Chapitre 6

Ordonnancement d'atelier au plus tôt

Ce chapitre présente une approche en ordonnancement sous contraintes pour la résolution exacte du problème d'open-shop ($O//C_{max}$) basée sur les toutes dernières techniques de propagation et de recherche combinées à une borne supérieure initiale obtenue de manière heuristique. Les techniques de randomisation et redémarrage combinées à l'enregistrement de nogoods favorisent la diversification de la recherche tout en évitant les redondances d'un redémarrage à l'autre. Cette approche domine les meilleures métaheuristiques et méthodes exactes sur de nombreuses d'instances.

Sommaire

Ce chapitre présente nos travaux [3, 4] sur les problèmes d'atelier définis au chapitre 4. Dans un problème d'atelier, une pièce doit être usinée ou assemblée sur différentes machines. Chaque machine est une ressource disjonctive, c'est-à-dire qu'elle ne peut exécuter qu'une tâche à la fois. Plus précisément, les tâches sont regroupées en n entités appelées travaux, lots ou jobs. Chaque lot est constitué de m tâches à exécuter sur m machines distinctes. Nous considérerons que le séquencement des tâches appartenant aux lots n'est pas imposé (open-shop). Le critère d'optimalité étudié est la minimisation de le délai total de l'ordonnancement ou date d'achèvement maximale (*makespan*). Notre approche reste valide pour d'autres problèmes d'atelier incluant notamment le job-shop et le flow-shop. Nous supposons que les durées d'exécution des tâches sont données par une matrice entière $P : m \times n$, dans laquelle $p_{ij} \geq 0$ est la durée d'exécution de la tâche $T_{ij} \in T$ du lot J_j réalisée sur la machine M_i.

Nous proposons plusieurs améliorations d'un algorithme top-down dédié aux problèmes d'open-shop qui a l'avantage, contrairement aux approches de type bottom-up, de fournir rapidement de bonnes solutions (voir section 2.4). Notre modèle repose sur des techniques d'inférence forte pour les contraintes disjonctives et temporelles ainsi que des raisonnements dédiés à la minimisation du délai total. Notre

contribution principale est de montrer que la combinaison de techniques de randomisation et redémarrage, et de méthodes d'inférence et de branchement dédiées à l'ordonnancement est une alternative efficace pour la résolution du problème d'open-shop. Les performances de notre algorithme de résolution surpassent celles des approches de la littérature sur un large spectre d'instances.
La section 6.1 introduit notre modèle en ordonnancement sous contraintes et la section 6.2 décrit les techniques améliorant l'algorithme top-down. Finalement, la section 6.3 présente nos résultats expérimentaux et étudie l'influence des différentes composantes de l'algorithme avant de le comparer aux autres approches.

6.1 Modèle en ordonnancement sous contraintes

Dans cette section, nous présentons notre modèle en ordonnancement sous contraintes pour le problème $O//C_{max}$. Nous précisons les règles et algorithmes choisis pour le filtrage de la contrainte disjonctive représentant les machines et les lots. Puis, nous expliquons la gestion du filtrage des contraintes temporelles et du branchement dans le modèle disjonctif. Nous présentons deux contraintes globales supplémentaires pour la gestion des intervalles interdits (*forbidden intervals*) et l'élimination d'une symétrie (*symmetry breaking*). Pour finir, nous détaillerons plusieurs stratégies de branchement en ordonnancement sous contraintes et justifions notre choix de branchement.

6.1.1 Contraintes disjonctives

La contrainte disjonctive présentée en section 3.3.1 représente une ressource de capacité unitaire. La contrainte est satisfaite si aucune paire de tâches de durée non nulle ne se chevauche. Une contrainte disjonctive est postée pour chaque lot et chaque machine.

Nous avons choisi l'implémentation proposée par Vilím [43] des règles *not first/not last*, *detectable precedence* et *edge finding*. Nous tirons avantage du calcul de la date d'achèvement minimale ECT_Ω (*earliest completion time*) des tâches d'un ensemble Ω pour estimer une borne inférieure sur le délai total de l'ordonnancement. En fait, la date d'achèvement minimale d'une machine M_i (respectivement un lot J_j) est égale à la valeur ECT_{M_i} (respectivement ECT_{J_j}). Par conséquent, le délai total de l'ordonnancement est supérieur aux dates d'achèvement minimales des ressources (lots et machines).

6.1.2 Contraintes temporelles

Nous rappelons que le graphe de précédence $\mathcal{G} = (\mathcal{T}, \mathcal{P})$ (voir section 3.4) est associé à un sous-réseau de contraintes où ses sommets représentent les tâches et les arcs représentent les contraintes de précédence. Deux tâches fictives T_{start} et T_{end} représentant le début et la fin de l'ordonnancement sont ajoutées. Un arc entre deux tâches T_{ij} et T_{kl} est ajouté à $\mathcal{P}$ si T_{ij} précède T_{kl} ($T_{ij} \preceq T_{kl}$). Les arcs initiaux de $\mathcal{P}$ sont tous ceux d'origine T_{start} ou de destination T_{end}. Il suffit d'ajouter les arcs correspondant aux séquences d'opérations dans des lots et de supprimer les contraintes disjonctives sur les lots pour modéliser les problèmes de job-shop et flow-shop.

La stratégie de branchement que nous allons utiliser ajoute des arcs entre les paires de tâches partageant une ressource (lot ou machine) jusqu'à ce que toutes ces paires soient connectées par un chemin de $\mathcal{G}$. Il s'agit donc d'un arbitrage partiel du graphe disjonctif qui induit un arbitrage complet par transitivité. À la fin de la recherche, le délai total de l'ordonnancement C_{max} est égal à la longueur du plus long chemin entre T_{start} et T_{end}, c'est-à-dire un chemin critique. La stratégie de branchement exploite la fermeture transitive de $\mathcal{G}$ pour éviter de créer des cycles ou de considérer les précédences transitives ou obligatoires. En effet, le graphe de précédence $\mathcal{G}$ est consistent si et seulement s'il ne contient aucun cycle.Une précédence est facilement détectée en raisonnant sur ces fenêtres de temps (règles précédences interdites et obligatoires), mais cela n'est pas nécessairement le cas lorsque la précédence est impliquée par transitivité. Les précédences respectent l'inégalité triangulaire. Par conséquent, si un arc (T_{ij}, T_{kl}) est transitif, c'est-à-dire que T_{ij} et T_{kl} sont reliés par un chemin dans $\mathcal{P}\backslash\{(T_{ij}, T_{kl})\}$, alors la précédence $T_{ij} \preceq T_{kl}$ est déduite.

Nous utilisons une contrainte globale modélisant le graphe de précédence introduit en section 3.4 pour le filtrage et le branchement. Bien sûr, les contraintes temporelles peuvent être gérées par l'ajout

des contraintes arithmétiques élémentaires associées. Mais, il existe des procédures efficaces dédiées à un problème voisin appelé *simple temporal problem* [36]. Ce problème considère un ensemble de variables temporelles entières $\{X_1, \ldots, X_n\}$ et un ensemble de contraintes $\{a_{ij} \leq X_j - X_i \leq b_{ij}\}$ où $b_{ij} \geq a_{ij} \geq 0$. Cesta et Oddi [156] ont proposé plusieurs algorithmes pour la gestion des informations temporelles permettant de (a) gérer dynamiquement l'ajout et le retrait de contraintes dans le réseau, et (b) exploiter la structure du réseau pour la propagation incrémentale et la détection des cycles.
La contrainte globale gère efficacement l'ajout et le retrait d'arcs dans le graphe de précédence. Son état est restauré lors d'un retour arrière, c'est-à-dire qu'elle maintient une pile représentant des changements du graphe. Ainsi, cette contrainte ne réalise aucune hypothèse sur l'ensemble des disjonctions à arbitrer, c'est-à-dire l'ensemble des arêtes du graphe disjonctif. Lors de l'ajout d'un arc, il est possible de détecter en temps constant sa transitivité ou la création d'un cycle en maintenant la fermeture transitive de $\mathcal{G}$. Frigioni *et al.* [157] ont proposé un algorithme pour maintenir la fermeture transitive d'un graphe orienté avec une complexité $O(n)$ pour une série d'insertions et de suppressions d'arcs. En outre, nous maintenons un ordre topologique de $\mathcal{G}$ grâce à l'algorithme simple et efficace proposé par Pearce et Kelly [158]. Notez que la connaissance de la fermeture transitive diminue la complexité totale pour maintenir l'ordre topologique.
La propagation des précédences est réalisée en temps linéaire par la contrainte globale alors que la propagation individuelle des précédences par le solveur peut atteindre le point fixe en temps quadratique si l'ordre de réveil des contraintes est malheureux. Dans notre cas, les plus longs chemins entre T_{start} et T_{ij}, et entre T_{ij} et T_{end} sont calculés pour mettre à jour la fenêtre de temps de la tâche T_{ij}. Puisque $\mathcal{G}$ est un graphe orienté acyclique, tous les plus longs chemins issus de T_{start} et terminant à T_{end} sont calculés en temps linéaire par une version incrémentale de l'algorithme *dynamic bellman algorithm for the single source longest path problem* [37]. Notre implémentation maintient dynamiquement un ordre topologique qui est une entrée de l'algorithme pour éviter des calculs redondants. À chaque appel, notre contrainte globale ne considère qu'un sous-graphe induit par les tâches dont les fenêtres de temps ont changé depuis le dernier appel. Ainsi, la complexité totale est réduite à $O(|\mathcal{P}|)$ par rapport à l'algorithme général en $O(|\mathcal{T}| \times |\mathcal{P}|)$ proposé par Cesta et Oddi [156].

6.1.3 Contraintes supplémentaires

Nous introduisons des contraintes supplémentaires qui améliorent la réduction des domaines en considérant le critère d'optimalité et une symétrie basique. Ces contraintes redondantes basées sur des règles de dominance ne sont pas nécessaires à la correction du modèle mais améliorent sa résolution. Elles peuvent être propagées en temps constant durant la recherche contrairement à d'autres bornes inférieures complexes ou règles de dominance appliquées par d'autres stratégies de recherche [159], comme la stratégie de branchement de Brucker.

6.1.3.1 Intervalles interdits

Les intervalles interdits sont des techniques de filtrage exclusivement dédiées à la minimisation du délai total de l'ordonnancement dans un atelier à cheminement libre. Les intervalles interdits sont des intervalles temporels dans lesquels aucune tâche ne peut débuter ou s'achever dans une solution optimale. Les fenêtres de temps des tâches sont ajustées pendant la recherche à partir de ces informations. Quand la date de début au plus tôt d'une tâche appartient à un intervalle interdit, elle peut être augmentée jusqu'à la borne supérieure de cet intervalle. Guéret et Prins [120] déterminent les intervalles interdits grâce à la résolution de $m + n$ problèmes *subset sum* (voir section 3.5). Les problèmes sont résolus avec une complexité $O(d \times n)$ [voir 63] où d représente, dans notre cas, le délai total de l'ordonnancement. Les fenêtres de temps sont ajustées en temps constant puisque ces problèmes sont résolus une fois pour toutes avant la résolution.

