithme* D2-ARRONDI *est un algorithme déterministe, polynomial, et* $N(1-(1-\frac{1}{N})^M)$*-approché.*

Proposition 2 *[14] Le rapport* $N(1-(1-\frac{1}{N})^M)$ *est atteint par l'algorithme* D2-ARRONDI *dans le cas* $M = 2$.

Nous conjecturons que le rapport est atteint également pour toutes les valeurs possibles de M et de N.

Il est intéressant de remarquer que le résultat d'approximation que nous avons obtenu avec l'algorithme D2-ARRONDI est le meilleur qu'il est possible d'obtenir à l'aide d'un algorithme basé sur le programme linéaire (4.1) que nous avons utilisé, puisque le saut d'intégralité de ce programme linéaire en nombres entiers est égal à $N(1-(1-\frac{1}{N})^M)$. Rappelons que le saut d'intégralité est le rapport maximum entre OPT et OPT_f sur toutes les instances, avec OPT (respectivement OPT_f) le coût d'une solution optimale pour le programme linéaire en nombres entiers (respectivement pour la relaxation linéaire du programme linéaire en nombres entiers).

Théorème 8 *[14] Le saut d'intégralité du programme linéaire en nombre entiers (4.1) est égal à* $N(1-(1-\frac{1}{N})^M)$.

4.3 Groupage de trafic dans un réseau WDM en étoile

Ce problème a été défini dans la section 2.3.2, page 32.

En utilisant une réduction du problème du sac à dos vers ce problème on montre le résultat suivant :

Théorème 9 *[18] Les problèmes de groupage de trafic dans une étoile passive, dans leur version maximisation ou minimisation, sont* **NP***-complets.*

4.3.1 Un algorithme polynomial exact pour $W = 2$ longueurs d'onde par fibre

Lorsque $W = 2$ on peut remarquer que dans chaque ligne (ou colonne) de la matrice de trafic T dans laquelle il y a au moins trois valeurs non nulles, il est possible de router optiquement au plus

une demande de trafic. Par contre, quand une ligne (ou colonne) contient seulement deux valeurs non nulles, il est éventuellement possible de router ces deux requêtes optiquement.

Nous transformons la matrice T en une matrice T' dans laquelle il est possible de router optiquement au plus une demande de trafic pour chaque ligne, et une demande de trafic pour chaque colonne : si une ligne i de T contient exactement deux valeurs t_{ij} et $t_{ij'}$ non nulles, on crée dans T' deux lignes $i1$ et $i2$ telles que $t'_{i1,j} = t_{ij}$, $t'_{i2,j} = t_{ij'}$. Les autres valeurs dans ces lignes sont égales à 0. De même, si une colonne de T contient exactement deux valeurs t_{ij} et $t_{i'j}$ non nulles, on crée dans T' deux colonnes $j1$ et $j2$ telles que $t'_{i,j1} = t_{ij}$, $t'_{i,j2} = t_{i'j}$. Les autres valeurs dans ces lignes sont égales à 0.

On transforme ensuite la matrice T' en une matrice $M = [m_{ij}]$, dans laquelle nous cherchons un couplage de poids maximum. Si une demande de trafic t'_{ij} ne peut pas être routée optiquement (i.e. $\sum_{k \neq j} t'_{ik} > C$ ou $\sum_{k \neq i} t'_{kj} > C$) alors on pose $m_{ij} = -\infty$. Sinon, on pose $m_{ij} = t'_{ij}$.

Puisqu'il y a dans M au plus une demande de trafic par ligne et une demande de trafic par colonne qui peuvent être optiquement routées, et puisque la demande de trafic m_{ij} est différente de $-\infty$ si et seulement si elle peut éventuellement être routée optiquement, le résultat d'un couplage de poids maximum dans le graphe biparti dont M est la matrice d'adjacence, est une solution optimale pour le problème de groupage de trafic dont la matrice de trafic est T. On obtient ainsi le théorème suivant :

Théorème 10 *[18] Les versions de minimisation et de maximisation du problème de groupage de trafic dans une étoile passive peuvent être résolus en temps polynomial si le nombre de longueurs d'ondes par fibre est égal à deux.*

Exemple 6 *On considère une instance avec 3 émetteurs et 3 récepteurs. La capacité de chaque longueur d'onde est 5 et le nombre de longueurs d'onde par fibre est égal à 2. Soit T la matrice de trafic :*

$$\mathbf{T} = \begin{pmatrix} 4 & 2 & 1 \\ 3 & 3 & 4 \\ 2 & 0 & 5 \end{pmatrix}.$$

Les matrices T' et M créées par l'algorithme sont les suivantes :

$$\mathbf{T'} = \begin{pmatrix} 4 & 2 & 0 & 1 \\ 3 & 0 & 3 & 4 \\ 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix} \quad et \quad \mathbf{M} = \begin{pmatrix} 4 & 2 & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty \\ -\infty & 0 & 0 & -\infty \\ -\infty & 0 & 0 & 5 \end{pmatrix}.$$

Le couplage de poids maximum dans M est constitué de $m_{1,1}$ et $m_{4,4}$, ce qui correspond à $t_{1,1}$ et $t_{3,3}$. Les demandes de trafic qui sont donc routées optiquement dans une solution optimale sont $t_{1,1}$ et $t_{3,3}$. Les autres demandes de trafic sont routées électroniquement.

4.3.2 Algorithmes approchés

Nous avons montré que l'approximation n'était pas conservée entre les versions de minimisation et de maximisation.

Théorème 11 *[18] Il n'est pas possible de déduire un algorithme d'approximation avec garantie de performance pour la version de maximisation (respectivement minimisation) du problème de groupage de trafic dans une étoile passive, à partir d'un algorithme d'approximation avec garantie de performance pour la version de minimisation (respectivement maximisation).*

Algorithme approché pour la version minimisation

Nous commençons par donner une formulation du problème de groupage de trafic dans une étoile passive, dans sa version minimisation, à l'aide d'un programme linéaire en nombres entiers.

Soit $x_{ij} \in \{0,1\}$ une variable qui indique si la quantité de trafic t_{ij} est routée de manière optique ($x_{ij} = 0$) ou si elle subit une conversion électronique au niveau du commutateur ($x_{ij} = 1$). L'objectif est le suivant :

$$\text{Minimiser} \sum_{i \in V_1, j \in V_2} x_{ij} t_{ij} \tag{4.9}$$

Nous avons deux types de contraintes :
les contraintes sur le nombre de longueurs d'ondes :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij} \geq |V_2| - W \tag{4.10}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij} \geq |V_1| - W \tag{4.11}$$

les contraintes sur le trafic :

$$\forall i \in V_1, \sum_{j \in V_2} (1 - x_{ij})(C - t_{ij}) \leq WC - \sum_{j \in V_2} t_{ij} \tag{4.12}$$

$$\forall j \in V_2, \sum_{i \in V_1} (1 - x_{ij})(C - t_{ij}) \leq WC - \sum_{i \in V_1} t_{ij} \tag{4.13}$$

Les inégalités (4.10) (respectivement 4.11) signifient qu'au plus W demandes de trafic par émetteur (respectivement récepteur) peuvent être routées optiquement, car chaque demande de trafic routée optiquement occupe une longueur d'onde à elle seule. Les inégalités (4.12) et (4.13) signifient que la bande passante non utilisée ($C - t_{ij}$) quand t_{ij} est routée optiquement, doit être inférieure ou égale à la bande passante disponible (i.e. la quantité totale de bande passante moins la quantité totale des demandes de trafic). Il est équivalent de dire que la quantité de trafic routée électroniquement doit être inférieure à la capacité d'un lien, C, multiplié par le nombre de liens disponibles (i.e. W moins le nombre de demandes de trafic qui sont routées optiquement).
Les contraintes (4.12) et (4.13) sont équivalentes à :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(C - t_{ij}) \geq C(|V_2| - W) \tag{4.14}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(C - t_{ij}) \geq C(|V_1| - W) \tag{4.15}$$

On peut remarquer que les contraintes sur le trafic impliquent les contraintes sur le nombre de longueurs d'ondes : si on divise par C les contraintes sur le nombre de longueurs d'ondes on obtient :

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(1 - \frac{t_{ij}}{C}) \geq |V_2| - W \tag{4.16}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(1 - \frac{t_{ij}}{C}) \geq |V_1| - W \tag{4.17}$$

Puisque $t_{ij} \leq C$, ces dernières inégalités impliquent les contraintes sur le nombre de longueurs d'ondes (4.10) et (4.11).

Nous pouvons donc formuler notre problème de la façon suivante :

$$\text{Minimiser} \sum_{i \in V_1, j \in V_2} x_{ij} t_{ij} \tag{4.18}$$

sous les contraintes

$$\forall i \in V_1, \sum_{j \in V_2} x_{ij}(C - t_{ij}) \geq C(|V_2| - W) \tag{4.19}$$

$$\forall j \in V_2, \sum_{i \in V_1} x_{ij}(C - t_{ij}) \geq C(|V_1| - W) \tag{4.20}$$

$$\forall i \in V_1, j \in V_2, x_{ij} \in \{0, 1\} \tag{4.21}$$

À partir de cette formulation on peut déduire le résultat suivant :

Théorème 12 *[18] Il existe un algorithme polynomial H_{2C}-approché pour la version minimisation du problème de groupage de trafic dans une étoile passive.*

Rappelons qu'on note H_k le k-ième nombre harmonique, i.e. $H_k = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{k}$. La preuve s'appuie sur une interprétation du programme linéaire en nombres entiers donné ci-dessus en tant que problème de multi-couverture de multi-ensembles contraint pour lequel Rajagopalan et Vazirani ont proposé un algorithme glouton approché [142].

Algorithme approché pour la version maximisation

De la même manière, il est possible de formuler le problème de groupage de trafic dans une étoile passive, sous sa version maximisation, à l'aide d'un programme linéaire en nombres entiers :

$$\text{Maximiser} \sum_{i \in V_1, j \in V_2} y_{ij} t_{ij} \tag{4.22}$$

sous les contraintes :

$$\forall i \in V_1, \sum_{j \in V_2} y_{ij}(C - t_{ij}) \leq CW - \sum_{j \in V_2} t_{ij} \tag{4.23}$$

$$\forall j \in V_2, \sum_{i \in V_1} y_{ij}(C - t_{ij}) \leq CW - \sum_{i \in V_1} t_{ij} \tag{4.24}$$

$$\forall i \in V_1, j \in V_2, y_{ij} \in \{0, 1\} \tag{4.25}$$

Ici y_{ij} indique si la demande de trafic t_{ij} est routée optiquement ($y_{ij} = 1$) ou électroniquement ($y_{ij} = 0$).

Théorème 13 *[18] Il existe un algorithme probabiliste polynomial $(2 + \frac{4}{5})$-approché pour la version maximisation du problème de groupage de trafic dans une étoile passive.*

La preuve s'appuie sur une transformation de ce problème en un problème de couplage avec demandes (*demand matching problem*) [161]. On utilise ensuite un résultat dû à Shepherd et Vetta [161]. Ils ont proposé en effet un algorithme probabiliste $(2 + \frac{4}{5})$-approché pour le problème de couplage avec demandes dans des graphes bipartis.

4.4 Communications hiérarchiques avec un nombre borné de clusters

Ce problème a été défini dans la section 2.2.5, page 31.

4.4.1 Résultat d'inapproximabilité

Nous avons obtenu le résultat suivant :

Théorème 14 *[8] Déterminer si une instance du problème* $P2(P)|biparti;\ (c_{ij},\epsilon_{ij}) = (1,0); p_i = 1|C_{max}$ *admet un ordonnancement de longueur au plus 3, est un problème* **NP**-*complet.*

La preuve utilise une réduction depuis le problème de décision du stable équilibré dans un graphe biparti [152] qui est un problème **NP**-complet. Ce problème, étant donnés un graphe biparti équilibré $B = (X \bigcup Y, E)$ avec $|X| = |Y| = n$ et un entier k, consiste à déterminer s'il existe dans B un stable constitué de k sommets dans X et de k sommets dans Y.

On peut en déduire le résultat d'inapproximabilité suivant :

Corollaire 3 *[8] Pour le problème* $P2(P)|biparti;(c_{ij},\epsilon_{ij}) = (1,0); p_i = 1|C_{max}$*, il n'existe pas d'algorithme polynomial avec un rapport d'approximation plus petit que* $4/3$*, à moins que* **P**=**NP**.

4.4.2 Un algorithme polynomial lorsque $C_{max} = 2$

En utilisant la programmation dynamique et une généralisation du problème d'ordonnancement $P2||C_{max}$, nous avons pu monter le théorème suivant :

Théorème 15 *[8] Il est possible, en temps polynomial, de déterminer si une instance du problème* $Pk(P)|prec;(c_{ij},\epsilon_{ij}) = (1,0); p_i = 1|C_{max}$ *admet un ordonnancement de longueur au plus 2.*

4.5 Conclusion

Concernant le problème des requêtes multiples, parmi les trois algorithmes déterministes que nous avons considérés, l'algorithme D2-ARRONDI est clairement le meilleur en terme de rapport d'approximation puisque $N(1-(1-1/N)^M) \leq \min\{N, M\}$. Concernant la non-approximabilité de ce problème, il serait intéressant d'obtenir une borne inférieure en fonction des paramètres N et M afin de la comparer au rapport d'approximation que nous avons obtenu.

Nous avons montré que le problème de groupage de trafic, dans une étoile passive, était **NP**-complet, à la fois pour les versions de minimisation et de maximisation considérées. Dans le cas particulier où chaque fibre optique ne peut transporter que 2 longueurs d'onde, nous avons obtenu un algorithme exact polynomial. Nous avons montré que les versions minimisation et maximisation de ce problème n'étaient pas équivalentes du point de vue de l'approximation, un algorithme avec garantie de performance pour une version ne permettant pas de déduire un algorithme avec garantie de performance pour l'autre version. En transformant le problème dans sa version minimisation, en un problème de multi-couverture de multi-ensembles contraint [142], nous avons obtenu un algorithme H_{2C}-approché. Pour la version maximisation, à l'aide d'une transformation en un problème de couplage avec demandes dans un graphe biparti [161], nous avons obtenu un algorithme $(2+\frac{4}{5})$-approché.

Puisque les solutions retournées par ces algorithmes sont à la fois des solutions du problème de

maximisation et du problème de minimisation, il serait intéressant de comparer expérimentalement les résultats obtenus avec ces deux algorithmes.

Chapitre 5

Approche point idéal

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [7, 12].

Dans ce chapitre nous considérons un problème d'ordonnancement bicritère sur une machine dans la section 5.1, ainsi que le problème de la coupe maximale bicritère dans la section 5.2. L'approche utilisée ici est celle du point idéal évoquée dans la section 1.2.1.

5.1 Le problème d'ordonnancement $1||\sum C_j, \sum w_j C_j$

Il existe plusieurs résultats concernant les problèmes d'ordonnancement biobjectif avec un critère de type min-max et l'autre de type min-somme. Ainsi Stein et Wein [164] ont considéré le *makespan* et la *somme pondérée des temps de complétude*. Ils ont prouvé qu'il existait toujours un ordonnancement (2,2)-simultané approché. À la suite de ce résultat, Aslam et al. [32] ont amélioré ces bornes, et Rasala et al. [143] ont généralisé ce résultat à d'autres paires de fonctions objectif, la première fonction objectif étant soit le *"maximum flow time"*, le *makespan*, ou le retard maximum, et la seconde fonction objectif étant soit le *"average flow time"*, la somme des temps de complétude, ou le nombre de tâches se terminant à temps.

Nous nous sommes intéressés à deux critères de type min-somme : la somme des temps de complétude de chacune des tâches et la somme pondérée des temps de complétude. Rappelons qu'il est possible d'obtenir très simplement, en temps polynomial, un ordonnancement optimal pour chacun de ces 2 critères (voir la section 2.2.3). Cependant un ordonnancement optimal pour un critère ne l'est généralement pas pour l'autre. Par conséquent, nous avons cherché à obtenir un ordonnancement qui soit (α, β)-simultané approché.

5.1.1 L'algorithme γ-ALG

Soit δ une permutation de l'ensemble des indices des tâches J telle que :

$$p_{\delta(1)} \leq p_{\delta(2)} \leq p_{\delta(3)} \leq \cdots \leq p_{\delta(n)}. \tag{5.1}$$

Soit π une permutation de l'ensemble des indices des tâches J telle que :

$$\frac{w_{\pi(1)}}{p_{\pi(1)}} \geq \frac{w_{\pi(2)}}{p_{\pi(2)}} \geq \frac{w_{\pi(3)}}{p_{\pi(3)}} \geq \ldots \geq \frac{w_{\pi(n)}}{p_{\pi(n)}}. \tag{5.2}$$

Comme nous l'avons vu dans la section 2.2.3, un ordonnancement optimal pour le critère de la somme des temps de complétude, est obtenu en ordonnançant les tâches selon l'ordre SPT donné par la permutation δ. De même, un ordonnancement optimal pour le critère de la somme pondérée des temps de complétude, est obtenu en ordonnançant les tâches selon l'ordre de Smith, donné par la permutation π. Notre approche consiste à fusionner ces 2 ordonnancements optimaux pour chacun des 2 critères, en un seul ordonnancement qui soit simultanément proche des 2 ordonnancements optimaux.

L'algorithme γ-ALG est décrit ci-dessous :

L'ALGORITHME γ-ALG
Entrée : Un paramètre $\gamma > 0$, un ensemble de tâches J.
Sortie : Un ordonnancement σ.

Étape 1 Ordonnancer les tâches de J selon l'ordre π.
Étape 2 Multiplier toutes les dates d'achèvement des tâches par le facteur $(1 + \gamma)$ et déplacer toutes les tâches vers la droite, sans changer leur durée d'exécution, de manière à ce que chaque tâche se termine à cette nouvelle date.
Étape 3 Prendre les tâches selon l'ordre δ, et les placer le plus tôt possible en utilisant les temps d'inactivités. Si un temps d'inactivité n'est pas suffisant pour placer une tâche, la tâche qui la suit est déplacée à gauche, et ainsi de suite, jusqu'à ce qu'il y ait suffisamment de place pour accueillir la tâche en question.

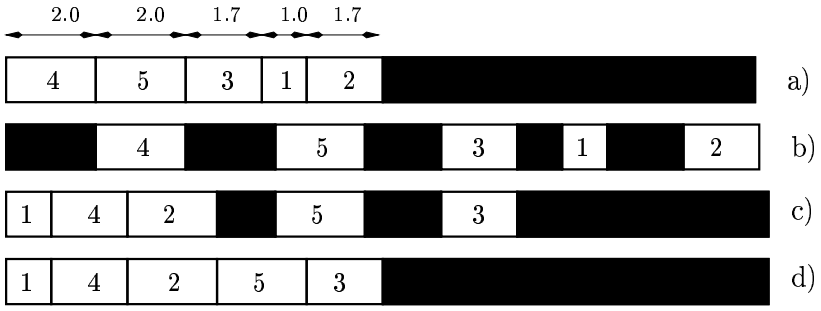


FIG. 5.1 – Une illustration de l'algorithme γ-ALG. Les durées d'exécution des tâches sont : $p_1 = 1$, $p_2 = p_3 = 1.7$ et $p_4 = p_5 = 2$.

Exemple 7 *Dans un premier temps toutes les tâches sont placées sur la machine, selon l'ordre π. Pour une illustration, voir la figure 5.1 a), où les 5 tâches sont placées selon l'ordre $\pi = (4, 5, 3, 1, 2)$. Ensuite, $\gamma > 0$ étant fixé, on multiplie chacun des temps de complétude des tâches par $(1 + \gamma)$ et on crée des temps d'inactivité, comme illustré sur la figure 5.1 b), avec $\gamma = 1$. Ensuite on considère chacune des tâches selon l'ordre δ, et on essaie de les placer le plus tôt possible dans l'ordonnancement. Supposons qu'on ait $\delta = (1, 2, 3, 4, 5)$. La tâche 1 va venir s'insérer en tête de l'ordonnancement, car le temps d'inactivité est suffisant. On passe ensuite à la tâche 2. Celle-ci ne*

Entrée :	$G(V,E)$ et **Al**	
Étape 1 :	Déterminer $(S_1, \overline{S_1})$ avec **Al** *t.q.* $W(S_1, \overline{S_1}) \geq \alpha.OPTW$	
Étape 2 :	Déterminer $(S_2, \overline{S_2})$ avec **Al** *t.q.* $L(S_2, \overline{S_2}) \geq \alpha.OPTL$	
Étape 3 :	Construire $(S_3, \overline{S_3})$ *t.q.* $S_3 = (S_1 \cap S_2) \cup (\overline{S_1} \cap \overline{S_2})$	
Étape 4 :	*Si*	$L(S_1, \overline{S_1}) \geq 0.5\, L(S_2, \overline{S_2})$
	Alors	Retourner $(S_1, \overline{S_1})$
	Sinon	*Si* $W(S_2, \overline{S_2}) \geq 0.5\, W(S_1, \overline{S_1})$
		Alors Retourner $(S_2, \overline{S_2})$
		Sinon Retourner $(S_3, \overline{S_3})$

TAB. 5.1 – L'algorithme **Bi-Approx**.

peut pas venir derrière la tâche 1, car le temps d'inactivité n'est plus suffisant. On déplace alors la tâche 4 vers la gauche, ce qui a pour effet de créer un temps d'inactivité derrière la tâche 4. Ce temps d'inactivité est maintenant suffisant pour accueillir la tâche 2. On obtient l'ordonnancement représenté à la figure 5.1 c). On passe ensuite à la tâche 3. Elle ne peut pas venir s'insérer derrière la tâche 2, car le temps d'inactivité n'est pas suffisant. On déplace alors la tâche 5 vers la gauche. Finalement la tâche 3 est insérée juste derrière la tâche 5. On obtient l'ordonnancement représenté à la figure 5.1 d).

Nous avons prouvé les résultats suivants :

Théorème 16 *[7] Pour tout $\gamma > 0$, l'algorithme γ-ALG est $(1 + \frac{1}{\gamma}, 1 + \gamma)$-simultané-approché.*

Théorème 17 *[7] Pour tout $\gamma > 1$, il existe une instance pour laquelle il n'existe aucun ordonnancement (x, y)-simultané-approché avec $1 < x < 1 + \frac{1}{\gamma}$ et $1 < y < 1 + \frac{\gamma - 1}{2+\gamma}$.*

5.2 La coupe maximale bicritère

Ce problème a été défini dans la section 2.1.3, page 26. Par la suite, un algorithme est dit (α, β)-simultané-approché s'il retourne une coupe qui est simultanément α-approchée sur le premier critère (le poids total) et β-approchée sur le second critère (la longueur totale).

5.2.1 Un algorithme déterministe

Nous avons montré qu'étant donné un algorithme α-approché **Al** pour le problème MAX-CUT monocritère, on peut construire un algorithme $(\alpha/2, \alpha/2)$-simultané-approché pour le problème MAX-CUT bicritère. L'algorithme appelé **Bi-approx** est donné dans le tableau 5.1. On note $OPTW$ et $OPTL$ les coûts des solutions optimales sur chacun des critères.

Théorème 18 *[12]* **Bi-Approx** *est un algorithme déterministe $(\alpha/2, \alpha/2)$-simultané-approché pour le problème* MAX-CUT *bicritère si* **Al** *est un algorithme déterministe α-approché pour le problème* MAX-CUT *monocritère.*

Exemple 8 *On considère l'instance de la figure 5.2 où $OPTW = OPTL = 18$. On exécute l'algorithme* **Bi-Approx** *où* **Al** *est l'algorithme 1/2-approché de Sahni et Gonzalez. Après les étapes 1 et 2 de* **Bi-Approx**, *on suppose que $(S_1, \overline{S_1}) = (\{a, e\}, \{b, c, d\})$ et $(S_2, \overline{S_2}) = (\{a, c, e\}, \{b, d\})$.*

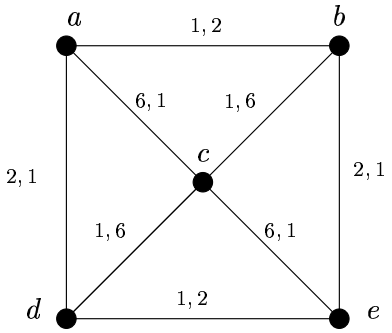


FIG. 5.2 – Illustration de l'exemple 8.

On a alors $W(S_1, \overline{S_1}) = 18$, $L(S_1, \overline{S_1}) = 8$, $W(S_2, \overline{S_2}) = 8$ *et* $L(S_2, \overline{S_2}) = 18$. *Comme* $L(S_1, \overline{S_1}) < 0.5\, L(S_2, \overline{S_2})$ *et* $W(S_2, \overline{S_2}) < 0.5\, W(S_1, \overline{S_1})$ *l'algorithme retourne la solution* $(S_3, \overline{S_3})$ t.q. $S_3 = \{a, e, b, d\}$ *et* $\overline{S_3} = \{c\}$. *On a* $W(S_3, \overline{S_3}) = 14$ *et* $W(S_3, \overline{S_3}) = 14$.

Il est possible de voir, à l'aide d'une instance, que l'analyse de l'algorithme **Bi-Approx** ne peut pas être améliorée, autrement dit que le rapport est asymptotiquement atteint.

En remplaçant l'algorithme **Al** dans **Bi-Approx** par la dérandomisation de l'algorithme de Goemans et Williamson [88, 89] qui est 0.87856-approché, on obtient le résultat suivant :

Corollaire 4 *[12] Il existe un algorithme déterministe* $(0.43928, 0.43928)$*-simultané-approché pour le problème* MAX-CUT *bicritère.*

En supposant que l'algorithme **Al** dans la procédure **Bi-Approx** est un algorithme qui retourne une solution optimale, on obtient le corollaire suivant :

Corollaire 5 *[12] Pour toute instance du problème* MAX-CUT *bicritère, il existe toujours une coupe réalisable qui approche le point idéal selon un rapport* $1/2$ *pour chaque critère.*

Se pose alors la question de savoir si le résultat précédent pourrait être amélioré. Le théorème suivant y répond par la négative.

Théorème 19 *[12] Aucun algorithme* (α, β)*-simultané-approché pour lequel* $\alpha \geq \beta > 1/2$ *(ou* $\beta \geq \alpha > 1/2$*) n'existe pour le problème* MAX-CUT *bicritère.*

5.2.2 Un algorithme probabiliste

Dans le cadre probabiliste, le problème MAX-CUT bicritère consiste à déterminer une coupe réalisable $(A, \overline{A})$ telle que

$$E[W(A, \overline{A})] \geq \alpha OPTW \text{ et } E[L(A, \overline{A})] \geq \beta OPTL$$

où $0 < \alpha \leq 1$ et $0 < \beta \leq 1$, avec $OPTW$ et $OPTL$ les coûts des solutions optimales sur chacun des critères.

Proposition 3 *[12] Aucun algorithme probabiliste* (α, β)*-simultané-approché pour le problème* MAX-CUT *bicritère où* $\alpha = \beta$ *et* $\alpha > 2/3$ *n'existe.*

Entrée :	G et **Al**	
Étape 1 :	Déterminer $(S_1, \overline{S_1})$ avec **Al** *t.q.* $W(S_1, \overline{S_1}) \geq \alpha\, OPTW$	
Étape 2 :	Déterminer $(S_2, \overline{S_2})$ avec **Al** *t.q.* $L(S_2, \overline{S_2}) \geq \alpha\, OPTL$	
Étape 3 :	Construire $(S_3, \overline{S_3})$ *t.q.* $S_3 := (S_1 \cap S_2) \cup (\overline{S_1} \cap \overline{S_2})$	
Étape 4 :	Soit $\gamma := (3 - \sqrt{5})/2$	
Étape 5 :	*Si*	$W(S_2, \overline{S_2}) \geq \gamma W(S_1, \overline{S_1})$
	Alors	*Si* $L(S_1, \overline{S_1}) \geq \gamma L(S_2, \overline{S_2})$
		Alors Retourner $(S_1, \overline{S_1})$ avec une probabilité 0.5 et $(S_2, \overline{S_2})$ avec une probabilité 0.5
		Sinon Retourner $(S_1, \overline{S_1})$ avec une probabilité γ et $(S_2, \overline{S_2})$ avec une probabilité $1 - \gamma$
	Sinon	*Si* $L(S_1, \overline{S_1}) \geq \gamma L(S_2, \overline{S_2})$
		Alors Retourner $(S_1, \overline{S_1})$ avec une probabilité $1 - \gamma$ et $(S_2, \overline{S_2})$ avec une probabilité γ
		Sinon Retourner $(S_3, \overline{S_3})$

TAB. 5.2 – L'algorithme **Ran Bi-Approx**.

