Etude de codages et voisinages d'un espace de recherche. Application à l'ordonnancement de tâches dans des cas contraints.

Israël Tsogbetse

THÈSE DE DOCTORAT DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTE

PREPARÉE À l'UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

École doctorale n° 37

SCIENCES PHYSIQUES POUR L'INGÉNIEUR ET MICROTECHNIQUES

Doctorat en Automatique

Par

TSOGBETSE Israël Kodjo

Étude de codages et voisinages d'un espace de recherche. Application à l'ordonnancement de tâches dans des cas contraints

Thèse présentée et soutenue à Belfort, le 12 février 2024

Composition du Jury :

BRAUNER Nadia	Professeur, Université Grenoble Alpes	Présidente
DEROUSSI Laurent	Maître de conférences HDR, Université Clermont Auvergne	Rapporteur
GUÉRET (JUSSIEN) Christelle	Professeur, Université d'Angers	Rapportrice
NICOD Jean-Marc	Professeur, SUPMICROTECH-ENSMM	Examinateur
RAMDANE-CHERIF Wahiba	Maître de conférences, Mines Nancy	Examinatrice
MANIER Marie-Ange	Professeur, Université de Technologie de Belfort Montbéliard	Directrice de thèse
BERNARD Julien	Maître de conférences, Université de Franche-Comté	Co-encadrant
MANIER Hervé	Maître de conférences, Univ. de Technologie de Belfort Montbéliard	Co-encadrant

Remerciements

J'aimerais à travers ces quelques lignes adresser ma reconnaissance aux personnes qui ont apporté un plus à ce travail.

Je tiens à exprimer ma profonde gratitude envers Marie-Ange Manier, Julien Bernard et Hervé Manier pour l'excellence de leur encadrement, leur soutien constant, et leurs directives éclairantes tout au long de cette période dédiée à ma thèse. Leur disponiblité et leur expertise ont joué un rôle essentiel dans ma progression scientifique. Les échanges et les activités que nous avons partagés ont grandement contribué à enrichir mon parcours de recherche. Je souhaite également remercier chaleureursement Sid Lamrous pour sa collaboration scientifique avec l'équipe.

Je remercie mon ancienne collègue de bureau Linfei Feng pour son soutien et pour tous les moments de convivialité que nous avons partagés. Mes remerciements s'étendent également à l'ensemble des membres de FEMTO-ST/DISC/OMNI pour les discussions informelles et scientifiques autour de sujets variés.

Je remercie mes collègues et amis doctorants qui ont partagé mon quotidien, égayant nos journées de cafés, de déjeuners, de discussions animées, et même de débats scientifiques passionnants. Un merci particulier à mes compagnons Grace, Bernard, Amine, Francis, Mira, Baptiste, Komi, Médane, Théresia, Teng, Réné, pour ne nommer que ceux-ci.

Je tiens à remercier mes parents et très sincèrement ma compagne. Leur dévouement et leurs efforts incessants n'ont ménagé aucune peine pour me procurer l'énergie nécessaire et me permettre de me consacrer aux études tout au long de ces années.

Pour conclure, mes remerciements s'adressent à Nadia Brauner, présidente du jury, mais aussi à Laurent Deroussi et Christelle Guéret pour avoir rapporté studieusement ce travail. J'exprime également ma reconnaissance envers les examinateurs Jean-Marc Nicod et Wahiba Ramdane-Cherif dont les commentaires et apports ont été grandement appréciés.

Résumé

Les métaheuristiques sont des méthodes de résolution de problèmes d'optimisation qui se basent majoritairement sur une représentation abstraite des solutions sous forme de codage direct ou indirect. L'amélioration d'une solution ou d'un ensemble de solutions en parallèle se fait via des manipulations de ces codages et des algorithmes d'évaluation de la qualité des solutions associées. Le passage d'une solution à une autre implique l'utilisation d'un ou plusieurs opérateurs qui permettent d'explorer l'espace de recherche. En général, les métaheuristiques exploitent ces opérateurs pour améliorer itérativement les solutions jusqu'à l'atteinte d'un optimum local (ou global). Une pléthore de métaheuristiques a été proposée pour résoudre les problèmes d'optimisation combinatoire dont les problèmes d'ordonnancement de tâches. Elles sont souvent dédiées à des classes spécifiques d'instances. Dans ce contexte, les chercheurs proposent fréquemment des algorithmes qui combinent diverses méthodes, en s'attachant à ajuster au mieux les paramètres des différentes parties de leurs algorithmes, mais les performances obtenues restent souvent comparables et l'efficacité dépendante de la classe d'instances résolues. Bien que les codages de solutions et les opérateurs de voisinage soient reconnus comme étant des composants essentiels au sein des métaheuristiques, ils sont rarement examinés conjointement de manière analytique et scientifique.

