Dauphine | PSL
UNIVERSITÉ PARIS

Optimisation de l'ordonnancement par méthodes de Monte Carlo sur machines parallèles

Réalisé par :
DRIDI Yassine & Islem Fatma Gamha

16 février 2026

Table des matières

Résumé

Les problèmes d'ordonnancement constituent une classe fondamentale de problèmes en optimisation combinatoire, avec de nombreuses applications industrielles telles que la planification de la production, la gestion des ressources informatiques ou encore l'organisation de projets complexes. Ces problèmes sont généralement NP-difficiles, ce qui rend les méthodes exactes impraticables dès que la taille des instances augmente.

Dans ce rapport, nous étudions l'utilisation des méthodes de Monte Carlo pour résoudre un problème d'ordonnancement sur machines parallèles non identiques. L'objectif est de minimiser la durée totale de production (makespan) et, lorsque des contraintes temporelles sont présentes, de réduire les retards par rapport aux deadlines. Nous proposons un prototype complet permettant de générer des instances synthétiques, de simuler des plannings, d'évaluer leur qualité à l'aide de métriques pertinentes, et d'améliorer progressivement les solutions à l'aide de méthodes Monte Carlo simples et guidées.

Plusieurs approches sont implémentées et comparées, notamment des heuristiques classiques de référence, une recherche Monte Carlo par échantillonnage aléatoire, et une méthode Monte Carlo guidée basée sur le recuit simulé. Les résultats expérimentaux montrent que la recherche Monte Carlo améliore significativement une baseline aléatoire, tandis que le recuit simulé permet d'obtenir des plannings de très haute qualité, avec un makespan minimal et un retard nul, pour un coût de calcul raisonnable. Ces résultats confirment l'intérêt des méthodes de Monte Carlo guidées pour l'optimisation de problèmes d'ordonnancement complexes.

1 Introduction

L'ordonnancement des tâches est un problème central en optimisation combinatoire et en recherche opérationnelle. Il intervient dans de nombreux domaines tels que la production industrielle, les systèmes informatiques distribués, la logistique, ou encore la gestion de projets. Dans ces contextes, il s'agit d'organiser l'exécution d'un ensemble de tâches sur des ressources limitées, tout en respectant des contraintes opérationnelles et en optimisant des critères de performance.

Parmi les critères d'optimisation les plus couramment étudiés figurent la durée totale de production, appelée *makespan*, ainsi que le respect des délais imposés par des dates limites (*deadlines*). Un ordonnancement inefficace peut entraîner des retards de livraison, une mauvaise utilisation des ressources ou une augmentation significative des coûts de production. Il est donc crucial de disposer de méthodes capables de produire des plannings de bonne qualité dans des délais raisonnables.

D'un point de vue théorique, la majorité des problèmes d'ordonnancement réalistes sont connus pour être NP-difficiles. Cela signifie qu'il n'existe pas de méthode exacte permettant de résoudre toutes les instances en temps polynomial, sauf à considérer des problèmes de très petite taille. En pratique, cette complexité impose le recours à des méthodes approchées, telles que les heuristiques et les méta-heuristiques, qui sacrifient l'optimalité garantie au profit de solutions de bonne qualité obtenues dans un temps de calcul acceptable.

Les méthodes de Monte Carlo constituent une famille d'approches particulièrement adaptées à ce type de problèmes. Elles reposent sur l'exploration stochastique de l'espace des solutions et sur l'évaluation des candidats par simulation. Ces méthodes sont simples à mettre en œuvre, flexibles et facilement adaptables à différentes variantes du problème. Toutefois, une exploration purement aléatoire peut rapidement atteindre ses limites, en particulier lorsque l'espace des solutions est très vaste.

Dans ce projet, nous étudions plusieurs approches basées sur les méthodes de Monte Carlo pour résoudre un problème d'ordonnancement sur machines parallèles non identiques. Nous commençons par implémenter des heuristiques classiques de référence afin de disposer de points de comparaison solides. Nous introduisons ensuite une recherche Monte Carlo par échantillonnage aléatoire, puis une méthode Monte Carlo guidée basée sur le recuit simulé, qui combine exploration aléatoire et recherche locale.