6.1.3.2 Élimination d'une symétrie

De nombreux problèmes de satisfaction de contraintes contiennent des symétries qui définissent des classes d'équivalence pour de nombreuses solutions. Les techniques d'élimination des symétries réduisent la découverte de solutions équivalentes en évitant de visiter les affectations partielles symétriques de

celles prouvées inconsistantes ou dominées. Une solution du problème $O//C_{max}$ peut être inversée en considérant la dernière tâche comme la première, la seconde comme l'avant-dernière et ainsi de suite. Ainsi, une solution obtenue par réflexion a la même délai total que la solution originale. Par conséquent, il n'est pas nécessaire de vérifier l'ordre inverse lorsque l'algorithme a prouvé qu'un ordre était sous-optimal. On peut briser cette symétrie par réflexion en choisissant une tâche quelconque T_{ij} et en imposant qu'elle débute dans la première moitié de l'ordonnancement : $s_{ij} \leq \left\lceil \frac{e_{\text{end}} - p_{ij}}{2} \right\rceil$. Notre algorithme sélectionne la première tâche dont la durée d'execution est maximale en suivant l'ordre lexicographique.

6.1.4 Stratégie de branchement

Il existe deux grandes catégories de stratégies de branchement en ordonnancement sous contraintes : l'affectation des dates de début aux tâches et l'arbitrage des disjonctions. La première catégorie mène à un branchement n-aire alors que la seconde engendre généralement des décisions binaires.

6.1.4.1 Affectation des dates de début

Parmi les branchements de la première catégorie, la stratégie classique *dom-minVal* (voir section 2.3) est généralement vouée à l'échec dans le cas d'optimisation d'un critère régulier en ordonnancement sous contraintes. En effet, l'ensemble des ordonnancements actifs (aucune tâche ne peut terminer plus tôt sans en retarder au moins une autre) est dominant, c'est-à-dire qu'il contient au moins une solution optimale. Une stratégie bien connue appelée *setTimes* [160] est incomplète dans le cas général, mais complète pour les critères réguliers. À chaque nœud, elle choisit une tâche parmi l'ensemble des tâches non ordonnancées et disponibles puis l'ordonnance au plus tôt. Lors d'un retour arrière, la stratégie marque la tâche ordonnancée au dernier point de choix comme indisponible jusqu'à ce que sa date de début au plus tôt change (par propagation).

6.1.4.2 Arbitrage des disjonctions

La seconde catégorie arbitre les disjonctions, c'est-à-dire qu'elle impose des précédences entre certaines paires de tâches. Le branchement n-aire proposé par Brucker *et al.* [104], noté *block*, est basé sur le calcul d'une solution heuristique (compatible avec l'arbitrage partiel courant) à chaque nœud pour déterminer quelles précédences ajouter. La stratégie exploite un théorème de dominance pour remettre en cause les précédences du chemin critique de cette solution heuristique en imposant dans chaque branche plusieurs précédences entre les tâches de ce chemin critique. Cette stratégie fixe simultanément de nombreuses précédences tout en restant complète.

Beck *et al.* [161] ont proposé un branchement binaire noté *profile* dans lequel une disjonction entre deux tâches critiques partageant la même ressource disjonctive est arbitrée à chaque nœud. Pour ce faire, il définit la demande individuelle comme la quantité (probabiliste) de ressource nécessaire à une activité à chaque instant t. Pour mesurer l'occupation d'une ressource, on additionne les courbes des demandes individuelles de ses tâches. À chaque nœud, on détermine le pic de demande sur l'ensemble des ressources. On identifie alors la paire de tâches qui participe le plus à la demande de la ressource à l'instant du pic de demande (ressource – instant) et l'on s'assure que la disjonction associée n'est pas arbitrée. L'ordre relatif des tâches doit alors être déterminé pour la disjonction choisie. Cette étape est réalisée par l'heuristique randomisée de sélection d'arbitrage appelée *centroid* [161]. Le centroid est une fonction déterministe du domaine vers les réels qui est calculée pour les deux tâches critiques. Le centroid est l'instant qui divise la demande individuelle en deux parties d'aires égales. On impose d'abord la précédence qui préserve l'ordre des centroids des deux tâches. Si les centroids sont placés au même instant, un ordre aléatoire est choisi. Plus récemment, Laborie [122] a proposé un branchement générique pour l'ordonnancement cumulatif basé sur la notion d'ensemble critique minimal (*minimal critical set* – MCS). En ordonnancement disjonctif, les MCSs sont simplement des paires d'activités partageant une ressource disjonctive. À chaque nœud, la stratégie consiste à (a) choisir un MCS en fonction d'une estimation de la réduction de l'espace de recherche engendré par sa résolution (b) appliquer une procédure de simplification sur le MCS choisi et (c) brancher sur les différentes précédences possibles jusqu'à ce qu'il ne reste plus aucun MCS.

Les exemples donnés en figure 6.1 illustrent les différentes formes de l'arbre de recherche en fonction de la stratégie de branchement. Dans ce chapitre, nous utiliserons les heuristiques de sélection *profile* et

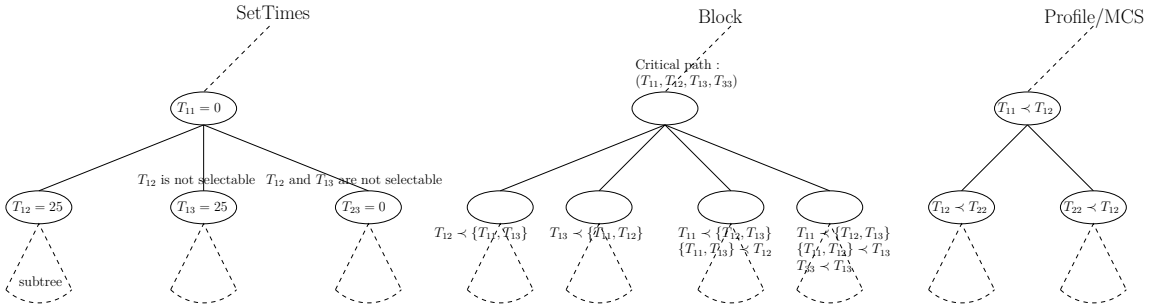

FIGURE 6.1 – La forme de l'arbre de recherche pour différentes stratégies de branchement. De gauche à droite, les stratégies *setTimes*, *block* et *profile*/MCS.

centroid dont la randomisation des points de choix binaires facilite l'application des techniques introduites en section 6.2.

6.2 Algorithme de résolution

En utilisant le modèle présenté en section 6.1, on peut maintenant résoudre le problème d'open-shop par la procédure d'optimisation top-down (voir section 2.4). Cette section décrit les améliorations apportées à l'algorithme de recherche par l'utilisation d'heuristiques et de techniques de redémarrage. La stratégie de branchement consiste à ajouter des contraintes de précédences en chaque nœud grâce à l'heuristique de sélection de disjonction *profile* et à l'heuristique randomisée de sélection d'arbitrage *centroid*. Des expérimentations préliminaires ont révélé deux principaux inconvénients à cette approche. Premièrement, les techniques de filtrage sont très efficaces dès qu'une borne supérieure précise est connue, mais ne font que ralentir la recherche dans le cas contraire. Ensuite, la légère randomisation de *centroid* provoque des variations importantes du temps de résolution, mais aussi de la qualité des premières solutions découvertes. Un tel comportement laisse supposer que certaines des premières décisions prises dans l'arbre de recherche ne sont jamais remises en question et provoque un phénomène de *thrashing* important (le même échec est découvert plusieurs fois).

Nous proposons une heuristique constructive randomisée (sans propagation) pour initialiser la borne supérieure initiale afin d'améliorer la sélection et la propagation des premières décisions de l'arbre de recherche. De plus, nous appliquons une stratégie de redémarrage dans l'espoir de diversifier la recherche et nous enregistrons des *nogoods* pour éviter d'explorer la même partie de l'espace de recherche d'un redémarrage à l'autre. Ces mécanismes peuvent être intégrés dans n'importe quel algorithme top-down et n'importe quelle stratégie de branchement même si l'enregistrement des *nogoods* est simplifié dans le cas d'un branchement binaire. Nous donnons maintenant une présentation détaillée de notre approche et discutons ses paramètres.

6.2.1 Principe de l'algorithme

La figure 6.2 récapitule notre algorithme générique pour la résolution des problèmes d'atelier : *Randomized and Restart Constraint Programming algorithm* (`RRCP`). L'algorithme de type top-down débute par le calcul d'une solution heuristique (blocs 1 et 2) et continue, si nécessaire, par une recherche complète (blocs 3 à 9) qui améliore itérativement la meilleure solution découverte jusqu'à présent.
L'heuristique calcule une solution initiale qui fournit la borne supérieure initiale C_{max}^{UB}. Nous avons élaboré une heuristique randomisée simple appelée `CROSH` présentée en section 6.2.2. Puis, un test d'optimalité (bloc 2) détermine s'il est nécessaire de démarrer la recherche complète. Dans le bloc 3, le modèle PPC défini en section 6.1 et divers composants du solveur sont initialisés.

La boucle principale de l'algorithme est exécutée entre les blocs 4 et 9. Après la réduction des domaines par propagation (bloc 4), l'algorithme a atteint une solution, une contradiction, ou ni l'une ni l'autre. Si une solution est découverte, elle est enregistrée et la nouvelle borne supérieure génère une coupe dynamique (bloc 5) qui est propagée pendant les retours en arrière (bloc 6). Si une contradiction est

levée et que toutes les branches du nœud racine ont été visitées, alors l'optimalité de la dernière solution découverte est prouvée entraînant l'arrêt de l'algorithme. Dans le cas contraire, l'algorithme effectue un retour arrière (bloc 6). À ce moment, la création d'un nouveau point de choix est nécessaire, mais nous examinons d'abord la possibilité de redémarrer. Nous avons retenu deux politiques de redémarrage (Luby et Walsh en bloc 7) qui sont présentées en section 6.2.3. Avant un redémarrage, nous enregistrons des *nogoods* (bloc 8) issus du chemin vers la dernière branche de l'arbre pour éviter une exploration redondante d'un redémarrage à l'autre et garder ainsi la trace des sous-problèmes prouvés sous-optimaux ou inconsistants. Un *nogood* est défini ici par la borne supérieure courante et l'ensemble des décisions de branchement (précédences). L'algorithme ne redémarre jamais avant un retour arrière pour éviter des effets de bord tels que l'enregistrement de *nogoods* incomplets ou un redémarrage après l'obtention de la preuve d'optimalité (par exemple à la racine de l'arbre). Lorsque l'algorithme ne redémarre pas, la recherche arborescente continue par la création d'un nouveau point de choix (bloc 9) en suivant la stratégie de branchement *profile*. La stratégie de branchement divise le problème principal en deux sous-problèmes disjoints en ajoutant temporairement une contrainte de précédence (branche gauche) puis son opposé (branche droite).
Notre algorithme contient donc deux paramètres : le nombre d'itération de l'heuristique construisant une solution initiale (bloc 1) ; la stratégie de redémarrage (bloc 7). Leurs valeurs sont discutées en section 6.3.1.