Cette thèse se propose de caractériser les codages de solutions et les opérateurs de voisinage usuellement utilisés en ordonnancement, et particulièrement pour le problème de type job shop et une de ses variantes, pour lesquels on cherche à minimiser le makespan. L'ambition est d'exploiter les propriétés des espaces de recherche induits par ces codages et opérateurs, dans le but d'améliorer la conception des métaheuristiques. La démarche que nous avons appliquée dans notre étude est structurée en deux volets principaux avec une gradation de la complexité du problème de job shop. Le premier volet porte sur la caractérisation d'espaces de recherche à travers une analyse de paysage de fitness, en utilisant des métriques issues de la littérature. Le second volet concerne l'évaluation des performances de diverses associations de codages et d'opérateurs de voisinage dans l'optique de dégager d'éventuelles corrélations avec les propriétés du paysage, pour l'émission de préconisations pour la conception de métaheuristiques. Cette démarche est utilisée dans un premier temps pour un job shop de base, puis pour une variante plus contrainte du job shop : le job shop flexible avec contraintes de transport. Nos travaux mettent en évidence la difficulté de lier les performances des associations testées avec les métriques usuelles. La comparaison des résultats obtenus pour le problème de base et sa variante plus contrainte nous amènent à émettre des réserves sur une généralisation systématique des caractéristiques des codages et des opérateurs pour cette catégorie de problème d'optimisation.

Mots clés : Ordonnancement d'atelier de type job shop, codages et opérateurs de recherche locale, métaheuristique, paysage de fitness, flexibilité, ressources de transport.

Abstract

Metaheuristics are optimization problem-solving methods that primarily rely on an abstract representation of solutions in the form of direct or indirect encoding. Improving a solution or a set of solutions in parallel is achieved through manipulations of these encodings and algorithms evaluating the quality of associated solutions. The transition from one solution to another involves the use of one or more operators to explore the search space. Generally, metaheuristics utilize these operators to iteratively enhance solutions until reaching a local (or global) optimum. A plethora of metaheuristics has been proposed to address combinatorial optimization problems, including task scheduling problems. These ones are often dedicated to specific classes of instances. In this context, researchers frequently propose algorithms that combine various methods, striving to optimize parameters across different parts of their algorithms. However, the achieved performance is often comparable, and efficiency depends on the class of instances addressed. While solution encodings and neighborhood operators are recognized as essential components within metaheuristics, they are rarely jointly examined in an analytical and scientific manner.

This thesis aims to characterize solution encodings and neighborhood operators commonly used in scheduling, particularly for the job shop problem and for one of its variants, in which the objective is to minimize the makespan. The ambition is to exploit the properties of the search spaces induced by these encodings and operators to enhance the design of metaheuristics. The approach applied in our study is structured into two main parts, with a gradation in the complexity of the job shop problem. The first part focuses on characterizing search spaces through a fitness landscape analysis, using metrics from the literature. The second part involves evaluating the performance of various combinations of encodings and neighborhood operators with the aim of identifying potential correlations with landscape properties. This is done to provide recommendations for the design of metaheuristics. This approach is initially applied to a basic job shop and then to a more constrained variant : the flexible job shop with transportation constraints. Our work highlights the challenge of linking the performance of tested combinations with standard metrics. The comparison of results obtained for the basic problem and its more constrained variant leads us to express reservations about a systematic generalization of encoding and operator characteristics for this category of optimization problems.

Keywords : Job shop scheduling, encodings and local search operators, metaheuristics, fitness landscape, flexibility, transportation resources.

Table des matières

Introduction générale **1**

1 Ordonnancement d'atelier et analyse de paysage de fitness **5**
1.1 Ordonnancement d'atelier . . . 6
1.1.1 Concepts généraux en ordonnancement . . . 6
1.1.2 Propriétés des ordonnancements . . . 7
1.1.3 Classification des problèmes d'atelier . . . 9
1.1.4 Problème d'ordonnancement job shop et extensions . . . 11
1.1.5 Résolution des problèmes d'ordonnancement d'atelier . . . 12
1.1.6 Composants classiques des métaheuristiques . . . 14
1.2 Analyse de paysage de fitness . . . 20
1.2.1 Notions préliminaires et outils de mesure . . . 22
1.2.2 Propriétés et métriques . . . 23
1.3 Bilan . . . 30

2 État de l'art **31**
2.1 Résolution des problèmes de type job shop . . . 32
2.1.1 Métaheuristiques pour le job shop classique . . . 32
2.1.2 Métaheuristiques pour les variantes de job shop . . . 34
2.1.3 Synthèse des métaheuristiques présentées pour les problèmes de type job shop . . . 36
2.2 Codages et opérateurs de voisinage en ordonnancement d'atelier . . . 39
2.2.1 Codages de solution . . . 39
2.2.2 Opérateurs de voisinage . . . 55
2.3 Paysages de fitness et problèmes d'ordonnancement . . . 58
2.4 Bilan . . . 61