L'objectif principal de ce travail est de concevoir un pipeline complet, allant de la génération d'instances synthétiques à la visualisation des plannings finaux, et d'analyser expérimentalement les performances des différentes méthodes proposées. Les résultats obtenus permettent de mettre en évidence l'intérêt des méthodes Monte Carlo guidées pour l'optimisation de problèmes d'ord

2 Le Problème

Le problème étudié dans ce projet appartient à la famille des problèmes d'ordonnancement sur machines parallèles. Ce type de problème est largement rencontré dans des contextes industriels réels, tels que les ateliers de production, les centres de calcul ou les plateformes de services, où plusieurs ressources sont disponibles pour exécuter un ensemble de tâches.

Contrairement au cas des machines identiques, nous considérons ici des *machines parallèles non identiques*. Dans ce modèle, les performances des machines diffèrent, et le temps nécessaire à l'exécution d'une tâche dépend de la machine sur laquelle elle est affectée. Cette hypothèse rend le problème plus général et plus réaliste, mais également plus complexe à résoudre.

Le système étudié est composé d'un ensemble fini de tâches, toutes disponibles au début de l'horizon de planification, et d'un ensemble de machines supposées toujours disponibles. Chaque tâche doit être exécutée exactement une fois, sur une seule machine, et ne peut pas être interrompue une fois son exécution commencée. De plus, une machine ne peut traiter qu'une seule tâche à la fois.

L'objectif principal est de déterminer à la fois l'affectation des tâches aux machines et leur ordre d'exécution sur chaque machine, de manière à minimiser la durée totale de production, appelée *makespan*. Lorsque des contraintes temporelles supplémentaires sont introduites sous la forme de deadlines, un objectif secondaire consiste à limiter les retards par rapport à ces dates limites.

La difficulté du problème provient du nombre extrêmement élevé de plannings possibles. Même pour des instances de taille modérée, l'espace des solutions est trop vaste pour être exploré de manière exhaustive. Cette explosion combinatoire justifie l'utilisation de méthodes stochastiques, capables d'explorer efficacement l'espace des solutions sans en examiner toutes les configurations possibles.

Dans ce contexte, les méthodes de Monte Carlo apparaissent comme une approche particulièrement adaptée, car elles permettent de combiner génération aléatoire de solutions, évaluation par simulation et amélioration progressive des plannings.

2.1 Modélisation formelle du problème

Nous considérons un ensemble de tâches

$$J = \{0, 1, \ldots, n-1\}$$

et un ensemble de machines parallèles

$$M = \{0, 1, \ldots, m-1\}.$$

Chaque tâche $j \in J$ doit être exécutée exactement une fois, sans préemption, sur une seule machine. Le temps de traitement de la tâche j sur la machine k est noté $p_{j,k}$ et peut varier d'une machine à l'autre, ce qui caractérise le modèle des machines parallèles non identiques.

Un planning est représenté par une famille de séquences :

$$S = (\pi_0, \pi_1, \ldots, \pi_{m-1}),$$

où π_k désigne la liste ordonnée des tâches affectées à la machine k. Cette représentation permet de décrire à la fois l'affectation des tâches aux machines et leur ordre d'exécution.

Un planning est dit valide s'il respecte les contraintes suivantes :

— chaque tâche apparaît exactement une fois dans l'union des séquences π_k ;
— une machine ne peut exécuter qu'une seule tâche à la fois ;
— les tâches ne sont pas préemptives.

À partir d'un planning donné, les temps de début S_j et de fin C_j de chaque tâche sont calculés par simulation, en tenant compte de l'ordre d'exécution sur les machines. Le critère principal d'optimisation est le makespan, défini par :

$$C_{\max} = \max_{j \in J} C_j.$$

Lorsque des deadlines d_j sont associées aux tâches, nous introduisons également la notion de retard :

$$T_j = \max(0, C_j - d_j).$$

Afin de comparer efficacement les solutions dans un cadre Monte Carlo, nous utilisons une fonction objectif agrégée, combinant le makespan et le retard total :

$$\text{score} = \alpha C_{\max} + \beta \sum_{j \in J} T_j,$$

où α et β sont des coefficients de pondération fixés expérimentalement.

3 Solutions proposées

Afin d'évaluer les méthodes proposées dans ce projet, nous utilisons des instances synthétiques générées aléatoirement. Ce choix permet de contrôler précisément la taille du problème, la difficulté des instances, ainsi que les paramètres influençant la structure des plannings.

