L'ordonnancement sur machines parallèles avec dates de début au plus tôt

Nous avons introduit les problèmes d'ordonnancement dans un chapitre précédent. Dans ce chapitre, nous présentons le problème d'ordonnancement sur machines parallèles. Nous étudions le cas où plusieurs machines identiques sont disposées parallèlement, avec des tâches arrivant à l'atelier à des moments différents (dates de début au plus tôt des tâches différentes). Ce problème peut être trouvé dans la composition de certains ateliers à cheminement unique, où au moins une opération peut être exécutée sur plusieurs équipements disposés parallèlement. Dans ces ateliers, les dates de fin des opérations précédentes imposent des dates de lancement minimales pour les opérations restantes. Ce scénario correspond au problème étudié dans ce chapitre.

Dans ce type de problème combinatoire, toute instance est caractérisée par le nombre de machines disponibles et la quantité de tâches à ordonnancer. A cela s'ajoute l'information disponible sur les tâches. Nous considérons que l'ensemble de données utilisées (temps de traitement, date de disponibilité et date de fin au plus tard) sont connues à l'avance.

Le contexte de recherche (chapitre 1) décrit une situation dans laquelle la qualité de service du centre de recherche CRIS peut être évaluée en mesurant le retard pris dans le développement de nouveaux produits. Ainsi, l'objectif du problème d'ordonnancement traité ici est la minimisation du retard total, qui correspond à la somme de retards encourus par les tâches lors de leur exécution.

Ce chapitre présente une description complète du problème ainsi qu'un ensemble de méthodologies adaptées à sa résolution. Des méthodes exactes et approchées sont considérées. Parmi les méthodes exposées, nous proposons une méthode basée sur la programmation

linéaire pour la solution exacte, et des méthodes heuristiques de différentes natures pour les solutions approchées. Les résultats expérimentaux sont utilisés pour évaluer et comparer les différentes méthodes testées et identifier la plus performante. Dans une analyse plus approfondie, nous étudions non seulement les performances obtenues, mais aussi les relations existantes entre ces performances et les paramètres uniques à chaque instance résolue. Ainsi, nous proposons des limites au-dessous desquelles, il serait inutile de tester des méthodes heuristiques car la quasi-totalité d'heuristiques décrites peuvent trouver des solutions de bonne qualité. Ces résultats sont également utilisés dans le développement d'un système d'aide à la décision, qui prend les données caractérisant chaque instance, pour identifier la ou les méthodes de résolution adaptées à sa résolution. Le fonctionnement de cette procédure est expliqué sur des instances réelles, issues du cas industriel réel.

3.1 Introduction

Dans des situations concurrentiels difficiles, au niveau stratégique et tactique, l'achat de multiples équipements ou l'embauche des équipes de travail supplémentaires peut sembler une décision utile et souhaitable afin d'accélérer une étape dans le cycle de production. Au niveau opérationnel, utiliser ces ressources au mieux devient une tâche difficile à exécuter, surtout dans les cas où les tâches ne sont pas identiques (voir chapitre 2).

Par conséquent, l'utilisation des équipements disposés de manière parallèle, a des conséquences sur les profits de l'entreprise, à cause des retards générés. Cela impacte l'image de l'entreprise comme porteuse de la recherche (dans la recherche des biocatalyseurs performants).

Dans les ateliers constitués de machines identiques, les décisions d'ordonnancement se résument à définir des séquences de passage sur les machines, autrement dit, à décider quelle tâche suit quelle tâche sur chaque machine, pour toutes les machines, et toutes les tâches.

Si nous considérons que tout délai supplémentaire peut générer des pertes, soit à cause des ventes non satisfaites, soit à cause des produits rejetés (notamment dans les produits périssables ou très sensibles aux conditions environnementales), il est dans l'intérêt de toute entreprise de maîtriser la gestion de la production, et ainsi contrôler les temps et les délais relatifs au management des ressources disponibles pour effectuer les tâches de production. De ce fait, la minimisation du retard total (la somme des retards des tâches) se présente comme une nécessité lorsque les surcoûts issus des retards potentiels sont très élevés. Cela s'applique dans le cas où des dates de fin souhaitées sont exprimées et connues dès le début de la période considérée. Les dates de fin souhaitée peuvent venir directement des clients, ou

des interactions entre différents postes afin de respecter un délai fixé.

De la même manière, les développements dans la résolution de ce problème peuvent conduire à des méthodes efficaces dans la gestion de l'ordonnancement des ateliers à cheminement unique, où certaines opérations disposent de plusieurs équipements travaillant en parallèle.

Le reste de ce chapitre traite de cette problématique, et propose plusieurs méthodologies pour le résoudre. Dans la section 3.2 nous présentons une description complète de ce problème, en donnant les notations nécessaires pour le reste du chapitre. Dans la section 3.3, un modèle mathématique est utilisé pour représenter ce problème. Étant donné que l'objectif reste celui de la minimisation du retard total, nous proposons dans la section 3.4 une méthodologie pour classifier les tâches entre celles qui seront en avance, celles qui seront en retard et le reste. Cette classification est utilisée dans une section ultérieure pour comprendre la composition des solutions extraites des méthodes simples basées sur les paramètres des instances résolues. Ces méthodes de résolution sont présentées dans la section 3.5, où la totalité des méthodes développées dans ce projet pour résoudre le problème d'ordonnancement décrit sont détaillées. Les résultats expérimentaux sont donnés pour évaluer et comparer ces méthodes dans la section 3.6. Finalement, nous présentons dans la section 3.7 une étude réalisée sur l'ensemble des résultats obtenus, afin d'identifier des limites des paramètres liés aux instances résolues, au-dessous desquelles tester ou comparer des méthodes de résolution ne représente aucun intérêt, parce que les écarts ne semblent pas significatifs. De la même manière, ces résultats sont utilisés dans une procédure qui, à partir des caractéristiques propres à chaque instance, permettent d'identifier les méthodes de résolution qui s'adaptent le mieux à sa résolution.

3.2 Présentation du problème

Le problème de l'ordonnancement de tâches avec des machines parallèles pour la minimisation du retard total, où au moins une des tâches arrive à l'atelier à une date différente du début de la période considérée, est distingué, en utilisant la nomenclature présente en [75], comme $P_m/r_i/\sum T_i$.

Ce problème consiste à ordonnancer un ensemble de N tâches sur M machines distribuées en parallèle. Nous définissons $\mathcal{N}$ l'ensemble $[1, ..., N]$. Un seul passage est nécessaire pour compléter chaque tâche. Dans la version traitée dans ce chapitre, nous faisons référence au problème d'ordonnancement avec des machines identiques, où les tâches arrivant à l'atelier ne peuvent pas être ininterrompues. Chaque machine peut traiter au maximum une tâche à

la fois. L'objectif cherché est la minimisation du retard total, qui correspond à la somme des retards dans la réalisation de chacune des tâches ($\sum T_i$).

Toute tâche $i \in \mathcal{N}$ est caractérisée par les paramètres suivants : date de disponibilité (arrivée à l'atelier r_i), temps de traitement (p_i) et date de fin souhaitée (d_i). Nous considérons uniquement les cas où $d_i \geq r_i + p_i$.

Toutes les données sont considérées connues à l'avance, et le comportement du système est dit stable (cas déterministe).

A propos des machines, nous considérons que leurs performances restent constantes dans le temps, et que les périodes de maintenance préventive peuvent être modélisés avec des tâches supplémentaires. Par conséquent, aucune période d'indisponibilité n'est considérée.

Dans ce problème d'ordonnancement, les tâches sont indépendantes : leurs paramètres r_i, p_i et d_i ne dépendent pas des autres tâches, ni des machines (machines identiques).

Ce problème est considéré comme NP-difficile, principalement parce qu'il est une généralisation du problème avec une seule machine, lequel a été prouvé NP-difficile par Koulamas en 1994 [65]. Nous utilisons des instances où $M < N$, car, dans le cas contraire, le problème est résolu de manière optimal sans aucune difficulté.

Les notations suivantes sont utilisées pour formuler le problème :

- M : Nombre de machines disponibles
- N : Nombre de tâches à ordonnancer
- r_i : Date de disponibilité de la tâche i. Aucune tâche ne peut être assignée à un machine avant cette date.
- p_i : Temps de traitement. Correspond au temps requis pour effectuer la tâche.
- d_i : Date de fin souhaitée. Les pénalités pour dépasser cette date sont identiques quelles que soient les tâches et elles sont proportionnelles au retard.
- C_i : Date de fin effective.
- T_i : Retard de la tâche i

Les valeurs pour les paramètres M et N sont, en toute logique, des entiers positifs. Pour les valeurs de r_i, p_i et d_i, nous supposons, sans perte de généralité, qu'ils prennent des valeurs entières et positives. Le retard d'une tâche est défini comme le temps passé après la date de fin souhaitée avant la fin de la tâche (3.1) :

$$T_i = max\,(C_i - d_i; 0) \tag{3.1}$$

L'objectif choisi pour ce problème ne considère pas les pénalités pour des tâches finies en avance. Dans le contexte industriel expliqué, les résultats des analyses du LPC fournis en avance sont souhaitables, mais pas encouragés.

Une solution complète pour une instance quelconque de la problématique décrite ci-dessus, identifié les M tâches à traiter en premier sur les machines, et leurs successeurs correspondants. Les dates d'exécution et de fin pour chaque tâche sont aussi fournies. Cela simplifie le suivi et la maîtrise des flux du système analysé.

Dans la section suivante, un modèle mathématique est utilisé pour décrire ce problème d'ordonnancement.

3.3 Modélisation mathématique : Modèle avec variables mixtes

Le modèle présenté dans cette section reprend des caractéristiques utilisées par [23] et [140]. Étant donné que notre modèle contient deux types de variables, il est considéré du type MIP (Mixed Integer Problem). Des variables binaires $x_{i,j}$ sont utilisées pour les décisions d'affectation afin de définir si une tâche j est ordonnancée immédiatement après une tâche i sur la même machine. Ces variables de décision sont définies comme suit :

$$x_{i,j} = \begin{cases} 1 & \text{Si la tâche } i \text{ précède immédiatement la tâche } j \text{ sur la même machine} \\ 0 & \text{Sinon} \end{cases}$$

Où $i, j \in \mathcal{N}$. Les machines, étant identiques, les décisions d'affectations concernent l'identification des tâches successives et non la machine sur laquelle elles seront réalisées.
Le deuxième type de variables sont les variables relatives aux temps. Ainsi, des variables entières sont utilisées pour représenter les dates de fin (C_i) et les retards (T_i). Nous utilisons des variables entières car toutes les caractéristiques sont supposées exprimées par des entiers positifs.

Afin de réduire le nombre de variables générées, nous utilisons deux variables supplémentaires pour simuler le début et la fin du planning d'une machine donnée. De cette manière, le début du planning pour une machine quelconque est signalé par une tâche 0. La fin est identifié par une tâche $N + 1$. Le nombre de tâches supplémentaires est contrôlé avec les contraintes considérées dans le modèle.

$$minZ = \sum_{i=1}^{N} T_i \tag{3.2}$$

sous contraintes :

$$\sum_{i=1}^{N} x_{0,i} \leq M \tag{3.3}$$

$$\sum_{i=1}^{N} x_{i,N+1} \leq M \tag{3.4}$$

$$\sum_{i\neq h, i=0}^{N} x_{i,h} = 1 \qquad \forall h \in \mathcal{N} \tag{3.5}$$

$$\sum_{i\neq h, i=1}^{N+1} x_{h,i} = 1 \qquad \forall h \in \mathcal{N} \tag{3.6}$$

$$C_i \geq p_i \times x_{0,i} \qquad \forall i \in \mathcal{N} \tag{3.7}$$

$$C_h \geq C_i + p_h - R\,(1 - x_{i,h}) \qquad \forall i, h \in \mathcal{N}; \quad i \neq h \tag{3.8}$$

$$T_i \geq C_i - d_i \qquad \forall i \in \mathcal{N} \tag{3.9}$$

$$x_{i,h} + x_{h,i} \leq 1 \qquad \forall i, h \in \mathcal{N}; \quad i \neq h \tag{3.10}$$

$$C_i \geq r_i + p_i; \quad \forall i \in \mathcal{N} \tag{3.11}$$

$$C_i, T_i \geq 0; \quad \forall i \in \mathcal{N} \tag{3.12}$$

$$x_{i,h} \in \{0,1\} \qquad \forall i \in \{0 \cup \mathcal{N}\}\,; \quad \forall h \in \{\mathcal{N} \cup N+1\}\,; \quad i \neq h \tag{3.13}$$

Dans ce modèle la fonction (3.2) exprime l'objectif du problème traité dans ce chapitre, la minimisation du retard total. Cette fonction correspond à la somme des retards des N tâches. La contrainte (3.3) garantit qu'une seule tâche est placée en première position du planning de chaque machine. Cela correspond aux tâches successeurs de la tâche supplémentaire 0. De manière similaire, la contrainte (3.4) limite le nombre de tâches en dernière position des différentes machines, juste avant la tâche supplémentaire $N + 1$. Ces deux contraintes limitent le nombre de machines utilisées. Les contraintes (3.5) et (3.6) sont des contraintes d'unicité des successeurs et prédécesseurs de chaque tâche h. Ces deux fonctions garantissent des solutions complètes. Afin de calculer les valeurs pour les dates de fin C_i, deux fonctions sont proposées. La contrainte (3.7) est utilisée pour calculer les dates de fin quand la tâche i est placée en première position (après la tâche supplémentaire 0). La contrainte (3.8) limite les valeurs concernant les dates de fin pour des tâches dans toute autre position. Pour cela, cette fonction utilise une valeur R, qui correspond à une valeur suffisamment grande. Pour les valeurs des variables relatives au retard, pour une tâche i, la contrainte (3.9) limite sa valeur à la différence entre la date de fin C_i et la date de fin souhaitée d_i. Cette fonction est complétée avec la définition des variables du retard, pour considérer uniquement les cas avec du retard positif. Afin d'éviter les sous-tours, la contrainte (3.10) empêche qu'une tâche soit successeur et prédécesseur de la même tâche. La contrainte (3.11) inclut la valeur des

dates de disponibilité r_i pour limiter les valeurs des dates de fin (C_i) : la date de fin d'une tâche i ne peut pas être inférieure à la somme de sa date de disponibilité et le temps de traitement. Les deux derniers contraintes (3.12) et (3.13) définissent les types de variables pour les variables de décision $x_{i,j}$, C_i et T_i.

Ce modèle intègre un total de $2 + 6N + (N+1)^2$ contraintes, $(N+1)^2$ variables binaires et $2N$ variables standard.

Cette représentation mathématique peut être résolue en utilisant tout logiciel dédié à la résolution des problèmes du type MIP («Mixed Integer Problem»). Comme pour tout problème NP-difficile, les méthodes de résolution utilisant la programmation linéaire garantissent l'optimalité pour des instances de petites tailles, avec des temps de calcul acceptables, mais cette durée explose avec des instances de grandes tailles. Dans la section des résultats expérimentaux (section 3.6), ce modèle mathématique est utilisé dans une méthode de programmation linéaire, afin de résoudre le problème traité dans ce chapitre. Pour cela, nous avons employé le logiciel CPLEX d'ILOG-IBM, lequel a montré des bonnes performances dans la résolution de problèmes linéaires de type MIP [64].

3.4 Classification des tâches : Les trois classes de tâches

La résolution des problèmes d'ordonnancement comme celui décrit dans ce chapitre s'effectue souvent par des méthodes approchées principalement à cause de sa complexité. En effet, le temps nécessaire pour trouver une solution optimale augmente de manière non-polynomiale par rapport à la taille de l'instance à résoudre. Cela augmente le nombre de variables utilisées ainsi que la mémoire requise et les calculs nécessaires [59].

Les caractéristiques d'une instance du problème d'ordonnancement traité dans ce chapitre permettent d'obtenir des estimations de la difficulté à trouver des solutions de qualité suffisante. Certains auteurs ont cherché à caractériser les instances par leur réseaux d'activité associé [37], [105], [38].

Afin d'établir une classification des instances par rapport aux caractéristiques des tâches et machines et ainsi établir un lien avec la politique des retards, nous proposons d'utiliser 3 classes pour identifier les tâches. Nous appelons politique des retards (ou de difficulté globale) la distribution constatée dans les données qui définissent si en moyenne les tâches d'une instance quelconque seront ou non en retard. La composition des classes est en relation avec cette politique de risque parce qu'elles sont définies par rapport à la valeur probable du retard.

La classification proposée dans cette section est dynamique : les tâches sont assignées aux classes en fonction d'un ordre établi Φ. Cette classification a été utilisée pour créer les méthodes heuristiques dans la section 3.5.2.2. Ainsi, en utilisant des thèses ou propositions développées pour problèmes avec différents fonctions objectives, nous cherchons à obtenir le meilleur comportement à l'intérieur de chaque classe. Une classification similaire a été proposée par Baptiste *et al.* [12] pour le problème avec une seule machine disponible.

Pour définir cette classification, nous identifions avec $m*$ la machine qui est disponible en premier, après avoir ordonnancé un ensemble Θ qui ne contient pas la totalité de tâches de l'ensemble $\mathcal{N}$. On note t le premier instant où la machine $m*$ est disponible. Pour les tâches restantes Γ, ($\Gamma \notin \Theta$), elles seront assignées dans les trois classes suivantes :

- $\gamma_{t \leq r_I}$: Ensemble de tâches non affectées avec $r_i > t$
- $\gamma_{t \leq d_I - p_I}$: Ensemble de tâches non affectées où $t + p_i \leq d_i$
- $\gamma_{t > d_I - p_I}$: Ensemble de tâches non affectées où $t + p_i > d_i$

La première classe $\gamma_{t \leq r_I}$ est composée des tâches de l'ensemble Γ dont la date de disponibilité est plus grande que t. Cette classe contient (pour un ordre Φ) des tâches qui seront traitées et finies avant la date de fin souhaitée, ou exactement à cette date. Par conséquent, aucun retard n'est considéré pour les tâches dans cette classe. D'autre part, si une tâche $i \in \gamma_{t \leq r_I}$ est affectée à la machine $m*$ à l'instant t, le temps d'inactivité sur cette machine peut être augmenté.