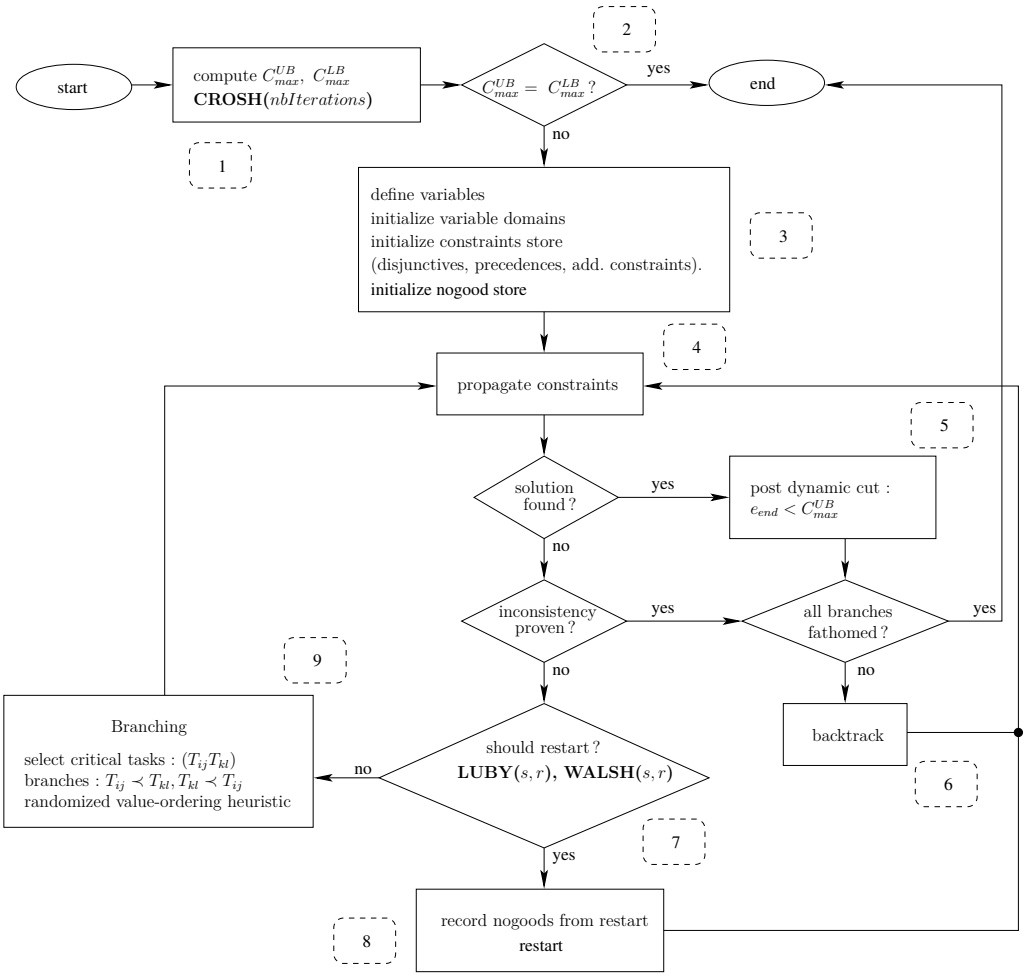

FIGURE 6.2 – Structure de RRCP. Les ellipses sont les états finaux et initiaux. Les rectangles sont des procédures ou actions. Les losanges sont des conditions *if-else*. Les rectangles en pointillé sont des étiquettes.

6.2.2 Solution initiale

Le filtrage avec inférence forte est coûteux, mais devient utile seulement lorsque les fenêtres de temps des tâches ne sont pas trop relâchées. En l'absence de contraintes de disponibilité et d'échéance, les fenêtres de temps dépendent fortement de la borne supérieure sur le délai total. De la même manière, la stratégie de branchement est très sensible aux fenêtres de temps puisque la demande individuelle et le *centroid* des tâches en dépendent. Par conséquent, il est important d'obtenir rapidement une borne supérieure de qualité sur le délai total de l'ordonnancement.

Nous avons élaboré une heuristique constructive randomisée pour le problème d'open-shop (*Constructive Randomized Open-Shop Heuristics* – `CROSH`) en combinant des règles de priorité pour les heuristiques de liste (voir section 4.3). En plus d'être une heuristique générique et simple à implémenter, les performances de `CROSH` se sont avérées tout à fait satisfaisantes (voir section 6.3.2). La première itération ordonnance les opérations en suivant la règle de priorité LPT dans laquelle les opérations sont triées par durées décroissantes. Les itérations suivantes traitent les opérations dans un ordre aléatoire sous une distribution uniforme. Le seul paramètre est le nombre d'itérations. La complexité d'une itération est égale à $O(m^2 \times n^2)$.

6.2.3 Techniques de redémarrage

Les politiques de redémarrage sont basées sur l'observation suivante : plus une recherche arborescente avec retour arrière passe de temps sans trouver de solution, plus il est probable qu'elle explore une partie stérile de l'espace de recherche. Les choix initiaux effectués par le branchement sont les moins informés et les plus importants puisqu'ils mènent aux plus grands sous-arbres et que l'exploration peut rarement remettre en cause les erreurs précoces. Cela peut entraîner des situations de *thrashing* où des échecs dus à un ensemble restreint de choix initiaux sont redécouverts plus profondément dans les sous-arbres.
Pour contrer cet inconvénient, les techniques de *shaving* et d'*intelligent backtracking* (voir section 2.2) ont été étudiées pour la résolution de problèmes d'atelier [119, 121, 122]. Le *shaving* essaie d'affecter une valeur à une variable et applique un algorithme de consistance. Si une inconsistance est détectée, la valeur peut être supprimée du domaine de la variable. Les algorithmes avec retour arrière intelligent essaient de compenser les erreurs initiales du branchement en analysant les échecs et identifiant les décisions responsables de l'échec courant.
Nous avons choisi d'incorporer dans notre approche des stratégies de redémarrage combinées à une heuristique d'arbitrage randomisée pour réduire le *thrashing* et remettre en cause les mauvais choix initiaux. Cette approche diversifie la recherche, nécessite moins de calculs en chaque nœud, mais explore plus de nœuds que le *shaving* et l'*intelligent backtracking*. Des expérimentations préliminaires sur le *shaving* ont montré qu'il n'était pas utile dans notre implémentation. Nous n'avons pas testé d'algorithme avec retour arrière intelligent parce qu'il requiert un mécanisme d'explication de tous les changements des domaines ainsi qu'une intégration profonde dans l'algorithme de recherche.
Nous rappelons d'abord la définition d'une politique universelle de redémarrage. Ensuite, nous décrivons le mécanisme d'enregistrement des *nogoods* qui évitent l'exploration de la même partie de l'espace de recherche d'un redémarrage à l'autre.

6.2.3.1 Politique universelle de redémarrage

Soit $A(x)$ un algorithme randomisé de type *Las Vegas*, c'est-à-dire que pour n'importe quelle entrée x, le résultat de A est toujours correct, mais son temps d'exécution $T_A(x)$ est une variable aléatoire. Une stratégie universelle de redémarrage (*universal restart strategy*) détermine la limite des longueurs de toutes les exécutions de l'algorithme pour n'importe quelle distribution des temps d'exécution.

Si la seule observation réalisable est la longueur d'une exécution et si l'on ne dispose d'aucune information sur la distribution des temps d'exécution $T_A(x)$, Luby *et al.* [162] ont montré qu'une planification universelle des longueurs limites de la forme $(1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots)$ donne un temps espéré de résolution qui est à un facteur logarithmique de celui donné par la meilleure limite fixe. De plus, le facteur d'amélioration des performances dû à une autre stratégie universelle est au mieux constant. Nous considérons deux paramètres qui sont le facteur d'échelle s (*scale factor*) et le facteur géométrique r (*geometric factor*). Le facteur d'échelle est la limite de base d'une stratégie de redémarrage. Notons

$\lambda_k = \frac{r^k-1}{r-1}$, le i-ème terme de la séquence est ($s = 1$ et $r = 2$ dans l'exemple précédent) :

$$\forall i > 0, \quad t_i = \begin{cases} sr^{k-1} & \text{si } i = \lambda_k \\ t_{i-\lambda_{k-1}} & \text{si } \lambda_{k-1} < i < \lambda_k \end{cases}$$
$$s = 2 \text{ and } r = 3 \Rightarrow \quad 2, 2, 2, 6, 2, 2, 2, 6, 2, 2, 2, 6, 18, \ldots$$

Walsh [163] a suggéré une autre stratégie universelle de la forme $s, sr, sr^2, sr^3, \ldots$ où l'accroissement des limites est exponentiel contrairement à l'accroissement linéaire de la stratégie de Luby.

Wu et van Beek [164] ont démontré analytiquement et empiriquement les lacunes liées à l'emploi de stratégies qui ne sont pas universelles. Ils ont aussi montré que le paramétrage des stratégies améliore les performances tout en conservant les garanties d'optimalité et de pire cas. Les redémarrages sont une composante clé de notre approche, nous évaluerons donc l'influence des facteurs d'échelle et géométrique pour identifier une bonne stratégie de redémarrage.

6.2.3.2 Enregistrement des nogoods avant les redémarrages

La sélection d'une disjonction est déterministe, mais peut varier d'un redémarrage à l'autre en fonction de l'évolution de la borne supérieure et des décisions d'arbitrage. En effet, la sélection de l'arbitrage est randomisée, mais seulement lorsqu'on ne peut pas identifier un ordre jugé prometteur grâce à *centroid*. Nous avons déjà évoqué les variations importantes de la résolution provoquées par cette légère randomisation. Quelquefois, au contraire, peu d'arbitrages aléatoires sont rendus et la même partie de l'arbre de recherche est souvent visitée d'un redémarrage à l'autre. Nous appliquons donc une technique d'enregistrement des *nogoods* avant les redémarrages similaire à celle de Lecoutre *et al.* [165] pour pallier cet inconvénient.

Dans notre cas, un *nogood* est défini pour la borne supérieure courante *ub* et correspond à un ensemble de précédences P, tel que toute solution associée à un arbitrage complet contenant P a un délai total supérieur à *ub*. Le même ensemble P de précédences peut être redécouvert d'un redémarrage à l'autre. L'enregistrement de P permet d'éviter une exploration redondante et de provoquer une meilleure diversification au gré des redémarrages. L'enregistrement des *nogoods* est peu intrusif puisqu'il a lieu juste avant les redémarrages (bloc 8 de la figure 6.2). À ce moment, on enregistre tous les *nogoods* représentant les sous-arbres prouvés sous-optimaux en suivant l'idée de Lecoutre *et al.*. Par conséquent, l'exploration accomplie durant cette étape est totalement mémorisée et cette partie de l'arbre de recherche ne sera plus visitée lors des étapes suivantes. Un nombre linéaire de *nogoods*, au regard du nombre de disjonctions, est enregistré à chaque redémarrage puisqu'un *nogood* est extrait de chaque décision négative (branche droite) du chemin allant vers la dernière branche de l'arbre de recherche binaire.