Il est intéressant de remarquer que la proposition 3 a une conséquence sur l'approximabilité du problème MAX-CUT bicritère, du point de vue déterministe. En effet, il ne peut pas exister un algorithme déterministe (α, β)-simultané-approché tel que $\alpha + \beta > 4/3$. Pour le voir, supposons que nous possédons un tel algorithme. En raison de la symétrie du problème, un algorithme (α, β)-simultané-approché est aussi un algorithme (β, α)-simultané-approché. On peut alors déterminer deux solutions $(S_1, \overline{S_1})$ et $(S_2, \overline{S_2})$ telles que $W(S_1, \overline{S_1}) \geq \alpha OPTW$, $L(S_1, \overline{S_1}) \geq \beta OPTL$, $W(S_2, \overline{S_2}) \geq \beta OPTW$ et $L(S_2, \overline{S_2}) \geq \alpha OPTL$. Considérons maintenant l'algorithme probabiliste qui consiste à retourner $(S_1, \overline{S_1})$ avec une probabilité 1/2 et $(S_2, \overline{S_2})$ avec une probabilité 1/2. On aurait en sortie une coupe $(\frac{\alpha+\beta}{2}, \frac{\alpha+\beta}{2})$-simultané-approchée pour laquelle $\frac{\alpha+\beta}{2} > 2/3$.

L'algorithme (appelé **Ransam** dans [106]) qui consiste à construire une coupe $(S, \overline{S})$ en plaçant de manière équiprobable un sommet $v \in V$ soit dans S, soit dans $\overline{S}$ a une espérance 1/2-approchée pour le problème MAX-CUT monocritère. On peut remarquer qu'il atteint les mêmes performances pour toute version multicritère de MAX-CUT. Nous avons cherché à savoir s'il existait un meilleur algorithme pour la version bicritère de MAX-CUT. Nous avons proposé l'algorithme suivant, noté **Ran Bi-Approx**. Celui-ci utilise un algorithme déterministe monocritère α-approché appelé **Al** (voir le tableau 5.2).

Théorème 20 *[12]* **Ran Bi-Approx** *est un algorithme probabiliste* $(\frac{\sqrt{5}-1}{2}\alpha, \frac{\sqrt{5}-1}{2}\alpha)$*-simultané-approché pour le problème* MAX-CUT *bicritère, si* **Al** *est un algorithme déterministe* α*-approché.*

En remplaçant **Al** dans **Ran Bi-Approx** par l'algorithme de Goemans et Williamson [88, 89] on obtient le corollaire suivant :

Corollaire 6 *[12] Il existe un algorithme probabiliste* $(0.54297, 0.54297)$*-simultané-approché pour le problème* MAX-CUT *bicritère.*

5.3 Conclusion

Dans ce chapitre, nous avons considéré un problème d'ordonnancement bicritère avec deux fonctions objectif de type min-somme, à savoir la somme des temps de complétude et la somme pondérée des temps de complétude.

Nous avons obtenu un algorithme polynomial, qui pour tout $\gamma > 0$, retourne un ordonnancement $(1+\frac{1}{\gamma}, 1+\gamma)$-simultané approché. De plus, nous avons montré que pour tout $\gamma > 1$, il n'existait pas d'algorithme (polynomial ou non) (x,y)-approché avec $1 < x < 1+\frac{1}{\gamma}$ et $1 < y < 1+\frac{\gamma-1}{2+\gamma}$. On peut remarquer que lorsque γ tend vers l'infini, ces deux bornes divergent. En effet, l'algorithme γ-ALG devient $(1,\infty)$-simultané approché, tandis qu'il n'existe pas d'algorithme $(1,2)$-simultané approché. Il serait intéressant de combler ce "gap". Nous conjecturons qu'il n'existe pas de constante $\beta > 1$ telle qu'il existe un algorithme $(1+\frac{1}{\gamma}, \beta)$-simultané approché pour n'importe quelle valeur de $\gamma > 0$ donnée. Beaucoup de questions demeurent ouvertes. Par exemple est-il possible de généraliser ce résultat dans le cas de plusieurs machines, où en présence de dates d'apparition ou contraintes de précédence entres tâches ?

Nous avons également considéré le problème MAX-CUT bicritère. Un algorithme déterministe $(0.439, 0.439)$-simultané-approché, et un algorithme probabiliste $(0.542, 0.542)$-simultané-approché ont été présentés.

Une amélioration de ces résultats – négatifs comme positifs – serait bien sûr intéressante afin de diminuer l'écart qui les sépare. Cependant, entre le rapport d'approximation que l'on obtient à l'aide d'un algorithme polynomial (0.439) et celui que l'on ne peut strictement pas dépasser (0.5 en raison de la non existence d'une telle solution pour certaines instances) se situe probablement un rapport d'inapproximabilité γ tel que tout algorithme $\gamma + \varepsilon$ est nécessairement exponentiel sauf si **P=NP**. La recherche d'un tel rapport serait un apport significatif dans la compréhension du problème.

La question se pose de savoir si ces résultats peuvent être étendus, pour plus de deux critères, pour le problème MAX-CUT. Malheureusement, il est possible de construire une instance qui montre qu'il n'est pas possible de construire un algorithme déterministe qui approche le point idéal avec une garantie de performance lorsque trois critères sont pris en compte. Cependant, l'algorithme probabiliste qui consiste à construire une coupe en mettant chaque sommet $v \in V$ de manière équiprobable soit dans S, soit dans $\overline{S}$, demeure 1/2-simultané-approché pour tout k.

Chapitre 6

Approche courbe de Pareto : Algorithmes approchés

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [9, 10, 13, 15].

6.1 Introduction

Dans ce chapitre, nous présentons les résultats que nous avons obtenus concernant les courbes de Pareto approchés avec garantie de performance, pour un problème d'ordonnancement bicritère ainsi que le problème du voyageur de commerce multicritère.

Cependant, contrairement au chapitre 7, le rapport d'approximation pour les algorithmes que nous avons obtenus, ne peut pas être rendu arbitrairement petit.

La section 6.2 est consacrée au problème d'ordonnancement bicritère $1||\sum w_j C_j, \sum c_j C_j$. Nous donnons dans la sous-section 6.2.1 un algorithme capable de générer les solutions supportées de la courbe de Pareto. La sous-section 6.2.2 est consacrée à l'approche budget qui sert dans la sous-section 6.2.3 à obtenir un algorithme polynomial approchant la courbe de Pareto. La section 6.3 est consacrée au problème du voyageur de commerce multicritère avec des distances 1 et 2 sur les arêtes. Dans la sous-section 6.3.1 nous traitons du cas bicritère à l'aide d'un algorithme de recherche locale, tandis qu'à la sous-section 6.3.2 nous traitons du cas général multicritère à l'aide d'un algorithme de type plus proche voisin. Dans la sous-section 6.3.3 nous donnons des résultats de non approximabilité pour le problème TSP$(1,2)$ k-critère.

6.2 Le problème $1||\sum w_j C_j, \sum c_j C_j$

Ce problème **NP**-difficile a été défini dans la section 2.2.3.

6.2.1 Détermination des solutions supportées

Nous avons montré que le problème était indocile, en exhibant une instance pour laquelle la courbe de Pareto possède un nombre exponentiel de solutions.

Théorème 21 *[13] Le problème* $1||\sum w_j C_j, \sum c_j C_j$ *est indocile.*

Malgré ce résultat négatif, il peut être intéressant de déterminer la courbe de Pareto exacte de ce problème. En effet, si les données du problème (durées des tâches) sont polynomiales par rapport à la taille de l'instance (nombre de tâches), alors le nombre de solutions de la courbe de Pareto exacte est polynomial.

Nous avons montré qu'il était possible de déterminer les solutions supportées de la courbe de Pareto, en s'inspirant d'une méthode précédemment utilisée par Bagchi [36] pour plusieurs problèmes bicritères d'ordonnancement (voir aussi [102]).

On introduit la fonction agrégée $\tilde{f}$ suivante : $\lambda\big(\sum w_j C_j\big) + (1-\lambda)\big(\sum c_j C_j\big) = \sum(\lambda w_j + (1-\lambda)c_j)C_j$, où $0 \leq \lambda \leq 1$. Lorsque λ est connu, on est face au problème monocritère $1||\sum(\lambda w_j + (1-\lambda)c_j)C_j$. Celui-ci peut être résolu en temps polynomial à l'aide de la règle de Smith. On obtient ainsi une solution Pareto optimale supportée.

On peut alors déterminer l'ensemble des solutions supportées en résolvant le problème monocritère $1||\sum(\lambda w_j + (1-\lambda)c_j)C_j$ pour toutes les valeurs possibles de λ. Afin d'avoir un algorithme efficace, il est nécessaire de déterminer un sous-ensemble L de valeurs de λ dont la taille est bornée polynomialement, et qui soit suffisant pour générer la totalité des solutions supportées.

En fait, en exploitant une idée déjà utilisée par Ravi et Goemans [144] pour un problème bicritère d'arbre couvrant de poids minimum, il est possible de montrer qu'il y a au plus $n(n-1)/2$ valeurs importantes de λ à considérer. Pour deux tâches j et j' on définit $r(j,\lambda)$ et $r(j',\lambda)$ comme suit pour $\lambda \in [0,1]$: $r(j,\lambda) = (\lambda w_j + (1-\lambda)c_j)/p_j$, et $r(j',\lambda) = (\lambda w_{j'} + (1-\lambda)c_{j'})/p_{j'}$. On peut remarquer que pour j et j' fixés, $r(j,\lambda)$ et $r(j',\lambda)$ sont deux droites qui soit s'intersectent pour une certaine valeur notée $\lambda_{j\,j'}$, soit ne s'intersectent pas. Dans le premier cas, l'ordre entre j et j' dans un ordonnancement optimal pour $1||\sum(\lambda w_j + (1-\lambda)c_j)C_j$ dépend de la position de λ par rapport à $\lambda_{j\,j'}$. Dans le second cas, l'ordre entre j et j' ne dépend pas de λ. Ces remarques découlent directement de la règle de Smith. Ce sont ces valeurs $\lambda_{j\,j'}$, pour toute paire de tâches j, j', qui constituent l'ensemble L.

À partir de ces considérations, nous avons obtenu un algorithme, noté **AG** (voir le tableau 6.1), qui construit l'ensemble des solutions supportées pour le problème bicritère
$1||\sum w_j C_j, \sum c_j C_j$.

Exemple 9 *On considère une instance du problème* $1||\sum w_j C_j, \sum c_j C_j$ *à quatre tâches dont les temps d'exécution, poids et coûts sont donnés dans le tableau 6.2. Le tableau 6.3 donne les différentes valeurs des* $\lambda_{j\,j'}$*. Les valeurs atteintes par toutes les permutations possibles des tâches sont données dans la figure 6.1. L'algorithme* **AG** *retourne l'ensemble des solutions supportées.*

6.2.2 Le problème $1|\sum c_j C_j \leq B|\sum w_j C_j$

Rappelons qu'avec l'approche budget, étant donnée une borne B sur le coût, on cherche à déterminer un ordonnancement de poids minimum dont le coût est inférieur à B. Ce problème **NP**-difficile est noté $1|\sum c_j C_j \leq B|\sum w_j C_j$. Rappelons également (voir la section 1.2.2) qu'étant donné un budget B pour le coût, une solution (α,β)-budget-approchée est une solution de coût au plus αB et de poids au plus βW, où W est le poids minimum des solutions dont le coût est au plus B.

Entrée :	n tâches
Étape 1 :	$L := \emptyset$; $P := \emptyset$
Étape 2 :	**Pour** tout couple de tâches j, j', **Faire**
	Déterminer $\lambda_{j\,j'}$ s'il existe et le mettre dans L.
Étape 3 :	**Pour** toute valeur de $\lambda \in L$, **Faire**
	• Déterminer un ordonnancement π en plaçant les tâches par ordre décroissant des rapports $(\lambda w_j + (1-\lambda)c_j)/p_j$.
	• En cas d'égalité, placer les tâches par ordre décroissant des rapports w_j/p_j.
	• Mettre π dans P.
	Fin Pour
Étape 4 :	Soit λ_{min} la plus petite valeur de $\lambda \in L$.
Étape 5 :	Déterminer un ordonnancement π_1 en plaçant les tâches par ordre décroissant des rapports $(\lambda_{min} w_j + (1-\lambda_{min})c_j)/p_j$; en cas d'égalité, placer les tâches par ordre décroissant des rapports c_j/p_j.
Étape 6 :	Mettre π_1 dans P.
Étape 7 :	Retourner P.

TAB. 6.1 – L'algorithme **AG** pour la détermination des solutions supportées pour le problème $1||\sum w_j C_j, \sum c_j C_j$.

j	1	2	3	4
p_j	1	2	3	4
c_j	1	4	9	16
w_j	4	3	2	1

TAB. 6.2 – Caractéristiques des tâches.

j \ j'	1	2	3
2	2/7		
3	3/8	6/11	
4	4/9	8/13	12/17

TAB. 6.3 – Valeurs de $\lambda_{j\,j'}$.

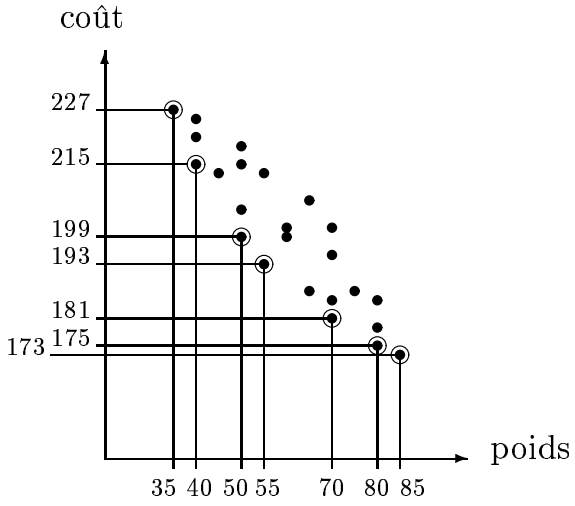


FIG. 6.1 – Valeurs atteintes par tous les ordonnancements possibles de l'instance donnée dans l'exemple 9. Les solutions supportées sont entourées.

Entrées :	n tâches, λ
Étape 1 :	Ordonner les tâches selon l'ordre décroissant des rapports $(\lambda w_j + (1-\lambda)c_j)/p_j$; les placer selon l'ordre décroissant des rapports w_j/p_j en cas d'égalité.

TAB. 6.4 – L'algorithme **ALP** retourne une solution qui, parmi celles qui minimisent $\sum(\lambda w_j + (1-\lambda)c_j)C_j$, a un poids minimum.

Cas particulier : $1|\sum C_j \leq B|\sum w_j C_j$

Dans ce cas particulier, qui demeure **NP**-difficile [102], le coût c_j de chaque tâche j est égal à 1.

On a vu dans la section précédente qu'il est possible de déterminer l'ensemble des solutions supportées en temps polynomial. En s'appuyant sur ce résultat, on peut construire un algorithme approché appelé **BUDGET** pour le problème $1|\sum C_j \leq B|\sum w_j C_j$. Celui-ci utilise deux autres algorithmes appelés **ALP** et **ALC** (voir les tableaux 6.4 et 6.5). L'algorithme **ALP** prend en entrée la quantité λ et retourne une permutation de poids minimum parmi celles qui minimisent $\sum_{j=1}^{n}(\lambda w_j + (1-\lambda)c_j)C_j$ tandis que **ALC**, qui prend aussi en entrée λ, retourne une permutation de coût minimum parmi celles qui minimisent $\sum_{j=1}^{n}(\lambda w_j + (1-\lambda)c_j)C_j$.

L'algorithme **BUDGET** est donné dans le tableau 6.6. Sauf cas particuliers, il existe généralement λ^* et deux solutions supportées $\pi_<$ et $\pi_>$, toutes deux optimales pour le problème $1||\sum(\lambda^* w_j + (1-\lambda^*)c_j)C_j$, telles que $c(\pi_<) \leq B$ et $c(\pi_>) \geq B$.

La permutation $\pi_<$ respecte la borne sur le coût mais son poids peut être éloigné de W, tandis que la permutation $\pi_>$ a un poids inférieur à W mais son coût peut être très supérieur à B. On cherche donc à retourner une solution π_{out} qui, comme $\pi_<$ et $\pi_>$, est optimale pour le problème $1||\sum(\lambda^* w_j + (1-\lambda^*)c_j)C_j$, mais dont le coût se situe entre $c(\pi_<)$ et $c(\pi_>)$. Pour ce faire, il est

Entrées :	n tâches, λ
Étape 1 :	Ordonner les tâches selon l'ordre décroissant des rapports $(\lambda w_j + (1-\lambda)c_j)/p_j$; les placer selon l'ordre décroissant des rapports c_j/p_j en cas d'égalité.

TAB. 6.5 – L'algorithme **ALC** retourne une solution qui, parmi celles qui minimisent $\sum(\lambda w_j + (1-\lambda)c_j)C_j$, a un coût minimum.

Entrées :	n tâches, un budget B		
Étape 1 :	$L := \emptyset$		
Étape 2 :	Déterminer pour toute paire de tâches j, j' la valeur $\lambda_{jj'}$ (si elle existe) et la mettre dans L.		
Étape 3 :	Soient λ_{min} et λ_{max} la plus petite et la plus grande valeur de $\lambda \in L$.		
Étape 4 :	Soient $\pi_1 = \mathbf{ALC}(\lambda_{min})$ et $\pi_2 = \mathbf{ALP}(\lambda_{min})$.		
Étape 5 :	*Si*	$B < c(\pi_1)$	
	Alors	Le problème n'a pas de solution.	
	Sinon	*Si*	$B > c(\pi_2)$
		Alors	Retourner π_2.
		Sinon	Soit $\lambda^* \in L$ tel que $c(\mathbf{ALC}(\lambda^*)) \leq B$ et $c(\mathbf{ALP}(\lambda^*)) \geq B$. Soient $\pi_{\leq} = \mathbf{ALC}(\lambda^*)$ et $\pi_{\geq} = \mathbf{ALP}(\lambda^*)$. Effectuer un *tri à bulles* pour passer de $\pi_{\leq}$ à $\pi_{\geq}$. Arrêter lorsque la permutation courante a un coût supérieur ou égal à B et la retourner.

TAB. 6.6 – L'algorithme **BUDGET** où $\mathbf{ALC}(\lambda)$ (respectivement $\mathbf{ALP}(\lambda)$) représente la solution retournée par **ALC** (respectivement **ALP**) pour λ.

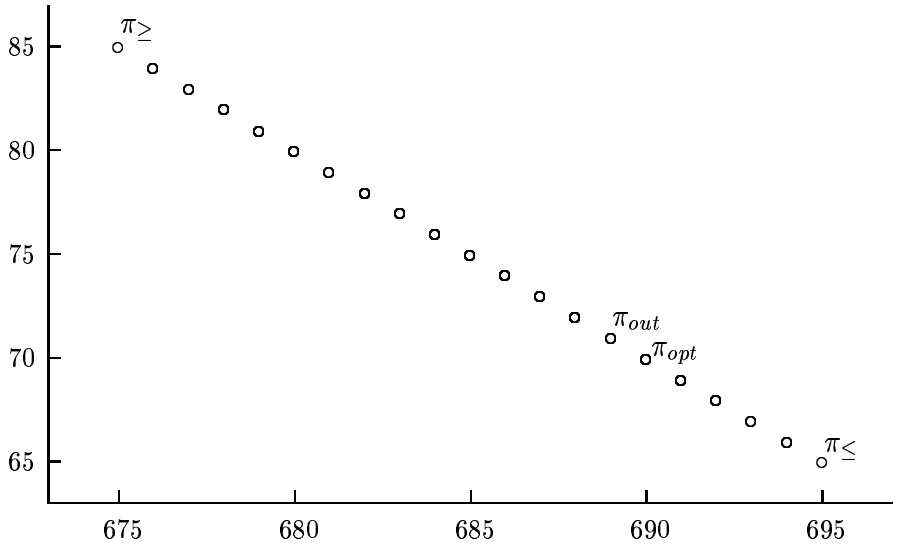


FIG. 6.2 – Valeurs (poids,coût) de toutes les permutations possibles de l'instance donnée dans l'exemple 10.

j	1	2	3	4	5
p_j	3	4	5	6	7
w_j	5	7	9	11	13
c_j	1	1	1	1	1

TAB. 6.7 – Caractéristiques des tâches pour l'instance de l'exemple 10.

possible d'utiliser un tri à bulles [118] entre les permutations $\pi_{\leq}$ et $\pi_{\geq}$. Rappelons que le tri à bulles consiste à promener une fenêtre de taille deux le long d'une liste non triée et à chaque fois que les deux éléments de la fenêtre ne sont pas ordonnés à les permuter. Le tri à bulles permet de passer de $\pi_{\leq}$ à $\pi_{\geq}$ par une suite d'échanges de tâches adjacentes. On promène la fenêtre sur $\pi_{\leq}$ et lorsque les deux tâches de la fenêtre sont dans un ordre différent de celui de $\pi_{\geq}$, on intervertit les tâches. Dans l'algorithme **BUDGET**, la première permutation intermédiaire dont le coût dépasse la borne B est retournée.

Théorème 22 *[13] L'algorithme* **BUDGET** *génère une solution* $(2,1)$*-budget-approchée pour le problème* $1|\sum C_j \leq B|\sum w_j C_j$.

Exemple 10 *Soit une instance du problème* $1|\sum C_j \leq B|\sum w_j C_j$ *à cinq tâches dont les caractéristiques sont données dans le tableau 6.7. On suppose que* $B = 70$. *On déroule l'algorithme* **BUDGET** *sur l'instance et on trouve* $\lambda_{min} = \lambda_{max} = 1/2$. *Ainsi* $\pi_1 = 1\,2\,3\,4\,5$ *et* $\pi_2 = 5\,4\,3\,2\,1$. *On a* $c(\pi_1) = 65$ *et* $c(\pi_2) = 85$. *Comme* $B = 70$, *on a* $\lambda^* = \lambda_{min}$, $\pi_{\leq} = \pi_1$ *et* $\pi_{\geq} = \pi_2$. *Le tri à bulles permet de trouver la solution* $\pi_{out} = 2\,3\,4\,1\,5$ *de coût* 71 *et de poids* 689 *(voir la figure 6.2).*

L'approximation n'est pas constante lorsque le coût c_j d'une tâche est différent de 1.

Entrées :	n tâches, $\varepsilon > 0$, **Al**
Étape 1 :	Déterminer π_1 une solution de poids minimum parmi les solutions qui minimisent le coût.
Étape 2 :	Déterminer π_2 une solution de coût minimum parmi les solutions qui minimisent le poids.
Étape 3 :	$P := \{\pi_1, \pi_2\}$
Étape 4 :	Soient $B_{min} := c(\pi_1)$ et $B_{max} := c(\pi_2)$.
Étape 5 :	Soit M *t.q.* $(1+\varepsilon)^M B_{min} \leq B_{max} \leq (1+\varepsilon)^{M+1} B_{min}$.
Étape 6 :	**Pour** $i = 1$ à M, **faire**
	$P := P \cup \{\mathbf{Al}((1+\varepsilon)^i B_{min})\}$
	Fin Pour
Étape 7 :	Retourner P.

TAB. 6.8 – Algorithme **CP** où **Al** est un algorithme (α, β)-budget-approché pour le problème $1|\sum c_j C_j \leq B|\sum w_j C_j$ et $\mathbf{Al}(B)$ est la solution retournée par **Al**.

Cas général

Pour le cas général, nous nous sommes appuyés sur une technique utilisée par Hall et al. [96] pour obtenir un algorithme 2-approché pour le problème $1|prec|\sum w_j C_j$ qui consiste à ordonner des tâches sujettes à des contraintes de précédence sur une seule machine afin de minimiser la somme pondérée des dates de terminaison. Il est possible de définir un programme linéaire dont la solution optimale permet de construire une solution $(2,2)$-budget-approchée.

Théorème 23 *[13] Il existe un algorithme $(2,2)$-budget-approché pour le problème $1|\sum c_j C_j \leq B|\sum w_j C_j$.*

6.2.3 Approche courbe de Pareto

Rappelons qu'une courbe de Pareto (α, β)-approchée pour le problème $1||\sum w_j C_j, \sum c_j C_j$ est un ensemble $P_{\alpha,\beta}$ tel que pour toute permutation Pareto optimale π^* il existe une permutation $\pi \in P_{\alpha,\beta}$ vérifiant :

$$\begin{aligned} c(\pi) &\leq \alpha c(\pi^*), \\ w(\pi) &\leq \beta w(\pi^*). \end{aligned}$$

Il est possible de mettre à profit l'approche budget pour construire une courbe de Pareto approchée. Nous avons proposé l'algorithme **CP**, donné dans le tableau 6.8, et qui utilise un découpage géométrique de l'intervalle des coûts.

Théorème 24 *[13] L'algorithme* **CP** *construit une courbe de Pareto $(\alpha(1+\varepsilon), \beta)$-approchée pour le problème $1||\sum w_j C_j, \sum c_j C_j$, si* **Al** *est un algorithme (α, β)-budget-approché pour le problème $1|\sum c_j C_j \leq B|\sum w_j C_j$.*

En remplaçant **Al** dans **CP** par l'algorithme $(2,2)$-budget-approché pour le problème $1|\sum c_j C_j \leq B|\sum w_j C_j$, on obtient le corollaire suivant :

Corollaire 7 *[13] Il existe un algorithme qui construit en temps polynomial une courbe de Pareto $(2+\varepsilon, 2)$-approchée pour le problème $1||\sum w_j C_j, \sum c_j C_j$.*

Entrée :	n tâches
Étape 1 :	$L := \emptyset$; $P' := \emptyset$
Étape 2 :	Déterminer l'ensemble des solutions supportées et les mettre dans L.
Étape 3 :	Trier L par ordre croissant des poids totaux.
Étape 4 :	**Pour** tout couple π_1, π_2 de solutions consécutives dans L, **faire** Effectuer un tri à bulles pour passer de π_1 à π_2 et placer toutes les permutations rencontrées dans P'. **Fin Pour**
Étape 5 :	Retourner P'.

TAB. 6.9 – Algorithme permettant de déterminer une courbe de Pareto $(2,1)$-approchée pour le problème $1||\sum w_j C_j, \sum C_j$.

De même, en remplaçant **Al** dans **CP** par l'algorithme **BUDGET** qui est $(2,1)$-budget-approché pour le problème $1|\sum C_j \leq B|\sum w_j C_j$, on obtient le corollaire suivant :

Corollaire 8 *[13] Il existe un algorithme qui construit en temps polynomial une courbe de Pareto $(2+\varepsilon,1)$-approchée pour le problème $1||\sum w_j C_j, \sum C_j$.*

Ce dernier résultat peut être amélioré en qualité et en complexité.

Théorème 25 *[13] Il existe un algorithme qui construit en temps polynomial une courbe de Pareto $(2,1)$-approchée pour le problème $1||\sum w_j C_j, \sum C_j$.*

6.3 Le voyageur de commerce multicritère avec distances 1 et 2

Le problème du voyageur de commerce multicritère a été défini dans la section 2.1.1. On cherche à produire en temps polynomial un ensemble $P_{\vec{\varepsilon}}$ de solutions $\vec{\varepsilon}$-approché tel que pour tout tour t^* (*a fortiori* les tours Pareto optimaux), il existe dans $P_{\vec{\varepsilon}}$ un tour t vérifiant :

$$\begin{aligned} \vec{D}_1(t) &\leq (1+\varepsilon_1)\vec{D}_1(t^*), \\ \vec{D}_2(t) &\leq (1+\varepsilon_2)\vec{D}_2(t^*). \end{aligned}$$

6.3.1 Le problème TSP$(1,2)$ bicritère

Nous avons proposé un algorithme basé sur la recherche locale pour le problème TSP$(1,2)$ bicritère, et qui retourne une courbe de Pareto approchée.

Rappelons que pour un problème de minimisation monocritère, la recherche locale remplace à chaque itération la solution courante par une meilleure solution dans son voisinage (voir la section 1.1.1). Pour le problème du voyageur de commerce, le voisinage *2-opt* est très utilisé [116]. Étant donné un tour t, son voisinage $\mathcal{N}_{2-opt}(t)$ consiste en l'ensemble des tours qui peuvent être obtenus à partir de t en enlevant deux arêtes non adjacentes de t (à savoir $e = [v_1, v_2]$ et $e' = [v_3, v_4]$ dans la figure 6.3) et en insérant deux nouvelles arêtes (à savoir $e'' = [v_2, v_4]$ et $e''' = [v_1, v_3]$ dans la figure 6.3) afin d'obtenir un nouveau tour t'.

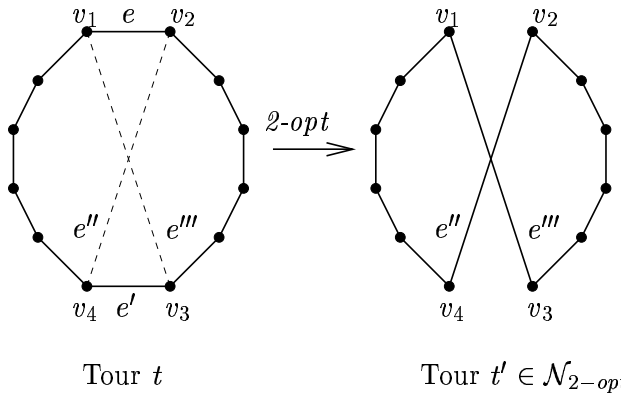


FIG. 6.3 – Le mouvement *2-opt* consiste à enlever deux arêtes non adjacentes pour en insérer deux nouvelles. On obtient ainsi un nouveau tour t', voisin de t.

