Université de Montréal

Méthodes heuristiques pour un problème d'ordonnancement avec contraintes sur les ressources

par
Véronique Bouffard

Département d'informatique et de recherche opérationnelle
Faculté des arts et sciences

Mémoire présenté à la Faculté des études supérieures
en vue de l'obtention du grade de maître ès sciences (M.Sc.)
en informatique

avril 2003

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Méthodes heuristiques pour un problème d'ordonnancement avec contraintes sur les ressources

présenté par :

Véronique Bouffard

a été évalué par un jury composé des personnes suivantes :

Michel Gendreau
président-rapporteur

Jacques A. Ferland
directeur de recherche

Michael Florian
membre du jury

Mémoire accepté le: 11 juin 2003

Sommaire

Notre étude présente différentes méthodes heuristiques adaptées à la résolution d'un problème d'ordonnancement de tâches avec contraintes sur les ressources. Ce problème consiste à déterminer un horaire pour un ensemble de tâches consommant des ressources disponibles en quantité limitée. Chaque tâche, de durée prédéterminée, doit être exécutée obligatoirement à l'intérieur d'une fenêtre de temps donnée. Pour la résolution du problème, nous utilisons une approche par pénalités où la quantité de ressources excédentaires nécessaire est minimisée.

Nous nous concentrons principalement sur la méthode de descente pure, la méthode de recherche taboue, la méthode du recuit simulé et trois différentes méthodes d'acceptation avec seuil. Trois techniques d'amélioration sont aussi étudiées : la diversification par construction aléatoire, la diversification de premier ordre et la recherche à voisinage variable. Nous effectuons une étude comparative pour déterminer la stratégie la plus efficace pour résoudre ce problème en particulier. Les résultats indiquent qu'une combinaison du recuit simulé et de la recherche à voisinage variable semble être la stratégie à privilégier.

Nous terminons avec une discussion concernant des pistes qu'il serait intéressant d'explorer pour améliorer l'efficacité des méthodes pour résoudre le problème d'ordonnancement avec contraintes sur les ressources à l'étude.

Mots Clés

Méthodes heuristiques, ordonnancement, recherche taboue, recuit simulé, acceptation avec seuil, diversification, recherche à voisinage variable, méthode constructive.

Summary

Our study review different heuristic methods that we adapt for the resolution of the resource-constrained scheduling problem. In this problem, the goal is to determine a schedule for a set of tasks requiring resources to be completed. Resource availabilities are limited. Each task last a certain amount of time and must be completed in a specified time window. We use a penalty approach where the sum of the resource constraint violations is minimized.

We study a steepest descent method, a tabu search method, a simulated annealing method and three different threshold accepting methods. Three improving strategies are also studied : a diversification based on randomly constructive solution, a first order diversification strategy and a variable neighborhood search approach. A comparative study is carried out to determine the most efficient technique to this problem. Results indicate that a combination of the simulated annealing and the variable neighborhood search techniques seems to be the best strategy.

We conclude with a discussion about promising avenues to increase the efficiency of the methods to solve the resource-constrained scheduling problem.

Key Words

Heuristics methods, scheduling, tabu search, simulated annealing, threshold accepting, diversification, variable neighborhood search, constructive method.

Table des matières

Liste des figures

Liste des tableaux

Liste des abréviations

ASS	:	Acceptation avec seuil standard
ASDM	:	Acceptation avec seuil à détérioration maximale
ASGDA	:	Acceptation avec seuil - grand déluge
DCA	:	Diversification par construction aléatoire
DPO	:	Diversification de premier ordre
DP	:	Descente pure
NST	:	Neighborhood Search Technique
RCS	:	Resource-Constrained Scheduling
RS	:	Recuit simulé
RT	:	Recherche taboue
RVV	:	Recherche à voisinage variable
RVVE	:	Recherche à voisinage variable avec échanges
RVVI	:	Recherche à voisinage variable imbriqué
TRV	:	Technique de recherche dans le voisinage

Remerciements

Je tiens d'abord à remercier profondément mon directeur de recherche, monsieur Jacques A. Ferland, qui a su me faire confiance dans tout ce que nous avons entrepris ensemble au cours de ces deux années de travail. Plusieurs embûches et changements de parcours ont parsemé notre collaboration mais sa gentillesse, sa grande compréhension et son incroyable sens de l'humour ont rendu ces épreuves faciles à surmonter. Il a ce merveilleux don de tirer le meilleur de nous sans trop en demander.

Je tiens aussi à remercier monsieur Guy Lapalme sans qui je n'aurais peut-être pas continué mes études jusqu'à ce niveau. Il m'a offert ma première expérience en recherche et m'a encouragée à poursuivre à la maîtrise en me faisant participer aux concours de bourses provinciaux et nationaux. Je tiens d'ailleurs à remercier le Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR) pour m'avoir accordé une bourse qui m'a permis de me consacrer entièrement à mes recherches. Dans ce sens, je tiens également à remercier la compagnie GIRO Inc. qui m'a remis une bourse de maîtrise.

Un petit mot aussi pour mes collègues de recherche opérationnelle de deuxième cycle, Claudine, Jean-Nicolas et Carl. Je me souviendrai toujours de ces cours presque privés que nous avons eu (l'optimisation n'étant pas la matière préférée des informaticiens !) et surtout de l'entraide mutuelle qui en résultait.

Merci bien sûr à ma coloc et amie, Hélène, qui m'a toujours encouragée. Je ne peux compter les fois où je l'ai entendue me dire : *"C'est toi la meilleure !"*. Merci de croire en moi et d'arriver à me faire oublier mes tracas.

Finalement, un merci spécial à ma famille et surtout à mes parents. Leur grande fierté à mon égard m'a toujours poussée à me dépasser sans jamais me relâcher. Surtout, je ne pourrai jamais assez les remercier de m'avoir permis d'étudier l'esprit tranquille en comblant le moindre de mes besoins.

Chapitre 1

Introduction

L'ordonnancement, qui peut être décrit comme un problème d'allocation de ressources sujet à des contraintes, est une des activités centrales de la planification et du contrôle de la production. Dès qu'il est nécessaire de répartir ou d'organiser un travail entre plusieurs machines ou même plusieurs personnes, nous faisons face à ce type de problème. La multitude de champs d'application de ceux-ci explique par ailleurs le fait que plusieurs variantes de ce problème soient largement étudiées dans la littérature.

Le problème d'ordonnancement de tâches avec contraintes sur les ressources dont il est question dans cet ouvrage en est un exemple. Dans la version étudiée, le problème consiste à déterminer un horaire pour un ensemble de tâches nécessitant des ressources pour leur exécution en respectant les temps de début et d'échéance fixés pour chacune des tâches et les disponibilités des ressources à chaque période. Deux suppositions importantes sont faites afin de diminuer la complexité du problème. Premièrement, il n'y a pas de préemption dans l'exécution des tâches, c'est-à-dire qu'une tâche commencée ne peut être interrompue et doit ainsi être complétée entièrement. Deuxièmement, il n'y a aucune relation de précédence entre les tâches.

Malgré ces deux suppositions, ce problème demeure complexe puisqu'il fait partie de la classe des problèmes NP-difficiles, c'est-à-dire qu'aucun algorithme pouvant résoudre ce problème en temps polynomial n'est connu. Ce fait peut être démontré en se basant sur les travaux de Blazewicz [5]. Ainsi, les techniques classiques utilisant l'énumération implicite de l'ensemble des solutions ne sont pas applicables pour les grands problèmes de la vie courante malgré les progrès réalisés ces dernières années. Plus souvent qu'autrement, des techniques constructives et amélioratives (heuristiques) doivent être employées pour résoudre efficacement les

problèmes d'ordonnancement. Nous nous concentrons dans cette étude sur cinq d'entre elles : une méthode constructive et quatres méthodes de recherche locale (méthode de descente pure, de recherche taboue, du recuit simulé et d'acceptation avec seuil). Diverses stratégies d'amélioration sont aussi testées pour chacune d'elles, dont deux recherches à voisinage variable.

Les objectifs principaux de notre étude sont d'adapter les différentes méthodes heuristiques générales mentionnées pour la résolution de ce problème en particulier, d'analyser différentes composantes de ces méthodes et surtout de comparer l'efficacité relative de ces techniques à résoudre le problème donné. Sans réinventer ces techniques, déjà fort étudiées, nous pensons que cette étude, effectuée sur une variante simple du problème, peut servir à faire ressortir les meilleurs atouts et surtout les faiblesses des techniques comparées.

Le prochain chapitre de cet ouvrage décrit en détail le problème d'ordonnancement avec contraintes sur les ressources, sujet de notre étude. Le chapitre 3 contient ensuite une revue exhaustive de la littérature sur ce problème. Les chapitres suivants présentent respectivement la méthode constructive (chapitre 4), et les différentes techniques de recherche locale (chapitre 5) et stratégies d'amélioration (chapitre 6) testées. Finalement, le chapitre 7 résume les expérimentations faites et les résultats numériques obtenus, et le chapitre 8 termine avec les conclusions à apporter suite à cette étude et les considérations futures.

Chapitre 2

Description du problème et du modèle mathématique

Le problème d'ordonnancement avec contraintes sur les ressources, connu sous le nom de *Resource-constrained scheduling problem (RCS)*, consiste, dans la variante étudiée, à déterminer un horaire pour N tâches. Pour formuler le problème, nous utilisons la notation suivante :

- → L'horizon de temps est partitionné en **périodes** j, $j = 1, 2, \ldots, J$, de longueur fixe.
- → Chaque type de **ressource** l, $l = 1, 2, \ldots, L$, est disponible en quantité limitée B_{lj} pour chaque période j.
- → Chaque **tâche** i, i=$1, 2, \ldots, N$,
  - ne peut être commencée avant une certaine période r_i (temps le plus tôt de la tâche),
  - s'échelonne sur un nombre prédéterminé de périodes d_i (durée),
  - nécessite une quantité donnée a_{il} de chaque ressource l à chacune des périodes de son exécution,
  - et doit être terminée au plus tard à une certaine période $f_i \leq J$ (échéance de la tâche).

Un horaire est représenté par le vecteur x, où chaque variable de décision x_i dénote la période de départ de la tâche i, i=1,2,...,N. Par définition, un horaire x est dit admissible si chaque tâche i est débutée à une de ses périodes de départ admissibles J_i, où

$$J_i = \{j : r_i \leq j \leq f_i - d_i + 1\}.$$

Pour formuler les contraintes sur les ressources, définissons, pour un horaire x, le sous-ensemble $E_j(x)$ des tâches en cours d'exécution à la période j, $j = 1, 2, \ldots, J$, par

$$E_j(x) = \{i : x_i \leq j \leq x_i + d_i - 1\}.$$

Un horaire admissible est donc réalisable si pour l, $l = 1, 2, \ldots, L$, et pour j, $j = 1, 2, \ldots, J$,

$$\sum_{i \in E_j(x)} a_{il} \leq B_{lj}.$$

L'**objectif** est de déterminer un horaire admissible réalisable. Pour ce faire, nous avons utilisé une approche par pénalités, où la somme des violations des contraintes sur les ressources est minimisée. Ce problème peut être formulé comme suit :

$$(P) \qquad \text{Min } P(x) = \sum_{1 \leq j \leq J} \sum_{1 \leq l \leq L} \max\{0, \sum_{i \in E_j(x)} a_{il} - B_{lj}\}$$

$$\text{Sujet à} \qquad x_i \in J_i, \qquad i = 1, 2, \ldots, N.$$

Bien sûr, lorsque $P(x) = 0$, l'horaire admissible x est alors réalisable. Par contre, s'il n'y a aucun horaire admissible satisfaisant toutes les contraintes de ressources, alors la solution optimale de (P) correspond à la solution admissible minimisant la quantité supplémentaire de ressources nécessaire à son exécution.

Chapitre 3

Littérature

Depuis plus de quarante ans maintenant, plusieurs problèmes d'ordonnancement différents ont été définis et étudiés. Les variantes du problème d'ordonnancement les plus étudiées sont les problèmes à machine unique (*single-machine*) et les problèmes d'ordonnancement de projets (*projects scheduling*) et leurs cas spéciaux, comme les problèmes d'atelier généralisés (*job shop problem*) et linéaires (*flow shop problem*). Les problèmes de la vie réelle quant à eux sont souvent très complexes et incluent une variété de contraintes spécifiques.

3.1 Variantes

Un exemple typique d'étude sur le problème d'ordonnancement à machine unique est celle de Janiak [31]. Les objectifs à atteindre pour ce problème sont habituellement la minimisation du retard total de chaque tâche par rapport à son échéance (Koulamas [37]), la minimisation de la durée totale (Baker [3] et Morton et Pentico [46]) et finalement, la minimisation, pondérée ou non, du nombre de tâches en retard (Potts et Van Wassenhove [53], Moore [45]).

Les problèmes d'ordonnancement de projets, où un projet est composé de tâches reliées entre elles par des relations de précédence, sont de loin les plus étudiés. Dans ces problèmes, les temps le plus tôt et les échéances ne sont pas spécifiés pour les tâches individuelles, mais plutôt sur le projet dans son ensemble. Plusieurs variantes de ce problème ont été développées avec différents objectifs (critère d'évaluation de la qualité de l'horaire), différents types de ressources (renouvelables ou non, préemptives ou non), et différentes contraintes. Parmi les travaux récents résumant les recherches effectuées sur ce problème, mentionnons

ceux de Brucker *et al.* [9], de Özdamar et Ulusoy [51] et de Icmeli *et al.* [30]. Deux cas spéciaux de ce problème sont aussi beaucoup documentés : le problème d'atelier généralisé et linéaire. Dans le problème d'atelier généralisé (*job shop problem*), le but est de trouver un horaire pour un ensemble de projets contenant chacun des tâches devant être exécutées selon un certain ordre sur des machines spécifiques. L'horaire, constitué des temps de départ de chaque tâche sur chaque machine, doit être optimal selon certains critères. Le problème d'atelier linéaire (*flow shop problem*) est plus restrictif car l'ordre de passage des tâches sur chaque machine doit être le même pour chaque projet.

3.2 Approches de résolution

Au niveau de la résolution de ces problèmes de type combinatoire, deux approches globales sont généralement à considérer : les méthodes exactes et les méthodes heuristiques. La différence majeure réside dans le fait que les méthodes exactes résolvent le problème jusqu'à l'obtention d'une solution optimale tandis que les méthodes heuristiques tentent plutôt de trouver une bonne solution dans un temps raisonnable.

3.2.1 Méthodes exactes

Les méthodes exactes requièrent habituellement beaucoup de temps et ne réussissent en temps normal qu'à résoudre certaines formes peu complexes du problème. Comme aucun algorithme exécutable en temps polynomial n'existe pour résoudre ces problèmes exactement, le temps de résolution augmente rapidement avec la taille du problème. Les problèmes de la vie courante sont ainsi presqu'impossibles à résoudre à l'aide d'une méthode exacte.

Les premiers travaux concernant les méthodes exactes et les problèmes d'ordonnancement ont surtout porté sur des méthodes utilisant la programmation dynamique (Carruthers et Battersby [11], Held et Karp [29]) dans les années soixante. Par la suite, plusieurs chercheurs se sont concentrés sur des algorithmes basés sur une énumération de type séparation et évaluation. Parmi les plus récents travaux sur ce sujet, mentionnons ceux de Demeulemeester et Herroelen [17, 18],

Mingozzi [43], Brucker *et al.* [10] et Sprecher [54]. Ces méthodes ont par contre toutes été développées pour le problème d'ordonnancement de projets, qui inclut des précédences sur les tâches. Elles ne réussissent de plus qu'à résoudre des petits problèmes, composés de dix à vingt tâches. Patterson présente un survol des méthodes exactes développées pour ce problème dans [52].

3.2.2 Méthodes heuristiques

Depuis plus de dix ans, les chercheurs se concentrent principalement sur les méthodes heuristiques pour la résolution des problèmes d'ordonnancement. Les publications dans ce domaine sont maintenant très nombreuses et variées. Les méthodes heuristiques peuvent être séparées en deux classes : les méthodes à une seule solution et les méthodes à population de solutions. Chacune de ces classes est composée de méthodes constructives, où la solution est construite à partir de rien et de méthodes amélioratives, où une solution est améliorée itérativement [22].

Pour une description et une classification complète et récente des méthodes heuristiques développées pour les problèmes d'ordonnancement de projets ainsi qu'une évaluation comparative de celles-ci, les travaux de Kolish et Hartmann [36] et de Hartmann et Koolish [28] sont à consulter.

Méthodes utilisant une population de solutions

Les deux principales méthodes basées sur une population de solutions sont la méthode de colonie de fourmis (*Ant Colony Method*) et la méthode génétique.

La méthode de colonie de fourmis est une méthode constructive introduite pour la première fois au début des années quatre-vingt-dix par Colorni, Dorigo et Maniezzo [15]. L'idée générale de cette méthode est de faire construire chaque solution d'une génération par des fourmis qui dispersent une certaine quantité de phéromone sur les chemins les ayant conduit à leur solution. Les fourmis de la génération suivante seront attirées par la phéromone et fouilleront plus intensivement autour des meilleures solutions. Les travaux de Bauer *et al.* [4] utilisent cette méthode pour solutionner le problème d'ordonnancement sur une

seule machine tandis que ceux de Colorni *et al.* [16] et Stützle [55] l'utilisent pour le problème d'atelier généralisé et linéaire. Dernièrement, Merkle *et al.* [41] ont développé une méthode de colonie de fourmis pour le problème d'ordonnancement de projets et l'ont comparée à plusieurs autres heuristiques avec beaucoup de succès en moyenne.

Plusieurs algorithmes utilisant le principe de la méthode génétique ont été développés pour différents problèmes d'ordonnancement. La méthode génétique est basée sur un encodage des solutions, sur des opérateurs de sélection et de croisement des solutions parents et sur un opérateur de mutation des enfants. À chaque itération, des solutions sont choisies parmi les solutions de la population et croisées pour en obtenir de nouvelles qui, elles, seront mutées afin d'en générer d'autres. Chen *et al.*[13], Murata *et al.* [47], et Nepalli *et al.*[48] ont appliqué ce principe au problème d'atelier linéaire. Gupta *et al.* se sont attardés quant à eux à la résolution d'un problème à une seule machine [24] visant à minimiser la variance du temps d'exécution total. Cheng *et al.* [14] proposent une revue des méthodes génétiques développées pour le problème d'atelier généralisé. Finalement, plusieurs auteurs se sont intéressés à cette technique pour le problème d'ordonnancement de projets dont Özdamar [50], Hartmann [27] et Alcaraz et Maroto [1].

Méthodes à une solution

Parmi les méthodes à une seule solution appliquées au problème d'ordonnancement, nous retrouvons principalement les méthodes utilisant des heuristiques d'insertion, et des variantes de la méthode du recuit simulé et de la méthode de recherche taboue. Ces méthodes fonctionnent toutes en construisant ou en améliorant itérativement une solution jusqu'à atteindre un certain objectif.

Les méthodes utilisant des heuristiques d'insertion sont des méthodes constructives. Habituellement employées pour des problèmes comportant des relations de précédence, ces méthodes trient, à chaque période, les tâches pouvant être planifiées immédiatement à l'aide d'un ou de plusieurs critères de priorité et tentent de les planifier en respectant les contraintes de ressources. Souvent ces méthodes sont exécutées plusieurs fois et/ou avec différents critères afin de générer différentes solutions et de ne garder que la meilleure parmi celles-ci. Une autre

astuce pour générer différentes solutions est d'ajouter un peu de hasard dans les critères de priorité. Les travaux de Boctor [6] et de Kolish [35], entre autres, explorent quelques-unes de ces avenues.

Les deux autres méthodes, la méthode de recherche taboue et la méthode du recuit simulé, démarrent avec une solution initiale, habituellement obtenue à l'aide d'une heuristique d'insertion, et tentent de l'améliorer en y apportant de petites modifications et ce jusqu'à ce que le processus stagne. Plusieurs chercheurs ont appliqué la méthode de recherche taboue à la résolution du problème d'ordonnancement de projets dont Lee et Kim [39], Baar *et al.* [2] et Gagnon [21]. D'autres l'ont fait pour le problème à une machine (Laguna *et al.* [38]) et pour les problèmes d'atelier (Nowicki et Smutnicki [49] et Liaw [40]).

Plusieurs chercheurs ont aussi développé des méthodes de recuit simulé pour les variantes du problème d'ordonnancement de projets ; Boctor [7] ainsi que Bouleimen et Lecocq [8] ont travaillé sur de tels algorithmes. Notre étude, quant à elle, a été fortement inspirée d'un article de Jeffcoat et Bulfin [32] dans lequel ils décrivent et implantent une méthode de recuit simulé pour le problème d'ordonnancement avec temps le plus tôt et échéance pour chaque tâche. Nous reprenons dans notre étude l'algorithme utilisé dans leurs travaux ainsi que les idées principales pour développer d'autres algorithmes de recherche locale, les implanter et les comparer. Ce problème, simplifié par rapport à celui impliquant des relations de précédence, nous a permis d'adapter facilement les méthodes connues à sa résolution. Il serait intéressant de s'inspirer de ces résultats pour poursuivre une étude similaire sur le problème avec relations de précédence.

Chapitre 4

Méthode constructive

La méthode constructive suivante est simple et directe. Premièrement, toutes les tâches sont ordonnées suivant un certain critère, et planifiées une à la suite de l'autre selon cet ordre, en tentant d'utiliser efficacement les ressources encore disponibles. Cette méthode est souvent qualifiée de vorace (*greedy*) car l'ordre des tâches à planifier est choisi a priori, et la période de départ de chaque tâche est fixée en fonction des tâches déjà planifiées sans remettre en question la planification déjà faite.

Pour ordonner les tâches, un critère défini en terme d'un sous-ensemble des ressources $\mathcal{L} \subseteq \{1, 2, \ldots, L\}$ est utilisé. Pour ce faire, à chaque tâche i est associée une valeur c_i égale au total des unités des ressources $l \in \mathcal{L}$ nécessaires à l'exécution de cette tâche ; c'est-à-dire que pour chaque tâche i, i=1,2,...,N,

$$c_i = d_i \sum_{l \in \mathcal{L}} a_{il}.$$

La période de départ de chaque tâche est déterminée séquentiellement selon l'ordre décroissant des valeurs c_i. Ainsi, les tâches nécessitant un plus grand nombre d'unités des ressources du sous-ensemble $\mathcal{L}$ seront exécutées en premier. Précisément, la période de départ de chaque tâche i est fixée, parmi ses périodes admissibles J_i, de telle sorte que la tâche est exécutée durant les périodes où la plus grande quantité de ressources $l \in \mathcal{L}$ est encore disponible. Cette méthode est résumée dans la figure 4.1.

Initialisation :

Soient :

$\mathcal{L} \subseteq \{1, 2, \ldots, L\}$, le sous-ensemble de ressources choisies

$x_i := 0$, i=1,2,...,N, l'horaire à construire

$c_i := d_i \sum_{l \in \mathcal{L}} a_{il}$, i=1,2,...,N, le critère d'ordre

$ba_{lj} := B_{lj}$, $l \in \mathcal{L}$, j=1,2,...,N, les ressources encore disponibles

Supposons que les tâches sont ordonnées de telle sorte que
$c_1 \geq c_2 \geq \cdots \geq c_N$

Algorithme :

POUR CHAQUE i DE 1 À N

$\quad$ *Résidu* $:= -\infty$

$\quad$ POUR CHAQUE s DE r_i À $f_i - d_i + 1$

$\quad\quad$ SI *Résidu* $< \sum_{l \in \mathcal{L}} \sum_{t=s}^{s+d_i-1} ba_{lt}$ ALORS

$\quad\quad\quad$ *Résidu* $:= \sum_{l \in \mathcal{L}} \sum_{t=s}^{s+d_i-1} ba_{lt}$

$\quad\quad\quad$ $x_i := s$

$\quad$ POUR CHAQUE t DE x_i À $x_i + d_i - 1$

$\quad\quad$ POUR CHAQUE $l \in \mathcal{L}$

$\quad\quad\quad$ $ba_{lt} := ba_{lt} - a_{il}$

x est l'horaire construit

FIG. 4.1 – Méthode constructive

Chapitre 5

Techniques de recherche dans le voisinage (recherche locale)

Les méthodes de recherche dans le voisinage (TRV), ou *Neighborhood Search Techniques (NST)*, sont des procédures itératives remplaçant l'horaire admissible courant x par un nouvel horaire x' situé dans son **voisinage** $N(x)$, jusqu'à ce qu'un horaire admissible x^* soit atteint, où x^* est acceptable par rapport à un certain critère. En général, la solution voisine $x' \in N(x)$ est générée en apportant une légère modification à x.

Toutes les techniques diffèrent essentiellement par la façon qu'elles ont de sélectionner la prochaine solution $x' \in N(x)$. La technique la plus simple, la méthode de *descente pure (DP)*, tente à chaque itération d'améliorer le plus possible la solution x jusqu'à l'atteinte d'un minimum local. La méthode de *recherche taboue (RT)* est similaire à celle-ci, à l'exception qu'elle permet de s'échapper d'un tel minimum local afin de continuer l'exploration des solutions admissibles.

Les deux autres méthodes, la méthode du *recuit simulé (RS)* et la méthode d'*acceptation avec seuil (AS)*, se basent plutôt sur un processus aléatoire pour déterminer une solution voisine de x. Elles diffèrent essentiellement par le critère que chacune utilise pour choisir si cette solution peut ou non devenir la prochaine solution courante.