La classe $\gamma_{t \leq d_I - p_I}$ contient les tâches qui ont des dates de disponibilité inférieures à la disponibilité de la machine $m*$ ($r_i < t$), mais qui ne seront pas en retard ($t + p_i \leq d_i$). Pour cette classe, ainsi que pour la classe $\gamma_{t \leq r_I}$, nous pouvons déduire une caractéristique pour toute tâche i affectée à la machine $m*$ à l'instant t. Le temps d'attente : $\delta_i = t - r_i$.

La dernière classe $\gamma_{t > d_I - p_I}$ correspond aux tâches qui seront très probablement en retard à condition d'être ordonnancées d'après Φ.

Nous utilisons les classes γ dans la section 3.5.2.2 pour expliquer la construction des solutions en utilisant un ensemble de règles de priorité.

La section suivante présente l'ensemble de méthodes de résolution testées dans ce chapitre.

3.5 Méthodes de résolution

Dans cette section, nous présentons les algorithmes adaptés à la résolution du problème d'ordonnancement de machines parallèles avec des dates de disponibilité de tâches différents. D'abord, nous présentons la méthode exacte utilisée : la résolution du problème linéaire.

Nous présentons ensuite un ensemble de méthodes approchées, à commencer par la généralisation des algorithmes de liste classiques. Ensuite, des méthodes de solution dynamiques construites sur la base de la classification de la section 3.4 sont décrites. Ces méthodes, grâce à des temps d'exécution réduits, sont utilisées comme points de départ pour les méthodes métaheuristiques présentées à la fin de cette section.

Ces méthodes sont évaluées dans une section ultérieure.

3.5.1 Méthode exacte

Dans cette sous-section, une méthode de résolution exacte est présentée. Il s'agit de la résolution du modèle mathématique présenté dans une section précédente. La résolution par des méthodes exactes permet de trouver des solutions optimales du problème étudié. Par comparaison, et sur certaines conditions (voir section 3.7), elles permettent aussi d'estimer la performance des méthodes approchées sur des instances de petites tailles.

Le modèle mathématique proposé dans la section 3.3 a été conçu avec des variables entières et réelles. Ce type de modèle est basé sur la programmation linéaire à variables mixtes (MIP Mixed Integer Problem). Des expérimentations peuvent être effectuées sur des solveurs commerciaux tels que CPLEX de ILOG-IBM. Nous avons testé ce modèle. Les résultats sont résumés dans la section 3.6.

La résolution du problème $P_m/r_i/\sum T_i$, nécessite de construire un total de M séquences de tâches, et d'en déduire les structures de données pour les valeurs C_i et T_i. Les méthodes exactes permettent de déterminer les solutions exactes des problèmes qui nous intéressent dans une durée d'exécution longue. Les bonnes performances obtenues sur des instances théoriques ne permettent pas la résolution efficace des problèmes réels.

La résolution des instances testées dans le logiciel CPLEX a été limitée à 1800 secondes. Comme tout autre logiciel pour la résolution de problèmes combinatoires, garantir l'optimalité de certaines instances n'est pas réalisable dans des temps raisonnables pour les problèmes classifiés comme NP-difficiles. Malgré cela, ces logiciels sont capables de fournir des solutions de très bonne performance, sans pour autant garantir que la solution optimale ait été trouvée.

Ces modèles ont été validées par des instances résolues avec des algorithmes d'énumération complète afin de vérifier l'optimalité des solutions fournies par le logiciel CPLEX.

3.5.2 Les méthodes heuristiques

Autre que proposer des solutions répondant à l'objectif de minimiser la somme des retards, les méthodes heuristiques sont capables de suivre le comportement dynamique des problématiques industrielles : la taille des problèmes est souvent trop grande, et les données varient facilement. Ceci nécessite d'une résolution plus rapide des problèmes. C'est pourquoi nous nous sommes tournés vers les méthodes approchées.

Le cas industriel décrit dans le chapitre 1 est composé de phases qui ne sont pas complètement standardisées (voir chapitre A pour des problématiques dans la standardisation de procédés). Les durées de certaines analyses physico-chimiques peuvent changer à cause de conditions environnementales non suivies. Sachant que pour l'instant l'information disponible ne permet pas de proposer des modèles stochastiques pour la gestion opérationnelle des équipements, trouver des méthodes de résolution rapides est la priorité.

Les méthodes de résolution proposées dans cette section ont été sélectionnées afin d'explorer l'espace de solutions par section, en commençant par des solutions uniques créées par des méthodes à liste, suivi par des explorations des voisinages proches. Afin de considérer des méthodes permettant d'échapper des optimaux locaux, des métaheuristiques ont été construites autour des recherches locales. Des métaheuristiques classiques sont aussi testées.

3.5.2.1 Algorithmes de liste classiques

Les méthodes de liste adaptées au problème $P_m/r_i/\sum T_i$ sont pour la plupart présentées dans [140]. Nous appelons algorithmes de liste, les algorithmes qui génèrèrent un ordre de priorité unique parmi les tâches à ordonnancer. Ainsi, ces méthodes permettent de construire des programmes pour la gestion des machines avec des listes de priorité qui établissent la séquence de passage des tâches. Afin d'utiliser ce type de méthodes dans la résolution du problèmes avec machines parallèles, les listes de priorité sont accompagnées d'une règle d'affectation :

- Au défaut d'autre mention, toute liste établissant l'ordre de passage des tâches est suivi, en affectant chaque tâche à la première machine disponible

Les méthodes couramment utilisées sont présentées ci-après :

- SPT (*Shortest Processing Time*) : Toute liste de priorité construite en SPT priorise les tâches avec les temps de traitement le plus court. Entre deux jobs (i, j) non ordonnancées, i précède j si $p_i < p_j$ Cette méthode trouve des solutions optimales pour le problème de minimisation de la somme des dates de fin effectives $\sum C_i$.

- EDD (*Earliest Due Date*) : la règle de priorité EDD ordonne les tâches selon leur date de fin souhaitée de manière croissante. La priorité est donnée à la tâche dont la date de fin souhaitée est la plus proche. Pour deux tâches non ordonnancées i et j, i précède j si $d_i < d_j$.
- LPT (*Longest Processing time*) : De manière opposée à la règle SPT, la règle de priorité LPT ordonne les tâches en ordre décroissant des temps de traitement. Ainsi, si les tâches i et j ne sont pas ordonnancées, i précède j si et seulement si $p_j < p_i$.
- $MinSlack$ (*Minimal Slack*) : Les tâches sont ordonnancées par rapport au temps supplémentaire $SL_i = d_i - p_i$. A cette définition nous ajoutons les dates de disponibilité r_i, donc $SL*_i = d_i - p_i - r_i$. Nous rappelons que nous considérons uniquement les cas où $r_i + p_i < d_i$.

3.5.2.2 Méthodes structurales dynamiques

Autre que les méthodes présentées dans la sous-section précédente, d'autres algorithmes de liste ont été adaptées au problème $P_m/r_i/\sum T_i$. Les méthodes décrites dans cette section construisent les solutions de manière dynamique : les tâches sont affectées une à une en suivant des règles qui utilisent le temps de disponibilité après d'avoir affectée la dernière tâche à l'une des machines.

Trois méthodes ont été adaptées dans [140] pour résoudre le problème étudié dans ce chapitre. Ces méthodes sont :

- $PRTT$ (*Priority Rule for Total Tardiness*)
- MDD (*Modified Due Date*)
- $ABHG$ (*Adapted BHG method*)

Dans ce mémoire, des méthodes développées à partir de la classification décrite dans la section 3.4 sont présentées. Elles sont identifiées par les initiales IS car elles peuvent être utilisées comme des solutions initiales (*Initial Solution IS*) pour d'autres méthodes heuristiques ou métaheuristiques. Plusieurs méthodes sont proposées. Chacune est identifiée avec l'index u (IS_u) Nous utilisons les trois classes γ pour expliquer le comportement des méthodes présentées ci-dessous.

Les méthodes IS utilisent la déclaration suivante :

« Méthode IS_u : Pour deux tâches qui ne sont pas encore ordonnancées, si la valeur $IS_u(s,t) \leq IS_u(i,t)$, alors la tâche s doit précéder la tâche i, et elle est affectée à la première machine disponible au moment t. »

L'utilisation de la classification décrite dans la section 3.4, et l'utilisation des méthodes

IS, permettent de tester des règles d'optimalité locales. D'après Emmons [45] pour les problèmes d'ordonnancement sur une seule machine et minimisation du retard totale, sous certaines conditions, des listes créées avec les méthodes SPT et EDD conduisent à des résultats optimaux. Ces conclusions ont été utilisées dans la définition de certaines des méthodes IS.

Comme nous l'avons défini dans la section 3.4, l'instant de première disponibilité de toutes les machines est signalé avec t. Les méthodes IS sont décrites par la suite :

Méthode IS_1

Afin de réduire l'impact des dates de disponibilité non nulles (le début de la période considérée), la méthode IS_1 permet l'introduction des temps d'inactivité entre tâches affectées à la même machine. Ainsi, la méthode IS_1 évalue les tâches dans les classes $\gamma_{t\leq r_i}$ et $\gamma_{t\leq d_i-p_i}$ uniquement par rapport aux dates de fin souhaitées (d_i).

$$IS_1\left(i,t\right) = max\left(max\left(r_i,t\right)+p_i,d_i\right) \tag{3.14}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_1 = d_i \tag{3.15}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t+p_i \leq d_i \rightarrow IS_1 = d_i \tag{3.16}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t+p_i \rightarrow IS_1 = t+p_i \tag{3.17}$$

Méthode IS_2

Au contraire que IS_1, la méthode IS_2 cherche à réduire l'introduction des temps d'inactivité. Pour cela, les tâches disponibles en première ou qui requièrent des temps de traitement bas, sont priorisées. Ainsi, les deux premières classes sont ordonnées exclusivement par la relation entre les valeurs de r_i, p_i et t.

$$IS_2\left(i,t\right) = min\left(max\left(r_i,t+p_i\right),d_i\right) \tag{3.18}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_2 = max\left(r_i,t+p_i\right) \tag{3.19}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t+p_i \leq d_i \rightarrow IS_2 = max\left(r_i,t+p_i\right) \tag{3.20}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t+p_i \rightarrow IS_2 = d_i \tag{3.21}$$

Méthode IS_3

La méthode IS_3 utilise les retards probables. Cette méthode considère que les tâches inscrites dans les classes $\gamma_{t \leq d_i - p_i}$ et $\gamma_{t \geq d_i - p_i}$ seront très probablement en retard. Dans ce cas, ces tâches sont ordonnancées par rapport aux temps de traitement (SPT). Si les données d'une instance vérifient cette considération, l'ordonnancement de ces classes peut être effectué de manière optimale ([66]).

$$IS_3\,(i,t) = max\,(r_i, t) + p_i \tag{3.22}$$

Où :

$$\gamma_{t \leq r_i} : \text{Si } t \leq r_i \rightarrow IS_3 = r_i + p_i \tag{3.23}$$

$$\gamma_{t \leq d_i - p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_3 = t + p_i \tag{3.24}$$

$$\gamma_{t > d_i - p_i} : \text{Si } d_i < t + p_i \rightarrow IS_3 = t + p_i \tag{3.25}$$

Méthode IS_4

Les tâches dans la classe $\gamma_{t \leq r_I}$ sont ordonnancées par rapport à l'ensemble de caractéristiques des instances. Ainsi la priorité est donnée aux tâches avec des dates de disponibilité, des temps de traitement et des temps supplémentaires ($Slack\ Sl_i$) bas. Pour les deux autres classes, cette méthode suppose que les tâches ne seront pas probablement en retard, donc les valeurs des dates de fin souhaitée sont considérées.

$$IS_4\,(i,t) = max\,(r_i, t) + p_i + d_i \tag{3.26}$$

Où :

$$\gamma_{t \leq r_i} : \text{Si } t \leq r_i \rightarrow IS_4 = r_i + p_i + d_i \tag{3.27}$$

$$\gamma_{t \leq d_i - p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_4 = t + p_i + d_i \tag{3.28}$$

$$\gamma_{t > d_i - p_i} : \text{Si } d_i < t + p_i \rightarrow IS_4 = t + p_i + d_i \tag{3.29}$$

Méthode IS_5

Dans la méthode IS_5, les tâches dans la classe $\gamma_{t \leq r_I}$ sont ordonnancées afin de réduire les temps d'inactivité, sauf dans les cas de grande disparité entre les valeurs des dates de fin souhaitée. Pour la classe $\gamma_{t \leq d_i - p_i}$ les tâches sont évaluées par leurs temps de traitement et dates de fin souhaitées. Les tâches qui seront très probablement en retard sont ordonnées en SPT.

$$IS_5\,(i,t) = max\,(r_i, t) + p_i + max\,(d_i - t, 0) \tag{3.30}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_5 = r_i + p_i + d_i - t \tag{3.31}$$

$$\gamma_{t\leq d_i - p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_5 = p_i + d_i \tag{3.32}$$

$$\gamma_{t> d_i - p_i} : \text{Si } d_i < t + p_i \rightarrow IS_5 = t + p_i + max\left(d_i - t, 0\right) \tag{3.33}$$

Méthode IS_6

Cette méthode cherche à réduire les temps d'inactivité et la précocité (tâches finies avant d_i) pour la classe $\gamma_{t\leq r_I}$. Les tâches dans la deuxième classe sont aussi ordonnées pour éviter de finir en avance. Les tâches dans la troisième classe sont ordonnancées en fonction de leur date de fin souhaitée.

$$IS_6\left(i,t\right) = max\left(r_i - t, 0\right) + p_i, max\left(d_i - t, 0\right) \tag{3.34}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_6 = r_i + p_i + d_i + 2t \tag{3.35}$$

$$\gamma_{t\leq d_i - p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_6 = p_i + d_i + t \tag{3.36}$$

$$\gamma_{t> d_i - p_i} : \text{Si } d_i < t + p_i \rightarrow IS_6 = p_i \tag{3.37}$$

Méthode IS_7

De la même manière que la méthode précédente, les tâches dans la première classe sont ordonnancées afin de réduire l'inactivité et la précocité. La dernière classe, contenant des tâches probablement en retard sont priorisées par rapport aux tâches non ordonnancées des autres classes. Cette structure a été proposée afin d'incrémenter la diversité de solutions considérées.

$$IS_7\left(i,t\right) = max\left(r_i - t, 0\right) + max\left(d_i - t, 0\right) \tag{3.38}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_7 = r_i + d_i - 2t \tag{3.39}$$

$$\gamma_{t\leq d_i - p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_7 = d_i - t \tag{3.40}$$

$$\gamma_{t> d_i - p_i} : \text{Si } d_i < t + p_i \rightarrow IS_7 = 0 \tag{3.41}$$

Méthode IS_8

Les classes $\gamma_{t\leq r_i}$ et $\gamma_{t\leq d_i-p_i}$ sont ordonnancées par rapport aux dates de disponibilité. Cela suppose que les tâches ont des dates d'arrivée comprises dans un intervalle réduit. Les tâches dans la dernière classe sont ordonnancées selon l'ordre SPT.

$$IS_8\left(i,t\right) = max\left(r_i - t, 0\right) + p_i + max\left(d_i - p_i - max\left(r_i - t, 0\right), 0\right) \tag{3.42}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_8 = d_i - t \tag{3.43}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_8 = d_i - t \tag{3.44}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t + p_i \rightarrow IS_8 = p_i \tag{3.45}$$

Méthode IS_9

La méthode IS_9 permet l'introduction des temps d'inactivité dans l'ordonnancement des deux premières classes. Les tâches qui sont en retard sont priorisées par rapport aux autres.

$$IS_9\left(i,t\right) = max\left(r_i - t, 0\right) + max\left(d_i - p_i - max\left(r_i - t, 0\right), 0\right) \tag{3.46}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_9 = d_i - p_i - t \tag{3.47}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_9 = d_i - p_i - t \tag{3.48}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t + p_i \rightarrow IS_9 = 0 \tag{3.49}$$

Méthode IS_{10}

Les tâches dans les deux premières classes, sont rangées en priorisant les tâches sans faire d'effort dans la réduction des temps d'inactivité ou la réduction des tâches finies en avance. Les tâches dans la troisième classe sont rangées de manière en ordre croissante du retard courant.

$$IS_{10}\left(i,t\right) = max\left(max\left(r_i, t\right) + p_i - d_i, 0\right) \tag{3.50}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_{10} = 0 \tag{3.51}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_{10} = max\left(t + p_i - d_i, 0\right) \tag{3.52}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t + p_i \rightarrow IS_{10} = t + p_i - d_i \tag{3.53}$$

Méthode IS_{11}

L'objectif de la méthode IS_{11} est de proposer une solution complètement différente aux autres méthodes dites dynamiques. Ainsi, cette méthode ordonnance les tâches de manière décroissante de la date de fin souhaitée. La classe $\gamma_{t\leq r_i}$ inclut l'effet des dates de disponibilité pour éviter les temps d'inactivité trop grands dans le début du programme par machine.

$$IS_{11}(i,t) = max(r_i,t) - d_i \tag{3.54}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_{11} = r_i - d_i \tag{3.55}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_{11} = t - d_i \tag{3.56}$$

$$\gamma_{t\geq d_i-p_i} : \text{Si } d_i < t + p_i \rightarrow IS_{11} = t - d_i \tag{3.57}$$

Méthode IS_{12}

Nous présentons la méthode IS_{12} qui correspond à la méthode $PRTT$ pour le problème avec machines parallèles et dates de disponibilité différentes.

$$IS_{12}(i,t) = max(r_i,t) + max(p_i + max(r_i,t), d_i) \tag{3.58}$$

Où :

$$\gamma_{t\leq r_i} : \text{Si } t \leq r_i \rightarrow IS_{12} = r_i + d_i \tag{3.59}$$

$$\gamma_{t\leq d_i-p_i} : \text{Si } t + p_i \leq d_i \rightarrow IS_{12} = t + d_i \tag{3.60}$$

$$\gamma_{t>d_i-p_i} : \text{Si } d_i < t + p_i \rightarrow IS_{12} = 2t + p_i \tag{3.61}$$

Cette méthode donne une très faible importance aux tâches contenues dans la classe $\gamma_{t>d_i-p_i}$, en lui assignant des valeurs très grandes par rapport aux autres classes. Ainsi, les tâches dans les classes $\gamma_{t\leq r_i}$ et $\gamma_{t\leq d_i-p_i}$ seront ordonnancées en priorité. Cette méthode peut conduire à une sous-utilisation de ressources due à l'impossibilité de traiter une tâche de la classe $\gamma_{t>d_i-p_i}$ avant qu'elle soit en retard.