3 Caractérisation de codages et voisinages pour le job shop classique **65**
3.1 Introduction . . . 66
3.2 Description du problème traité . . . 66
3.3 Codages et opérateurs de voisinage . . . 67
3.3.1 Nouvelle proposition de codage . . . 67
3.3.2 Les codages et opérateurs de voisinage retenus . . . 69
3.3.3 Moteurs d'ordonnancement . . . 69
3.4 Utilisation des espaces de recherche . . . 70
3.4.1 Marche aléatoire . . . 70
3.4.2 Tirage aléatoire uniforme . . . 71

3.4.3 Recherche locale . . . 71
3.5 Instances JSP . . . 74
3.6 Caractérisation des paysages pour le JSP . . . 75
3.6.1 Distance de corrélation . . . 76
3.6.2 Taux de neutralité . . . 79
3.6.3 Probabilité d'échappement cumulée . . . 80
3.6.4 Distribution des types de points . . . 81
3.6.5 Synthèse sur la caractérisation des paysages . . . 82
3.7 Performance d'une métaheuristique : recherche taboue . . . 83
3.7.1 Résultats des tests . . . 83
3.7.2 Synthèse sur les performances . . . 88
3.8 Bilan . . . 88

4 Caractérisation de codages et voisinages pour le FJSPT 91
4.1 Introduction . . . 92
4.2 Description de la variante FJSPT . . . 92
4.3 Codages et opérateurs de voisinage pour le FJSPT . . . 93
4.4 Moteurs d'ordonnancement . . . 94
4.5 Espaces de recherche . . . 96
4.6 Instances FJSPT . . . 96
4.7 Caractérisation des paysages pour le FJSPT . . . 98
4.7.1 Distance de corrélation . . . 98
4.7.2 Taux de neutralité . . . 100
4.7.3 Probabilité d'échappement cumulée . . . 102
4.7.4 Distribution des types de points . . . 104
4.7.5 Synthèse sur la caractérisation des paysages . . . 105
4.8 Performance avec la recherche taboue . . . 108
4.8.1 Résultats des tests . . . 108
4.8.2 Synthèse sur les performances . . . 112
4.9 Bilan . . . 113

Conclusion générale 115

Bibliographie 119

Annexe A Compléments performances de la recherche taboue sur les instances FJSPT 131

Table des figures

1.1 Exemple d'ordonnancement quelconque . . . 8
1.2 Exemple d'ordonnancement semi-actif . . . 8
1.3 Exemple d'ordonnancement actif . . . 9
1.4 Exemple d'ordonnancement sans délai . . . 9
1.5 Typologie des problèmes d'ordonnancement d'atelier . . . 10
1.6 Exemple de solution représentée par un codage direct . . . 15
1.7 Exemple de solution représentée par un codage indirect . . . 16
1.8 Composants classiques des métaheuristiques . . . 19
1.9 Représentation de paysage de fitness [Olson, 2013] . . . 20
1.10 Caractéristiques d'un paysage de fitness [Olson, 2020] . . . 21
1.11 Illustration de paysage doux, rugueux et extrêmement rugueux . . . 26
1.12 Influence de la rugosité sur l'évolutivité dans un paysage de fitness . . . 28
1.13 Types de points d'un paysage de fitness [Hoos and Stützle, 2005] . . . 29

2.1 Solution représentée avec le codage `job` . . . 41
2.2 Solutions représentées avec le codage `ope` . . . 42
2.3 Solutions représentées avec le codage `mch` . . . 42
2.4 Solution représentée avec le codage `mtx` . . . 43
2.5 Solution représentée avec le codage `rkey` . . . 44
2.6 Solution représentée avec le codage `osma` . . . 45
2.7 Solution représentée avec le codage `osmc` . . . 46
2.8 Solution représentée avec le codage `osta` . . . 46
2.9 Solution représentée avec le codage `rosta` . . . 47
2.10 Solution représentée avec le codage `tja` . . . 48
2.11 Solution représentée avec le codage `osra` . . . 48
2.12 Solution représentée avec le codage `tta` . . . 49
2.13 Solution représentée avec le codage `rajts` . . . 50
2.14 Illustration de l'opérateur de voisinage `ins` . . . 56
2.15 Illustration de l'opérateur de voisinage `swp` . . . 56
2.16 Illustration de l'opérateur de voisinage `rev` . . . 57

3.1 Solution représentée avec le codage `tim` . . . 67
3.2 Illustration du processus d'ordonnancement avec le codage `tim` . . . 68
3.3 Illustration de l'opérateur de voisinage `ins` modifié pour le codage `ope` . . . 74
3.4 Nombre d'instances par job et par machine [Strassl and Musliu, 2022] . . . 75
3.5 Description d'un `box-plot`[3] . . . 76

3.6 Distance de corrélation pour les instances classiques regroupées par codage-voisinage . . 77
3.7 Distance de corrélation pour les instances générées regroupées par codage-voisinage . . 78
3.8 Distance de corrélation pour les instances générées avec `tim-tim` 79
3.9 Taux de neutralité pour les instances classiques et générées 80
3.10 Probabilité d'échappement cumulée pour les instances classiques et générées 81
3.11 Distribution des types de points pour les instances classiques et générées 82
3.12 Rangs moyens de codage-voisinage en fonction des nombres de machines et de jobs . . . 87