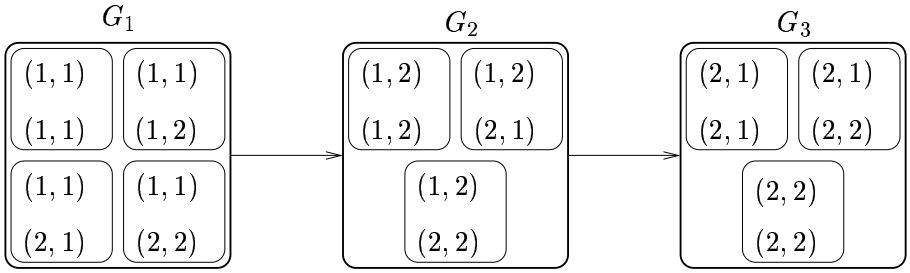


FIG. 6.4 – Selon la relation de préférence **pref1**, un couple d'arêtes dont les vecteurs de distances appartiennent à G_i est préféré à tout couple d'arêtes dont les vecteurs de distances appartiennent à G_j si et seulement si $i < j$.

Dans le cadre monocritère, une solution est préférée à une autre si sa longueur est plus petite. Dans le cadre multicritère, la notion de préférence entre deux solutions voisines est à définir. La notion de préférence n'étant utile que pour comparer deux solutions voisines, celle-ci peut être définie en lien avec la définition même du voisinage. Pour le voisinage 2-opt, deux tours voisins ne diffèrent que de deux couples d'arêtes ($t - \{e, e'\} = t' - \{e'', e'''\}$ dans la figure 6.3) et la préférence peut s'exprimer sur les vecteurs de distances de ces deux couples d'arêtes. Dans la figure 6.4, tous les vecteurs de distances possibles sont répartis en trois groupes $\{G_1, G_2, G_3\}$ reflétant la préférence **pref1** (une autre relation de préférence appelée **pref2** est définie dans la figure 6.5). L'idée est de favoriser en premier lieu les couples d'arêtes dont l'une a un couple de distances $(1, 1)$. Ensuite on favorise les couples restant contenant au moins une arête ayant un vecteur de distances $(1, 2)$.

Définition 22 (préférence pref1) *Pour deux couples d'arêtes (e, e') et (e'', e''') tels que $(\vec{d}(e), \vec{d}(e')) \in G_i$ et $(\vec{d}(e''), \vec{d}(e''')) \in G_j$, le couple (e, e') est préféré au couple (e'', e''') si et seulement si $i < j$.*

L'algorithme **BLS** (pour *Bicriteria Local Search*) que nous avons proposé est décrit dans le tableau 6.10 ci-dessous. Basé sur la *recherche locale*, cet algorithme produit deux solutions $\{t_1, t_2\}$ telles que la première (t_1) favorise le premier critère et permet d'approcher les tours t^* vérifiant $\vec{D}_1(t^*) \leq \vec{D}_2(t^*)$ tandis que la seconde (t_2) favorise le second critère et permet d'approcher les

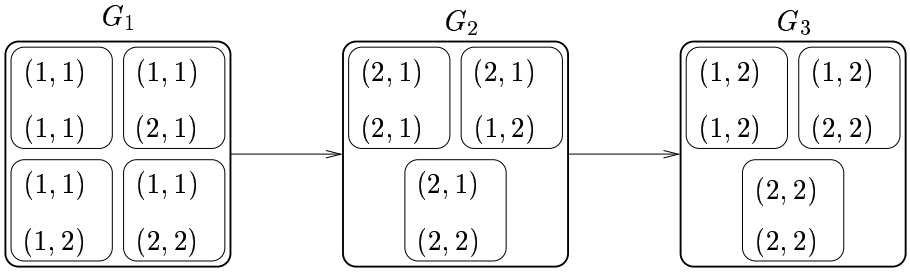


FIG. 6.5 – Selon la relation de préférence **pref2**, un couple d'arêtes dont les vecteurs de distances appartiennent à G_i est préféré à tout couple d'arêtes dont les vecteurs de distances appartiennent à G_j si $i < j$.

Entrée :	$G(V, E)$
Étape 1 :	Construire t_1 à l'aide de **pref1**.
Étape 2 :	Construire t_2 à l'aide de **pref2**.
Étape 3 :	Retourner $\{t_1, t_2\}$.

TAB. 6.10 – L'algorithme de recherche locale **BLS**.

tours t^* vérifiant $\vec{D}_2(t^*) \leq \vec{D}_1(t^*)$. L'algorithme pour construire t_1 est donné dans le tableau 6.11. L'algorithme pour construire t_2 est identique, à la différence que la relation **pref2** est utilisée à la place de **pref1**.

On peut montrer le résultat suivant :

Lemme 1 *[9, 10] La solution t_1 $(\frac{1}{2}, \frac{1}{2})$-domine toute solution t^* Pareto optimale telle que $\vec{D}_1(t^*) \leq \vec{D}_2(t^*)$. La solution t_2 $(\frac{1}{2}, \frac{1}{2})$-domine toute solution t^* Pareto optimale telle que $\vec{D}_1(t^*) \geq \vec{D}_2(t^*)$.*

On obtient donc le théorème suivant :

Théorème 26 *[9, 10] Les solutions t_1 et t_2 constituent un ensemble de Pareto $(1/2, 1/2)$-approché pour le problème* TSP$(1, 2)$ *bicritère.*

L'analyse de l'algorithme **BLS** en terme de garantie de performance ne peut pas être améliorée car on peut exhiber une instance pour laquelle le rapport est atteint.

Entrée :	$G(V, E)$ et **pref1**
Étape 1 :	Soit t_1 un tour réalisable quelconque.
Étape 2 :	**Tant qu'** il existe $t'_1 \in \mathcal{N}_{2-opt}(t_1)$ tel que $t_1 - \{e, e'\} = t'_1 - \{e'', e'''\}$ et (e'', e''') est préféré à (e, e') selon **pref1**, **faire** $t_1 := t'_1$ **Fin Tant que**
Étape 3 :	Retourner t_1.

TAB. 6.11 – Algorithme de recherche locale bicritère pour le TSP$(1, 2)$: construction de t_1.

On peut montrer que le nombre d'itérations de l'algorithme de recherche locale bicritère donné dans le tableau 6.11, est borné par $3n$. La taille du voisinage 2-opt étant $\mathcal{O}(n^2)$, la durée d'exécution de cet algorithme est donc $\mathcal{O}(n^3)$.

La section suivante est consacrée à une version plus générale du problème TSP$(1,2)$ où le nombre de critères noté k est supérieur ou égal à 2.

6.3.2 Le problème TSP$(1,2)$ k-critère

Il est sans doute possible de généraliser l'algorithme **BLS** pour le TSP$(1,2)$ à k critères pour $k \geq 3$, mais ceci semble extrêmement fastidieux à cause de la relation de préférence qu'il faudrait définir entre toute paire de deux couples d'arêtes. Lorsque k est grand, avec 2^{2k} couples de vecteurs possibles, un tel ordre est difficilement maniable.

Nous avons proposé un algorithme de type *plus proche voisin* qui génère en temps $\mathcal{O}(n^2 k!)$ une courbe de Pareto $\frac{k-1}{k+1}$-approchée lorsque $k \geq 3$ et une courbe de Pareto $(1/2, 1/2)$-approchée lorsque $k = 2$. Le fait que la complexité en temps de l'algorithme dépende de $k!$ n'est pas surprenant, car la taille d'une courbe de Pareto approchée n'est pas nécessairement polynomiale par rapport au nombre k de critères [135].

Traditionnellement, l'algorithme du plus proche voisin [147] consiste à débuter par un sommet quelconque et à insérer de manière gloutonne le sommet le plus proche du dernier sommet inséré. L'adaptation de cet algorithme dans le cadre multicritère soulève deux problèmes :

- Comment traduire la notion de proximité lorsque plusieurs objectifs sont considérés ?
- Combien de solutions doivent être générées pour obtenir une approximation de la courbe de Pareto ?

Nous avons adopté une approche similaire à celle suivie pour **BLS** : Diviser l'espace objectif en plusieurs portions, et retourner une solution approchant chaque portion à l'aide d'une relation de préférence convenablement choisie.

Pour toute instance du TSP$(1,2)$ k-critère, l'espace des coûts est divisé en $k!$ portions comme suit : Chaque portion est identifiée par une permutation de $\{1, 2, \ldots, k\}$. Étant donnée une permutation L de $\{1, 2, \ldots, k\}$, un tour t est dans la portion identifiée par L si $\vec{D}_{L(1)}(t) \leq \ldots \leq \vec{D}_{L(k)}(t)$. Pour la notion de proximité, on introduit une relation de préférence sur l'ensemble des vecteurs de distances possibles qui ressemble à l'ordre lexicographique. Cette relation de préférence qui dépend de L (notée $\prec_L$) est définie en utilisant $k+1$ ensembles notés $S_1, \ldots, S_{k+1}$ et définis comme suit :

$$\begin{aligned} S_q &= \{\vec{a} \in \{1,2\}^k \mid \forall j \leq k+1-q \quad \vec{a}_{L(j)} = 1\}, \quad \text{pour } 1 \leq q \leq k \\ S_{k+1} &= \{1,2\}^k. \end{aligned}$$

De manière concrète, S_1 contient le vecteur $\{1\}^k$, S_2 contient S_1 plus le vecteur ayant un 1 sur toutes les coordonnées sauf $L(k)$, S_3 contient S_2 plus tous les vecteurs ayant un 1 sur toutes les coordonnées sauf $L(k-1)$ et $L(k)$, et ainsi de suite jusqu'à S_{k+1} contenant tous les vecteurs possibles.

Définition 23 *Pour toute arête e, on dit que e est S_q-*préférée *(pour $\prec_L$) si $\vec{d}(e) \in S_q \backslash S_{q-1}$ (où $S_0 = \emptyset$). Pour deux arêtes e et e' telles que e est S_q-préférée et e' est $S_{q'}$-préférée, on dit que $\vec{d}(e)$ est préféré (respectivement faiblement préféré) à $\vec{d}(e')$ et on le note $\vec{d}(e) \prec_L \vec{d}(e')$ (respectivement $\vec{d}(e) \preccurlyeq_L \vec{d}(e')$) si et seulement si $q < q'$ (respectivement $q \leq q'$).*

Un exemple où $k = 3$ et L est la permutation identité est donné en figure 6.6.

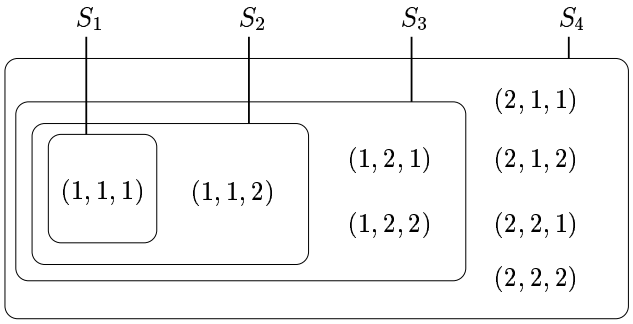


FIG. 6.6 – On a $(1,1,1) \prec_L (1,1,2) \prec_L (1,2,1) \preccurlyeq_L (1,2,2) \prec_L (2,1,1) \preccurlyeq_L (2,1,2) \preccurlyeq_L (2,2,1) \preccurlyeq_L (2,2,2)$ pour L étant la permutation identité et $k = 3$.

L'algorithme que nous avons proposé pour le TSP$(1,2)$ k-critère est donné dans le tableau 6.12. Appelé **kNN** (*pour k-criteria Nearest Neighbor*), il se déroule en $k!$ étapes. Une permutation L de $\{1, 2, \ldots, k\}$ est déterminée à chaque étape. On construit avec L une relation de préférence $\prec_L$ et finalement, un tour est généré à l'aide de la règle du plus proche voisin. Dans la définition de **kNN**, $p(v)$ où v est un sommet du graphe désigne le sommet qui suit immédiatement v dans la construction du tour p.

Théorème 27 *[15] L'algorithme* **kNN** *retourne un ensemble de Pareto $\frac{k-1}{k+1}$-approché pour le problème* TSP$(1,2)$ *k-critère lorsque $k \geq 3$ et un ensemble de Pareto $(\frac{1}{2}, \frac{1}{2})$-approché lorsque $k = 2$.*

En construisant une famille d'instances pour laquelle **kNN** atteint le pire cas en terme d'approximation, on peut montrer que l'analyse de l'algorithme **kNN** pour le TSP$(1,2)$ k-critère ne peut être améliorée, autrement dit que les bornes énoncées dans le théorème 27 sont atteintes.

6.3.3 Non-approximabilité par rapport au nombre de solutions

Nous avons cherché à obtenir des résultats de non approximabilité pour le problème TSP$(1,2)$ k-critère. Traditionnellement, un résultat d'inapproximabilité pour un problème de minimisation monocritère est la donnée d'un rapport ρ tel que pour tout $\delta > 0$, aucun algorithme polynomial ne peut prétendre être $(\rho - \delta)$-approché, sauf si **P**=**NP**. Étant donné un tel résultat de non approximabilité pour un problème monocritère Π, il est clair que l'on obtient directement un résultat négatif pour toute version multicritère de Π. Par exemple, l'inapproximabilité du TSP$(1,2)$ monocritère a été étudiée dans les articles [77, 78, 134] et la meilleure borne inférieure est $1 + 1/740 - \delta$ (pour tout $\delta > 0$). Par conséquent, pour tout $\delta > 0$, aucun algorithme polynomial ne peut construire une courbe de Pareto $(1/740 - \delta)$-approchée sur chaque critère, sauf si **P**=**NP**.

Nous avons cherché à obtenir un autre type de résultat d'inapproximabilité. On peut facilement faire le constat suivant : plus le nombre de solutions contenues dans une courbe de Pareto approchée est grand, plus celle-ci a de chances d'être une bonne approximation de la courbe exacte. Ainsi, le meilleur rapport d'approximation que peut atteindre une courbe de Pareto approchée peut être relié au nombre de solutions qu'elle contient. Plus formellement, $\vec{\varepsilon}$ pour une courbe de Pareto approchée $P_{\vec{\varepsilon}}$ est une fonction de $|P_{\vec{\varepsilon}}|$. Nous avons donc cherché à savoir quel pouvait être le meilleur rapport d'approximation $\vec{\varepsilon}$ en fonction de $|P_{\vec{\varepsilon}}|$.

Pour ce faire, la technique que nous avons mise en œuvre peut se résumer de la manière suivante. Si on considère des instances pour lesquelles la totalité (ou une grande part) de la courbe de Pareto

Entrée : $G(V, E)$
Étape 1 : $P_{\vec{\varepsilon}} := \emptyset$
Étape 2 : **Pour** toutes les permutations L de $\{1, 2, \ldots, k\}$ **Faire**
Prendre arbitrairement $v \in V$.
$W := \{v\}$
$u := v$
Tant que $W \neq V$ **Faire**
Prendre $r \in V \backslash W$ *t.q.* r est le sommet le plus proche de u par $\preccurlyeq_L$
$W := W \cup \{r\}$
$p(u) := r$
$u := r$
Fin Tant que
$p(r) := v$
$P_{\vec{\varepsilon}} := P_{\vec{\varepsilon}} \cup \{p\}$
Fin Pour
Étape 3 : Retourner $P_{\vec{\varepsilon}}$.

TAB. 6.12 – L'algorithme **kNN** où $p(v)$ désigne le sommet qui suit immédiatement $v \in V$ dans le tour p.

exacte P est connue et si on suppose qu'on l'approche avec un ensemble de points $P' \subset P$ tel que $|P'| = x$ alors le meilleur rapport d'approximation $\vec{\varepsilon}$ tel que P' est une courbe de Pareto $\vec{\varepsilon}$-approchée dépend de x. En effet, il doit nécessairement y avoir une solution dans P' qui approche au moins deux (ou plus) solutions de P.

Considérons par exemple l'instance I du problème TSP$(1,2)$ bicritère donnée en figure 6.7 où les arêtes non-représentées ont un vecteur distance $(2, 2)$. L'instance admet (entre autres) les trois tours Pareto optimaux suivants : $t_1 = v_a^1\, v_d^1\, v_b^1\, v_e^1\, v_c^1\, v_a^2\, v_d^2\, v_b^2\, v_e^2\, v_c^2$ de poids $\vec{D}(t_1) = (10, 20)$, $t_2 = v_a^1\, v_e^1\, v_d^1\, v_c^1\, v_b^1\, v_a^2\, v_e^2\, v_d^2\, v_c^2\, v_b^2$ de poids $\vec{D}(t_2) = (20, 10)$, et $t_3 = v_a^1\, v_e^1\, v_d^1\, v_c^1\, v_b^1\, v_a^2\, v_d^2\, v_b^2\, v_e^2\, v_c^2$ de poids $\vec{D}(t_3) = (15, 15)$.

On peut montrer que sur cette instance, la meilleure courbe de Pareto $(\varepsilon, \varepsilon)$-approchée, de taille x, va être constituée de x solutions parmi celles qui sont Pareto optimales. Si $x = 1$ alors le meilleur choix pour l'instance I est de retourner t_3 qui constitue une courbe de Pareto $(1/2, 1/2)$-approchée. Pour l'instance I, on peut remarquer qu'aucune courbe de Pareto réduite à une seule solution n'est $(1/2 - \delta, 1/2 - \delta)$-approchée, pour tout $\delta > 0$. Ainsi, en général pour le problème TSP$(1,2)$ bicritère, il n'existe pas d'ensemble de Pareto réduit à un seul tour qui soit $(1/2 - \delta, 1/2 - \delta)$-approché. *A fortiori*, aucun algorithme, polynomial ou exponentiel, ne peut prétendre retourner une telle courbe.

Par la suite, nous avons généralisé cette technique au problème TSP$(1,2)$ k-critère, en proposant une famille d'instances pour laquelle on connaît un grand nombre de tours Pareto optimaux couvrant un large spectre des valeurs possibles.

Théorème 28 *[15] Pour tout $k \geq 2$, toute courbe de Pareto $\vec{\varepsilon}$-approchée avec au plus x solutions pour le* TSP$(1,2)$ *k-critère satisfait :*

$$\varepsilon \geq max_{i=2,\ldots,k}\left\{\frac{1}{(2i-1)r(i,x)-1}\right\}$$
$$\text{où } r(i,x) = min\{r|\ x \leq C^r_{r+i-1} - 1\}.$$

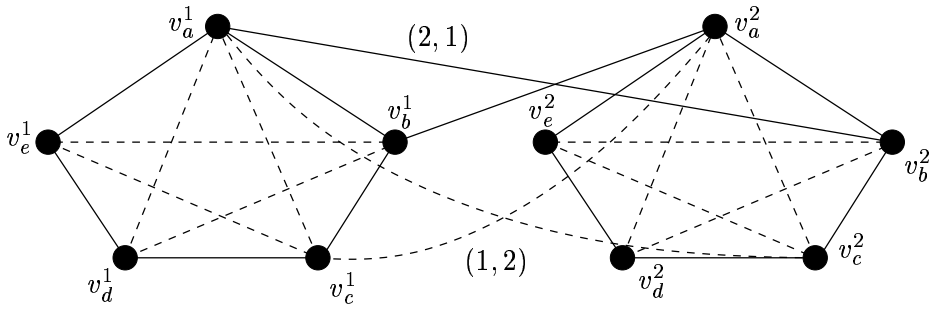


FIG. 6.7 – Pour cette instance et tout $\delta > 0$, il n'existe pas de tour qui soit $(1/2-\delta, 1/2-\delta)$-approché de tous les autres.

k \ x	1	2	3	4	5	6	7	8	9
2	0.500	0.200	0.125	0.090	0.071	0.058	0.050	0.043	0.038
3	0.500	0.250	0.125	0.111	0.111	0.071	0.071	0.071	0.071
4	0.500	0.250	0.166	0.111	0.111	0.076	0.076	0.076	0.076

TAB. 6.13 – Valeurs numériques de ε obtenues par le théorème 28 pour certaines valeurs de k et de x.

Le tableau 6.13 illustre les valeurs numériques de ε obtenues par le théorème 28 pour certaines valeurs de k et x.

6.4 Conclusion

Nous avons considéré le problème d'ordonnancement bicritère $1||\sum w_j C_j, \sum c_j C_j$. Le caractère **NP**-difficile du problème étant déjà connu, nous avons montré à l'aide d'une instance que la courbe de Pareto exacte pouvait être de taille exponentielle. Un algorithme polynomial basé sur la technique d'agrégation des critères a permis de générer toutes les solutions supportées pour ce problème. Par le biais d'une approche budget, nous avons pu construire deux algorithmes d'approximation avec garantie de performance. Le premier tire partie de la génération des solutions supportées tandis que le second est basé sur la programmation linéaire. Enfin, des algorithmes permettant de construire une courbe de Pareto approchée avec garantie de performance sur chacun des critères ont pu être dérivés des algorithmes précédents. La question se pose de savoir s'il est possible d'améliorer ces résultats d'approximation, et d'obtenir éventuellement un schéma d'approximation polynomial.

Nous avons considéré une généralisation du problème TSP$(1,2)$ à plusieurs critères. Par le biais d'un algorithme de recherche locale, nous avons pu construire une courbe de Pareto $(1/2, 1/2)$-approchée pour le cas bicritère. Nous avons aussi étudié le cas k-critère, et une extension de l'algorithme du *plus proche voisin* nous a permis de construire une courbe de Pareto approchée avec garantie de performance dont le rapport dépend malheureusement de k. Là aussi, la question de savoir si ces rapports d'approximation peuvent être améliorés demeure ouverte. Le fait que les algorithmes proposés atteignent leur borne sur certaines instances laisse à penser que les techniques algorithmiques déployées à cet effet doivent être différentes de celles présentées ici. La technique d'assemblage de sous-cycles (*subtour patching*), ayant fait ses preuves pour le cas monocritère, est encore non adaptée au cas multicritère. Il s'agit peut-être d'une piste à explorer.

Nous nous sommes aussi intéressés à des résultats négatifs concernant la non existence d'ensembles $\vec{\varepsilon}$-approchés. En mettant en relation le nombre de solutions dans un ensemble et le nombre de critères, nous avons montré que l'on pouvait dériver des résultats disant que cet ensemble est au mieux $\vec{\varepsilon}$-approché de la courbe de Pareto exacte, pour certaines valeurs de $\vec{\varepsilon}$. Cette technique peut sans doute être appliquée à d'autres problèmes. Ces résultats d'inapproximabilité se basant sur des arguments de non existence pourraient dans l'avenir être complétés par des résultats d'inapproximabilité basés sur des arguments de complexité.

Chapitre 7

Approche courbe de Pareto : Schémas d'approximation totalement polynomiaux

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [16, 17].

7.1 Introduction

En optimisation monocritère, une technique classique pour obtenir des schémas d'approximation totalement polynomiaux consiste à définir dans un premier temps un algorithme pseudo-polynomial, puis à modifier l'instance en procédant à des arrondis [170]. Ce genre d'approche peut-il s'appliquer à des problèmes multicritères afin d'obtenir des courbes de Pareto approchées ?

Nous avons considéré le problème $Rm||C_{max}, \sum_j c_j$, et nous avons montré qu'à partir d'un programme dynamique pseudo-polynomial on pouvait obtenir un schéma d'approximation totalement polynomial pour construire une courbe de Pareto approchée. Ces travaux sont exposés à la section 7.2.

Plutôt que de procéder problème par problème, peut-on identifier une classe de problèmes d'optimisation multicritères admettant un schéma d'approximation totalement polynomial pour calculer une courbe de Pareto approchée ?

Papadimitriou et Yannakakis ont prouvé qu'il existait toujours une courbe de Pareto ε-approchée de taille polynomiale, pour tout $\varepsilon > 0$ [135]. De plus, ils ont donné une condition suffisante pour l'existence d'un schéma d'approximation totalement polynomial calculant une courbe de Pareto approchée. Cette condition impose que les fonctions objectifs du problème d'optimisation multi-

m général			m constante	
référence	[125]	[162]	[112]	*
qualité	$(2+\frac{1}{\varepsilon})T, (1+\varepsilon)C$	$2T, C$	$(1+\varepsilon)T, (1+\varepsilon)C$	$(1+\varepsilon)T, C$
temps	$poly(m,n)$	$poly(m,n)$	$n(\frac{m}{\varepsilon})^{\mathcal{O}(m)}$	$\mathcal{O}(n(\frac{n}{\varepsilon})^m)$

TAB. 7.1 – Principaux résultats concernant le problème $Rm||C_{max}, \sum_j c_j$. Notre contribution est indiquée par '*'.

critère soient *linéaire*. Nous avons montré que cette méthode pouvait être appliquée au problème $Pm||\sum w_j C_j, C_{max}$. Cette approche est développée à la section 7.3.

Nous avons également développé une méthode alternative à celle de Papadimitriou et Yannakakis afin d'obtenir des résultats génériques d'existence de tels schémas d'approximation. Pour cela nous nous sommes appuyés sur des résultats de Woeginger [173] qui a identifié une classe de problèmes d'optimisation monocritères admettant des schémas d'approximation totalement polynomiaux. Cette classe contient un ensemble de problèmes qui peuvent être résolus à l'aide d'un programme dynamique simple vérifiant certaines propriétés. Nous avons montré que cette approche pouvait être étendue dans le contexte des problèmes d'optimisation multiobjectifs. Cette approche est exposée en deux temps. Dans la section 7.4 (respectivement 7.5) nous traitons du lien entre un programme dynamique exact (respectivement arrondi) et l'obtention d'une courbe de Pareto exacte (respectivement approchée). Des exemples d'applications à des problèmes d'ordonnancement sont exposés dans la section 7.6.

7.2 Le problème $Rm||C_{max}, \sum_j c_j$

Rappelons que ce problème a été défini dans la section 2.2.4. Soit $C_{opt}(T)$ le coût minimum d'un ordonnancement qui a un *makespan* égal à T. Nous avons proposé un algorithme qui étant donné T, retourne pour tout $\varepsilon > 0$ un ordonnancement avec un *makespan* au plus égal $(1+\varepsilon)T$ et de coût au plus $C_{opt}(T)$, et ceci en temps $\mathcal{O}(n(n/\varepsilon)^m)$. De plus, nous avons adapté notre algorithme afin d'obtenir une courbe de Pareto $(\varepsilon, 0)$-approchée.

Le tableau 7.1 indique les références de la littérature concernant ce problème. L'algorithme de Jansen et Porkolab [112] est moins bon vis à vis du coût de la solution retournée. De plus, notre méthode est plus simple et ne requiert que la connaissance de T, en contrepartie la complexité de notre algorithme est plus importante.

Remarque 1 *Il existe des problèmes d'ordonnancement pour lesquels il n'est pas nécessaire d'ordonnancer toutes les tâches. Dans ce genre de problème, il faut décider pour chaque tâche si elle est ordonnancée ou si elle est rejetée. Dans ce dernier cas, il y a une pénalité à payer c_j qui dépend de la tâche rejetée j. L'objectif peut être de minimiser la somme du* makespan *des tâches acceptées avec la pénalité pour les tâches rejetées. Bartal et al. [41] ont considéré les versions en ligne et hors ligne de ce problème dans le cas de machines identiques. Epstein et Sgall [79] ont considéré le cas de machines reliées. Enfin, Hoogeveen, Skutella et Woeginger [103] ont considéré une version préemptive, avec des machines non reliées, et ont présenté une classification des différentes variantes du problème.*

Il est intéressant de remarquer que les résultats que nous avons obtenus pour le problème $Rm||C_{max}, \sum_j c_j$ peuvent s'appliquer au problème où les tâches peuvent être rejetées, dans le cas non préemptif, et sur des machines non reliées. Il suffit d'ajouter une machine "fictive" $m+1$, telle que la durée

1. Initialement, poser $E_c(1, 0, p_{21}) = c_{21}$, $E_c(1, p_{11}, 0) = c_{11}$,
   et sinon $E_c(j, m_1, m_2) = +\infty$ pour $j \geq 1$.
2. **Pour** $j = 1$ **jusqu'à** $n - 1$
3. **Pour** $m_1 = 0$ **jusqu'à** T par pas de 1
4. **Pour** $m_2 = 0$ **jusqu'à** T par pas de 1
5. $E_c(j+1, m_1 + p_{1j}, m_2) \leftarrow \min\{E_c(j+1, m_1 + p_{1j}, m_2),\ E_c(j, m_1, m_2) + c_{1j}\}$
6. $E_c(j+1, m_1, m_2 + p_{2j}) \leftarrow \min\{E_c(j+1, m_1, m_2 + p_{2j}),\ E_c(j, m_1, m_2) + c_{2j}\}$

TAB. 7.2 – L'algorithme de programmation dynamique $PD1$.

d'exécution de toutes les tâches j soit nulle sur cette machine, et de poser $c_{m+1,j} = c_j$ et $c_{i,j} = 0$ pour $1 \leq i \leq m$. La machine $m+1$ recevra toutes les tâches rejetées. Par conséquent, nous pouvons obtenir un schéma d'approximation totalement polynomial pour construire une courbe de Pareto $(\varepsilon, 0)$-approchée en temps $\mathcal{O}(\frac{n^2}{\varepsilon}(n/\varepsilon)^{m+1})$.