Toutes les méthodologies présentées ci-dessus garantissent des solutions réalisables pour tout taille valide d'instance ($M < N$). Dans tous les cas, les solutions obtenues sont des listes priorité S, avec autant de places que des tâches à ordonnancer. La position de la tâche dans la liste S détermine le niveau de priorité de la tâche. Chaque liste S doit être implémentée de droit à gauche, avec la première position étant occupée par la première tâche à ordonnancer. Nous présentons un exemple d'utilisation de cette liste dans la figure 3.1. Les données des

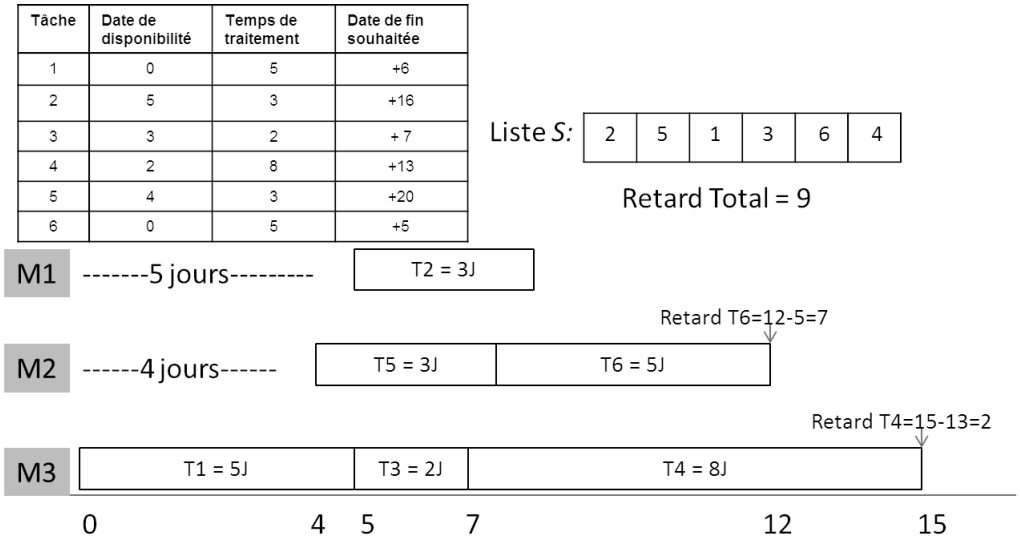

FIGURE 3.1 – Exemple d'utilisation des listes de priorité

tâches sont données. L'exemple est présente un problème d'ordonnancement de 6 tâches et 3 machines.

Une comparaison des performances des solutions obtenues avec les méthodes présentées dans cette section est inclue dans la section 3.6.

La facilité d'implémentation de ces solutions permet d'envisager l'utilisation de ces algorithmes comme des méthodes pour générer des solutions initiales (des points de départ), afin d'effectuer des explorations du type métaheuristique. Certaines des méthodes présentées dans la section suivante, utilisent les listes S produites par les méthodes $IS_{1\leq u\leq 12}$ comme des points de départ.

3.5.3 Les méthodes métaheuristiques

Dans cette partie nous présentons des méthodes de résolution des problèmes combinatoires du type métaheuristiques, qui ont été adaptées pour résoudre le problème traité dans ce chapitre. De la même manière que les méthodes heuristiques et malgré le fait que ces méthodes ne garantissent pas l'optimalité des solutions, ces algorithmes ont l'avantage de donner des solutions de bonne qualité pour des temps de calcul relativement courts.

L'objectif de cette section est de proposer des méthodes de résolution qui améliorent les performances obtenues avec les méthodes présentées dans la section précédente. Parmi

les critères d'évaluation de méthodes de résolution, la priorité est de réduire les indicateurs relatifs aux retards cumulés, même si cela implique une augmentation du temps de calcul, si celui reste raisonnable.

Les méthodes métaheuristiques testées pour le problème traité dans cette section sont :

1. Recherche Locale
2. Algorithme à colonie de fourmis
3. Algorithme génétique
4. Recherche Taboue
5. Recherche locale itérative
6. Recherche locale avec multiples points de départ

3.5.3.1 Recherche Locale (*Local Search* LS)

En utilisant la représentation de solution décrite dans la section précédente, où une liste S décrit les priorités données aux tâches, et dans l'intérêt de chercher des solutions proches (très similaires) à celles crées avec les procédures $IS_{1\leq u\leq 12}$, nous avons défini un ensemble de méthodes de recherche locale. Ces méthodes utilisent l'une des listes S obtenues avec les méthodes $IS_{1\leq u\leq 12}$, pour construire un voisinage en utilisant une fonction spécifique. Chaque nouvelle solution est alors évaluée. La meilleure solution trouvée est gardée comme résultat de la méthode. Les différentes versions présentées ci-dessous, sont des variations de la procédure décrite, où la fonction de voisinage est modifiée. Nous appelons fonction de voisinage aux mouvements permettant de créer des nouvelles solutions à partir d'une solution racine. L'algorithme 1 montre la structure de base utilisée par toutes les versions de recherche local proposées.

Dans le cas du problème résolu dans cette partie, la fonction d'évaluation est le retard total, en utilisant une règle d'affectation assignant chaque tâche dans l'ordre donné dans la liste S à la première machine disponible. Par la suite, nous donnons une description plus détaillée de chaque fonction de voisinage testée.

Meilleure Insertion Totale (MIT)

La première fonction de voisinage testée consiste en un mouvement d'insertion lequel est réalisé pour l'ensemble de tâches. Ainsi, chaque tâche i de la liste S est déplacée vers toute position possible, afin d'obtenir une nouvelle liste S^* à chaque fois. Le mouvement s'exécute jusqu'à que la tâche soit placée dans toutes les positions de la liste S. Chaque liste S^* est

Algorithme 1 Meilleure Insertion

1: $S \leftarrow$ *Solution Initiale* (Liste de priorité)
2: *Solution Locale* = *Fonction d'évaluation* (S)
3: **Pour** chaque $i = 1$ à Tâches **Faire**
4: $S^* \leftarrow$ Fonction de voisinage en $S_{[i]}$
5: **Si** *Fonction d'évaluation* $(S^*) <$ *Solution Locale* **Alors**
6: *Solution Locale* = *Total Retard* S^*
7: $S_{locale} \leftarrow S^*$
8: **Fin Si**
9: **Fin Pour**
10: Meilleure solution $\leftarrow$ *Solution Locale*
11: Configuration obtenue $\leftarrow S_{locale}$

une version de la liste S où seulement la tâche i a changé de position.
La figure 3.2 montre un exemple une liste S de 5 tâches. La solution initiale S ordonnance les tâches dans l'ordre 5-3-1-2-4. La méthode MIT construit 12 nouvelles solutions, en déplaçant une tâche à la fois, vers toutes les positions possibles. La figure 3.2 montre le résultat de l'application de ces mouvement sur la tâche 4. Le résultat de ce mouvement est un ensemble de 4 nouvelles solutions. Une fois toutes les positions possibles pour la tâche 4 sont testées,

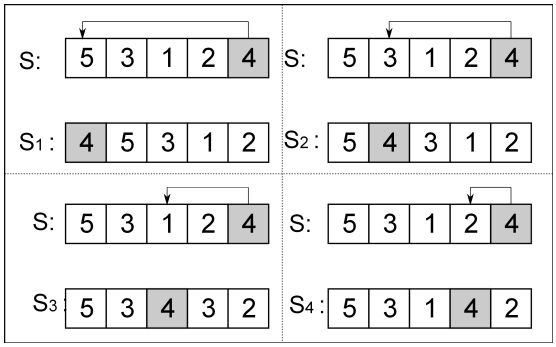

FIGURE 3.2 – *BI 1* : Exemple avec 5 tâches. S représente la solution initiale. $S^1 ... S^4$ sont le voisinage de S.

une nouvelle tâche sera déplacée en partant de la solution initiale $S = 5 - 3 - 1 - 2 - 4$. Au total, le voisinage d'une solution avec la méthode MIT est composé d'un total de $N\,(N-1)$ listes S^*.

Meilleure Insertion Avant (MIA)

La fonction précédente effectue un total de $(N-1)$ mouvements pour obtenir le voisinage de la solution S, pour une tâche. Cela implique un temps de calcul relativement grand sans garantie d'amélioration. Afin de réduire ce temps, nous proposons des méthodes qui limitent les mouvements, en essayant de produire uniquement des solutions qui pourront éventuellement réduire le retard de la liste initiale (S). Pour cela, nous partons sur l'hypothèse que des temps d'inactivité sont inclus dans les programmes des machines. Pour augmenter l'utilisation de ces temps d'inactivité, la fonction de voisinage MIA change les priorités initiales, en avançant des tâches pour les effectuer avant la date de la solution S. Ainsi, pour une tâche dans une position p, les listes S_{p-1}, S_{p-2}, ..., S_0 sont considérées. A chaque fois la tâche est rétrogradée d'une position, jusqu'à être assignée à la première position (qui correspond à la priorité maximale).
Ainsi, au total, cette fonction génère $\frac{N(N-1)}{2}$ listes S^* à partir d'une liste S.

Meilleure Insertion Partielle Guidée ($MAPG$)

Afin de considérer des structures moins rigides, et ainsi augmenter la diversité de solutions pour les méthodes à population comme les algorithmes génétiques, nous proposons cette fonction de voisinage partiel. L'objectif est de construire un voisinage partiel de la solution S, en espérant éviter les optimaux locaux. Pour cela, l'algorithme réduit le cycle de la proposition 3 à une valeur inférieure à N. Pour les tests comparatifs des recherches locales, cette valeur est fixée à $N/2$. Une liste de candidats guide cette méthode afin d'éviter les répétitions. Les mouvements d'insertion s'exécutent sur l'ensemble des positions possibles (de la même manière que la méthode MIT).
Au total, le voisinage d'une solution est composé d'un total de $\frac{N}{2}(N-1)$ listes S^*.

Meilleure Insertion Partielle Avant Guidée ($MIPAG$)

En combinant les méthodes MIA et $MAPG$ nous trouvons une méthode qui effectue une recherche avec un comportement aléatoire pour la sélection des tâches à déplacer, et les positionne en amont. Cela réduit la taille du voisinage et réduit les opérations, avec une probabilité importante de diversification. Avec une limite de $N/2$ tâches déplacées, dans le cas extrême, un total de $\frac{N}{2}\left(\frac{3N}{4}+\frac{1}{2}\right)$ solutions sont testées.

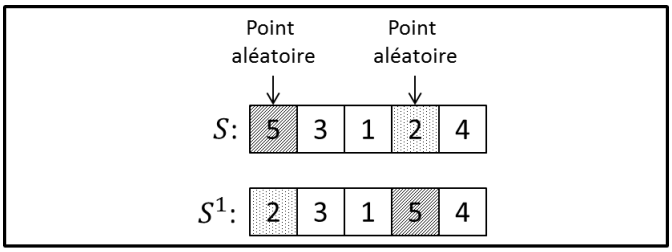

FIGURE 3.3 – Échange aléatoire : Exemple avec 5 tâches, les positions 1 et 4 sont sélectionnées de manière aléatoire

Meilleure Insertion Partielle (MAP)

Une version simplifiée et plus aléatoire que la méthode $MAPG$ est proposée. Cette méthode n'utilise pas la liste de candidats pour la sélection aléatoire des tâches à déplacer. Cela réduit l'espace de solutions testées, mais augmente la probabilité de diversification pour les méthodes de population. Cette méthode équivaut à faire varier le nombre de tâches à déplacer (fixé pour $MAPG$ à $N/2$). Dans le cas extrême, le voisinage aura la même taille que celle décrite dans pour le $MAPG$.

Échange Aléatoire (EA)

Une dernière fonction de voisinage est proposée pour considérer les mouvements d'échange des priorités entre tâches. Cet algorithme sélectionne deux tâches de manière aléatoire et change les tâches contenues. La figure 3.3 montre un exemple d'un mouvement d'échange entre deux tâches dans une liste S contenant 5 tâches. Une solution S^* obtenue avec EA signifierait ne peut pas être trouvée avec aucune des autres recherches locales proposées.

L'ensemble des méthodes de recherche locale génèrent des voisinages dont la structure répond aux besoins de diversification et intensification de certaines méthodes métaheuristiques construites sur des populations de recherche. Ainsi, les différentes versions de recherche locale, et particulièrement les fonctions utilisées pour construire les voisinages, sont utilisées dans les méthodes métaheuristiques présentées ci-dessous.

3.5.3.2 Recherche Locale Itérative (*Iterated Local Search* ILS)

La première méthode testée propose d'effectuer des recherche de manière itérative à partir des listes S des méthodes $IS_{1\leq u\leq 12}$, et appliquer l'une des méthodes de recherche locale. A chaque itération, la solution racine change. Ainsi des nouveaux voisinages sont générés.

Pour définir la fonction de voisinage à utiliser, nous avons utilisé les conclusions de Tang et Lou [127], issues de l'application d'une série de mouvements pour résoudre un problème similaire au problème étudie dans ce chapitre.

L'objectif de la méthode ILS est de tester les deux méthodes LS qui construisent des voisinages correspondants à une recherche intensive complète (voisinages complets) en les couplant avec une méthode de diversification. D'après les descriptions données dans la sous-section 3.5.3.1, les méthodes MIT et EA peuvent être utilisées dans la recherche intensive. Nous appliquons une méthode partielle de type EA pour introduire la diversité voulue. Ainsi, nous proposons donc deux versions de cette méthode.

- $ILS1 \leftarrow$ Intensification par MIT et diversification par EA
- $ILS2 \leftarrow$ Intensification et diversification par EA

Le pseudo algorithme 2 décrit la structure générale de la recherche locale itérative.

Algorithme 2 Recherche Locale Itérative ILS

$S \leftarrow$ *Solution Initiale* (Liste de priorité)
Solution ILS = *Fonction d'évaluation*(S)
Pour chaque $Iteration = 1$ à $MaxIter$; $Iteration++$ **Faire**
  $S^* \leftarrow$ *Fonction Intensification sur* S
  Si *Fonction d'évaluation* $(S^*) <$ *Solution* ILS **Alors**
    Solution ILS = *Fonction d'évaluation* (S^*)
    $S_{ILS} \leftarrow S^*$
  Fin Si
  $S \leftarrow$ *Fonction Diversification sur* (S_{ILS})
Fin Pour
Meilleure solution $\leftarrow$ *Solution* ILS
Configuration obtenue $\leftarrow S_{ILS}$

Ainsi deux voisinages complets sont construits à chaque itération. Le résultat des fonctions de diversification et intensification est la meilleure solution du voisinage construit.
Les fonctions d'intensification et diversification sont définies selon la mise en œuvre choisi. La fonction d'évaluation est le calcul du retard total.

3.5.3.3 Recherche Locale Partielle avec Multiples Départs (*Multi Start Partial Local Search* $MSPLS$)

Les performances des méthodes ILS décrites dans la sous-section précédente ont des temps de calcul croissants avec le nombre de tâches à ordonnancer. Ce comportement pénalise

cette méthode dans la résolution d'instances plus grandes.

Afin de réduire l'impact de la taille de l'instance sur le temps de résolution, nous avons développé une recherche locale partielle, qui a été hybridée avec un algorithme de réinitialisation.
Ainsi, la méthode proposée utilise l'une des fonctions de voisinage, pour générer une partie des solutions possibles, à partir de la solution racine S. Une fois ce voisinage partiel créé, le meilleur élément parmi les solutions S^* du voisinage dévient la solution racine, afin d'effectuer une nouvelle recherche locale partielle. Ce processus est reproduit avec un nombre d'itérations limité. Une condition établi que si le nombre d'itérations dépasse une limite fixée, la méthode prend remplace la solution racine avec la meilleure solution obtenue dès le début de la recherche. Cette réinitialisation permet d'échapper des optimaux locaux. Le pseudo algorithme 3 décrit cette méthode en détail.

Algorithme 3 Recherche Local Partielle avec Multiples Départs

$S \leftarrow$ *Solution Initiale* (Liste de priorité)
Solution MSPLS $=$ *Fonction d'évaluation*(S)
$S_{MSPLS} \leftarrow S$
Pour chaque $Iteration = 1$ à $MaxIteration$; $Iteration++$ **Faire**
  $S^* \leftarrow$ *Meilleur (Voisinage Partiel de S)*
  $S \leftarrow S^*$
  Si *Fonction d'évaluation*$(S) <$ *Solution MSPLS* **Alors**
    Solution MSPLS $=$ *Fonction d'évaluation*(S)
    $S_{MSPLS} \leftarrow S$
    $ItAmelioration \leftarrow Iteration$
  Fin Si
  Si $ItAmelioration >$ *Max Nombre d'itérations sans améliorations* **Alors**
    $S \leftarrow S_{MSPLS}$
  Fin Si
Fin Pour
Meilleure solution $\leftarrow$ *Solution MSPLS*
Configuration obtenue $\leftarrow S_{MSPLS}$

Nous avons développé deux versions de cette métaheuristique, afin de tester les fonctions de voisinage partielle les plus générales. La fonction $MAPG$ génère des voisinages partielles sans avoir des restrictions dans le voisinage créé. La fonction EA peut être implémentée de manière à générer des voisinages partielles et complètes, créés de manière aléatoire. Ainsi, les deux versions de de la méthode $MSPLS$ développées sont :

– $MSPLS1 \leftarrow$ Voisinage créé par $MAPG$
– $MSPLS2 \leftarrow$ Voisinage créé par EA

A part les méthodes issues des recherches locales, nous proposons, par la suite, des méta-heuristiques connues pour leurs performances dans la résolution de problèmes combinatoires.

3.5.3.4 Recherche Taboue

Une méthode inspirée par la recherche taboue a été adaptée afin de résoudre le problème traité dans ce chapitre. L'algorithme proposé combine deux fonctions de voisinage pour explorer les solutions proches d'une solution racine. La meilleure solution créée devient la solution initiale pour construire des nouveaux voisinages dans un cycle itératif.

Afin de considérer des voisinages complets, et en suivant la même logique que pour les structures ILS, les fonctions voisinage choisies sont MIT et EA. Les interactions entre ces deux fonctions dépendent de leur position dans l'algorithme. Nous proposons trois structures de cette méthode. La figure 3.4 montre une représentation de ces structures.