4.1 Exemples d'ordonnancement avec différentes règles d'affectation 95
4.2 Distance de corrélation pour les instances FJSPT 98
4.3 Distance de corrélation regroupée par famille d'instances 99
4.4 Taux de neutralité pour les instances FJSPT 100
4.5 Taux de neutralité regroupé par famille d'instances 101
4.6 Probabilité d'échappement cumulée pour les instances FJSPT 102
4.7 Probabilité d'échappement cumulée regroupée par famille d'instances 103
4.8 Distribution des types de points regroupée par famille d'instances 106
4.9 Performance de codage-voisinage pour les instances FJSPT 109
4.10 Performance de codage-voisinage pour les instances FJSPT regroupées par famille 111

A.1 Nombre de meilleurs $C_{\max}$ trouvés pour chaque codage-voisinage 132
A.2 Nombre de meilleurs $C_{\max}$ trouvés pour chaque codage-voisinage par famille 133
A.3 Nombre de meilleurs $C_{\max}$ moyens pour chaque codage-voisinage 134
A.4 Nombre de meilleurs $C_{\max}$ moyens pour chaque codage-voisinage par famille 135

Liste des tableaux

1.1 Règles de priorité tirées de [Cheng et al., 1996] . . . 16
1.2 Techniques d'analyse de paysage de fitness de [Malan and Engelbrecht, 2013] . . . 23

2.1 Approches métaheuristiques pour les problèmes de type job shop . . . 37
2.2 Description des notations utilisées pour le JSP/FJSPT . . . 39
2.3 Durées opératoires de l'instance illustrative . . . 40
2.4 Temps de transport de l'instance illustrative . . . 40
2.5 Synthèse des caractéristiques des codages . . . 51
2.6 Nombre de solutions générés par les différents codages . . . 52
2.7 Codages et opérateurs de voisinage associés les plus courants dans la littérature . . . 63

3.1 Codages et opérateurs de voisinages investigués pour le JSP . . . 69
3.2 Instances JSP classiques . . . 75
3.3 Instances JSP générées (`gen`) de [Strassl and Musliu, 2022] . . . 75
3.4 Synthèse de l'analyse des paysages JSP avec les couples codage-voisinage . . . 83
3.5 Rangs de recherche taboue de codage-voisinage pour 242 instances classiques . . . 85
3.6 Rangs de recherche taboue de codage-voisinage pour 2500 instances générées . . . 86

4.1 Codages et opérateurs de voisinages investigués pour le FJSPT . . . 93
4.2 Instances FJSPT . . . 96
4.3 Ratios entre les temps de traitement et de transport . . . 97
4.4 Distribution des types de points (%) . . . 104
4.5 Synthèse de l'analyse des paysages FJSPT : neutralité, rugosité, évolutivité . . . 107
4.6 Rang moyen de codage-voisinage pour les instances FJSPT . . . 110

Introduction générale

Les problèmes d'optimisation combinatoires sont des problèmes qui impliquent une recherche dans un ensemble fini de solutions possibles, souvent définies par des variables discrètes. L'objectif est de trouver la meilleure solution parmi toutes les possibilités. Les problèmes d'optimisation combinatoire se rencontrent dans de nombreux domaines appliqués tels que la logistique, la gestion de production, la conception de réseaux, la planification des horaires de personnel médical et la gestion des risques en finance pour ne citer que ceux-ci. Parmi les exemples les plus courants de problèmes d'optimisation combinatoire se trouvent les problèmes de tournées au sens large, l'ordonnancement d'atelier, le sac à dos, le problème de coloration de graphe et l'affectation quadratique.

L'ordonnancement d'atelier, en particulier, est un problème de premier ordre en gestion de production parce qu'il régit fortement le pilotage opérationnel de l'industrie. Résoudre un problème d'ordonnancement implique d'affecter des ressources à un ensemble de tâches, et de trouver un ordre et des dates d'exécution pour ces tâches, dans le but de minimiser ou maximiser une fonction objectif. En raison des multiples combinaisons possibles, ces problèmes sont considérés comme complexes, nécessitant ainsi des méthodes de résolution élaborées. Deux grandes catégories de méthodes se dégagent dans ce contexte : les méthodes exactes, privilégiant une résolution optimale mais souvent limitées face à des problèmes de grande taille, et les méthodes approchées. Bien que ces dernières ne garantissent pas l'optimalité des solutions, elles offrent des solutions d'une qualité relativement intéressante. Au sein des méthodes approchées, les métaheuristiques et heuristiques sont largement répandues, comme en témoignent les revues dédiées à la résolution de divers problèmes d'ordonnancement d'atelier [Nouri et al., 2016a; Dhiflaouia et al., 2018; Zhang et al., 2019b; Dauzère-Pérès et al., 2023].