Ce chapitre débute par la description de deux concepts communs à toutes les techniques de recherche locale étudiées : le choix du voisinage et de la solution initiale. Une description générale et détaillée de chacune des techniques suit dans les quatre sections suivantes.

5.1 Concepts communs

5.1.1 Notion de voisinage

Les techniques itératives que sont les méthodes de recherche locale peuvent être vue comme une suite de déplacements à l'intérieur de l'espace des solutions admissibles où, à chaque étape, le déplacement est limité vers une nouvelle solution admissible appartenant à un sous-ensemble de celles-ci situées dans le voisinage de la solution courante. Le voisinage employé contrôle alors cette démarche. Règle générale, plus le voisinage est large, meilleure est la qualité des optimums locaux et finaux. Par contre, plus le voisinage contient de solutions, plus il est long et difficile de trouver la meilleure solution dans celui-ci. Un compromis s'impose donc au niveau de la dimension du voisinage.

Dans notre implantation, le voisinage utilisé est celui introduit par Jeffcoat et Bulfin dans [32]. Pour chaque solution x, son voisinage $N(x)$ contient tous les horaires admissibles obtenus de x en déplaçant la période de départ d'une seule tâche d'une seule période. Il contient donc un maximum de $2 \times N$ solutions. Nous dénotons par $x(i,g)$ et $x(i,d)$, les horaires dérivés de x en devançant ou en retardant respectivement la tâche i :

$$x(i,g) = [x_1, x_2, \ldots, x_{i-1}, x_i - 1, x_{i+1}, \ldots, x_N],$$
$$x(i,d) = [x_1, x_2, \ldots, x_{i-1}, x_i + 1, x_{i+1}, \ldots, x_N].$$

L'horaire $x(i,g)$ ($x(i,d)$) est appelé "voisin de gauche (droite) par rapport à la tâche i". Si $x_i - 1 \geq r_i$, alors $x(i,g)$ est inclus dans $N(x)$ et, de façon similaire, $x(i,d)$ est inclus dans $N(x)$ si $x_i + 1 \leq f_i - d_i + 1$.

5.1.2 Solution initiale

Toute technique de recherche locale démarre avec une solution initiale. Plusieurs méthodes peuvent être utilisées pour générer cette solution initiale et son choix peut influencer l'efficacité des techniques de recherche dans le voisinage. Il est donc important de s'y attarder. Par contre, il est à noter que par la struc-

ture du voisinage employée, nous souhaitons qu'il existe, pour toutes paires de solutions x et x', une suite de mouvements permettant de se déplacer de x à x'. Donc, peu importe la solution initiale utilisée, la solution optimale peut toujours être atteinte.

Deux avenues sont généralement à explorer pour la génération de cette solution de départ. La première et la plus simple consiste à en générer une aléatoirement. Dans notre cas, ceci se résume à générer aléatoirement, pour chaque i=1,2,...,N, une période de départ x_i comprise entre r_i et $f_i - d_i + 1$. La deuxième avenue nécessite souvent un peu plus de connaissances sur le problème à résoudre. Elle consiste à utiliser une méthode constructive de type vorace comme celle présentée au chapitre 4. En moyenne, cette dernière technique nous permet d'obtenir des solutions initiales de meilleure qualité ; nous l'avons utilisée dans tous nos tests avec le sous-ensemble de ressources $\mathcal{L} = \{1, 2, \ldots, L\}$.

5.2 Technique de descente pure

À chaque itération de cette méthode de descente, aussi appelée "*steepest descent*", le meilleur horaire admissible x' dans le voisinage $N(x)$ de la solution courante x,

$$x' = \underset{z \in N(x)}{\operatorname{argmin}} \{P(z)\},$$

devient la solution courante pour la prochaine itération. La méthode de descente pure est résumée dans la figure 5.1.

5.2.1 Critère d'arrêt

Cette procédure s'arrête lorsqu'un premier minimum local est atteint, c'est-à-dire lorsque $P(x') \geq P(x)$. Toutes les autres techniques présentées dans ce chapitre comportent des mécanismes pour éviter d'être piégées dans le premier minimum local atteint. En effet, ce dernier peut être de très mauvaise qualité et dépend beaucoup de la solution initiale utilisée.

Initialisation :
Soient :
x^0, un horaire admissible initial
$x := x^0$, la solution courante
$x^* := x^0$, la meilleure solution rencontrée jusqu'ici
$fin :=$ FAUX

Algorithme :
TANT QUE $fin \neq$ VRAI
$PC := P(x^*)$

Exploration des voisins
POUR CHAQUE i DE 1 À N
SI $P(x^*) = 0$, ALORS
$fin :=$ VRAI

Voisin de gauche par rapport à la tâche i
SI $x(i,g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS
SI $P(x(i,g)) < P(x^*)$, ALORS
$x^* := x(i,g)$

Voisin de droite par rapport à la tâche i
SI $x(i,d)$ est admissible (c-à-d. $x_i + 1 \leq f_i - d_i + 1$), ALORS
SI $P(x(i,d)) < P(x^*)$, ALORS
$x^* := x(i,d)$

Évaluation du critère d'arrêt
SI $P(x^*) = PC$, ALORS
$fin :=$ VRAI

$x := x^*$

x est l'horaire admissible généré

FIG. 5.1 – Méthode de descente pure

5.3 Méthode de recherche taboue

La méthode de recherche taboue (*Tabu Search*) a été développée indépendamment par Glover[23] et Hansen[25]. Plusieurs variantes de cet algorithme peuvent être imaginées. Dans la nôtre, à chaque itération, un horaire admissible x' parmi un sous-ensemble de solutions $N_V(x) \subseteq N(x)$ est choisi comme solution courante pour la prochaine itération. Aussi longtemps que $P(x') < P(x)$, cette méthode se comporte de façon similaire à la méthode de descente. Par contre, dans le cas où $P(x') \geq P(x)$, se déplacer de x à x' n'entraîne aucune amélioration et peut même entraîner une détérioration de la fonction économique $P(x)$. Néanmoins, ce mouvement permet de s'échapper d'un minimum local pour continuer l'exploration et est donc permis.

Par contre, étant donné que la valeur de la fonction économique $P(x)$ n'est pas monotone décroissante, une mesure de sécurité pour empêcher la procédure d'entrer dans un mouvement cyclique est nécessaire, d'autant plus que notre définition du voisinage fait en sorte que si $x' \in N(x)$, alors $x \in N(x')$. Ce retour en arrière doit donc être enrayé par un mécanisme de mémoire empêchant le processus de revenir à une solution récemment visitée. Dans ce but, une **liste taboue(LT)**, de longueur limitée, est employée pour mémoriser les modifications dernièrement utilisées pour générer les solutions courantes.

Puisque seules les modifications sont mémorisées, et non les solutions, ce mécanisme a comme défaut de rendre taboues des solutions qui n'ont encore jamais été visitées. Étant donné que certaines de ces solutions dites taboues peuvent être vraiment bonnes, un **critère d'aspiration** est introduit pour contrecarrer le statut tabou d'une telle solution.

Les prochaines sections contiennent une description plus détaillée des différentes composantes de cette méthode dont l'exploration du sous-voisinage, le contenu de la liste taboue, et la définition du critère d'aspiration et des critères d'arrêt. La figure 5.2 quant à elle résume la procédure complète.

Initialisation :

Soient :

x^0, un horaire admissible initial

$x := x^0$, la solution courante

$x^* := x^0$, la meilleure solution rencontrée jusqu'ici

$nbIt := 0$, le nombre d'itérations

$nbItInutile := 0$, le nombre d'itér. successives sans améliorer x^*

$fin :=$ FAUX

Algorithme :

TANT QUE $fin \neq$ VRAI

$Z := \varnothing$

Déterminer le sous-ensemble de tâches $V \subseteq \{1, 2, \ldots, N\}$

Exploration des voisins

POUR CHAQUE $i \in V$

SI $P(x^*) = 0$, ALORS $fin :=$ VRAI

Voisin de gauche par rapport à la tâche i

SI $x(i, g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS

Critère d'aspiration

SI $P(x(i, g)) < P(x^*)$, ALORS

$x^* := x := x(i, g)$

$nbItInutile := 0$

ALLER À **mj** :

SI $(i, x_i - 1) \notin LT$, ALORS

$Z := Z \cup \{x(i, g)\}$

Voisin de droite par rapport à la tâche i

SI $x(i, d)$ est admissible (c-à-d. $x_i + 1 \leq f_i - d_i + 1$), ALORS

Critère d'aspiration

SI $P(x(i, d)) < P(x^*)$, ALORS

$x^* := x := x(i, d)$

$nbItInutile := 0$

ALLER À **mj** :

SI $(i, x_i + 1) \notin LT$, ALORS

$Z := Z \cup \{x(i, d)\}$

Remplacement de x

$x := \underset{z \in Z}{\text{argmin}} \{P(z)\}$

mj : Mettre à jour LT

Évaluation des critères d'arrêt
$nbIt := nbIt + 1$
$nbItInutile := nbItInutile + 1$

SI $nbIt \geq nbItMax$ OU $nbItInutile \geq nbItInutileMax$, ALORS
$fin :=$ VRAI

x^* est l'horaire admissible généré

FIG. 5.2 – Méthode de recherche taboue

5.3.1 Choix et exploration du sous-voisinage $N_V(x)$

Contrairement à la méthode de descente pure, à chaque itération de cette méthode, il est possible de n'explorer qu'une portion seulement du voisinage de la solution courante x, $N_V(x)$ afin d'y trouver la prochaine solution x'. L'exploration d'une partie du voisinage permet d'effectuer plus d'itérations pour explorer de façon plus superficielle le voisinage d'un plus grand nombre de solutions dans un même laps de temps.

Dans notre cas, le sous-ensemble $N_V(x)$ contient les solutions voisines de x, $x(i,g)$ et $x(i,d)$ pour chaque $i \in V \subseteq \{1,2,\ldots,N\}$. Le sous-ensemble V peut être sélectionné aléatoirement ou par une méthode plus complexe tentant de prédire quelles sont les tâches dont le déplacement de leur période de départ est le plus susceptible d'améliorer la valeur de la fonction économique de la solution courante. Les deux avenues décrites ci-dessous ont été explorées lors des expérimentations numériques rapportées au chapitre 7.

Sélection par épuisement d'une permutation aléatoire

Pour implanter cette sélection, une permutation des tâches $i \in \{1,2,\ldots,N\}$ est générée aléatoirement. À chaque itération, un nombre maximal $n \in [1,N]$ de tâches de cette permutation est exploré pour déterminer x', la prochaine solution. Il faut noter que toutes les tâches sont parcourues dans l'ordre de la permutation avant d'en générer une nouvelle même si la solution courante change au cours de

ce parcours.

Pour bien illustrer le parcours des tâches, notons *perm*, le vecteur contenant la permutation des tâches et $p \in \{1, 2, \ldots, N\}$, la position dans cette permutation de la dernière tâche utilisée pour générer le sous-voisinage à l'étape précédente. Dans le cas où $p+n \leq N$, le sous-ensemble V, à l'itération courante, est alors composé des tâches $perm_{p+1}$ à $perm_{p+n}$ et le parcours se fait dans cet ordre. Par contre, si $p+n > N$ (c'est-à-dire qu'il reste moins de n tâches à parcourir dans la permutation), alors une nouvelle permutation, notons-la $perm'$, est générée de telle sorte que ses $N-p$ premières tâches soient les tâches $perm_{p+1}$ à $perm_N$ (dans cet ordre) et que ses p dernières tâches soient composées d'une permutation des tâches $perm_1$ à $perm_p$. Le sous-ensemble V, à l'itération courante, est alors composé des tâches $perm'_1$ à $perm'_n$.

Sélection utilisant un critère basé sur les périodes

Le principe sous-jacent à cette sélection est qu'à chaque itération, les tâches dans V sont choisies en favorisant les plus prioritaires selon un critère reflétant la violation des contraintes sur les ressources.

Étant donné la solution courante x, la priorité p_i de la tâche i est spécifiée en terme de la violation des contraintes sur les ressources lors de la première et de la dernière périodes de son exécution. Ainsi, pour chaque tâche $i=1, 2, \ldots, N$,

$$p_i = P_{x_i}(x) + P_{x_i+d_i-1}(x)$$

où $P_j(x)$ est la violation associée à chaque période j, $j=1, 2, \ldots, J$:

$$P_j(x) = \sum_{l=1} \max\{0, \sum_{i \in E_j(x)} a_{il} - B_{lj}\}.$$

Nous escomptons que plus la priorité d'une tâche est élevée, plus nous avons avantage à déplacer le moment de son exécution.

Par contre, pour éviter d'être trop glouton, nous choisissons plutôt les tâches séquentiellement selon un processus probabiliste biaisé en faveur des tâches plus prioritaires. À chaque étape, le processus de sélection se résume comme suit :

1. Considérer les tâches dans l'ordre original $i=1,2,\ldots,N$ et soit p_i la priorité de la tâche i.
2. Déterminer un nombre aléatoire α dans l'intervalle $[0, \sum_{i=1}^{N} p_i]$.
3. Déterminer le plus petit indice τ tel que $\sum_{i=1}^{\tau} p_i \geq \alpha$.
4. La tâche d'indice τ est sélectionnée et sa priorité est mise à 0 (c-à-d. $p_\tau := 0$).

5.3.2 Liste taboue

La liste taboue LT et le voisinage sont les éléments les plus importants de cette méthode. Une attention particulière doit donc être portée à sa définition.

Dans notre cas, supposons que l'élément $x' \in N_V(x) \subseteq N(x)$ choisi comme solution courante pour la prochaine itération est $x(i,g)$ ou $x(i,d)$. Dans le nouvel horaire, la tâche i est débutée à la période x_i-1 ou x_i+1 plutôt qu'à la période x_i. Comme mesure de sécurité, nous interdisons à la tâche i d'être débutée à la période x_i dans la solution courante des $|LT|$ prochaines itérations, où $|LT|$ est un nombre fixe correspondant à la longueur de la liste taboue. Ainsi, un horaire admissible $z \in N_V(x)$ est appelé **tabou**, et ne peut être sélectionné comme solution courante pour la prochaine itération si $z = x(i,g)$ et la paire $(i, x_i-1) \in LT$, ou si $z = x(i,d)$ et la paire $(i, x_i+1) \in LT$.

Le mécanisme est implanté en utilisant une liste cyclique LT, mise à jour à chaque itération en y ajoutant la paire (i, x_i) correspondant à la tâche i modifiée et à la période x_i où cette tâche est débutée dans la solution courante. Puisque la longueur de la liste, $|LT|$, est fixe, si la liste est pleine quand la paire (i, x_i) y est ajoutée, la paire la plus ancienne est supprimée de la liste.

Longueur variable ou fixe

La longueur de la liste taboue, $|LT|$, est un paramètre important pour l'efficacité de la procédure. Une liste trop longue limite l'exploration des solutions étant donné que beaucoup de solutions restent taboues longtemps. D'un autre côté, une liste trop courte ne prévient pas le cyclage à court terme. Malheureusement, il est difficile d'en déterminer la longueur optimale a priori.

De plus, pour certains problèmes, il s'est avéré plus efficace de n'utiliser qu'une portion variable de la liste taboue [56]. Pour ce faire, à chaque itération, nous ne considérons que les $|\tilde{LT}|$ derniers éléments de la liste où $|\tilde{LT}|$ est un nombre aléatoire tel que

$$t_{min} \leq |\tilde{LT}| \leq |LT|$$

et $t_{min} = \lfloor 0.8 \times |LT| \rfloor$.

5.3.3 Critère d'aspiration

Nous utilisons le **critère d'aspiration** le plus commun qui consiste à accepter toute solution de meilleure qualité que la meilleure solution rencontrée jusqu'ici. Ainsi, même si la solution $z \in N_V(x) \subseteq N(x)$ est taboue, si sa valeur $P(z)$ est inférieure à celle de la meilleure solution x^* rencontrée jusqu'ici, $P(x^*)$, z devient la meilleure solution (remplaçant x^*) et la solution courante (remplaçant x) pour la prochaine itération.

5.3.4 Critères d'arrêt

Finalement, cette méthode se termine lorsqu'un des critères d'arrêt, spécifiés en fonction d'un nombre maximum d'itérations ($nbItMax$) et d'un nombre maximum d'itérations consécutives sans amélioration de la fonction économique ($nbItInutileMax$), est atteint. Évidemment, la procédure s'arrête aussi dans le cas où une solution réalisable est trouvée (c-à-d. $P(x) = 0$).

5.4 Méthode du recuit simulé

La troisième méthode étudiée, celle du recuit simulé (*Simulated Annealing*), est aussi une procédure itérative permettant des modifications n'améliorant pas nécessairement la fonction économique. Kirkpatrick *et al.* [34] et Cerny [12] ont été les premiers à résoudre un problème d'optimisation avec cette approche déjà bien connue en mécanique statistique et utilisée pour simuler l'évolution d'un

système physique instable par rapport à un point d'équilibre thermodynamique à une température donnée (voir [42]).

À chaque itération de cette technique probabiliste, un horaire admissible x' est choisi aléatoirement dans le voisinage $N(x)$ de l'horaire courant x. Cet horaire x' remplace x comme solution courante si $\Delta P := P(x') - P(x) < 0$. Par contre, x' peut remplacer x même si aucune amélioration de la fonction économique P n'est induite en allant de x à x' (i.e si $\Delta P \geq 0$). Le choix d'effectuer ou non ce mouvement ascendant dépend d'une probabilité décroissante par rapport à la valeur de ΔP et au nombre d'itérations déjà complétées. Précisément, x' remplace x avec une probabilité $\exp(-\frac{\Delta P}{T})$, où le paramètre T, appelé **température**, décroît avec le nombre d'itérations complétées. Pour simuler ce comportement, un nombre aléatoire r est généré uniformément dans l'intervalle $(0, 1)$ et si

$$r < \exp(-\frac{\Delta P}{T}),$$

alors x' remplace x. Cette procédure est résumée à la figure 5.3.

5.4.1 Parcours du voisinage

À chaque itération, les tâches $i \in \{1, 2, \ldots, N\}$ sont parcourues dans un ordre aléatoire. Pour l'exploration des voisins, nous procédons comme dans l'algorithme proposé par Jeffcoat et Bulfin dans [32]. Dans celui-ci, les voisins de x par rapport à chaque tâche i sont explorés de gauche à droite. La première solution améliorante rencontrée remplace la solution x. Si aucun des deux voisins n'est meilleur que x, le test probabiliste est effectué pour savoir si le voisin de gauche, ou sinon le voisin de droite, a la chance d'être accepté malgré la détérioration possiblement entraînée.

5.4.2 Procédure de refroidissemnt

Dans cette variante, les voisins admissibles obtenus à partir de toutes les tâches sont considérés avant de modifier la température T. Le paramètre $R \in (0, 1)$, appelé **facteur de refroidissement**, est utilisé pour diminuer la tempéra-

Initialisation :

Soient :

x^0, un horaire admissible initial
$x := x^0$, la solution courante
$x^* := x^0$, la meilleure solution rencontrée jusqu'ici
T^0, la température initiale
$T := T^0$, la température courante
$R \in (0,1)$, le facteur de refroidissement
$nbIt := 0$, le nombre d'itérations
$nbItInutile := 0$, le nombre d'itér. successives sans améliorer PC
$fin :=$ FAUX

Algorithme :

TANT QUE $fin \neq$ VRAI
$PC := P(x)$

Exploration des voisins
POUR CHAQUE $i \in \{1, 2, \ldots, N\}$
SI $P(x^*) = 0$, ALORS
$fin :=$ VRAI

TESTS DÉTERMINISTES
Voisin de gauche par rapport à la tâche i
SI $x(i,g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS
SI $\Delta P := P(x(i,g)) - P(x) < 0$, ALORS
$x := x(i,g)$
ALLER À **nouv** :

Voisin de droite par rapport à la tâche i
SI $x(i,d)$ est admissible (c-à-d. $x_i + 1 \leq f_i - d_i + 1$), ALORS
SI $\Delta P := P(x(i,d)) - P(x) < 0$, ALORS
$x := x(i,d)$
ALLER À **nouv** :

TESTS PROBABILISTES
Voisin de gauche par rapport à la tâche i
SI $x(i,g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS
Sélectionner un nombre aléatoire $r \in [0,1]$
$\Delta P := P(x(i,g)) - P(x)$
SI $r < \exp(-\frac{\Delta P}{T})$, ALORS
$x := x(i,g)$
ALLER À **nouv** :

Voisin de droite par rapport à la tâche i
SI $x(i,d)$ est admissible (c-à-d. $x_i+1 \leq f_i-d_i+1$), ALORS
Sélectionner un nombre aléatoire $r \in [0,1]$
$\Delta P := P(x(i,d)) - P(x)$
SI $r < \exp(-\frac{\Delta P}{T})$, ALORS
$x := x(i,d)$
ALLER À **nouv** :

nouv : SI $P(x) < P(x^*)$, ALORS
$x^* := x$

Ajustement de la température
$T := T \times R$

SI $P(x) < PC$, ALORS $nbItInutile := 0$

Évaluation des critères d'arrêt
$nbIt := nbIt + 1$
$nbItInutile := nbItInutile + 1$

SI $nbIt \geq nbItMax$ OU $nbItInutile \geq nbItInutileMax$, ALORS
$fin :=$ VRAI

x^* est l'horaire admissible généré

FIG. 5.3 – Méthode du recuit simulé

ture (c-à-d. $T := R \times T$).

5.4.3 Critères d'arrêt

Les critères d'arrêt sont définis en terme d'un nombre maximal de températures successives ($nbItMax$) et d'un nombre maximal de températures successives où la valeur de la solution courante PC avant l'exploration de chaque tâche ($nbItInutileMax$) n'est pas améliorée. Enfin, si une solution réalisable est atteinte ($P(x) = 0$), la procédure s'arrête aussi.

5.5 Méthode d'acceptation avec seuil

La méthode d'acceptation avec seuil (*Threshold Accepting*) est une procédure itérative proposée par Dueck *et al.* [20] qui peut être vue comme une variante déterministe du recuit simulé.

Comme dans la méthode du recuit simulé, à chaque itération, nous générons aléatoirement une solution $x' \in N(x)$. Dans ce cas-ci, la décision de remplacer x par x' ne dépend pas d'une probabilité mais plutôt d'une fonction $\gamma(x, x')$ qui mesure la qualité de l'horaire x', et d'un **seuil** dr qui contrôle en quelque sorte la détérioration. Bien entendu, lorsque x' est de meilleure qualité que x, le remplacement s'effectue immédiatement. Le parcours du voisinage utilisé avec le recuit simulé et décrit à la section 5.4.1 est employé. La figure 5.4 présente l'algorithme implanté.

5.5.1 Test d'acceptation

Trois variantes de cette méthode ont été développées par Dueck *et al.* [19, 20], chacune ayant sa propre définition de $\gamma(x, x')$ et dr. Ces trois variantes se résument comme suit.

Standard

Dans la variante standard, qui est en fait la version originale de l'algorithme, le seuil dr est mis à jour de façon similaire à la température T dans le recuit simulé. Ainsi, toutes les tâches sont explorées avec la même valeur de dr avant d'ajuster le seuil à l'aide d'un facteur $a \in (0, 1)$ (c-à-d. $dr := a \times dr$).

La fonction $\gamma(x, x')$ est définie comme suit :

$$\gamma(x, x') = P(x') - P(x).$$

Il s'ensuit que x' remplace x à condition que la détérioration induite par x' n'excède pas une certaine valeur limite (seuil) qui décroît avec le nombre d'itéra-

Initialisations :

Soient :

x^0, un horaire admissible initial

$x := x^0$, la solution courante

$x^* := x^0$, la meilleure solution rencontrée jusqu'ici

dr^0, un seuil initial

$dr := dr^0$, le seuil courant

$nbIt := 0$, le nombre d'itérations

$nbItInutile := 0$, le nombre d'itér. successives sans améliorer PC

$fin :=$ FAUX

Algorithme :

TANT QUE $fin \neq$ VRAI

$PC := P(x)$

Exploration des voisins

POUR CHAQUE $i \in \{1, 2, \ldots, N\}$

SI $P(x^*) = 0$, ALORS

$fin :=$ VRAI

Voisin de gauche par rapport à la tâche i

SI $x(i, g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS

SI $\gamma(x, x(i, g)) < dr$, ALORS

$x := x(i, g)$

ALLER À **nouv** :

Voisin de droite par rapport à la tâche i

SI $x(i, d)$ est admissible (c-à-d. $x_i + 1 \leq f_i - d_i + 1$), ALORS

SI $\gamma(x, x(i, d)) < dr$, ALORS

$x := x(i, d)$

ALLER À **nouv** :

nouv : SI $P(x) < P(x^*)$, ALORS

$x^* := x$

Ajuster dr

SI $P(x) < PC$, ALORS

$nbItInutile := 0$

Évaluation des critères d'arrêt
$nbIt := nbIt + 1$
$nbItInutile := nbItInutile + 1$

SI $nbIt \geq nbItMax$ OU $nbItInutile \geq nbItInutileMax$, ALORS
$fin :=$ VRAI

x^* est l'horaire admissible généré

FIG. 5.4 – Méthode d'acceptation avec seuil

tions complétées.