Les temps de traitement $p_{j,k}$ sont générés de manière aléatoire selon une loi uniforme discrète sur un intervalle fixé. Cette génération introduit une hétérogénéité entre les machines et reflète des différences de performance réalistes entre les ressources.

Lorsque des deadlines sont considérées, celles-ci sont construites à partir d'une estimation de la charge moyenne du système. Plus précisément, une estimation du makespan moyen est calculée à partir des durées de traitement, puis les deadlines sont définies comme une fraction de cette valeur, à laquelle est ajouté un bruit aléatoire modéré. Cette approche permet de générer des instances ni trop laxistes, ni trop contraignantes.

Chaque instance est caractérisée par :

- le nombre de tâches ;
- le nombre de machines ;
- la matrice des temps de traitement $p_{j,k}$;
- éventuellement, un ensemble de deadlines associées aux tâches.

Afin de garantir la reproductibilité des expériences, chaque instance est générée à partir d'une graine aléatoire fixée et sauvegardée au format JSON. Ce format facilite la réutilisation des instances dans l'ensemble du pipeline expérimental, depuis la simulation jusqu'à l'évaluation des méthodes Monte Carlo.

3.1 Simulation et métriques

L'évaluation d'un planning donné repose sur une simulation temporelle détaillée de son exécution. Cette simulation constitue un élément central du projet, car elle permet de comparer objectivement des solutions candidates générées par différentes méthodes d'optimisation.

À partir d'un planning représenté par des séquences de tâches sur chaque machine, la simulation reconstruit les temps de début et de fin de chaque tâche. Sur une machine donnée, les tâches sont exécutées de manière strictement séquentielle, dans l'ordre imposé par le planning. Le début d'une tâche dépend à la fois de la fin de la tâche précédente sur la machine et, le cas échéant, de sa date de disponibilité.

La simulation permet ainsi de calculer, pour chaque tâche j :

— son temps de début S_j,
— son temps de fin C_j.

À partir de ces valeurs, plusieurs métriques globales sont définies afin d'évaluer la qualité d'un planning. Le critère principal est le *makespan*, défini comme le temps de fin de la dernière tâche :

$$C_{\max} = \max_{j \in J} C_j.$$

Lorsque des deadlines d_j sont associées aux tâches, nous introduisons également la notion de retard. Le retard d'une tâche est défini par :

$$T_j = \max(0, C_j - d_j).$$

Le retard total correspond alors à la somme des retards sur l'ensemble des tâches :

$$\sum_{j \in J} T_j.$$

Afin de comparer efficacement les solutions dans un cadre Monte Carlo, nous utilisons une fonction objectif agrégée combinant ces métriques :

$$\text{score} = \alpha C_{\max} + \beta \sum_{j \in J} T_j,$$

où α et β sont des coefficients de pondération fixés expérimentalement. Cette formulation permet de réduire le problème multi-critères à une optimisation scalaire, tout en conservant une interprétation claire des résultats.

3.2 Méthodes d'optimisation

Dans ce projet, plusieurs méthodes de résolution ont été implémentées afin d'évaluer progressivement l'apport des méthodes de Monte Carlo pour l'ordonnancement. Ces méthodes vont de simples heuristiques de référence à des approches Monte Carlo guidées plus élaborées.

Toutes les méthodes produisent des plannings complets, qui sont ensuite évalués à l'aide du simulateur décrit précédemment. Les performances sont comparées sur la base des métriques définies à la Section 5.

3.3 Heuristiques de référence

Avant d'introduire des méthodes Monte Carlo, plusieurs heuristiques classiques ont été implémentées afin de servir de baselines.

La première méthode est la baseline **Random**. Elle consiste à assigner chaque tâche à une machine choisie aléatoirement, puis à ordonner les tâches sur chaque machine de manière aléatoire. Cette méthode ne tient compte d'aucune information du problème et génère des plannings de faible qualité, mais elle permet de mesurer le gain apporté par des approches plus structurées.

La règle **SPT** (*Shortest Processing Time*) ordonne les tâches par durée moyenne de traitement croissante. Cette heuristique est connue pour réduire efficacement le temps moyen de traitement et donne souvent de bons résultats lorsque l'objectif principal est la minimisation du makespan.