Les *nogoods* sont propagés individuellement dans Lecoutre *et al.* grâce aux techniques de *watch literals*. Nous avons implémenté une contrainte globale qui assure la propagation unitaire des *nogoods*. La propagation unitaire consiste à simplifer les clauses lorsqu'un littéral est instancié : chaque clause le contenant est supprimée ; la négation du littéral est supprimée de chaque clause la contenant. Notre implémentation reste naïve et pourrait être améliorée par les techniques de *watch literals* [166]. En pratique, le nombre de *nogoods* est relativement bas puisqu'ils ne sont enregistrés qu'avant les redémarrages et leur propagation ne revêt pas un aspect critique dans notre approche. De plus, nous supprimons les *nogoods* qui sont subsumés par les nouveaux venus lors de leur enregistrement avant un redémarrage.

6.3 Évaluations expérimentales

Trois jeux d'instances pour $O//C_{max}$ sont disponibles dans la littérature. Le premier est constitué de 60 problèmes proposés par Taillard [103] comprenant entre 16 tâches (4 lots et 4 machines) et 400 tâches (20 lots et 20 machines). Il est considéré comme facile puisque la preuve d'optimalité est triviale, c'est-à-dire que la date d'achèvement maximale C_{max} est égale à la borne inférieure C^{LB}_{max}. Brucker *et al.* [104] ont proposé 52 instances difficiles allant de 3 lots et 3 machines à 8 lots et 8 machines. Finalement, le dernier jeu d'instances est constitué de 80 instances proposées par Guéret et Prins [105]. Leurs tailles varient de 3 lots et 3 machines jusqu'à 10 lots et 10 machines et le temps d'achèvement maximal est

toujours strictement supérieur à la borne inférieure. Notons que la borne inférieure C_{max}^{LB} est toujours égale à 1000 pour les instances de Brucker et Guéret-Prins.

Nous avons réalisé différentes expérimentations dans le but de (a) configurer les paramètres (section 6.3.1) (b) étudier l'impact des différentes composantes de l'algorithme (section 6.3.2) et (c) comparer `RRCP` aux meilleures approches de la littérature (section 6.3.3). Nous avons élaboré deux jeux d'expérimentations indépendants pour réaliser l'étape (a) en un temps raisonnable. En utilisant les meilleurs paramètres identifiés à l'étape (a), les résultats du principal jeu d'expérimentations servent de support aux étapes (b) et (c). Dans ce jeu principal, plusieurs variantes de l'algorithme sont appliquées sur toutes les instances. Ces variantes déterminent la solution initiale avec `CROSH` ou LPT et utilisent une recherche arborescente avec, ou sans, stratégie de redémarrage (Luby/Walsh) qui enregistre, ou pas, des *nogoods*. L'algorithme étant randomisé, chaque instance est résolue 20 fois sans limite de temps. Les temps de résolution prennent en compte ceux de l'heuristique fournissant la solution initiale. Notre implémentation est basée sur le solveur Choco (Java), notamment son module d'ordonnancement (tâches, contraintes de partage de ressource et temporelles, branchements), et celui pour les redémarrages avec enregistrement des *nogoods*. Étant donné que ces fonctionnalités ont été intégrées dans les livraisons récentes de `choco` ($\geq$ 2.0.0), notre algorithme peut être facilement reproduit (voir chapitre 9). Un module additionnel fournit les heuristiques, construit le modèle et configure le solveur.
Toutes les expérimentations ont été menées sur une grille de calcul composée de machines Linux, dans laquelle chaque nœud possède 1 GB de mémoire vive et un processeur AMD cadencé à 2.2 GHz.

6.3.1 Réglage des paramètres

Les paramètres de notre algorithme `RRCP` sont présentés en section 6.2. Une étude expérimentale justifie ici le choix des paramètres dans le réglage final.

6.3.1.1 Solution initiale

Dans cette section, nous discutons comment fixer le nombre d'itérations de `CROSH` signalé en bloc 1 de la figure 6.2. Ce jeu d'expérimentations tente de trouver un bon compromis entre le temps passé à calculer la solution heuristique et la qualité de celle-ci. Idéalement, on souhaite arrêter l'heuristique dès que la recherche complète peut améliorer cette solution plus rapidement ou prouver son optimalité.

Par conséquent, nous avons discrétisé le nombre d'itérations selon l'ordre de grandeur 1, 10, 100, 1000, 5000, 10000, 25000. Le nombre maximum d'itérations est limité à 25000 parce que les temps d'exécution de `CROSH` dépassent une limite globale de 30 secondes pour les plus grandes instances ($15 \times 15,\ 20 \times 20$). Toutes les instances sont résolues avec une variante de l'algorithme sans redémarrage et dont la solution initiale est déterminée par `CROSH`. Chaque instance est résolue 20 fois avec une limite de temps de 180 secondes.

Nous avons déduit le nombre d'itérations pour chaque taille de problème à partir du pourcentage d'instances résolues optimalement et du temps moyen de résolution d'une instance. Le nombre d'itérations des instances dont la taille est inférieure à 6×6 est fixé à 1000 puisque celles-ci sont résolues facilement par la recherche complète. Ensuite, le nombre d'itérations est fixé à 10000 jusqu'à la taille 9×9 et à 25000 autrement. En complément, une limite de temps globale (indépendante de la taille des instances) est fixée à 20 secondes.

6.3.1.2 Paramètres des politiques de redémarrage

Cette section traite du réglage des paramètres des stratégies de redémarrage signalés au bloc 7 de la figure 6.2 (facteurs d'échelle et géométrique). Nous avons sélectionné un ensemble de 23 instances dont la taille est comprise entre 6×6 et 20×20 (8 GP*, 9 j*, 6 tai*) et dont les distributions des temps d'exécution sont différentes pour identifier les bonnes valeurs des paramètres. Nous mesurons l'influence des paramètres sur l'efficacité des stratégies de redémarrage en considérant le nombre de problèmes résolus optimalement dans une certaine limite de temps comme proposé par Wu et van Beek [164]. Le facteur d'échelle s est discrétisé en ordre de grandeur $10^{-2}, 10^{-1} \ldots,\ 10^{2}$ et le facteur géométrique r en ordre de grandeur $2,\ 3, \ldots,\ 10$ pour Luby et $1.1,\ 1.2, \ldots,\ 2$ pour Walsh. Puis, le facteur d'échelle est multiplié par le nombre de tâches $n \times m$ pour prendre en compte la taille d'une instance. En effet, le facteur d'échelle

est souvent dépendant de la taille, profondeur et largeur, de l'arbre de recherche. Cette multiplication équilibre le nombre de redémarrages pour les différentes tailles des instances. Pour chaque instance, les 20 exécutions des variantes de l'algorithme où la solution initiale est donnée par LPT (toutes les exécutions débutent avec la même solution initiale) ont une limite de temps de 180 secondes.

Le meilleur réglage des paramètres est estimé en choisissant celui qui maximise le nombre d'instances résolues optimalement et en cas d'égalité celui qui minimise le temps moyen de résolution d'une instance. Remarquez que les moyennes intègrent les résultats des exécutions pour lesquelles le modèle n'a pas prouvé l'optimalité dans la limite de temps alloué (180 secondes). Ces moyennes sont donc en fait des bornes inférieures. Le tableau 6.1 fournit les résultats des expériences pour les deux stratégies de redémarrage avec enregistrement des *nogoods*. Nous donnons le pourcentage d'instances résolues, le temps moyen de

	Luby				Walsh			
	(s, r).	%	$\bar{t}$	$\bar{n}$	(s, r)	%	$\bar{t}$	$\bar{n}$
Best	(1,3)	82.6	43.1	10055	(1,1.1)	82.6	43.0	9863
Acceptable	(1, *)	82.0	44.5	10347	(0.1,*)	80.2	48.0	10973
Average	(*,*)	75.5	58.6	16738	(*,*)	76.2	54.6	11359
NotAcceptable	(0.01, *)	72.9	67.4	31676	(100, *)	70.5	67.1	12600

TABLE 6.1 – Identification de bons paramètres pour les stratégies de redémarrage avec enregistrement des *nogoods*.

résolution et le nombre moyen de nœuds visités pendant l'exploration pour différents jeux de paramètres. Le symbole * représente l'ensemble des valeurs testées pour un paramètre. La ligne intitulée Best donne les résultats pour les deux paires de paramètres choisis dans le réglage final. La ligne intitulée Average donne les résultats moyens sur l'ensemble des combinaisons de paramètres possibles. Les lignes intitulées Acceptable et NotAcceptable donnent respectivement les résultats moyens de paramétrages où la valeur du facteur d'échelle entraîne une amélioration ou une dégradation significative des performances. Comme espéré, l'estimation de bons paramètres (Best, Acceptable) entraîne une amélioration perceptible des performances par rapport à des stratégies non paramétrées (Average, NotAcceptable). De nombreux paramétrages obtiennent des résultats comparables et leurs performances dépendent principalement de la valeur du facteur d'échelle (par exemple, Acceptable and NotAcceptable). L'équivalence des stratégies de Luby et Wash avec enregistrement des *nogoods* est montrée expérimentalement avec le réglage final de leurs paramètres en section 6.3.2.2. Par conséquent, les autres sections discutent seulement des résultats obtenus avec la stratégie de Luby.

6.3.2 Analyse de sensibilité

Nous rapportons ici les résultats de l'évaluation des techniques présentées en section 6.2 et justifions expérimentalement leur utilisation. Nous rapportons uniquement les résultats des instances dont la taille est supérieure à 6×6 parce que les temps de résolution très courts des petites instances ne sont pas discriminants. Nous rappelons que l'algorithme est exécuté 20 fois pour chaque instance à cause de sa randomisation.

6.3.2.1 Solution initiale

Dans cette section, nous étudions l'influence de `CROSH` sur la résolution des différents jeux d'instances, et comparons `CROSH` à LPT, c'est-à-dire sa première itération. Une analyse préliminaire, non détaillée ici, a montré que `CROSH` exécutait 10000 itérations en moins de 1 seconde pour les instances dont la taille est inférieure à 9×9 et 25000 itérations en moins de 5 secondes pour les instances de taille 10×10. Pour les grandes instances de Taillard, les instances « faciles » sont souvent résolues optimalement en quelques secondes, mais les temps d'exécution montent quelquefois jusqu'à 20 secondes pour atteindre 25000 itérations.