Grand déluge

Dans la méthode du grand déluge (*Great Deluge Algorithm*), le seuil dr correspond en quelque sorte à un certain niveau d'eau. La solution $x' \in N(x)$ remplace x comme solution courante seulement si sa qualité $P(x')$ est en-dessous du seuil, et ce, indépendemment de la qualité de la solution courante $P(x)$. Nous utilisons donc

$$\gamma(x, x') = P(x').$$

Le seuil dr décroît linéairement avec le nombre d'itérations. La décroissance est contrôlée par un seul paramètre, δ, comme suit :

$$dr := dr - \delta.$$

Le seuil initial est habituellement fixé à la valeur de la fonction économique de la solution initiale (c-à-d. $dr^0 := P(x^0)$).

Cette méthode a été pensée dans le contexte d'un problème de maximisation en faisant l'analogie avec un pays où il pleut tout le temps. Le pays s'inonde peu à peu. Pour survivre, il faut trouver une montagne toujours plus haute pour être à l'abri de la noyade. Le seuil représente ainsi le niveau d'eau, le facteur δ, la vitesse à laquelle le pays s'inonde et la qualité de la solution courante, la hauteur de la montagne sur laquelle nous nous trouvons.

Détérioration maximale

La méthode de détérioration maximale (*Record-to-Record Travel*) est semblable à la précédente en ce sens que le choix de remplacer ou non x par x' ne dépend pas de la valeur de $P(x)$, mais uniquement de $P(x')$. Ainsi,

$$\gamma(x, x') = P(x')$$

et x est remplacée par x' à condition que $P(x')$ n'excède pas un seuil spécifié en terme d'une détérioration maximale de x^* acceptable. Par conséquent,

$$dr := P(x^*) + \mu$$

où μ est un paramètre fixe. La diminution de la valeur du seuil dr est donc proportionnelle à celle de $P(x^*)$.

5.5.2 Critères d'arrêt

Les critères d'arrêt utilisés pour cette méthode sont, comme pour le recuit simulé, le nombre total de parcours complets de toutes les tâches ($nbItMax$) et le nombre de parcours complets de toutes les tâches où la valeur de la solution courante au début du parcours n'est pas améliorée ($nbItInutileMax$). Également, la procédure s'arrête lorsqu'une solution réalisable est générée (c-à-d. $P(x) = 0$).

Chapitre 6

Stratégies d'amélioration

Deux stratégies générales et complémentaires sont habituellement exploitées pour améliorer les techniques de recherche dans le voisinage. Il s'agit de l'intensification et de la diversification. La première vise à fouiller plus en profondeur dans les régions les plus prometteuses de l'espace des solutions admissibles. La diversification, quant à elle, a pour but l'exploration plus large de ce même domaine, c'est-à-dire l'exploration d'un maximum de ses recoins.

Nous nous concentrons sur la diversification étant donné que les expériences numériques semblent indiquer que les techniques de recherche dans le voisinage ont réellement avantage à être appliquées plusieurs fois à partir de solutions différentes pour des périodes plus courtes plutôt qu'une seule fois pour une plus longue période. Plusieurs stratégies peuvent être utilisées pour fouiller plus largement l'espace des solutions. Les prochaines sections décrivent celles auxquelles nous nous sommes attardés : la diversification par construction aléatoire, la diversification de premier ordre et la recherche à voisinage variable.

6.1 Diversification par construction aléatoire

La diversification par construction aléatoire (DCA) consiste à générer de nouvelles solutions initiales à l'aide d'une variante de la méthode constructive décrite au chapitre 4. Dans cette variante, au lieu de trier les tâches i selon l'ordre décroissant du critère c_i, celles-ci sont triées en utilisant une sélection proportionnelle. Ceci permet d'ajouter un peu d'aléatoire dans cette technique vorace et donc de générer différentes solutions en utilisant le même sous-ensemble $\mathcal{L}$ de ressources. La procédure pour déterminer un ordre des tâches selon une sélection

proportionnelle est similaire à celle employée à la section 5.3.1 pour sélectionner les tâches en utilisant un critère basé sur les périodes pour engendrer le sous-voisinage. Il suffit de remplacer p_i par c_i dans cette dernière procédure.

6.2 Diversification de premier ordre

Cette stratégie de diversification de premier ordre (DPO) a été introduite dans [33] pour le problème d'affectation quadratique. Elle se résume comme suit. Supposons que x^* est l'horaire admissible généré à l'aide d'une technique de recherche locale. Pour générer une nouvelle solution initiale à partir de x^*, il suffit de modifier les périodes courantes de départ de certaines tâches i de x_i^* à x_i^*+1 ou x_i^*-1, de façon à générer un nouvel horaire admissible en détériorant la valeur de P le moins possible, ou mieux, en l'améliorant. L'horaire admissible généré après la modification d'un certain pourcentage λ des tâches devient alors la nouvelle solution initiale.

Concrètement, cette technique est implantée de la même façon que la méthode de recherche taboue en utilisant la liste des tâches déjà modifiées comme liste taboue. De plus, lorsque cette technique est utilisée en combinaison avec une méthode de recherche taboue, il est important de mettre à jour la liste taboue à l'intérieur de la diversification de premier ordre, de façon à ne pas retourner à la solution x^* à partir de laquelle la nouvelle solution initiale est générée par la DPO.

6.3 Recherche à voisinage variable

L'idée de base de la recherche à voisinage variable (RVV), proposée par Hansen *et al.* [26] sous le nom de *Variable Neighborhood Search*, consiste à changer systématiquement de voisinage au cours d'une méthode de recherche locale. Ainsi, cette méthode explore différents voisinages de complexité croissante autour d'une même solution, et ne se déplace vers une nouvelle solution courante que lorsqu'il y a une amélioration de la fonction économique [44].

Initialisation :
Soient :
x^0, un horaire admissible initial
$x^* := x^0$, la meilleure solution rencontrée jusqu'ici
N^k, $k = 1, \ldots, \bar{k}$, les structures de voisinages
$nbIt := 0$, le nombre d'itérations
$nbItInutile := 0$, le nombre d'itér. successives sans améliorer x^*
$fin :=$ FAUX

Algorithme :
TANT QUE $fin \neq$ VRAI
$k := 1$

TANT QUE $k \leq \bar{k}$
SI $P(x^*) = 0$, ALORS $fin :=$ VRAI

Perturbation
Générer aléatoirement $x' \in N^k(x^*)$

Recherche locale
Appliquer une TRV avec x' comme solution initiale et N^k comme structure de voisinage. Soit x'' l'horaire admissible généré.

Descente ou non
SI $P(x'') < P(x^*)$, ALORS
$x^* := x''$
$k := 1$
$nbItInutile := 0$
SINON, ALORS
$k := k + 1$

Évaluation des critères d'arrêt
$nbIt := nbIt + 1$
$nbItInutile := nbItInutile + 1$

SI $nbIt \geq nbItMax$ OU
$nbItInutile \geq nbItInutileMax$, ALORS
$fin :=$ VRAI

x^* est l'horaire admissible généré

FIG. 6.1 – Méthode de recherche à voisinage variable

Comme son nom l'indique, cette méthode nécessite la définition de plusieurs structures de voisinage $N^k(x)$, $k = 1, \ldots, \bar{k}$, pouvant être imbriquées ($N^k(x) \subseteq N^{k+1}(x)$). À chaque itération, un minimum local x'' est identifié à partir de $x' \in N^k(x^*)$ à l'aide d'une TRV utilisant la structure de voisinage N^k. Si le minimum local x'' est meilleur que x^*, x'' remplace x^* et la prochaine recherche se fait en utilisant la structure de voisinage N^1. Sinon, la structure de voisinage N^{k+1} est employée pour la prochaine itération. La figure 6.1 résume cette méthode.

Il est important que x', qui sert de solution initiale pour la TRV, soit choisie de façon aléatoire dans $N^k(x^*)$ afin d'éviter le cyclage pouvant découler de l'usage d'une règle déterministe [44].

6.3.1 Structures de voisinage

Nous utilisons quatre structures de voisinage. Les trois premiers voisinages sont imbriqués ($N^1 \subseteq N^2 \subseteq N^3$) l'un dans l'autre tandis que le quatrième, N^4, plus complexe que les trois précédents, ne l'est pas (voir figure 6.2).

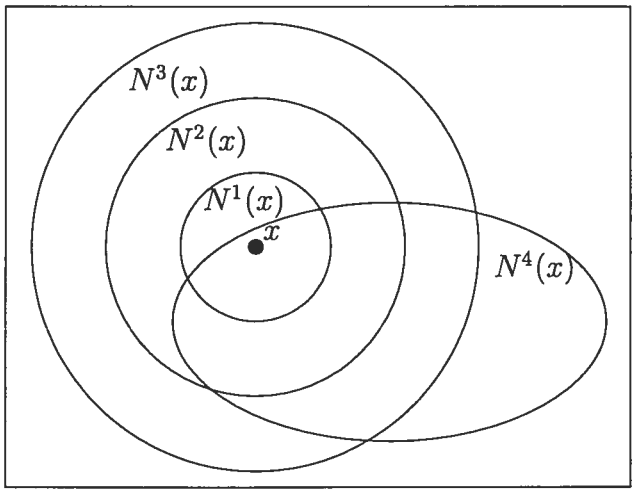

FIG. 6.2 – Structures de voisinage

La première structure de voisinage, N^1, est celle déjà décrite à la section 5.1.1.

La deuxième structure de voisinage, N^2, est très similaire à N^1 sauf que chaque tâche i peut être déplacée de une ou de deux périodes à gauche et à

droite. Donc, en plus de $x(i,g)$ et $x(i,d)$, les solutions $x(i,gg)$ et $x(i,dd)$, où

$$x(i,gg) = [x_1, x_2, \ldots, x_{i-1}, x_i - 2, x_{i+1}, \ldots, x_N] \text{ et}$$
$$x(i,dd) = [x_1, x_2, \ldots, x_{i-1}, x_i + 2, x_{i+1}, \ldots, x_N],$$

sont des voisins potentiels de x pour chaque tâche i, $i = 1, 2, \ldots, N$.

La troisième structure de voisinage, N^3, est très similaire à N^2 sauf que chaque tâche i peut aussi être déplacée de trois périodes à gauche et à droite. Donc, en plus de $x(i,g)$, $x(i,d)$, $x(i,gg)$ et $x(i,dd)$, les solutions $x(i,ggg)$ et $x(i,ddd)$, où

$$x(i,ggg) = [x_1, x_2, \ldots, x_{i-1}, x_i - 3, x_{i+1}, \ldots, x_N] \text{ et}$$
$$x(i,ddd) = [x_1, x_2, \ldots, x_{i-1}, x_i + 3, x_{i+1}, \ldots, x_N],$$

sont des voisins potentiels de x pour chaque tâche i, $i = 1, 2, \ldots, N$.

La quatrième structure employée, N^4, est très différente des trois autres. Les voisins de x sont déterminés en échangeant la période de départ de deux tâches différentes i et j. Pour chaque paire (i,j), $i, j = 1, 2, \ldots, N, i \neq j$, le voisin de x par rapport à celle-ci, est défini comme étant

$$x(i,j) = [x_1, \ldots, \underset{\uparrow}{x_j}, \ldots, \underset{\uparrow}{x_i}, \ldots, x_N].$$
$$\qquad\qquad\quad i \qquad\quad j$$

Ainsi, si $x_i \in J_j$ et $x_j \in J_i$, alors $x(i,j) \in N^4(x)$. Ce voisinage a le désavantage d'être de dimension très variable. En effet, il peut, dépendemment de la solution courante x, contenir entre zéro et $\frac{N!}{2\cdot(N-2)!} = \frac{N\cdot(N-1)}{2}$ solutions. L'exploration du voisinage peut donc être très longue et ardue s'il est grand, comme très courte et peu utile s'il est petit. Par contre, comme cette structure de voisinage n'est utilisée qu'en dernier recours, nous espérons que cette lacune n'aura pas beaucoup d'impact.

Deux méthodes de recherches à voisinage variable différentes sont considérées dans les tests numériques présentés au chapitre 7. Les deux méthodes utilisent chacune trois structures de voisinage ($\bar{k} = 3$). La première méthode, nous la nommons recherche à voisinage variable avec échanges (RVVE), utilise en ordre les structures de voisinage N^1, N^2 et N^4. La deuxième, nous la nommons recherche à voisinage variable imbriqué (RVVI), utilise plutôt les trois structures de voisinage

imbriquées N^1, N^2 et N^3.

6.3.2 Recherche à voisinage variable et recherche taboue

Lorsqu'une méthode de recherche taboue est utilisée dans le cadre d'une recherche à voisinage variable, il faut ajuster l'implantation au niveau du sous-voisinage et du contenu de la liste taboue pour chaque structure de voisinage.

Voisinages N^2 et N^3

Au niveau du sous-voisinage, la définition que nous employons pour N^1 doit être modifiée pour que N_V^2 inclue, pour chaque $i \in V$, les solutions $x(i, gg)$ et $x(i, dd)$, en plus des solutions $x(i, g)$ et $x(i, d)$. Similairement, N_V^3 doit inclure, pour chaque $i \in V$, les solutions $x(i, ggg)$ et $x(i, ddd)$, en plus des solutions $x(i, g)$, $x(i, d)$, $x(i, gg)$ et $x(i, dd)$. La sélection des tâches $i \in V$ se fait comme pour N_V^1.

Avec ces voisinages, la liste taboue est définie comme pour N^1, c'est-à-dire que si $x(i, g)$, $x(i, d)$, $x(i, gg)$ ou $x(i, dd)$ (et $x(i, ggg)$ ou $x(i, ddd)$ pour N^3) est choisie comme solution courante pour la prochaine itération alors la paire (i, x_i) s'ajoute à la fin de la liste. La vérification du caractère tabou d'une solution pour les structures N^2 et N^3 se fait de la même façon que pour la structure N^1.

Voisinage N^4

La détermination d'un sous-voisinage N_V^3 requiert un ajustement au niveau du contenu de V. En effet, l'ensemble V dans ce cas n'est plus un sous-ensemble des tâches mais plutôt un sous-ensemble des différentes paires de tâches. La sélection par épuisement d'une permutation aléatoire décrite à la section 5.3.1 est utilisée pour ce nouvel ensemble V, la permutation étant constituée dans ce cas de toutes les différentes paires de tâches.

En ce qui concerne la liste taboue pour la structure de voisinage N^3, à chaque fois que x' remplace la solution courante x, deux paires, soient (i, x_i) et (j, x_j), sont insérées dans la liste taboue plutôt qu'une seule. De plus, la vérification du

caractère tabou d'une solution pour la structure N^3 nécessite deux opérations au lieu d'une seule car une solution est considérée taboue si une des deux paires (i, x_j) ou (j, x_i) appartient à la liste taboue.

6.3.3 Recherche à voisinage variable et recuit simulé

Le recuit simulé a une structure qui s'accommode facilement à un changement de la structure du voisinage.

Voisinages N^2 et N^3

Le parcours des voisinages N^2 et N^3 est complété en considérant toutes les tâches $i \in \{1, 2, \ldots, N\}$ dans un ordre aléatoire. Pour la structure de voisinage N^2, nous effectuons d'abord, pour chaque tâche i, le test déterministe successivement avec $x(i, gg)$, $x(i, dd)$, $x(i, g)$ et $x(i, d)$ puis au besoin nous complétons avec le test probabiliste avec ces quatre solutions dans le même ordre. Pour la structure de voisinage N^3, le test déterministe est d'abord effectué sur, en ordre, $x(i, ggg)$, $x(i, ddd)$, $x(i, gg)$, $x(i, dd)$, $x(i, g)$ et $x(i, d)$. Au besoin, nous poursuivons avec le test probabiliste sur ces six solutions. Comme pour le voisinage N^1, la première solution qui satisfait un des tests remplace la solution courante x.

Voisinage N^4

Le parcours du voisinage N^4 est complété en considérant toutes les paires de tâches différentes dans un ordre aléatoire. Pour chaque paire de tâches (i, j), le test déterministe est d'abord effectué avec $x(i, j)$, suivi du test probabiliste. Comme pour les voisinages N^1, N^2 et N^3, la première solution admissible qui satisfait un des tests remplace la solution courante x.

6.3.4 Recherche à voisinage variable et acceptation avec seuil

Les ajustements de parcours pour les structures de voisinage N^2, N^3 et N^4 sont les mêmes que pour le recuit simulé.

6.3.5 Critères d'arrêt

L'algorthme de recherche à voisinage variable se termine soit parce qu'une solution réalisable est obtenue, soit parce qu'un nombre maximal d'itérations sont complétées ou soit parce qu'un nombre maximal d'itérations successives sans améliorer x^* sont complétées.

Chapitre 7

Résultats numériques

Puisque peu de travaux portent sur cette variante du problème d'ordonnancement avec contraintes sur les ressources, il est évidemment difficile d'essayer de comparer l'efficacité des méthodes présentées ici avec celles d'autres méthodes déjà étudiées et rapportées dans la littérature. En effet, la plupart des travaux sur l'ordonnancement portent sur des problèmes d'horaire de projet où il y a des relations de précédence entre les tâches, plutôt que des temps le plus tôt et le plus tard pour chacune d'elles. Nous avons donc seulement comparé entre elles les six méthodes de recherche dans le voisinage présentées, soit : la descente pure, la recherche taboue, le recuit simulé et les trois variantes de l'acceptation avec seuil (standard, grand déluge et détérioration maximale). Nous avons aussi vérifié la pertinence d'utiliser des stratégies d'amélioration.

La méthodologie employée pour effectuer l'ensemble des tests numériques est résumée à la section 7.1. La section 7.2 contient un compte-rendu des efforts investis pour choisir les valeurs à donner aux différents paramètres de chaque méthode. Ensuite, les résultats numériques concernant les stratégies d'amélioration sont décrits à la section 7.3. Finalement, dans la section 7.4, nous comparons les résultats obtenus avec chaque méthode ainsi qu'avec chaque méthode couplée à la technique d'amélioration la plus performante pour celle-ci.

7.1 Méthodologie

7.1.1 Génération des problèmes

Pour générer les problèmes dont nous nous sommes servis pour tester les différentes méthodes, nous avons utilisé une variante de la procédure décrite par Jeffcoat et Bulfin dans [32]. La différence entre notre façon et la leur réside dans la spécification des disponibilités des ressources B_{lj}. En effet, les problèmes générés en utilisant leur méthode sont trop faciles à résoudre selon nous.

Tous les problèmes employés comportent $N = 100$ tâches nécessitant $L = 4$ types de ressources pour être exécutées. L'exécution des tâches se fait sur un horizon de $J = 100$ périodes. Les durées d_i des tâches sont générées à partir d'une distribution uniforme discrète sur l'intervalle [1,20]. Chaque tâche nécessite une seule unité de ressource du premier type, simulant ainsi la necessité d'utiliser une machine pour accomplir cette tâche. Les besoins pour les trois autres ressources sont générés de façon à être distribués uniformément sur les intervalles [0,5], [0,10] et [0,15], respectivement.

Étant donné ces valeurs, les disponibilités en ressources ainsi que le temps le plus tôt et l'échéance pour chaque tâche sont ensuite définis de telle sorte qu'au moins une solution réalisable existe. Ainsi, chaque problème possède une solution réalisable x, c'est-à-dire pour laquelle $P(x) = 0$. Pour y arriver, un horaire fictif s est d'abord construit comme suit. La première tâche démarre à la période 1, la deuxième démarre à la période suivant la fin de la première (c-à-d. à la période $s_1 + d_1$), et ainsi de suite. Lorsque l'échéance d'une tâche i dépasse la période $J = 100$, celle-ci est démarrée à la période 1 plutôt qu'à la période $s_{i-1} + d_{i-1}$, et une deuxième séquence recommence. Ce processus s'arrête lorsque toutes les tâches ont été cédulées. Ensuite, pour chaque tâche i, nous déterminons son temps le plus tôt r_i et son échéance f_i en choisissant deux nombres entiers aléatoires u et $v \in [1, 10]$ et en définissant $r_i = \max\{1, s_i - u\}$ et $f_i = \min\{s_i + d_i + v, 100\}$. Finalement, nous spécifions pour chaque ressource l et chaque période j, une disponibilité égale à la demande en ressources de l'horaire s augmentée d'un nombre entier choisi aléatoirement dans l'intervalle $[0, 1]$ pour $l = 1$, $[0, 5]$ pour $l = 2$, $[0, 10]$ pour $l = 3$ et $[0, 15]$ pour $l = 4$. La structure des fichiers de données traités par les programmes est illustrée dans la figure 7.1 et la procédure de

génération des problèmes est résumée dans la figure 7.2.

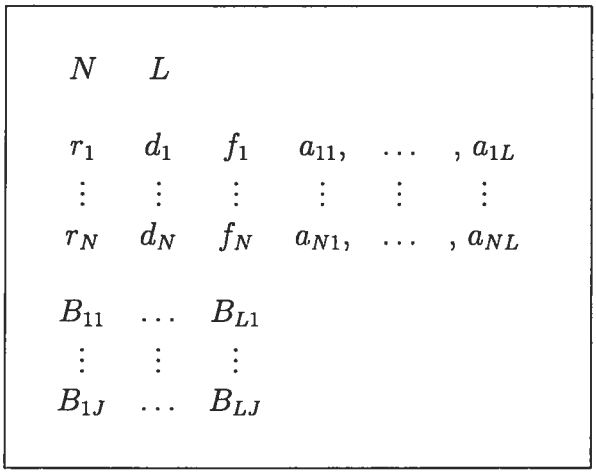

FIG. 7.1 – Structure du fichier de données

Trois ensembles distincts de problèmes ont été générés à l'aide de la technique décrite ci-dessus. Nous avons utilisé un premier ensemble de 50 problèmes pour mettre au point les divers paramètres nécessaires à l'exécution des méthodes. Un deuxième ensemble, composé également de 50 problèmes, a été utilisé pour identifier la meilleure stratégie d'amélioration pour chaque technique de résolution. Finalement, un dernier ensemble de 100 problèmes a servi à comparer entre elles les différentes heuristiques simples et couplées avec une technique d'amélioration. Les 200 problèmes réunis ont une valeur moyenne de $P(x^0)$, où x^0 est la solution initiale générée à l'aide de la méthode constructive (voir la section 5.1.2), égale à 421.59.

7.1.2 Environnement de programmation

Toutes les méthodes ont été implantées en utilisant le langage de programmation C. Les test ont été effectués sur un ordinateur doté d'un processeur *AMD Athlon 750MHz* et fonctionnant sous le système d'opération *Linux*.

Soient :

L := 4, le nombre de types de ressource
J := 100, le nombre de périodes
N := 100, le nombre de tâches

Demande en ressources et durée de chaque tâche
POUR CHAQUE i DE 1 À N

a_{i1} := 1
a_{i2} := nombre aléatoire entier dans l'intervalle $[0, 5]$
a_{i3} := nombre aléatoire entier dans l'intervalle $[0, 10]$
a_{i4} := nombre aléatoire entier dans l'intervalle $[0, 20]$
d_i := nombre aléatoire entier dans l'intervalle $[1, 20]$

Génération de l'horaire fictif s
$s_1 := 1$
POUR CHAQUE i DE 2 À N

$$s_i := \begin{cases} s_{i-1} + d_{i-1} & \text{, si } s_{i-1} + d_{i-1} + d_i - 1 \leq J \\ 1 & \text{, sinon} \end{cases}$$

Temps le plus tôt et échéance pour chaque tâche
POUR CHAQUE i DE 1 À N

u := nombre aléatoire entier dans l'intervalle $[1, 10]$

$$r_i := \begin{cases} s_i - u & \text{, si } s_i - u \geq 1 \\ 1 & \text{, sinon} \end{cases}$$

v := nombre aléatoire entier dans l'intervalle $[1, 10]$

$$f_i := \begin{cases} s_i + d_i + v & \text{, si } s_i + d_i + v \leq 100 \\ 100 & \text{, sinon} \end{cases}$$

Disponibilité en ressources pour chaque période
POUR CHAQUE j DE 1 À J

w := nombre aléatoire entier dans l'intervalle $[0, 1]$
$B_{1j} := \sum_{i \in E_j(s)} a_{i1} + w$
w := nombre aléatoire entier dans l'intervalle $[0, 5]$
$B_{2j} := \sum_{i \in E_j(s)} a_{i2} + w$
w := nombre aléatoire entier dans l'intervalle $[0, 10]$
$B_{3j} := \sum_{i \in E_j(s)} a_{i3} + w$
w := nombre aléatoire entier dans l'intervalle $[0, 20]$
$B_{4j} := \sum_{i \in E_j(s)} a_{i4} + w$

FIG. 7.2 – Génération des exemples de problèmes

7.2 Mise au point des paramètres

Tous les algorithmes étudiés comportent des paramètres dont les valeurs influencent grandement leur efficacité. Leur mise au point requiert donc beaucoup de doigté et de minutie. Nous consacrons beaucoup d'énergie à choisir les bonnes valeurs pour les paramètres afin d'accroître l'efficacité des méthodes. Deux critères sont utilisés pour mesurer l'efficacité d'une méthode pour un temps de résolution donné lorsque nous utilisons diverses valeurs des paramètres :

- le nombre de problèmes où la meilleure solution est obtenue avec ces valeurs des paramètres parmi toutes celles testées (critère relatif)
- la valeur moyenne des meilleures solutions générées pour l'ensemble des problèmes en utilisant ces valeurs des paramètres (critère absolu).

Les valeurs des paramètres $nbItMax$ et $nbItInutileMax$ sont déterminées de sorte que le temps moyen d'exécution soit de 30 secondes.

Pour déterminer les valeurs plus prometteuses à tester pour les paramètres, nous analysons un grand nombre de valeurs possibles un utilisant les 5 premiers problèmes. Pour cette étape, le temps d'exécution est limité à 30 secondes, sans égard aux valeurs de $nbItMax$ et $nbItInutileMax$, étant donné que le nombre d'itérations nécessaires afin d'atteindre 30 secondes d'exécution peut varier beaucoup avec la valeur du paramètre à fixer.