La règle **EDD** (*Earliest Due Date*) ordonne les tâches selon leurs deadlines croissantes. Elle est particulièrement adaptée aux problèmes dans lesquels le respect des délais est un critère important. Sur certaines instances, cette heuristique permet d'obtenir des plannings sans retard.

3.4 Monte Carlo Random Search

La méthode Monte Carlo Random Search repose sur un échantillonnage aléatoire de l'espace des solutions. À chaque itération, un planning valide est généré aléatoirement, puis évalué à l'aide du simulateur.

Le meilleur planning rencontré depuis le début de la recherche est conservé selon une stratégie *best-so-far*. Cette approche permet d'améliorer significativement la baseline Random, simplement en sélectionnant les meilleures solutions parmi un grand nombre d'échantillons.

Cependant, cette méthode présente une limitation importante : chaque solution est générée indépendamment des précédentes, ce qui empêche toute exploitation de l'information contenue dans les bonnes solutions déjà identifiées. En pratique, cela conduit souvent à une stagnation des performances après un certain nombre d'itérations.

3.5 Monte Carlo guidé : recuit simulé

Le recuit simulé (*Simulated Annealing*) est une méthode Monte Carlo guidée combinant exploration aléatoire et recherche locale. L'algorithme démarre à partir d'un planning initial, généralement généré aléatoirement, puis explore l'espace des solutions à l'aide de

mouvements locaux.

Les voisinages utilisés dans ce projet incluent :

— l'échange de deux tâches sur une même machine ;
— le déplacement d'une tâche d'une machine vers une autre ;
— l'échange de tâches entre deux machines.

À chaque itération, un planning voisin est évalué. S'il améliore la fonction objectif, il est systématiquement accepté. Dans le cas contraire, il peut être accepté avec une probabilité dépendant d'un paramètre de température.

La température décroît progressivement au cours de l'algorithme, ce qui permet une exploration large au début de la recherche, puis une exploitation plus fine des meilleures régions de l'espace des solutions. Cette stratégie permet d'échapper aux minima locaux et d'obtenir des solutions de très haute qualité.

4 Discussions

Les méthodes présentées dans ce projet ont été évaluées à l'aide d'un protocole expérimental visant à garantir la robustesse et la reproductibilité des résultats.

Les expériences sont menées sur des instances synthétiques générées aléatoirement, composées principalement de 50 tâches et 5 machines parallèles non identiques. Des deadlines sont associées aux tâches afin d'introduire une contrainte temporelle réaliste.

Les méthodes stochastiques (Random, Monte Carlo Random Search et recuit simulé) sont exécutées sur plusieurs graines aléatoires. Cela permet d'évaluer la stabilité des résultats et de réduire l'impact du hasard sur les conclusions.

Les paramètres algorithmiques sont fixés de manière à assurer un compromis raisonnable entre qualité des solutions et temps de calcul. Pour chaque méthode, les performances sont mesurées en termes de makespan, de retard total et de score global.

Les résultats sont ensuite agrégés sous forme de moyennes, d'écarts-types, de minima et de maxima, ce qui permet une comparaison statistique fiable entre les différentes approches.

4.1 Analyse des résultats

Cette section présente les résultats expérimentaux obtenus pour comparer les différentes méthodes d'ordonnancement implémentées : baselines (Random, SPT, EDD), Monte Carlo Random Search (MC) et recuit simulé (SA). Les expériences ont été conduites sur des instances synthétiques générées selon le protocole décrit au niveau des notebooks.

4.2 Comparaison quantitative des méthodes

Nous commençons par une comparaison quantitative des méthodes à partir d'exécutions répétées (plusieurs graines aléatoires) pour les méthodes stochastiques. Les métriques étudiées sont :

— le makespan $C_{\max}$,
— le retard total $\sum_j T_j$,
— le score agrégé score $= \alpha C_{\max} + \beta \sum_j T_j$.

Tableau récapitulatif (moyenne/écart-type). Le Tableau 1 synthétise les performances agrégées par méthode. Pour les méthodes déterministes (SPT, EDD), une seule exécution suffit (écart-type nul). Pour les méthodes stochastiques, les statistiques sont calculées sur plusieurs seeds.

TABLE 1 – Résumé statistique des performances par méthode (moyenne, écart-type, min, max).