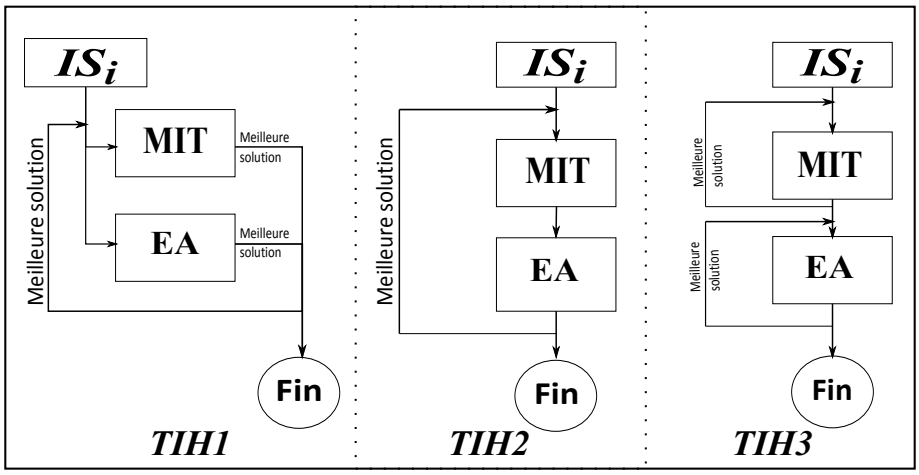

FIGURE 3.4 – Structures de la recherche Tabu

Méthode Tabu 1 $TIH1$

La première structure est détaillée dans l'algorithme 4. Cette méthode utilise l'une des solutions obtenues avec les méthodes IS_u comme racine pour créer un voisinage avec les fonctions MIT et EA. Le voisinage créé est directement liée à la solution souche : les mouvements utilisent la même racine. La meilleure solution de ce voisinage est utilisée dans la génération suivante comme solution initiale. La fonction objectif reste le calcul du retard total.

Algorithme 4 : Tabu Inspired Heuristic TIH

$S \leftarrow$ *Solution Initiale* (Liste de priorité)
Solution TIH = *Fonction d'évaluation* (S)
Solution Locale $= \infty$
Pour chaque $Iteration = 1$ à $MaxIter$; $Iteration++$ **Faire**
  Pour chaque $Nei = 1$ à *Taille voisinage* ; $Nei++$ **Faire**
    $i \leftarrow$ *Tâche choisie aléatoirement, non incluse dans la liste Taboue*
    $S^* \leftarrow$ *Fonction Intensification sur* S
    Si *Fonction d'évaluation* $(S^*) <$ *Solution Locale* **Alors**
      Solution Locale = *Fonction d'évaluation* (S^*)
      $S_{locale} \leftarrow S^*$
      Si *Solution Locale* $<$ *Solution TIH* **Alors**
        $S_{TIH} \leftarrow S_{locale}$
        Solution TIH $\leftarrow$ *Solution Locale*
      Fin Si
    Fin Si
    $j, k \leftarrow$ *Tâches choisies aléatoirement, non inclues dans la liste Taboue*
    $S^* \leftarrow$ *Fonction de diversification de* i, j *sur* S
    Si Fonction d'évaluation $(S^*) <$ *Solution TIH* **Alors**
      Solution Locale = *Fonction d'évaluation* (S^*)
      $S_{locale} \leftarrow S^*$
      Si *Solution Locale* $<$ *Solution TIH* **Alors**
        $S_{TIH} \leftarrow S_{locale}$
        Solution TIH $\leftarrow$ *Solution Locale*
      Fin Si
    Fin Si
  Fin Pour
  $S \leftarrow S_{local}$
Fin Pour
Meilleure solution $\leftarrow$ *Solution TIH*
Configuration obtenue $\leftarrow S_{TIH}$

Méthode Tabu 2 $TIH2$

L'algorithme proposé pour $TIH1$ est modifié pour proposer la deuxième version de cet algorithme. Cette structure construit un voisinage de la solution initiale seulement à partir de la fonction MIT. La meilleure solution de ce voisinage est utilisée comme départ pour créer des solutions avec EA. La meilleure solution de ce dernier groupe est utilisée comme solution de départ pour recommencer l'algorithme.

Méthode Tabu 3 $TIH3$

Cette version de la recherche Tabu explore un voisinage plus large de la solution racine. Deux voisinages sont créés en série. Le premier est construit uniquement avec la fonction MIT, où des itérations sont considérées pour effectuer une recherche similaire à celle proposée dans l'algorithme $ILS1$, dans le facteur de diversification. La meilleure solution est utilisée pour créer un voisinage entier avec EA. Le résultat de ce dernier est la réponse de cette méthode.

Les trois méthodes utilisent des listes $Tabu$ pour échapper aux optimaux locaux. Ces listes dépendent des fonctions de voisinage utilisées. Ainsi, pour la fonction MIT une liste de candidats est utilisée, tandis qu'une matrice $N*N$ contient la liste $Taboue$ pour la fonction EA.

En générale, pour déterminer la durée maximale d'inclusion d'un élément sur la liste, nous utilisons l'expression suivante 3.62 :

$$Tabutenure = \frac{N}{2M} \tag{3.62}$$

Ainsi, la sélection des tâches non inclues dans la liste n'est pas pénalisée par le nombre de mouvements tabous.

3.5.3.5 Algorithmes Génétiques GA

Les algorithmes génétiques sont souvent utilisés dans la résolution heuristique des problèmes combinatoires. Nous avons employé cette méthode pour résoudre le problème d'ordonnancement de machines parallèles.

Cette méthode est basée sur la génération de populations et la sélection naturelle. Des opérations de croisement et de mutation sont effectuées afin d'extraire les gènes qui caractérisent les aspects performants d'une solution quelconque. Dans cette métaheuristique, les chromosomes sont des solutions complètes, et les gènes sont les tâches avec les priorités assignées. Certains composants de base sont détaillés par la suite :

Représentation de la solution

Des structures homogènes doivent être adaptées pour effectuer les différentes opérations nécessaires pour la sélection, les croisements et les mutations. Des structures vectorielles sont utilisées de manière similaire que les listes S définis précédemment.

Population Initiale

Étant un algorithme de population, cette méthode utilise un ensemble de solutions pour effectuer la sélection des parents pour construire la génération suivant. Cette population est initialisée avec des méthodes de construction simple pour réduire l'impact sur les temps de exécution. La qualité de cette population initiale est décisive pour les performances des résultats obtenus à la fin de l'algorithme. Dans la version adaptée ici, nous avons utilisé l'ensemble de méthodes IS_u présentées dans la section 3.5.2.2, en ajoutant des nouveaux chromosomes issus d'une construction complètement aléatoire.

Croisement et mutation

Les opérations utilisées pour chercher des nouvelles solutions sont exécutées sur deux principes : le croisement et la mutation. Le croisement suppose qu'en accouplant deux bonnes solutions, le chromosome résultat peut avoir des performances supérieures à celles de ces parents. La mutation est une source de diversité dans la progéniture. Un opérateur est défini pour sélectionner les parents, pour effectuer les croisements et les mutations.

La sélection des parents s'effectue sur le principe de la roulette, où chaque chromosome de la génération parent a une probabilité de sélection relative à sa fonction d'évaluation. Dans le cas du retard total, cette probabilité est inversement proportionnelle au retard total obtenu. Un processus aléatoire sélectionne avec les probabilités données, les parents à utiliser.

Le croisement s'effectue sur un seul point, afin d'augmenter les chances d'obtenir des solutions valides. En effet, si les parents sont décomposés en plusieurs parties, il est peu probable que la solution obtenue soit réalisable. Néanmoins, un processus de correction est défini pour proposer des solutions valides, avec la conséquence d'une possible perte des gènes des parents.

La mutation est définie sur un intervalle sélectionné de manière aléatoire. Cet intervalle correspond à des points aléatoirement sélectionnés, entre lesquels les gènes seront inversés. Cela implique des changements dans la priorité pour traiter les tâches, sans forcément engendrer des solutions performantes.

Deux critères d'arrêt sont définis : le premier dépend du nombre d'itérations sans amélioration, et le deuxième correspond au nombre total d'itérations minimal avant arrêt.

3.5.3.6 Algorithme de colonies des fourmis *ACO*

Une méthodologie souvent utilisée dans la résolution des problèmes combinatoires est l'algorithme de colonies des fourmis. Problèmes classiques de la littérature comme le TSP (Travelling Salesman Problem) et d'autres problèmes d'ordonnancement ont été traités avec cette méthode métaheuristique.

La principale caractéristique de cette méthode est l'utilisation d'un graphe des nœuds et arcs représentant, dans le cas du problème traité dans ce chapitre, les tâches et les relations de précédence entre elles. Cette représentation est utilisée pour construire les différents chemins trouvés par les fourmis qui le parcourent. Des phéromones sont déposées sur les arcs de ce graphe pour inciter les fourmis à préférer un chemin sur un autre. Les arcs courts concentrent plus de phéromones que les arcs larges. Cette notion de distance est introduite avec la visibilité. Pour un arc entre deux nœuds (tâches), $\tau_{a,b}$ symbolise les phéromones déposées sur l'arc entre les tâches a et b, tandis que $\eta_{a,b}$ représente la visibilité. La figure 3.5 montre un exemple avec 4 tâches. Le chemin où les phéromones est le plus concentré, correspond à la solution du problème.

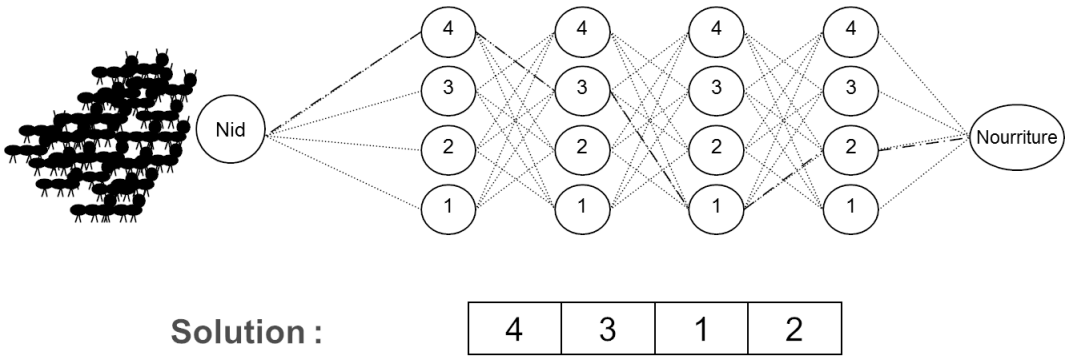

FIGURE 3.5 – Colonie de fourmis : Exemple avec 4 tâches

La probabilité de sélection des nœuds pour une fourmi k est donnée par un système à deux phases tel qu'il a été décrit par Dorigo *et al.* [42], avec les expressions 3.63 et 3.64, où $Succ\,(a)$ est l'ensemble de possibles successeurs d'une tâche a, et α et β les paramètres de contrôle des phéromones et de la visibilité.

$$b = \left\{ max_{b \in Succ(a)} \left\{ \tau_{a,b}^{\alpha} \times \eta_{a,b}^{\beta} \right\} \quad \text{si} \quad \phi \leq \phi_0; \right. \tag{3.63}$$

$$p_{a,b}^{k}(t) = \begin{cases} \frac{\tau_{a,b}^{\alpha}(t) \times \eta_{a,b}^{\beta}}{\sum_{b \in Succ(a)} \tau_{a,b}^{\alpha}(t) \times \eta_{a,b}^{\beta}}, & \text{si}(b \in Succ(a)) \quad || \quad (\phi > \phi_0); \\ 0, & \text{sinon} \end{cases} \tag{3.64}$$

Lorsqu'une fourmi construit un chemin complet, la concentration de phéromone sur les arcs empruntés est modifiée avec l'expression de décadence 3.65, où θ est un paramètre avec des valeurs entre 0 et 1, et η_0 est la concentration initiale de phéromone sur l'arc.

$$\tau_{a,b} = (1 - \theta)\tau_{a,b} + \theta\tau_0 \tag{3.65}$$

Une mise à jour globale des phéromones est exécutée lorsque toutes les fourmis ont fini leurs chemins d'après l'expression 3.66, où $\Delta\tau_{a,b}^{k} = \frac{1}{TT_k}$ avec TT_k le retard total trouvé avec le chemin construit par la fourmi k.

$$\tau_{a,b} = (1 - \lambda)\tau_{a,b} + \lambda \sum_{k=1}^{N} \Delta\tau_{a,b}^{k} \tag{3.66}$$

λ est le paramètre d'évaporation globale de la méthode compris entre 0 et 1, utilisé pour augmenter les probabilités de diversité.

3.6 Résultats expérimentaux

Cette partie présente les résultats de l'application numérique des méthodes décrites dans ce chapitre, sur des instances théoriques et réelles du problème d'ordonnancement dans un atelier composé d'un ensemble de machines disposées de manière parallèle. Cette section se décline sur plusieurs parties. La sous-section 3.6.1 présente les conditions expérimentales données, lesquelles ont été fixées pour tous les tests effectués. Un comparaison des performances des méthodes structurales dynamiques (méthodes dans la section 3.5.2.2) est donnée dans la sous-section 3.6.3. Une première évaluation de méthodes métaheuristiques construites sur des explorations de voisinage (recherche local LS) est réalisée. Ce test a pour objectif de comparer les différentes fonctions de voisinage présentées.

Les meilleures méthodes issues des comparaisons précédentes sont confrontées aux méthodes de recherches locales itérative ILS, et les méthodes $MSPLS$ et TIH. Finalement

TABLE 3.1 – Groupes $\alpha; \beta$

		β_1	β_2	β_3	β_4	β_5
		0,02	**0,27**	**0,52**	**0,77**	**1,02**
α_1	**0,2**	1	2	3	4	5
α_2	**0,7**	6	7	8	9	10
α_3	**1,2**	11	12	13	14	15
α_4	**1,7**	16	17	18	19	20
α_5	**2,2**	21	22	23	24	25

les résultats sont comparés aux performances obtenues des méthodes métaheuristiques classiques GA et ACO.

Dans une section ultérieure, une analyse approfondie est réalisée sur l'ensemble de résultats obtenus pour tous les tests décrits, et sa relation avec l'indicateur N/M.

Des tests effectués sur les données réelles sont présentés à la fin de ce chapitre, en incluant les conclusions des deux sections principales de ce chapitre.

3.6.1 Conditions expérimentales

Toutes les méthodologies développées sont testées sur des conditions identiques. Les méthodes heuristiques ont été programmées en langage C++, et testées sur un ensemble d'instances créées sur le modèle présenté en [140].

Tel qu'il est décrit par l'auteur, les temps de traitement sont distribués de manière uniforme dans l'intervalle $[0, 100]$. Nous définissons pour une instance donnée, la valeur TPr comme la somme des temps de traitement de l'ensemble de tâches qui composent l'instance. Cette valeur est utilisée pour le calcul des temps de disponibilité et date de fin souhaitée des tâches, données dans les équations 3.67 et 3.68.

$$r_i \in [0, \alpha \times TPr/M] \tag{3.67}$$

$$d_i \in [r_i + p_i, (r_i + p_i) + \beta \times TPr/M] \tag{3.68}$$

Les paramètres α et β ont été définis en [140]. La combinaison des valeurs de ces paramètres établie la difficulté de l'instance. Les valeurs testées pour ces paramètres sont données dans le tableau 3.1. Une nomenclature est aussi incluse. Désormais, les nombres entiers du tableau représentent chaque couple, qui nous appellerons groupe. Ainsi, le groupe 5 fait référence au groupe d'instances créées avec $\alpha = 0.2$ et $\beta = 1.02$. Un total de 25 groupes est considéré.

TABLE 3.2 – Temps d'exécution méthode exacte

Groupe	$N = 12$ $M = 2$	$N = 12$ $M = 3$	$N = 12$ $M = 5$
1	>1800	>1800	558,514
2	>1800	>1800	62,032
3	>1800	>1800	182,428
4	>1800	>1800	55,602
5	1,396	0,246	0,09
6	>1800	>1800	13,832
7	>1800	>1800	149,414
8	>1800	>1800	8,416
9	>1800	437,5	1,716
10	0,844	0,13	0,106
11	>1800	>1800	6,08
12	>1800	>1800	2,526
13	>1800	167,312	2,694
14	>1800	111,13	1,078
15	14,988	0,362	0,084
16	23,11	3,686	0,288
17	40,698	2,674	0,134
18	89,336	6,374	0,464
19	8,128	0,808	0,132
20	1,198	0,098	0,066
21	2,132	0,252	0,074
22	0,49	0,13	0,07
23	0,324	0,126	0,094
24	0,116	0,326	0,054
25	0,346	0,096	0,06

L'ensemble de méthodes sont testées avec des instances créées pour des problèmes avec 2, 3 et 5 machines, et 12, 40 et 100 tâches.

L'ordinateur utilisée pour les tests possède un système d'exploitation Windows 7, avec un processeur Intel i5 et 4 gigaoctets de mémoire rapide.

Les sous-sections suivantes incluent les résultats des tests effectuées sur les méthodes proposées. Les résultats sont déclinés par type d'heuristique.

3.6.2 Limites de la méthode exacte

Nous avons testé le modèle mathématique développé pour le problème traité dans ce chapitre, présenté dans la section 3.3, sous les conditions décrites ci-dessus. Étant un problème NP-difficile, les performances de la résolution par la méthode de la programmation linéaire sont limitées à des tailles d'instance relativement petites.

Ainsi, avec une limite de temps d'exécution fixée à 1800 secondes, le solveur du logiciel CPLEX d'ILOG-IBM garantie des solutions optimales à certaines instances de 12 tâches.

Les résultats par rapport aux temps d'exécution sont donnés dans le tableau 3.2. Les temps sont données en secondes.

Ces résultats ont permis de confirmer l'impact des groupes définis dans les conditions expérimentales (sous-section 3.6.1), et de la taille de l'instance (valeurs N et M) sur la difficulté de résoudre le problème d'ordonnancement sur machines parallèles.