7.2.1 Méthode de recherche taboue

La méthode de recherche taboue est la plus difficile à mettre au point selon notre expérience étant donné que plusieurs parties de l'algorithme peuvent être modifiées. De plus, étant donné le nombre de paramètres à fixer et leur grande variabilité, tester toutes les combinaisons possibles de valeurs est alors une tâche gigantesque. Nous analysons donc les divers paramètres et les diverses options dans l'ordre suivant. Nous commençons par déterminer la longueur de la liste taboue $|LT|$ pour ensuite choisir le sous-voisinage $N_V(x)$ le plus approprié. Finalement, nous testons la pertinence d'utiliser une liste taboue de longueur variable et analysons en dernier lieu une version moins vorace de l'algorithme. Voici les détails de ces expériences numériques.

Longueur de la liste taboue

Nous estimons tout d'abord la meilleure longueur de la liste taboue $|LT|$ en utilisant $V = \{1, 2, \ldots, N\}$ et le parcours du voisinage basé sur l'épuisement d'une permutation (section 5.3.1). Dans un premier temps, nous considérons 10 longueurs pour la liste taboue, allant de 10 à 100 éléments. Les trois longueurs les plus efficaces, selon nos critères, sont 80, 90 et 100.

Nous poursuivons alors les tests avec ces trois longueurs. Pour obtenir des temps moyens d'exécution de 30 secondes, $nbItMax$ est fixé à 10000 tandis que $nbItInutileMax$ est fixé à 2500 pour $|LT|=80$, 2700 pour $|LT|=90$, et 3000 pour $|LT|=100$. Les résultats du tableau 7.1 indiquent que la liste taboue contenant 80 éléments permet d'obtenir le plus de meilleures solutions et des solutions d'une meilleure qualité en général que les deux autres longueurs.

$\|LT\|$	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
80	31.5	30.2	6501.5	26
90	32.8	30.8	6671.4	20
100	35.5	30.9	6619.3	11

TAB. 7.1 – Résultats comparatifs pour différentes longueurs de LT dans la méthode de recherche taboue

Définition du sous-voisinage $N_V(x)$

Par la suite, nous tentons de trouver la meilleure définition pour le sous-voisinage $N_V(x)$ en utilisant une liste taboue de 80 éléments. Deux composantes de $N_V(x)$ sont à définir : l'ordre dans lequel les tâches sont explorées (la sélection par épuisement d'une permutation aléatoire ou la sélection utilisant un critère basé sur les périodes) et le pourcentage de tâches à explorer par itération.

Dans un premier temps, nous considérons 9 valeurs différentes (20, 30, 40, 50, 60, 70, 80, 90, 100 (%)) pour le pourcentage de tâches à explorer à chaque itération pour chacun des deux types de parcours des tâches. Pour la suite des tests, nous conservons les deux meilleures valeurs pour chaque type de parcours. Les résultats indiquent que pour le parcours basé sur une permutation aléatoire,

les valeurs à conserver sont 30% et 60% tandis que pour le parcours utilisant un critère basé sur les périodes, elles sont plutôt 60% et 100%.

Pour tester ces quatre combinaisons, nous déterminons les valeurs de $nbItMax$ et $nbItInutileMax$ en fonction du type de parcours et du pourcentage de voisins explorés à chaque itération. Celles-ci sont différentes puisque :

- plus le pourcentage est bas, plus une itération est complétée rapidement,
- le parcours utilisant le critère par période demande plus de temps de calcul que celui basé sur une permutation.

Pour le parcours basé sur une permutation, nous fixons les critères d'arrêt $nbItMax$ et $nbItInutileMax$ respectivement à 17000 et 5500 pour l'exploration de 30% des tâches par itération et à 14000 et 3800 pour l'exploration de 60%. Pour le parcours utilisant un critère basé sur les périodes, nous fixons plutôt $nbItMax$ et $nbItInutileMax$ respectivement à 13000 et 3600 pour l'exploration de 60% des tâches par itération et à 13800 et 2400 pour l'exploration de 100%. Les résultats du tableau 7.2 montrent que les solutions de meilleure qualité sont, en général, obtenues avec le parcours basé sur une permutation avec exploration de seulement 30% des tâches au cours d'une itération.

Sélection des tâches dans V	Pourcentage de tâches dans V (%)	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
permutation aléatoire	30	29.7	30.2	12337.6	19
	60	32.0	30.2	8917.0	12
critère par période	60	31.4	30.1	8834.6	9
	100	30.6	29.7	6291.8	14

TAB. 7.2 – Résultats comparatifs pour différents sous-voisinages dans la méthode de recherche taboue

Liste taboue de longueur variable ou fixe

La troisième étape consiste à déterminer s'il y a un avantage à utiliser une liste taboue de longueur variable, en utilisant les valeurs précédemment identifiées pour les autres paramètres. Pour conserver un temps moyen d'exécution de 30 secondes, la valeur de $nbItInutileMax$ est augmentée à 5750 itérations lorsque la liste de longueur variable est utilisée. Le tableau 7.3 résume les résultats obtenus.

Longueur de la liste taboue	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
fixe	29.7	30.2	12337.6	23
variable	27.8	30.6	12546.9	31

TAB. 7.3 – Résultats comparatifs pour deux types de listes de tabous dans la méthode de recherche taboue

De façon générale, les résultats indiquent que l'utilisation d'une liste taboue de longueur variable permet d'obtenir de meilleures solutions. Cette stratégie est donc utilisée dans tous les tests subséquents.

Méthode moins vorace

Finalement, nous comparons une version moins vorace de cet algorithme à la version originale, en utilisant les valeurs précédentes des paramètres. Dans cette variante, le sous-voisinage est entièrement exploré à chaque itération afin d'identifier la meilleure solution dans celui-ci au lieu de procéder comme dans la variante originale où une itération s'arrête dès qu'une solution meilleure que la meilleure solution rencontrée jusqu'ici est trouvée. Cette variante moins vorace est illustrée dans la figure 7.3.

Pour obtenir un temps moyen d'exécution d'environ 30 secondes pour la version moins vorace, nous limitons la valeur de $nbItInutileMax$ à seulement 4500. Les résultats résumés au tableau 7.4 indiquent que la variante originale génère des solutions de plus basse violation en général. En conséquence, nous utilisons seulement la variante originale pour la suite des expérimentations.

Variante	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
originale	27.8	30.6	12546.9	32
moins vorace	29.4	30.1	11580.8	21

TAB. 7.4 – Résultats comparatifs pour deux variantes de la méthode de recherche taboue

Initialisation :

Soient :

x^0, un horaire admissible initial
$x := x^0$, la solution courante
$x^* := x^0$, la meilleure solution rencontrée jusqu'ici
$nbIt := 0$, le nombre d'itérations
$nbItInutile := 0$, le nombre d'itér. successives sans améliorer x^*
$fin :=$ FAUX

Algorithme :

TANT QUE $fin \neq$ VRAI
$Z := \varnothing$
Déterminer le sous-ensemble de tâches $V \subseteq \{1, 2, \ldots, N\}$

Exploration des voisins
POUR CHAQUE $i \in V$
SI $P(x^*) = 0$, ALORS $fin :=$ VRAI

Voisin de gauche par rapport à la tâche i
SI $x(i, g)$ est admissible (c-à-d. $x_i - 1 \geq r_i$), ALORS
Critère d'aspiration
SI $P(x(i, g)) < P(x^*)$, ALORS
$x^* := x(i, g)$
$nbItInutile := 0$
$Z := Z \cup \{x(i, g)\}$

SI $(i, x_i - 1) \notin LT$, ALORS
$Z := Z \cup \{x(i, g)\}$

Voisin de droite par rapport à la tâche i
SI $x(i, d)$ est admissible (c-à-d. $x_i + 1 \leq f_i - d_i + 1$), ALORS
Critère d'aspiration
SI $P(x(i, d)) < P(x^*)$, ALORS
$x^* := x(i, d)$
$nbItInutile := 0$
$Z := Z \cup \{x(i, d)\}$

SI $(i, x_i + 1) \notin LT$, ALORS
$Z := Z \cup \{x(i, d)\}$

Remplacement de x
$x := \underset{z \in Z}{\text{argmin}} \{P(z)\}$

Mettre à jour LT

Évaluation des critères d'arrêt
$nbIt := nbIt + 1$
$nbItInutile := nbItInutile + 1$

SI $nbIt \geq nbItMax$ OU $nbItInutile \geq nbItInutileMax$, ALORS
$fin :=$ VRAI

x^* est l'horaire admissible généré

FIG. 7.3 – Variante moins vorace de la méthode de recherche taboue

7.2.2 Méthode du recuit simulé

La méthode du recuit simulé requiert la mise au point de deux paramètres. Ces deux paramètres, la température initiale T^0 et le facteur de refroidissement R, déterminent la vitesse à laquelle l'équilibre est atteint. Comme ces deux paramètres sont très étroitement liés, nous les testons simultanément.

Dans un premier temps, nous considérons les combinaisons générées avec cinq valeurs pour la température initiale (10, 20, 30, 40 et 50) et pour le facteur de refroidissement (0.98, 0.99, 0.995, 0.999, 0.9995). Parmi ces 25 combinaisons, les quatres suivantes engendrent de meilleurs résultats : $T^0 = 10$ avec $R = 0.9995$, $T^0 = 20$ avec $R = 0.999$, $T^0 = 30$ avec $R = 0.999$, et $T^0 = 30$ avec $R = 0.999$.

Par contre, les tests subséquents nous permettent d'observer que l'utilisation de $T^0 = 10$ avec $R = 0.9995$ ne permet pas d'obtenir l'équilibre en 30 secondes. Les résultats sont alors encourageants en général si ce n'est de la qualité très variable des solutions finales générées. Pour cette température initiale, une valeur pour le facteur de refroidissement égale à 0.99925 est donc plutôt employée pour la suite. Pour obtenir un temps moyen d'exécution de 30 secondes, $nbItMax$ est fixé à 10000 pour toutes les combinaisons et $nbItInutileMax$ est fixé à 150 pour la première combinaison, 1000 pour la deuxième, 650 pour la troisième et 300 pour la dernière. Les résultats présentés au tableau 7.5 indiquent que le changement de facteur de refroidissement est très profitable pour la première combinaison puisqu'elle devient celle pour laquelle le plus grand nombre de meilleures solutions sont générées, dépassant de loin sa plus proche rivale ($T^0 = 30$ avec $R = 0.999$). Il faut

cependant noter que la qualité moyenne des solutions est légèrement inférieure avec cette dernière. Pour les prochaines expérimentations, la combinaison $T^0 = 10$ avec $R = 0.99925$ est préférée aux autres.

T^0	R	Valeur moyenne de $P(X^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
10	0.99925	10.7	30.0	5876.9	22
20	0.999	12.0	30.7	6071.4	10
30	0.999	10.2	30.4	6116.7	14
40	0.999	11.8	29.7	6065.0	8

TAB. 7.5 – Résultats comparatifs pour différentes valeurs de T^0 et R dans la méthode du recuit simulé

7.2.3 Méthode d'acceptation avec seuil standard

La méthode d'acceptation avec seuil standard comporte deux paramètres principaux : le seuil initial dr^0 et le facteur de diminution du seuil $a \in (0, 1)$. Dans un premier temps, des tests sont faits pour des valeurs de dr^0 égales à 10, 20, 30, 40 et 50. Pour chaque valeur de dr^0, nous testons 5 valeurs de a : 0.997, 0.998, 0.999, 0.9992 et 0.9995. Les résultats sont meilleurs avec les deux combinaisons suivantes : $dr^0 = 30$ avec $a = 0.9995$ et $dr^0 = 20$ avec $a = 0.9992$. De plus, ces expériences préliminaires permettent d'estimer que le nombre d'itérations pouvant être complétées en 30 secondes est au moins égal à 7000 avec ces valeurs de paramètres.

Nous choisissons de poursuivre les tests avec trois valeurs de dr^0 : 20, 25 et 30. Pour chacune de ces valeurs, nous déterminons deux valeurs de a de façon à ce que le seuil dr, après respectivement 6800 et 7200 itérations, soit inférieur à un. En effet, à partir du moment où le seuil devient inférieur à un, plus aucune solution détériorante n'est acceptée et le processus stagne habituellement assez rapidement. Nous avons observé ce phénomène avec la combinaison $dr^0 = 20$ et $a = 0.9992$ qui peut obtenir de très bons résulats (comme cela est le cas avec les 5 premiers problèmes) mais peut aussi en obtenir de très mauvais : raison pour laquelle nous avons plutôt opté pour la règle précédente pour déterminer de bonnes valeurs de a pour un seuil initial donné. Ainsi, nous poursuivons les tests avec des valeurs de a égales à 0.99955 et 0.99957 pour $dr^0 = 20$, 0.99952 et 0.99955

pour $dr^0 = 25$, et 0.9995 et 0.99952 pour $dr^0 = 30$. Pour ces tests, $nbItMax$ est fixé à 9000 pour toutes les combinaisons et $nbItInutileMax$ est fixé à 500 et 300 pour $dr^0 = 20$, à 550 et 250 pour $dr^0 = 25$ et à 550 et 400 pour $dr^0 = 30$ et ce pour les deux valeurs respectives de a associées à chaque seuil initial. Les meilleurs résultats sont obtenus avec un seuil initial égal à 25 et un facteur a égal à 0.99952. Un sommaire des résultats obtenus est présenté au tableau 7.6.

dr^0	a	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
20	0.99955	21.7	30.0	7215.3	9
	0.99957	21.4	30.0	7303.8	10
25	0.99952	20.8	29.9	7320.1	14
	0.99955	22.9	29.9	7425.9	6
30	0.99950	22.8	29.9	7394.9	8
	0.99952	21.2	30.1	7514.7	9

TAB. 7.6 – Résultats comparatifs pour différentes valeurs de dr^0 et a dans la méthode d'acceptation avec seuil standard

7.2.4 Méthode d'acceptation avec seuil - grand déluge

La variante grand déluge de la méthode d'acceptation avec seuil a l'avantage de n'utiliser qu'un seul paramètre puisque le seuil initial dr^0 est fixé à la valeur de la solution initiale. Ainsi, le seul paramètre à déterminer est δ qui représente la vitesse à laquelle le seuil diminue. Rappelons que le seuil est une borne supérieure sur la valeur des solutions pouvant remplacer la solution courante au cours d'une itération.

Dans [19], l'instigateur de cette méthode propose une règle pour estimer δ. D'après ses expériences, l'utilisation d'une valeur de δ inférieure à 1% de la différence moyenne entre la valeur de la solution courante et le seuil dr est la meilleure approche. Pour notre part, nous utilisons un autre raisonnement afin d'avoir une idée de la valeur à utiliser. Nous calculons la diminution moyenne de $P(x)$ par itération qui serait nécessaire pour atteindre une solution finale réalisable en divisant la moyenne de la valeur des solutions initiales par le nombre d'itérations pouvant être complétées en 30 secondes et estimé à 6000. Cette mécanique nous conduit à une valeur de δ égale à 0.07. Dans un premier temps,

nous expérimentons donc avec 5 problèmes les valeurs suivantes pour δ : 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10 et 0.125. Les résultats les plus encourageants, en 30 secondes d'exécution, sont obtenus pour des valeurs de δ égales à 0.07, 0.08 et 0.09.

Nous testons et comparons alors les résultats obtenus pour des valeurs de δ égales à 0.07, 0.075, 0.08, 0.085 et 0.09. Dans ces tests, le paramètre $nbItMax$ est fixé à 10000 pour toutes les valeurs de δ et le paramètre $nbItInutileMax$ est fixé à 1300, 1500, 1775, 1975 et 2175 pour des valeurs respectives de δ égales à 0.07, 0.075, 0.08, 0,085 et 0.09. Les résultats du tableau 7.7 indiquent que les meilleurs résultats, tant au niveau du nombre de meilleures solutions générées que de la valeur moyenne des solutions, sont obtenus avec une valeur de δ égale à 0.085.

δ	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
0.070	19.4	30.0	6878.6	12
0.075	19.6	30.0	6709.6	9
0.080	19.9	30.0	6661.9	10
0.085	19.3	30.0	6580.5	14
0.090	21.3	30.1	6500.7	10

TAB. 7.7 – Résultats comparatifs pour différentes valeurs de δ dans la méthode d'acceptation avec seuil - grand déluge

7.2.5 Méthode d'acceptation avec seuil à détérioration maximale

Cette variante de la méthode d'acceptation avec seuil nécessite la mise au point d'un seul paramètre tout comme la variante grand déluge. Le paramètre μ à estimer borne supérieurement la différence maximale entre la qualité de la prochaine solution courante et la qualité de la meilleure solution générée jusqu'ici. N'ayant aucune information sur la manière d'estimer a priori la valeur de μ, nous explorons largement les valeurs possibles en testant les 10 valeurs suivantes : 1, 2, 5, 7, 10, 12, 15, 17, 20 et 30. Les valeurs les plus efficaces lorsque la méthode est appliquée pour une durée de 30 secondes sur les cinq premiers problèmes sont 7 et 10.

À la lumière de ces premiers tests, nous poursuivons avec les quatre valeurs suivantes pour μ : 7, 8, 9 et 10. Pour obtenir un temps moyen d'exécution de 30 secondes, les paramètres $nbItMax$ et $nbItInutileMax$ sont fixés à 6000 et 200, respectivement. Les résultats du tableau 7.8 indiquent que pour les 50 problèmes testés, ceux obtenus avec une valeur de μ égale à 7 sont les meilleurs.

μ	Valeur moyenne de $P(x^*)$	Temps d'exécution moyen (sec)	Nombre moyen d'itérations	Nombre de meilleures solutions
7	22.3	30.4	5938.4	21
8	23.0	29.7	5872.6	18
9	22.6	30.0	6000.0	14
10	26.0	29.7	6000.0	2

TAB. 7.8 – Résultats comparatifs pour différentes valeurs de μ dans la méthode d'acceptation avec seuil à détérioration maximale

7.3 Expérimentations avec différentes stratégies d'amélioration

Dans la deuxième partie des expérimentations numériques, nous nous appliquons à déterminer, pour chaque méthode testée, la meilleure stratégie d'amélioration parmi les quatres techniques étudiées : la diversification par construction aléatoire (DCA), la diversification de premier ordre (DPO), la recherche à voisinage variable avec échanges (RVVE) et la recherche à voisinage variable imbriqué (RVVI).

Pour chaque technique de recherche locale, nous comparons l'efficacité des quatre stratégies d'amélioration en limitant le temps total d'exécution à 30 secondes. Le nombre de fois où la technique de recherche locale peut être réinitialisée à partir de différentes solutions initiales dépend alors seulement du temps d'exécution alloué à celle-ci. Pour ce faire, notre implantation de la recherche à voisinage variable est légèrement modifiée pour tenir compte seulement du temps d'exécution maximal alloué comme critère d'arrêt plutôt que de fixer un nombre maximal d'itérations et un nombre maximal d'itérations sans amélioration de la meilleure solution (voir section 6.3.5).

De plus, pour la méthode de recherche taboue, du recuit simulé et les trois techniques d'acceptation avec seuil, nous testons deux niveaux d'utilisation, moyen et intense, pour chacune des quatre stratégies d'amélioration testées. Le niveau moyen consiste à limiter le temps d'exécution de chaque application de la technique de recherche dans le voisinage (TRV) concernée à environ six secondes. Ainsi, en moyenne cinq applications d'une TRV, à partir de cinq solutions initiales différentes, seront effectuées par exécution. Le niveau intense consiste à ne laisser que trois secondes pour l'exécution de la recherche locale, afin d'effectuer un plus grand nombre d'applications, soit une dizaine en moyenne, mais de durées plus courtes.

Étant donné que les valeurs des paramètres sont déterminées préalablement dans la section 7.2 en fonction d'une durée d'exécution moyenne de trente secondes, nous tentons d'abord de quantifier l'impact d'une diminution de la durée d'exécution sur la valeur optimale des paramètres. Nous suivons donc approximativement la même démarche et employons les mêmes problèmes qu'à la section 7.2 pour déterminer les meilleures valeurs des paramètres en limitant respectivement le temps moyen d'exécution à trois et six secondes. Pour la méthode de recherche taboue, nous conservons par contre la même définition du voisinage, l'utilisation de la liste taboue de longueur variable et la variante originale de l'algorithme. En conséquence, nous cherchons seulement la meilleure longueur maximale de la liste taboue à employer pour chaque temps moyen d'exécution. Toutes les valeurs des paramètres des autres techniques sont remis en question.

Les prochaines sections contiennent, pour chaque technique, les résultats des expérimentations faites avec les stratégies d'amélioration. Les valeurs des paramètres employées pour chaque temps moyen d'exécution sont de plus présentées sous forme de tableau dans chacune des sections.

7.3.1 Méthode de descente pure

Étant donné que la méthode de descente pure s'arrête au premier minimum local rencontré, il est évident qu'elle ne peut qu'être avantagée par l'emploi d'une des stratégies étudiées qui lui offrent toutes une façon de s'échapper de ce minimum local pour continuer l'exploration de nouvelles solutions admissibles. En

effet, la diversification par construction aléatoire et la diversification de premier ordre sont deux mécanismes distincts pour obtenir de nouvelles solutions initiales à partir desquelles la méthode de descente pure peut générer un nouveau minimum local. Quant à la recherche à voisinage variable, le changement de structure de voisinage lors de l'atteinte d'un minimum local permet de continuer l'exploration puisqu'un minimum local par rapport à une structure de voisinage n'est pas nécessairement un minimum local par rapport à une autre structure.

Avant d'effectuer les tests comparatifs, nous cherchons d'abord à évaluer la meilleure valeur de λ (le pourcentage de tâches dont la planification doit être changée) à employer pour la diversification de premier ordre. D'abord, nous testons les valeurs comprises entre 30% et 100%, avec un intervalle de 10% entre chaque valeur testée, en utilisant le premier ensemble de 50 problèmes. Comme les meilleurs résultats sont obtenus avec des valeurs de λ égales à 70%, 80% et 90%, les valeurs de λ égales à 75% et 85% sont aussi considérées. Les résultats pour ces cinq valeurs de λ présentés au tableau 7.9 indiquent que les solutions ayant une meilleure valeur en moyenne sont générées en changeant la cédule de 75% des tâches. De plus, en ne considérant que les valeurs de λ égales à 75% et 90%, nous constatons que le nombre de problèmes pour lesquels la solution générée lorsqu'une valeur de λ égale à 75% est utilisée est au moins aussi bonne que celle générée lorsqu'une valeur de λ égale à 90% est utilisée est égal à 28, contre 25 pour l'inverse. Par conséquent, même si les résultats du tableau 7.9 concernant le nombre de meilleures solutions semblent indiquer que l'utilisation d'un λ égal à 90% est plus efficace, il en est autrement. Tous les tests subséquents avec la diversification de premier ordre sont donc faits avec une valeur de λ égale 75%.

λ (%)	Valeur moyenne de $P(x^*)$	Nombre de diversifications	Nombre de meilleures solutions
70	35.1	89	13
75	31.6	84	13
80	34.3	79	8
85	33.3	74	10
90	32.5	70	16

TAB. 7.9 – Résultats comparatifs pour différentes valeurs de λ pour la diversification de premier ordre appliquée à la méthode de descente pure

À la lumière des résultats obtenus dans les tests comparatifs présentés au tableau 7.10, nous constatons que la diversification de premier ordre et la recherche à voisinage variable imbriqué sont de loin les plus susceptibles d'améliorer l'efficacité de la méthode de descente pure. En effet, ces stratégies génèrent, parmi les problèmes testés, toutes les meilleures solutions atteintes avec l'une ou l'autre des stratégies d'amélioration. De plus, les résultats indiquent que la recherche à voisinage variable imbriqué est plus efficace que la DPO. Nous conservons donc la méthode hybride composée de la méthode de descente pure et de la recherche à voisinage variable imbriqué pour les tests finaux. Il faut noter que la colonne *Nombre moyen de diversifications* du tableau 7.10 indique le nombre moyen de fois où la TRV est réinitialisée avec une nouvelle solution initiale dans le cadre de la diversification par construction aléatoire et celle de premier ordre. Avec les deux variantes de la recherche à voisinage variable, ce nombre indique plutôt le nombre moyen de changements de la structure de voisinage.

Stratégie d'amélioration	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Temps d'exécution moyen (sec)	Nombre de meilleures solutions
DCA	144.1	124.6	30.0	0
DPO	33.9	85.2	30.0	21
RVVE	94.8	1839.0	30.0	0
RVVI	30.2	1532.9	30.0	30

TAB. 7.10 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées à la méthode de descente pure

7.3.2 Méthode de recherche taboue

Les tests effectués en limitant à 3 et 6 secondes le temps d'exécution indique qu'il est nécessaire de diminuer la longueur de la liste taboue pour obtenir de meilleurs résultats. Cette constatation peut s'expliquer par le fait qu'avec une longue liste taboue, une solution reste interdite trop longtemps par rapport au nombre d'itérations total. En effet, nous obtenons de meilleurs résultats avec une liste taboue contenant entre 60 et 70 éléments au maximum au lieu d'entre 80 et 100 comme nous obtenons pour un temps d'exécution de 30 secondes. En particulier, nous obtenons les meilleurs résultats avec une liste contenant au maximum 70 mouvements tabous (voir le tableau 7.11 pour les valeurs des paramètres en fonction du temps d'exécution moyen).