7.2.1 Un programme dynamique

Dans la suite, nous ne considérons que le cas de deux machines, mais la généralisation pour un nombre constant $m > 2$ de machines est directe. Soit $J = \{1, 2, \ldots, n\}$ l'ensemble des tâches. On suppose, dans cette section seulement, que les temps d'exécution p_{ij} sont des entiers positifs. Pour un ordonnancement s, éventuellement partiel, c'est-à-dire ne comprenant qu'un sous-ensemble de toutes les tâches, on note $C(s)$ et $T(s)$ son coût et son *makespan*. On définit $C_{opt}(T)$, pour tout T, comme étant le coût minimum d'un ordonnancement qui a un *makespan* égal à T s'il en existe un, i.e. $C_{opt}(T) = \min_{s, T(s)=T} C(s)$. S'il n'existe pas d'ordonnancement avec un *makespan* égal à T, on pose $C_{opt}(T) = +\infty$. Nous décrivons dans cette section, un algorithme de programmation dynamique pseudo-polynomial, qui étant donné T retourne un ordonnancement s avec un *makespan* égal à T et un coût $C_{opt}(T)$ si un tel ordonnancement existe.

On note $s = (m_1, m_2) = (J_1, J_2)$ pour indiquer que m_1 (respectivement m_2) est le temps d'exécution total sur la machine 1 (respectivement 2), et que J_1 (respectivement J_2) est l'ensemble des tâches ordonnancées sur la machine 1 (respectivement 2). Si $s = (J_1, J_2)$ le *makespan* est défini par $T(s) = \max\{m_1 = \sum_{j \in J_1} p_{1j}, m_2 = \sum_{j \in J_2} p_{2j}\}$, et le coût par $C(s) = \sum_{j \in J_1} c_{1j} + \sum_{j \in J_2} c_{2j}$.

Les états du programme dynamique sont les tuples $E(j, m_1, m_2)$, avec $1 \leq j \leq n$ et $m_1, m_2 \in \{0, 1, \ldots, T\}$. On pose $E_c(j, m_1, m_2) = +\infty$ s'il n'existe pas d'ordonnancement faisant intervenir les tâches $\{1, 2, \ldots j\}$ tel que la charge totale sur la machine 1 (respectivement 2) soit égale à m_1 (respectivement m_2). Sinon, $E_c(j, m_1, m_2)$ est le coût minimum des ordonnancements de ce type. Le programme dynamique, noté $PD1$, est donné dans le tableau 7.2. Le programme dynamique $PD1$ ne conserve que les coûts des solutions, et non les solutions elles mêmes, mais à l'aide de variables supplémentaires, on peut facilement retrouver les ordonnancements qui atteignent ces coûts.

Pour obtenir une solution avec un *makespan* égal à T et un coût $C_{opt}(T)$, on sélectionne tous les états $E(n, T, m_2)$ et $E(n, m_1, T)$ (s'il en existe), avec $m_1, m_2 \in \{0, 1, \ldots, T\}$, et parmi ces états on choisit celui ayant le plus petit coût E_c.

Le nombre d'états est $n(T+1)^2$ et par conséquent cet algorithme est seulement pseudo-polynomial.

1. Initialement, poser $E_c(1,0,p_{21}) = c_{21}$, $E_c(1,p_{11},0) = c_{11}$, et sinon $E_c(j,m_1,m_2) = +\infty$ pour $j \geq 1$.
2. Remplacer p_{ij} par $[p_{ij}]$ pour $i \geq 1$ et $j \geq 1$
3. **Pour** $j = 1$ **jusqu'à** $n-1$
4. **Pour** $m_1 = 0$ **jusqu'à** $(1+2\varepsilon)T$ par pas de l
5. **Pour** $m_2 = 0$ **jusqu'à** $(1+2\varepsilon)T$ par pas de l
6. $E_c(j+1, m_1+p_{1j}, m_2) \leftarrow \min\{E_c(j+1, m_1+p_{1j}, m_2),\ E_c(j,m_1,m_2)+c_{1j}\}$
7. $E_c(j+1, m_1, m_2+p_{2j}) \leftarrow \min\{E_c(j+1, m_1, m_2+p_{2j}),\ E_c(j,m_1,m_2)+c_{2j}\}$
8. **Fin Pour**
9. Examiner tous les états obtenus qui ont un *makespan* compris entre T et $(1+2\epsilon)T$, et parmi ceux-ci retourner l'ordonnancement s_{dyna} qui a le plus petit coût.

TAB. 7.3 – L'algorithme de programmation dynamique $PD1_\varepsilon$. Dans la suite, on note $Dyna(T,\varepsilon) = s_{dyna}$.

7.2.2 Courbe de Pareto $(\varepsilon, 0)$-approchée

Soit $0 < \varepsilon \leq 1$. Dans la suite on suppose, sans perte de généralité, que $1/\varepsilon$ est un entier.

Afin d'obtenir un schéma d'approximation totalement polynomial, nous nous sommes inspirés de la méthode classique de Horowitz et Sahni [105]. On subdivise l'intervalle $]0,(1+2\varepsilon)T]$ en $n/\varepsilon + 2n$ sous-intervalles, $\mathcal{I}_k =](k-1)l, kl]$, $1 \leq k \leq n/\varepsilon + 2n$, chacun de longueur $l = \varepsilon T/n$. Pour un réel $x \in]0,(1+2\varepsilon)T]$, on note $[x]$ l'extrémité droite de l'intervalle auquel x appartient, i.e. $[x] = kl$, avec k tel que $x \in \mathcal{I}_k$. On pose $[0] = 0$.

L'algorithme approché consiste à appliquer l'algorithme exact $PD1$ précédent sur une instance modifiée dans laquelle chaque durée d'exécution p_{ij} est remplacée par $[p_{ij}]$. L'algorithme obtenu, noté $PD1_\varepsilon$, est donné dans le tableau 7.3.

Il est possible de montrer le théorème suivant :

Théorème 29 *[16] Pour une instance donnée du problème $Rm||C_{max}, \sum_j c_j$, étant donnés $T > 0$ et $\varepsilon > 0$, si il existe un ordonnancement avec un* makespan *compris entre T et $(1+\varepsilon)T$, alors l'ordonnancement s_{dyna} retourné par le programme dynamique $PD1_\varepsilon$ satisfait $(1-\varepsilon)T \leq T(s_{dyna}) \leq (1+2\varepsilon)T$ et $C(s_{dyna}) \leq \min_{t\in[T,(1+\varepsilon)T]} C_{opt}(t) \leq C_{opt}(T)$. De plus le temps d'exécution est $\mathcal{O}(n(n/\varepsilon)^m)$. Pour le problème où le rejet des tâches est autorisé, le temps d'exécution est $\mathcal{O}(n(n/\varepsilon)^{m+1})$.*

Dans la suite on note $Dyna(T,\varepsilon) = s_{dyna}$. On pose $T_{opt} = \min_s T(s)$, le minimum étant pris sur tous les ordonnancements s.

Remarque 2 *Une question intéressante est de considérer le cas $T = T_{opt}$ et de se demander s'il est nécessaire de connaître la valeur de T_{opt} pour obtenir un ordonnancement approché. En fait, nous avons montré que cela n'était pas le cas :*

Théorème 30 *[16] Sans connaître la valeur de T_{opt}, il est possible d'obtenir, en temps $\mathcal{O}((n/\varepsilon)^{m+1}\log m)$, un ordonnancement s^* tel que $T_{opt} \leq T(s^*) \leq (1+\varepsilon)T_{opt}$ et $C(s^*) \leq C_{opt}(T_{opt})$.*

1. Subdiviser l'intervalle $[D/m, \tilde{D}]$ en $k = \lceil \log(m\tilde{D}/D)/\log(1+\varepsilon) \rceil + 1 = \mathcal{O}(n/\varepsilon)$ sous-intervalles géométriques.
2. Appliquer l'algorithme $Dyna(T, \varepsilon)$ sur l'extrémité gauche de chacun de ces sous-intervalles pour obtenir une série d'ordonnancements.
3. Calculer $Dyna(\tilde{D}, \varepsilon)$.
4. Parmi les ordonnancements obtenus, retourner un sous-ensemble d'ordonnancement non dominés.

TAB. 7.4 – L'algorithme $PD1_{itéré}$.

Il est possible d'utiliser l'algorithme précédent afin d'obtenir une courbe de Pareto approchée. Soit $d_j = \min_i p_{ij}$ le temps minimum d'exécution de la tâche j et $D = \sum_j d_j$. Par la suite on suppose que $D > 0$. Soit M_j la machine sur laquelle la tâche j a le plus petit coût. On pose $\tilde{D} = \sum_j p_{M_j j}$. L'algorithme est donné dans le tableau 7.4.

Théorème 31 *[16] Le sous-ensemble retourné par l'algorithme $PD1_{itéré}$ constitue une courbe de Pareto $(\varepsilon, 0)$-approchée. De plus, il peut être obtenu en temps $\mathcal{O}(n(\frac{n}{\varepsilon})^{m+1})$. Dans le cas où il est possible de rejeter des tâches, le temps d'exécution est $\mathcal{O}(n(\frac{n}{\varepsilon})^{m+2})$.*

7.3 Fonctions objectifs linéaires

Il existe un résultat général, dû à Papadimitriou et Yannakakis, qui permet d'établir un lien entre le problème consistant à construire une courbe de Pareto ε-approchée pour un problème multicritère $\mathcal{A}$ comportant des fonctions objectifs linéaires, et la complexité d'un problème monocritère défini à partir de $\mathcal{A}$.

Définition 24 *Une fonction objectif est dite* linéaire *si elle peut s'exprimer comme un produit scalaire entre la solution s (exprimée comme un vecteur) et un vecteur coût (dont chaque composante est positive ou nulle) de même dimension que s.*

Dans la suite on fait l'hypothèse que toutes les composantes du vecteur s ont un coût borné polynomialement.

Définition 25 *La version* exacte *de $\mathcal{A}$ est le problème monocritère suivant : Étant donnés une instance I de $\mathcal{A}$, un entier B et une pondération des critères linéaires, i.e. une fonction objectif $\tilde{f} = \sum_{i=1}^{k} \lambda_i f_i$ (chaque fonction objectif f_i étant linéaire), déterminer s'il existe une solution s telle que la fonction objectif pondérée $\tilde{f}$ soit exactement égale à B pour cette solution, i.e. $\tilde{f}(s) = B$.*

Théorème 32 *[135] Sous l'hypothèse que toutes les fonctions objectifs sont linéaires, il existe un schéma d'approximation totalement polynomial pour construire une courbe de Pareto ε-approchée pour toute instance du problème $\mathcal{A}$, si il existe un algorithme pseudo-polynomial pour la version exacte de $\mathcal{A}$.*

Cette approche peut être appliquée aux problèmes d'ordonnancement multicritères dont les fonctions objectifs sont linéaires. Nous avons ainsi considéré le problème $P2||\sum w_j C_j, C_{max}$ (voir la section 2.2.4 page 29) et montré comment le théorème 32 pouvait être utilisé.

Application pour le problème $P2||\sum w_jC_j, C_{max}$
On ne considère que les solutions qui ne comportent aucun temps d'inactivité. Il est possible d'exprimer la somme pondérée des temps de complétude comme une fonction linéaire. Pour cela, nous introduisons des variables y_{ijm}, qui sont égales à 1 à chaque fois que les tâches i et j sont exécutées sur la même machine m et que la tâche i est exécutée avant la tâche j, et qui valent 0 sinon. Une solution s est donc représentée sous la forme d'un vecteur dont les composantes sont les variables y_{ijm}, et comme chaque composante vaut soit 0, soit 1, l'hypothèse faite plus haut qui stipulait que toutes les composantes du vecteur s ont un coût borné polynomialement est vérifiée. Nous avons alors $\sum w_jC_j = \sum_j p_jw_j + \sum_{ijm} p_iw_jy_{ijm}$, ce qui montre que le critère $\sum w_jC_j$ peut s'exprimer comme une fonction linéaire.

Bien que le *makespan* ne soit pas un critère linéaire, on peut considérer à la place les charges sur chacune des machines, i.e. on note l_i la date à laquelle se termine la dernière tâche affectée à la machine i. Il est facile de voir que la charge sur chacune des machines est un critère linéaire, et que de plus si on est capable de trouver une courbe de Pareto approchée pour le problème tricritère $P2||l_1, l_2, \sum w_jC_j$, alors on peut s'en servir pour déterminer une courbe de Pareto approchée pour le problème bicritère $P2||\sum w_jC_j, C_{max}$.

Il n'existe pas (à notre connaissance) d'algorithme pseudo-polynomial capable de résoudre la version exacte du problème $P2||l_1, l_2, \sum w_jC_j$. Mais il est possible, à l'aide d'un programme dynamique, d'obtenir un algorithme pseudo-polynomial pour une version exacte restreinte du problème $\mathcal{A}$ dans laquelle on ne considère que des solutions telles que sur chaque machine les tâches respectent la règle de Smith. Rappelons (voir la section 2.2.3) que selon cette règle, si les tâches sont numérotées de manière à vérifier $p_1/w_1 \leq p_2/w_2 \leq \ldots \leq p_n/w_n$, alors sur chaque machine les tâches qui ont le plus petit indice doivent être placées en premier. Cela ne pose pas de problème car toute solution ne respectant pas la règle de Smith est dominée par une solution qui respecte cette règle. On obtient donc le résultat suivant :

Théorème 33 *[17] Il existe un schéma d'approximation totalement polynomial pour calculer une courbe de Pareto ε-approchée pour toute instance du problème $P2||\sum w_jC_j, C_{max}$.*

Cependant cette approche ne fonctionne que pour les fonctions objectifs linéaires, or en ordonnancement on peut considérer des fonctions objectifs plus compliquées telles que $\sum C_j^2$ par exemple. On peut obtenir une courbe de Pareto approchée en utilisant des techniques d'arrondis classiques, comme celles utilisées dans la section 7.2. Dans la section suivante on explore cette voie.

7.4 Programmation dynamique et courbe de Pareto optimale

En nous basant sur le cas monocritère [173], nous avons formulé un programme dynamique générique pour les problèmes d'optimisation multiobjectifs.

Nous considérons un problème d'optimisation générique $\mathcal{A}$ avec γ critères. Chaque fonction objectif G_i, $i = 1, \ldots, \gamma$ est positive. Pour chaque instance I de $\mathcal{A}$, l'entrée peut être structurée sous la forme de n vecteurs $X_1, \ldots, X_n \in \mathbb{N}^\alpha$. Chaque vecteur X_k, $k = 1, \ldots, n$, consiste en α entiers positifs $[x_{k1}, x_{k2}, \ldots, x_{k\alpha}]$ codés sous forme binaire. Le nombre α est un entier positif qui ne dépend que de l'instance I.

Typiquement chaque vecteur X_i contient les informations au sujet d'une tâche, telles que sa durée, sa date d'introduction, etc. On suppose qu'il est possible d'ordonner les vecteurs X_i selon une certaine règle (par exemple SPT). Si $\bar{x} = \sum_{k=1}^{n} \sum_{i=1}^{\alpha} x_{ki}$, alors la taille de l'instance I est $\Theta(n + \log \bar{x})$.

PROGRAMME DYNAMIQUE (PD) :

1. Initialiser $\mathcal{S}_0$
2. **Pour** $k = 1$ **jusqu'à** n
3. $\quad\mathcal{S}_k := \emptyset$
4. $\quad$**Pour** chaque $\mathbb{S} \in \mathcal{S}_{k-1}$ et chaque $F \in \mathcal{F}$
5. $\quad\quad$ajouter $F(X_k, \mathbb{S})$ à $\mathcal{S}_k$
6. **Fin Pour**
7. Retourner le sous-ensemble $\mathcal{S}_n^*$ constitué de toutes les solutions non dominées de $\mathcal{S}_n$

TAB. 7.5 – Un algorithme de programmation dynamique (PD) générique.

L'algorithme de programmation dynamique pour le problème $\mathcal{A}$ est composé de n phases. Durant la k-ième phase, $k = 1, \ldots, n$, il traite l'entrée X_k et produit un ensemble d'états $\mathcal{S}_k$. Chaque état de $\mathcal{S}_k$ est un vecteur $\mathbb{S} = [s_1, \ldots, s_\beta, g_1, \ldots, g_\gamma] \in \mathbb{N}^{\beta+\gamma}$. Le nombre β est un entier positif qui ne dépend que de $\mathcal{A}$, et est donc indépendant de toute instance de $\mathcal{A}$. Les γ dernières composants sont les valeurs des γ fonctions objectifs sur cet état.

Durant la phase d'initialisation de l'algorithme de programmation dynamique (PD), l'ensemble d'états $\mathcal{S}_0$ est constitué d'un sous-ensemble fini de $\mathbb{N}^{\beta+\gamma}$. Durant la k-ième phase ($k = 1, \ldots, n$) du programme dynamique, l'ensemble d'états $\mathcal{S}_k$ est obtenu à partir de l'ensemble précédent d'états $\mathcal{S}_{k-1}$ de la manière suivante : $\mathbb{S}_k = \{F(X_k, \mathbb{S}) : F \in \mathcal{F}, \mathbb{S} \in \mathcal{S}_{k-1}\}$, où l'ensemble $\mathcal{F}$ est constitué d'un nombre fini de fonctions $\mathbb{N}^\alpha \times \mathbb{N}^{\beta+\gamma} \to \mathbb{N}^{\beta+\gamma}$.

Une solution *partielle* est une solution associée à un sous-ensemble des X_i. Le programme dynamique associe à chaque état $\mathbb{S} = [s_1, \ldots, s_\beta, g_1, \ldots, g_\gamma]$ qu'il calcule, une solution partielle s telle que $G_i(I, s) = g_i$, $1 \leq i \leq \gamma$. Afin de ne pas alourdir les notations, nous utiliserons parfois "solution $\mathbb{S}$" pour signifier "solution s".

Soit PD le programme dynamique générique représenté dans le tableau 7.5.

Une solution s de l'instance I est appelée une solution *potentielle* du programme dynamique PD, si il existe un état initial $\mathbb{S}_0 \in \mathcal{S}_0$, et une suite de n fonctions $F_{i_1}, \ldots, F_{i_n}$, avec $F_{i_l} \in \mathcal{F} = \{F_1, \ldots, F_{|\mathcal{F}|}\}$, tel que si on pose $\mathbb{S}_l = F_{i_l}(X_l, \mathbb{S}_{l-1})$ $(1 \leq l \leq n)$ alors s est la solution associée à $\mathbb{S}_n$. Soit $\mathcal{PS}$ l'ensemble des solutions potentielles. Comme $\mathcal{S}_k$ est un ensemble, il n'y a pas de copies multiples (si deux solutions différentes ont le même état, le programme dynamique n'en conserve qu'une seule), et par conséquent $\mathcal{S}_n \subseteq \mathcal{PS}$.

Condition 1 *Étant donnée une instance I de $\mathcal{A}$, pour toute solution s de I, il existe une solution potentielle $s^* \in \mathcal{PS}$ telle que $G_i(I, s) \geq G_i(I, s^*)$ pour tout $i = 1, \ldots, \gamma$.*

Cette condition implique directement le résultat suivant :

Théorème 34 *[17] Si la condition 1 est satisfaite alors, pour toute instance I du problème $\mathcal{A}$, le programme dynamique* PD *retourne une courbe de Pareto optimale $P(I)$.*

7.5 Programmation dynamique arrondie et courbe de Pareto approchée

Nous avons proposé une généralisation de la notion de $[D, \Delta]$-proximité, introduite par Woeginger, afin d'obtenir des conditions suffisantes pour l'existence d'un schéma d'approximation totalement polynomial calculant une courbe de Pareto approchée pour des problème d'ordonnancement variés.

Nous supposons qu'il existe un vecteur $D = [d_1, \ldots, d_\beta, d_{\beta+1}, \ldots, d_{\beta+\gamma}] \in \mathbb{N}^{\beta+\gamma}$, que nous appelons par la suite le vecteur *degré*. Ce vecteur degré dépend du problème $\mathcal{A}$ et du programme dynamique, mais ne dépend pas de l'instance de $\mathcal{A}$ considérée.

Définition 26 *Pour un réel $\Delta > 1$ et deux vecteurs $\mathbb{S}, \mathbb{S}' \in I\!N^{\beta+\gamma}$ avec $\mathbb{S} = [s_1, \ldots, s_{\beta+\gamma}]$ et $\mathbb{S}' = [s'_1, \ldots, s'_{\beta+\gamma}]$, on dit que $\mathbb{S}$ est $[D, \Delta]$-proche de $\mathbb{S}'$ si $\Delta^{-d_l} s_l \leq s'_l \leq \Delta^{d_l} s_l$, pour $l = 1, \ldots, \beta+\gamma$.*

Il s'agit d'une relation symétrique.

Condition 2 *(sur les fonctions de $\mathcal{F}$)*
Pour tout $\Delta > 1$, $F \in \mathcal{F}$, $X \in I\!N^\alpha$, et pour tout $\mathbb{S}, \mathbb{S}' \in I\!N^{\beta+\gamma}$, la condition suivante est respectée : Si $\mathbb{S}$ est $[D, \Delta]$-proche de $\mathbb{S}'$, alors $F(X, \mathbb{S})$ est $[D, \Delta]$-proche de $F(X, \mathbb{S}')$.

Condition 3 *(conditions techniques) Elles ne sont pas énoncées ici, mais il s'agit de conditions qui sont généralement vérifiées dans un algorithme de programmation dynamique, et qui assurent entre autres que l'ensemble des états $\mathcal{S}_k$ n'augmente pas trop vite en taille.*

Définition 27 *Un problème d'optimisation $\mathcal{A}$ est dit* ex-DPM-benevolent *si il existe un vecteur degré D tel que le programme dynamique PD satisfait les conditions 1 à 3.*

Théorème 35 *[17] Si le problème $\mathcal{A}$ est un problème d'optimisation multiobjectif* ex-DPM-benevolent, *alors il existe un schéma d'approximation totalement polynomial capable de construire une courbe de Pareto ε-approchée, pour toute instance de $\mathcal{A}$.*

La preuve de ce théorème s'appuie sur une technique, déjà utilisée par Ibarra et Kim [109], qui consiste, en fonction du paramètre Δ de la condition 2, à subdiviser de manière géométrique l'espace des états en "boîtes". On modifie ensuite l'algorithme de programmation dynamique PD de manière à ne retenir, pendant chacune des étapes, qu'une seule solution potentielle à l'intérieure de chacune des "boîtes".

Parmi les conditions indiquées plus haut, c'est la condition 2 qui est généralement la plus difficile à vérifier.

Il est possible de considérer également les courbes de Pareto $(\varepsilon, 0)$-approchées.

Rappelons que la composante g_i du vecteur état $\mathbb{S} = [s_1, \ldots, s_\beta, g_1, \ldots g_\gamma]$ est la valeur de la fonction G_i $(1 \leq i \leq \gamma)$ sur cet état. Étant donnés deux états $\mathbb{S}$ et $\mathbb{S}'$, on écrit $\mathbb{S}|_{g_i} = \mathbb{S}'|_{g_i}$ si ces deux états ont la même valeur sur la composante g_i. On note par $F|_{g_i}(X, \mathbb{S})$ la valeur de g_i que l'on obtient sur le prochain état obtenu à l'aide de la fonction F appliquée sur l'état $\mathbb{S}$ et l'entrée X.

Définition 28 *Le critère G_i est dit* indépendant *si pour tout $F \in \mathcal{F}$, et pour tous états $\mathbb{S}, \mathbb{S}'$, on a $F|_{g_i}(X, \mathbb{S}) = F|_{g_i}(X, \mathbb{S}')$ à chaque fois que $\mathbb{S}|_{g_i} = \mathbb{S}'|_{g_i}$.*

Il est possible d'affiner le théorème 35 pour tenir compte du fait qu'un critère est indépendant.

Théorème 36 *[17] Si $\mathcal{A}$ est un problème d'optimisation multiobjectif* ex-DPM-benevolent, *avec un critère G_γ indépendant, alors il existe un schéma d'approximation totalement polynomial pour construire une courbe de Pareto $(\varepsilon, 0)$-approchée.*

7.6 Exemples d'application à des problèmes d'ordonnancement

Dans cette section, nous illustrons les théorèmes 35 et 36 sur quelques problèmes d'ordonnancement. Parmi les problèmes évoqués ci-dessous, Woeginger avait déjà considéré dans [173] les problèmes monocritères suivants : $P2||C_{max}$, $P2||\sum C_j^3$ et $P2||\sum w_jC_j$, par contre il n'avait pas considéré des problèmes d'ordonnancement à base de lots.

7.6.1 Le problème $P2||C_{max}, \sum C_j^2$

Nous décrivons le programme dynamique, noté PD_{ex1}, pour ce problème. Les tâches sont numérotées selon l'ordre SPT, i.e. $p_1 \leq \ldots \leq p_n$. Il est en effet possible de montrer que les seules ordonnancements qu'il est intéressant de considérer sont ceux pour lesquels, sur chaque machine, les tâches sont ordonnancées en respectant cet ordre. On a $\alpha = 1$, $\beta = 2$, et $\gamma = 2$. Pour $k = 1, \ldots, n$ on a $X_k = [p_k]$. Un état $\mathbb{S} = [s_1, s_2, g_1, g_2]$ de $\mathcal{S}_k$ représente une solution partielle (sans temps d'inactivité) pour les k premières tâches : s_1 (respectivement s_2) est le temps d'exécution total sur la première (respectivement seconde) machine, et g_1, g_2 sont les valeurs des fonctions objectifs pour l'ordonnancement partiel. On pose $F_1(p_k, s_1, s_2, g_1, g_2) = [s_1 + p_k, s_2, \max\{s_1 + p_k, s_2\}, g_2 + (s_1 + p_k)^2]$, et $F_2(p_k, s_1, s_2, g_1, g_2) = [s_1, s_2 + p_k, \max\{s_1, s_2 + p_k\}, g_2 + (s_2 + p_k)^2]$. La fonction F_1 (respectivement F_2) calcule le prochain état lorsque la k-ième tâche va sur la première (respectivement deuxième) machine. Enfin on a $\mathcal{F} = \{F_1, F_2\}$ et $\mathcal{S}_0 = \{[0, 0, 0, 0]\}$. On peut montrer que ce programme dynamique satisfait la condition 1. En utilisant le théorème 34 on obtient le corollaire suivant :

Corollaire 9 *Le programme dynamique PD_{ex1} retourne l'ensemble des solutions Pareto optimales $P(I)$ pour le problème $P2||C_{max}, \sum C_j^2$.*

Il est possible de montrer qu'avec le vecteur degré $D = [1, 1, 1, 2]$ la condition 2 est satisfaite, et comme la condition 3 l'est également, on obtient en utilisant le théorème 35 le corollaire suivant :

Corollaire 10 *Pour le problème $P2||C_{max}, \sum C_j^2$, pour toute instance I, il est possible de construire une courbe de Pareto ε-approchée $P_\varepsilon(I)$ en temps polynomial par rapport à $|I|$ et $1/\varepsilon$.*

Avec une approche similaire, on peut obtenir des schémas d'approximation totalement polynomiaux pour une combinaison de plusieurs critères tels que : l_i : la charge de la machine i, C_{max} : le *makespan*, $\sum c_{ij}$: le coût total de l'ordonnancement, $\sum C_j^\lambda$: la somme des temps de complétude à la puissance λ, $\sum w_jC_j$: la somme pondérée des temps de complétude. Ceci, parce que dans le cas de machines identiques où reliées (respectivement non reliées), quelques uns de ces critères (voir le tableau 7.6 (respectivement le tableau 7.7)) sont compatibles entres eux, i.e. il existe une permutation unique des vecteurs X_i qui permet à l'aide d'un programme dynamique de trouver une solution optimale pour chacun des critères. Par contre, comme indiqué dans le tableau 7.6, les critères $\sum w_jC_j$ et $\sum w'_jC_j$ ne sont généralement pas compatibles car la règle de Smith pour chacun de ces critères n'ordonne pas les tâches dans le même ordre. Ces résultats restent valides pour un nombre constant de machines.

Remarquons que nous pouvons obtenir une courbe de Pareto $(\varepsilon, 0)$-approchée pour les critères l_i et $\sum c_{ij}$ qui sont indépendants au sens de la définition 28.

7.6.2 Le problème $1(batch, C_{batch}, p_{max})||\Sigma w_jC_j, C_{max}$

Ce problème a été défini dans la section 2.2.3. Par la suite nous supposons que les tâches ont été indexées de manière à respecter l'ordre SPT, c'est-à-dire $p_1 \leq \ldots \leq p_n$.

Qm	l_i	C_{max}	$\sum c_{ij}$	$\sum C_j^\lambda$	$\sum w_j C_j$
l_i	+	+	+	+	+
C_{max}	+	+	+	+	+
$\sum c_{ij}$	+	+	+	+	+
$\sum C_j^\lambda$	+	+	+	+	−
$\sum w_j C_j$	+	+	+	−	−

TAB. 7.6 – Quelques critères compatibles (indiqués avec des +) dans le cas de machines identiques ou reliées.