Notons C_{max}^{UB} la borne supérieure sur le délai total associée à la solution initiale fournie par `CROSH`. L'écart à l'optimum est le quotient de la différence entre la borne supérieure et l'optimum sur l'optimum : $\frac{C_{max}^{UB} - C_{max}}{C_{max}}$. Le graphe à gauche de la figure 6.3 illustre la relation entre l'écart à l'optimum de la solution

initiale de CROSH et le temps de résolution de RRCP (donné sur l'axe horizontal). Chaque instance est représentée par un point dont la coordonnée x est le temps moyen de résolution de RRCP (échelle logarithmique) alors que sa coordonnée y est l'écart moyen à l'optimum de CROSH. On constate que la qualité de la solution initiale est satisfaisante puisque l'écart à l'optimum ne dépasse jamais 4% et que les solutions initiales sont optimales ou proches de l'optimum pour de nombreuses instances. On s'aperçoit aussi que le temps de résolution de RRCP n'est pas clairement relié à l'écart à l'optimum de CROSH, notamment pour les instances de Taillard. Par contre, les écarts à l'optimum augmentent pour les instances difficiles de Guéret-Prins et Brucker, alors qu'ils diminuent sur les grandes instances de Taillard. Une analyse plus approfondie, non détaillée ici, a montré que CROSH a découvert l'optimum au moins une fois pour 28 des 40 instances de Taillard, alors que cela n'est arrivé respectivement que pour 6 des 40 instances de Guéret-Prins, et pour 3 des 26 instances de Brucker. De plus, toutes les exécutions ont découvert l'optimum pour 10 grandes instances de Taillard (15×15, 20×20).

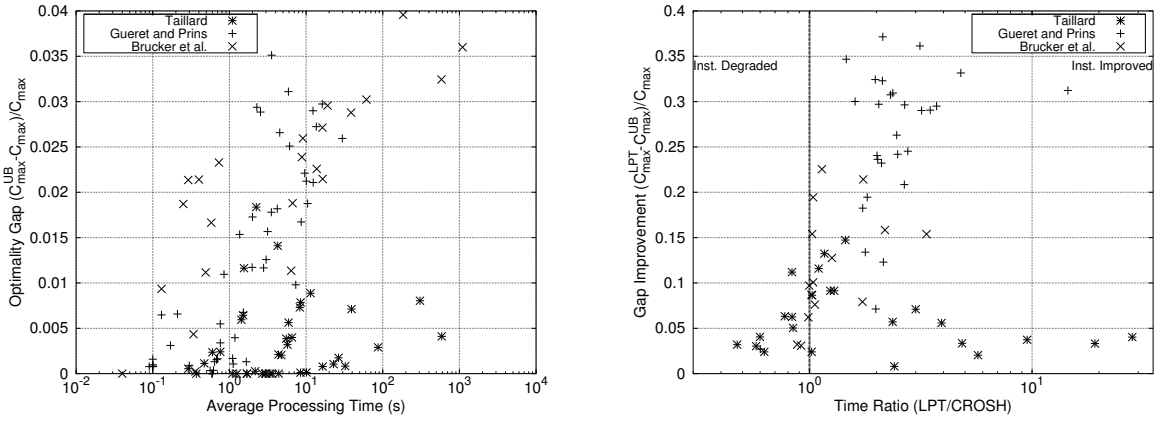

FIGURE 6.3 – Écart à l'optimum de la solution initiale et influence de CROSH sur les performances de RRCP.

Notons $C^{LPT}_{max} \geq C^{UB}_{max}$ la borne supérieure obtenue par LPT (la première itération de CROSH). L'amélioration de l'écart à l'optimum de CROSH par rapport à LPT est estimée en calculant le quotient de la différence entre les bornes supérieures fournies par LPT et CROSH sur l'optimum : $\frac{C^{LPT}_{max} - C^{UB}_{max}}{C_{max}}$. L'amélioration moyenne de l'écart à l'optimum est seulement de 5.7% pour les instances de Taillard, car LPT fournit déjà une borne supérieure précise sur ces instances. Mais, cette amélioration augmente respectivement jusqu'à 10.6% et 25.8% pour les instances de Brucker et Guéret-Prins parce que LPT est moins bien adaptée à ces jeux d'instances.
Le graphe de droite dans la figure 6.3 montre l'influence de CROSH sur notre algorithme. Chaque point représente une instance résolue par des variantes de notre algorithme avec redémarrage et enregistrement des *nogoods*. La coordonnée x est le quotient (échelle logarithmique) du temps moyen de résolution quand la solution initiale est donnée par LPT sur celui obtenu avec une solution initiale fournie par CROSH alors que la coordonnée y est l'amélioration moyenne de l'écart à l'optimum. Les 67 instances dont la taille est strictement supérieure à 6×6 et dont le temps moyen de résolution est compris entre 2 secondes et 1800 pour au moins une variante sont considérées pour dessiner ce graphe. Tous les points situés à droite de la droite ($x = 1 (= 10^0)$) correspondent aux instances dont la résolution est améliorée par l'utilisation de CROSH. L'amélioration des temps de résolution semble liée à celle de l'écart à l'optimum à l'exception des instances de Taillard. En dépit d'améliorations semblables des écarts à l'optimum, certaines instances de Taillard sont résolues 10 fois plus rapidement en utilisant CROSH alors que la résolution des quelques instances situées à gauche de la droite ($x = 1$) est dégradée.

6.3.2.2 Stratégies de redémarrage

Dans cette section, nous étudions l'influence des stratégies de redémarrage de Luby et Walsh sur la résolution et nous montrons expérimentalement leur équivalence dans le contexte du problème d'open-

shop. En utilisant les réglages finaux indiqués dans le tableau 6.1, nous montrons d'abord l'intérêt des redémarrages puis celui d'enregistrer des *nogoods* avant ces redémarrages grâce aux deux graphes de la figure 6.4. Les 61 instances dont la taille est strictement supérieure à 6×6 et dont le temps moyen de résolution est compris entre 2 secondes et 1800 pour au moins une des variantes sont considérées pour dessiner ces graphes. La solution initiale est fournie par `CROSH`.
Le graphe de gauche analyse l'effet sur la résolution des stratégies de redémarrage. Chaque instance est représentée par un point dont la coordonnée x est le quotient du temps moyen de résolution sans redémarrage sur le temps moyen de résolution avec redémarrage et la coordonnée y est le quotient du nombre de nœuds visités sans redémarrage sur le nombre de nœuds visités avec redémarrage. Les échelles des deux axes sont logarithmiques. Tous les points situés au-dessus ou à droite du point (1,1) représentent des instances dont la résolution est améliorée par les redémarrages (quadrant en haut à droite). Au contraire, tous les points situés en dessous ou à gauche du point (1,1) sont des instances dont la résolution est dégradée par les redémarrages (quadrant en bas à gauche). Comme espéré, tous les points se situent autour de la diagonale ($x = y$) puisque le nombre de nœuds visités est à peu près proportionnel au temps de résolution (les quadrants en haut à gauche et en bas à droite sont vides). Les redémarrages améliorent globalement la résolution et certaines instances sont même résolues approximativement 100 fois plus vite grâce aux redémarrages. Par contre, les performances sur un certain nombre d'instances situées en bas à gauche du point (1, 1) sont dégradées.

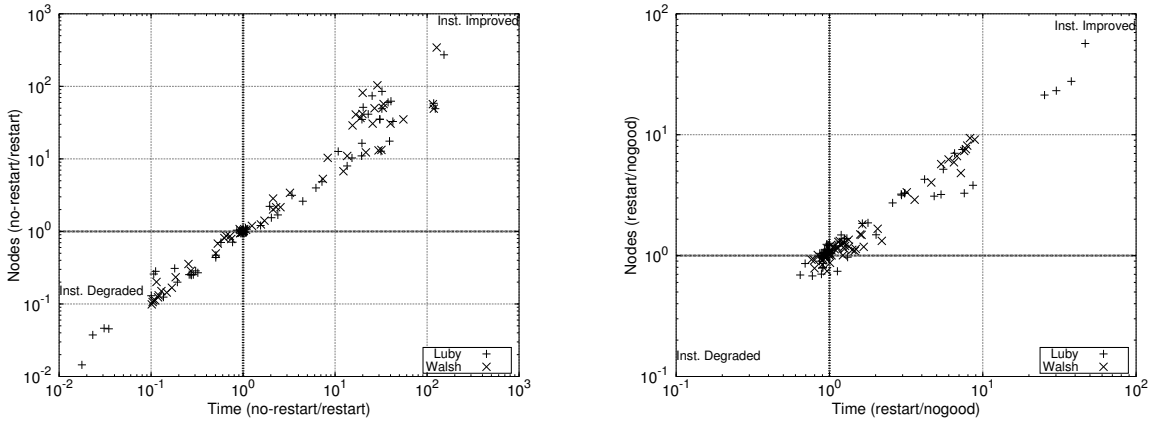

FIGURE 6.4 – Influence des redémarrages (à gauche) et de l'enregistrement des *nogoods* sur les stratégies de redémarrage (à droite).

De la même manière, le graphe de droite montre le gain obtenu par l'enregistrement des *nogoods* avant les redémarrages (les coordonnées de chaque point sont les quotients des temps de résolution et des nombres de nœuds sans et avec enregistrement des *nogoods*). On s'aperçoit que l'enregistrement des *nogoods* améliore la résolution avec redémarrages par un facteur compris entre 1 et 10 pour une grande majorité des instances.
Finalement, en combinant les redémarrages et l'enregistrement des *nogoods*, on obtient les résultats tracés dans le graphe à gauche de la figure 6.5 (les coordonnées de chaque point sont les quotients du temps de résolution et du nombre de nœuds sans redémarrage sur ceux avec redémarrage et enregistrement des *nogoods*). Ainsi, toutes les instances dont la résolution était dégradée par les redémarrages seuls dans la figure 6.4 ont disparu grâce à l'enregistrement des *nogoods*, tout en gardant les effets positifs dus aux redémarrages.
Nous avons montré ici que les redémarrages seuls peuvent améliorer grandement la résolution des problèmes d'open-shop mais manquent de robustesse. En simplifiant, les redémarrages aident à trouver de bonnes solutions rapidement mais, une fois celles-ci connues, la preuve d'optimalité peut demander beaucoup de temps. L'équilibre entre des redémarrages fréquents pour améliorer rapidement la borne supérieure et une exploration plus longue de l'espace de recherche pour prouver l'optimalité est difficile à trouver. L'enregistrement des *nogoods* avant les redémarrages compense cet inconvénient et améliore significativement la résolution comme l'illustre le graphe à gauche de la figure 6.5.