Temps d'exécution moyen (sec)	$\|TL\|$	$NbItMax$	$NbItInutileMax$
3	70	1500	450
6	70	3000	1000
30	80	17000	5750

TAB. 7.11 – Valeurs des paramètres utilisés pour la méthode de recherche taboue en fonction du temps moyen d'exécution alloué

Les résultats résumés au tableau 7.12 révèlent clairement que la diversification de premier ordre à un niveau d'utilisation moyen est la meilleure stratégie d'amélioration pour la recherche taboue. En effet, cette stratégie génère la meilleure solution pour le plus grand nombre de problèmes et les solutions obtenues avec cette méthode ont la plus basse violation finale en moyenne. Nous poursuivons donc les tests avec le niveau d'utilisation moyen de la diversification de premier ordre pour la méthode de recherche taboue.

Stratégie d'amélioration	Niveau de diversification	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Nombre de meilleures solutions
DCA	moyen	34.6	4.4	7
	intense	42.7	10.0	3
DPO	moyen	27.6	7.1	23
	intense	31.1	15.5	8
RVVE	moyen	35.0	3.1	7
	intense	36.0	9.1	4
RVVI	moyen	34.5	3.1	8
	intense	34.2	8.4	5

TAB. 7.12 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées à la méthode de recherche taboue

7.3.3 Méthode du recuit simulé

Les résultats présentés au tableau 7.13 concernant les meilleures valeurs des paramètres pour différents temps d'exécution indiquent que plus le temps alloué à l'exécution de la méthode est court, moins la température initiale doit être élevée et plus le taux de refroidissement doit être élevé afin d'atteindre l'équilibre thermodynamique avant la fin de l'exécution. Notons que si cet équilibre n'est

pas atteint, la valeur des solutions finales générées est très variable.

Temps d'exécution moyen (sec)	T^0	R	$NbItMax$	$NbItInutileMax$
3	5	0.994	600	20
6	7	0.997	1200	15
30	10	0.99925	10000	150

TAB. 7.13 – Valeurs des paramètres utilisés pour le recuit simulé en fonction du temps moyen d'exécution alloué

Les résultats numériques des tests comparatifs sont résumés au tableau 7.14. Ils indiquent qu'avec le recuit simulé, la recherche à voisinage variable imbriqué est la stratégie d'amélioration la plus efficace et ce à n'importe quel niveau d'utilisation. La violation finale moyenne des solutions générées de même que le nombre de problèmes pour lesquels cette méthode engendre la meilleure solution parmi celles générées par toutes les méthodes comparées démarquent nettement cette stratégie par rapport aux trois autres et ce aux deux niveaux d'utilisation (moyen et intense). Pour finir, l'utilisation à un niveau intense de la recherche à voisinage variable imbriqué semble plus efficace qu'à un niveau moyen d'après les résultats obtenus. En effet, la recherche à voisinage variable imbriqué à un niveau intense obtient une plus basse valeur moyenne de $P(x^*)$ et génère la meilleure solution pour un plus grand nombre de problèmes. Cette stratégie est donc conservée pour les tests suivants.

Stratégie d'amélioration	Niveau de diversification	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Nombre de meilleures solutions
DCA	moyen	10.9	4.6	2
	intense	12.3	9.8	0
DPO	moyen	10.9	4.0	4
	intense	12.7	8.9	4
RVVE	moyen	15.4	2.0	0
	intense	10.6	3.3	7
RVVI	moyen	7.3	3.2	18
	intense	6.5	6.1	29

TAB. 7.14 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées au recuit simulé

7.3.4 Méthode d'acceptation avec seuil standard

De façon similaire à la méthode du recuit simulé, dont celle-ci est fortement inspirée, l'ajustement des paramètres pour des temps moyens d'exécution inférieurs indique qu'il est profitable de diminuer le seuil initial dr^0 et d'accélérer la descente du seuil de façon à ce que celui-ci atteigne la valeur zéro peu avant la fin de l'exécution. À ce moment, seules les modifications améliorantes, c'est-à-dire pour lesquelles $P(x') < P(x)$, sont acceptées. Cette règle permet d'estimer a priori la plage de valeurs la plus susceptible d'engendrer de bons résultats. Les valeurs exactes des différents paramètres employées pour chaque temps moyen d'exécution sont présentées au tableau 7.15.

Temps d'exécution moyen (sec)	dr^0	a	$NbItMax$	$NbItInutileMax$
3	10	0.9966	750	15
6	10	0.9983	1450	20
30	25	0.99952	9000	550

TAB. 7.15 – Valeurs des paramètres utilisés pour la méthode d'acceptation avec seuil standard en fonction du temps moyen d'exécution alloué

Les résultats obtenus pour les différentes stratégies présentés au tableau 7.16 indiquent que la diversification de premier ordre à n'importe quel niveau d'utilisation est la meilleure stratégie d'amélioration pour la méthode d'acceptation avec seuil standard. En effet, les deux niveaux de cette stratégie obtiennent les meilleures solutions pour le plus grand nombre de problèmes (14 problèmes pour chacun) et les solutions générées pour tous les problèmes ont les deux plus basses valeurs moyennes de P (18.3 pour le niveau moyen et 18.8 pour le niveau intense). De plus, les résultats indiquent que l'utilisation de la DPO à un niveau moyen génère des solutions dont la violation est la plus basse en moyenne. Par conséquent, cette stratégie est conservée pour les tests finaux concernant la méthode d'acceptation avec seuil standard.

Stratégie d'amélioration	Niveau de diversification	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Nombre de meilleures solutions
DCA	moyen	19.7	4.1	11
	intense	19.5	9.0	6
DPO	moyen	18.3	4.0	14
	intense	18.8	8.8	14
RVVE	moyen	21.7	2.0	7
	intense	19.0	4.0	10
RVVI	moyen	21.7	2.0	7
	intense	19.0	3.9	10

TAB. 7.16 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées à la méthode d'acceptation avec seuil standard

7.3.5 Méthode d'acceptation avec seuil - grand déluge

Encore une fois, les tests effectués en diminuant le temps d'exécution révèlent qu'il est nécessaire, dans la méthode d'acceptation avec seuil - grand déluge, d'accélérer la diminution du seuil en augmentant la valeur du paramètre δ. Le tableau 7.17 présente, pour chaque temps d'exécution, les valeurs de δ permettant d'après les tests d'obtenir les meilleurs résultats.

Temps d'exécution moyen (sec)	δ	$NbItMax$	$NbItInutileMax$
3	0.80	725	200
6	0.40	1500	400
30	0.085	10000	1975

TAB. 7.17 – Valeurs des paramètres utilisés pour la méthode d'acceptation avec seuil - grand déluge en fonction du temps moyen d'exécution alloué

La première constatation que nous pouvons faire à partir des résultats présentés au tableau 7.18 concernant les différentes stratégies d'amélioration et les différents niveaux de diversification est que la recherche à voisinage variable imbriqué semble plus efficace que les trois autres stratégies. En effet, son utilisation à un niveau ou l'autre engendre la solution ayant la plus basse valeur de P pour 38 des 50 problèmes et les deux niveaux d'utilisation permettent d'obtenir des solutions ayant la plus basse violation en moyenne, soit 21.5. En outre, cette stratégie employée à un niveau moyen génère la meilleure solution pour 6 problèmes de

plus que lorsqu'elle est employée à un niveau intense. Nous décidons donc de conserver pour la suite des tests l'utilisation de la recherche à voisinage variable imbriqué à un niveau d'utilisation moyen avec la méthode du grand déluge.

Stratégie d'amélioration	Niveau de diversification	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Nombre de meilleures solutions
DCA	moyen	29.2	11.0	1
	intense	40.3	23.6	0
DPO	moyen	27.0	9.5	6
	intense	35.6	18.7	3
RVVE	moyen	23.8	6.7	10
	intense	25.2	13.5	6
RVVI	moyen	21.5	7.1	23
	intense	21.5	14.0	17

TAB. 7.18 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées à la méthode d'acceptation avec seuil - grand déluge

7.3.6 Méthode d'acceptation avec seuil à détérioration maximale

Les tests effectués en utilisant des temps moyens d'exécution de 3 et 6 secondes indiquent que le temps alloué à l'exécution de cette méthode n'a pas beaucoup d'influence sur la meilleure valeur de μ à employer. Seules les valeurs de $NbItMax$ et $NbItInutileMax$ sont donc ajustées afin d'obtenir le temps d'exécution moyen voulu tel qu'indiqué au tableau 7.19.

Temps d'exécution moyen (sec)	μ	$NbItMax$	$NbItInutileMax$
3	7	600	500
6	7	1200	1000
30	7	6000	200

TAB. 7.19 – Valeurs des paramètres utilisés pour la méthode d'acceptation avec seuil à détérioration maximale en fonction du temps moyen d'exécution alloué

Tant au niveau d'utilisation moyen qu'intense, les résultats obtenus avec la diversification de premier ordre surclassent ceux obtenus avec toutes les autres

stratégies d'amélioration. En effet, la méthode d'acceptation avec seuil à détérioration maximale avec la diversification de premier ordre au niveau moyen ou intense engendre la meilleure solution pour 29 des 50 problèmes en plus d'engendrer des solutions ayant en moyenne une plus basse violation des contraintes sur les ressources. En outre, les résultats indiquent que l'utilisation de la diversification de premier ordre à un niveau intense est plus efficace que son utilisation à un niveau moyen. Un sommaire des résultats est présenté au tableau 7.20.

Stratégie d'amélioration	Niveau de diversification	Valeur moyenne de $P(x^*)$	Nombre moyen de diversifications	Nombre de meilleures solutions
DCA	moyen	22.8	4.1	5
	intense	25.1	9.1	5
DPO	moyen	20.3	4.0	13
	intense	20.2	9.0	17
RVVE	moyen	22.8	3.8	7
	intense	23.5	7.0	8
RVVI	moyen	22.8	3.8	7
	intense	21.8	7.3	13

TAB. 7.20 – Résultats comparatifs pour différentes stratégies d'amélioration appliquées à la méthode d'acceptation avec seuil à détérioration maximale

7.4 Expérimentations finales

Le but des prochaines expérimentations est de déterminer la meilleure méthode pour résoudre le problème d'ordonnancement avec contraintes sur les ressources à l'étude. Pour ce faire, nous utilisons un dernier ensemble de 100 problèmes. Nous nous intéressons d'abord aux résultats globaux obtenus (valeur moyenne de $P(x^*)$ et nombre de meilleures solutions) pour les méthodes et leur combinaison avec une stratégie d'amélioration, puis nous complétons une analyse comparative de celles-ci deux à deux.

En ce qui concerne la comparaison globale, le tableau 7.21 contient un résumé des résultats obtenus pour chaque technique de recherche dans le voisinage employée seule et couplée avec la stratégie d'amélioration la plus performante pour celle-ci. Nous avons volontairement exclu la méthode de descente pure de ce résumé étant donné la trop piètre qualité des solutions qu'elle génère. Les résultats

pour chaque méthode concernent la valeur moyenne de $P(x^*)$, où x^* est la solution générée par cette méthode pour un problème, le nombre de problèmes pour lesquels la meilleure solution parmi celles engendrées par l'une ou l'autre des méthodes testées est générée par cette méthode et le temps moyen d'exécution utilisé par cette méthode. Les résultats détaillés obtenus par chaque méthode pour chaque problème sont présentés aux annexes A à F.

Méthode	Valeur moyenne de $P(x^*)$	Nombre de meilleures solutions	Temps d'exécution moyen (sec)
DP + RVVI	34.0	0	30.0
RT	26.5	0	30.9
RT + DPO	26.1	0	30.7
RS	9.6	36	29.8
RS + RVVI	7.0	69	29.7
ASS	21.7	0	29.8
ASS + DPO	20.1	1	31.0
ASGD	20.3	1	30.3
ASGD + RVVI	20.2	1	30.8
ASDM	21.3	0	30.1
ASDM + DPO	20.4	1	30.9

TAB. 7.21 – Résultats comparatifs pour les différentes méthodes avec un temps d'exécution moyen de 30 secondes

Les résultats du tableau 7.21 indiquent que la méthode du recuit simulé, qu'elle soit employée seule ou avec une recherche à voisinage variable, domine largement les autres méthodes. En ordre, les six méthodes obtenant le plus de succès sont le recuit simulé avec la recherche à voisinage variable imbriqué (RS+RVVI), le recuit simulé (RS), la méthode d'acceptation avec seuil standard avec la diversification de premier ordre (ASS+DPO), la méthode d'acceptation avec seuil de type grand déluge avec la recherche à voisinage variable imbriqué (ASGD+RVVI), la méthode d'acceptation avec seuil de type grand déluge (ASGD) et la méthode d'acceptation avec seuil à détérioration maximale avec la diversification de premier ordre (ASDM+DPO).

Pour comparer les méthodes deux à deux, nous utilisons la matrice du tableau 7.22 dans laquelle l'entrée (i, j) représente le nombre de problèmes pour lesquels la méthode i génère une solution au moins aussi bonne en terme de sa valeur que la méthode j.

	DP + RVVI	RT	RT + DPO	RS	RS + RVVI	ASS	ASS + DPO	ASGD	ASGD + RVVI	ASDM	ASDM + DPO
DP + RVVI	100	29	25	3	0	15	8	13	15	12	10
RT	73	100	43	2	1	30	30	27	30	27	25
RT + DPO	77	60	100	4	0	37	33	37	26	26	27
RS	98	98	97	100	37	95	91	93	92	96	93
RS + RVVI	100	100	100	71	100	98	99	97	96	99	99
ASS	87	75	69	9	2	100	47	46	45	47	44
ASS + DPO	93	73	73	11	2	63	100	54	52	58	48
ASGD	87	77	70	8	4	60	54	100	49	56	54
ASGD + RVVI	86	75	79	10	6	60	53	56	100	62	60
ASDM	89	80	76	5	2	59	45	48	40	100	51
ASDM + DPO	93	78	78	8	1	60	59	53	44	59	100

TAB. 7.22 – Comparaison par paire des différentes méthodes en fonction du nombre de meilleures solutions générées

À partir de cette matrice, nous obtenons la matrice du tableau 7.23 où l'entrée (i, j) représente le nombre net de problèmes pour lesquels la méthode i génère une meilleure (si le nombre est positif) ou une pire (si le nombre est négatif) solution que la méthode j. En effet, l'entrée (i, j) de cette matrice antisymétrique est obtenue en soustrayant l'entrée (j, i) de l'entrée (i, j) de la matrice du tableau 7.22.

Le tableau 7.23 vient corroborer les conclusions tirées à partir du tableau 7.21 sur les six meilleures méthodes à une exception près. En effet, la méthode d'acceptation avec seuil standard avec la diversification de premier ordre (ASS+DPO) obtient des solutions dont la valeur moyenne est inférieure à celle des solutions générées par les combinaisons ASGD+RVVI et ASDM+DPO (donc la première semble meilleure selon ce critère), mais ces deux dernières méthodes réussissent à générer une solution au moins aussi bonne que la première méthode pour au moins un problème de plus que celle-ci (donc la première semble moins bonne selon ce critère). Surtout, ce tableau met en relief la grande efficacité de la combinaison recuit simulé/recherche à voisinage variable imbriqué qui obtient une meilleure solution que toute autre méthode pour au moins 34 problèmes de plus. Outre les six méthodes les plus efficaces déjà établies, les cinq autres combinaisons les plus prometteuses sont, dans l'ordre, la méthode d'acceptation avec seuil

	DP + RVVI	RT	RT + DPO	RS	RS + RVVI	ASS	ASS + DPO	ASGD	ASGD + RVVI	ASDM	ASDM + DPO
DP + RVVI	0	-44	-52	-95	-100	-72	-85	-74	-71	-77	-83
RT	44	0	-17	-96	-99	-45	-43	-50	-45	-53	-53
RT + DPO	52	17	0	-93	-100	-32	-40	-33	-53	-50	-51
RS	95	96	93	0	-34	86	80	85	82	91	85
RS + RVVI	100	99	100	34	0	96	97	93	90	97	98
ASS	72	45	32	-86	-96	0	-16	-14	-15	-12	-16
ASS + DPO	85	43	40	-80	-97	16	0	0	-1	13	-11
ASGD	74	50	33	-85	-93	14	0	0	-7	8	1
ASGD + RVVI	71	45	53	-82	-90	15	1	7	0	22	16
ASDM	77	53	50	-91	-97	12	-13	-8	-22	0	-8
ASDM + DPO	83	53	51	-85	-98	16	11	-1	-16	8	0

TAB. 7.23 – Comparaison par paire des différentes méthodes en fonction du nombre net de meilleures solutions générées

à détérioration maximale (ASDM), la méthode d'acceptation avec seuil standard (ASS), la méthode de recherche taboue avec la diversification de premier ordre (RT+DPO), la méthode de recherche taboue (RT), et finalement, la méthode de descente pure avec la recherche à voisinage variable imbriqué (DP+RVVI).

Dans le but d'avoir une idée du comportement des méthodes, nous avons tracé les graphiques de l'évolution de la solution courante en fonction du nombre d'itérations complétées pour un problème représentatif. Ces graphiques sont présentés aux figures 7.4 à 7.14.

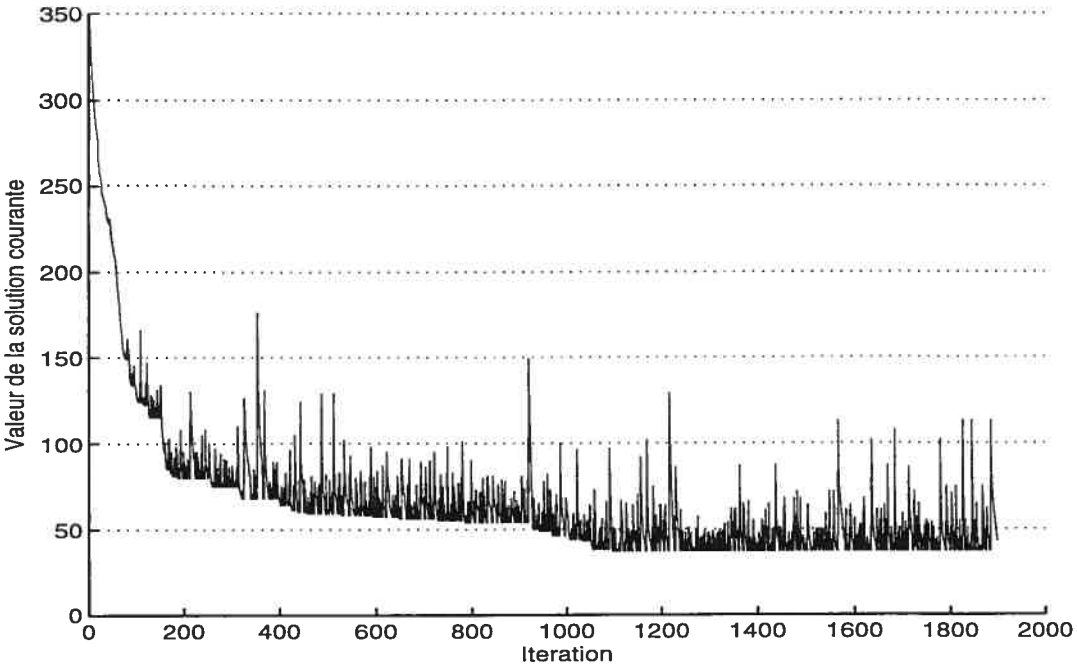

FIG. 7.4 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode de descente pure avec la recherche à voisinage variable imbriqué

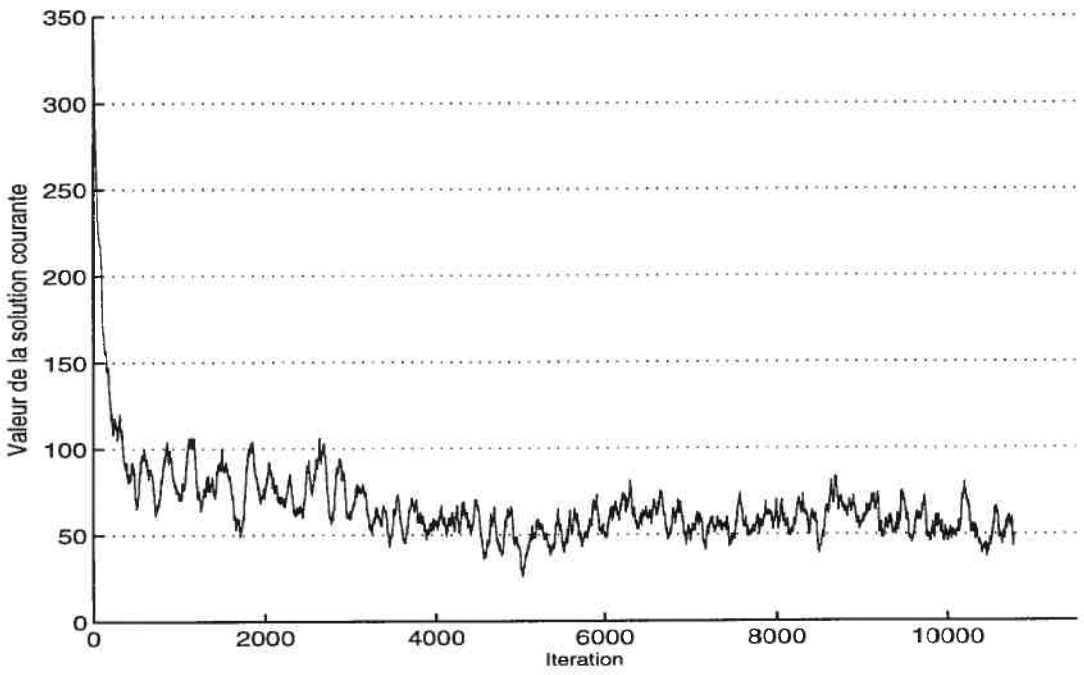

FIG. 7.5 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode de recherche taboue

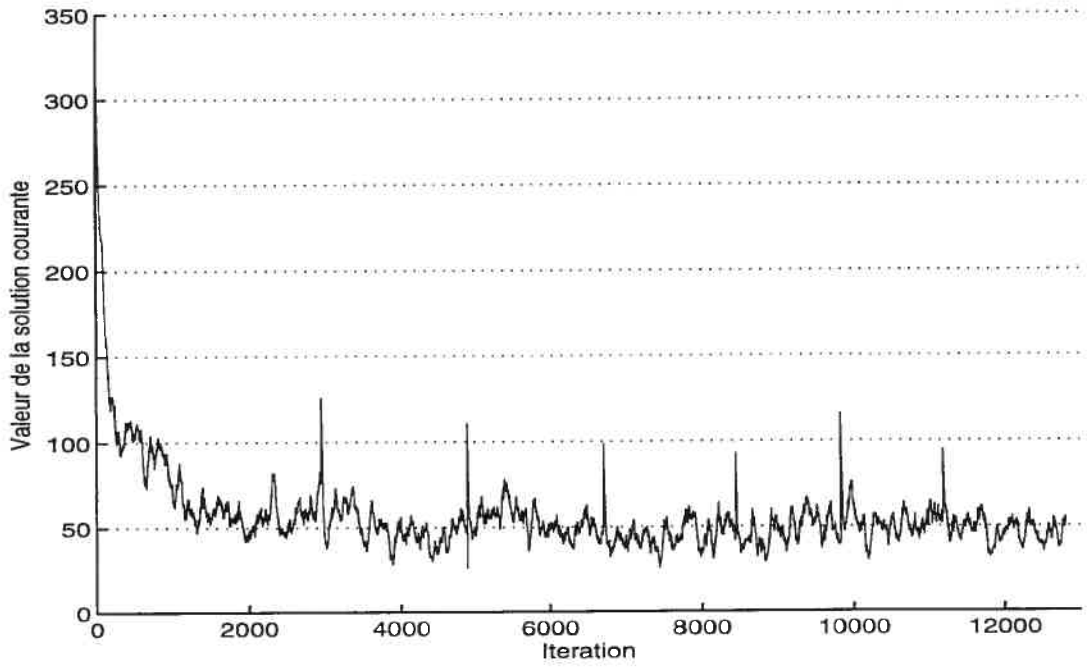

FIG. 7.6 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode de recherche taboue avec la diversification de premier ordre

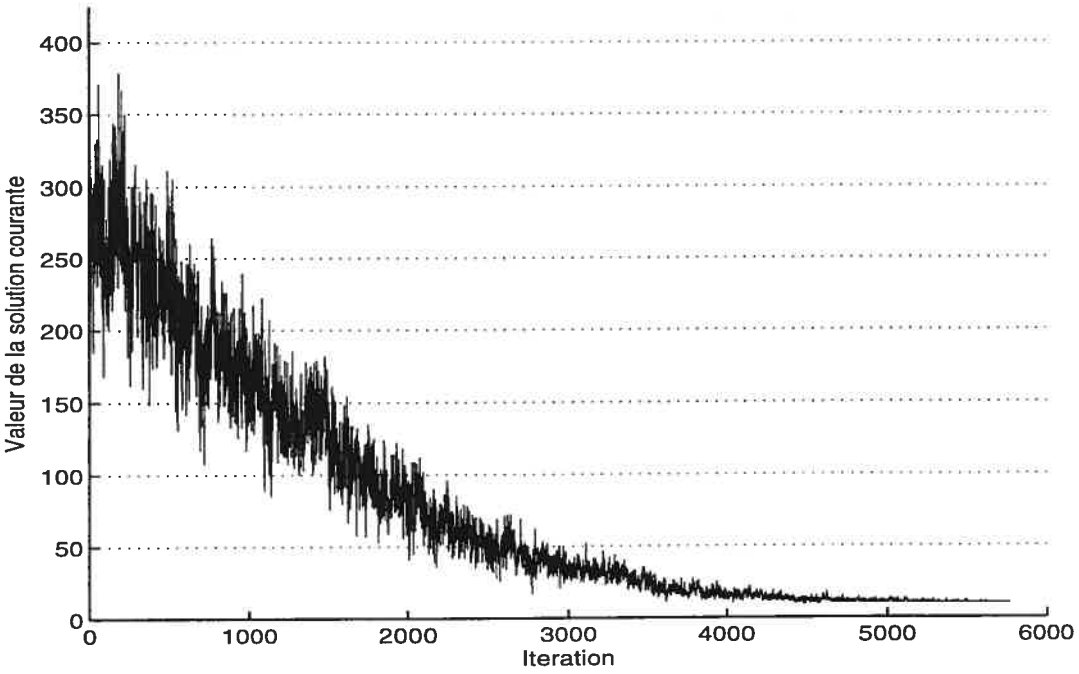

FIG. 7.7 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode du recuit simulé