3.6.3 Comparaison des méthodes structurales dynamiques

Pour ce premier test, nous utilisons les indicateurs de performance décrits dans la section 1.1.2. Les méthodes de résolutions sont comparées par rapport à l'indicateur $RBKS$. L'indicateur GAP est utilisé pour donner un aperçu de la homogénéité des réponses obtenues. Dans cette sous-section nous évaluons les performances des méthodes dites structurales présentées dans la section 3.5.2.2. Dans [140] les avantages de la méthode $ABHG$ par rapport aux méthodes MDD et $PRTT$ sont montrés. Les résultats présentés dans le tableau 3.3 concernent les heuristiques $ABHG, IS_1, ..., IS_{12}$.

Dans le tableau 3.3 nous présentons les résultats pour les meilleures heuristiques.

Les méthodes présentées dans la section 3.5.2.2 et qui sont inclues dans le tableau, sont celles qui obtiennent en moyenne les meilleures solutions connues pour les instances testées (indicateur $RBKS$). De cette manière, la première méthode de chaque groupe est celle qui a obtenu en moyenne, les meilleurs résultats parmi les méthodes testées. Chaque méthode est accompagnée des indicateurs $RBKS$ et GAP. Pour plus de détails, le tableau inclut les méthodes ayant des valeurs $RBKS$ avec une différence inférieure à 20% de la meilleure heuristique, pour chaque groupe.

Ainsi, pour les instances du groupe 1, en moyenne, les heuristiques $ABHG$ et IS_{12} ont obtenu les meilleurs résultats. De cette manière, la méthode $ABHG$ a trouvée 3.40% des meilleures solutions pour l'ensemble d'instances résolues, avec un écart moyen proche de 2.2%. En générale, nous cherchons des valeurs élevées de l'indicateur $RBKS$, accompagnées des valeurs proches du 0% pour l'indicateur GAP. Pour ce groupe, les autres heuristiques ont des indicateurs $RBKS$ à plus de 20% d'écart des méthodes montrées.

Le tableau 3.3 confirme les résultats du [140] par rapport aux performances de la méthode $ABHG$. Pour l'ensemble de groupes, cette méthode trouve des solutions très performantes pour les deux indicateurs de performance affichés. Les méthodes IS_7 et IS_{12} apparaissent comme des méthodes ayant des performances, parfois meilleures que celles obtenues par la méthode $ABHG$.

TABLE 3.3 – Tests numériques sur les méthodes de structure dynamique

Groupe	Method	RBKS	GAP
1	$ABHG$	9,44%	2,20%
	IS_{12}	7,78%	4,53%
2	$ABHG$	3,33%	4,98%
3	$ABHG$	1,67%	11,80%
4	$ABHG$	11,67%	24,16%
5	$ABHG$	71,11%	18,71%
6	IS_{12}	5,56%	18,25%
	IS_7	5,56%	29,25%
	$ABHG$	3,89%	11,71%
7	$ABHG$	5,56%	25,07%
8	$ABHG$	17,78%	64,56%
9	$ABHG$	90,00%	8,57%
10	$ABHG$	98,33%	0,81%
	IS_7	87,22%	12,07%
	IS_{12}	87,22%	12,07%
11	$ABHG$	7,22%	28,24%
	IS_7	6,67%	31,53%
12	$ABHG$	45,00%	44,81%
	IS_7	37,22%	53,03%
	IS_{12}	36,11%	54,02%
13	$ABHG$	88,89%	10,29%
	IS_7	78,33%	20,65%
	IS_{12}	78,33%	20,70%
14	$ABHG$	97,22%	2,52%
	IS_7	92,78%	7,07%
	IS_{12}	92,78%	7,07%
15	$ABHG$	98,89%	1,11%
	IS_7	97,22%	2,78%
	IS_{12}	97,22%	2,78%
	IS_4	91,11%	8,89%
	IS_5	91,11%	8,89%
	IS_6	91,11%	8,89%
	IS_1	86,67%	13,33%
	IS_8	86,67%	13,33%
16	$ABHG$	16,11%	30,84%
	IS_7	13,89%	33,93%
17	IS_{12}	76,67%	19,54%
	IS_7	76,11%	19,17%
	$ABHG$	76,11%	18,15%
18	$ABHG$	96,11%	3,44%
	IS_7	91,67%	8,17%
	IS_{12}	91,67%	8,21%
	IS_4	78,33%	21,40%
	IS_5	78,33%	21,40%
	IS_6	78,33%	21,40%
19	IS_7	98,89%	1,11%
	IS_{12}	98,89%	1,11%
	$ABHG$	98,89%	1,11%
	IS_4	91,67%	7,83%
	IS_5	91,67%	7,83%
	IS_6	91,67%	7,83%
	IS_1	86,67%	12,94%
	IS_8	86,67%	12,94%
20	$ABHG$	100,00%	0,00%
	IS_7	99,44%	0,56%
	IS_{12}	99,44%	0,56%
	IS_4	95,00%	5,00%
	IS_5	95,00%	5,00%
	IS_6	95,00%	5,00%
	IS_1	87,22%	12,78%
	IS_8	87,22%	12,78%
21	IS_7	25,56%	27,47%
	IS_{12}	22,78%	29,99%
	$ABHG$	21,11%	29,68%
22	IS_7	91,67%	6,80%
	IS_{12}	91,67%	6,80%
	$ABHG$	90,00%	8,08%
23	IS_7	97,78%	2,22%
	IS_{12}	97,22%	2,54%
	$ABHG$	96,67%	2,99%
	IS_4	86,67%	12,34%
	IS_5	86,67%	12,34%
	IS_6	86,67%	12,34%
	IS_1	86,67%	12,78%
	IS_8	86,67%	12,78%
24	IS_7	99,44%	0,56%
	IS_{12}	99,44%	0,56%
	$ABHG$	99,44%	0,56%
	IS_1	94,44%	5,56%
	IS_8	94,44%	5,56%
	IS_4	93,89%	6,11%
	IS_5	93,89%	6,11%
	IS_6	93,89%	6,11%
25	IS_7	100,00%	0,00%
	IS_{12}	100,00%	0,00%
	$ABHG$	100,00%	0,00%
	IS_1	98,33%	1,67%
	IS_4	98,33%	1,67%
	IS_5	98,33%	1,67%
	IS_6	98,33%	1,67%
	IS_8	98,33%	1,67%

Des tableaux complets incluant les valeurs $RBKS$ et GAP moyennes pour la totalité de méthodes décrits dans la section 3.5.2.2 sont données dans l'annexe B.

Les temps d'exécution des méthodes testées dans cette sous-section sont donnés dans le tableau 3.4.

Les temps sont donnés en secondes, et ils correspondent aux temps moyens minimales et maximales pour les différents groupes considérés. D'une manière générale, les temps d'exécution des méthodes testées dans cette partie dépendent du nombre de tâches et de machines. Les résultats détaillés par groupe des temps d'exécution sont présentés dans l'annexe B.

Ces performances, étant calculées sur la globalité de résultats, peuvent être comparées par la suite, avec les autres méthodes heuristiques proposées.

3.6.4 Comparaison méthodes de recherche locale LS

Afin de tester les différentes fonctions de voisinage proposées pour effectuer des recherches locales, nous avons implémenté ces fonctions sur une recherche locale avec des répétitions. Ainsi, la meilleure solution d'un voisinage est utilisée comme racine pour construire un nouveau voisinage. Ces méthodes nécessitent d'une solution initiale. Pour inclure les différents types de solution des méthodes de la section 3.5.2.2, les fonctions de voisinage sont testées avec l'ensemble de méthodes $IS_{1\leq u\leq 12}$.

Dans le tableau 3.5 nous présentons les résultats pour chaque groupe défini (tableau 3.1). Les indicateurs $RBKS$ et GAP sont inclus. Les résultats sont données pour les meilleures méthodes par rapport à la valeur de l'indicateur $RBKS$.

Pour simplifier l'affichage des résultats, nous utilisons la nomenclature suivante :

- $LS_{1,0}, ..., LS_{12,0}$: Recherche locale avec MIT, pour $IS_1, ...IS_{12}$
- $LS_{1,1}, ..., LS_{12,1}$: Recherche locale avec MIA, pour $IS_1, ...IS_{12}$
- $LS_{1,2}, ..., LS_{12,2}$: Recherche locale avec $MAPG$, pour $IS_1, ...IS_{12}$
- $LS_{1,3}, ..., LS_{12,3}$: Recherche locale avec $MIPAG$, pour $IS_1, ...IS_{12}$
- $LS_{1,4}, ..., LS_{12,4}$: Recherche locale avec MAP, pour $IS_1, ...IS_{12}$
- $LS_{1,5}, ..., LS_{12,5}$: Recherche locale avec EA, pour $IS_1, ...IS_{12}$

Les résultats dans le tableau 3.5 sont rangées par groupe, affichant les meilleures performances moyennes obtenues. Ainsi, pour le groupe 1, le voisinage créé avec la fonction EA (échanges aléatoires) obtient, en moyenne, la plus grande valeur pour l'indicateur $RBKS$ parmi les méthodes testées dans cette section, en utilisant les solutions initiales IS_9 et IS_{12}. Dans le cas où les performances d'une fonction de voisinage sont identiques pour toute solution initiale donnée, le résultat est indiqué avec $LS_{i,u}$. Ainsi, par exemple pour le groupe 17,

TABLE 3.4 – Temps de calcul des méthodes de structure dynamique. Les temps sont donnés en secondes

	Machines	$N = 12$		$N = 40$		$N = 100$	
		t_{min}	t_{max}	t_{min}	t_{max}	t_{min}	t_{max}
	$M = 2$	0,003	0,090	0,030	0,084	0,161	0,331
IS_1	$M = 3$	0,004	0,014	0,037	0,085	0,201	0,576
	$M = 5$	0,005	0,020	0,050	0,100	0,281	0,369
	$M = 2$	0,003	0,008	0,030	0,083	0,165	0,263
IS_2	$M = 3$	0,004	0,009	0,037	0,080	0,199	0,610
	$M = 5$	0,005	0,016	0,050	0,099	0,294	0,370
	$M = 2$	0,003	0,008	0,030	0,101	0,165	0,354
IS_3	$M = 3$	0,004	0,010	0,037	0,083	0,201	0,595
	$M = 5$	0,005	0,019	0,050	0,103	0,278	0,373
	$M = 2$	0,003	0,024	0,030	0,097	0,162	0,314
IS_4	$M = 3$	0,004	0,012	0,037	0,077	0,200	0,733
	$M = 5$	0,005	0,038	0,050	0,105	0,286	0,362
	$M = 2$	0,003	0,008	0,030	0,130	0,165	0,281
IS_5	$M = 3$	0,004	0,010	0,038	0,067	0,203	0,761
	$M = 5$	0,005	0,023	0,050	0,101	0,281	0,369
	$M = 2$	0,003	0,007	0,030	0,098	0,162	0,318
IS_6	$M = 3$	0,004	0,012	0,037	0,067	0,200	0,828
	$M = 5$	0,005	0,037	0,051	0,084	0,282	0,370
	$M = 2$	0,003	0,008	0,030	0,127	0,162	0,284
IS_7	$M = 3$	0,004	0,010	0,037	0,067	0,201	0,737
	$M = 5$	0,005	0,027	0,051	0,106	0,290	0,347
	$M = 2$	0,003	0,008	0,030	0,175	0,163	0,306
IS_8	$M = 3$	0,004	0,009	0,037	0,077	0,200	0,546
	$M = 5$	0,005	0,022	0,051	0,124	0,279	0,366
	$M = 2$	0,003	0,008	0,030	0,118	0,167	0,331
IS_9	$M = 3$	0,004	0,011	0,037	0,084	0,201	0,692
	$M = 5$	0,005	0,018	0,051	0,096	0,291	0,361
	$M = 2$	0,003	0,026	0,030	0,090	0,172	0,357
IS_{10}	$M = 3$	0,004	0,010	0,037	0,071	0,211	0,670
	$M = 5$	0,005	0,023	0,051	0,087	0,305	0,374
	$M = 2$	0,003	0,007	0,030	0,088	0,174	0,344
IS_{11}	$M = 3$	0,004	0,012	0,038	0,063	0,214	0,618
	$M = 5$	0,005	0,041	0,050	0,080	0,308	1,601
	$M = 2$	0,003	0,008	0,030	0,070	0,162	0,397
IS_{12}	$M = 3$	0,004	0,014	0,037	0,088	0,201	0,549
	$M = 5$	0,005	0,019	0,050	0,092	0,276	0,374
	$M = 2$	0,000	0,001	0,000	0,007	0,007	0,014
$ABHG$	$M = 3$	0,000	0,002	0,001	0,014	0,019	0,032
	$M = 5$	0,000	0,004	0,015	0,026	0,166	0,371

TABLE 3.5 – Tests numériques sur les méthodes de recherche locale

Groupe	Method	RBKS	Gap
1	$LS_{9,5}$	45,00%	0,46%
	$LS_{12,5}$	41,67%	0,30%
2	$LS_{6,5}$	36,67%	0,60%
	$LS_{5,5}$	36,67%	0,61%
	$LS_{12,5}$	36,67%	0,64%
	$LS_{2,5}$	32,22%	0,54%
3	$LS_{5,5}$	36,67%	1,55%
	$LS_{6,5}$	35,56%	1,50%
4	$LS_{9,5}$	37,78%	5,76%
5	$LS_{7,5}$	88,33%	5,87%
	$LS_{4,5}$	87,22%	4,89%
6	$LS_{5,5}$	36,67%	1,59%
	$LS_{7,5}$	36,11%	1,52%
7	$LS_{9,5}$	38,89%	3,32%
	$LS_{2,5}$	35,00%	2,33%
8	$LS_{2,5}$	70,56%	10,46%
	$LS_{3,5}$	70,00%	8,66%
9	$LS_{3,5}$	100,00%	0,00%
	$LS_{1,5}$	99,44%	0,56%
	$LS_{4,5}$	99,44%	0,56%
	$LS_{7,5}$	99,44%	0,56%
	$LS_{8,5}$	99,44%	0,56%
	$LS_{10,5}$	99,44%	0,56%
10	$LS_{i,5}$	100,00%	0,00%
11	$LS_{7,5}$	57,78%	1,21%
12	$LS_{7,5}$	93,89%	3,10%
13	$LS_{12,5}$	99,44%	0,56%
14	$LS_{12,5}$	100,00%	0,00%
15	$LS_{i,5}$	100,00%	0,00%
	$LS_{7,2}$	99,44%	0,56%
	$LS_{7,3}$	99,44%	0,56%
	$LS_{7,4}$	99,44%	0,56%
	$LS_{12,2}$	99,44%	0,56%
	$LS_{12,3}$	99,44%	0,56%
	$LS_{12,4}$	99,44%	0,56%
16	$LS_{6,5}$	83,33%	0,64%
	$LS_{4,5}$	82,78%	0,68%
	$LS_{12,5}$	82,78%	0,85%
	$LS_{8,5}$	80,00%	0,61%
17	$LS_{i,5}$	98,89%	0,77%
18	$LS_{12,5}$	100,00%	0,00%
19	$LS_{7,5}$	100,00%	0,00%
	$LS_{12,5}$	100,00%	0,00%
	$LS_{7,0}$	99,44%	0,56%
	$LS_{7,1}$	99,44%	0,56%
	$LS_{7,2}$	99,44%	0,56%
	$LS_{7,3}$	99,44%	0,56%
	$LS_{7,4}$	99,44%	0,56%
	$LS_{12,0}$	99,44%	0,56%
	$LS_{12,1}$	99,44%	0,56%
	$LS_{12,2}$	99,44%	0,56%
	$LS_{12,3}$	99,44%	0,56%
	$LS_{12,4}$	99,44%	0,56%
20	$LS_{i,5}$	100,00%	0,00%
	IS_{12}	99,44%	0,56%
21	$LS_{4,5}$	94,44%	0,21%
	$LS_{7,5}$	93,33%	0,17%
22	$LS_{1,5}$	100,00%	0,00%
	$LS_{2,5}$	100,00%	0,00%
	$LS_{3,5}$	100,00%	0,00%
	$LS_{7,5}$	100,00%	0,00%
	$LS_{8,5}$	100,00%	0,00%
	$LS_{10,5}$	100,00%	0,00%
	$LS_{12,5}$	100,00%	0,00%
	$LS_{4,5}$	99,44%	0,56%
	$LS_{5,5}$	99,44%	0,56%
	$LS_{6,5}$	99,44%	0,56%
	$LS_{9,5}$	99,44%	0,56%
23	$LS_{i,5}$	100,00%	0,00%
	$LS_{7,2}$	100,00%	0,00%
	$LS_{7,5}$	100,00%	0,00%
	$LS_{7,0}$	99,44%	0,56%
	$LS_{7,1}$	99,44%	0,56%
	$LS_{7,3}$	99,44%	0,56%
	$LS_{12,0}$	99,44%	0,56%
	$LS_{12,1}$	99,44%	0,56%
	$LS_{12,2}$	99,44%	0,22%
24	$LS_{i,5}$	100,00%	0,00%
	IS_7	99,44%	0,56%
	IS_{12}	99,44%	0,56%
25	IS_{12}	100,00%	0,00%
	$LS_{i,5}$	100,00%	0,00%
	$LS_{1,0}$	100,00%	0,00%
	$LS_{1,1}$	100,00%	0,00%
	$LS_{1,2}$	100,00%	0,00%
	$LS_{4,0}$	100,00%	0,00%
	$LS_{4,1}$	100,00%	0,00%
	$LS_{4,2}$	100,00%	0,00%
	$LS_{4,3}$	100,00%	0,00%
	$LS_{5,0}$	100,00%	0,00%
	$LS_{5,1}$	100,00%	0,00%
	$LS_{5,2}$	100,00%	0,00%
	$LS_{5,3}$	100,00%	0,00%
	$LS_{6,0}$	100,00%	0,00%
	$LS_{6,1}$	100,00%	0,00%
	$LS_{6,2}$	100,00%	0,00%
	$LS_{6,3}$	100,00%	0,00%
	IS_7	100,00%	0,00%
	$LS_{8,0}$	100,00%	0,00%
	$LS_{8,1}$	100,00%	0,00%
	$LS_{8,2}$	100,00%	0,00%
	$LS_{1,3}$	99,44%	0,56%
	$LS_{4,4}$	99,44%	0,56%
	$LS_{5,4}$	99,44%	0,56%
	$LS_{6,4}$	99,44%	0,56%
	$LS_{8,3}$	99,44%	0,56%

les voisinages construits utilisant la fonction EA (Échanges aléatoires) obtiennent des performances identiques pour l'ensemble de solutions de départ définies. Afin de faciliter l'affichage des résultats, le tableau 3.5 contient uniquement les résultats pour les méthodes obtenant des $RBKS$ avec un écart inférieur à 1% de la valeur obtenue par la meilleure heuristique.