Dans une dynamique d'amélioration des performances des métaheuristiques, de nombreuses études se sont consacrées à la création de nouvelles adaptations d'algorithmes existants [Yan et al., 2021; Momenikorbekandi and Abbod, 2023], mais également et surtout à l'hybridation de divers algorithmes [Zhang et al., 2012; Li and Gao, 2016; Chen et al., 2020]. Il y a quelques années, [Sörensen, 2015] a invité les artisans de métaheuristiques à une évaluation plus critique de leurs méthodes. Ayant constaté la panoplie de méthodes d'optimisation existantes et la tendance croissante à en créer de nouvelles qui, pour la plupart, ne sont rien d'autre qu'une transmutation des anciennes, l'auteur a appelé à une consolidation des acquis. Cette consolidation passerait en partie par la déconstruction des méthodes, combinée à des tests statistiques sur les composants et les paramètres. Ceci pourrait révéler les composants et les paramètres qui contribuent réellement à l'efficacité des métaheuristiques.

Parmi les composants traditionnels des métaheuristiques, figurent les représentations de solutions, appelées codages, et les opérateurs de recherche. Dans [Rothlauf, 2006], Franz Rothlauf a rappelé l'influence des

codages sur le comportement et l'efficacité des algorithmes génétiques. En outre, il est revenu sur des travaux antérieurs, à l'instar de [Goldberg, 1989] et [Radcliffe, 1994], pour non seulement mettre en lumière quelques directives sur une conception appropriée des représentations, mais aussi montrer que les représentations de solution et les opérateurs de recherche dépendent les uns des autres. [Talbi, 2009], dans son livre, est allé dans le même sens en attestant que le codage joue un rôle majeur dans l'efficacité de toute métaheuristique et qu'il constitue une étape essentielle dans la conception d'une métaheuristique. Il ajouta que l'efficacité d'une représentation de solutions est également liée aux opérateurs de recherche appliqués sur cette représentation.

Le recours à une métaheuristique implique l'exploration d'un espace de recherche. L'exploration de l'espace s'inscrit dans un large plan dénommé paysage de fitness ; le paysage reflétant potentiellement toutes les possibilités d'exploration. [Marmion, 2013] a montré que la définition du paysage est étroitement liée à la dynamique des métaheuristiques, en se référant à la notion de voisinage et à la fonction d'évaluation. Cependant, pour un problème donné, l'ensemble des solutions du paysage est défini par la représentation des solutions et l'exploration est assurée par un opérateur de recherche avec le concours de la fonction d'évaluation. La représentation de solutions est donc significative tant pour les paysages que pour les métaheuristiques. La relation ainsi dépeinte entre les métaheuristiques et les paysages de fitness explique l'intérêt porté à l'analyse de paysage de fitness dans l'appréhension des problèmes d'ordonnancement.

Malgré le concours non négligeable des représentations de solutions et des opérateurs de recherche à l'efficacité des métaheuristiques, force est de constater que leur choix repose le plus souvent sur l'intuition et le hasard, sans aucun justificatif sur l'opportunité de leur usage. Subséquemment, au cas où ces choix s'avèrent inappropriés pour le problème considéré, ils peuvent conduire l'heuristique à des résultats équivoques. Ceci, en ce sens qu'il est possible que les conclusions de certains travaux en ce qui concerne l'efficacité des algorithmes soient erronées en raison de l'utilisation de composants inadéquats [Vlašic et al., 2020].

Sur la base des éléments énoncés ci-dessus, il ressort que le choix de bons codages et opérateurs de recherche pour un problème donné serait fondamental pour garantir la pertinence des performances de l'heuristique qui les comprend. La présente thèse intitulée «*Étude de codages et voisinages d'un espace de recherche. Application à l'ordonnancement de tâches dans des cas contraints*» s'inscrit dans une dynamique générale d'amélioration de la conception des algorithmes d'optimisation de type métaheuristique. L'idée fondamentale de la démarche consiste à trouver les raisons pour lesquelles tel ou tel autre choix de composant contribue à l'efficacité d'une heuristique. Plus précisément, il s'agit de mesurer la pertinence des codages en association avec des opérateurs de voisinage classiques et de formuler des préconisations concernant ces deux composants. Le problème d'optimisation considéré est celui du type job shop. Le plan consiste à dérouler notre approche méthodologique pour le job shop classique avant d'étendre l'étude à un cas plus complexe, en l'occurrence le job shop flexible avec des ressources de transport. L'extension de la démarche a pour but de déterminer l'impact des contraintes additionnelles sur un problème de base et d'observer si les résultats ou les choix d'éléments de métaheuristique pour ce problème peuvent être transposés pour des variantes plus contraintes. Pour répondre aux objectifs déclinés, pour chacun de nos deux problèmes, l'étude sera organisée en trois volets. Premièrement, nous caractériserons différentes associations de codages et d'opérateurs de voisinage (codage–voisinage) à l'aide d'une analyse de paysage de fitness. Ensuite, une étude comparative des performances en termes de convergence avec une recherche taboue sera menée. Enfin, une analyse synthétique sera effectuée dans le but d'établir des préconisations.