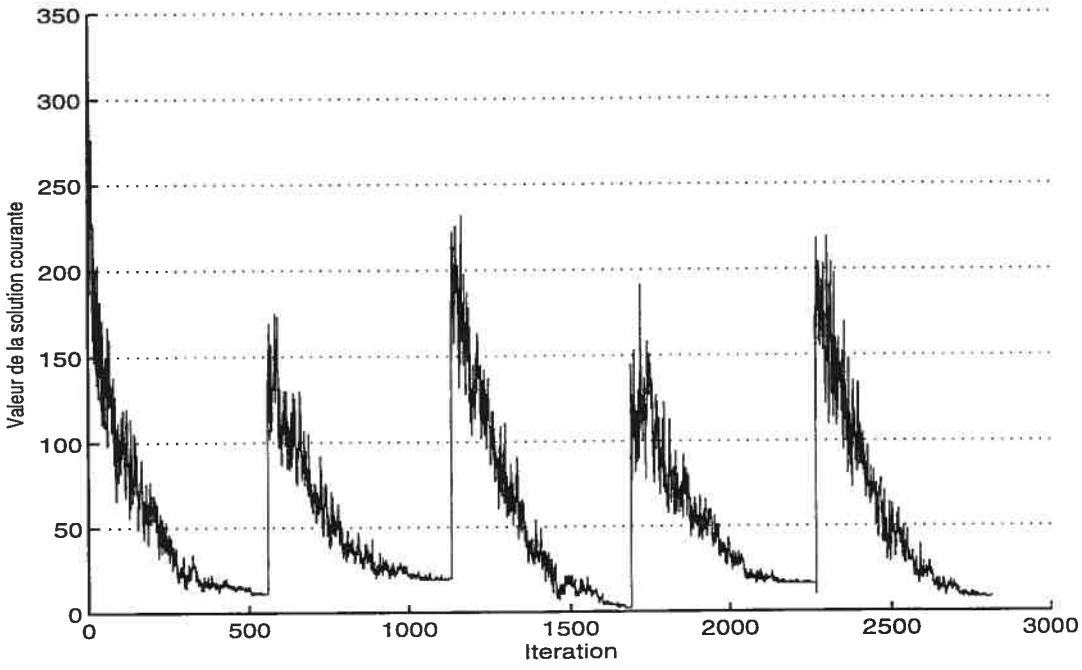

FIG. 7.8 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode du recuit simulé avec la recherche à voisinage variable imbriqué

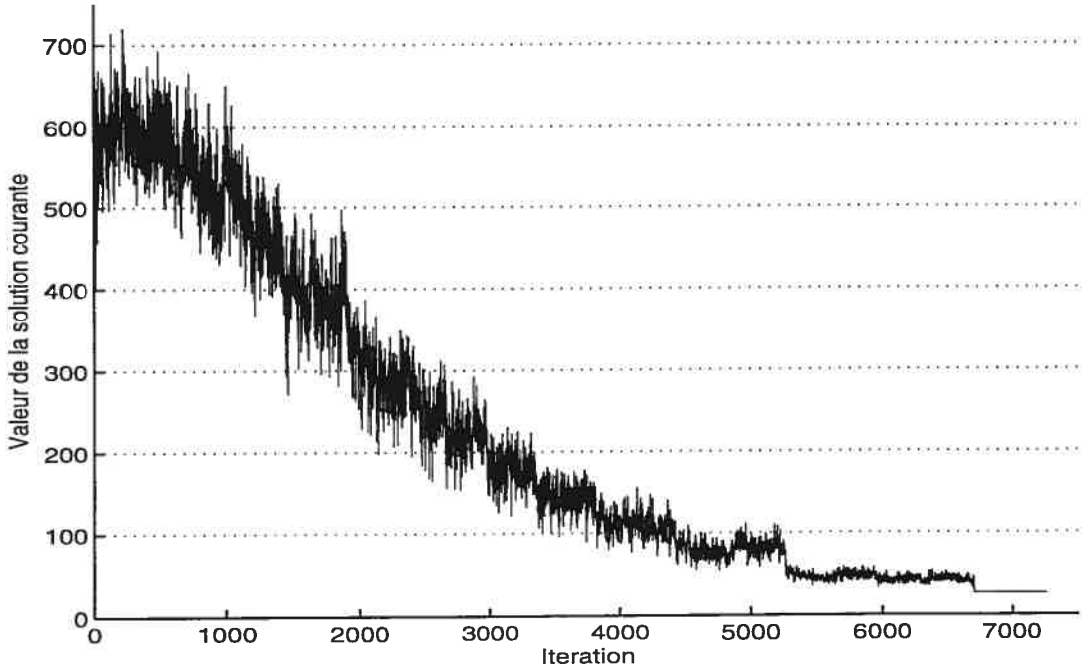

FIG. 7.9 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil standard

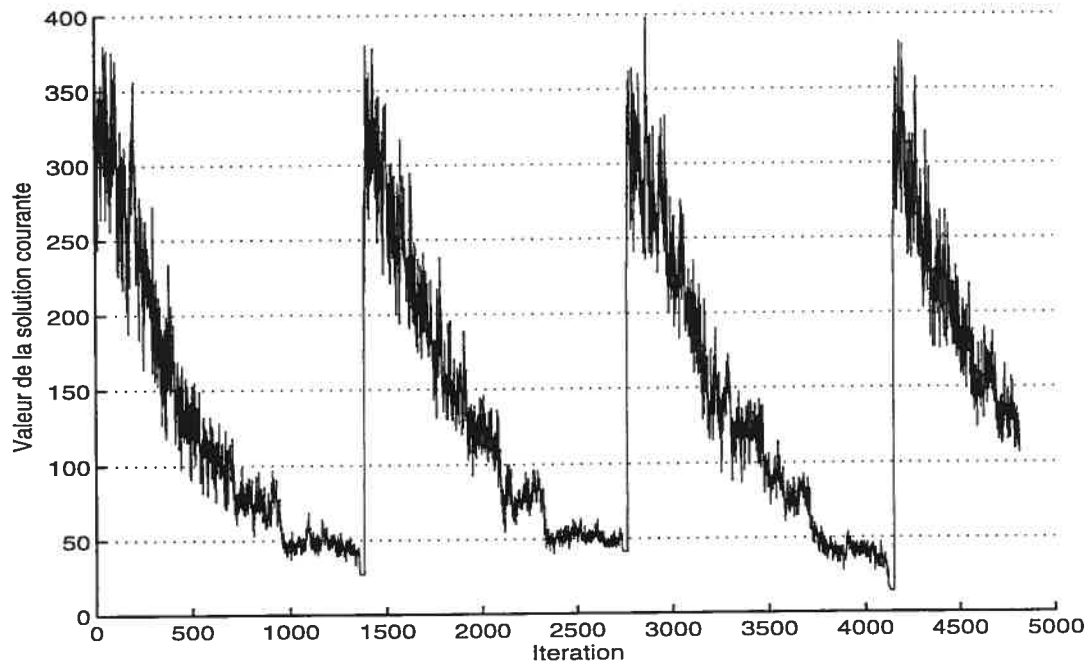

FIG. 7.10 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil standard avec la diversification de premier ordre

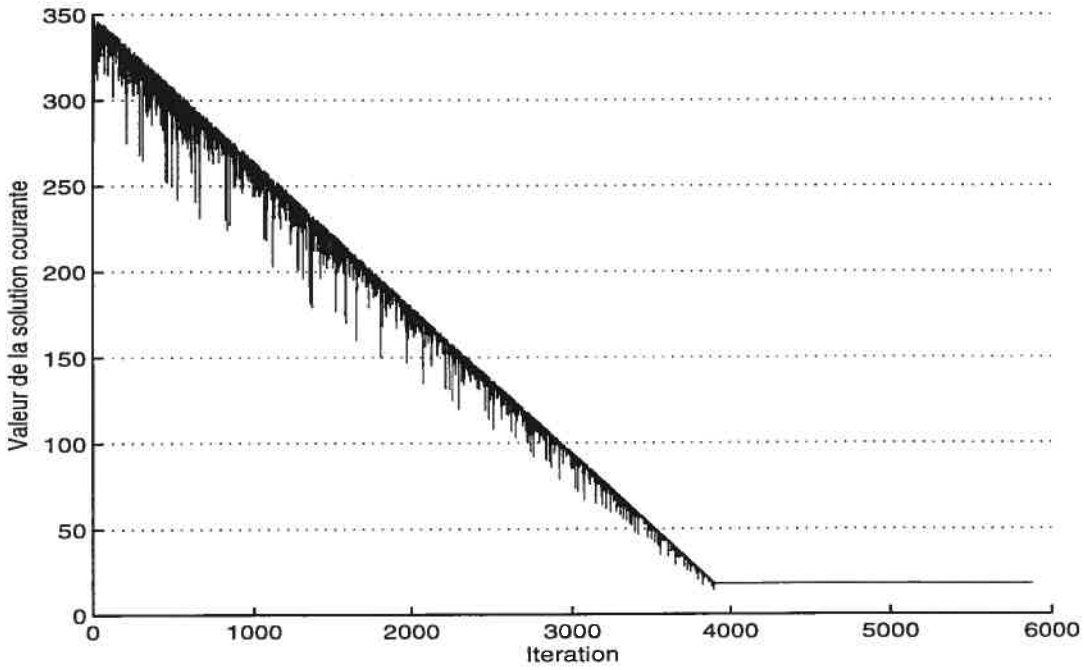

FIG. 7.11 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil - grand déluge

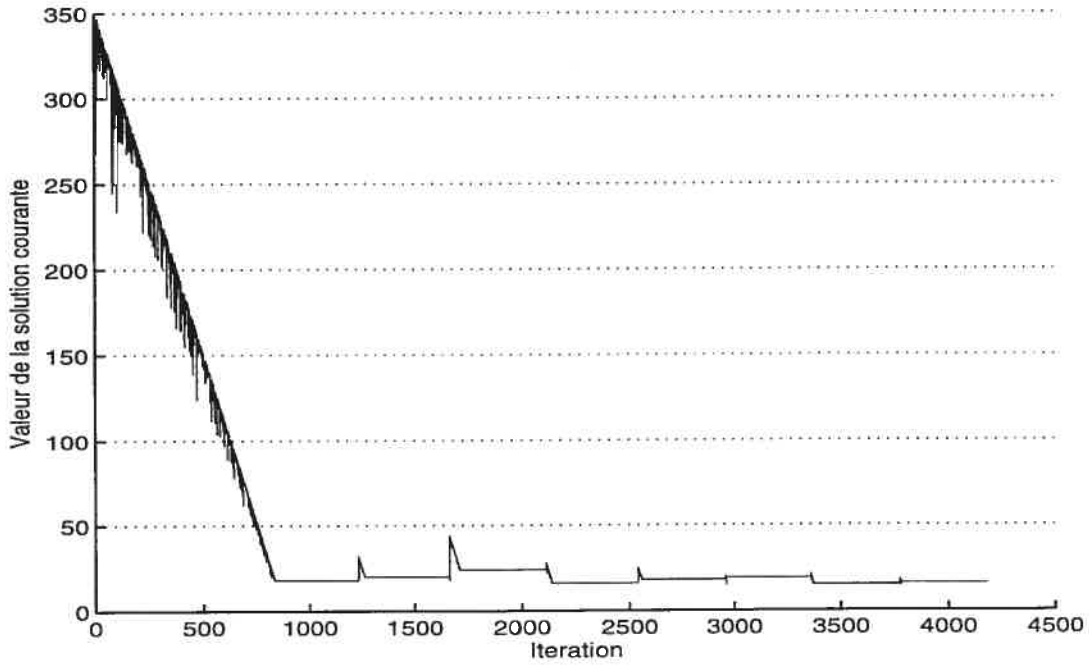

FIG. 7.12 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil - grand déluge avec la recherche à voisinage variable imbriqué

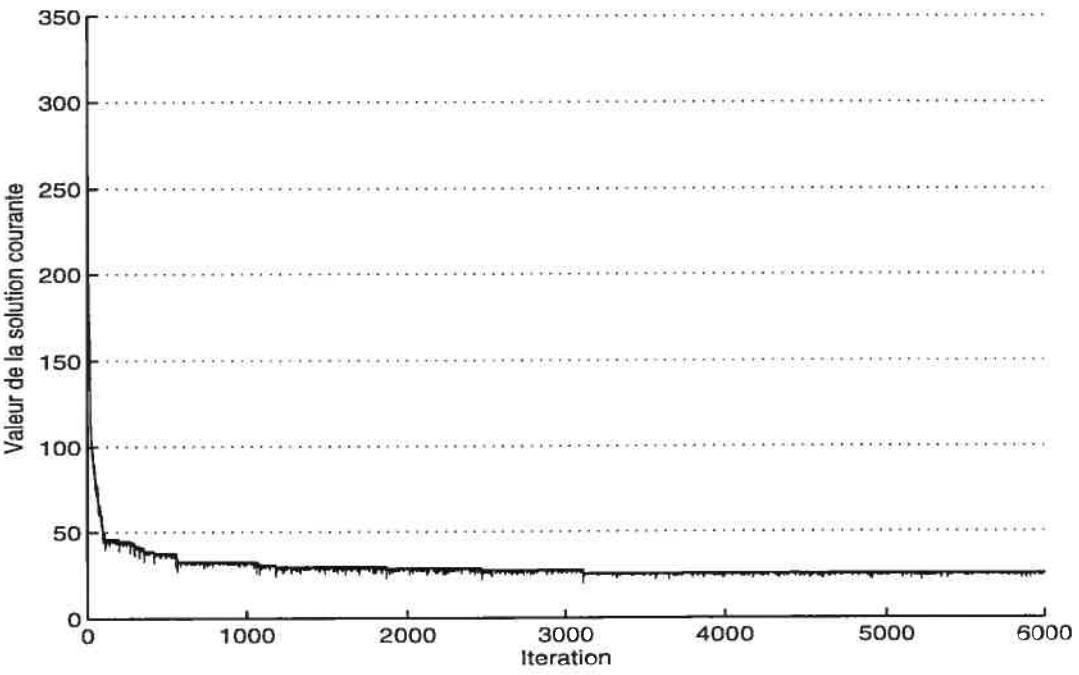

FIG. 7.13 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil à détérioration maximale

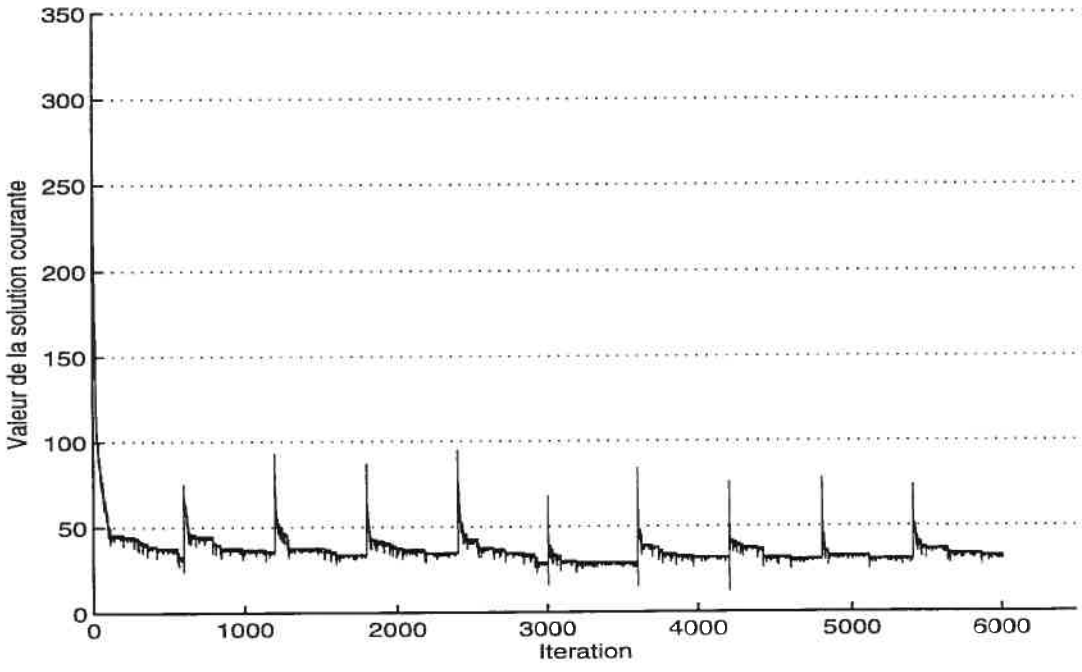

FIG. 7.14 – Valeur de la solution courante par itération pour un problème résolu à l'aide de la méthode d'acceptation avec seuil à détérioration maximale avec la diversification de premier ordre

7.5 Conclusion

En somme, les résultats obtenus avec la méthode du recuit simulé indiquent qu'elle est de loin la plus efficace des techniques de recherche locale testées pour la résolution du problème d'ordonnancement avec contraintes sur les ressources. En effet, cette technique se démarque favorablement de toutes les autres et ce, qu'elle soit employée seule ou avec une technique d'amélioration.

L'utilisation d'une stratégie d'amélioration est en outre bénéfique pour toutes les méthodes. La plus petite amélioration est observable pour la méthode d'acceptation avec seuil - grand déluge. Ceci peut probablement s'expliquer par le fait que les valeurs des paramètres pour cette méthode (comme pour les autres d'ailleurs) sont déterminées en fonction d'une solution initiale dont la valeur est beaucoup plus élevée en général que celle des solutions utilisées pour réinitialiser la méthode lors de l'emploi d'une stratégie d'amélioration. Le comportement de la méthode est illustrée dans le graphique 7.12 où les pics correspondent aux réinitialisations. Rappelons que, dans cette méthode, la valeur de la solution initiale sert à déterminer la valeur du seuil initial. Ainsi, les meilleures valeurs des paramètres à employer pour cette méthode, plus que pour les autres, sont directement liées à cette valeur de la solution initiale. Il s'ensuit que la convergence du seuil de la méthode d'acceptation avec seuil - grand déluge vers la valeur zéro est trop rapide après une réinitialisation avec la valeur du paramètres δ que nous utilisons. Pour corriger ce comportement, il faudrait plutôt utiliser une technique pour diminuer le seuil en fonction de $P(x')$ et du seuil courant afin que la diminution de dr ne soit pas débalancée (trop rapide ou trop lente) par rapport à celle de $P(x)$. Dueck compare d'ailleurs une telle formule avec la formule originale dans [19] avec succès.

Les résultats indiquent que la meilleure variante de la méthode d'acceptation avec seuil, la variante grand déluge, n'est que légèrement supérieure aux deux autres variantes lorsqu'aucune stratégie d'amélioration n'est employée. En effet, l'écart de la violation moyenne des solutions générées par la variante grand déluge par rapport à celle des solutions générées par la pire des trois variantes, la variante standard, n'est que de 1.4, et la variante grand déluge génère au maximum 14 meilleures solutions de plus que les deux autres variantes. Par contre, lorsque ces variantes sont couplées à une stratégie d'amélioration, la variante grand déluge

ne retire presqu'aucun bénéfice tandis que la variante standard en profite plus largement.

Une remarque importante doit être faite au sujet des deux types de recherche à voisinage variable testées. Il s'avère que celle utilisant les trois structures de voisinage imbriquées N^1, N^2 et N^3 domine celle utilisant plutôt les structures de voisinage N^1, N^2 et N^4, où N^4 procède par l'échange des périodes de départ des tâches entre elles. En effet, d'après les résultats concernant les stratégies d'amélioration, pour une même technique de recherche dans le voisinage et un même niveau d'utilisation, la recherche à voisinage variable imbriqué (RVVI) génère des solutions dont la violation moyenne des contraintes sur les ressources est au moins aussi basse que celle des solutions générées par la recherche à voisinage variable avec échanges (RVVE). La différence est de plus vraiment remarquable lorsque la descente pure est employée comme TRV. Dans ce cas, l'utilisation de la RVVI engendre des solutions dont la valeur moyenne est 30.2 contre 94.8 pour la RVVE.

Finalement, il est nécessaire d'apporter un bémol concernant les piètres résultats impliquant la méthode de recherche taboue. Il faut rappeler que pour cette méthode, les valeurs des paramètres sont déterminées en séquence au lieu d'être déterminées simultanément comme cela est le cas pour toutes les autres méthodes. L'approche utilisée pour la recherche taboue a le désavantage d'entraîner un biais vers les valeurs des paramètres employées pour en déterminer d'autres. Par contre, quand le nombre de paramètres et de valeurs que ceux-ci peuvent prendre sont élevés, comme c'est le cas pour la recherche taboue, cette approche doit être considérée. Malheureusement, il semble que cette façon de faire a désavantagé la méthode de recherche taboue par rapport aux autres.

Chapitre 8

Conclusion

En conclusion, le problème d'ordonnancement avec contraintes sur les ressources que nous étudions permet, par sa grande simplicité, d'adapter et de comparer facilement des techniques générales de recherche dans le voisinage destinées à le résoudre. Ce problème, à la différence d'un problème d'ordonnancement de projets, ne considère pas des relations de précédence entre les tâches mais plutôt des temps le plus tôt et le plus tard pour chaque tâche individuelle.

Le but poursuivi dans notre étude est d'implanter diverses techniques de recherche dans le voisinage présentées dans la littérature, d'appliquer certaines stratégies d'amélioration à ces méthodes et finalement, de comparer l'efficacité des méthodes et des combinaisons méthodes/stratégies d'amélioration pour la résolution du problème d'ordonnancement en un temps moyen donné. Pour ce faire, nous utilisons deux critères principaux d'efficacité afin d'évaluer une méthode : le nombre de problèmes pour lesquels cette méthode génère la meilleure des solutions générées par n'importe laquelle des méthodes considérées et la valeur moyenne de la fonction économique des solutions générées pour les problèmes testés. Le temps moyen d'exécution alloué à chaque résolution est d'environ 30 secondes.

Nos expérimentations nous ont permis d'identifier les meilleures combinaisons de méthodes et de stratégies d'amélioration pour notre formulation du problème d'ordonnancement de tâches. D'après les résultats obtenus et les critères considérés, la méthode du recuit simulé avec la recherche à voisinage variable utilisant des structures de voisinage imbriquées est la plus performante. En fait, la méthode du recuit simulé, employée seule ou non, s'est montrée nettement supérieure aux autres durant toutes les expérimentations.

Pour approfondir cette étude, il serait intéressant d'utiliser d'autres temps d'exécution afin de voir si les conclusions resteraient les mêmes. Dueck dans [19] affirme que la méthode d'acceptation avec seuil - grand déluge obtient d'excellents résultats à long terme si la vitesse de diminution du seuil est très basse. Ainsi, peut-être que cette méthode réussirait à surpasser le recuit simulé si nous lui laissions le temps. Ceci resterait à vérifier.

En outre, une suite logique à cette étude serait d'effectuer une étude semblable sur un problème d'ordonnancement de projets pour voir si les conclusions seraient les mêmes. Comme les méthodes d'acceptation avec seuil sont relativement peu étudiées en comparaison des trois autres et que cette étude indique qu'elles sont en mesure de les concurrencer, il est nécessaire de se demander s'il en serait autrement avec les problèmes d'ordonnancement de projets. La même question se pose aussi pour la méthode de recherche à voisinage variable utilisant des structures de voisinage imbriquées.

Bibliographie

[1] ALCARAZ, J., AND MAROTO, C. "A Genetic Algorithm for the Resource-Constrained Project Scheduling Problem". In *Proceedings of the Sixth International Workshop on Project Management and Scheduling* (Istanbul, 1998), Barbarosoğlu, G, Karabati, S., Özdamar, L., and Ulusoy, G., Eds., Boğaziçi University Printing Office, pp. 7–10.

[2] BAAR, T., BRUCKER, P., AND KNUST, S. "Tabu-Search Algorithms and Lower Bounds for the Resource-Constrained Project Scheduling Problem". In *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization* (1998), Voss, S., Martello, S., Osman, I., and Roucairol, C., Eds., Kluwer Academic, pp. 1–18.

[3] BAKER, K. *"Introduction to Sequencing and Scheduling"*. John Wiley and Sons, New York, 1974.

[4] BAUER, A., BULLNHEIMER, B., HARTL, R.F., AND STRAUSS, C. "An Ant Colony Optimization Approach for the Single Machine Total Tardiness Problem". In *Proceedings of the 1999 Congress on Evolutionary Computation (CEC99)* (Washington D.C., USA, 1999), IEEE Press, pp. 1445–1450.