Pour le problème résolu dans cette section, et avec l'ensemble de solutions initiales données, la méthode de recherche locale avec un voisinage basé sur les échanges entre deux éléments obtient des bonnes performances pour la totalité de groupes testés.

Ces résultats nous permettent de conclure que les méthodes heuristiques d'intensification, comme ceux testées dans cette section, ont des performances limitées dues à la composition des voisinages des solutions initiales $IS_{1\leq u\leq 12}$. Les résultats performants de la fonction EA confirment cette conclusion. La fonction EA inclut des mouvements de diversification qui seraient difficilement atteignables par les autres fonctions de voisinage.

Les temps d'exécution sont donnés dans le tableau 3.6. Les temps sont donnés en secondes. D'après le tableau 3.6, la fonction de voisinage EA a, en moyenne, les temps d'exécution maximales plus élevés pour les instances de petite taille. En revanche, pour les instances de taille plus importante, cette fonction a des temps d'exécution au-dessous des autres méthodes testées.

Le détail des résultats obtenus et des temps d'exécution par groupe et par taille d'instance sont inclus dans l'annexe B.

La sous-section suivante présente des méthodes qui incluent ces deux types d'explorations de l'espace de recherche.

3.6.5 Comparaison métaheuristiques de recherche locale

Cette sous-section présente les résultats des comparaisons effectuées sur l'ensemble de méthodes métaheuristiques. Le tableau suivant inclut les résultats des tests effectués sur les méthodes métaheuristiques TIH, ILS et $MSPLS$.

Les méthodes construites sur les structures $TIHILS$, et $MSPLS$ utilisent un point de départ donné par chacune des solutions $IS_{1\leq u\leq 12}$ de la section 3.5.2.2. Nous utilisons la nomenclature suivante pour simplifier l'affichage des résultats.

- $h_{1,1}, ..., h_{12,1}$: méthodes avec $TIH1$ pour IS_1 à IS_{12}
- $h_{1,2}, ..., h_{12,2}$: méthodes avec $TIH2$ pour IS_1 à IS_{12}
- $h_{1,3}, ..., h_{12,3}$: méthodes avec $TIH3$ pour IS_1 à IS_{12}
- $h_{1,4}, ..., h_{12,4}$: méthodes avec $ILS1$ pour IS_1 à IS_{12}

TABLE 3.6 – Temps de calcul des méthodes de recherche locale. Les temps sont donnés en secondes

	Machines	$N = 12$		$N = 40$		$N = 100$	
		t_{min}	t_{max}	t_{min}	t_{max}	t_{min}	t_{max}
$LS_{i,0}$	$M = 2$	0,000	0,001	0,000	0,004	0,000	0,018
	$M = 3$	0,000	0,001	0,000	0,015	0,000	0,039
	$M = 5$	0,000	0,002	0,000	0,004	0,000	0,052
$LS_{i,1}$	$M = 2$	0,000	0,001	0,000	0,003	0,000	0,012
	$M = 3$	0,000	0,001	0,000	0,003	0,000	0,020
	$M = 5$	0,000	0,001	0,000	0,004	0,000	0,025
$LS_{i,2}$	$M = 2$	0,000	0,001	0,000	0,016	0,000	0,615
	$M = 3$	0,000	0,001	0,000	0,031	0,000	1,032
	$M = 5$	0,000	0,001	0,000	0,032	0,000	1,315
$LS_{i,3}$	$M = 2$	0,000	0,003	0,000	0,032	0,000	0,222
	$M = 3$	0,000	0,001	0,000	0,031	0,000	0,345
	$M = 5$	0,000	0,001	0,000	0,018	0,000	0,544
$LS_{i,4}$	$M = 2$	0,000	0,003	0,000	0,019	0,000	0,484
	$M = 3$	0,000	0,001	0,000	0,021	0,000	0,687
	$M = 5$	0,000	0,003	0,000	0,024	0,000	1,237
$LS_{i,5}$	$M = 2$	0,001	0,014	0,013	0,035	0,072	0,141
	$M = 3$	0,001	0,019	0,015	0,041	0,087	0,176
	$M = 5$	0,002	0,040	0,019	0,061	0,112	0,274

- $h_{1,5}, ..., h_{12,5}$: méthodes avec $ILS2$ pour IS_1 à IS_{12}
- $h_{1,6}, ..., h_{12,6}$: méthodes avec $MSPLS1$ pour IS_1 à IS_{12}
- $h_{1,7}, ..., h_{12,7}$: méthodes avec $MSPLS2$ pour IS_1 à IS_{12}

Nous avons décomposé les résultats en deux tableaux : le tableau 3.7 inclut les résultats pour les 16 premiers groupes. Les autres groupes sont dans le tableau 3.8

Dans les tableaux 3.7 et 3.8, pour chaque groupe, les résultats des méthodes de solution ayant obtenu les meilleures performances sont inclus avec les indicateurs $RBKS$ et GAP.

Dans les tableaux 3.7 et 3.8, si une heuristique i trouve des solutions à performances égales pour toute solution initiale IS_u, elle est signalé avec $h_{u,i}$. De même pour les cas où à partir d'une solution initiale u, toutes les heuristiques ont trouvé des solutions également performantes.

Pour les premiers groupes (1 à 17), d'après les indicateurs $RBKS$ et GAP, les heuristiques $h_{u,1}$, $h_{u,2}$ et $h_{u,7}$ ont trouvé des bonnes solutions pour la plupart des groupes. Les résultats montrent pour les groupes 10, 13, 14 et 15 que l'heuristique $h_{u,7}$ est peu influencée par la solution initiale utilisée.

Le tableau 3.8 confirme les bonnes performances de l'heuristique $h_{u,7}$, laquelle a obtenu des très bonnes performances pour 6 des 8 groupes dans ce tableau. Parmi ces résultats, pour les groupes 18, 19, 20, 22, 24 et 25 cette heuristique ne semble pas être influencée par la solution initiale utilisée. Les instances des groupes inclus dans ce tableau ont été résolues efficacement par un grand nombre de méthodes.

Le tableau 3.9 donne les temps d'exécution moyens pour les méthodes testées dans cette section.

Les temps d'exécution du tableau 3.9 montrent que les temps d'exécution de certaines méthodes sont affectés par l'instance résolue. Ainsi, pour les instances avec $N = 100$ et $M = 2$, les temps d'exécution de l'heuristique $h_{i,1}$ sont entre 0.164 et 0.421, tandis que pour l'heuristique $h_{i,7}$ ces temps sont compris entre $<$0.001 et 1.136. Les différences de temps proviennent essentiellement du groupe de l'instance résolue (voir détail dans l'annexe B).

Pour finir l'analyse des tests expérimentaux, le tableau 3.10 inclut les résultats obtenus par les algorithmes génétiques et l'algorithme de colonie de fourmis, avec les caractéristiques données précédemment.

Dans ce tableau, les performances sont en général en-dessous des performances obtenues avec les métaheuristiques basées sur la recherche locale. Néanmoins, il est intéressante que pour les premiers groupes, la méthode ACO a des performances supérieures à celles obtenues par la méthode AG. Ce comportement s'inverse pour le reste de groupes où la méthode AG

TABLE 3.7 – Indicateurs de performance pour les méthodes TIH, ILS et $MSPLS$: Première partie

Groupe	Method	RBKS	Gap
1	$h_{8,7}$	48,33%	0,34%
	$h_{12,1}$	46,67%	0,26%
2	$h_{2,7}$	40,56%	0,46%
	$h_{6,7}$	36,11%	0,45%
3	$h_{12,7}$	41,11%	1,29%
	$h_{5,2}$	39,44%	1,02%
4	$h_{7,2}$	44,44%	3,89%
	$h_{5,1}$	44,44%	4,07%
	$h_{7,1}$	41,67%	3,49%
5	$h_{7,2}$	90,56%	2,50%
	$h_{8,1}$	90,56%	3,68%
	$h_{4,1}$	90,00%	2,86%
	$h_{4,2}$	90,00%	2,68%
	$h_{5,1}$	90,00%	2,95%
	$h_{6,1}$	90,00%	3,06%
	$h_{7,1}$	90,00%	2,72%
	$h_{12,2}$	90,00%	3,74%
6	$h_{7,7}$	42,78%	1,26%
	$h_{12,7}$	38,89%	1,06%
7	$h_{3,7}$	46,67%	1,29%
8	$h_{3,7}$	76,11%	5,41%
	$h_{2,7}$	76,11%	6,03%
	$h_{11,7}$	76,11%	5,75%
9	$h_{1,7}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,7}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{7,1}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,7}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,7}$	100,00%	0,00%
	$h_{10,7}$	100,00%	0,00%
	$h_{11,7}$	100,00%	0,00%
	$h_{12,1}$	100,00%	0,00%
10	$h_{i,7}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%

Groupe	Method	RBKS	Gap
10	$h_{6,2}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,6}$	100,00%	0,00%
	$h_{12,1}$	100,00%	0,00%
	$h_{12,2}$	100,00%	0,00%
11	$h_{12,2}$	57,78%	1,41%
	$h_{4,7}$	51,67%	1,20%
12	$h_{3,7}$	96,67%	1,07%
13	$h_{i,7}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
14	$h_{i,7}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{7,1}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,4}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,2}$	100,00%	0,00%
	$h_{12,1}$	100,00%	0,00%
	$h_{12,2}$	100,00%	0,00%

Groupe	Method	RBKS	Gap
15	$h_{i,7}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,4}$	100,00%	0,00%
	$h_{1,5}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,4}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{5,6}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{6,4}$	100,00%	0,00%
	$h_{6,6}$	100,00%	0,00%
	$h_{7,1}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,4}$	100,00%	0,00%
	$h_{7,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,4}$	100,00%	0,00%
	$h_{8,5}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,2}$	100,00%	0,00%
	$h_{9,6}$	100,00%	0,00%
	$h_{12,1}$	100,00%	0,00%
	$h_{12,2}$	100,00%	0,00%
	$h_{12,4}$	100,00%	0,00%
	$h_{12,5}$	100,00%	0,00%
	$h_{12,6}$	100,00%	0,00%
16	$h_{1,1}$	81,67%	0,58%
	$h_{4,1}$	81,67%	0,63%
	$h_{12,1}$	81,67%	0,61%
	$h_{6,1}$	81,11%	0,65%
	$h_{12,2}$	81,11%	0,55%
17	$h_{3,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{12,2}$	100,00%	0,00%

TABLE 3.8 – Indicateurs de performance pour les méthodes TIH, ILS et $MSPLS$: Deuxième partie

Groupe	Method	RBKS	Gap
18	$h_{i,7}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,4}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{5,6}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{6,4}$	100,00%	0,00%
	$h_{6,6}$	100,00%	0,00%
	$h_{7,1}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,4}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
	$h_{12,1}$	100,00%	0,00%
	$h_{12,2}$	100,00%	0,00%
	$h_{12,4}$	100,00%	0,00%
19	$h_{u,7}$	100,00%	0,00%
	$h_{7,i}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,4}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,4}$	100,00%	0,00%
	$h_{4,5}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{5,5}$	100,00%	0,00%
	$h_{5,6}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{6,4}$	100,00%	0,00%
	$h_{6,5}$	100,00%	0,00%
	$h_{6,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,4}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,2}$	99,44%	0,12%
	$h_{9,6}$	99,44%	0,13%

Groupe	Method	RBKS	Gap
20	$h_{u,7}$	100,00%	0,00%
	$h_{4,i}$	100,00%	0,00%
	$h_{5,i}$	100,00%	0,00%
	$h_{6,i}$	100,00%	0,00%
	$h_{7,i}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,4}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,4}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,2}$	100,00%	0,00%
	$h_{9,6}$	100,00%	0,00%
21	$h_{12,1}$	95,56%	0,21%
22	$h_{u,7}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%
	$h_{4,4}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{5,6}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{6,4}$	100,00%	0,00%
	$h_{6,6}$	100,00%	0,00%
	$h_{7,1}$	100,00%	0,00%
	$h_{7,2}$	100,00%	0,00%
	$h_{7,4}$	100,00%	0,00%
	$h_{7,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
23	$h_{u,7}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,4}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{4,1}$	100,00%	0,00%
	$h_{4,2}$	100,00%	0,00%

Groupe	Method	RBKS	Gap
23	$h_{4,4}$	100,00%	0,00%
	$h_{4,6}$	100,00%	0,00%
	$h_{5,1}$	100,00%	0,00%
	$h_{5,2}$	100,00%	0,00%
	$h_{5,4}$	100,00%	0,00%
	$h_{5,6}$	100,00%	0,00%
	$h_{6,1}$	100,00%	0,00%
	$h_{6,2}$	100,00%	0,00%
	$h_{6,4}$	100,00%	0,00%
	$h_{6,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,4}$	100,00%	0,00%
	$h_{8,6}$	100,00%	0,00%
24	$h_{u,7}$	100,00%	0,00%
	$h_{4,i}$	100,00%	0,00%
	$h_{5,i}$	100,00%	0,00%
	$h_{6,i}$	100,00%	0,00%
	$h_{7,i}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{1,1}$	100,00%	0,00%
	$h_{1,2}$	100,00%	0,00%
	$h_{1,3}$	100,00%	0,00%
	$h_{1,4}$	100,00%	0,00%
	$h_{1,6}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,4}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{8,1}$	100,00%	0,00%
	$h_{8,2}$	100,00%	0,00%
	$h_{8,4}$	100,00%	0,00%
	$h_{8,5}$	99,44%	0,56%
	$h_{8,6}$	100,00%	0,00%
	$h_{9,6}$	100,00%	0,00%
25	$h_{u,7}$	100,00%	0,00%
	$h_{1,i}$	100,00%	0,00%
	$h_{4,i}$	100,00%	0,00%
	$h_{5,i}$	100,00%	0,00%
	$h_{6,i}$	100,00%	0,00%
	$h_{7,i}$	100,00%	0,00%
	$h_{8,i}$	100,00%	0,00%
	$h_{12,i}$	100,00%	0,00%
	$h_{2,1}$	100,00%	0,00%
	$h_{2,2}$	100,00%	0,00%
	$h_{2,4}$	100,00%	0,00%
	$h_{2,6}$	100,00%	0,00%
	$h_{3,1}$	100,00%	0,00%
	$h_{3,2}$	100,00%	0,00%
	$h_{3,4}$	100,00%	0,00%
	$h_{3,6}$	100,00%	0,00%
	$h_{9,1}$	100,00%	0,00%
	$h_{9,2}$	99,44%	0,56%
	$h_{9,6}$	100,00%	0,00%

TABLE 3.9 – Temps de calcul pour les méthodes TIH, ILS et $MSPLS$

	Machines	$N = 12$		$N = 40$		$N = 100$	
		t_{min}	t_{max}	t_{min}	t_{max}	t_{min}	t_{max}
	$M = 2$	0,003	0,025	0,029	0,066	0,164	0,421
$h_{i,1}$	$M = 3$	0,004	0,038	0,036	0,102	0,197	0,280
	$M = 5$	0,005	0,050	0,049	0,158	0,276	0,376
	$M = 2$	0,003	0,012	0,029	0,049	0,165	0,414
$h_{i,2}$	$M = 3$	0,004	0,051	0,036	0,112	0,200	0,568
	$M = 5$	0,005	0,020	0,049	0,207	0,282	0,380
	$M = 2$	0,003	0,012	0,067	0,141	1,107	2,709
$h_{i,3}$	$M = 3$	0,004	0,036	0,044	0,169	0,765	1,326
	$M = 5$	0,005	0,015	0,046	0,155	0,557	1,135
	$M = 2$	0,003	0,012	0,028	0,047	0,160	0,377
$h_{i,4}$	$M = 3$	0,003	0,049	0,034	0,356	0,192	1,567
	$M = 5$	0,004	0,013	0,045	0,175	0,272	0,385
	$M = 2$	0,000	0,013	0,000	0,061	0,007	0,464
$h_{i,5}$	$M = 3$	0,000	0,083	0,000	0,167	0,008	0,362
	$M = 5$	0,000	0,015	0,000	0,184	0,012	0,447
	$M = 2$	0,000	0,022	0,000	0,084	0,010	0,706
$h_{i,6}$	$M = 3$	0,000	0,166	0,001	0,260	0,013	0,531
	$M = 5$	0,000	0,018	0,001	0,212	0,018	0,830
	$M = 2$	0,000	0,031	0,000	0,132	0,000	1,136
$h_{i,7}$	$M = 3$	0,000	0,062	0,000	0,274	0,000	0,757
	$M = 5$	0,000	0,045	0,000	0,298	0,000	0,923

TABLE 3.10 – Indicateurs de performance $RBKS_i$ et GAP pour les métaheuristiques classiques

Groupe	**Method**	**RBKS**	**Gap**
1	ACO	32,22%	11,25%
	GA1	28,89%	1,21%
2	ACO	27,22%	20,92%
	GA1	19,44%	3,14%
3	ACO	26,11%	29,12%
	GA1	21,67%	7,73%
4	ACO	28,33%	42,97%
	GA1	21,11%	22,09%
5	GA1	63,89%	28,84%
6	ACO	31,11%	26,04%
	GA1	29,44%	4,83%
7	ACO	30,00%	42,20%
	GA1	26,11%	17,45%
8	ACO	30,56%	65,10%
	GA1	27,22%	59,44%
9	GA1	77,78%	20,24%
10	GA1	98,89%	0,84%
11	ACO	32,78%	35,30%
	GA1	28,89%	14,00%
12	GA1	63,33%	30,47%
13	GA1	95,56%	3,87%
14	GA1	98,33%	1,38%
15	GA1	100,00%	0,00%
16	ACO	33,33%	39,77%
	GA1	31,11%	20,98%
17	GA1	87,78%	10,16%
18	GA1	97,78%	1,97%
19	GA1	100,00%	0,00%
20	GA1	100,00%	0,00%
21	GA1	43,33%	18,85%
22	GA1	96,67%	2,24%
23	GA1	99,44%	0,56%
24	GA1	100,00%	0,00%
25	ACO	99,44%	0,56%
	GA1	100,00%	0,00%

TABLE 3.11 – Temps de calcul pour les métaheuristiques AG et ACO

	Machines	$N = 12$		$N = 40$		$N = 100$	
		t_{min}	t_{max}	t_{min}	t_{max}	t_{min}	t_{max}
	$M = 2$	0,000	0,071	0,000	2,451	0,004	57,926
GA	$M = 3$	0,000	0,048	0,000	1,965	0,004	102,768
	$M = 5$	0,000	0,063	0,000	2,106	0,004	46,045
	$M = 2$	0,000	1,395	0,008	57,072	0,174	24,367
ACO	$M = 3$	0,000	1,439	0,008	56,874	0,174	25,107
	$M = 5$	0,000	1,717	0,008	56,575	0,173	25,764

TABLE 3.12 – Groupes avec peu de méthodes trouvant la meilleure solution

	β_1	β_2	β_3	β_4	β_5
α_1	**1**	**2**	**3**	**4**	**5**
α_2	**6**	**7**	**8**		
α_3	**11**	**12**			
α_4	**16**	**17**			
α_5	**21**				

obtient des résultats acceptables. Les temps d'exécution sont données dans le tableau 3.11.