Méthode	score_mean	score_std	Cmax_mean	sumT_mean
Greedy EDD	51.0	0.0	51.0	0.0
Greedy SPT	59.0	0.0	59.0	0.0
SA (iters=5000)	52.0	1.0	52.0	0.0
MC Random (K=5000)	112.0	3.0	112.0	0.0
Random	254.0	59.0	158.0	479.0

Ce tableau met en évidence trois faits principaux. Premièrement, la baseline **Random** produit des solutions de faible qualité, avec des scores élevés et une variabilité importante. Deuxièmement, **Monte Carlo Random Search** améliore nettement Random en sélectionnant le meilleur échantillon parmi K solutions, mais reste globalement inférieur aux méthodes guidées. Troisièmement, le **recuit simulé** atteint des scores très faibles et stables, comparables (voire meilleurs) que les heuristiques gloutonnes, tout en restant générique.

Figure : score moyen par méthode. La Figure 1 illustre visuellement la comparaison par score moyen. Elle permet de constater immédiatement l'écart important entre Random et les méthodes structurées, ainsi que la supériorité du recuit simulé par rapport à MC Random Search.

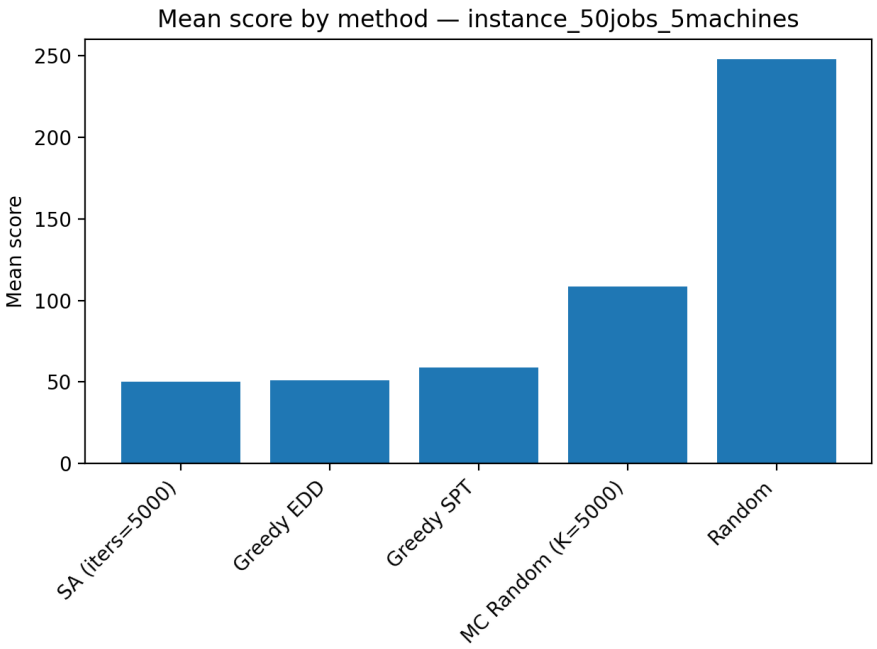

FIGURE 1 – Score moyen par méthode sur l'instance `instance_50jobs_5machines`.

4.3 Analyse de la convergence

Au-delà du score final, il est crucial d'analyser la dynamique de recherche. Nous comparons donc l'évolution du meilleur score rencontré (*best-so-far*) en fonction du nombre d'évaluations.

Figure : convergence MC vs SA. La Figure 2 compare la convergence de Monte Carlo Random Search et du recuit simulé. La recherche Monte Carlo par échantillonnage aléatoire améliore rapidement au début, mais atteint un *plateau* : les améliorations deviennent rares car chaque solution est générée indépendamment, sans exploitation des bonnes configurations déjà identifiées.

À l'inverse, le recuit simulé continue à améliorer la solution grâce à : (i) l'exploration locale via les voisinages, et (ii) l'acceptation probabiliste de certaines dégradations, qui aide à éviter les minima locaux.