[5] BLAZEWICZ, J. "Complexity of Computer Scheduling Algorithms under Resources Constraints". In *Proceedings First Meeting AFCET-SMF on Applied Mathematics* (Palaiseau, Poland, 1978), pp. 169–178.

[6] BOCTOR, F. "Some Efficient Multi-Heuristic Procedures for Resource-Constrained Project Scheduling". *European Journal of Operational Research 49* (1990), 3–13.

[7] BOCTOR, F. "Resource-Constrained Project Scheduling by Simulated Annealing". *International Journal in Production Research 34* (1996), 2335–2351.

[8] BOULEIMEN, K., AND LECOCQ, H. "A New Efficient Simulated Annealing Algorithm for the Resource-Constrained Project Scheduling Problem". In *Proceedings of the Sixth International Workshop on Project Management*

and Scheduling (Istanbul, 1998), Barbarosoğlu, G., Karabati, S., Özdamar, L., and Ulusoy, G., Eds., Boğaziçi University Printing Office, pp. 19–22.

[9] BRUCKER, P., DREXL, A., MÖHRING, R., NEUMANN, K., AND PESCH, E. "Resource-Constrained Project Scheduling : Notation, Classification, Models and Methods". *European Journal of Operational Research 112* (1999), 3–41.

[10] BRUCKER, P., KNUST, S., SCHOO, A., AND THIELE, O. "A Branch-and-Bound Algorithm for the Resource-Constrained Project Scheduling Problem". *European Journal of Operational Research 107* (1998), 272–288.

[11] CARRUTHERS, J.A., AND BATTERSBY, A. "Advances in Critical Path Methods". *Operational Research Quarterly 17* (1966), 359–380.

[12] CERNY, V. "Thermodynamical Approach to the Traveling Salesman Problem : an Efficent Simulation Algorithm". *Journal of Optimization Theory and Applications 45* (1985), 41–51.

[13] CHEN, C.L., VEMPATI, V.S., AND ALJABER, N. "An Application of Genetic Algorithms for Flow Shop Problems". *European Journal of Operational Research 80* (1995), 389–396.

[14] CHENG, R., GEN, M., AND TSUJIMURA, Y. "A Tutorial Survey of Job Shop Scheduling problems Using Genetic Algorithms - I : Representation". *Computers and Industrial Engineering 30*, 4 (1996), 983–997.

[15] COLORNI, A., DORIGO, M., AND MANIEZZO, V. "Investigation of Some Properties of an Ant Algorithm". In *Proceedings of European Conference on Artificial Life ECAL'91* (Paris, France, 1991), Elsevier Publishing, pp. 134–142.

[16] COLORNI, A., DORIGO, M., MANIEZZO, V., AND TRUBIAN, M. "Ant System for Job-Shop Scheduling". *JORBEL - Belgian Journal of Operations Research, Statistics and Computer Science 34* (1994), 39–53.

[17] DEMEULEMEESTER, E., AND HERROELEN, W. "A Branch-and-Bound Procedure for the Generalized Resource-Constrained Project Scheduling Problem". *Operations Research 45*, 2 (1992), 201–212.

[18] DEMEULEMEESTER, E., AND HERROELEN, W. "A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem". *Management Science 38*, 12 (1997), 1803–1818.

[19] DUECK, G. "New Optimization Heuristics : The Great Deluge Algorithm and the Record-to-Record Travel". *Journal of Computational Physics 104* (1993), 86–92.

[20] DUECK, G., AND SCHEUER, T. "Threshold Accepting : A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing". *Journal of Computational Physics 90* (1990), 161–175.

[21] GAGNON, M. *"Modélisation et résolution heuristique de l'allocation des ressources en gestion de projets"*. Thèse Ph.D., Université Laval, Québec, Canada, Apr. 2002.

[22] GENDREAU, M., AND POTVIN, J.-Y. "Metaheuristics in Combinatorial Optimization". Tech. Rep. CRT-2002-36, Centre de recherche sur les transports, Université de Montréal, Oct. 2002.

[23] GLOVER, F. "Future Paths for Integer Programming and Links to Artificial Intelligence". *Computers and Operations Research 13* (1986), 533–549.

[24] GUPTA, M.C., GUPTA, A., AND KUMAR, A. "Minimizing Flow Time Variance in a Single Machine Using Genetic Algorithms". *European Journal of Operational Research 70* (1993), 289–303.

[25] HANSEN, P. "The Steepest Ascent Mildest Descent Heuristic for Combinatorial Programming". In *Congress on Numerical Methods in Combinatorial Optimization* (Capri, Italie, 1986).

[26] HANSEN, P., AND MLADENOVIĆ, N. "Variable Neighborhood Search : Principles and Applications". *European Journal of Operations Research 130* (2001), 449–467.

[27] HARTMANN, S. "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling". *Naval research Logistics 45* (1998), 733–750.

[28] HARTMANN, S., AND KOLISH, R. "Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem". *European Journal of Operational Research 127* (2000), 394–407.

[29] HELD, M., AND KARP, R.M. "A Dynamic Approach to Sequencing Problems". *Journal of Society for Industrial and Applied Mathematics 10*, 1 (1962), 196–210.

[30] ICMELI, O. AND ERENGUC, S.S. AND ZAPPE, C.J. "Project Scheduling Problems : A survey". *International Journal of Operations and Production Management 13* (1993), 80–91.

[31] JANIAK, A. "Minimization of the Maximum Tardiness in One-Machine Scheduling Problem Subject to Precedence and Resource Constraints". *Systems Analysis Modelling Simulation 4* (1987), 549–556.

[32] JEFFCOAT, D.E., AND BULFIN, R.L. "Simulated-Annealing for Resource-Constrained Scheduling". *European Journal of Operational Research 70* (1993), 43–51.

[33] KELLY, J.P., LAGUNA, M., AND GLOVER, F. "A Study of Diversification strategie for the Quadratic Assignment Problem". *Computers and Operations Research 21* (1994), 885–893.

[34] KIRKPATRICK, S., GELATT, JR, C.D., AND VECCHI, M.P. "Optimization by Simulated Annealing". *Science 220*, 4598 (1983), 671–680.

[35] KOLISH, R. "Efficient Priority Rules for the Resource-Constrained Project Scheduling Problem". *Journal of Operations Management 14* (1996), 179–192.

[36] KOLISH, R, AND HARTMANN, S. "Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling : Classification and Computational Analysis". In *Project Scheduling : Recent Models, Algorithms and Applications* (1999), J. Węglarz, Ed., Kluwer Academic, pp. 147–178.

[37] KOULAMAS, C. "The Total Tardiness Problems : Review and Extensions". *Operations Research 42*, 6 (1994), 1025–1041.

[38] LAGUNA, M., BARNES, J.W., AND GLOVER, F. "Tabou Search Methods for a Single Machine Scheduling Problem". *Journal of Intelligent Manufacturing 2* (1991), 63–74.

[39] LEE, J.-K., AND KIM, Y.-D. "Search Heuristics for Resource Constrained Project Scheduling". *Journal of the Operational Research Society 47* (1996), 678–689.

[40] LIAW, C.-F. "A Tabou Search Algorithm for the Open Shop Scheduling Problem". *Computers and Operations Research 26* (1999), 109–126.

[41] MERKLE, D., MIDDENDORF, M., AND SCHMECK, H. "Ant Colony Optimization for Resource-Constrained Project Scheduling". In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)* (Las Vegas, Nevada, USA, 2000), Morgan Kaufmann, pp. 893–900.

[42] METROPOLIS, N., ROSENBLUTH, A., ROSENBLUTH, M., TELLER, A., AND TELLER, E. "Equation of State Calculations by Fast Computing Machine". *Journal of Chemical Physics 21* (1953), 1087–1091.

[43] MINGOZZI, A., MANIEZZO, V., RICCIARDELLI, S., AND BIANCO, L. "An Exact Algorithm for the Resource-Constrained Project Scheduling Problem

Based on a New Mathematical Formulation". *Management Science 44* (1998), 714–729.

[44] MLADENOVIĆ, N., AND HANSEN, P. "Variable Neighborhood Search". *Computers and Operations Research 24*, 11 (1997), 1097–1100.

[45] MOORE, J. "A N-job One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs". *Management Science 15*, 1 (1968), 102–109.

[46] MORTON, T.E., AND PENTICO, D.W. *"Heuristic Scheduling Systems with Applications to Production Systems and Project Management"*. John Wiley and Sons, New York, 1993.

[47] MURATA, T., ISHIBUCHI, H., AND TANAKA, H. "Genetic Algorithms for Flow Shop Scheduling Problem". *Computers and Industrial Engineering 30*, 4 (1996), 1061–1071.

[48] NEPALLI, V.R., CHEN, C.L., AND GUPTA, J.N.D. "Genetic Algorithms for the Two-sate Bicriteria Flow Shop Problem". *European Journal of Operational Research 95* (1996), 356–373.

[49] NOWICKI, E., AND SMUTNICKI, C. "A Fast Taboo Search Algorithm for the Job Shop Problem". *Management Science 42*, 6 (1996), 797–813.

[50] ÖZDAMAR, L. "A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling". *IEEE Transactions on Systems, Man, and Cybernetics 29* (1999), 44–59.

[51] ÖZDAMAR, L., AND ULUSOY, G. "A Survey on the Resource-Constrained Project Scheduling Problem". *IIE Transactions 27* (1995), 574–586.

[52] PATTERSON, J. "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem". *Management Science 30*, 7 (1984), 854.

[53] POTTS, C.N., AND VAN WASSENHOVA, L.N. "Algorithms for Scheduling a Single Machine to Minimize the Weighted Number of Late Jobs". *Management Science 34*, 7 (1988), 834–858.

[54] SPRECHER, A. "Scheduling Resource-Constrained Projects Competitively at Modest Memory Requirements". *Management Science 46*, 5 (2000), 710–723.

[55] STÜTZLE, T. "An Ant Approach to the Flow Shop Problem". In *Proceedings of the 6th European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)* (Aachen, 1998), vol. 3, Verlag Mainz, pp. 1560–1564.

[56] TAILLARD, E. "Robust Tabu Search for the Quadratic Assignment Problem". *Parallel Computing 17* (1991), 443–455.

Annexe A

Résultats finaux détaillés pour la méthode de descente pure

Les résultats finaux obtenus pour chaque problème résolu avec la méthode de descente pure sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau A.1 et ceux obtenus avec l'utilisation de la recherche à voisinage variable imbriqué sont présentés au tableau A.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la recherche à voisinage variable imbriqué est employée, le nombre total de changements de voisinage (*Changements de k*) et le nombre de retours à la structure de voisinage N^1 (*Retours à* N^1) sont mentionnés tandis que le nombre d'itérations complétées (*Nombre d'itérations*) est mentionné sinon.

Lorsque la méthode de descente pure est employée seule, la méthode se termine lorsqu'un minimum local est atteint. Lorsqu'une stratégie d'amélioration est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et chaque application de la descente pure se termine lorsqu'un minimum local par rapport au voisinage courant est atteint.

TAB. A.1: Résultats détaillés pour la méthode de descente pure

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
1	562	366	0	49
2	528	273	0	56
3	514	267	0	42
4	419	281	1	37
5	466	227	0	45
6	368	201	0	43
7	450	207	0	58
8	539	265	1	67
9	366	218	0	39
10	402	217	0	58
11	368	190	0	45
12	453	209	1	61
13	508	230	0	50
14	515	216	0	61
15	448	231	0	51
16	374	224	1	37
17	453	234	0	66
18	451	225	0	52
19	355	222	0	34
20	468	238	0	43
21	529	237	1	50
22	438	181	0	55
23	511	232	0	69
24	396	277	0	36
25	503	234	1	59
26	434	217	0	50
27	320	203	0	31
28	350	231	0	42
29	304	150	1	51
30	258	200	0	18
31	410	169	0	54
32	335	222	0	40

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
33	444	208	0	65
34	590	304	1	69
35	468	242	0	61
36	372	208	0	42
37	380	271	1	31
38	449	263	0	45
39	413	191	0	56
40	336	204	0	34
41	447	299	0	37
42	394	176	1	53
43	398	218	0	40
44	349	164	0	50
45	396	195	0	42
46	388	135	1	59
47	358	126	0	68
48	498	254	0	73
49	374	205	0	36
50	623	294	1	69
51	416	219	0	47
52	338	178	0	53
53	503	218	0	67
54	458	218	1	61
55	357	240	0	35
56	467	253	0	48
57	458	212	0	69
58	428	199	1	59
59	404	269	0	53
60	288	130	0	42
61	348	233	0	28
62	363	280	0	29
63	426	153	1	63
64	377	210	0	46
65	613	280	0	77

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
66	434	250	1	48
67	586	305	0	61
68	290	204	0	30
69	577	349	0	47
70	523	263	0	63
71	293	163	0	41
72	453	231	0	55
73	368	224	0	40
74	418	228	0	55
75	308	180	1	47
76	514	247	0	56
77	501	339	0	44
78	338	167	0	55
79	381	274	1	32
80	314	126	0	48
81	461	298	0	50
82	408	216	0	48
83	293	148	0	38
84	339	242	1	26
85	412	162	0	51
86	381	195	0	50
87	432	217	0	59
88	356	242	0	32
89	439	257	1	36
90	248	105	0	45
91	432	315	0	34
92	671	287	0	69
93	720	203	1	82
94	473	214	0	74
95	319	154	0	39
96	352	212	0	31
97	416	243	1	35
98	374	222	0	45

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
99	446	244	0	57
100	378	178	0	51
Moy :	423.6	224.5	0.2	49.3
Écart-type :	87.7	48.6	0.4	12.7

TAB. A.2: Résultats détaillés pour la méthode de descente pure avec la recherche à voisinage variable imbriqué

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
1	562	42	30	716	247
2	528	54	30	667	228
3	514	31	30	684	236
4	419	39	30	684	237
5	466	34	30	739	253
6	368	42	30	718	248
7	450	42	30	715	247
8	539	34	30	759	258
9	366	19	30	716	245
10	402	20	30	696	238
11	368	27	30	753	259
12	453	39	30	736	249
13	508	28	30	798	273
14	515	27	30	774	262
15	448	58	30	633	217
16	374	31	30	741	252
17	453	28	30	775	263
18	451	43	30	763	261

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
19	355	21	30	737	251
20	468	31	30	675	232
21	529	16	30	723	246
22	438	64	30	721	244
23	511	44	30	709	240
24	396	48	30	669	229
25	503	23	30	727	249
26	434	26	30	719	245
27	320	30	30	701	240
28	350	37	30	695	237
29	304	49	30	794	269
30	258	46	30	788	267
31	410	17	30	732	249
32	335	30	30	696	238
33	444	28	30	629	217
34	590	64	30	718	245
35	468	31	30	714	244
36	372	24	30	722	244
37	380	33	30	708	243
38	449	25	30	722	246
39	413	17	30	677	231
40	336	20	30	714	242
41	447	47	30	703	242
42	394	34	30	792	267
43	398	38	30	668	227
44	349	34	30	760	261
45	396	56	30	745	251
46	388	26	30	817	276
47	358	36	30	652	225
48	498	41	30	746	253
49	374	6	30	753	256
50	623	45	30	766	261
51	416	40	30	651	224

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
52	338	19	30	782	267
53	503	32	30	694	237
54	458	37	30	735	251
55	357	30	30	709	243
56	467	46	30	753	255
57	458	43	30	673	231
58	428	14	30	721	250
59	404	35	30	736	250
60	288	30	30	796	270
61	348	36	30	701	239
62	363	28	30	698	238
63	426	48	30	684	231
64	377	38	30	730	247
65	613	34	30	701	238
66	434	27	30	718	245
67	586	28	30	749	255
68	290	26	30	771	260
69	577	43	30	700	237
70	523	26	30	662	224
71	293	11	30	750	255
72	453	64	30	777	267
73	368	28	30	667	227
74	418	38	30	743	254
75	308	30	30	694	238
76	514	29	30	750	255
77	501	39	30	710	242
78	338	41	30	727	245
79	381	26	30	719	247
80	314	22	30	801	274
81	461	20	30	824	281
82	408	22	30	706	242
83	293	33	30	718	245
84	339	32	30	810	274

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
85	412	25	30	760	261
86	381	45	30	734	249
87	432	36	30	679	231
88	356	42	30	682	234
89	439	39	30	739	253
90	248	20	30	795	268
91	432	42	30	750	255
92	671	32	30	683	233
93	720	55	30	695	236
94	473	32	30	744	252
95	319	34	30	734	250
96	352	37	30	733	248
97	416	23	30	756	256
98	374	50	30	732	250
99	446	35	30	674	228
100	378	34	30	751	254
Moy :	423.6	34.0	30.0	725.6	247.4
Écart-type :	87.7	11.4	0.0	41.0	13.5

Annexe B

Résultats finaux détaillés pour la méthode de recherche taboue

Les résultats finaux obtenus pour chaque problème résolu avec la méthode de recherche taboue sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau B.1 et ceux obtenus avec l'utilisation de la diversification de premier ordre au niveau d'utilisation moyen sont présentés au tableau B.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la diversification de premier ordre est employée le nombre de réinitialisation à l'aide de la DPO est mentionné (*Nombre de diversifications*) tandis que le nombre d'itérations complétées (*Nombre d'itérations*) est mentionné si aucune stratégie d'amélioration n'est utilisée.

Lorsque la méthode de recherche taboue est employée seule, le nombre maximal d'itérations est fixé à 17000 et le nombre maximal d'itérations sans amélioration de x^* est fixé à 5750. Lorsqu'une stratégie d'amélioration est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et, pour chaque application de la recherche taboue, le nombre maximal d'itérations est fixé à 1500 et le nombre maximal d'itérations sans amélioration de x^* est fixé à 450.

TAB. B.1: Résultats détaillés pour la méthode de recherche taboue

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
1	562	33	40	10405
2	528	27	32	7485
3	514	23	23	3829
4	419	33	30	6479
5	466	22	29	6142
6	368	29	40	10771
7	450	36	24	3840
8	539	24	34	8106
9	366	19	37	9383
10	402	42	29	6278
11	368	30	24	3971
12	453	14	35	8861
13	508	24	19	2071
14	515	15	38	10047
15	448	31	29	6076
16	374	52	21	2874
17	453	33	23	3715
18	451	26	25	4417
19	355	24	16	972
20	468	32	40	10511
21	529	27	29	6009
22	438	21	30	6450
23	511	31	23	3743
24	396	26	33	7735
25	503	35	30	6593
26	434	20	26	4896
27	320	14	24	4184
28	350	26	26	5030
29	304	20	42	11344
30	258	39	23	3866
31	410	21	28	5551
32	335	22	33	7949

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
33	444	14	26	4643
34	590	21	34	8338
35	468	21	28	5634
36	372	17	41	16686
37	380	35	22	3278
38	449	26	36	8961
39	413	15	31	7181
40	336	27	22	3218
41	447	26	30	6243
42	394	15	22	3323
43	398	21	41	13479
44	349	31	42	15112
45	396	23	41	16614
46	388	27	16	947
47	358	53	16	719
48	498	35	41	12702
49	374	22	17	1361
50	623	40	35	8555
51	416	13	41	12186
52	338	17	25	4391
53	503	27	41	13113
54	458	42	25	4452
55	357	26	37	8915
56	467	12	36	9139
57	458	32	31	7091
58	428	12	42	16192
59	404	42	33	8207
60	288	20	42	14513
61	348	18	42	16411
62	363	35	21	3025
63	426	25	26	5146
64	377	21	37	9118
65	613	39	31	7103

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
66	434	23	34	8196
67	586	25	32	7415
68	290	21	31	7099
69	577	25	41	14448
70	523	16	41	13812
71	293	29	39	10078
72	453	18	41	11730
73	368	26	23	3912
74	418	22	42	12331
75	308	24	27	5052
76	514	25	34	8155
77	501	27	28	5942
78	338	33	25	4422
79	381	23	39	10054
80	314	14	24	4532
81	461	26	22	3238
82	408	23	42	11543
83	293	30	22	3556
84	339	23	24	3871
85	412	24	32	7598
86	381	25	30	6323
87	432	26	41	11064
88	356	27	27	5515
89	439	37	21	2725
90	248	22	19	2240
91	432	16	36	8982
92	671	20	34	8189
93	720	60	18	1885
94	473	35	22	3197
95	319	31	42	11326
96	352	17	41	12824
97	416	23	35	8548
98	374	27	42	16073

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
99	446	43	31	6682
100	378	40	26	5329
Moy :	423.6	26.5	30.9	7434.7
Écart-type :	87.7	8.9	7.7	3974.4

TAB. B.2: Résultats détaillés pour la méthode de recherche taboue avec la diversification de premier ordre

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
1	562	34	31	7
2	528	14	31	6
3	514	22	31	6
4	419	28	31	7
5	466	16	31	7
6	368	31	30	6
7	450	30	30	8
8	539	19	31	8
9	366	23	31	8
10	402	38	31	8
11	368	19	31	7
12	453	20	31	8
13	508	13	31	7
14	515	16	31	8
15	448	43	31	8
16	374	46	30	8
17	453	25	31	7
18	451	34	31	7

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
19	355	16	31	8
20	468	31	30	7
21	529	22	30	7
22	438	36	31	5
23	511	24	31	6
24	396	19	31	7
25	503	27	31	8
26	434	19	31	8
27	320	19	30	6
28	350	26	31	6
29	304	11	30	7
30	258	46	31	8
31	410	13	31	7
32	335	29	30	6
33	444	14	31	6
34	590	46	31	6
35	468	20	31	8
36	372	19	30	7
37	380	23	30	7
38	449	27	30	7
39	413	18	30	7
40	336	12	30	6
41	447	19	31	7
42	394	22	31	7
43	398	30	31	8
44	349	43	30	7
45	396	47	30	8
46	388	17	31	8
47	358	35	31	8
48	498	31	30	5
49	374	15	30	7
50	623	29	30	6
51	416	25	30	8

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
52	338	14	30	8
53	503	37	31	7
54	458	31	31	7
55	357	24	31	8
56	467	11	31	6
57	458	31	30	7
58	428	18	31	6
59	404	39	31	5
60	288	36	31	8
61	348	15	31	7
62	363	26	31	7
63	426	27	31	9
64	377	29	30	8
65	613	36	31	5
66	434	12	30	5
67	586	30	31	7
68	290	22	31	7
69	577	29	31	7
70	523	11	31	5
71	293	20	30	7
72	453	48	31	7
73	368	18	31	8
74	418	33	31	8
75	308	21	30	7
76	514	34	31	7
77	501	16	30	5
78	338	32	30	6
79	381	29	31	7
80	314	17	31	6
81	461	16	31	7
82	408	20	31	8
83	293	29	31	7
84	339	23	31	6

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
85	412	20	30	6
86	381	19	31	9
87	432	20	30	6
88	356	38	30	6
89	439	41	31	6
90	248	15	31	8
91	432	21	31	6
92	671	35	31	6
93	720	58	30	8
94	473	28	31	6
95	319	42	31	7
96	352	40	31	6
97	416	15	30	6
98	374	32	31	7
99	446	17	31	7
100	378	30	31	5
Moy :	423.6	26.1	30.7	6.9
Écart-type :	87.7	10.1	0.5	1.0

Annexe C

Résultats finaux détaillés pour la méthode du recuit simulé

Les résultats finaux obtenus pour chaque problème résolu avec la méthode du recuit simulé sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau C.1 et ceux obtenus avec l'utilisation de la recherche à voisinage variable imbriqué au niveau intense sont présentés au tableau C.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la recherche à voisinage variable est employée, le nombre de retours à la structure de voisinage N^1 (*Retours à* N^1) et le nombre total de changements de voisinage (*Changements de* k) sont mentionnés tandis que le nombre d'itérations complétées (*Nombre d'itérations*) est mentionné sinon.

Lorsque la méthode du recuit simulé est employée seule, $nbItMax$ est fixé à 10000 et $nbItInutileMax$ est fixé à 150. Lorsque la recherche à voisinage variable imbriqué est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et, pour chaque application du recuit simulé, $nbItMax$ est fixé à 600 et $nbItInutileMax$ est fixé à 20.