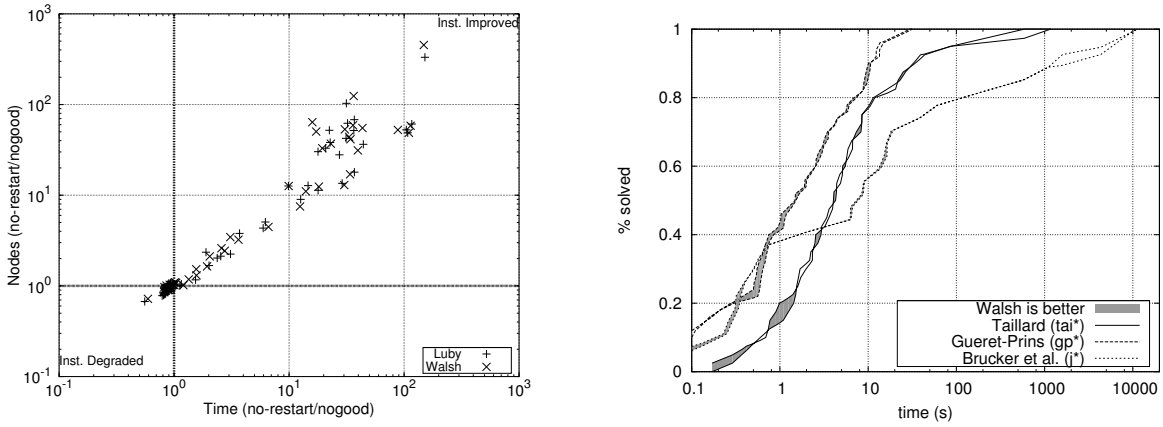

FIGURE 6.5 – Influence des redémarrages avec enregistrement des *nogoods* sur la résolution (à gauche) et équivalence des stratégies de redémarrage avec enregistrement des *nogoods* (à droite).

Finalement, le graphe à droite dans la figure 6.5 montre que les réglages finaux des stratégies de Luby et Walsh obtiennent des performances équivalentes. Pour chaque jeu d'instances, et pour chaque stratégie, le pourcentage d'instances résolues est dessiné comme une fonction du temps de résolution. Les deux courbes associées au même jeu d'instances sont représentées par des lignes dont le motif est identique. L'aire entre ces deux courbes est remplie en gris lorsque la stratégie de Walsh est meilleure que celle de Luby et reste blanche dans le cas contraire. Dans notre cas, les deux stratégies sont équivalentes puisque les deux courbes associées à chaque jeu d'instances sont presque confondues. De plus, ce graphique révèle un ordre croissant de difficulté des jeux d'instances pour notre algorithme : Guéret-Prins $\prec$ Taillard $\prec$ Brucker.

Le tableau 6.2 récapitule les temps moyens de résolution (heure :minute) et le nombre de nœuds visités (millions de nœuds) pour les trois instances les plus difficiles (les temps de résolution sont supérieurs à 1800 secondes). `RRCP` utilise la solution initiale fournie par `CROSH`. On remarque d'abord que l'algorithme sans redémarrage obtient les meilleures performances puisqu'il est nécessaire d'explorer de très nombreux nœuds pour obtenir la preuve d'optimalité. L'enregistrement des *nogoods* revêt alors un aspect critique puisqu'il permet de diviser par trois les temps de résolution. Sans enregistrement des *nogoods*, tous les redémarrages entre la découverte de l'optimum et le dernier sont inutiles puisque l'information est totalement perdue d'un redémarrage à l'autre. Malgré l'enregistrement des *nogoods*, le nombre important de redémarrages, notamment avec la stratégie de Luby, tend à accroître le temps de résolution à cause de la répétition de la sélection et de la propagation des points de choix initiaux. La stratégie de Walsh avec enregistrement des *nogoods* produit des résultats légèrement meilleurs que ceux de la stratégie de Luby. Par contre, son efficacité dépend lourdement de l'enregistrement des *nogoods* puisque les exécutions sont plus longues.

Instance		No Restart		Restart				Nogood Recording			
				Luby		Walsh		Luby		Walsh	
Name	Opt	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$	$\bar{t}$	$\bar{n}$
`j7-per0-0`	1048	1 :43	1.21	5 :56	4.66	18 :10	12.76	2 :10	1.57	2 :03	1.25
`j8-per0-1`	1039	2 :13	1.16	10 :22	5.95	23 :12	12.29	3 :07	1.65	3 :00	1.38
`j8-per10-2`	1002	1 :03	0.56	8 :46	5.11	8 :50	4.72	1 :17	0.68	1 :13	0.57

TABLE 6.2 – Moyennes du temps moyen de résolution $\bar{t}$ (heure :minute) et du nombre de nœuds $\bar{n}$ (en millions) pour différentes variantes appliquées aux trois instances les plus difficiles.

6.3.2.3 Robustesse

Pour finir, nous analysons la robustesse de RRCP quand on applique la stratégie de Luby avec enregistrement des *nogoods* et que la solution initiale est donnée par CROSH. La robustesse signifie, dans notre cas, la sensibilité à la borne supérieure initiale et aux arbitrages randomisés d'une exécution de RRCP à une autre. Pour chaque instance, nous calculons le quotient de l'écart-type sur la moyenne du temps de résolution : $\frac{\mathtt{std}(\bar{t})}{\bar{t}}$. Puis, nous calculons la moyenne de ces quotients pour chaque jeu d'instances. Le jeu d'instances de Taillard obtient le plus haut quotient égal à 62% parce que la preuve d'optimalité est triviale. Les temps de résolution varient énormément en fonction de l'optimalité de la solution initiale d'une exécution à l'autre. En effet, l'instance est résolue sans exploration si la solution initiale est optimale et, dans le cas contraire, le lancement de la recherche arborescente complète entraîne une augmentation du temps de résolution. Au contraire, l'algorithme est plus robuste pour les instances de Guéret-Prins et Brucker où ces quotients sont respectivement égaux à 16% et 9%. En effet, l'algorithme passe plus de temps à prouver l'optimalité de la dernière solution qu'à l'atteindre, car le nombre de nœuds visités pour obtenir la preuve d'optimalité est très élevé.

6.3.3 Comparaison avec d'autres approches

Dans cette section, nous comparons RRCP (sans limite de temps) aux approches de la littérature (voir section 4.3). La variante de RRCP testée est celle utilisant la stratégie de Luby et une solution initiale donnée par CROSH. Les tableaux 6.3, 6.4 et 6.5 récapitulent respectivement les résultats de différentes approches de la littérature sur les jeux d'instances de Taillard, Brucker et Guéret-Prins. Ces tableaux incluent les meilleurs résultats obtenus par un algorithme génétique [GA-Pri – 112], un algorithme de colonies de fourmis [ACO-Blu – 114], un algorithme d'essaim de particules [PSO-Sha – 115], un branch-and-bound avec retour arrière intelligent [BB-Gue – 119], un branch-and-bound avec test de consistance [BB-Do – 121], et une reformulation vers SAT [SAT-Ta – 31]. Les travaux cités ci-dessus mentionnent quelquefois plusieurs résultats obtenus avec différentes variantes de leur approche et nous n'avons gardé que les meilleurs d'entre eux dans les Tableaux 6.3, 6.4 et 6.5. La colonne Opt donne l'optimum pour chaque instance. La valeur de l'objectif est indiquée en gras lorsqu'il est égal à optimum. Plus précisemment :

GA-Pri : nous rapportons la valeur de l'objectif après une exécution unique mais pas le temps d'exécution.

ACO-Blu : les résultats des 20 exécutions pour chaque instance sont obtenus sur des PCs avec un processeur AMD Athlon cadencé à 1.1 Ghz tournant sous Linux. Nous rapportons la valeur moyenne de l'objectif (Avg) et le temps moyen de résolution ($\bar{t}$). La meilleure valeur de l'objectif (Best) obtenue pendant les 20 exécutions est précisée lorsqu'elle n'est pas égale à l'optimum pour toutes les instances.

PSO-Sha : les résultats des 20 exécutions pour chaque instance de PSO-Sha sont obtenus sur des PCs avec un processeur AMD Athlon cadencé à 1.8 Ghz tournant sous Windows XP. Nous rapportons les mêmes informations que pour ACO-Blu. La meilleure valeur de l'objectif de PSO-Sha est précédée du symbole † si l'opérateur de décodage n'est pas hybridé avec une recherche arborescente tronquée.

BB-Gue : nous ne mentionnons pas les temps d'exécution, car la limite de 250 000 backtracks était souvent atteinte (approximativement 3 heures de temps CPU sur un Pentium PC cadencé à 133 MHz).

BB-Do : nous mentionnons les temps de résolution t de BB-Do obtenus sur un Pentium II 333 Mhz tournant sous MSDOS avec une limite de temps de 5 heures car ils ont identifié de nombreuses solutions optimales. Le symbole – indique que leur algorithme bottom-up a été interrompu sans trouver de solution. La meilleure borne supérieure est précisée entre parenthèses lorsque leur algorithme top-down a été interrompu avant d'obtenir la preuve d'optimalité.

SAT-Ta : les temps de résolution de SAT-Ta obtenus sur un Intel Xeon cadencé à 2.8GHz et 4GB de mémoire vive sont donnés à l'exception de ceux des instances `j7-per0-0` et `j8-per0-1` qui ont été résolues en parallèle sur 10 Mac mini (PowerPC G4, 1.42GHz, 1GB de mémoire vive) en divisant chaque problème en 120 sous-problèmes. Les solutions optimales ont été découvertes et prouvées en 13 heures de calcul (noté M dans le tableau 6.4).

CNR-Cha : Chatzikokolakis *et al.* [113] interrompent leur recherche locale après 120 minutes, mais seulement si le temps écoulé depuis la dernière amélioration dépasse 30 minutes. CNR-Cha ne précise ni les temps de résolution, ni les meilleures solutions découvertes. Par conséquent, ces résultats ne sont pas mentionnés dans les tableaux 6.3, 6.4 et 6.5.

MCS-Lab : Laborie [122] exécute sur un portable Dell Latitude D600 cadencé à 1.4 GHz son algorithme bottom-up en imposant une limite de 5 secondes sur la résolution de chaque sous-problème. L'algorithme est interrompu sans retourner de solution dès que la limite de temps est atteinte pour un sous-problème. MCS-Lab ne précise pas les temps de résolution et n'apparaissent pas non plus dans les tableaux. Nous utiliserons, à titre indicatif, une estimation du temps de résolution basée sur les considérations présentés ci-dessous issues de la section 2.4.

Un algorithme bottom-up résout $C_{max} - C_{max}^{LB}$ problèmes insatisfiables et un unique problème satisfiable donnant l'optimum. Ce nombre de sous-problèmes dépend donc de l'instance considérée même si la valeur de C_{max}^{LB} est toujours égale à 1000 pour les jeux d'instances de Brucker et Guéret-Prins. MCS-Lab précise que les premières itérations étaient courtes et que les dernières s'allongeaient durant une transition de phase. Par conséquent, notre estimation du temps de résolution de MCS-Lab est inspirée par la variante dichotomique de bottom-up : $5 \times \lceil \log_2 \left(C_{max} - C_{max}^{LB} + 1 \right) + 1 \rceil$ secondes.