Les temps d'exécution des métaheuristiques GA et ACO sont, en général, plus importants que pour le reste d'heuristiques développées et testées dans ce chapitre.

L'ensemble de résultats obtenus et présentés dans les sections précédentes nous permettent d'identifier des groupes dont la résolution de leurs instances représente une difficulté plus importante, par rapport au nombre de méthodes trouvant la meilleure solution connue. Le tableau 3.12 contient ces groupes identifiés.

Avec l'objectif d'identifier les heuristiques ayant les meilleures performances dans la résolution du problème d'ordonnancement présenté dans ce chapitre, nous effectuons une comparaison réduite aux groupes dans le tableau 3.12 pour l'ensemble de méthodes testées.

Pour les méthodes de recherche locale, l'heuristique construite avec la fonction EA (échanges aléatoires), ayant comme solution initiale la solution trouvée avec IS_7 a eu, en moyenne, la meilleure performance pour les instances du groupe 11. Cette même fonction de voisinage, combinée avec la solution initiale IS_6 a obtenue des performances très élevées dans les instances du groupe 16.

Les méthodes métaheuristiques basées sur la recherche locale ont obtenu les meilleurs résultats pour la quasi-totalité des groupes concernés (groupes dans le tableau 3.12). Les algorithmes $MSPLS2$, $TIH2$ et $TIH1$ apparaissent comme les méthodes les plus adaptées à la résolution de la plupart des instances des groupes étudiés, avec une préférence pour la méthode $MSPLS2$ qui obtient la meilleure solution pour 9 des 13 groupes concernés (tableau 3.12). Également, cette méthode est peu influencée par les solutions initiales utilisées, faisant d'elle une méthode robuste.

Par rapport aux temps d'exécution obtenus, à l'exception de l'algorithme de colonie de fourmis, les méthodes heuristiques ont, en moyenne, des temps d'exécution inférieurs à ceux de la méthode exacte (CPLEX), pour les 25 groupes testés. Pour les méthodes signalées comme les plus efficaces, la méthode $MSPLS2$ semble avoir des temps d'exécution plus grandes que les méthodes $TIH1$ et $TIH2$. Ces temps dépendent non seulement de la taille de l'instance mais également du groupe associé à l'instance. Dans l'annexe B le détail des temps d'exécution pour l'ensemble de méthodes approchées sont donnés. Ces résultats permettent de vérifier que la méthode $MSPLS2$ à des temps d'exécution inférieurs pour certains groupes.

Les méthodes $TIH1$ et $MSPLS2$ ont été testées sur des instances issues de la littérature pour le problème $P_m//\sum T_i$. Ces résultats sont inclus dans le tableau 3.13. Les méthodes $TIH1$ et $MSPLS2$ trouvent la solution optimale dans 80% des instances, avec des temps de traitement acceptables. Dans le tableau 3.13, une comparaison avec les méthodes proposées par Tanaka et Araki [126] (B&B) et par Deng *et al.* [48] (HDDE) est présentée. Ces résultats montrent des meilleures performances pour la méthode $TIH1$ que pour la méthode $MSPLS2$ pour la quasi totalité d'instances.

L'ensemble de résultats expérimentaux sont utilisés dans la section suivante pour proposer leur intégration dans un système d'aide à la décision adaptable au cas réel présenté dans le chapitre 1.

Autre que l'identification d'une méthode dominante, ce chapitre offre une deuxième analyse des solutions obtenues. Cette analyse est présentée dans la section suivante. L'objectif est d'établir une relation entre la composition de chaque groupe, les performances obtenues des méthodes de résolution de cette section, et les caractéristiques N et M de chaque instance. De la même manière, les résultats de cette analyse sont ajoutés au système d'aide à la décision de la section suivante.

TABLE 3.13 – Instances proposées par Tanaka et Araki

		TIH1		**MSPLS2**		**HDDE Deng *et al.***	**B&B Tanaka-Araki**
N	M	**RBKS**	**Temps**	**RBKS**	**Temps**	**RBKS**	**Temps**
20	2	84,80%	0,009	79,20%	0,026	100%*	0.429
	3	89,60%	0,012	84,00%	0,028	-	0.296
	4	88,00%	0,013	84,80%	0,030	96%*	0.150
	5	90,40%	0,014	89,60%	0,033	-	0.079
	6	95,20%	0,016	92,00%	0,037	100%*	0.051
	7	92,80%	0,018	94,40%	0,041	-	0.040
	8	96,80%	0,020	92,80%	0,044	98%*	0.025
	9	97,60%	0,021	96,00%	0,047	-	0.113
	10	100,00%	0,022	98,40%	0,050	100%*	0.030
25	2	81,60%	0,014	68,00%	0,041	-	1.156
	3	68,80%	0,016	64,80%	0,042	-	19.846
	4	70,40%	0,020	70,40%	0,047	-	47.470
	5	69,60%	0,022	67,20%	0,049	-	14.147
	6	79,20%	0,025	79,20%	0,053	-	0.368
	7	78,40%	0,029	76,00%	0,060	-	0.163
	8	80,80%	0,031	78,40%	0,068	-	0.110
	9	82,40%	0,033	86,40%	0,073	-	0.184
	10	92,80%	0,035	95,20%	0,076	-	0.071

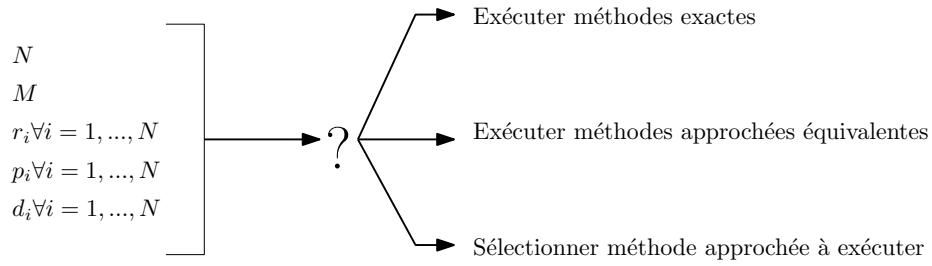

FIGURE 3.6 – Étude de l'impact des paramètres d'entrée

3.7 Étude sur l'impact du nombre de tâches sur les performances

Le principal objectif de cette section est de donner une interprétation alternative aux résultats obtenus dans les sections précédentes autre que la sélection des méthodes performants. Nous cherchons des valeurs pour la relation N, M pour lesquelles les résultats montrent un comportement homogène des méthodes testées. Ces valeurs seraient la limite pour tester des méthodes de résolution, au-dessous de laquelle, la plupart des méthodes obtiennent des solutions très proches en fonction de l'évaluation du retard total. Également, cette étude inclut les performances de la méthode exacte dans la résolution du problème traité. Dans la figure 3.6, l'objectif de cette étude est montré.

Afin d'atteindre cet objectif, nous proposons d'établir une relation entre les performances des méthodes testées et un indicateur composé du nombre de machines M et tâches N. Cette relation doit tenir en compte des paramètres relatifs à la distribution des caractéristiques r_i et d_i pour une instance donnée. Par conséquent, cette analyse s'effectue de manière indépendante pour chacun des groupes définis dans le tableau 3.1.

Les paramètres N et M sont implémentés sur un indicateur N^2/M. Ce choix a été pris afin de réduire les probabilités de valeurs répétées pour l'indicateur utilisé. Par exemple, pour une instance avec $M = 2$ et $N = 12$ tâches, une autre instance avec le même indicateur doit être composé de 8 machines et 24 tâches exactement, tandis que si l'indicateur N/M est utilisé, des instances avec $N = 12, M = 2$, $N = 18, M = 3$ et $N = 24, M = 4$ ont le même indicateur.

Pour chaque groupe, les valeurs de l'indicateur N^2/M sont comparées avec un indicateur estimateur de la taille de l'intervalle qui contient les solutions obtenues de toutes les méthodes testées. Nous proposons d'utiliser l'écart type des résultats, pour une instance donnée. Si nous considérons une instance p, la solution (le retard total) obtenue par les différents méthodes

TABLE 3.14 – Test de normalité des indicateurs

Index	Result
GAP	Loi normale non confirmée avec $p-value < 0.10$
$RBKS$	Loi normale non confirmée avec $p-value < 0.10$
Temps moyen	Loi normale non confirmée avec $p-value < 0.10$
Retard moyen	Loi normale non confirmée avec $p-value < 0.10$
d_i/r_i	Loi normale non confirmée avec $p-value < 0.10$
$\frac{\sigma_p}{max_{\forall p \in GP}(\sigma_p)}$	Loi normale confirmée avec $p-value < 0.05$

est représenté par $h_{p,1}, ...h_{p,K}$, avec K le total de méthodes testées, l'écart type σ_p est donné par l'équation 3.69.

$$\sigma_p = \sqrt{var\left(h_{p,1}, ..., h_{p,K}\right)} \tag{3.69}$$

Afin de valider notre analyse, nous avons appliqué le test de Fisher pour garantir un effet non négligeable des paramètres N, M, α et β.

Le test de Fisher s'exécute pour chaque paramètre de manière individuelle, en utilisant des données où le reste de paramètres sont fixés. La même procédure est suivie pour les interactions N/M et α/β.

Le test de Fisher dépend d'un comportement normal du paramètre évalué. Nous avons réalisé le test de Shapiro-Wilk [115]. Autres indicateurs potentiels ont été aussi testés. Les résultats sont donnés dans le tableau 3.14, où GP est l'ensemble d'instances considérées à chaque analyse.

Le test de Fisher a été appliqué en concluant que les paramètres N, M, α et β et les relations N^2/M et $\alpha - \beta$ ont des impacts non négligeables sur l'indicateur $\frac{\sigma_p}{max_{\forall p \in GP}(\sigma_p)}$. Ces résultats ont obtenu des $p-value$ inférieures de 0.05.

Pour la suite de cette analyse, elle sera exécutée de manière individuelle pour chacun des groupes définis dans le tableau 3.1, car, comme nous l'avons montré, les valeurs des paramètres α et β ont un impact non négligeable sur l'indicateur choisi ($\frac{\sigma_p}{max_{\forall p \in GP}(\sigma_p)}$).

Pour le groupe 1 ($\alpha = 0.2, \beta = 0.02$), la relation entre les indicateurs N^2/M et $\frac{\sigma_p}{max_{\forall p \in GP}(\sigma_p)}$ est représentée avec la graphique 3.7.

D'une manière similaire, tous les résultats des 25 groupes définis ont des comportements linéaires comme celui montré pour le groupe 1. Nous avons utilisé la régression des moindres carrées afin d'obtenir une expression mathématique de $\frac{\sigma_p}{max_{\forall p \in GP}(\sigma_p)}$ en fonction de N^2/M pour chaque groupe. Le coefficient de détermination (R^2) a été utilisé pour évaluer les mo-

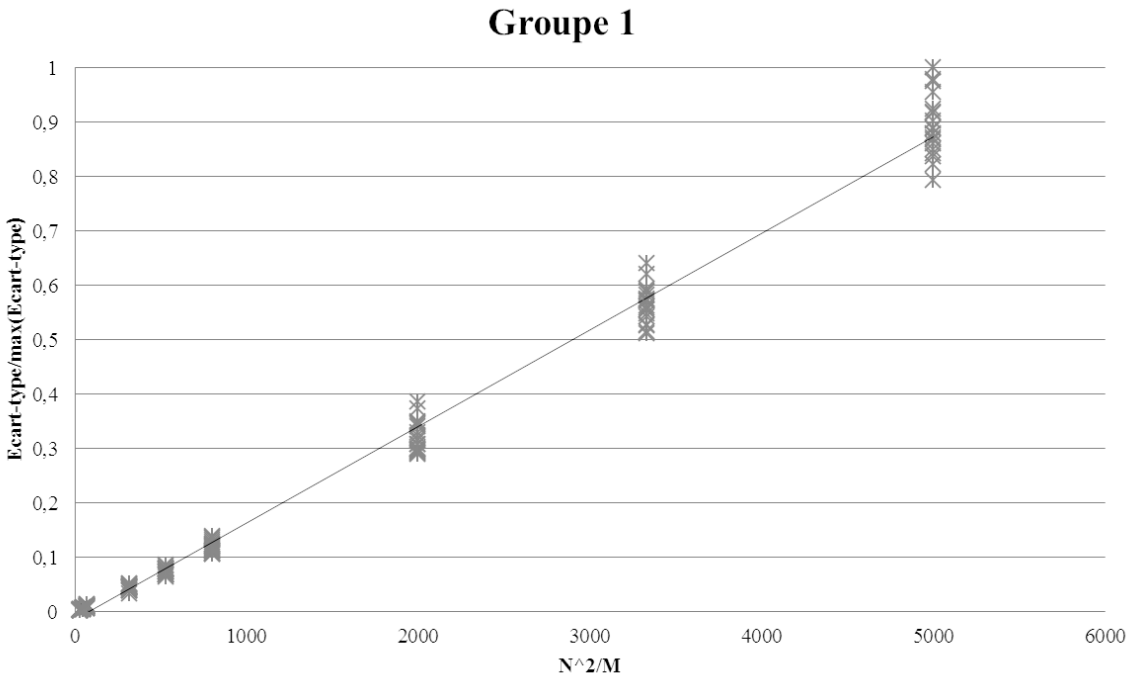

FIGURE 3.7 – Écart type $\frac{\sigma_p}{max_{\forall p \in GP(\sigma_p)}}$ vrs N^2/M pour le groupe 1

dèles obtenus. Les coefficients de détermination pour des modèles linéaires correspondent au coefficient de corrélation.

Les coefficients de corrélation obtenus nous permettent de stipuler que pour les conditions expérimentales fixées (section 3.6.1), il existe une relation linéaire entre les indicateurs $\frac{\sigma_p}{max_{\forall p \in GP(\sigma_p)}}$ et N^2/M pour les 25 groupes.

Les modèles obtenus sont de la forme $\frac{\acute{\sigma_p}}{max_{\forall p \in GP(\sigma_p)}} = A * \left(\frac{N^2}{M}\right) + B$. Les coefficients A, B et le coefficient de détermination R^2 sont montrés dans le tableau 3.15.

Les valeurs dans le tableau 3.15 peuvent être interprétées pour obtenir des valeurs pour le paramètre N, avec l'équation 3.70.

$$N = \sqrt{\frac{\left(\frac{\sigma_p}{max_{\forall p \in GP(\sigma_p)}} - B\right) * M}{A}} \tag{3.70}$$

Cette équation nous permet de trouver, pour chaque groupe, et avec $M = [2, 3, 5]$, les valeurs où la droit de la graphique 3.7 approche des valeurs 0 pour l'indicateur $\frac{\sigma_p}{max_{\forall p \in GP(\sigma_p)}}$.

Les tables 3.16, 3.17 et 3.18 montrent ces résultats pour chaque groupe et chaque valeur de M.

Pour les instances avec 2 machines d'après le tableau 3.16, seulement les groupes 4 et 5

TABLE 3.15 – Coefficient de régression linéaire

Groupe	A	B	R^2
1	1.77×10^{-04}	-1.295×10^{-02}	0,99
2	1.71×10^{-04}	-1.253×10^{-02}	0,99
3	1.73×10^{-04}	-1.357×10^{-02}	0,99
4	1.61×10^{-04}	-1.113×10^{-02}	0,98
5	1.54×10^{-04}	-1.051×10^{-02}	0,98
6	1.69×10^{-04}	-2.693×10^{-02}	0,98
7	1.80×10^{-04}	-2.044×10^{-02}	0,99
8	1.68×10^{-04}	-1.835×10^{-02}	0,98
9	1.62×10^{-04}	-1.973×10^{-02}	0,98
10	1.51×10^{-04}	-1.761×10^{-02}	0,96
11	1.75×10^{-04}	-2.489×10^{-02}	0,98
12	1.72×10^{-04}	-2.314×10^{-02}	0,98
13	1.61×10^{-04}	-2.345×10^{-02}	0,98
14	1.73×10^{-04}	-2.228×10^{-02}	0,98
15	1.54×10^{-04}	-1.907×10^{-02}	0,98
16	1.57×10^{-04}	-2.384×10^{-02}	0,98
17	1.71×10^{-04}	-2.497×10^{-02}	0,98
18	1.57×10^{-04}	-2.218×10^{-02}	0,98
19	1.48×10^{-04}	-2.087×10^{-02}	0,98
20	1.60×10^{-04}	-2.072×10^{-02}	0,98
21	1.77×10^{-04}	-2.358×10^{-02}	0,99
22	1.62×10^{-04}	-2.320×10^{-02}	0,98
23	1.65×10^{-04}	-2.355×10^{-02}	0,98
24	1.52×10^{-04}	-2.140×10^{-02}	0,97
25	1.52×10^{-04}	-1.946×10^{-02}	0,98

TABLE 3.16 – Nombre de tâches minimales avec ($M = 2$)

$M = 2$	β_1	β_2	β_3	β_4	β_5
α_1	12,09	12,10	12,52	11,76	11,68
α_2	17,88	15,07	14,78	15,59	15,25
α_3	16,89	16,41	17,06	16,04	15,74
α_4	17,45	17,11	16,79	16,80	16,08
α_5	16,33	16,95	16,91	16,77	15,98

TABLE 3.17 – Nombre de tâches minimales avec ($M = 3$)

$M = 3$	β_1	β_2	β_3	β_4	β_5
α_1	14,80	14,82	15,34	14,40	14,31
α_2	21,89	18,46	18,10	19,10	18,68
α_3	20,68	20,09	20,90	19,65	19,28
α_4	21,37	20,95	20,57	20,58	19,69
α_5	19,99	20,76	20,71	20,53	19,58

sont intéressants à analyser à partir de 12 tâches. Pour les autres groupes, la plupart des groupes sont intéressants à partir de 15 tâches.