La suite de ce mémoire de thèse est structurée en quatre chapitres. Le premier chapitre expose et clarifie les concepts liés aux problèmes d'ordonnancement d'atelier, aux métaheuristiques, ainsi qu'à l'analyse de paysage de fitness. L'objectif est de garantir une bonne compréhension des travaux présentés dans le reste du manuscrit. Il propose une description des problèmes spécifiques de type job shop et procède à une synthèse des techniques de mesure des propriétés de paysage jugées adaptées à ce type de problème. En outre, le chapitre passe en revue les composants classiques des métaheuristiques.

Le deuxième chapitre examine les travaux existants dans la littérature sur la résolution approchée des problèmes de type job shop, dans le but de mettre en lumière la problématique abordée dans cette thèse. En plus de présenter les diverses métaheuristiques employées, il souligne le manque d'attention portée aux codages de solutions et aux opérateurs de voisinage, deux composants essentiels des métaheuristiques, dans le processus de conception des algorithmes d'optimisation. Ce chapitre détaille des codages de solutions adaptés pour le job shop et ses variantes, ainsi que des opérateurs de voisinage classiques. Il propose également une revue bibliographique sur l'utilisation de l'analyse de paysage de fitness dans le contexte des problèmes d'ordonnancement.

Dans le troisième chapitre, nous mettons en œuvre notre démarche sur la variante de base du problème de job shop. Dans une première phase, nous examinons les espaces de recherche générés par différentes combinaisons de codages et d'opérateurs de voisinage. Cette étude initiale repose sur l'analyse de quatre propriétés des paysages de fitness. Dans un second temps, nous évaluons la convergence de ces combinaisons en utilisant une métaheuristique de type recherche locale. Enfin, nous confrontons les caractéristiques des paysages aux performances pour identifier d'éventuelles corrélations et formuler des recommandations concernant les codages et les opérateurs de voisinage.

Le quatrième chapitre s'inscrit dans la continuité de l'étude menée au chapitre 3. Nous réitérons la même approche méthodologique, mais cette fois-ci en abordant une variante plus contrainte du problème : le job shop flexible avec ressources de transport. L'objectif ultime est de comparer les résultats obtenus avec ceux du job shop classique afin de déterminer la possibilité d'une généralisation des caractéristiques des codages et des opérateurs de voisinage pour cette catégorie de problème d'ordonnancement d'atelier.

Le manuscrit se termine par une conclusion générale dans laquelle nous résumons nos contributions et proposons quelques perspectives à nos travaux.

Les travaux présentés dans ce manuscrit ont fait l'objet de diverses communications devant la communauté scientifique locale, nationale et internationale, et d'un article en cours de révision dans un journal international référencé en Recherche Opérationnelle :

Article en revue internationale

Tsogbetse I., Bernard J., Manier H., and Manier M.-A., Influence of Encoding and Neighborhood in Landscape Analysis and Tabu Search Performance for Job Shop Scheduling Problem. Soumis à *European Journal of Operational Research*. En cours de 1re révision.

Communications dans des conférences d'audience internationale

Tsogbetse I., Bernard J., Manier H., and Manier M.-A., Fitness landscape analysis and tabu search for the Flexible Job shop Scheduling Problem with Transportation. *21th EU/ME meeting on Emerging optimization methods : from metaheuristics to quantum metaheuristics* (EU/ME'23), Troyes, France (17-21 april 2023),

Tsogbetse I., Bernard J., Manier H., and Manier M.-A., Impact of Encoding and Neighborhood on Landscape Analysis for the Job Shop Scheduling Problem. IFAC-PapersOnline *10th IFAC Conference on Manufacturing Modelling, Management and Control* (IFAC MIM'22), Nantes, France (22-24 june 2022), Volume 55, Issue 10, pp. 1237-1242. `https://doi.org/10.1016/j.ifacol.2022.09.559`

Communications dans des conférences d'audience nationale

Bernard J., Manier H., Manier M.-A. and Tsogbetse I., Recherche tabou pour le jobshop : temps fixe versus nombre d'itérations fixe. `Soumis à` *25ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (ROADEF'24), Amiens, France (4-7 mars 2024).

Tsogbetse I., Bernard J., Manier H., and Manier M.-A., Caractérisation d'espaces de recherche pour le problème de job shop flexible avec ressources de transport. *24ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (ROADEF'23), Rennes, France (20-23 février 2023). `https://roadef2023.sciencesconf.org/436003/`

Tsogbetse I., Bernard J., Manier H., and Manier M.-A., Étude de paysages de fitness pour l'ordonnancement d'atelier de type jobshop. *23ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision* (ROADEF'22), Lyon, France (23-25 février 2022). `https://roadef2022.sciencesconf.org/378763`

Communications à des séminaires internes

Tsogbetse I., Etude des voisinages d'un espace solution. Application à l'ordonnancement de tâches dans des cas contraints, *Journée des doctorants du DISC (Département Informatique des Systèmes Complexes)*, FEMTO-ST Institute, Mésandans (24 mars 2022).