Nous rappelons que nos expérimentations ont été menées sur une grille de calcul composée de machines Linux, où chaque nœud possède 1 GB de mémoire vive et un processeur AMD cadencé à 2.2 GHz. Nous rapportons la moyenne sur 20 exécutions du temps de résolution $\bar{t}$ et du nombre de nœuds visités $\bar{n}$. Remarquez que la comparaison des temps de résolution n'est pas toujours significative en raison des différentes configurations matérielles.

Résultats sur les instances de Taillard (Table 6.3) Ce jeu d'instances est réputé facile puisque la preuve d'optimalité est triviale et qu'il est ainsi résolu efficacement par les métaheuristiques. Ainsi, six instances seulement restent ouvertes après une unique exécution de GA-Pri. ACO-Blu et PSO-Sha ferment les dernières instances avec de bons temps de résolution même si certaines exécutions ne découvrent pas l'optimum. Ces échecs ne sont pas clairement reliés à la taille des instances. Au contraire, CNR-Cha obtient ses plus faibles résultats sur ce jeu d'instances puisqu'il ne découvre que huit solutions optimales parmi les instances de taille 7×7 et 10×10, et ne fournit aucun résultat pour les grandes instances. Les temps moyens de résolution de `RRCP` sont comparables à ceux des métaheuristiques sauf pour les instances `tai_20_20_02` et `tai_20_20_08`. Cependant, toutes les métaheuristiques échouent au moins une fois sur l'instance `tai_20_20_08`. On constate aussi que `CROSH` est plus efficace que des métaheuristiques complexes sur de nombreuses grandes instances (si le nombre moyen de nœuds visités $\bar{n}$ est nul, alors toutes les exécutions de `CROSH` découvrent l'optimum).

BB-Do est la première méthode exacte à résoudre toutes les instances 10×10 et la plupart des 15×15 et 20×20. Les performances de leur algorithme bottom-up surpassent clairement celle de leur algorithme top-down sur ce jeu d'instances puisqu'il n'a besoin que de résoudre un unique sous-problème satisfiable pour découvrir l'optimum. Dans notre cas, la précision de la borne supérieure initiale donnée par `CROSH` compense les inconvénients relatifs à l'usage d'un algorithme top-down. De plus, la diversification issue de la randomisation et des redémarrages aide à remettre en cause les mauvais choix initiaux alors que plusieurs instances restent sans solution avec BB-Do, par exemple `tai_15_15_02`. Avec des configurations matérielles équivalentes, les performances de `RRCP` outrepassent clairement celles de SAT-Ta, la première méthode exacte à fermer ce jeu d'instances. Par contre, les temps de résolution de SAT-Ta sont une fonction linéaire du nombre de clauses, donc de la taille des instances, ce qui n'est pas nécessairement le cas pour `RRCP`. Contrairement aux autres approches, SAT-Ta ne résout pas l'instance `tai_20_20_08` plus difficilement que les autres 20×20. Pour finir, MCS-Lab ne rapporte aucun résultat sur le jeu d'instances de Taillard.

Résultats sur les instances de Brucker (Table 6.4) Le jeu d'instances de Brucker est né en réaction à la faible difficulté des instances de Taillard. GA-Pri est l'approche la plus affectée puisqu'elle ne découvre que cinq solutions optimales. Les temps de résolution de ACO-Blu et PSO-Sha augmentent par rapport à ceux sur le jeu d'instances de Taillard et certaines solutions optimales ne sont jamais découvertes. Au contraire, CNR-Cha rapporte l'amélioration de 3 bornes supérieures et la preuve de 17 solutions optimales. `RRCP` obtient de bons temps de résolution par rapport aux métaheuristiques à l'exception de

Instance		Metaheuristics					Exact Algorithms			
		GA-Pri	ACO-Blu		PSO-Sha		BB-Do	SAT-Ta	RRCP	
Name	Opt		Avg	$\bar{t}$	Avg	$\bar{t}$	t	t	$\bar{t}$	$\bar{n}$
tai_7_7_1	435	436	**435**	2.1	**435**	2.9	0.4	21	1.6	355
tai_7_7_2	443	447	**443**	19.2	**443**	12.2	0.9	24	1.6	448
tai_7_7_3	468	472	**468**	16.0	**468**	9.2	30.9	30	4.3	1159
tai_7_7_4	463	**463**	**463**	1.7	**463**	3.0	5.3	20	1.5	478
tai_7_7_5	416	417	**416**	2.3	**416**	2.9	2.0	22	0.8	157
tai_7_7_6	451	455	451.4	24.8	**451**	13.5	95.8	45	11.5	3945
tai_7_7_7	422	426	422.2	23.0	**422**	13.6	167.7	33	2.3	602
tai_7_7_8	424	**424**	**424**	1.2	**424**	2.3	5.0	20	0.6	189
tai_7_7_9	458	**458**	**458**	1.1	**458**	1.3	0.8	21	0.3	109
tai_7_7_10	398	**398**	**398**	1.6	**398**	2.8	53.2	20	0.5	105
tai_10_10_1	637	**637**	637.4	40.1	**637**	9.4	30.2	98	8.3	1214
tai_10_10_2	588	**588**	**588**	3.0	**588**	3.5	70.6	95	4.8	667
tai_10_10_3	598	**598**	**598**	27.9	**598**	10.1	185.5	92	8.5	1162
tai_10_10_4	577	**577**	**577**	2.6	**577**	2.6	29.7	92	2.2	264
tai_10_10_5	640	**640**	**640**	8.6	**640**	4.0	32.0	96	6.6	830
tai_10_10_6	538	**538**	**538**	2.6	**538**	1.1	32.7	95	0.4	0
tai_10_10_7	616	**616**	**616**	5.2	**616**	3.9	30.9	103	4.4	403
tai_10_10_8	595	**595**	**595**	15.0	**595**	7.0	44.1	95	6.0	633
tai_10_10_9	595	**595**	**595**	5.1	**595**	4.1	39.8	97	5.8	541
tai_10_10_10	596	**596**	**596**	7.5	**596**	5.0	29.1	95	5.6	541
tai_15_15_1	937	**937**	**937**	14.3	**937**	4.3	481.4	523	4.4	0
tai_15_15_2	918	**918**	**918**	21.1	**918**	9.1	–	567	26.5	2190
tai_15_15_3	871	**871**	**871**	14.3	**871**	4.3	611.6	543	3.4	0
tai_15_15_4	934	**934**	**934**	14.2	**934**	3.9	570.1	560	1.7	0
tai_15_15_5	946	**946**	**946**	25.7	**946**	5.7	556.3	541	8.5	1760
tai_15_15_6	933	**933**	**933**	16.6	**933**	4.7	574.5	560	3.0	0
tai_15_15_7	891	**891**	**891**	20.1	**891**	10.4	724.6	566	16.5	1896
tai_15_15_8	893	**893**	**893**	14.2	**893**	17.3	614.0	546	1.3	0
tai_15_15_9	899	**899**	899.7	4.1	899.2	26.6	646.9	568	39.2	4053
tai_15_15_10	902	**902**	**902**	18.1	**902**	6.9	720.1	586	22.9	2081
tai_20_20_1	1155	**1155**	**1155**	54.1	**1155**	16.6	3519.8	3105	32.4	3340
tai_20_20_2	1241	**1241**	**1241**	79.7	**1241**	23.5	–	3559	588.4	45606
tai_20_20_3	1257	**1257**	**1257**	48.6	**1257**	19.6	4126.3	2990	3.0	0
tai_20_20_4	1248	**1248**	**1248**	49.1	**1248**	19.6	–	3442	2.7	0
tai_20_20_5	1256	**1256**	**1256**	49.1	**1256**	19.6	3247.3	3603	3.7	0
tai_20_20_6	1204	**1204**	**1204**	49.3	**1204**	19.6	3393.0	2741	10.2	1879
tai_20_20_7	1294	**1294**	**1294**	65.0	**1294**	25.4	2954.8	2912	86.9	8620
tai_20_20_8	1169	1171	1170.3	27.9	1170	50.9	–	2990	305.8	25503
tai_20_20_9	1289	**1289**	**1289**	48.6	**1289**	78.2	3593.8	3204	1.7	0
tai_20_20_10	1241	**1241**	**1241**	48.8	**1241**	78.2	4936.2	3208	1.1	0

TABLE 6.3 – Résultats sur les instances de Taillard.

quelques instances pour lesquelles la comparaison est délicate puisque les temps de résolution de RRCP sont supérieurs, mais les métaheuristiques ne découvrent pas toujours l'optimum.

L'algorithme top-down de BB-Do a prouvé l'optimalité de huit instances 7×7 parmi neuf. MCS-Lab a fermé trois des six dernières instances ouvertes (j8-per0-2, j8-per10-0, and j8-per10-1) et SAT-Ta ferma plus tard les trois dernières (j7-per0-0, j8-per0-1, and j8-per10-2). À l'exception des instances j7-per10-2, j8-per10-0 et j8-per10-1, les temps de résolution de RRCP sont meilleurs que ou similaires aux estimations pour MCS-Lab qui varient entre 5 et 35 secondes. Les temps de résolution de SAT-Ta restent supérieurs à ceux de RRCP, spécialement pour les instances j7-per0-0 et j8-per0-1 pour lesquelles leurs expériences ont demandé un temps de calcul très important. Sur ce jeu d'instances, leurs temps de résolution n'ont plus un comportement linéaire en fonction de la taille du problème.

Résultats sur les instances de Guéret-Prins (Table 6.5) Le jeu d'instance de Guéret-Prins semble difficile puisque les métaheuristiques sont moins efficaces. Les résultats de ACO-Blu et PSO-Sha se dégradent au fur et à mesure que la taille du problème augmente. Malgré des temps d'exécution plus importants, ils découvrent moins de solutions optimales, surtout ACO-Blu. En comparaison, GA-Pri est plus efficace que sur le jeu d'instances de Brucker et CNR-Cha déclare améliorer 12 bornes supérieures. RRCP est particulièrement bien adapté à ce jeu d'instances puisque ses performances surpassent celles des métaheuristiques pour toutes les tailles d'instances.