Rm	l_i	C_{max}	$\sum c_{ij}$
l_i	+	+	+
C_{max}	+	+	+
$\sum c_{ij}$	+	+	+

TAB. 7.7 – Quelques critères compatibles (indiqués avec des +) dans le cas de machines non reliées.

Ce problème respecte la propriété suivante qui sert de base à un algorithme de programmation dynamique. Rappelons qu'un critère est dit *régulier* si il est non décroissant en fonction du temps de complétude des tâches.

Lemme 2 *[46] Pour toute fonction objectif régulière, il existe un ordonnancement optimal* $(B_{i_1}, \ldots, B_{i_k})$*, tel que* $B_{i_1} = \{1, 2, \ldots, i_1\}, B_{i_2} = \{i_1 + 1, i_1 + 2, \ldots, i_2\}, \ldots, B_{i_k} = \{i_{k-1} + 1, i_{k-1} + 2, \ldots, i_k\}$*, avec* $i_k = n$*.*

Ce lemme montre qu'un ordonnancement optimal peut être spécifié en indiquant uniquement l'ensemble des tâches $i_1, i_2, \ldots i_k = n$. Tout ordonnancement de ce type sera appelé un ordonnancement de type SPT.

Pour la version non pondérée dans laquelle tous les w_j sont égaux à 1, Brucker et al. [46] ont proposé des algorithmes polynomiaux pour les problèmes monocritères avec les critères suivants : le nombre de tâches en retards, la somme pondérée des temps de complétude, le retard (*lateness*) maximum d'une tâche.

Remarquons que le problème où l'on cherche à minimiser le *makespan* C_{max} peut être résolu de manière triviale en mettant toutes les tâches dans un seul lot.

La situation avec deux critères est nettement plus difficile. Nous avons ainsi montré, en considérant une instance particulière, que le nombre de solutions Pareto optimales pouvait ne pas être polynomialement bornée lorsqu'on considère les critères C_{max} et $\sum_j C_j$ [17].

Nous avons considéré la version générale pondérée et nous avons proposé un programme dynamique, noté PD_{ex2}, pour ce problème.

On pose $\alpha = 2$, $\beta = 1$ et $\gamma = 2$. Pour $k = 1, \ldots, n$ on définit $X_k = [p_k, w_k]$. Un état $\mathbb{S} = [s_1, g_1, g_2]$ dans $\mathcal{S}_k$ représente un ordonnancement partiel (sans temps d'inactivité) pour les k premières tâches : s_1 est le poids total des tâches qui n'ont pas encore été affectées à un lot, g_1 est la somme pondérée des temps de complétude des k premières tâches et g_2 est le temps d'exécution total de l'ordonnancement partiel. On définit $F_1(p_k, w_k, s_1, g_1, g_2) = [0, g_1 + (s_1 + w_k)(g_2 + p_k), g_2 + p_k]$, et $F_2(p_k, w_k, s_1, g_1, g_2) = [s_1 + w_k, g_1, g_2]$.

La fonction F_1 calcule le prochain état lorsque un nouveau lot est créé à la suite de l'ordonnancement partiel et que toutes les tâches non encore affectées sont placées à l'intérieur de ce nouveau lot avec la k-ième tâche. Comme les tâches sont triées selon l'ordre SPT, la longueur de ce nouveau lot est égale à p_k. La fonction F_2 calcule le prochain état lorsque on décide d'ordonnancer la k-ième tâche plus tard. Cette tâche sera ordonnancée dans un lot lorsque la fonction F_1 sera appelée. On pose $\mathcal{F} = \{F_1, F_2\}$ et $\mathcal{S}_0 = \{[0, 0, 0]\}$. La sortie du programme dynamique est l'ensemble des vecteurs non dominés $\mathbb{S} \in \mathcal{S}_n$ avec $s_1 = 0$.

Il est facile de montrer que ce programme dynamique peut générer tout ordonnancement de type SPT. Par conséquent le lemme 2 implique que la condition 1 est satisfaite et on peut donc déduire à l'aide du théorème 34 le résultat suivant :

Corollaire 11 *[17] Le programme dynamique PD_{ex2} retourne l'ensemble des solutions Pareto optimales $P(I)$ pour toute instance du problème $1(batch, C_{batch}, p_{max})||\Sigma w_j C_j, C_{max}$.*

On peut montrer que la condition 3 est vérifiée, de même que la condition 2 avec le vecteur degré $D = [1, 2, 1]$. Le *makespan* est un critère indépendant pour le programme dynamique PD_{ex2}, par conséquent à l'aide du théorème 36 on obtient le résultat suivant :

Corollaire 12 *[17] Pour le problème $1(batch, C_{batch}, p_{max})||\Sigma w_j C_j, C_{max}$, pour toute instance I, il est possible de construire une courbe de Pareto $(\varepsilon, 0)$-approchée $P_\varepsilon(I)$ en temps polynomial par rapport à $|I|$ et $1/\varepsilon$.*

7.7 Conclusion

En développant un algorithme ad hoc, nous avons montré qu'il existe un schéma d'approximation totalement polynomial pour calculer une courbe de Pareto $(\varepsilon, 0)$-approchée pour toute instance du problème $Rm||C_{max}, \sum_j c_j$.

Nous avons ensuite cherché à obtenir des résultats génériques d'existence de tels schémas d'approximation. Un premier résultat, dû à Papadimitriou et Yannakakis, permet de prendre en compte les problèmes d'optimisation multicritères pour lesquels chaque fonction objectif est linéaire. Nous avons montré que le problème $P2||\sum w_j C_j, C_{max}$ rentrait dans ce cadre et que par conséquent on pouvait obtenir un schéma d'approximation totalement polynomial pour calculer une courbe de Pareto ε-approchée.

Nous avons ensuite présenté une extension de l'approche proposée par Woeginger [173], dans le cadre de l'optimisation monocritère, pour les problèmes d'optimisation multicritères et tout particulièrement les problèmes d'ordonnancement. Nous avons ainsi montré qu'une large famille de problèmes d'ordonnancement bicritères admettait des schéma d'approximation totalement polynomiaux pour calculer des courbes de Pareto ε et/ou $(\varepsilon, 0)$-approchées. Il serait intéressant de voir si d'autres types de problèmes d'optimisation combinatoire multicritères peuvent être abordés ce cette manière.

Il est intéressant de relever que le problème d'ordonnancement sur une seule machine $1||\sum w_j C_j, \sum w'_j C_j$ a été traité de deux manières différentes : avec l'approche point idéal dans le chapitre 5 et avec l'approche courbe de Pareto approchée dans le chapitre 6. Il serait intéressant d'obtenir des résultats similaires dans le cas de m machines.

Quatrième partie

Mécanismes

Chapitre 8

Recherche de solutions stables pour l'ordonnancement de tâches sur deux machines parallèles

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet de la publication [21].

8.1 Introduction

Nous nous situons dans le contexte de la théorie des jeux non coopératifs. Nous recherchons des solutions offrant un bon compromis entre leur degré de stabilité et leur qualité. Cette problématique, évoquée à la section 1.3.1, est mise en application ici sur un problème classique d'ordonnancement.

Nous considérons un réseau composé de m liens parallèles et un ensemble de n agents. Chaque agent a une certaine quantité de trafic qu'il doit router sur l'un des m liens. De manière équivalente on peut voir ce problème sous la forme d'un problème d'ordonnancement, en faisant correspondre à chaque lien une machine (ou processeur), et à chaque agent une tâche. Chaque agent cherche à maximiser son propre profit. Il existe 2 principaux modèles selon la manière dont est calculé ce profit :

- Dans le modèle KP [119], le *profit* d'une tâche est l'inverse du temps de complétude de la machine sur laquelle elle est ordonnancée. On appelle temps de complétude d'une machine, la date à laquelle se termine la dernière tâche ordonnancée sur cette machine.
- Dans le modèle CKN [57], le *profit* d'une tâche est l'inverse de son temps de complétude.

Dans les 2 cas, la fonction objectif globale est le *makespan* de l'ordonnancement (date à laquelle la dernière tâche se termine).

Par rapport au problème bien connu $Pm||C_{max}$, défini à la section 2.2.4, on cherche à obtenir un ordonnancement de bonne qualité pour le *makespan*, et qui soit en même temps *stable*, c'est-à-dire tel qu'aucune tâche ne soit incitée à changer de machine. Plus précisément, on suppose que

chaque machine utilise une politique locale, connue de tous, afin d'ordonnancer les tâches qui lui sont affectées. Si une tâche décide de quitter la machine sur laquelle elle est actuellement ordonnancée pour aller sur une autre machine, elle sait qu'elle sera ordonnancée selon la politique de cette dernière, et elle peut par conséquent calculer son nouveau temps de complétude. Dans ce modèle, une solution stable correspond donc à un équilibre de Nash, et le problème revient à chercher parmi les équilibres de Nash celui ayant le plus petit *makespan*.

Dans le modèle KP il existe toujours un équilibre de Nash (pur) qui est optimal vis à vis du *makespan*, et par conséquent le prix de la stabilité pour ce modèle est égal à 1. Ce résultat est une conséquence directe du résultat suivant : il est toujours possible dans le modèle KP de transformer une solution initiale quelconque en une solution qui soit un équilibre de Nash (pur) sans augmenter la valeur du *makespan* de cette solution [83].

Il est naturel de se poser la question de savoir si un tel résultat existe pour le modèle CKN. Le prix de la stabilité dépend de la politique locale utilisée par chacune des machines. Nous avons supposé que chaque machine ordonnance les tâches qui lui sont affectées selon la politique LPT, i.e. chaque machine commence par ordonnancer les plus longues tâches d'abord. Contrairement à ce qui se passe pour le modèle KP, il n'existe pas toujours une solution optimale qui est un équilibre de Nash. En effet, il n'est pas difficile de voir qu'il existe un unique équilibre de Nash (pur) et que ce dernier peut être obtenu en utilisant l'algorithme de liste LPT. On en déduit que le prix de la stabilité est égal au rapport d'approximation de l'algorithme de liste LPT vis à vis du *makespan*, à savoir $4/3 - 1/(3m)$ [90] (voir la section 2.2.4).

Une manière d'améliorer ce rapport consiste à considérer des équilibres de Nash α-approchés au lieu d'équilibres de Nash exacts.

Exemple 11 *Considérons 2 machines (chacune utilisant la politique LPT) et les tâches suivantes : 2 tâches t_1 et t_2 de longueur 3 et 3 tâches t_3, t_4 et t_5 de longueur 2. Le seul équilibre de Nash (pur) est l'ordonnancement dans lequel les tâches de longueur 3 sont ordonnancées au temps 0, et sont suivies par les tâches de longueur 2. Cette solution a un* makespan *égal à 7, tandis que la solution optimale qui a un* makespan *égal à 6 est un équilibre de Nash 2-approché.*

Notre contribution concerne l'étude du compromis qu'il est possible d'atteindre entre le rapport d'approximation, vis à vis du *makespan*, et la valeur de α, vis à vis d'un équilibre de Nash α-approché, dans le cas de $m = 2$ machines. Nous avons proposé un algorithme qui fournit une solution $\frac{8}{7}$-approchée pour le *makespan* et qui est en même temps un équilibre de Nash 3-approché. De plus, nous avons montré qu'en modifiant légèrement le schéma d'approximation polynomial de Graham (voir la section 2.2.4) on pouvait obtenir un ordonnancement avec un *makespan* arbitrairement proche de l'optimum et qui soit un équilibre de Nash α-approché avec α une constante. Enfin nous avons obtenu des bornes liant α et le meilleur rapport d'approximation qu'il est possible d'atteindre.

8.2 Une variante de LPT

Soient $\{t_1, \ldots, t_n\}$ les n tâches qui doivent être ordonnancées sur les 2 machines identiques. Nous notons par P_1 la machine avec la charge maximale, et par P_2 l'autre machine. Soit n_i le nombre de tâches ordonnancées sur P_i. Soit x_i la i-ième tâche sur P_1, et y_i la i-ième tâche sur P_2. On note par $l(t_i)$ le temps d'exécution (où durée) de la tâche t_i.

Nous représentons un ordonnancement par 2 ensembles A et B, avec A (respectivement B) qui est l'ensemble des tâches ordonnancées sur P_1 (respectivement P_2). Sur chaque machine, les tâches sont ordonnancées selon l'ordre LPT. Soit $\xi = (A, B)$ un ordonnancement, et soit a (respectivement b) un

Ordonner les tâches selon l'algorithme de liste LPT. Soit LPT l'ordonnancement ainsi obtenu.
Soit $S = \sum_{i=1}^{n} l(t_i)$.
Si $n_2 \geq 2$ et $((n_1 = 3$ et $l(x_1) + l(x_2) + l(x_3) > (\frac{4}{7})S)$ ou $(n_1 = 4))$
  Si $n_1 = 3$
    Soit $\xi_1 = swap(LPT, \{x_1\} \leftrightarrow \{y_2\})$,
    $\xi_2 = swap(LPT, \{x_2\} \leftrightarrow \{y_2\})$ et
    $\xi_3 = swap(LPT, \{x_1\} \leftrightarrow \{y_1\})$.
    Retourner l'ordonnancement qui a le plus petit *makespan* parmi LPT, ξ_1, ξ_2 et ξ_3.
  Si $n_1 = 4$
    Soit $\xi_4 = swap(LPT, \{x_3, x_4\} \leftrightarrow \{y_2\})$.
    Retourner l'ordonnancement qui a le plus petit *makespan* parmi LPT et ξ_4.
Sinon Retourner LPT.

TAB. 8.1 – L'algorithme LPT_{swap}.

sous-ensemble de tâches ordonnancées sur P_1 (respectivement P_2). Nous notons par $swap(\xi, a \leftrightarrow b)$ l'ordonnancement $((A \setminus a) \cup b, (B \setminus b) \cup a)$. Dans cet ordonnancement, chaque processeur exécute ses tâches selon la politique LPT (si 2 tâches ont la même longueur, celle ayant le numéro d'identification le plus petit est ordonnancée en premier).

On considère l'algorithme noté LPT_{swap} et décrit dans le tableau 8.1. Nous avons montré les résultats suivants :

Théorème 37 *[21] Pour $m = 2$ machines, l'algorithme LPT_{swap} est $\frac{8}{7}$-approché vis à vis du* makespan.

Théorème 38 *[21] L'ordonnancement retourné par LPT_{swap}, sur deux machines dont les politiques sont LPT, est un équilibre de Nash 3-approché.*

À partir des 2 précédents théorèmes, on en déduit le corollaire suivant :

Corollaire 13 *Dans le cas de 2 machines ayant chacune une politique LPT, le prix de la stabilité α-approchée est au plus $\frac{8}{7}$, pour tout $\alpha \geq 3$.*

Nous avons montré également le résultat suivant :

Théorème 39 *[21] Soit $\varepsilon > 0$. Si les politiques des processeurs sont LPT, alors le prix de la stabilité α-approchée est au moins $\frac{8}{7}$, pour tout $\alpha \leq 2.1 - \varepsilon$.*

Une question intéressante concerne la relation entre le rapport d'approximation et la valeur de α, en particulier existe-t'il des algorithmes avec un rapport d'approximation plus petit et qui retournent tout de même des équilibres de Nash approchés ? La section suivante répond à cette question.

8.3 Une variante de l'algorithme de Graham

Nous avons considéré l'algorithme représenté dans le tableau 8.2, et qui s'inspire de l'algorithme de Graham [90] légèrement modifié. Cet algorithme, noté par la suite OPT-LPT(k), est un schéma d'approximation polynomial, et son rapport d'approximation est égal à $1+\varepsilon$, avec $\varepsilon = \frac{1}{2+2\lfloor \frac{k}{2} \rfloor}$, sous la condition que les k tâches les plus longues soient ordonnancées de manière optimale [90].

(i) Soit k une constante (entière).
(ii) Obtenir un ordonnancement optimal pour les k plus longues tâches, tel que :
- Une fois que les tâches ont été affectées aux machines, elles sont ordonnancées sur chaque machine selon l'ordre LPT.
- Si deux tâches ont la même longueur, celle qui a le numéro d'identification le plus petit est ordonnancée en premier.

(iii) Ordonnancer les $n-k$ tâches restantes en utilisant la règle LPT.

TAB. 8.2 – L'algorithme OPT-LPT(k).

Théorème 40 *[21] L'algorithme OPT-LPT(k) produit un ordonnancement qui est un équilibre de Nash α-approché, avec $\alpha < k-2$.*

Théorème 41 *[21] Soit ε tel que $0 < \varepsilon < 1$ et $k \geq 5$. L'algorithme OPT-LPT(k) peut retourner des ordonnancements qui sont des équilibres de Nash α-approchés avec $\alpha \geq k-2-\varepsilon$.*

À partir du théorème 40, et en utilisant le fait que le rapport d'approximation de l'algorithme OPT-LPT(k) est $\frac{1}{2+2\lfloor\frac{k}{2}\rfloor}$, on peut en déduire le résultat suivant :

Corollaire 14 *Soit $k \geq 5$ un entier. Dans le cas de 2 machines, si la politique de chaque machine est LPT, le prix de la stabilité α-approchée est au plus $1+\varepsilon$, avec $\varepsilon = \frac{1}{4+2\lfloor\frac{k}{2}\rfloor} < \frac{1}{k}$, pour tout $\alpha \geq k$.*

Si l'on recherche un algorithme $\frac{8}{7}$-approché et qui retourne une solution aussi stable que possible, alors l'algorithme LPT_{swap} est meilleur que l'algorithme OPT-LPT(k). En effet la solution retournée par LPT_{swap} est trouvée plus rapidement (pour OPT-LPT(k) on a 64 ordonnancements à comparer tandis que pour LPT_{swap} il y a au plus 4 ordonnancements à comparer), et OPT-LPT(k) peut retourner une solution qui est un équilibre de Nash 4-approché (à comparer avec un équilibre de Nash 3-approché pour LPT_{swap}). D'un autre côté, l'algorithme OPT-LPT(k) est utile pour obtenir des rapports d'approximations plus faibles.

8.4 Compromis : stabilité et rapport d'approximation

Nous avons montré que si l'on voulait obtenir un prix de la stabilité α-approchée inférieur ou égal à $(1+\varepsilon)$, alors α devait être au moins de l'ordre $\Theta(\varepsilon^{-1/2})$.

Théorème 42 *[21] Soit $\varepsilon > 0$ et $k > 0$ tels que $\varepsilon = \frac{1}{k(k+1)}$. Pour obtenir un prix de la stabilité α-approchée plus petit que $1+\varepsilon$, il est nécessaire d'avoir $\alpha \geq k$.*

Théorème 43 *[21] Le prix de la stabilité α-approchée est au moins 1.1 (respectivement $\frac{9}{8}$), pour tout $\alpha \leq \frac{10}{3}$ (respectivement $\alpha \leq \frac{8}{3}$).*

La figure 8.1 illustre le compromis entre le rapport d'approximation de tout algorithme et le degré de stabilité d'un ordonnancement qu'il peut retourner. La zone grise sombre illustre les résultats du théorème 42 (pour $2 \leq k \leq 10$), et des théorèmes 39 et 43. Chaque point (x, α) qui appartient à la zone grise sombre indique un résultat négatif, à savoir qu'il n'existe aucun algorithme avec

un rapport d'approximation x et qui retourne des solutions qui soient des équilibres de Nash α-approchés. La zone grise claire illustre les résultats du théorème 40 et du corollaire 13 : chaque point (x, α) appartenant à cette zone signifie qu'il existe un algorithme (soit OPT-LPT(k), soit LPT_{swap}) x-approché qui retourne des équilibres de Nash α-approchés .

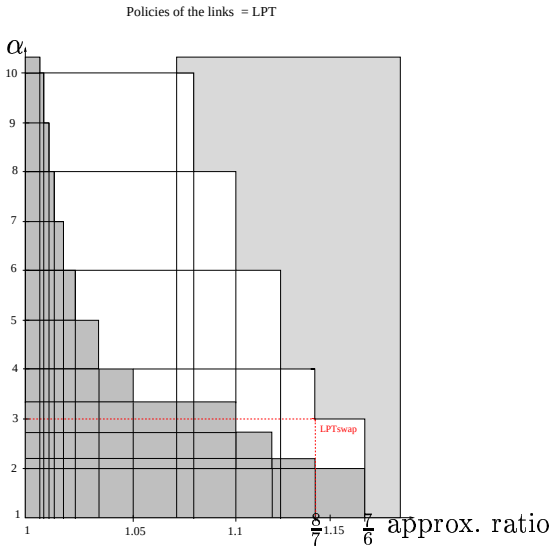


FIG. 8.1 – *Gris clair* (respectivement *Gris sombre*) : Relation entre α et le rapport d'approximation qu'il est possible (respectivement pas possible) d'obtenir si un algorithme retourne un équilibre de Nash α-approché.

8.5 Conclusion

Nous avons établi des relations entre le degré de stabilité α et la valeur du rapport d'approximation pour le *makespan*. En considérant que la politique de chaque lien était LPT, nous avons montré que le prix de la stabilité α-approchée était au moins $8/7$ pour tout $\alpha < 2.1$, et au plus $8/7$ pour tout $\alpha \geq 3$. Nous avons proposé un algorithme qui dans le cas de 2 machines garantit un rapport d'approximation égal à $8/7$ pour le *makespan* et retourne une solution qui est un équilibre de Nash 3-approché. Nous avons montré que le prix de la stabilité α-approchée était supérieur ou égal à $1 + \varepsilon$ pour tout $\alpha \leq k$, où k est une constante en $\Theta(\varepsilon^{-1/2})$, et qu'il était inférieur ou égal à $1 + \varepsilon$ pour tout $\alpha \geq \frac{1}{\varepsilon}$.

Nous avons étudié le prix de la stabilité approchée dans le cas de deux machines. Il pourrait être intéressant de généraliser ces résultats au cas de $m > 2$ machines. On peut montrer que dans ce cas, l'algorithme OPT-LPT(k) fournit des équilibres de Nash $(k - m + 1)$-approchés. De plus, résultats d'impossibilité que nous avons obtenu dans le cas de deux machines sont également valables dans le cas de $m > 2$ machines.

Une question intéressante est de savoir s'il existe des politiques déterministes meilleures que LPT vis à vis du prix de la stabilité approchée. On peut montrer que la politique SPT n'apporte rien par rapport à la politique LPT. Plus précisément, pour un rapport d'approximation donné, le degré de stabilité d'une solution assurant ce rapport, dans le cas où la politique des machines est SPT, est toujours plus mauvais que le degré de stabilité d'une solution (que l'on peut obtenir à l'aide des algorithmes que nous avons proposés) assurant ce rapport, dans le cas où la politique des

machines est LPT. L'utilisation d'une politique probabiliste sur chaque machine est en revanche intéressante [138].

Des algorithmes de recherche locale ont été utilisés pour le problème $P||C_{max}$ [47, 48, 156]. Schuurman et Vredeveld [156] ont étudié les rapports d'approximation de différents algorithmes de recherche locale. À notre connaissance, le meilleur algorithme de recherche locale connu pour le problème $P||C_{max}$ utilise le voisinage *push* [156], et admet un rapport d'approximation égal à 8/7. Cependant le nombre de mouvements à effectuer avant d'atteindre un optimum local n'est pas nécessairement polynomial. L'algorithme LPT_{swap} peut être considéré comme un algorithme de recherche locale qui commence à être appliqué non pas sur une solution initiale quelconque, mais sur une solution obtenue par l'algorithme de liste LPT.

Chapitre 9

Algorithmes avec véracité garantie pour l'ordonnancement de tâches sur des machines parallèles

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet de la publication [20].

9.1 Introduction

Dans ce chapitre nous considérons le problème d'ordonnancement $P||C_{max}$, qui rappelons le (voir la section 2.2.4), consiste à ordonnancer un ensemble de tâches sur des machines parallèles, de manière à minimiser le *makespan*. Le contexte dans lequel nous étudions ce problème est celui de la théorie des jeux, et plus précisément celui de la recherche d'algorithmes ou de mécanismes, avec véracité garantie (voir la section 1.3.2). Nous sommes donc en présence de m machines identiques et de n tâches (agents) ayant chacune une durée d'exécution. Chaque tâche cherche à minimiser son temps de complétude. La durée réelle de chaque tâche, n'est connue que d'elle seule.

L'étude de problèmes d'ordonnancement avec des agents "égoïstes" s'est intensifiée ces dernières années. Depuis les travaux de Nisan et Ronen [130] de nombreux articles ont vu le jour [3, 31, 33, 35]. Cependant, tous ces travaux considèrent que les agents sont les machines, alors que dans notre cas se sont les tâches qui sont les agents. Christodoulou et al. [57] ont considéré le même modèle, mais uniquement dans le contexte des mécanismes de coordination. Ils ont proposé différents mécanismes de coordination, avec un prix de l'anarchie meilleur que celui de SPT, mais ces mécanismes ne sont pas avec véracité garantie.

Si on suppose que chaque tâche ne peut pas déclarer une valeur plus petite que sa valeur réelle, alors un algorithme avec véracité garantie peut facilement être obtenu dans le contexte centralisé en ordonnançant chacune des tâches avec la politique SPT. Le rapport d'approximation de cet

algorithme est égal à $(2 - 1/m)$, où m est le nombre de machines. De plus, cet algorithme peut être facilement transformé en un mécanisme de coordination. Nous avons cherché à savoir si on pouvait trouver d'autres algorithmes ou mécanismes avec véracité garantie, avec un meilleur rapport d'approximation.

Nous avons considéré, comme dans [31], des algorithmes ou mécanismes probabilistes. Ainsi l'on suppose que chaque tâche cherche à maximiser l'espérance de son profit, et l'on dit d'un algorithme probabiliste qu'il est à véracité garantie si, pour chaque agent, la stratégie qui consiste à déclarer sa vraie durée est celle qui maximise son profit en moyenne, indépendamment de la stratégie adoptée par les autres agents. Nous avons proposé un algorithme probabiliste ainsi qu'un mécanisme probabiliste, tous les deux à véracité garantie et qui ont un meilleur rapport d'approximation que SPT. Notre algorithme centralisé s'applique à un nombre quelconque de machines et des durées d'exécution des tâches arbitraires, tandis que notre mécanisme de coordination s'applique en présence de deux machines et pour des durées d'exécution des tâches qui sont des puissances d'une constante.

9.2 Préliminaires

On considère m machines et n tâches $T_1, \ldots, T_n$. On note l_i la durée d'exécution (où longueur) de la tâche T_i. On dit de la tâche T_i qu'elle est plus grande que la tâche T_j si et seulement si $l_i > l_j$ ou ($l_i = l_j$ et $i > j$). La durée de chaque tâche n'est connue que par le "propriétaire" de cette tâche.

Nous avons supposé comme dans [57] que les tâches ne pouvaient pas diminuer leur longueur et par conséquent chaque tâche déclare une valeur b supérieure ou égale à sa vraie durée. Le but de chaque tâche est de minimiser son temps de complétude, et elle peut donc mentir si cela contribue à minimiser cet objectif.

Il est possible de considérer deux modèles d'exécution des tâches :

- dans le premier, utilisé dans la section 9.3, si une tâche T_i déclare une valeur $b > l_i$, alors son temps d'exécution demeure l_i,
- dans le deuxième, utilisé dans la section 9.4, si une tâche T_i déclare une valeur $b > l_i$, alors son temps d'exécution est b, c'est-à-dire que T_i (ou son propriétaire) ne connaîtra pas le résultat de son exécution avant b unités de temps après le début d'exécution de T_i. Ce modèle est appelé par la suite, modèle *faible* d'exécution.

Par la suite, dans un soucis de simplification, lorsque l'on utilise le terme mécanisme, il est sous-entendu qu'il s'agit d'un mécanisme avec véracité garantie.

Un mécanisme probabiliste consiste en une distribution de probabilité sur un ensemble de mécanismes déterministes. Par exemple étant donnés deux mécanismes déterministes $M1$ et $M2$, on peut former un mécanisme probabiliste de la manière suivante : avec la probabilité p on utilise le mécanisme $M1$, et avec la probabilité $(1 - p)$ on utilise le mécanisme $M2$. Ainsi dans le contexte distribué (voir la section 9.4), étant donné le mécanisme probabiliste, chaque tâche déclare une valeur qui représente sa durée. On annonce alors aux tâches le mécanisme utilisé, et chaque tâche choisit la machine sur laquelle elle désire s'exécuter en tenant compte des politiques suivies sur chaque machine.

Un mécanisme probabiliste est dit *avec véracité garantie* si, pour chaque tâche, son temps de complétude en moyenne, lorsqu'elle dit la vérité, est plus petit que son temps de complétude en moyenne lorsqu'elle déclare une valeur plus grande que la réalité. Formellement, un mécanisme probabiliste M est dit avec véracité garantie si $E_i(l_i) \leq E_i(b_i)$, pour tout i et $b_i \geq l_i$, avec $E_i(b_i)$ le temps moyen de complétude de la tâche T_i lorsqu'elle déclare b_i.