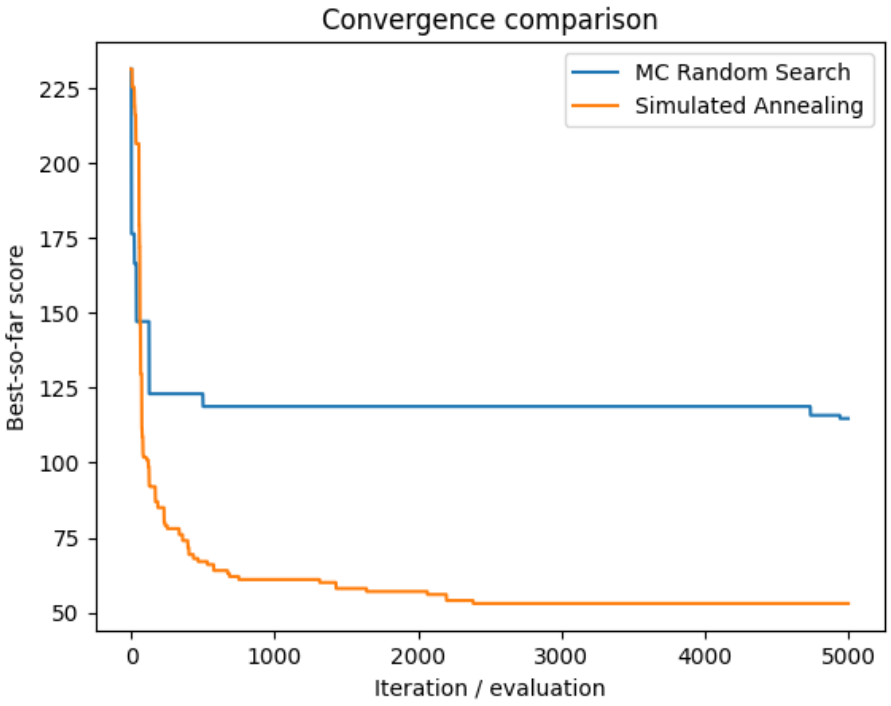

FIGURE 2 – Courbes de convergence (best-so-far) : MC Random Search vs Recuit simulé.

4.4 Analyse qualitative du planning final

Pour compléter l'analyse quantitative, nous présentons le planning final obtenu par recuit simulé sous forme de diagramme de Gantt. Cette visualisation permet de vérifier qualitativement la cohérence du planning, la répartition de charge et l'éventuelle présence de goulots d'étranglement.

Figure : diagramme de Gantt. La Figure 3 présente le diagramme de Gantt du meilleur planning SA. Sur l'instance étudiée, on observe une bonne répartition de la charge entre machines, un niveau d'inactivité limité, et un makespan réduit. De plus, la présence d'un retard total nul ($\sum_j T_j = 0$) indique que toutes les tâches respectent les deadlines.

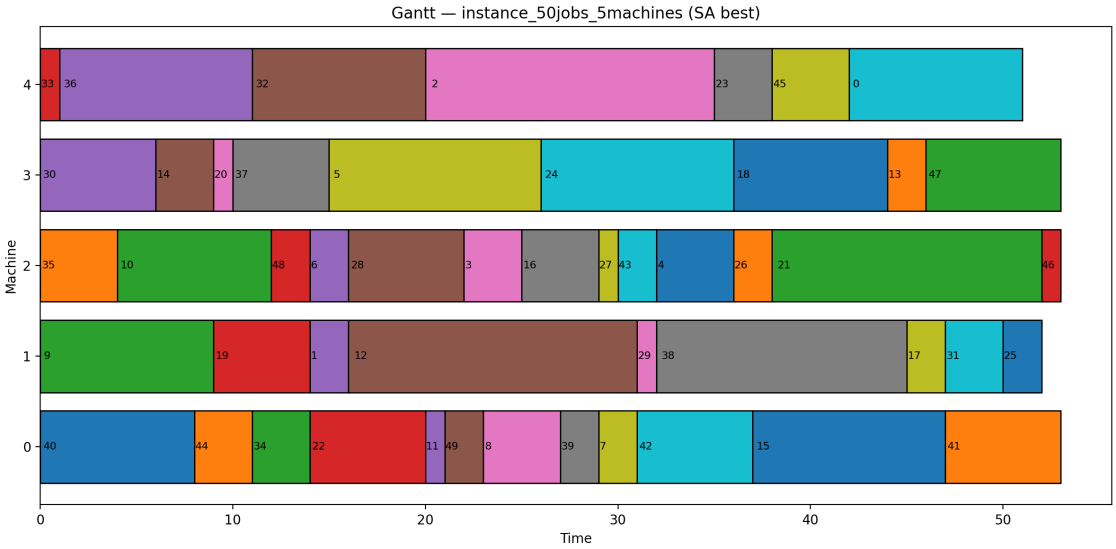

FIGURE 3 – Diagramme de Gantt du meilleur planning obtenu par recuit simulé.