BB-Gue a prouvé l'optimalité de toutes les instances jusqu'à la taille 6×6 et de quelques instances de taille supérieure. BB-Do ne rapporte aucun résultat sur ce jeu d'instance et MCS-Lab le ferme, mais de

Instance		Metaheuristics							Exact Algorithms			
		GA-Pri	ACO-Blu			PSO-Sha			BB-Do	SAT-Ta	RRCP	
Name	Opt		Best	Avg	$\bar{t}$	Best	Avg	$\bar{t}$	t	t	$\bar{t}$	$\bar{n}$
j6-per0-0	1056	1080	**1056**	**1056**	27.4	**1056**	**1056**	42.1	133.0	817	38.7	11032
j6-per0-1	1045	**1045**	**1045**	1049.7	61.3	**1045**	**1045**	59.7	5.2	57	0.3	198
j6-per0-2	1063	1079	**1063**	**1063**	38.8	**1063**	**1063**	72.6	18.0	57	0.6	223
j6-per10-0	1005	1016	**1005**	**1005**	10.6	**1005**	**1005**	45.5	14.4	52	0.8	263
j6-per10-1	1021	1036	**1021**	**1021**	11.3	**1021**	**1021**	21.0	4.6	46	0.3	177
j6-per10-2	1012	**1012**	**1012**	**1012**	1.4	**1012**	**1012**	8.5	13.8	51	0.5	188
j6-per20-0	1000	1018	**1000**	1003.6	31.1	**1000**	**1000**	77.5	10.7	60	0.4	208
j6-per20-1	1000	**1000**	**1000**	**1000**	0.8	**1000**	**1000**	1.5	0.4	46	0.2	161
j6-per20-2	1000	1001	**1000**	**1000**	3.9	**1000**	**1000**	30.6	1.0	40	0.4	179
j7-per0-0	1048	1071	**1048**	1052.7	207.9	1050	1051.2	104.9	(1058)	M	7777.2	1564192
j7-per0-1	1055	1076	1057	1057.8	91.6	**†1055**	1058.8	155.8	9421.8	428	16.5	3265
j7-per0-2	1056	1082	1058	1059	175.9	**1056**	1057	124.5	9273.5	292	16.4	3120
j7-per10-0	1013	1036	**1013**	1016.7	217.6	**1013**	1016.1	183.8	2781.9	332	19.1	3981
j7-per10-1	1000	1010	**1000**	1002.5	189.9	**1000**	**1000**	81.9	1563.0	121	6.4	1276
j7-per10-2	1011	1035	1016	1019.4	180.7	1013	1014.9	125.6	15625.1	1786	583.1	128289
j7-per20-0	1000	**1000**	**1000**	**1000**	0.4	**1000**	**1000**	1.9	48.8	66	0.1	0
j7-per20-1	1005	1030	**1005**	1007.6	259.1	1007	1008	143.2	318.8	132	8.9	2130
j7-per20-2	1003	1020	**1003**	1007.3	257.3	**1003**	1004.7	160.9	2184.9	132	13.8	3150
j8-per0-1	1039	1075	**1039**	1048.7	313.5	**1039**	1043.3	220.8		M	11168.9	1648700
j8-per0-2	1052	1073	**1052**	1057.1	323.4	**1052**	1053.6	271.9		870	61.3	9379
j8-per10-0	1017	1053	1020	1026.9	346.5	1020	1026.1	205.0		2107	184.5	24548
j8-per10-1	1000	1029	1004	1012.4	308.9	1002	1007.6	202.2		8346	1099.3	165875
j8-per10-2	1002	1027	1009	1013.7	399.4	**1002**	1006	162.8		7789	4596.5	673451
j8-per20-0	1000	1015	**1000**	1001	237.2	**1000**	1000.6	136.9		148	9.1	2104
j8-per20-1	1000	**1000**	**1000**	**1000**	2.6	**1000**	**1000**	4.5		136	0.4	128
j8-per20-2	1000	1014	**1000**	1000.6	286.2	**1000**	**1000**	105.8		144	6.7	1512

TABLE 6.4 – Résultats sur les instances de Brucker.

nouveau sans préciser les temps de résolution. Les temps de résolution de RRCP sont strictement inférieurs aux estimations pour MCS-Lab qui varient entre 40 et 50 secondes. Les temps de résolution de SAT-Ta sont de nouveau une fonction de la taille des instances mais restent peu compétitifs par rapport à ceux de RRCP.

6.4 Conclusion

Nous avons présenté une approche en ordonnancement sous contraintes pour résoudre le problème d'open-shop. L'algorithme s'appuie sur le calcul d'une solution initiale de manière heuristique avant de résoudre un modèle déclaratif basé sur un langage de haut niveau (tâches, ressources, précédences) grâce à une recherche arborescente de type top-down complétée par un mécanisme de randomisation et de redémarrage.
Les résultats expérimentaux sont comparables à ceux des métaheuristiques sur le jeu d'instances de Taillard. De plus, RRCP obtient de meilleures solutions que les métaheuristiques en un temps moindre sur les jeux d'instances de Brucker et Guéret-Prins. Les résultats expérimentaux surpassent ceux des méthodes exactes pour lesquelles les temps de résolution des instances sont disponibles.
Au chapitre 7, nous proposerons et étudierons de nouvelles heuristiques de sélection pour l'arbitrage du graphe disjonctif et nous appliquerons notre algorithme aux problèmes d'open-shop et de job-shop.

Instance		Metaheuristics							Exact Algorithms			
		GA-Pri	ACO-Blu			PSO-Sha			BB-Gue	SAT-Ta	RRCP	
Name	Opt		Best	Avg	$\bar{t}$	Best	Avg	$\bar{t}$		t	$\bar{t}$	$\bar{n}$
gp06-01	1264	**1264**	**1264**	1264.7	30.8	**1264**	**1264**	176.1	**1264**	57	0.3	80
gp06-02	1285	**1285**	**1285**	1285.7	48.7	**1285**	**1285**	147.8	**1285**	65	0.2	172
gp06-03	1255	**1255**	**1255**	**1255**	30.0	**1255**	1255.6	133.1	**1255**	72	0.1	124
gp06-04	1275	**1275**	**1275**	**1275**	25.9	**1275**	**1275**	60.8	**1275**	63	0.1	67
gp06-05	1299	1300	**1299**	1299.2	39.9	**1299**	**1299**	159.6	**1299**	65	0.1	67
gp06-06	1284	**1284**	**1284**	**1284**	43	**1284**	**1284**	109.4	**1284**	65	0.1	68
gp06-07	1290	**1290**	**1290**	**1290**	10.5	**1290**	**1290**	1.6	**1290**	77	0.1	63
gp06-08	1265	1266	**1265**	1265.2	71.9	**1265**	1265.5	134.3	**1265**	71	0.1	52
gp06-09	1243	1243	**1243**	**1243**	9.8	**1243**	1243.1	156.5	1264	72	0.2	170
gp06-10	1254	**1254**	**1254**	**1254**	4.6	**1254**	**1254**	79.8	**1254**	57	0.3	241
gp07-01	1159	**1159**	**1159**	**1159**	86.9	**1159**	1159.3	223.7	1160	99	0.9	367
gp07-02	1185	**1185**	**1185**	**1185**	80.3	**1185**	**1185**	1.2	1191	148	0.6	4
gp07-03	1237	**1237**	**1237**	**1237**	40.9	**1237**	**1237**	9.5	1242	132	0.7	54
gp07-04	1167	**1167**	**1167**	**1167**	59.2	**1167**	**1167**	160.4	1167	131	0.7	144
gp07-05	1157	**1157**	**1157**	**1157**	124.4	**1157**	**1157**	139.1	1191	141	0.8	304
gp07-06	1193	**1193**	**1193**	1193.9	152.4	**1193**	1193.1	198.6	1200	127	0.8	306
gp07-07	1185	**1185**	**1185**	1185.1	91.1	**1185**	**1185**	1.4	1201	102	0.6	48
gp07-08	1180	1181	**1180**	1181.4	206.7	**1180**	**1180**	139.4	1183	144	0.7	117
gp07-09	1220	**1220**	**1220**	1220.1	127.9	**1220**	**1220**	143.9	**1220**	150	0.7	177
gp07-10	1270	**1270**	**1270**	1270.1	65.6	**1270**	**1270**	0.5	**1270**	127	0.6	4
gp08-01	1130	1160	**1130**	1132.4	335.0	**†1130**	1140.3	277.3	1195	160	2.6	1485
gp08-02	1135	1136	**1135**	1136.1	228.4	**1135**	1135.4	258.3	1197	190	1.2	304
gp08-03	1110	1111	1111	1113.7	336.3	**1110**	1114	240.3	1158	197	1.6	622
gp08-04	1153	1168	1154	1156	275.7	**1153**	1153.2	308.1	1168	227	1.4	566
gp08-05	1218	**1218**	1219	1219.8	347.7	**1218**	1218.9	56.6	**1218**	247	1.2	206
gp08-06	1115	1128	1116	1123.2	359.2	**1115**	1126.9	249.6	1171	175	2.3	1498
gp08-07	1126	1128	**1126**	1134.6	296.8	**1126**	1129.8	287.3	1157	204	3.6	2775
gp08-08	1148	**1148**	**1148**	1149	277.4	**1148**	**1148**	179.3	1191	183	2.0	1281
gp08-09	1114	1120	1117	1119	279.0	**1114**	1114.3	223.6	1142	189	2.0	1140
gp08-10	1161	**1161**	**1161**	1161.5	281.3	**1161**	1161.4	217.1	**1161**	203	1.1	245
gp09-01	1129	1143	1135	1142.8	412.9	**1129**	1133.2	376.3	1150	323	3.6	1691
gp09-02	1110	1114	1112	1113.7	430.8	**†1110**	1114.1	335.9	1226	327	10.7	8000
gp09-03	1115	1118	1118	1120.4	428.0	†1116	1117	313.4	1150	395	2.8	1422
gp09-04	1130	1131	**1130**	1140	549.7	**1130**	1135.8	328.7	1181	340	4.3	2219
gp09-05	1180	**1180**	**1180**	1180.5	295.9	**1180**	**1180**	22.3	**1180**	362	1.7	266
gp09-06	1093	1117	**1093**	1195.6	387.0	**1093**	1094.1	277.2	1136	401	4.6	2387
gp09-07	1090	1119	1097	1101.4	431.4	1091	1096.5	376.4	1173	339	5.9	3483
gp09-08	1105	1110	1106	1113.7	376.2	1108	1108.3	334.6	1193	349	3.1	1446
gp09-09	1123	1132	1127	1132.5	402.6	**†1123**	1126.5	358.6	1218	316	3.2	1537
gp09-10	1110	1130	1120	1126.3	435.8	†1112	1126.5	297.7	1166	355	6.1	2784
gp10-01	1093	1113	1099	1109	567.5	**1093**	1096.8	455.7	1151	470	29.8	6661
gp10-02	1097	1120	1101	1107.4	501.7	**1097**	1099.1	382.7	1178	526	9.7	3140
gp10-03	1081	1101	1082	1098	658.7	**†1081**	1090.3	450.8	1162	535	13.6	4196
gp10-04	1077	1090	1093	1096.6	588.1	1083	1092.1	371.8	1165	515	12.4	3921
gp10-05	1071	1094	1083	1092.4	636.4	†1073	1092.2	314.1	1125	515	16.3	4782
gp10-06	1071	1074	1088	1104.6	595.5	**1071**	1074.3	289.7	1179	508	12.4	3894
gp10-07	1079	1083	1084	1091.5	389.6	†1080	1081.1	167.4	1172	523	8.7	2188
gp10-08	1093	1098	1099	1104.8	615.9	†1095	1097.6	324.5	1181	498	10.5	3477
gp10-09	1112	1121	1121	1128.7	554.5	†1115	1127	428.2	1188	541	10.1	3303
gp10-10	1092	1095	1097	1106.7	562.5	**1092**	1094	487.9	1172	656	7.4	1724

TABLE 6.5 – Résultats sur les instances de Guéret-Prins.