Dans le contexte centralisé (voir la section 9.3), l'ordonnancement retourné est obtenu de la manière suivante : les tâches déclarent leur longueur, et avec la probabilité p elles sont ordonnancées

Soit $\{T_1, T_2, \ldots, T_n\}$ les n tâches devant être exécutées sur les $m \geq 2$ machines identiques, $\{P_1, P_2, \ldots, P_m\}$. On suppose que $l_1 \leq l_2 \leq \cdots \leq l_n$. Les tâches sont ordonnancées alternativement sur les machines $P_1, P_2, \ldots, P_m$, dans l'ordre SPT, et T_{i+1} ne commence à être exécutée que lorsque exactement $\frac{1}{m}$ de la fraction de la tâche T_i a été exécutée. Ainsi T_1 commence à s'exécuter sur P_1 au temps 0, T_2 est ordonnancée sur P_2 au temps $\frac{l_1}{m}$, T_3 est ordonnancée sur P_3 (sur P_1 si $m = 2$) lorsque $\frac{1}{m}$ de la fraction de T_2 a été exécutée, c'est-à-dire au temps $\frac{l_1}{m} + \frac{l_2}{m}$, et ainsi de suite.

TAB. 9.1 – L'algorithme SPT_δ.

Avec une probabilité $\frac{m}{m+1}$, retourner l'ordonnancement SPT_δ ; et avec la probabilité $\frac{1}{m+1}$, retourner l'ordonnancement LPT.

TAB. 9.2 – L'algorithme LS_δ.

sur les machines selon un algorithme $M1$, tandis qu'avec la probabilité $1-p$ elles sont ordonnancées sur les machines selon un algorithme $M2$.

9.3 Algorithme centralisé avec véracité garantie

Nous avons proposé, dans le contexte centralisé, un algorithme probabiliste qui est avec véracité garantie. L'idée que nous avons utilisée consiste à coupler l'algorithme bien connu LPT avec une variante de l'algorithme SPT.

La variante de l'algorithme SPT, notée SPT_δ par la suite, est décrite dans le tableau 9.1. Un ordonnancement retourné par cet algorithme est appelé un ordonnancement SPT_δ par la suite. La figure 9.1 montre un exemple d'un tel ordonnancement lorsque $m = 3$.

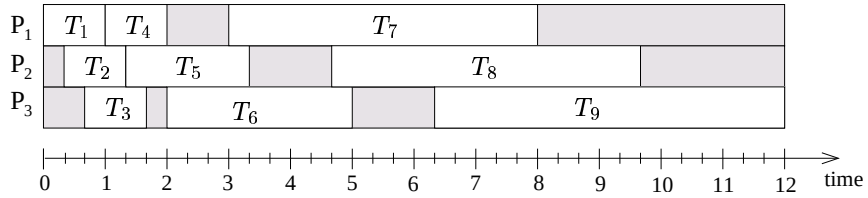


FIG. 9.1 – Un ordonnancement SPT_δ.

Nous avons montré que, malgré les temps d'inactivité ajoutés, cet algorithme avait le même rapport d'approximation que l'algorithme classique SPT.

Théorème 44 *[20] L'algorithme SPT_δ est $2 - \frac{1}{m}$-approché.*

Nous avons alors considéré l'algorithme suivant, noté LS_δ par la suite et décrit dans le tableau 9.2.

Nous avons montré les résultats suivants :

Théorème 45 *[20] L'algorithme LS_δ est $2 - \frac{1}{m+1}\left(\frac{5}{3} + \frac{1}{3m}\right)$-approché en moyenne.*

Soit $p \in \mathbb{R}$ tel que $0 \leq p \leq 1$. Avec la probabilité p, l'ordonnancement retourné est de type SPT. Avec la probabilité $(1-p)$, l'ordonnancement retourné est de type SPT-LPT.

TAB. 9.3 – L'algorithme $SSL(p)$.

Soit $p \in \mathbb{R}$ tel que $0 \leq p \leq 1$. Soit $\varepsilon > 0$ un réel positif plus petit que toutes les durées (réelles) des tâches. La première machine P_1 ordonnance, à partir du temps 0, ses tâches selon l'ordre SPT. La deuxième machine P_2 ordonnance avec la probabilité p ses tâches selon l'ordre SPT, en faisant commencer la première tâche à l'instant ε ; et P_2 ordonnance avec la probabilité $(1-p)$, ses tâches selon l'ordre LPT, en faisant commencer la première tâche à l'instant 0.

TAB. 9.4 – Le mécanisme de coordination correspondant à l'algorithme $SSL(p)$.

Théorème 46 *[20] L'algorithme LS_δ est à véracité garantie.*

Lorsque $m = 2$, le rapport d'approximation en moyenne de l'algorithme LS_δ est $\frac{25}{18} < 1.39$. Ce rapport est meilleur que celui de l'algorithme $SSL(p)$ introduit dans la section 9.4, mais il faut remarquer que l'algorithme LS_δ n'est pas un mécanisme de coordination puisque chaque machine doit connaître les tâches exécutées sur les autres machines.

On peut aussi remarquer que puisque les rapports d'approximation des algorithmes SPT_δ et LPT sont respectivement égaux à $2 - \frac{1}{m}$ et $\frac{4}{3} - \frac{1}{3m}$, un ordonnancement retourné par l'algorithme LS_δ est toujours, dans le pire des cas, $2 - \frac{1}{m}$-approché, ce qui n'est pas pire que le rapport atteint par un ordonnancement SPT.

9.4 Mécanisme de coordination avec véracité garantie

On se place dans le cas de 2 machines. Nous appelons ordonnancement SPT-LPT, un ordonnancement dans lequel la première machine, notée P_{SPT}, utilise la politique SPT, tandis que la deuxième machine, notée P_{LPT}, utilise la politique LPT. Il est facile de voir alors qu'une tâche T_i s'exécute sur P_{SPT}, si la somme des durée des tâches qui sont plus petites qu'elle, est inférieure ou égale, à la somme des durées des tâches qui sont plus grandes qu'elle. Dans le cas contraire, cette tâche s'exécute sur P_{LPT}.

Nous avons considéré l'algorithme suivant, noté $SSL(p)$ par la suite, et représenté dans le tableau 9.3. Il est possible de transformer cet algorithme centralisé $SSL(p)$ en un mécanisme de coordination probabiliste en introduisant un délai sur l'une des machines. Le mécanisme de coordination obtenu est donné dans le tableau 9.4.

Nous avons établi un rapport d'approximation pour ce mécanisme.

Théorème 47 *[20] L'algorithme $SSL(p)$ est en moyenne $(\frac{4}{3} + \frac{p}{6})$-approché.*

Par la suite nous utilisons le modèle faible d'exécution. Nous faisons l'hypothèse que toutes les durées réelles des tâches sont des puissances d'une constante C (connue). Cela signifie qu'une tâche ne peut que déclarer une longueur qui vérifie cette propriété.

Théorème 48 *[20] Soit $p \in \mathbb{R}$ tel que $\frac{2}{3} < p \leq 1$. L'algorithme $SSL(p)$ est à véracité garantie si les tâches ont des durées qui sont des puissances d'une constante $C \geq \frac{4-3p}{2-p}$.*

La figure 9.2 *Gauche* donne une illustration du théorème 48 : si les durées des tâches sont des puissances d'une constante supérieure ou égale à $C(p)$, l'algorithme $SSL(p)$ est à véracité garantie. La figure 9.2 *Droit* illustre le théorème 47 et montre le rapport d'approximation en moyenne de l'algorithme $SSL(p)$.

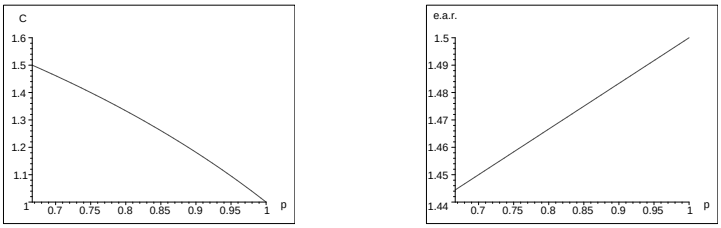


FIG. 9.2 – *Gauche :* Si les durées des tâches sont des puissances d'une constante supérieure ou égale à $C(p)$, l'algorithme $SSL(p)$ est à véracité garantie. *Droit :* Le rapport d'approximation en moyenne de l'algorithme $SSL(p)$.

Remarque 3 *En fait une condition suffisante pour que l'algorithme $SSL(p)$ soit à véracité garantie est d'avoir pour tout i : $l_{i+1} = l_i$ ou $l_{i+1} \geq C \times l_i$. Par conséquent, si toutes les durées des tâches appartiennent à un ensemble $S = \{x_1, x_2, \ldots, x_k\}$ tel que pour chaque j on ait $x_{j+1} \geq C \times x_j$, alors l'algorithme $SSL(p)$ est à véracité garantie.*

Si l'on ne fait aucune hypothèse sur les durées des tâches, alors l'algorithme $SSL(p)$ n'est pas à véracité garantie.

Théorème 49 *[20] Soit $p \in \mathbb{R}$ tel que $0 \leq p < 1$. L'algorithme $SSL(p)$ n'est pas à véracité garantie si les durées des tâches peuvent être quelconque.*

Théorème 50 *[20] Soit $p \in \mathbb{R}$ tel que $0 \leq p < \frac{1}{2}$. L'algorithme $SSL(p)$ n'est pas à véracité garantie, même si les durées des tâches sont des puissances d'une constante B ($B > 1$) (quelque soit la valeur de B).*

9.5 Autres mécanismes de coordination : résultats négatifs

Il est possible d'imaginer d'autres algorithmes faisant intervenir les ordonnancements LPT, SPT et SPT-LPT. Par exemple l'algorithme noté $SL(p)$ retourne avec la probabilité p un ordonnancement SPT et avec la probabilité $(1 - p)$ un ordonnancement LPT. De même l'algorithme noté $LSL(p)$ retourne avec la probabilité p un ordonnancement LPT et avec la probabilité $(1 - p)$ un ordonnancement SPT-LPT. Il est possible, comme à la section 9.4, de transformer ces algorithmes en des mécanismes de coordination en introduisant des délais. Nous avons obtenu les résultats négatifs suivants concernant la véracité de ces algorithmes :

Théorème 51 *[20] Soit $p \in \mathbb{R}$ tel que $0 \leq p < 1$. L'algorithme $SL(p)$ n'est pas à véracité garantie si les durées des tâches peuvent être quelconques.*

Théorème 52 *[20] Soit $p \in \mathbb{R}$ tel que $0 \leq p < \frac{1}{2}$. L'algorithme $SL(p)$ n'est pas à véracité garantie, même si les durées des tâches sont des puissances d'une constante B ($B > 1$) (quelque soit la valeur de B).*

Théorème 53 *[20] Soit $p \in \mathbb{R}$ tel que $0 \leq p < 1$. L'algorithme $LSL(p)$ n'est pas à véracité garantie, même si les durées des tâches sont des puissances d'une constante B ($B > 1$) (quelque soit la valeur de B).*

Ces résultats négatifs sont valides pour le modèle faible d'exécution et par conséquent pour l'autre modèle aussi.

9.6 Conclusion

Pour le problème d'ordonnancement $Pm||C_{max}$ nous avons proposé un algorithme probabiliste qui est à véracité garantie, et qui a un rapport d'approximation en moyenne égal à $2-\frac{1}{m+1}\left(\frac{5}{3}+\frac{1}{3m}\right)$ meilleur que celui de SPT (e.g. si $m = 2$ alors ce rapport est plus petit que 1.39 à comparer avec 1.5 pour SPT).

Dans le cas de deux machines, nous avons proposé un mécanisme de coordination probabiliste dans lequel la première machine ordonnance toujours les tâches selon l'ordre SPT, tandis que la deuxième machine ordonnance les tâches selon l'ordre SPT (respectivement LPT) avec une probabilité $p > \frac{2}{3}$ (respectivement $1-p$). Le rapport d'approximation en moyenne de ce mécanisme égal à $\frac{4}{3}+\frac{p}{6}$ est meilleur que celui de SPT égal à $\frac{3}{2}$. Nous avons montré que ce mécanisme était à véracité garantie si les tâches avaient toutes des durées égales à des puissances d'une constante supérieure ou égale à $\frac{4-3p}{2-p}$, mais pas dans le cas général. Enfin nous avons considéré d'autres mécanismes de coordination, pour lesquels nous avons donné des résultats négatifs sur leur véracité garantie.

L'originalité de nos algorithmes réside dans le fait que nous n'utilisons pas de fonction de paiement pour rétribuer les agents. Dans le cadre d'un séjour à l'université d'Athènes, Fanny Pascual et Laurent Gourvès ont obtenu des résultats complémentaires par rapport à ceux exposés ici [138], notamment des résultats d'inapproximabilité.

Nous avons récemment étudié le cas des machines reliées. Sous certaines hypothèses, nous avons pu proposer des algorithmes à véracité garantie ayant un meilleur rapport d'approximation que SPT dans ce cas là également [22].

Chapitre 10

Recherche de solutions "équitables"

Sommaire

Les travaux exposés dans ce chapitre ont fait l'objet des publications [6, 19].

10.1 Introduction

Dans ce chapitre nous recherchons des solutions équitables avec l'approche exposée dans la section 1.3.3. Nous avons considéré deux problèmes. La section 10.2 est consacrée au problème de l'arbre couvrant de poids minimum, et la section 10.3 est consacrée au problème d'ordonnancement $Pm||\sum_j C_j$.

10.2 Le jeu de l'arbre couvrant de poids minimum

On suppose qu'un service doit être apporté à un ensemble de clients à travers un réseau G. On considère G comme étant un graphe connexe non orienté. Parmi tous les nœuds de G on distingue la racine, ou source, notée r. L'ensemble des sommets est noté V^r tandis que $V = V^r \backslash \{r\}$ représente les clients.

Chaque arête $e \in E$ a un coût positif, supposé entier, noté c_e. Le service peut être apporté directement à un client s'il existe une arête entre la source et ce dernier où *via* d'autres clients. Ainsi, la structure minimale pouvant être utilisée est un arbre contenant la source et couvrant tous les clients. Cet arbre possède un coût global (la somme des coûts des arêtes empruntées) que les clients doivent se partager. Ceci est un *jeu coopératif* classique car la coopération a pour but de réduire le coût global. De manière formelle, on a pour chaque coalition $S \subseteq V$:

$$c(S) = \min\{\sum_{e \in T_S} c_e \mid T_S \text{ est un arbre couvrant de } G[S \cup \{r\}]\},$$

où $G[S \cup \{r\}]$ représente le sous-graphe de G induit par l'ensemble des sommets $S \cup \{r\}$, et $c(S)$ est le coût contracté par la coalition S, c'est-à-dire le prix que les joueurs de la coalition S doivent payer collectivement pour avoir accès au service.

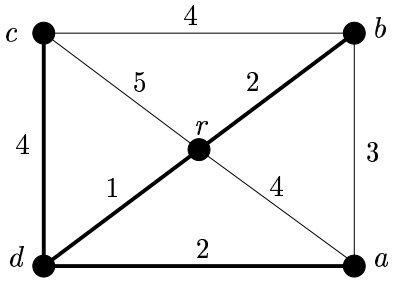


FIG. 10.1 – Selon la règle de Bird, le sommet a doit payer 2, le sommet b doit payer 2, le sommet c doit payer 4 et le sommet d doit payer 1.

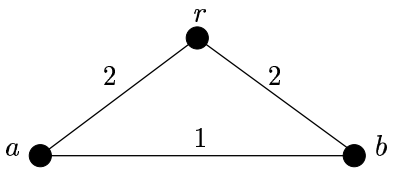


FIG. 10.2 – Illustration de l'exemple 13.

Le partage des coûts pour ce problème a d'abord été étudié par Claus et Kleitman [60] tandis que Bird [43] a traité ce problème avec une approche orientée théorie des jeux et a proposé une règle d'allocation du coût global connue sous le nom de *règle de Bird*. Cette règle consiste à assigner à chaque agent le coût de l'arête qui lui est incidente dans l'unique chemin entre lui-même et la source dans un arbre couvrant de poids minimum.

Soit T_G l'ensemble de tous les arbres couvrants de poids minimum de G et soit C_{opt} le coût global de tout arbre de cet ensemble. Soit T un arbre de T_G et v un sommet de V. Il existe un seul chemin entre r et v dans T. Ce chemin utilise exactement une arête $[x, v]$, où $x \in V^r \backslash \{v\}$. Soit $\beta(T, v) = c_{[x,v]}$ le coût de cette arête. La règle de Bird alloue à v la quantité $\beta(T, v)$, *i.e.* le prix demandé à v est $x_v = \beta(T, v) = c_{[x,v]}$.

Exemple 12 *Dans l'instance donnée en figure 10.1, un arbre couvrant de poids minimum est surligné en gras. Selon la règle de Bird, le sommet a doit payer 2, le sommet b doit payer 2, le sommet c doit payer 4 et le sommet d doit payer 1.*

La règle de Bird assure qu'aucune coalition, autre que la grande coalition V, n'est incitée à se former. Autrement dit, l'ensemble des allocations $\mathcal{C}_{Bird}$ émanant de cette règle est toujours dans le cœur $\mathcal{C}$ du jeu de l'arbre couvrant de poids minimum [67, 92]. Cependant, comme en général il existe plus d'un arbre couvrant de poids minimum pour un réseau donné, cette règle ne conduit pas toujours à une allocation unique des coûts, et d'un point de vue individuel, cette règle peut demander à un client un prix très grand comparé à celui qu'il aurait à payer avec un arbre couvrant de poids minimum différent.

Exemple 13 *Pour l'instance donnée en figure 10.2, selon que l'arête* $[r, a]$ *ou l'arête* $[r, b]$ *appartient à l'arbre couvrant, le prix que doit payer le sommet a est soit* 2, *soit* 1.

Par conséquent, le mécontentement d'un agent est d'autant plus grand que le prix qu'il lui est demandé est éloigné du plus petit possible. Conformément à ce qui est exposé dans la définition 21

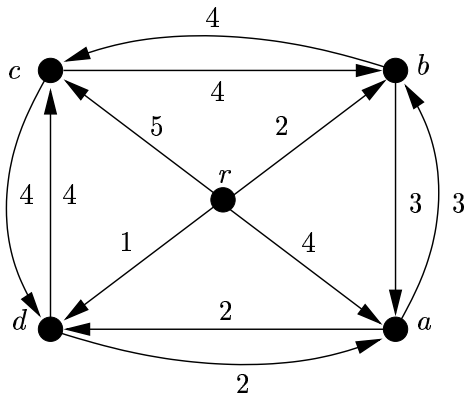


FIG. 10.3 – Transformation du graphe non orienté de la figure 10.1 en un graphe orienté.

nous avons définit le mécontentement de v en rapport avec T et la règle de Bird comme suit :

$$\Delta_v(T, \mathcal{C}_{Bird}) = \frac{\beta(T, v)}{\min_{T' \in T_G}\{\beta(T', v)\}}.$$

Nous avons alors étudié les deux problèmes d'optimisation suivants :
- ARBRE COUVRANT-MPM qui consiste à déterminer un arbre couvrant de poids minimum qui minimise le *pire* mécontentement ;
- ARBRE COUVRANT-MMM qui consiste à déterminer un arbre couvrant de poids minimum qui minimise le mécontentement *moyen*.

Bien que pour beaucoup d'instances, une solution qui minimise le pire mécontentement minimise aussi le mécontentement moyen, ce n'est pas vrai en général.

10.2.1 Le rapport de mécontentement et la règle de Bird

On construit à partir de $G(V^r, E, c)$ un graphe orienté $H = (V^r, A, c)$ comme suit :
- pour chaque paire de sommets $x, y \in V \times V^r$ telle que $x \neq y$, on crée deux arcs (x, y) et (y, x) dans A s'il existe une arête $[x, y]$ dans E et on pose $c_{(x,y)} = c_{(y,x)} = c_{[x,y]}$;
- pour tout sommet $x \in V$ tel que $[r, x] \in E$, on crée l'arc (r, x) de coût $c_{[r,x]}$ dans A.

Ainsi, pour chaque arbre couvrant de poids minimum de G, il existe dans H une arborescence couvrante de poids minimum (notée $aocpm^r$ par la suite) correspondante enracinée en r qu'on note B. Par la suite, on travaille uniquement avec les arborescences sur le graphe orienté H, et on note T_H l'ensemble des $aocpm^r$ de H.

Exemple 14 *Le figure 10.3 présente la transformation du graphe de la figure 10.1 en un graphe orienté selon les règles données ci-dessus.*

Pour déterminer le rapport de mécontentement d'un sommet $v \in V$ dans une arborescence B, il est nécessaire de savoir quels sont les prix qu'il pourrait avoir à payer. On note cet ensemble $F(v)$. De part la règle de Bird, il est clair que $\beta(B, v)$ ne peut prendre de valeurs que dans $\{c_{(x,v)} \mid (x, v) \in A\}$. Cette condition est nécessaire mais pas suffisante. Nous avons proposé une procédure **PRIX** (voir le tableau 10.1) qui fournit l'ensemble des prix qu'un sommet pourrait avoir à payer. Elle consiste

Entrées :	Un graphe orienté $H = (V^r, A, c)$, un sommet $v \in V$
Étape 1 :	Déterminer une *aocpm*^{r} de H et soit C_{opt} son coût total.
Étape 2 :	$L := \{c_{(x,v)} \mid (x,v) \in A\}$
Étape 3 :	$F(v) := \emptyset$
Étape 4 :	**Pour** chaque l de L **faire**
	$A' := A \backslash \{(x,v) \mid c_{(x,v)} \neq l\}$
	$H' := (V^r, A', c_{\mid A'})$
	Déterminer une *aocpm*^{r} B' dans H'.
	Si B' existe et que son coût total est C_{opt}
	Alors $F(v) := F(v) \cup \{l\}$
	Fin Pour
Étape 5 :	Retourner $F(v)$.

TAB. 10.1 – Procédure **PRIX**. On note $c_{\mid A'}$ l'ensemble des coûts pour le sous-ensemble $A' \subseteq A$.

à retirer des arcs incidents au sommet pour le forcer à payer un certain prix. Parmi ces prix, on distingue le plus bas noté $F_{min}(v)$ et le plus haut noté $F_{max}(v)$.

Déterminer une *aocpm*^{r} nécessite un temps $\mathcal{O}(mn)$ (où $m = |A|$ et $n = |V^r|$) [75]. Par conséquent, la complexité en temps de **PRIX** est $\mathcal{O}(mn^2)$. Il est possible d'améliorer l'algorithme en évitant de générer plusieurs fois les mêmes arborescences, mais la complexité dans le pire des cas de l'algorithme demeure inchangée.

10.2.2 Minimiser le pire mécontentement

Parmi toutes les arborescences couvrantes de poids minimum, on en cherche une qui minimise le pire mécontentement. Ce problème, noté ARBRE COUVRANT-MPM, peut être décrit comme suit :

$$\text{argmin}_{B \in T_H} \max_{v \in V} \{\Delta_v(B, \mathcal{C}_{Bird})\}$$

où

$$\Delta_v(B, \mathcal{C}_{Bird}) = \frac{\beta(B,v)}{\min_{B' \in T_H}\{\beta(B',v)\}}.$$

Bien que la procédure **PRIX** détermine tous les prix que la règle de Bird peut assigner à un sommet, elle n'est pas en elle même suffisante pour déterminer un arbre couvrant de poids minimum qui minimise le pire mécontentement. En effet, on peut construire des instances pour lesquelles cet arbre couvrant optimal ne correspond à aucune des arborescences générées par **PRIX**.

La quantité $\max_{v \in V}\{F_{max}(v)/F_{min}(v)\}$ est une borne supérieure pour le pire mécontentement. L'algorithme **ALGO 1** (voir tableau 10.2) que nous avons proposé pour résoudre le problème ARBRE COUVRANT-MPM fait décroître cette borne supérieure, jusqu'à ce qu'elle atteigne la valeur optimale. De manière itérative, on retire des arcs de H jusqu'à ce que toute *aocmp*^{r} de H soit optimale pour le pire mécontentement.

Théorème 54 *[6] L'algorithme* **ALGO 1** *retourne une solution optimale pour le problème* ARBRE COUVRANT-MPM *et s'exécute en temps polynomial.*

Exemple 15 *L'instance G est donnée dans la figure 10.4(a). Un arbre couvrant de poids minimum dans G possède un coût $C_{opt} = 6$. Le graphe orienté H correspondant à G est donné dans la*

Entrée :	Un graphe orienté $H = (V^r, A, c)$
Étape 1 :	Déterminer une *aocpm*^{r} B de H et soit C_{opt} son coût total.
Étape 2 :	Pour chaque sommet $v \in V$, déterminer $F(v)$ à l'aide de **PRIX**.
Étape 3 :	$A' := A$
Étape 4 :	Sélectionner $v' \in V$ tel que $\frac{F_{max}(v')}{F_{min}(v')} = \max_{v \in V} \left\{ \frac{F_{max}(v)}{F_{min}(v)} \right\}$.
Étape 5 :	$A'' := A'$
Étape 6 :	$A' := A' \backslash \{(x, v') \mid c_{(x,v')} = F_{max}(v')\}$
Étape 7 :	Déterminer une *aocpm*^{r} B' de $H' = (V^r, A', c_{\mid A'})$.
	Si B' n'existe pas ou son coût total est supérieur à C_{opt}
	Alors Aller à l'**Étape 8**.
	Sinon Retirer $F_{max}(v')$ de $F(v')$.
	Aller à l'**Étape 4**.
Étape 8 :	Déterminer une *aocpm*^{r} B'' de $H'' = (V^r, A'', c_{\mid A''})$.
Étape 9 :	Retourner B''.

TAB. 10.2 – L'algorithme **ALGO 1** détermine une arborescence dont l'arbre couvrant correspondant est optimal pour le problème ARBRE COUVRANT-MPM.

figure 10.4(b). On a $F(x) = \{2, 3\}$, $F(y) = \{1, 2, 3\}$ *et* $F(z) = \{1, 3\}$, *ainsi* $\frac{F_{max}(x)}{F_{min}(x)} = 3/2$, $\frac{F_{max}(y)}{F_{min}(y)} = 3$ *et* $\frac{F_{max}(z)}{F_{min}(z)} = 3$. *Le pire mécontentement survient sur les sommets y et z. Dans la figure 10.4(c), l'algorithme retire* (r, z). *Avec cette nouvelle instance, il est toujours possible de déterminer une aocpm*^{r} *de coût 6. On a* $\frac{F_{max}(x)}{F_{min}(x)} = 3/2$, $\frac{F_{max}(y)}{F_{min}(y)} = 3$ *et* $\frac{F_{max}(z)}{F_{min}(z)} = 1$. *Dans la figure 10.4(d), l'algorithme retire* (r, y). *Avec cette nouvelle instance, il est toujours possible de déterminer une aocpm*^{r} *de coût 6. On a* $\frac{F_{max}(x)}{F_{min}(x)} = 3/2$, $\frac{F_{max}(y)}{F_{min}(y)} = 2$ *et* $\frac{F_{max}(z)}{F_{min}(z)} = 1$. *Le pire mécontentement survient au sommet y. Dans la figure 10.4(e), l'algorithme retire* (x, y) *mais il n'y a plus d' aocpm*^{r} *de coût 6. Ainsi, l'algorithme détermine sur le graphe de la figure 10.4(d) une aocpm*^{r} *et retourne l'arbre correspondant (voir la figure 10.4(f)). Finalement, le pire mécontentement est de 2.*

10.2.3 Minimiser le mécontentement moyen

Minimiser la somme ou la moyenne des mécontentements sont des problèmes équivalents, ainsi le problème ARBRE COUVRANT-MMM peut s'écrire :

$$\text{argmin}_{B \in T_H} \sum_{v \in V} \Delta_v(B, \mathcal{C}_{Bird}).$$

On associe à chaque arc (x, y), un poids $w_{(x,y)}$ égal au mécontentement du sommet y si l'arc (x, y) appartient à l'*aocpm*^{r}. Ainsi on transforme le graphe initial $G(V^r, E, c)$ en un graphe orienté $H = (V^r, A, c)$ comme suit :

- pour chaque arête $[x, y]$ de G telle que $x \in V$, $y \in V$ et $c_{(x,y)} \geq F_{min}(y)$, on ajoute l'arc (x, y) dans A avec un coût $c_{(x,y)}$ et un poids $w_{(x,y)} = c_{(x,y)}/F_{min}(y)$;
- pour chaque arête $[r, x]$ de G telle que $x \in V$ et $c_{(r,x)} \geq F_{min}(x)$, on ajoute l'arc (r, x) dans A avec un coût $c_{(r,x)}$ et un poids $w_{(r,x)} = c_{(r,x)}/F_{min}(x)$.