TAB. C.1: Résultats détaillés pour la méthode du recuit simulé

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
1	562	11	31	4158
2	528	8	31	4630
3	514	12	30	5257
4	419	19	31	5033
5	466	7	31	4879
6	368	10	30	4658
7	450	4	30	5270
8	539	3	30	5094
9	366	4	30	4245
10	402	2	30	4639
11	368	8	29	4420
12	453	3	29	4424
13	508	2	29	4058
14	515	0	24	4653
15	448	30	29	4194
16	374	2	31	4400
17	453	5	28	4935
18	451	9	30	3870
19	355	4	29	4911
20	468	12	32	5251
21	529	7	30	3711
22	438	1	31	5257
23	511	10	31	4162
24	396	5	30	4154
25	503	55	30	4215
26	434	4	30	4891
27	320	7	30	4854
28	350	9	29	4477
29	304	14	30	4615
30	258	2	31	4463
31	410	3	30	4892
32	335	10	30	4563

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
33	444	4	30	4752
34	590	16	29	3823
35	468	15	30	4542
36	372	7	29	4357
37	380	7	30	3610
38	449	9	30	4752
39	413	8	29	3707
40	336	5	30	4238
41	447	4	30	4417
42	394	5	29	4936
43	398	7	30	4723
44	349	21	30	4764
45	396	15	30	3711
46	388	9	30	4794
47	358	15	31	4112
48	498	12	29	3747
49	374	6	29	5096
50	623	26	29	4521
51	416	4	32	4333
52	338	9	29	4604
53	503	4	30	5542
54	458	28	30	3558
55	357	5	30	4391
56	467	2	30	4403
57	458	12	29	4473
58	428	5	29	5085
59	404	24	29	3698
60	288	8	30	4658
61	348	4	29	4281
62	363	4	30	4678
63	426	8	29	3824
64	377	7	29	4655
65	613	18	28	4987

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
66	434	5	29	5240
67	586	22	29	4259
68	290	9	29	4770
69	577	11	29	4192
70	523	18	30	5577
71	293	14	31	4566
72	453	9	30	4871
73	368	8	29	4362
74	418	21	31	3685
75	308	13	30	3896
76	514	5	30	4218
77	501	5	31	4408
78	338	3	30	4203
79	381	8	30	4203
80	314	1	30	5779
81	461	4	30	3932
82	408	7	29	4674
83	293	2	31	5154
84	339	17	31	5476
85	412	14	31	4320
86	381	4	31	4655
87	432	20	30	4367
88	356	9	29	4043
89	439	7	30	4086
90	248	4	30	4452
91	432	8	30	5779
92	671	9	30	4397
93	720	6	29	4152
94	473	15	29	3986
95	319	18	30	4592
96	352	8	29	5028
97	416	9	31	4913
98	374	8	30	4473

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
99	446	7	32	5272
100	378	20	30	4365
Moy :	423.6	9.6	29.8	4533.5
Écart-type :	87.7	7.8	1.0	492.9

TAB. C.2: Résultats détaillés pour la méthode du recuit simulé avec la recherche à voisinage variable imbriqué

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
1	562	11	31	4	3
2	528	9	30	5	5
3	514	11	31	5	3
4	419	15	31	4	2
5	466	9	31	5	3
6	368	6	31	5	4
7	450	6	30	5	4
8	539	10	31	5	3
9	366	4	31	5	3
10	402	4	29	5	4
11	368	2	30	4	3
12	453	3	31	5	3
13	508	4	28	4	3
14	515	6	31	5	3
15	448	6	29	5	5
16	374	6	31	4	2
17	453	1	30	5	4
18	451	3	31	5	3

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
19	355	2	30	4	3
20	468	8	31	5	3
21	529	9	31	5	3
22	438	4	31	4	3
23	511	13	31	5	3
24	396	4	31	5	5
25	503	2	31	5	3
26	434	3	30	4	3
27	320	7	30	4	3
28	350	2	29	4	2
29	304	3	31	5	3
30	258	15	31	5	3
31	410	0	16	3	2
32	335	9	31	4	2
33	444	3	30	4	3
34	590	17	31	5	5
35	468	12	29	4	2
36	372	3	29	4	3
37	380	4	31	5	3
38	449	17	31	5	4
39	413	6	31	5	3
40	336	2	31	5	4
41	447	3	31	5	5
42	394	2	30	4	2
43	398	11	30	4	3
44	349	16	31	5	3
45	396	23	31	6	5
46	388	6	30	4	2
47	358	19	31	5	4
48	498	7	29	4	2
49	374	1	29	4	2
50	623	21	31	5	4
51	416	3	30	3	2

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
52	338	0	21	4	3
53	503	3	30	4	2
54	458	18	31	5	3
55	357	8	28	3	1
56	467	1	31	5	3
57	458	8	29	4	3
58	428	4	31	5	4
59	404	10	28	3	2
60	288	7	30	5	4
61	348	4	28	3	2
62	363	5	31	5	3
63	426	12	31	5	3
64	377	7	30	5	5
65	613	11	29	5	5
66	434	0	15	3	2
67	586	10	31	5	3
68	290	4	29	4	2
69	577	5	31	5	5
70	523	5	30	4	2
71	293	10	29	5	4
72	453	4	31	5	3
73	368	2	31	5	3
74	418	11	29	4	2
75	308	15	29	5	4
76	514	7	30	4	2
77	501	9	31	4	3
78	338	4	31	6	5
79	381	3	28	3	2
80	314	1	30	3	2
81	461	4	29	5	5
82	408	6	31	5	3
83	293	1	29	4	3
84	339	12	31	5	3

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
85	412	9	31	5	3
86	381	0	20	4	3
87	432	10	30	4	2
88	356	4	30	3	2
89	439	17	31	5	3
90	248	8	29	5	4
91	432	5	31	5	3
92	671	11	31	5	3
93	720	9	30	4	3
94	473	8	29	4	3
95	319	6	30	4	3
96	352	5	30	5	4
97	416	3	31	5	3
98	374	5	30	5	4
99	446	11	31	5	3
100	378	1	29	4	2
Moy :	423.6	7.0	29.7	4.5	3.1
Écart-type :	87.7	5.0	2.6	0.7	0.9

Annexe D

Résultats finaux détaillés pour la méthode d'acceptation avec seuil standard

Les résultats finaux obtenus pour chaque problème résolu avec la méthode d'acceptation avec seuil standard sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau D.1 et ceux obtenus avec l'utilisation de la diversification de premier ordre au niveau moyen sont présentés au tableau D.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la diversification de premier ordre est employée, le nombre de réinitialisations à l'aide de la DPO (*Nombre de diversifications*) est mentionné tandis que le nombre d'itérations complétées (*Nombre d'itérations*) est mentionné sinon.

Lorsque la méthode d'acceptation avec seuil standard est employée seule, $nbItMax$ est fixé à 9000 et $nbItInutileMax$ est fixé à 550. Lorsqu'une stratégie d'amélioration est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et, pour chaque application de la méthode d'acceptation avec seuil standard, $nbItMax$ est fixé à 1450 et $nbItInutileMax$ est fixé à 20.

TAB. D.1: Résultats détaillés pour la méthode d'acceptation avec seuil standard

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
1	562	29	30	6713
2	528	17	30	6751
3	514	32	29	6723
4	419	41	30	5413
5	466	26	29	6741
6	368	28	30	6725
7	450	27	30	5780
8	539	29	30	5783
9	366	18	30	6736
10	402	19	31	6966
11	368	21	29	6878
12	453	19	29	6726
13	508	12	29	6738
14	515	16	30	6733
15	448	19	30	6710
16	374	20	31	6725
17	453	23	29	6486
18	451	23	30	6709
19	355	19	29	6715
20	468	20	30	6783
21	529	24	30	5712
22	438	27	30	6708
23	511	19	30	6724
24	396	14	30	6748
25	503	16	29	6724
26	434	14	30	6721
27	320	12	30	6735
28	350	26	29	6709
29	304	28	31	6861
30	258	34	32	7130
31	410	19	29	5616
32	335	14	29	6755

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
33	444	17	30	6745
34	590	37	30	6777
35	468	22	29	6731
36	372	18	30	6740
37	380	20	29	6059
38	449	10	29	6720
39	413	9	30	7038
40	336	26	30	5752
41	447	20	29	6726
42	394	15	31	6880
43	398	20	30	6833
44	349	26	30	6710
45	396	35	30	6713
46	388	18	29	6728
47	358	28	31	6770
48	498	31	30	6708
49	374	13	30	6430
50	623	26	30	6708
51	416	11	30	6715
52	338	10	29	6766
53	503	16	29	6751
54	458	21	31	6753
55	357	14	30	6068
56	467	20	30	6138
57	458	32	29	6717
58	428	23	29	5779
59	404	24	30	6593
60	288	15	29	6718
61	348	29	30	6743
62	363	22	30	6707
63	426	27	30	5383
64	377	21	29	6803
65	613	32	29	6723

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
66	434	13	30	6847
67	586	29	30	6243
68	290	17	29	6715
69	577	17	31	6724
70	523	14	29	6718
71	293	25	29	6718
72	453	19	29	6429
73	368	28	30	6716
74	418	21	30	6717
75	308	14	32	7054
76	514	32	29	6707
77	501	24	30	6720
78	338	21	29	5296
79	381	23	31	6755
80	314	24	29	6793
81	461	12	29	6724
82	408	20	30	6712
83	293	21	30	6106
84	339	18	29	6711
85	412	25	30	5846
86	381	23	31	5604
87	432	20	29	6739
88	356	13	29	6723
89	439	25	32	7143
90	248	21	29	6773
91	432	15	31	6818
92	671	60	29	6772
93	720	21	30	5450
94	473	23	30	6706
95	319	16	30	6825
96	352	19	31	6866
97	416	19	30	6748
98	374	18	29	5968

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
99	446	24	30	6724
100	378	21	29	6519
Moy :	423.6	21.7	29.8	6574.3
Écart-type :	87.7	7.4	0.6	405.7

TAB. D.2: Résultats détaillés pour la méthode d'acceptation avec seuil standard avec la diversification de premier ordre

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
1	562	21	31	3
2	528	15	31	3
3	514	28	31	3
4	419	33	31	3
5	466	27	31	3
6	368	20	31	3
7	450	31	31	3
8	539	27	31	3
9	366	28	31	3
10	402	15	31	3
11	368	26	31	3
12	453	13	31	3
13	508	8	31	3
14	515	16	31	3
15	448	19	31	3
16	374	15	31	3
17	453	20	31	3
18	451	16	31	3

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
19	355	20	31	3
20	468	15	31	3
21	529	20	31	3
22	438	19	31	3
23	511	15	31	3
24	396	15	31	3
25	503	72	31	3
26	434	18	31	3
27	320	17	31	3
28	350	14	31	3
29	304	14	31	3
30	258	26	31	3
31	410	15	31	3
32	335	19	31	3
33	444	16	31	3
34	590	31	31	3
35	468	24	31	3
36	372	24	31	3
37	380	19	31	3
38	449	19	31	3
39	413	18	31	3
40	336	15	31	3
41	447	19	31	3
42	394	16	31	3
43	398	14	31	3
44	349	23	31	3
45	396	36	31	3
46	388	16	31	3
47	358	29	31	3
48	498	22	31	3
49	374	23	31	3
50	623	17	31	3
51	416	20	31	3

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
52	338	11	31	3
53	503	17	31	3
54	458	26	31	3
55	357	18	31	3
56	467	10	31	3
57	458	14	31	3
58	428	11	31	3
59	404	26	31	3
60	288	17	31	3
61	348	25	31	3
62	363	14	31	3
63	426	16	31	3
64	377	23	31	3
65	613	26	31	3
66	434	16	31	3
67	586	19	31	3
68	290	17	31	3
69	577	19	31	3
70	523	11	31	3
71	293	24	31	3
72	453	15	31	3
73	368	18	31	3
74	418	23	31	3
75	308	25	31	3
76	514	25	31	3
77	501	28	31	3
78	338	14	31	3
79	381	23	31	3
80	314	15	31	3
81	461	4	31	3
82	408	20	31	3
83	293	21	31	3
84	339	16	31	3

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
85	412	13	31	3
86	381	24	31	3
87	432	28	31	3
88	356	19	31	3
89	439	29	31	3
90	248	23	31	3
91	432	20	31	3
92	671	30	31	3
93	720	21	31	3
94	473	21	31	3
95	319	16	31	3
96	352	19	31	3
97	416	19	31	3
98	374	10	31	3
99	446	20	31	3
100	378	12	31	3
Moy :	423.6	20.1	31.0	3.0
Écart-type :	87.7	7.9	0.0	0.0

Annexe E

Résultats finaux détaillés pour la méthode d'acceptation avec seuil - grand déluge

Les résultats finaux obtenus pour chaque problème résolu avec la méthode d'acceptation avec seuil - grand déluge sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau E.1 et ceux obtenus avec l'utilisation de la recherche à voisinage variable imbriqué au niveau moyen sont présentés au tableau E.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la recherche à voisinage variable imbriqué est employée, le nombre de changements de structure du voisinage (*Changements de k*) et le nombre de retours à la structure de voisinage N^1 (*Retours à* N^1) sont mentionnés. Lorsqu'aucune stratégie d'amélioration n'est utilisée, le nombre d'itérations complétées (*Nombre d'itérations*) est plutôt mentionné.

Lorsque la méthode d'acceptation avec seuil - grand déluge est employée seule, $nbItMax$ est fixé à 10000 et $nbItInutileMax$ est fixé à 1975. Lorsqu'une stratégie d'amélioration est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et, pour chaque application de la méthode d'acceptation avec seuil - grand déluge, $nbItMax$ est fixé à 1500 et $nbItInutileMax$ est fixé à 400.

TAB. E.1: Résultats détaillés pour la méthode d'acceptation avec seuil - grand déluge

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
1	562	20	36	6353
2	528	17	34	5989
3	514	27	34	5685
4	419	23	30	4643
5	466	21	31	5213
6	368	12	28	4142
7	450	25	33	4972
8	539	29	34	5966
9	366	25	29	3997
10	402	18	30	4490
11	368	25	27	4004
12	453	16	31	5118
13	508	17	33	5733
14	515	21	34	5795
15	448	18	33	5041
16	374	14	28	4177
17	453	25	31	5008
18	451	12	31	5122
19	355	7	28	4069
20	468	14	33	5284
21	529	24	33	5912
22	438	22	31	4857
23	511	20	34	5731
24	396	19	30	4389
25	503	44	33	5358
26	434	22	31	4812
27	320	14	27	3569
28	350	14	28	3892
29	304	19	25	3306
30	258	28	24	2673
31	410	13	29	4637

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
32	335	11	27	3763
33	444	13	33	5043
34	590	30	36	6547
35	468	20	33	5248
36	372	26	27	4040
37	380	23	28	4170
38	449	16	31	5047
39	413	15	30	4661
40	336	25	27	3602
41	447	19	31	5002
42	394	20	30	4382
43	398	11	30	4511
44	349	22	28	3818
45	396	12	30	4490
46	388	13	29	4367
47	358	23	28	3916
48	498	22	34	5566
49	374	21	28	4118
50	623	32	38	6893
51	416	27	30	4533
52	338	13	27	3756
53	503	22	33	5636
54	458	21	33	5097
55	357	15	28	4007
56	467	10	33	5342
57	458	27	31	5020
58	428	13	30	4839
59	404	24	30	4444
60	288	18	24	3158
61	348	18	27	3849
62	363	14	29	4075
63	426	16	31	4804
64	377	24	28	4124

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
65	613	26	37	6881
66	434	8	30	4992
67	586	26	37	6507
68	290	11	25	3256
69	577	19	37	6522
70	523	28	34	5791
71	293	30	24	3076
72	453	8	32	5214
73	368	12	28	4168
74	418	17	31	4692
75	308	23	25	3327
76	514	30	34	5638
77	501	16	34	5668
78	338	22	26	3678
79	381	32	29	4081
80	314	13	26	3492
81	461	26	31	5059
82	408	18	30	4558
83	293	11	26	3247
84	339	18	27	3757
85	412	18	31	4603
86	381	18	28	4236
87	432	23	31	4751
88	356	21	27	3901
89	439	11	32	5016
90	248	16	22	2690
91	432	25	31	4748
92	671	59	37	7177
93	720	28	41	8090
94	473	29	32	5183
95	319	24	26	3454
96	352	21	29	3859
97	416	24	29	4569

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre d'itérations
98	374	20	28	4106
99	446	26	31	4891
100	378	15	28	4237
Moy :	423.6	20.3	30.3	4709.2
Écart-type :	87.7	7.5	3.4	996.1

TAB. E.2: Résultats détaillés pour la méthode d'acceptation avec seuil - grand déluge avec la recherche à voisinage variable imbriqué

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
1	562	28	31	8	4
2	528	8	31	8	5
3	514	37	30	6	3
4	419	28	31	6	3
5	466	21	31	6	2
6	368	19	31	7	3
7	450	28	31	6	3
8	539	38	31	7	4
9	366	19	31	7	4
10	402	8	31	8	5
11	368	14	31	7	3
12	453	20	31	7	3
13	508	12	30	7	4
14	515	39	31	7	4
15	448	32	31	7	4
16	374	34	30	7	4
17	453	16	30	7	5

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
18	451	15	30	7	4
19	355	22	30	7	4
20	468	24	31	7	4
21	529	13	31	7	4
22	438	15	31	7	4
23	511	13	31	6	2
24	396	15	30	7	5
25	503	35	31	8	5
26	434	14	31	6	3
27	320	14	31	7	3
28	350	14	31	7	3
29	304	32	31	8	4
30	258	28	30	8	5
31	410	10	31	8	5
32	335	20	31	7	4
33	444	31	31	8	5
34	590	28	31	7	3
35	468	20	31	7	3
36	372	17	30	7	4
37	380	13	31	9	6
38	449	21	30	6	3
39	413	5	30	6	3
40	336	10	31	7	4
41	447	5	31	7	4
42	394	17	31	7	4
43	398	13	31	7	3
44	349	24	30	7	4
45	396	21	31	6	3
46	388	8	31	7	3
47	358	27	31	6	3
48	498	7	31	6	3
49	374	8	30	7	4
50	623	22	30	7	4

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
51	416	22	30	6	3
52	338	20	31	8	5
53	503	30	31	7	3
54	458	17	31	8	5
55	357	20	31	7	4
56	467	11	31	7	3
57	458	25	30	6	3
58	428	6	31	7	3
59	404	14	31	7	3
60	288	11	31	7	4
61	348	27	30	7	4
62	363	17	31	8	5
63	426	14	31	7	4
64	377	14	31	8	5
65	613	40	31	7	4
66	434	9	31	7	4
67	586	22	31	7	3
68	290	18	30	7	4
69	577	22	31	6	3
70	523	27	31	8	4
71	293	32	31	7	3
72	453	28	31	8	5
73	368	27	30	7	4
74	418	33	30	6	3
75	308	19	31	7	4
76	514	20	31	7	4
77	501	31	31	7	3
78	338	13	31	6	2
79	381	23	31	7	4
80	314	21	31	6	2
81	461	15	31	6	2
82	408	27	31	7	4
83	293	21	31	8	4

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Changements de k	Retours à N^1
84	339	21	31	9	6
85	412	15	30	7	5
86	381	13	31	7	4
87	432	24	30	6	3
88	356	17	31	7	4
89	439	23	31	7	4
90	248	11	31	8	5
91	432	22	31	8	4
92	671	41	31	9	6
93	720	36	31	9	6
94	473	31	31	9	6
95	319	21	31	7	4
96	352	19	31	6	2
97	416	10	31	7	4
98	374	15	31	9	7
99	446	16	30	6	3
100	378	6	31	8	4
Moy :	423.6	20.2	30.8	7.1	3.9
Écart-type :	87.7	8.7	0.4	0.8	1.0

Annexe F

Résultats finaux détaillés pour la méthode d'acceptation avec seuil à détérioration maximale

Les résultats finaux obtenus pour chaque problème résolu avec la méthode d'acceptation avec seuil à détérioration maximale sans l'utilisation d'une stratégie d'amélioration sont présentés au tableau F.1 et ceux obtenus avec l'utilisation de la diversification de premier ordre au niveau intense sont présentés au tableau F.2. Pour chaque problème, nous indiquons la valeur de la solution initiale ($P(x^0)$), la valeur de la meilleure solution générée ($P(x^*)$) et le temps d'exécution utilisé (*Temps CPU (sec)*). Lorsque la diversification de premier ordre est employée, le nombre de réinitialisations de la méthode d'acceptation avec seuil à détérioration maximale à l'aide de la DPO (*Nombre de diversifications*) est mentionné tandis que le nombre d'itérations complétées (*Nombre d'itérations*) est mentionné sinon.

Lorsque la méthode d'acceptation avec seuil à détérioration maximale est employée seule, $nbItMax$ est fixé à 6000 et $nbItInutileMax$ est fixé à 200. Lorsqu'une stratégie d'amélioration est utilisée, l'exécution totale de la procédure est limitée à 30 secondes et, pour chaque application de la méthode d'acceptation avec seuil à détérioration maximale, $nbItMax$ est fixé à 600 et $nbItInutileMax$ est fixé à 500.

TAB. F.1: Résultats détaillés pour la méthode d'acceptation avec seuil à détérioration maximale

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de d'itérations
1	562	31	31	4726
2	528	18	31	5993
3	514	13	30	4767
4	419	30	23	3845
5	466	14	29	4916
6	368	21	31	2956
7	450	37	32	2978
8	539	22	30	2861
9	366	20	30	5957
10	402	22	32	5907
11	368	17	30	4709
12	453	19	29	5061
13	508	10	30	4172
14	515	14	31	4034
15	448	16	31	3983
16	374	22	32	4055
17	453	19	8	1419
18	451	31	31	917
19	355	18	29	5265
20	468	39	32	5925
21	529	17	30	3460
22	438	23	31	3521
23	511	23	30	5169
24	396	13	30	5101
25	503	55	31	5751
26	434	18	30	4694
27	320	17	30	5455
28	350	20	31	3113
29	304	15	30	2262
30	258	39	30	3548
31	410	11	30	5065

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de d'itérations
32	335	20	31	5696
33	444	23	31	4241
34	590	28	30	4853
35	468	32	32	5128
36	372	11	30	4145
37	380	21	29	3909
38	449	22	31	5671
39	413	13	29	1984
40	336	9	31	5950
41	447	20	30	2932
42	394	14	31	1883
43	398	16	30	5917
44	349	26	31	5527
45	396	25	30	3763
46	388	10	30	4524
47	358	19	31	5344
48	498	24	31	3347
49	374	21	31	4314
50	623	30	31	4496
51	416	13	30	5816
52	338	12	29	5960
53	503	20	30	4832
54	458	30	33	1284
55	357	25	31	5947
56	467	13	30	4390
57	458	26	31	5042
58	428	19	29	4843
59	404	29	31	4780
60	288	20	30	3671
61	348	15	30	5454
62	363	21	31	5843
63	426	23	30	5552
64	377	46	31	3843

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de d'itérations
65	613	27	30	5250
66	434	19	29	3404
67	586	25	30	4027
68	290	14	30	5629
69	577	27	31	3946
70	523	10	30	2751
71	293	15	31	4906
72	453	43	30	5868
73	368	22	30	2196
74	418	17	30	5170
75	308	14	30	5412
76	514	25	32	5658
77	501	21	30	3804
78	338	28	29	4634
79	381	15	31	4492
80	314	14	30	5810
81	461	13	30	5915
82	408	17	31	5241
83	293	23	30	5524
84	339	22	31	3547
85	412	26	30	4054
86	381	18	30	5270
87	432	15	30	2642
88	356	23	30	3782
89	439	22	30	5550
90	248	17	30	3267
91	432	15	31	5637
92	671	26	30	2493
93	720	16	31	4652
94	473	23	31	5439
95	319	31	30	5277
96	352	21	32	4290
97	416	15	30	5432

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de d'itérations
98	374	17	30	4491
99	446	21	31	5487
100	378	34	30	4408
Moy :	423.6	21.3	30.1	4478.2
Écart-type :	87.7	8.0	2.4	1200.1

TAB. F.2: Résultats détaillés pour la méthode d'acceptation avec seuil à détérioration maximale avec la diversification de premier ordre

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
1	562	33	30	8
2	528	15	31	9
3	514	19	31	9
4	419	24	30	8
5	466	25	31	9
6	368	23	31	9
7	450	25	31	9
8	539	20	31	9
9	366	26	31	9
10	402	8	31	9
11	368	18	31	9
12	453	14	31	9
13	508	13	31	9
14	515	9	31	9
15	448	19	31	9
16	374	25	30	8
17	453	18	31	9

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
18	451	26	31	9
19	355	18	31	9
20	468	18	31	9
21	529	15	31	9
22	438	18	31	9
23	511	17	31	9
24	396	10	31	9
25	503	61	31	9
26	434	11	31	9
27	320	17	31	9
28	350	12	31	9
29	304	13	31	9
30	258	32	31	9
31	410	10	31	9
32	335	24	31	9
33	444	18	31	9
34	590	31	31	9
35	468	31	31	9
36	372	11	31	9
37	380	21	30	9
38	449	28	30	8
39	413	10	31	9
40	336	11	31	9
41	447	27	31	9
42	394	16	31	9
43	398	14	31	9
44	349	29	31	9
45	396	26	31	9
46	388	13	31	9
47	358	29	30	8
48	498	11	31	9
49	374	21	31	9
50	623	15	31	9

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
51	416	11	31	9
52	338	9	31	9
53	503	7	31	9
54	458	35	31	9
55	357	16	31	9
56	467	22	31	9
57	458	27	31	9
58	428	17	31	9
59	404	29	31	9
60	288	18	30	9
61	348	15	31	9
62	363	25	31	9
63	426	15	31	9
64	377	35	31	9
65	613	32	31	9
66	434	27	31	9
67	586	30	31	9
68	290	14	30	9
69	577	31	30	8
70	523	9	31	9
71	293	19	31	9
72	453	33	31	9
73	368	17	31	9
74	418	24	31	9
75	308	22	31	9
76	514	24	31	9
77	501	21	31	9
78	338	32	31	9
79	381	19	31	9
80	314	11	31	9
81	461	10	31	9
82	408	17	31	9
83	293	25	31	9

... suite à la page suivante

... suite de la page précédente

Problème	$P(x^0)$	$P(x^*)$	Temps CPU (sec)	Nombre de diversifications
84	339	14	31	9
85	412	20	31	9
86	381	15	31	9
87	432	13	31	9
88	356	17	31	9
89	439	24	31	9
90	248	20	31	9
91	432	23	31	9
92	671	23	31	9
93	720	33	30	8
94	473	24	30	8
95	319	24	31	9
96	352	16	30	8
97	416	10	31	9
98	374	24	31	9
99	446	26	31	9
100	378	21	31	9
Moy :	423.6	20.4	30.9	8.9
Écart-type :	87.7	8.3	0.3	0.3