Tsogbetse I., Manier M.-A., Bernard J., and Manier H., Study of the neighborhoods of a solution space Application to the scheduling of tasks in constrained cases, *Virtual poster session FEMTO-ST Ambassadors 2021 Contest* (15 June 2021).

Chapitre 1
Ordonnancement d'atelier et analyse de paysage de fitness

Ce premier chapitre introduit des généralités et des définitions de notions récurrentes en ce qui concerne l'ordonnancement, les problèmes d'atelier de production, les méthodes de résolution aussi bien que l'analyse de paysage de fitness et ses métriques. Ces éléments permettent de clarifier la terminologie adoptée dans la suite du document.

Contenu

1.1 Ordonnancement d'atelier . . . 6
- 1.1.1 Concepts généraux en ordonnancement . . . 6
- 1.1.2 Propriétés des ordonnancements . . . 7
- 1.1.3 Classification des problèmes d'atelier . . . 9
- 1.1.4 Problème d'ordonnancement job shop et extensions . . . 11
- 1.1.5 Résolution des problèmes d'ordonnancement d'atelier . . . 12
- 1.1.6 Composants classiques des métaheuristiques . . . 14

1.2 Analyse de paysage de fitness . . . 20
- 1.2.1 Notions préliminaires et outils de mesure . . . 22
- 1.2.2 Propriétés et métriques . . . 23

1.3 Bilan . . . 30

1.1 Ordonnancement d'atelier

Les systèmes de production industrielle sont généralement régis par un système de décision à trois niveaux hiérarchiques : le niveau stratégique, le niveau tactique et le niveau opérationnel. Les niveaux tactique et opérationnel se rapportent à la planification à moyen terme, au pilotage et au suivi quotidien des flux de matière et de travail. Les problèmes d'ordonnancement d'atelier de production trouvent leur place au sein des deux derniers niveaux de décision [Giard, 2003; Esquirol and Lopez, 2008].

L'ordonnancement est défini par [Pinedo, 2012] comme étant un processus décisionnel d'allocation de ressources à des tâches sur des périodes de temps données dans le but d'optimiser un ou plusieurs objectifs. Un problème d'ordonnancement comporte deux sous-problèmes distincts : le *séquencement*, qui se rapporte à l'acte de déterminer l'ordre dans lequel des tâches spécifiques doivent être exécutées, et l'*affectation*, qui implique l'attribution de ces tâches à des ressources. Résoudre un problème d'ordonnancement revient à planifier dans le temps l'exécution d'un ensemble de tâches en respectant concomitamment les contraintes de temps et les contraintes de capacité de ressources [Carlier and Chrétienne, 1988; Esquirol and Lopez, 1999]. Les problèmes d'ordonnancement sont des exemples classiques de problèmes d'optimisation combinatoire.

1.1.1 Concepts généraux en ordonnancement

Les concepts récurrents liés à la définition du problème d'ordonnancement sont décrits ci-après.

Tâche : c'est l'entité élémentaire de travail délimitée dans le temps par une date de début et/ou une date de fin. Sa réalisation nécessite une durée et une intensité de consommation de ressources en estimant que cette intensité est constante sur la durée de la réalisation. Dans certains cas, l'exécution d'une tâche peut être divisée en plusieurs intervalles temporels tandis que pour d'autres, une fois démarrée, la tâche ne peut être interrompue avant qu'elle ne soit complètement achevée. On parle respectivement de problèmes préemptifs et non-préemptifs.

En ordonnancement d'atelier classique, les tâches, encore appelées *opérations*, sont souvent regroupées en lots dénommés travaux ou *jobs*. À chaque job est associé une gamme opératoire qui n'est rien d'autre qu'une liste d'opérations. La gamme opératoire est qualifiée de libre lorsque l'ordre d'exécution des opérations est aléatoire. La gamme est linéaire lorsqu'un ordre total des opérations est prédéfini pour le job. Un job est également caractérisé par une date de disponibilité (date de début au plus tôt) et une date de livraison impérative ou non (date de fin au plus tard).

Ressource : c'est un moyen technique ou humain nécessaire à l'exécution d'une tâche. Elle est disponible en quantité limitée et sa capacité est le plus souvent supposée constante.

En fonction de leurs caractéristiques, plusieurs types de ressources sont observés. Une ressource est dite renouvelable si après avoir été utilisée par une ou plusieurs tâches, elle est à nouveau disponible avec la même capacité (ouvrier, machine, local, équipement, etc.). Dans le cas contraire, elle est dite non renouvelable ou consommable (matière première, budget, énergie, etc.). Une ressource est dite disjonctive lorsqu'elle ne peut exécuter qu'une seule tâche à la fois tandis qu'une ressource cumulative peut être utilisée pour plusieurs tâches simultanément en nombre limité cependant.