Dans le problème ARBRE COUVRANT-MMM, on cherche parmi les arborescences optimales pour le coût, une qui soit de poids minimum. L'idée de l'algorithme que nous avons proposé est, pour

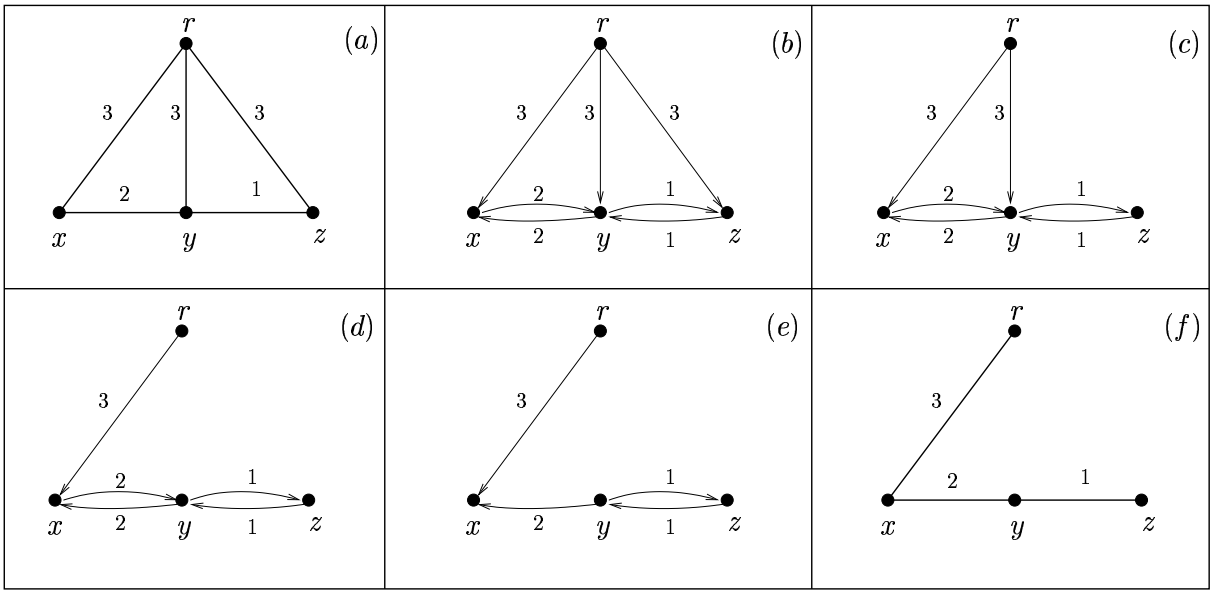


FIG. 10.4 – Illustration de l'exemple 15.

Entrée :	Un graphe orienté $H = (V^r, A, c)$
Étape 1	pour chaque sommet $v \in V$, déterminer $F(v)$ à l'aide de **PRIX**
Étape 2	Déterminer une *aocpm*^{r} B de H et soit $D = \sum_{v \in V} \Delta_v(B, \mathcal{C}_{Bird})$
Étape 3	$\lambda := \frac{D}{D+1}$
Étape 4	Déterminer une *aocpm*^{r} B^* de $H^* = (V^r, A, \tilde{c})$
Étape 5	Retourner B^*

TAB. 10.3 – L'algorithme **ALGO 2**.

chaque arc (x, y), de combiner $c_{(x,y)}$ et $w_{(x,y)}$ en une unique fonction de coût composite $\tilde{c}_{(x,y)}$ telle que toute arborescence optimale pour cette fonction composite soit aussi optimale pour le problème ARBRE COUVRANT-MMM :

$$\tilde{c}_{(x,y)} = \lambda c_{(x,y)} + (1 - \lambda) w_{(x,y)} \tag{10.1}$$

où $0 < \lambda < 1$. Avec un λ assez proche de 1, on peut obtenir une solution optimale pour le problème ARBRE COUVRANT-MMM. Dans l'algorithme **ALGO 2** (voir le tableau 10.3), λ dépend de l'instance et sa valeur est donnée explicitement.

Théorème 55 *[6] L'algorithme* **ALGO 2** *retourne une solution optimale pour le problème* ARBRE COUVRANT-MMM *et s'exécute en temps* $\mathcal{O}(mn^2)$.

Un exemple complet de l'exécution de l'algorithme **ALGO 2** est donné dans la figure 10.5.

L'algorithme **ALGO 2**, moyennant une modification de la définition de λ à l'étape 3, peut aussi fonctionner pour des coûts qui ne sont pas entiers.

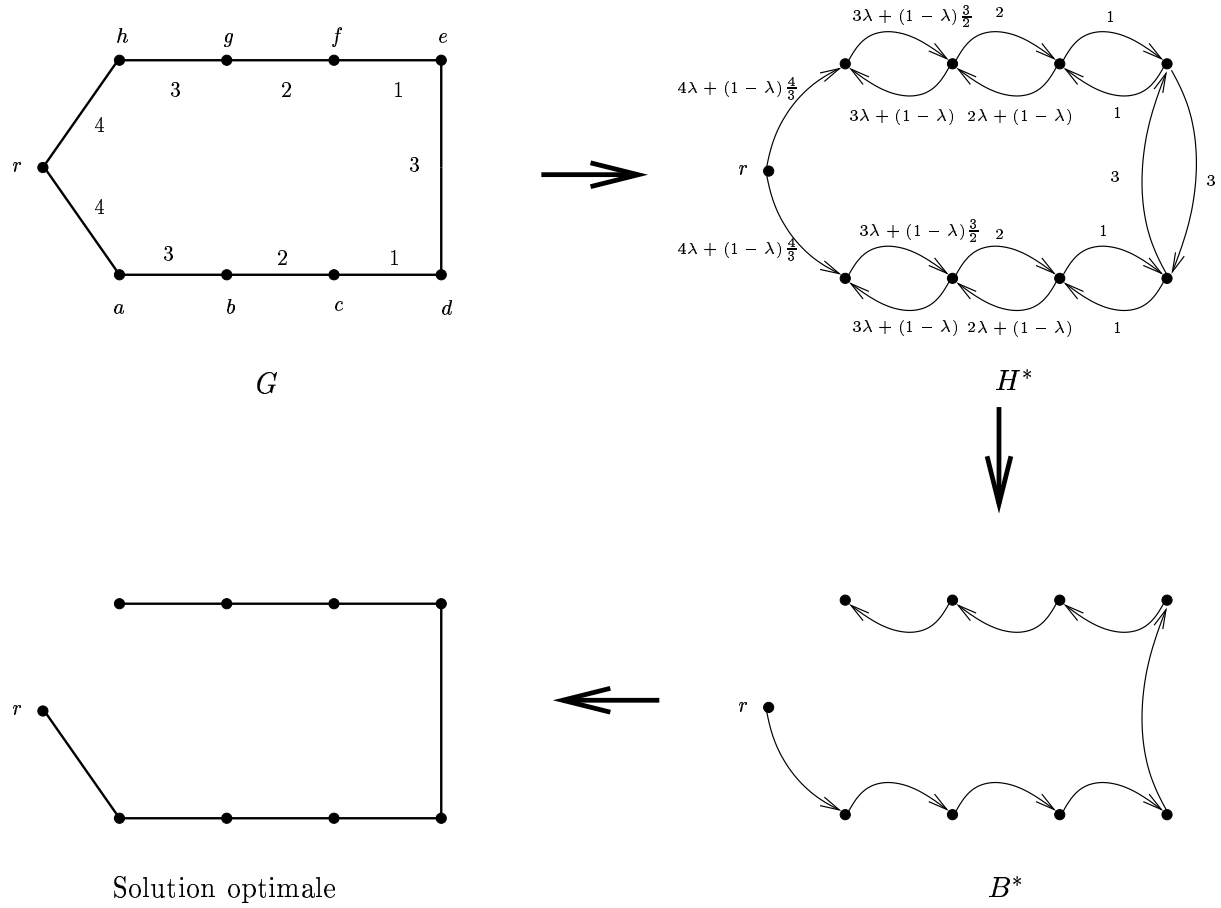


FIG. 10.5 – La liste de tous les prix possibles est d'abord déterminée : $F(a) = F(h) = \{3, 4\}$, $F(b) = F(g) = \{2, 3\}$, $F(c) = F(f) = \{1, 2\}$, $F(d) = F(e) = \{1, 3\}$. Une $aocpm^r$ B est déterminée et la somme de ses mécontentements est $D = 4/3 + 3/2 + 1 + 3 + 1 + 1 + 1 = \frac{71}{6}$. On a $\lambda = D/(D+1) = 71/77$. H^* est alors construit et une $aocpm^r$ B^* est déterminée.

10.3 Le problème MIN MAX SCT et mesures d'équité

Nous avons considéré le problème d'ordonnancement $Pm||\sum_j C_j$, défini à la section 2.2.4, dans lequel nous avons introduit des critères supplémentaires : somme des temps de complétude par machine, équité globale et équité individuelle. On considère ainsi un ensemble de n tâches, $T_1, T_2, \cdots, T_n$, avec des durées d'exécution $l_1, l_2, \cdots, l_n$ et un ensemble de m machines (ou processeurs) identiques $P_1, P_2, \cdots, P_m$. Étant donné un ordonnancement, on note C_j le temps de complétude de la tâche T_j. On considère comme critère, la somme des temps de complétude : $\sum_{i=1}^{n} C_i$. Rappelons, comme cela a été indiqué à la section 2.2.4, qu'il est possible de trouver tous les ordonnancements optimaux pour le problème $Pm||\sum_j C_j$ à l'aide de la famille des ordonnancements SPT.

On note X le vecteur des temps de complétude dont la $i^{\text{ième}}$ coordonnée est le temps de complétude de la tâche T_i. On note $\overrightarrow{X}$ le vecteur X ordonné de manière décroissante. Pour toute instance $\mathcal{I}$, on note $V(\mathcal{I})$ l'ensemble des vecteurs X qui sont induits par tous les ordonnancements.

Nous avons cherché à savoir si parmi la famille des ordonnancements SPT qui sont optimaux vis à vis du critère de la somme des temps de complétude, il était possible d'obtenir des ordonnancements qui soient à la fois de bonne qualité pour les critères suivants :

- le *maximum de la somme des temps de complétude par machine* : $\max_{1\leq i\leq m} \sum_{T_j \in P_i} C_j$. Ce critère prend en compte le fait de vouloir distribuer autant que possible la somme des temps de complétude sur chacune des m machines du système. De plus il est possible d'interpréter ce critère de la manière suivante : la somme des temps de complétude des tâches sur une machine donnée représente le nombre moyen de tâches en attente (i.e. non encore terminées) par unité de temps sur cette machine. Ce critère représente donc le nombre moyen maximum de tâches non encore exécutées par machine par unité de temps sur une période donnée. Le problème monocritère $Pm||\max_{1\leq i\leq m}\sum_{T_j\in P_i} C_j$ est noté par la suite MIN MAX SCT.

- *l'équité globale* [121] : Pour deux vecteurs $X, Y \in V(\mathcal{I})$, on note $X \preccurlyeq Y$ si $X_i \leq Y_i$ pour tout i. Le *rapport d'approximation global* de X, noté $c_{gbl}(X)$ et appelé également par la suite *rapport d'équité globale*, est le plus petit α tel que $\overrightarrow{X} \preccurlyeq \alpha\overrightarrow{Y}$ pour tout $Y \in V(\mathcal{I})$. De manière informelle $c_{gbl}(X)$ est le plus petit α pour lequel $\overrightarrow{X}$ est une α-approximation, coordonnée par coordonnée, de tout vecteur $Y \in V(\mathcal{I})$. Le meilleur rapport d'approximation global qu'il est possible d'atteindre pour l'instance $\mathcal{I}$ est alors égal à

$$c^*_{gbl}(\mathcal{I}) = \inf_{X\in V(\mathcal{I})} c_{gbl}(X).$$

- *l'équité individuelle* : Le *facteur de contentement individuel* compare le temps de complétude d'une tâche avec le plus petit temps de complétude de cette même tâche dans n'importe quel ordonnancement. De manière formelle, $c_{ind}(X)$ est défini comme étant le plus petit α tel que $X \preccurlyeq \alpha Y$ pour tout $Y \in V(\mathcal{I})$. Le meilleur rapport qu'il est possible d'atteindre sur l'instance $\mathcal{I}$ est alors défini par

$$c^*_{ind}(\mathcal{I}) = \inf_{X\in V(\mathcal{I})} c_{ind}(X).$$

  Ce rapport est appelé par la suite le *rapport d'équité locale*.
  Au lieu de considérer l'ensemble des vecteurs X qui sont induits par tous les ordonnancements possibles, on peut se restreindre à une famille particulière d'ordonnancements. On note ainsi $V_{SPT}(\mathcal{I})$ l'ensemble des vecteurs des temps de complétude qui sont induits par tous les ordonnancements de type SPT pour l'instance $\mathcal{I}$. On peut définir $c_{ind\_SPT}(X)$ comme étant le

Ordonner les n tâches de manière croissante selon leur durée. À chacune des étapes i, pour $1 \leq i \leq n$, placer la i-ième tâche sur la machine $P_{i \bmod m}$.

TAB. 10.4 – L'algorithme $\text{SPT}_{\text{glouton}}$.

plus petit α tel que $X \preccurlyeq \alpha Y$ pour tout $Y \in V_{SPT}(\mathcal{I})$, ainsi que

$$c^*_{ind\_SPT}(\mathcal{I}) = \inf_{X \in V_{SPT}(\mathcal{I})} c_{ind\_SPT}(X).$$

On mesure ainsi l'équité individuelle restreinte aux ordonnancements de type SPT.

Notre approche est similaire à celle de Bruno et al. [50] qui ont cherché à savoir si parmi tous les ordonnancements optimaux pour la somme des temps de complétude, il était possible d'en trouver un qui minimise le *makespan*. Ils ont prouvé que le problème était $\mathcal{NP}$-difficile.

10.3.1 L'algorithme $\text{SPT}_{\text{glouton}}$

Il est facile de s'apercevoir que le problème qui consiste à minimiser la somme des temps de complétude et le problème MIN MAX SCT sont différents. On peut en effet produire une instance pour laquelle les solutions optimales de chacun de ces problèmes sont différentes. Nous avons montré que la complexité de ces deux problèmes n'était pas du tout la même.

Théorème 56 *[19] Le problème* MIN MAX SCT *est* **NP**-*difficile.*

Nous avons obtenu un algorithme approché pour ce problème en utilisant une variante de l'algorithme de liste classique SPT. L'algorithme noté $\text{SPT}_{\text{glouton}}$ est représenté dans le tableau 10.4. Cet algorithme est optimal pour le problème $Pm||\sum_j C_j$ et il permet d'obtenir une solution approchée pour le problème MIN MAX SCT.

Théorème 57 *[19] L'algorithme $SPT_{glouton}$ est $(3 - \frac{3}{m} + \frac{1}{m^2})$-approché pour le problème* MIN MAX SCT.

À l'aide d'une famille d'instances, il est possible de montrer que l'algorithme $\text{SPT}_{\text{glouton}}$ ne permet pas d'obtenir un rapport d'approximation meilleur que $2 - \frac{2}{m^2+m}$.

10.3.2 Équité globale et individuelle

Nous avons étudié les ordonnancements retournés par l'algorithme $\text{SPT}_{\text{glouton}}$ vis à vis des critères d'équité.

Nous notons $P \parallel$ all le problème qui consiste à trouver un ordonnancement ayant le plus petit rapport d'équité globale possible. Nous avons obtenu le résultat suivant :

Théorème 58 *[19] Pour toute instance $\mathcal{I}$ du problème $Pm \parallel$ all, pour tout $m \geq 1$, on a $c^*_{gbl}(\mathcal{I}) \leq 2 - \frac{1}{m}$. De plus le vecteur X des temps de complétude de tout ordonnancement retourné par l'algorithme $SPT_{glouton}$ satisfait $c_{gbl}(X) \leq 2 - \frac{1}{m}$.*

Le rapport obtenu par l'algorithme $\text{SPT}_{\text{glouton}}$ est optimal dans le cas de 2 machines.

Théorème 59 *[19] Il n'est pas possible d'avoir $c^*_{gbl}(\mathcal{I}) < 2 - \frac{1}{m}$ pour toute instance $\mathcal{I}$ du problème $Pm \parallel$ all.*

Il est à noter que Philips et al. [139] ont présenté un algorithme 3-approché pour le rapport d'approximation global dans le cas où les tâches ont des dates d'arrivé.

Pour l'équité individuelle nous avons obtenu le résultat suivant. On note $P \parallel ind\_all$ le problème qui consiste à trouver un ordonnancement ayant le plus petit rapport d'équité individuelle possible.

Théorème 60 *[19] Pour toute instance $\mathcal{I}$ du problème $Pm \parallel ind\_all$, pour tout $m \geq 1$, on a $c^*_{ind}(\mathcal{I}) \leq 1 + \frac{n-1}{m}$. De plus, le vecteur X des temps de complétude de tout ordonnancement retourné par l'algorithme $SPT_{glouton}$ satisfait $c_{ind}(X) \leq 1 + \frac{n-1}{m}$.*

Théorème 61 *[19] Il n'est pas possible d'avoir $c^*_{ind}(\mathcal{I}) < 1 + \frac{n-1}{m}$ pour toutes les instances $\mathcal{I}$ du problème $Pm \parallel ind\_all$.*

Nous avons également étudié l'équité individuelle restreinte aux ordonnancements de type SPT. On note $Pm \parallel ind\_all_{SPT}$ le problème qui consiste à trouver un ordonnancement ayant le plus petit rapport d'équité individuelle restreinte aux ordonnancements de type SPT.

Théorème 62 *[19] Pour toute instance $\mathcal{I}$ du problème $Pm \parallel ind\_all_{SPT}$, on a $c^*_{ind\_SPT}(\mathcal{I}) \leq 2$. De plus, tout ordonnancement X de type SPT satisfait $c_{ind\_SPT}(X) \leq 2$.*

Il est possible de montrer que cette borne est la meilleure possible.

Théorème 63 *[19] Soit $\varepsilon > 0$. Il n'est pas possible d'avoir $c^*_{ind\_SPT}(\mathcal{I}) < 2 - \varepsilon$ pour toute instance $\mathcal{I}$ du problème $Pm \parallel ind\_all_{SPT}$.*

10.4 Conclusion

L'équité est subjective, tout particulièrement pour les problèmes de partage de coûts. Cette notion, souvent conflictuelle selon les situations, n'est pas clairement établie. Nous avons introduit une nouvelle mesure qui prend en compte le mécontentement des agents et nous avons appliqué ce concept pour le jeu classique de l'arbre couvrant de poids minimum. Sans se restreindre au problème étudié, nous pensons que le rapport de mécontentement peut être étudié pour une large palette de problèmes et particulièrement les problèmes de partage de coûts. Nous avons proposé deux algorithmes qui nous permettent de minimiser le pire mécontentement ainsi que le mécontentement moyen et nous avons montré qu'une solution optimale pour le premier problème n'est pas forcement optimale pour le second. Par conséquent, la question de savoir si une solution optimale pour le problème ARBRE COUVRANT-MPM peut être très éloignée d'une solution optimale pour le problème ARBRE COUVRANT-MMM se pose.

Il est possible de construire une instance qui montre qu'une solution optimale pour ARBRE COUVRANT-MMM peut avoir un pire mécontentement qui est le double de l'optimum. De manière symétrique, une solution optimale pour ARBRE COUVRANT-MPM peut avoir un mécontentement moyen qui est le double de l'optimum. Par contre, la question de savoir si en général ces rapports sont bornés par une constante demeure ouverte.

Enfin, on peut remarquer que l'ensemble des allocations trouvées en utilisant le résultat de Bird, $\mathcal{C}_{Bird}$, sont strictement incluses dans le cœur $\mathcal{C}$ du jeu de l'arbre couvrant. Par conséquent, la question de trouver une allocation $x \in \mathcal{C}$ qui minimise ($\max_i$ ou $\sum_i$) $\Delta_i(x, \mathcal{C})$, au lieu de déterminer une allocation $x \in \mathcal{C}_{Bird}$ minimisant ($\max_i$ ou $\sum_i$) $\Delta_i(x, \mathcal{C}_{Bird})$, demeure ouverte.

Nous avons étudié le problème MIN MAX SCT dans lequel on cherche à minimiser la somme maximum des dates de fin d'exécution des tâches par machine. Nous avons montré que ce problème

est **NP**-difficile et qu'un algorithme glouton de type SPT permet d'obtenir un rapport d'approximation inférieur à 3. La question de savoir s'il existe un schéma d'approximation pour ce problème demeure ouverte. Nous avons de plus étudié deux mesures d'équité pour les tâches. Nous avons montré que le rapport d'équité globale était inférieur à $2 - 1/m$ pour l'algorithme $\mathrm{SPT}_{\mathrm{glouton}}$. Sachant que le rapport d'approximation (classique) d'un algorithme de liste SPT est égal à $2 - 1/m$ il n'était pas possible d'avoir un meilleur rapport. De plus nous avons montré qu'aucun algorithme ne peut avoir, sur toutes les instances, un rapport d'équité globale inférieur à $2 - 1/m$. Nous avons également montré que l'algorithme $\mathrm{SPT}_{\mathrm{glouton}}$ possède le meilleur rapport d'équité individuelle possible.

Cinquième partie

Conclusion

Chapitre 11

Conclusion et perspectives

Sommaire

11.1 Bilan

Tout au long des travaux évoqués dans les chapitres précédents, nous avons dû faire face à diverses difficultés. Tout d'abord, et c'est classique, l'absence de ressources de calculs illimitées et le fait de vouloir traiter des instances de tailles "réelles" imposent des algorithmes de faible complexité. D'autre part l'exigence bien compréhensible d'obtenir des solutions de bonne qualité face à des problèmes souvent **NP**-difficiles impose, soit d'utiliser des algorithmes avec garantie de performance, soit l'utilisation de métaheuristiques réputées pour obtenir de bonnes solutions en pratique pour de nombreux problèmes d'optimisation combinatoire. Celles-ci s'appuient souvent sur la recherche locale dont la composante essentielle est le voisinage utilisé. Nous avons donc cherché soit des algorithmes polynomiaux avec garantie de performance (voir le chapitre 4), soit à améliorer les métaheuristiques en nous concentrant sur l'utilisation de grands voisinages, c'est-à-dire de taille exponentielle, examinables de manière efficace, c'est-à-dire en temps polynomial (voir le chapitre 3). Une autre voie de recherche, non exposée ici faute de place, a consisté à utiliser des méthodes hybrides, combinant par exemple des méthodes exactes, telle que la programmation par contraintes et des métaheuristiques [25, 26].

Une autre source de difficulté est apparue en considérant la présence de multiples fonctions objectifs à optimiser simultanément. Celles-ci sont souvent en conflit l'une avec l'autre. Une approche possible consiste alors à rechercher une solution qui soit la plus proche possible d'une solution "idéale" qui serait optimale pour chacune des fonctions objectifs (voir le chapitre 5). Une autre approche consiste à rechercher l'ensemble des solutions non dominées par d'autres solutions, c'est la courbe de Pareto. Cet ensemble est souvent de taille exponentielle, où alors impossible à calculer en temps polynomial, et il faut par conséquent rechercher des courbes de Pareto approchée : soit avec garantie de performance (voir les chapitres 6 et 7), soit à l'aide de métaheuristiques (voir le chapitre 3).

Enfin, la prise en compte d'agents individualistes impose de nouvelles contraintes sur les algorithmes qu'il faut concevoir. Par exemple, face à des agents qui ont la possibilité de remettre en cause la solution qui leur est fournie, qui même si elle est de bonne qualité d'un point de vue global, peut se révéler moins satisfaisante pour un agent particulier vis à vis de sa fonction objectif locale,

il est nécessaire de rechercher des solutions stables. Celles-ci peuvent correspondre à des équilibres de Nash approchées. Il faut donc trouver de bons compromis entre le degré de stabilité d'une solution et sa qualité par rapport à la fonction objectif globale qu'on cherche à optimiser (voir le chapitre 8). Une autre manière d'assurer la stabilité consiste à se préoccuper du mécontentement de chaque agent. C'est l'approche que nous avons suivie dans le chapitre 10 en introduisant des mesures d'équité. Enfin face à la présence d'agents qui peuvent être tentés de manipuler un algorithme à leur avantage en déclarant de fausses informations, il est nécessaire de concevoir des algorithme avec véracité garantie ou, dans un contexte décentralisé, des mécanismes de coordination (voir le chapitre 9).

11.2 Perspectives

Au delà des perspectives de recherche mentionnées dans les conclusions des chapitres précédents, quelles autres perspectives peut-on envisager à plus long terme ?

Concernant l'optimisation multicritère, nous avons dans la grande majorité des cas considéré des problèmes bicritères. On peut vouloir considérer davantage de critères. On peut remarquer que si l'approche point idéal permet d'obtenir des résultats bicritères, son utilisation pour davantage de critères semble difficile. En effet, peu de problèmes semblent permettre de construire une solution approchée pour trois critères conflictuels à partir de trois solutions, chacune optimale pour un des critères pris indépendamment. L'approche courbe de Pareto semble passer plus facilement le cap du bicritère, mais il est possible que la qualité en terme de garantie de performance qu'on peut espérer diminue au fur et à mesure que le nombre de critères augmente, comme c'est le cas pour le problème du voyageur de commerce multicritère.

Nous nous sommes aussi intéressés à des résultats négatifs concernant la non existence d'ensembles $\vec{\varepsilon}$-approchés. En mettant en relation le nombre de solutions dans un ensemble et le nombre de critères, on peut dériver des résultats disant que cet ensemble est au mieux $\vec{\varepsilon}$-approché de la courbe de Pareto, pour certaines valeurs de $\vec{\varepsilon}$. Si cette technique a été utilisée pour le problème TSP$(1,2)$ multicritère, elle peut cependant être généralisée pour d'autres problèmes. Ces résultats d'inapproximabilité se basant sur des arguments de non existence pourraient dans l'avenir être complétés par des résultats d'inapproximabilité basés sur des arguments de complexité.

On peut observer qu'il existe une similitude entre les problématiques rencontrées en recherche locale et celles rencontrées dans la théorie des jeux :

- Quelle est dans le pire cas la qualité d'un optimum local ? Quelle est dans le pire cas la qualité d'un équilibre de Nash ?
- Quelle est la complexité de la détermination d'un optimum local ? Quelle est la complexité de la détermination d'un équilibre de Nash ?

De même il existe une similitude entre l'exécution d'un algorithme de recherche locale (la suite des solutions courantes à chaque itération) et la dynamique des choix des joueurs (les états successifs). L'un converge vers un optimum local, l'autre vers un équilibre de Nash. Tous deux sont le résultat d'une succession d'améliorations locales.

En tirant profil de cet analogie, des résultats récents mais isolés ont vu le jour. Étant donné un algorithme de recherche locale, il est parfois possible de définir un mécanisme de coordination dont les équilibres purs sont des optima locaux vis à vis de la structure de voisinage [110]. De même des connexions entre le prix de l'anarchie et la qualité des optima locaux ont été exploitées dans [72]. De plus, récemment [81], une connexion entre la théorie de la PLS complexité, issue des algorithmes de recherche locale, et la théorie des jeux a été établie. Ces résultats nous laisse à penser qu'il y matière à recherche dans ce domaine pour établir de nouveaux liens et exploiter ces similitudes.

On peut se poser la question de la prise en compte de critères multiples en théorie des jeux. Il existe par exemple plusieurs notions de cœur dans un cadre multicritère [82]. Une autre voie de recherche serait de considérer des mécanismes avec garantie de performance dans un cadre online (voir par exemple [34]).

Enfin, ajoutons que la gamme des problèmes d'optimisation combinatoires qui peuvent être étudiés sous l'angle de la théorie des jeux est considérable. Nous avons ainsi considéré le problème d'ordonnancement classique $P||C_{max}$ sous différents angles le long des chapitres 9 et 8. Une multitude d'autres problèmes sont envisageables. Nous avons ainsi récemment étudié le routage de paquets autonomes dans un réseau de type store-and-forward qui est un chemin, un arbre, ou bien un anneau. Nous avons étudié la performance de différents mécanismes de coordination vis à vis de la minimisation de la date d'arrivée moyenne des paquets ainsi que la date d'arrivée du dernier paquet. Nous avons montré des différences significatives entre ces mécanismes, certains ayant un prix de l'anarchie constant alors que d'autres ont un prix de l'anarchie non borné [138].

Bibliographie

[1] R.K. Ahuja, Ö. Ergun, J.B. Orlin, and A.P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123 :75–102, 2002.

[2] B. Alidaee and N.K. Womer. Scheduling with time dependent processing times : Review and extensions. *Journal of the Operational Research Society*, 50(7) :711–720, 1999.

[3] P. Ambrosio and V. Auletta. Deterministic monotone algorithms for scheduling on related machines. In *Proceedings of WAOA 2004*, pages 267–280, 2004.

[4] E.J. Anderson, C.A. Glass, and C.N. Potts. Machine scheduling. In E.H.L Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 361–414. Wiley, 1997.

[5] E. Angel and E. Bampis. A multi-start dynasearch algorithm for the time dependent single-machine total weighted tardiness scheduling problem. *European Journal of Operational Research*, 162(1) :281–289, 2005.

[6] E. Angel, E. Bampis, L. Blin, and L. Gourvès. Fair cost-sharing methods for the minimum spanning tree game. *soumis à Information Processing Letters*, 2005.

[7] E. Angel, E. Bampis, and A.V. Fishkin. A note on scheduling to meet two min-sum objectives. In *Proceedings of the ninth international workshop on project management and scheduling (PMS'2004)*, pages 143–146, Nancy, France, 2004.

[8] E. Angel, E. Bampis, and R. Giroudeau. Non-approximability results for the hierarchical communication problem with a bounded number of clusters. In *Euro-Par 2002*, pages 217–224. Springer, 2002. LNCS 2400.