D'après le tableau 3.17, pour les instances avec 3 machines, un minimum de 15 tâches est nécessaire pour que ces problèmes soient intéressants dans la comparaison de méthodes de résolution. Selon le groupe, cette valeur peut augmenter jusqu'à 21 ou 22 tâches.

Finalement, pour les instances avec 5 machines, d'après nos résultats, un minimum de 19 tâches est nécessaire pour comparer des heuristiques. Pour certains groupes cette valeur peut monter jusqu'à 26 tâches.

TABLE 3.18 – Nombre de tâches minimales avec ($M = 5$)

$M = 5$	β_1	β_2	β_3	β_4	β_5
α_1	19,11	19,13	19,80	18,59	18,47
α_2	28,26	23,84	23,36	24,66	24,11
α_3	26,70	25,94	26,98	25,36	24,89
α_4	27,59	27,05	26,55	26,57	25,42
α_5	25,81	26,80	26,74	26,51	25,27

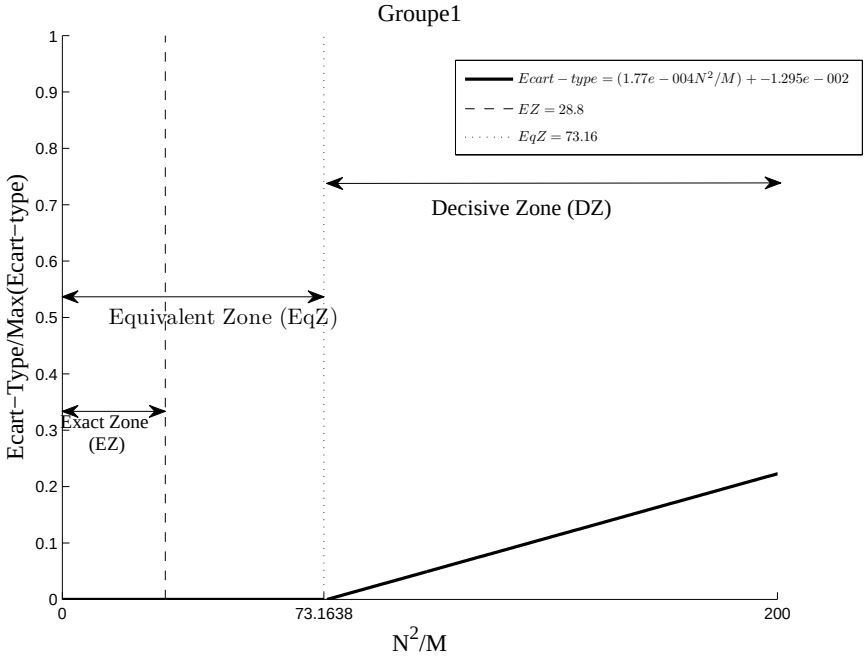

FIGURE 3.8 – Zones Ez, EqZ et DZ pour le groupe 1

Avec l'indicateur N^2/M les résultats des sections précédentes peuvent être utilisés pour représenter graphiquement les objectifs montrés dans la figure 3.6. Trois zones peuvent ainsi être définies :

- EZ «Exact Zone» : Zone où la méthode exacte garantie l'optimalité de la solution sous un temps d'exécution inférieur à 1800 secondes.
- EqZ «Equivalent Zone» : Zone où d'après les conclusions de la section 3.7, les heuristiques obtiennent des solutions proches et équivalents.
- DZ «Decisive Zone» : Zone où les résultats des heuristiques testées sont suffisamment différentes pour identifier les méthodes efficaces.

La figure 3.8 montre ces trois zones pour le groupe 1.

Dans l'annexe B les groupes 2 à 25 sont montrés. Les sous-sections suivantes décrivent la mise en place des outils et méthodes décrites dans les sections précédentes, dans un système d'aide à la décision pour l'ordonnancement d'un atelier sur machines parallèles.

Les valeurs limites des zones EqZ pour les 25 groupes sont dans la figure 3.9.

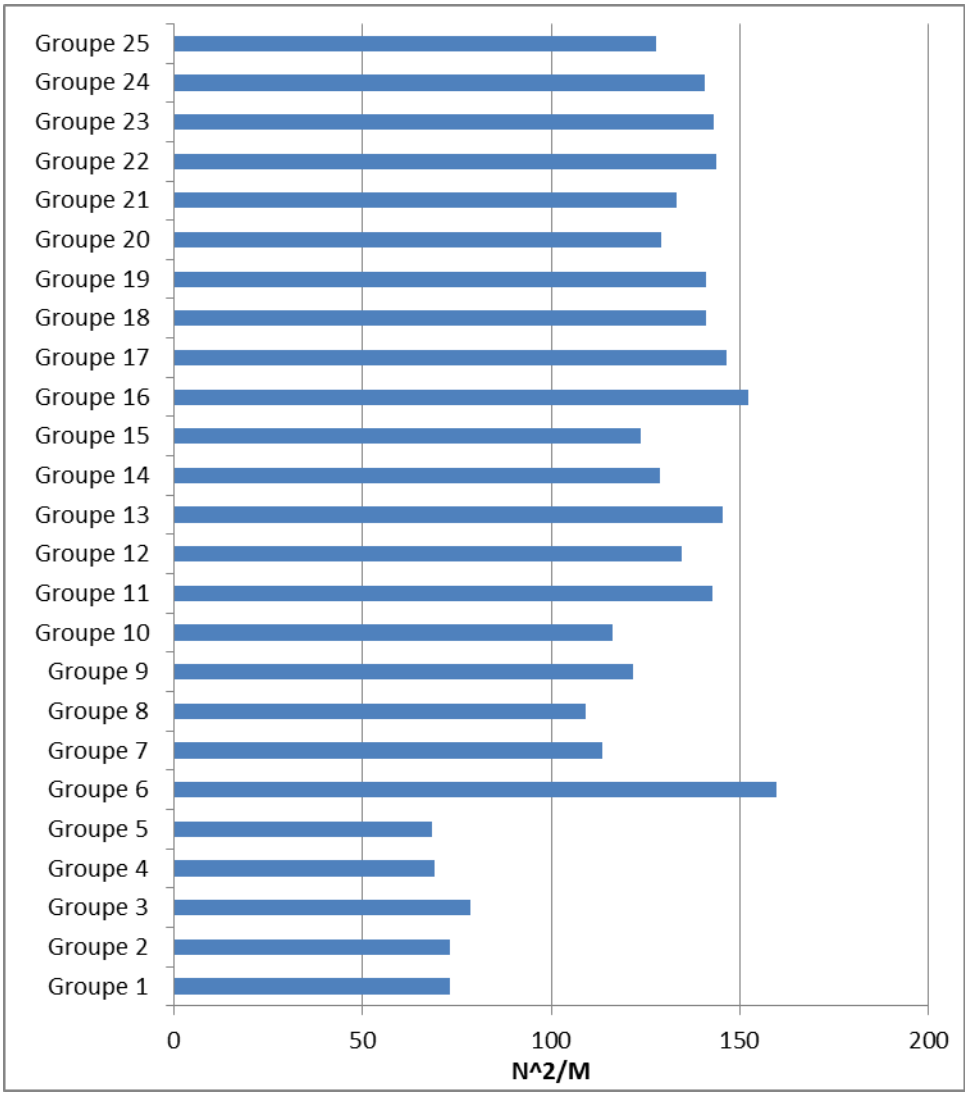

FIGURE 3.9 – Zones EqZ pour les 25 groupes

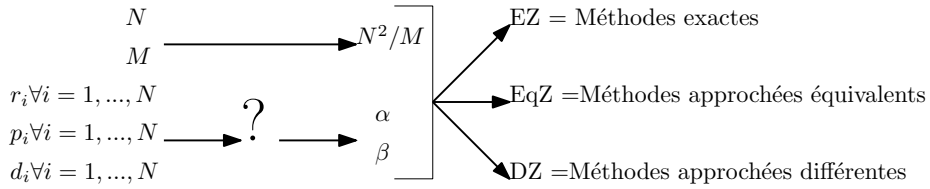

FIGURE 3.10 – Données d'entrée

Application aux systèmes d'aide à la décision

Par la suite, l'objectif de cette sous-section est de décrire l'application des conclusions des sections précédentes sur un système d'aide à la décision dédié à la gestion de tâches dans un atelier composé de machines identiques en parallèle. Un exemple d'application effectué sur l'information issue du cas réel décrit dans le chapitre 1 est donné.

Ayant défini l'indicateur N^2/M, et les zones EZ, EqZ et DZ par groupe, notre objectif est de, pour une instance quelconque, établir son groupe d'appartenance et son indicateur N^2/M afin d'identifier quelle méthode s'adapte le mieux à sa résolution.

Données d'entrée

De la même manière que pour la description du problème, les données d'entrée pour toute instance sont les caractéristiques N, M, r_i, p_i et d_i. L'indicateur N^2/M est déduit facilement. Pour identifier le groupe d'appartenance (groupes du tableau 3.1), il est nécessaire de calculer les valeur α et β. La figure 3.10 résume cette problématique.

D'autre part, les cas considérés doivent respecter certaines conditions :

- Seules les instances avec $N > M$ sont considérées.
- Les valeurs des paramètres r_i, p_i et d_i sont connues, et des entiers positifs.
- Pour toute instance valide, pour tout tâche i, $d_i \geq r_i + p_i$.
- Les temps de traitement sont considérés indépendants.
- Les dates de disponibilité r_i sont distribuées aléatoirement avec une loi uniforme dans un intervalle $[R_{min}, R_{max}]$.
- Si $\delta_i = d_i - r_i - p_i$, donc pour toutes les tâches, les δ_i sont distribués aléatoirement avec une loi uniforme dans un intervalle $[\Delta_{min}, \Delta_{max}]$.

Pour les valeurs des dates de disponibilité et des valeurs δ_i, nous considérons les intervalles définis dans les équations 3.67 et 3.68. Les valeurs supérieures et inférieures de α et β ne sont pas prises en compte car ils font référence à d'autres problèmes différents à celui traité dans ce chapitre. La valeur moyenne des valeurs réelles des r_i et δ_i peut être également utilisé

pour calculer les valeurs des α et β, avec les équations (3.71) et (3.72) :

$$\alpha \approx \left(\frac{2M * \sum_{i=0}^{N} r_i}{N * \sum_{i=0}^{N} p_i} \right) \tag{3.71}$$

$$\beta \approx \left(\frac{2M * \sum_{i=0}^{N} \delta_i}{N * \sum_{i=0}^{N} p_i} \right) \tag{3.72}$$

Procédure de décision

Ayant identifié les différents valeurs d'entrée, nous proposons d'utiliser une procédure de décision afin de sélectionner la méthode de résolution à appliquer pour l'instance traitée.

Pour cela, avec les différents indicateurs calculés, la procédure cherche par rapport au couple $\alpha; \beta$, et la valeur de l'indicateur N^2/M, si l'instance traitée est dans la zone EZ, EqZ ou DZ. Si l'instance est dans les zones EZ ou EqZ, la méthode la plus rapide peut ainsi être sélectionnée. D'après les résultats de la section 3.6, les méthodes testées ont des performances équivalentes dans ces zones. Si l'instance est dans la zone DZ, la décision dépend de la politique du temps d'exécution admise : si le temps d'exécution est limité par l'application, le critère de temps est donc priorisé par rapport au critère de qualité. Cela a un impact réduit sur les performances de la méthode car les méthodes considérées trouvent en moyenne des solutions de bonne qualité. Seules les méthodes ayant eu les meilleures performances dans les tests réalisés dans la section 3.6 sont proposées pour la zone DZ.

Les temps sont données par groupe comme dans les figures 3.11 et 3.12 pour le groupe 1. Dans ces figures, seules les heuristiques ayant les meilleures performances sont montrées. Pour une seule valeur de N^2/M, plusieurs heuristiques donnent des résultats acceptables, avec des temps d'exécution différents.

Cette procédure de décision a été appliquée au cas réel présenté dans le chapitre 1. Les résultats sont montrés dans le chapitre 5.

3.8 Conclusion

Dans ce chapitre nous avons considéré le problème d'ordonnancement de tâches sur des machines parallèles $P_m/r_i/\sum T_i$ dans lequel un ensemble de tâches doivent être affectées sur un ensemble de machines identiques disposées en parallèle. Chaque tâche est caractérisée par une date de disponibilité, un temps de traitement et une date de fin souhaitée.

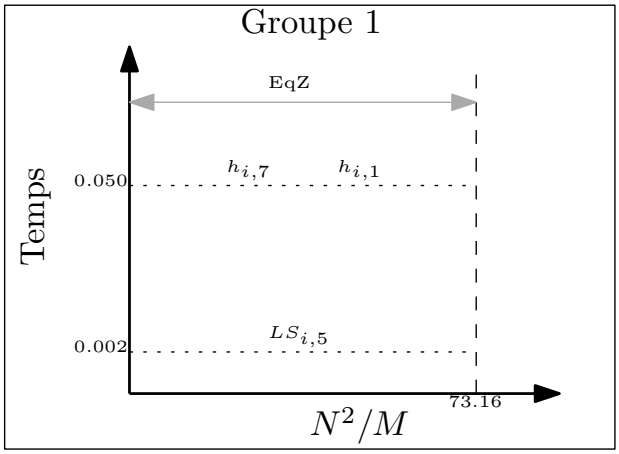

FIGURE 3.11 – Temps - N^2/M pour groupe 1 (zone EZ)

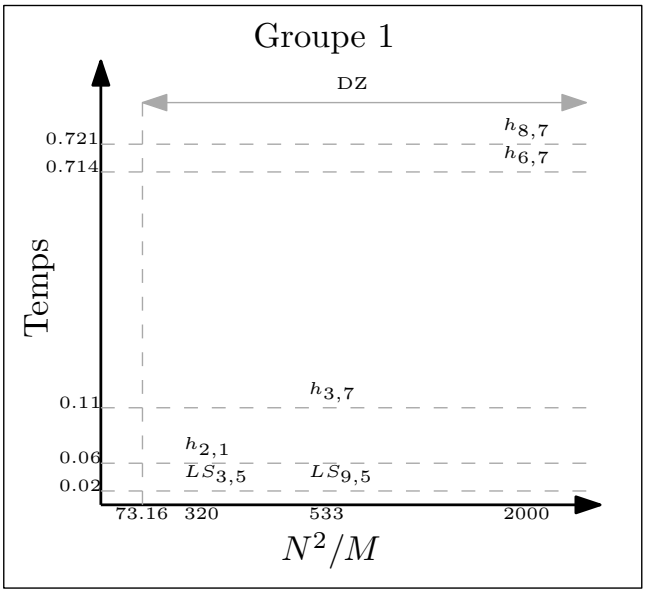

FIGURE 3.12 – Temps - N^2/M pour groupe 1 (zone DZ)

Nous avons traité ce problème sur deux approches différentes. Dans la première partie, des méthodes de résolution sont proposées et testées sur plusieurs instances de différente difficulté. Dans la deuxième partie, une analyse approfondie de ces résultats a permis l'identification d'une relation entre les performances obtenues et les caractéristiques de chaque instance. Ces instances ont été générées pour des exemples avec 12, 40 et 100 tâches, affectées à 2, 3 ou 5 machines. Deux paramètres α et β sont utilisés pour générer les différents caractéristiques des tâches. 25 combinaisons différentes sont considérées.

Parmi les méthodes de résolution testées, des méthodes de liste ont été utilisées comme des solutions initiales pour des recherches locales, recherches taboues et autres métaheuristiques.

Les résultats des tests effectués montrent que la méthode $MSPLS2$ (Multi Start Partial Local Search), laquelle effectue une recherche locale partielle avec des voisinages très diversifiés par l'utilisation d'un mouvement d'échanges aléatoires, et une procédure de réinitialisation en cas de stagnation des résultats autour d'un optimum local, est la méthode qui obtient en moyenne les meilleurs résultats. Cette méthode est relativement rapide, faisant d'elle une métaheuristique efficace pour ce problème d'ordonnancement.

L'ensemble de résultats ont été également analysés afin d'identifier les groupes d'instances les plus intéressants dans le développement de nouvelles méthodes de résolution. Parmi les 25 groupes testés pour la combinaison des paramètres α et β, nous avons identifié 13 groupes où les méthodes testées ont eu les plus bas performances. Une analyse sur ces groupes a confirmé les bonnes performances de la méthode $MSPLS2$. D'autre part, cette méthode a un comportement robuste indépendant de la solution initiale utilisée.

Dans la deuxième partie de ce chapitre, l'analyse de la totalité des données disponibles a conduit vers l'identification des valeurs pour les paramètres N, M, α et β pour lesquelles les résultats des tests montrent un comportement très homogène des méthodes testées, et par conséquent, un faible intérêt pour la recherche des nouvelles méthodes. Cette analyse a été réalisée avec la comparaison de deux indicateurs N^2/M et l'écart-type des solutions obtenues.

Les résultats de cette analyse sont proposés sur des tableaux pour chacune des valeurs testées du nombre de machines utilisées, et les 25 groupes définis. Ainsi, par exemple, pour les cas de 5 machines, pour des valeurs hautes de α et β, des instances d'au minimum 25 tâches doivent être considérées pour obtenir des conclusions lors de la comparaison de méthodes de résolution.

Également, les développements pour résoudre ce problème peuvent être utilisés dans la résolution de problèmes d'ordonnancement plus complexes comme c'est le cas de l'ordonnan-

cement d'atelier à cheminement unique avec des opérations où une plusieurs ressources sont disponibles (machines en parallèle). Pour cela, une procédure permettant d'identifier la méthode à utiliser dans la résolution de chaque instance est donnée. Cette procédure peut être facilement adaptée aux systèmes d'aide à la décision dans la gestion opérative des ateliers ayant la configuration décrite dans ce chapitre.