4.5 Synthèse

Les résultats expérimentaux montrent que :

- les heuristiques gloutonnes (SPT/EDD) donnent une référence déterministe forte ;
- MC Random Search améliore fortement Random mais atteint rapidement un plateau ;
- le recuit simulé (Monte Carlo guidé) obtient les meilleures solutions observées, avec un excellent compromis qualité/temps de calcul.

Ces observations motivent l'utilisation de méthodes Monte Carlo guidées pour des problèmes d'ordonnancement combinatoires, en particulier lorsque la structure des instances rend les heuristiques simples potentiellement sous-optimales.

4.6 Discussion et perspectives

Les résultats expérimentaux obtenus dans ce projet mettent clairement en évidence l'intérêt des méthodes de Monte Carlo pour les problèmes d'ordonnancement complexes. Ils permettent également de mieux comprendre les forces et les limites des différentes approches étudiées.

La baseline Random, bien que simple, joue un rôle important en fournissant un point de référence faible. Les performances très limitées de cette méthode illustrent la difficulté intrinsèque du problème et justifient l'utilisation de méthodes plus élaborées.

La recherche Monte Carlo Random Search constitue une amélioration significative par

rapport à cette baseline. Toutefois, l'absence de mécanisme d'exploitation empêche cette méthode de tirer pleinement parti des bonnes solutions déjà identifiées. Cela se traduit par une stagnation des performances après un certain nombre d'itérations.

Le recuit simulé se distingue par sa capacité à combiner exploration et exploitation. L'acceptation probabiliste de solutions dégradées joue un rôle clé dans l'évitement des minima locaux, tandis que la décroissance progressive de la température permet une exploitation fine des meilleures régions de l'espace des solutions.

Un résultat particulièrement intéressant est que le recuit simulé parvient à rivaliser avec des heuristiques spécialisées telles que SPT et EDD, malgré son caractère générique. Cela suggère que les méthodes Monte Carlo guidées constituent une alternative robuste et flexible aux heuristiques classiques, en particulier lorsque la structure du problème devient plus complexe.

Enfin, il convient de noter que les résultats présentés ici portent sur des instances synthétiques de taille modérée. Bien que représentatives, ces instances ne couvrent pas l'ensemble des situations rencontrées en pratique, ce qui ouvre la voie à des travaux complémentaires.

5 Conclusion

Dans ce projet, nous avons étudié l'application des méthodes de Monte Carlo à un problème d'ordonnancement sur machines parallèles non identiques. Un prototype complet a été développé, couvrant l'ensemble du pipeline d'optimisation, depuis la génération des instances jusqu'à la visualisation des plannings finaux.

Plusieurs approches ont été implémentées et comparées, incluant des heuristiques classiques de référence, une recherche Monte Carlo par échantillonnage aléatoire, et une méthode Monte Carlo guidée basée sur le recuit simulé. Les résultats expérimentaux montrent que si la recherche Monte Carlo aléatoire permet déjà d'améliorer une baseline naïve, le recuit simulé se révèle nettement supérieur en termes de qualité des solutions et de stabilité.

En particulier, le recuit simulé permet d'obtenir des plannings de très haute qualité, caractérisés par un makespan minimal et un retard nul, pour un coût de calcul raisonnable. Ces résultats confirment l'intérêt des méthodes de Monte Carlo guidées pour l'optimisation de problèmes d'ordonnancement complexes.

Plusieurs perspectives peuvent être envisagées pour prolonger ce travail. Parmi celles-ci figurent l'étude de méthodes Monte Carlo plus avancées, telles que le Monte Carlo Tree Search, l'hybridation avec des heuristiques déterministes, l'extension à des formulations multi-objectifs, ou encore l'application de ces méthodes à des données industrielles réelles.

A Annexe A : Jupyter Notebook

Cette annexe contient le code source et l'analyse détaillée effectués dans le cadre de ce projet. Vous pouvez retrouver l'implémentation complète dans le fichier `.ipynb` joint à ce rapport.