[9] E. Angel, E. Bampis, and L. Gourvès. Approximating the pareto curve with local search for the bicriteria TSP(1,2) problem (extended abstract). In A. Lingas and B.J. Nilsson, editors, *Proceedings of FCT'2003*, volume 2751 of *LNCS*, pages 39–48. Springer, 2003.

[10] E. Angel, E. Bampis, and L. Gourvès. Approximating the pareto curve with local search for the bicriteria TSP(1,2) problem. *Theoretical Computer Science*, 310 :135–146, 2004.

[11] E. Angel, E. Bampis, and L. Gourvès. A dynasearch neighborhood for the bictriteria traveling salesman problem. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *LNEMS*, pages 153–176. Springer, 2004.

[12] E. Angel, E. Bampis, and L. Gourvès. Approximation algorithms for the bi-criteria weighted MAX-CUT problem. In *Proceedings of WG'2005*, volume 3787 of *LNCS*, pages 331–340. Springer, 2005. version étendue à paraître dans Discrete Applied Mathematics (2006).

[13] E. Angel, E. Bampis, and L. Gourvès. Approximation results for a bicriteria job scheduling problem on a single machine without preemption. *Information Processing Letters*, 94(1) :19–27, 2005.

[14] E. Angel, E. Bampis, and L. Gourvès. Approximation results on a multiple-query optimization problem. Technical report, IBISC, Université d'Evry, 2006.

[15] E. Angel, E. Bampis, L. Gourvès, and J. Monnot. (Non)-approximability for the multi-criteria TSP(1,2). In *Proceedings of FCT'2005*, volume 3623 of *LNCS*, pages 329–340. Springer, 2005.

[16] E. Angel, E. Bampis, and A. Kononov. A FPTAS for approximating the unrelated parallel machines scheduling problem with costs. In *Proceedings of European Symposium on Algorithms (ESA) 2001*, pages 194–205. Springer, 2001.

[17] E. Angel, E. Bampis, and A. Kononov. On the approximate tradeoff for bicriteria batching and parallel machine scheduling problems. *Theoretical Computer Science*, 306 :319–338, 2004.

[18] E. Angel, E. Bampis, and F. Pascual. Traffic grooming in a passive star wdm network. In *SIROCCO 2004, 21-23 Juin 2004, Smolenice Castle, Slovakia*, volume 3104 of *LNCS*, pages 1–12. Springer, 2004.

[19] E. Angel, E. Bampis, and F. Pascual. Propriétés des ordonnancements SPT : approximation et critères de satisfaction. In *ROADEF 2005, 14-16 février, Tours*, page 61, 2005.

[20] E. Angel, E. Bampis, and F. Pascual. Truthful algorithms for scheduling selfish tasks on parallel machines. In *The 1st Workshop on Internet and Network Economics (WINE 2005), 15-17 décembre 2005, Hong Kong*, volume 3828 of *LNCS*, pages 698–707. Springer, 2005.

[21] E. Angel, E. Bampis, and F. Pascual. The price of approximate stability for a scheduling game problem. In *Euro-Par 2006*. Springer, 2006. LNCS 4128.

[22] E. Angel, E. Bampis, F. Pascual, and A.-A. Tchetgnia. Truthful algorithms for some scheduling problems. Technical report, MPRI Report, 2006.

[23] E. Angel, E. Bampis, F. Pascual, and J. Warnier. Couplage contraint et exploration efficace d'un voisinage exponentiel pour le problème de tournées de véhicules. In *ROADEF 2006, 6-8 février, Lille*, 2006.

[24] E. Angel, E. Bampis, F. Pascual, and J. Warnier. Restricted matchings and efficient exploration of an exponential neighborhood for the vehicle routing problem. Technical Report RR 2006-01, IBISC, Université d'Evry, 2006.

[25] E. Angel, E. Bampis, and M. Rahoual. Approches de coopération entre les colonies de fourmis et la programmation par contraintes pour le problème de repliement de protéines. In *ROADEF 2005, 14-16 février, Tours*, 2005.

[26] E. Angel, E. Bampis, and M. Rahoual. Mixing constraint programming and ants metaheuristic for the protein folding problem. In *Metaheuristics International Conference, MIC'2005, Vienne*, pages 57–62, July 2005.

[27] E. Angel and V. Zissimopoulos. On the quality of local search for the quadratic assignment problem. *Discrete Applied Mathematics*, 82 :15–25, 1998.

[28] E. Angel and V. Zissimopoulos. On the classification of NP-complete problems in terms of their correlation coefficient. *Discrete Applied Mathematics*, 99 :261–277, 2000.

[29] E. Anshelevich, A. Dasgupta, J. Kleinberg, É. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *In Proc. of FOCS 2004*, pages 295–304, 2004.

[30] K.R. Apt. *Principles of Constraint Programming*. Cambridge University Press, 2003.

[31] A. Archer and E. Tardos. Truthful mechanisms for one-parameter agents. In *Proceedings of FOCS'2001*, pages 482–491, 2001.

[32] J. Aslam, A. Rasala, C. Stein, and N. Young. Improved bicriteria existence theorems for scheduling. In *Symposium on Discrete Algorithms (SODA) Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 846–847, 1999.

[33] V. Auletta, R. De Prisco, P. Penna, and P. Persiano. Deterministic truthful approximation mechanisms for scheduling related machines. In *In Proc. of STACS 2004*, pages 608–619, 2004.

[34] V. Auletta, R. De Prisco, P.P. Persiano, and G. Persiano. On designing truthful mechanisms for online scheduling. In *SIROCCO 2005*, pages 3–17. Springer, 2005. LNCS 3499.

[35] Y. Azar and M. Sorani. Truthful approximation mechanisms for scheduling selfish related machines. In *In Proc. of STACS 2005*, pages 69–82, 2005.

[36] U. Bagchi. Simultaneous minimization of mean and variation of flow time and waiting time in single machine systems. *Operations Research*, 37 :118–125, 1989.

[37] E. Bampis, R. Giroudeau, and J.-C. König. A heuristic for the precedence constrained multiprocessor scheduling problem with hierarchical communications. In H. Reichel and S. Tison, editors, *Proceedings of STACS*, volume 1770 of *LNCS*, pages 443–454. Springer-Verlag, 2000.

[38] E. Bampis, R. Giroudeau, and J.-C. König. On the hardness of approximating the precedence constrained multiprocessor scheduling problem with hierarchical communications. Technical Report 34, LaMI, Université d'Évry Val d'Essonne, to appear in RAIRO Operations Research, 2001.

[39] E. Bampis, R. Giroudeau, and A. Kononov. Scheduling tasks with small communication delays for clusters of processors. In *SPAA*, pages 314–315. ACM, 2001.

[40] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. *Operations Research*, 36 :493–513, 1998.

[41] Y. Bartal, S. Leonardi, A. Marchetti-Spaccamela, J. Sgall, and L. Stougie. Multiprocessor scheduling with rejection. *SIAM Journal on Discrete Mathematics*, 13(1) :64–78, 2000.

[42] P. Berman and M. Karpinski. 8/7-approximation algorithm for $(1, 2)$-TSP. In *Proceedings of SODA*, pages 641–648, 2006.

[43] C. Bird. On cost allocation for spanning tree : a game theoretic approach. *Networks*, 6 :335–350, 1976.

[44] M. Bläser and L.S. Ram. An improved approximation for TSP with distances one and two. In *Proceedings of FCT'2005*, volume 3623 of *LNCS*, pages 504–515. Springer, 2005.

[45] P. Brucker. *Scheduling Algorithms*. Springer, 2001.

[46] P. Brucker, A. Gladky, H. Hoogeveen, M.Y. Kovalyov, C. Potts, T. Tautenhahn, and S. van de Velde. Scheduling a batching machine. *Journal of scheduling*, 1 :31–54, 1998.

[47] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems i. *Discrete Applied Mathematics*, 65 :97–122, 1996.

[48] P. Brucker, J. Hurink, and F. Werner. Improving local search heuristics for some scheduling problems ii. *Discrete Applied Mathematics*, 72 :47–69, 1997.

[49] M. Bruglieri, F. Maffioli, and M. Ehrgott. Cardinality constrained minimum cut problems : complexity and algorithms. *Discrete Applied Mathematics*, 137 :311–341, 2004.

[50] J. Bruno, E.G. Coffman Jr., and R. Sethi. Algorithms for minimizing mean flow time. In *Proceedings of IFIP Congress*, pages 504–510. North-Holland, 1974.

[51] J.L. Bruno, E.G. Coffman, and R. Sethi. Scheduling independents tasks to reduce mean finishing time. *Communication of the ACM*, 17 :382–387, 1974.

[52] P.M. Camerini, G. Galbiati, and F. Maffioli. The complexity of multi-constrained spanning tree problems. In *Theory of Algorithms, Colloquium Pecs 1984*, pages 53–101, 1984.

[53] F. Cappello, P. Fraignaud, B. Mans, and A.L. Rosenberg. HiHCoHP-Towards a Realistic Communication Model for Hierarchical HyperClusters of Heterogeneous Processors. In *Proceedings of IPDPS'01*, 2001.

[54] J. Carlier and P. Villon. A new heuristic for the traveling salesman problem. *RAIRO*, 24 :245–253, 1990.

[55] B. Chen, C.N. Potts, and G.J. Woeginger. A review of machine scheduling : Complexity, algorithms and approximability. Technical Report Woe-29, TU Graz, 1998.

[56] T.C.E. Cheng, A. Janiak, and M.Y. Kovalyov. Bicriterion single machine scheduling with resource dependent processing times. *SIAM J. on Optimization*, 8(2) :617–630, 1998.

[57] G. Christodoulou, E. Koutsoupias, and A. Nanavati. Coordination mechanisms. In *In Proc. of ICALP 2004*, volume 3142 of *LNCS*, pages 345–357. Springer, 2004.

[58] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical report, GSIA, Carnegie Mellon University, 1976.

[59] E. Clarke. Multipart pricing of public goods. linear programming and vickrey auctions. Unpublished manuscript, 2001.

[60] A. Claus and D. Kleitman. Cost allocation for spanning tree. *Networks*, 3 :289–304, 1973.

[61] C.A. Coello Coello, D.A. Van Veldhuizen, and G.B. Lamont. *Evolutionary Algorithms for solving Multiobjective Problems*. Luwer, New York, 2002.

[62] R.K. Congram, C.N. Potts, and S.L. van de Velde. An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14 :52–67, 2002.

[63] R.W. Conway, W.L. Maxwell, and L.W. Miller. *Theory of Scheduling*. Addison-Wesley, 1967.

[64] J. Correa, A. Schulz, and N.S. Moses. Selfish routing in capacitated networks. *Mathematics in Operations Research*, 29 :961–976, 2004.

[65] H.A.J. Crauwels, C.N. Potts, and L.N. Van Wassenhove. Local search heuristics for the single machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 10 :341–350, 1998.

[66] P. Crescenzi, X. Deng, and C.H. Papadimitriou. On approximating a scheduling problem. *Journal of Combinatorial Optimization*, 5 :287–297, 2001.

[67] I. Curiel. *Cooperative Game Theory and Applications*. Kluwer Academic Publishers, 1997.

[68] A. Czumaj, P. Krysta, and B. Vöcking. Selfish traffic allocation for server farms. In *Proceedings of STOC 2002*, pages 287–296, 2002.

[69] A. Czumaj and B. Vöcking. Tight bounds for worst-case equilibria. In *In Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 413–420, 2002.

[70] P. Czyżak and A. Jaszkiewicz. Pareto simulated annealing - a metaheuristic technique for multiple-objective combinatorial optimization. *Journal of multi-criteria decision analysis*, 7(1) :34–47, 1998.

[71] V.G. Deineko and G.J. Woeginger. A study of exponential neighborhoods for the traveling salesman problem and the quadratic assignment problem. *Mathematical Programming, Ser. A*, 87 :519–542, 2000.

[72] N. Devanur, N. Garg, R. Khandekar, V. Pandit, A. Saberi, and V. Vazirani. Price of anarchy, locality gap, and network service provider game. In *The 1st Workshop on Internet and Network Economics (WINE 2005), 15-17 décembre 2005, Hong Kong*, volume 3828 of *LNCS*, pages 1046–1055. Springer, 2005.

[73] J. Dréo, A. Pétrowski, P. Siarry, and É Taillard. *Métaheuristiques pour l'optimisation difficile.* eyrolles, 2003.

[74] R. Dutta and G.N. Rouskas. On optimal traffic grooming in wdm rings. *IEEE Journal on selected areas in communications*, 20(1) :110–121, 2002.

[75] J. Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B 71*, pages 233–240, 1967.

[76] M. Ehrgott. *Multicriteria optimization*, volume 491 of *LNEMS*. Springer, 2000.

[77] L. Engebretsen. An explicit lower bound for TSP with distances one and two. *Algorithmica*, 35(4) :301–318, 2003.

[78] L. Engebretsen and M. Karpinski. Approximation hardness of TSP with bounded metrics. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Proceedings of ICALP'2001*, volume 2076 of *LNCS*, pages 201–212, 2001.

[79] L. Epstein and J. Sgall. Approximation schemes for scheduling on uniformly related and identical parallel machines. In *ESA, Lecture Notes in Computer Science 1643*, pages 151–162, 1999.

[80] O. Ergun, J.B. Orlin, and A.S. Feldman. Creating very large scale neighborhoods out of smaller ones by compounding moves : A study on the vehicle routing problem. WP 4393-02, MIT Sloan School of Management, 2002.

[81] A. Fabrikant, C.H. Papadimitriou, and K. Talwar. The complexity of pure nash equilibria. In *STOC'04*, pages 604–612, 2004.

[82] F.R. Fernandez, M.A. Hinojosa, and J. Puerto. Multi-criteria minimum cost spanning tree games. *European Journal of Operational Research*, 158(2) :399–408, 2004.

[83] D. Fotakis, S. Kontogiannis, E. Koutsoupias, M. Mavronicolas, and P. Spirakis. The structure and complexity of nash equilibria for a selfish routing game. In *In Proc. of ICALP 2002*, volume 2380 of *LNCS*, pages 123–134. Springer, 2002.

[84] C. Gagné, M. Gravel, and W.L. Price. Optimisation multi-objectifs à l'aide d'un algorithme de colonie de fourmis. *Information Systems and Operational Research (INFOR)*, 42(1) :23–42, 2004.

[85] X. Gandibleux, N. Mezdaoui, and A. Fréville. A tabu search procedure to solve multiobjective combinatorial optimization problems. In R. Gaballero, F. Ruiz, and R. Steuer, editors, *Advances in multiple objective and goal programming*, volume 455 of *LNEMS*, pages 291–300. springer, 1997.

[86] M.R. Garey and D.S. Johnson. *Computers and intractability. A guide to the theory of NP-completeness.* W. H. Freeman, San Francisco, 1979.

[87] M. Gendreau, G. Laporte, and Potvin J.-Y. Vehicle routing : modern heuristics. In E.H.L Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 311–336. Wiley, 1997.

[88] M.X. Goemans and D.P. Williamson. .879-approximation algorithms for max cut and max 2sat. In *Proceedings of STOC*, pages 422–431, 1994.

[89] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM*, 42(6) :1115–1145, 1995.

[90] R. Graham. Bounds on multiprocessor timing anomalies. *SIAM Jr. on Appl. Math.*, 17(2) :416–429, 1969.

[91] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling : A survey. *Annals of Discrete Mathematics*, 5 :287–326, 1979.

[92] D. Granot and G. Huberman. On minimum cost spanning tree games. *Mathematical Programming*, 21 :1–18, 1981.

[93] T. Groves. Incentive in teams. *Econometrica*, 41(4) :617–631, 1973.

[94] G. Gutin. Exponential neighbourhood local search for the traveling salesman problem. *Computers and Operations Research*, 26(4) :313–320, 1999.

[95] G. Gutin, A. Yeo, and A. Zverovitch. Exponential neighborhoods and domination analysis for the TSP. In G. Gutin and A.P. Punnen, editors, *Traveling salesman problem and its variations*, volume 12 of *Combinatorial Optimization*, pages 223–256. Kluwer, 2002.

[96] L.A. Hall, A.S. Schulz, D.B. Shmoys, and J. Wein. Scheduling to minimize average completion time : off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22 :513–544, 1997.

[97] M.P. Hansen and A. Jaszkiewicz. Evaluating the quality of approximations to the non-dominated set. Technical Report IMM-REP-1998-7, Institute Of Mathematical Modelling, Technical University of Denmark, 1998.

[98] P. Hansen. Bicriterion path problems. In G. Fandel and T. Gal, editors, *3rd Conf. Multiple Criteria Decision Macking Theory and Application*, volume 177 of *LNEMS*, pages 109–127. Springer, 1979.

[99] A. Hertz, B. Jaumard, C. Ribeiro, and W. Formosinho Filho. A multi-criteria tabu search approach to cell formation problems in group technology with multiple objectives. *RAIRO-Recherche Opérationnelle/Operations Research*, 28(3) :303–328, 1994.

[100] D.S. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM Journal on Computing*, 11 :555–556, 1982.

[101] D.S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems.* PWS Publishing Company, 1996.

[102] H. Hoogeveen. *Single-Machine Bicriteria Scheduling.* PhD thesis, Eindhoven University of Technology, 1992.

[103] H. Hoogeveen, M. Skutella, and G.J. Woeginger. Preemptive scheduling with rejection. In *ESA, Lecture Notes in Computer Science 1879*, pages 268–277, 2000.

[104] H.H Hoos and T. Stützle. *Stochastic local search foundations and applications.* Elsevier, Morgan Kaufmann Publisher, 2005.

[105] E. Horowitz and S. Sahni. Exact and approximate algorithms for scheduling non identical processors. *Journal of the Association for Computing Machinery*, 23 :317–327, 1976.

[106] J. Hromkovič. *Algorithmics for Hard Problems : Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*. Springer, 2001.

[107] J. Hurink. Efficient calculation of a best neighbor for a one machine batching problem. Technical Report 180, Osnabrücker Schriften zur Mathematik, 1996.

[108] J. Hurink. An exponential neighborhood for a one-machine batching problem. *OR Spectrum*, 21(4) :461–476, 1999.

[109] O. Ibarra and C.E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM*, 22 :463–468, 1975.

[110] N. Immorlica, L. Li, V.S. Mirrokni, and A. Schulz. Coordination mechanisms for selfish scheduling. In *The 1st Workshop on Internet and Network Economics (WINE 2005), 15-17 décembre 2005, Hong Kong*, volume 3828 of *LNCS*, pages 55–69. Springer, 2005.

[111] A. Itai, M. Rodeh, and S.L. Tanimoto. Some matching problems for bipartite graphs. *Journal of the ACM*, 25 :517–525, 1978.

[112] K. Jansen and L. Porkolab. Improved approximation schemes for scheduling unrelated parallel machines. In *STOC*, pages 408–417, 1999.

[113] A. Jaszkiewicz. Genetic local search for multi-objective combinatorial optimization. *European Journal of Operational Research*, 137 :50–71, 2002.

[114] A. Jaszkiewicz. Evaluation of multiple objective metaheuristics. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *LNEMS*, pages 65–89. Springer, 2004.

[115] D.B. Johnson and J.M. Lafuente. A controlled single pass classification algorithm with application to multilevel clustering. Information Science and Retrieval ISR-18, Cornell University, Oct 1970.

[116] D.S. Johnson and L.A. McGeoch. The traveling salesman problem : a case study in local optimization. In E.H.L. Aarts and J.K. Lenstra, editors, *Local search in combinatorial optimization*, pages 215–310. John Wiley & Sons, 1997.

[117] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–104. Plenum, 1972.

[118] D. E. Knuth. *The Art of Computer Programming : Sorting and Searching*, volume 3. Addison Wesley, 1998.

[119] E. Koutsoupias and C.H. Papadimitriou. Worst-case equilibria. In *In Proc. of STACS 1999*, volume 1563 of *LNCS*, pages 404–413. Springer, 1999.

[120] M.Y. Kovalyov. On one machine scheduling to minimize the number of late items and the total tardiness. Technical report, Institute of Engineering Cybernetics, Academy of Sciences of Byelorussian SSR, Minsk, Byelorussia, 1991. Preprint N4.

[121] A. Kumar and J. Kleinberg. Fairness measures for ressource allocation. In *Proceedings of 41st IEEE Symposium on Foundations of Computer Science*, pages 75–85, 2000.

[122] E.L. Lawler. A "pseudopolynomial" algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1 :331–342, 1977.

[123] E.L. Lawler. A fully polynomial approximation scheme for the total tardiness problem. *Operations Research Letters*, 1 :207–208, 1982.

[124] J.K. Lenstra, D.B. Shmoys, and E. Tardos. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming A*, 46 :259–271, 1990.

[125] J.-H. Lin and J.S. Vitter. ϵ-approximation algorithms with minimum packing constraint violation. In *STOC'92*, pages 771–782, 1992.

[126] R. Lipton, E. Markakis, and A. Mehta. Playing large gamed using simple strategies. In *ACM Conference on Electronic Commerce*, pages 36–41, 2003.

[127] K. Munagala, S. Babu, R. Motwani, and J. Widom. The pipelined set cover problem. In *Proceedings of the Tenth International Conference on Database Theory*, 2005.

[128] A. Munier and C. Hanen. An approximation algorithm for scheduling dependent tasks on m processors with small communication delays. In *IEEE Symposium on Emerging Technologies and Factory Automation*, Paris, 1995.

[129] J.F. Nash. Non-cooperative games. *Annals of Mathematics*, 54 :286–295, 1951.

[130] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. STOC'1999*, pages 129–140, 1999.

[131] M.J. Osborne and A. Rubinstein. *A course in game theory.* The MIT Press, 1994.

[132] C. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, 2001.

[133] C.H. Papadimitriou and K. Steiglitz. Combinatorial optimization : Algorithms and complexity. Dover Publication, Inc, 1982.

[134] C.H Papadimitriou and M. Yannakakis. The traveling salesman problem with distances one and two. *Mathematics Of Operations Research*, 18(1) :1–11, 1993.

[135] C.H. Papadimitriou and M. Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *Proceedings 41th Annual IEEE Symposium on Foundations of Computer Science*, pages 86–92, 2000.

[136] L. Paquete, M. Chiarandini, and T. Stützle. Pareto local optimum sets in biobjective traveling salesman problem : An experimental study. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for Multiobjective Optimisation*, volume 535 of *LNEMS*, pages 177–199. Springer, 2004.

[137] J. Park and A. Segev. Using common subexpressions to optimize multiple queries. In *Proceedings of the Fourth International Conference on Data Engineering*, pages 311–319, 1988.

[138] F. Pascual. *Optimisation dans les réseaux : De l'approximation polynomiale à la théorie des jeux.* PhD thesis, IBISC, Université d'Evry, 2006.

[139] C. Phillips, C. Stein, and J. Wein. Scheduling jobs that arrive over time. In *Proceedings of the Fourth Workshop on Algorithms and Data Structures*, volume 955 of *LNCS*, pages 86–97. Springer, 1995.

[140] C.N. Potts and S.L. van de Velde. Dynasearch-iterative local improvement by dynamic programming : Part 1, the traveling salesman problem. Technical report, Faculty of Mathematical Studies, University of Southampton,U.K., 1995.

[141] C.N. Potts and L.N. Van Wassenhove. A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33 :363–377, 1985.

[142] S. Rajagopalan and V.V. Vazirani. Primal-dual RNC approximation algorithms for set cover and covering integer programs. *SIAM journal on Computing*, 28 :526–541, 1999.

[143] A. Rasala, C. Stein, and E. Torng. Existence theorems, lower bounds and algorithms for scheduling to meet two objectives. In *Proceedings 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 723–731, 2002.

[144] R. Ravi and M. Goemans. The constrained minimum spanning tree problem. In R.G. Karlsson and A. Lingas, editors, *Proceedings of SWAT'1996*, volume 1097 of *LNCS*, pages 66–75, 1996.

[145] V.J. Rayward-Smith. UET scheduling with unit interprocessor communication delays. *Discrete Applied Mathematics*, 18 :55–71, 1987.

[146] C.R. Reeves. *Modern heuristic techniques for combinatorial problems.* Blackwell Scientific Publications, 1993.

[147] D.J. Rosenkrantz, R.E. Stearns, and P.M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM Journal on Computing*, pages 563–581, 1977.

[148] T. Roughgarden. Stackelberg scheduling strategies. In *Proceedings of STOC 2001*, pages 104–113, 2001.

[149] T. Roughgarden. The price of anarchy is independent of the network topology. In *Proceedings of STOC 2002*, pages 341–364, 2002.

[150] T. Roughgarden and É. Tardos. How bad is selfish routing. *Journal of the ACM*, 49(2) :236–259, 2002.

[151] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 249–260, 2000.

[152] R. Saad. Scheduling with communication delays. *JCMCC*, 18 :214–224, 1995.

[153] S. Sahni and T. Gonzalez. P-complete approximation problems. *Journal of the ACM*, 23(3) :555–565, 1976.

[154] V.I. Sarvanov and N.N. Doroshko. The approximate solution of the travelling salesman problem by a local search algorithm that searches neighborhoods of exponential cardinality in quadratic time (in russian). *Software : Algorithms and Programs*, 31 :8–11, 1981. Mathematical Institute of the Belorussian Academy of Sciences, Minsk.

[155] V.I. Sarvanov and N.N Doroshko. The approximate solution of the travelling salesman problem by a local search algorithm with scanning neighborhoods of factorial cardinality in cubic time (in russian). *Software : Algorithms and Programs*, 31 :11–13, 1981. Mathematical Institute of the Belorussian Academy of Sciences, Minsk.

[156] P. Schuurman and T. Vredeveld. Performance guarantees of local search for multiprocessor scheduling. In *Proc. IPCO'2001*, volume 2081 of *LNCS*, pages 370–382. Springer, 2001.

[157] T.K. Sellis. Multiple-query optimization. *Transactions on Database Systems*, 13(1) :23–52, 1988.

[158] T.K. Sellis and S. Ghosh. On the multiple-query optimization. *IEEE Transactions on Knowledge and Data Engineering*, 2(2) :262–266, 1990.

[159] P. Serafini. Some consideration about computational complexity for multi objective combinatorial problems. In J. Jahn and W. Krabs, editors, *Recent advances and historical development of vector optimization*, volume 294 of *LNEMS*. Springer, 1986.

[160] P. Serafini. Simulated annealing for multiple objective optimization problems. In *Proceedings of the tenth international conference on multiple criteria decision making*, pages 87–96, 1992.

[161] B. Shepherd and A. Vetta. The demand matching problem. In *Proc. IPCO'2002*, volume 2337 of *LNCS*, pages 457–474. Springer, 2002.

[162] D.B. Shmoys and E. Tardos. An approximation algorithm for the generalized assignment problem. *Mathematical Programming A*, 62 :461–474, 1993.

[163] W.E. Smith. Various optimizers for single-stage production. *Naval Research Logistic Quaterly*, 3 :59–66, 1956.

[164] C. Stein and J. Wein. On the existence of schedules that are near-optimal for both makespan and total weighted completion time. *Operations Research Letters*, 21(3) :115–122, 1997.

[165] E. Talbi, M. Rahoual, M. Mabed, and C. Dhaenens. A hybrid evolutionary approach for multicriteria optimization problems : Application to the flow shop. In E. Zitzler, K. Deb, L. Thiele, C.A. Coello Coello, and D. Corne, editors, *Proceedings of EMO 2001*, volume 1993 of *LNCS*, pages 416–428. Springer, 2001.

[166] I.H. Toroslu and A. Cosar. Dynamic programming solution for multiple query optimization. *Information Processing Letters*, 92(3) :149–155, 2004.

[167] P. Toth and D. Vigo. *The Vehicle Routing Problem.* Society for Industrial and Applied Mathematic, 2001.

[168] M.A. Trick. Scheduling multiple variable-speed machines. In *1st Conference of Integer Programming and Combinatorial Optimization*, pages 485–494, 1990.

[169] TSPLIB. http ://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsplib.html.

[170] V. Vazirani. *Approximation algorithms.* Springer, 2001.

[171] A. Vetta. Nash equilibria in competitive societies with applications to facility location, traffic routing and auctions. In *In Proc. of FOCS 2002*, pages 416–425, 2002.

[172] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *J. Finance*, 16 :8–37, 1961.

[173] G.J. Woeginger. When does a dynamic programming formulation guarantee the existence of an FPTAS ?, 2001. Electronic Colloquium on Computational Complexity, Report 84 (short version appeared in SODA'99, 820–829).

[174] E. Zitzler. *Evolutionnary Algorithms for Multiobjective Optimization : Methods and Applications.* PhD thesis, Swiss federal Institute of Technology Zurich, November 1999.

[175] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms on test functions of different difficulty. In A.S. Wu, editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 121–122, 1999.

[176] E. Zitzler, M. Laumanns, and S. Bleuler. A tutorial on evolutionary multiobjective optimization. In X. Gandibleux, M. Sevaux, K. Sörensen, and V. T'kindt, editors, *Metaheuristics for multiobjective optimisation*, volume 535 of *LNEMS*. Springer, 2004.

Index

Index