Contrainte : c'est une restriction sur les valeurs que peuvent prendre conjointement une ou plusieurs variables de décision comme les variables de séquencement ou les variables d'affectation. [Esquirol and Lopez, 1999] distingue deux groupes de contraintes : les contraintes temporelles et les contraintes de ressources.

- Contraintes temporelles : elles renferment d'une part des contraintes de temps alloué définissant les dates de disponibilité et les dates échues des tâches et d'autre part des contraintes d'antériorité (précédence). Les contraintes d'antériorité décrivent les positionnements relatifs devant être respectés entre les tâches ; elle représentent généralement des contraintes de gammes dans le cas des problèmes d'ordonnancement d'atelier.
- Contraintes de ressources : elles expriment la nature, la quantité et la disponibilité des moyens utilisés par les tâches. Pour les ressources renouvelables, à leur nature disjonctive ou cumulative sont associées des contraintes disjonctives et cumulatives.

D'autres contraintes fréquemment rencontrées sont entre autres [T'Kindt and Billaut, 2002; Zhang, 2012] :

- préemption/non-préemption : lorsque la préemption est autorisée, il est possible de prévoir l'interruption d'une tâche pour qu'elle puisse éventuellement être reprise ensuite par une autre ressource ;
- avec/sans attente : dans un système d'ordonnancement sans attente, les tâches constitutives d'un job se suivent immédiatement sans délai d'attente ;
- avec/sans stock : on parle de stock lorsqu'entre les machines, l'atelier de production dispose d'un espace de stockage dont la capacité est limitée ;
- avec/sans réentrance : la réentrance est une contrainte qui indique qu'un job peut éventuellement passer plusieurs fois sur la même machine lors de sa réalisation ;
- fenêtres temporelles : elles font référence à des contraintes particulières comme les écarts minima et/ou maxima entre les tâches ; on parle de time lags.

Objectif : la résolution d'un problème d'ordonnancement repose généralement sur l'atteinte d'un objectif de minimisation ou de maximisation d'un ou plusieurs critères d'évaluation. Cet objectif est défini par une *fonction objectif*. Les diverses formes de critères se présentent comme suit :

- critères liés au temps, tels que :
  - la date de complétion de l'ensemble de tâches ; c'est le critère le plus couramment utilisé ; il est encore appelé *makespan* ;
  - le retard maximal, moyen ou total par rapport aux dates de fin prévues ;
- critères liés aux ressources, tels que :
  - la quantité maximale, moyenne ou pondérée de ressources nécessaires pour réaliser les tâches ;
  - la charge de chaque ressource ;
- critères liés aux coûts de production, de transport, de stockage, etc.

1.1.2 Propriétés des ordonnancements

Un ordonnancement est admissible s'il respecte toutes les contraintes du problème traité. Avec les ordonnancements admissibles, certaines propriétés générales peuvent être identifiées [T'Kindt and Billaut, 2002; Tignon, 2020]. Dans la suite de cette sous-section, les exemples illustratifs concernent des problèmes avec 2 jobs (travaux) J_1 et J_2 de 2 opérations (tâches) chacun à ordonnancer sur 3 machines M_1, M_2 et M_3. J_1 doit être traité d'abord sur M_1 puis sur M_2 tandis que J_2 doit être traité par M_3 puis par M_2.

Ordonnancement semi-actif : un ordonnancement est dit semi-actif si aucune tâche ne peut être exécutée plus tôt sans changer l'ordre d'exécution des tâches sur l'une des machines, ou sans violer une contrainte. La figure 1.1 est un exemple d'ordonnancement non semi-actif car la 2^e opération de J_1 pourrait commencer en 1 plutôt qu'en 4 sans changer l'ordre des tâches sur chaque ressource, ni violer les contraintes de précédence. La figure 1.2 est un exemple d'ordonnancement semi-actif correspondant.

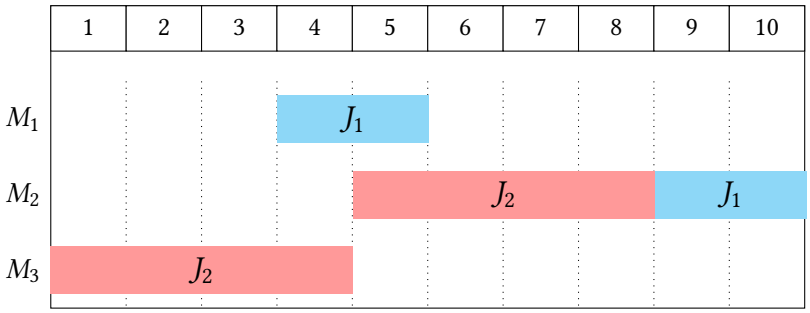


FIGURE 1.1 – Exemple d'ordonnancement quelconque

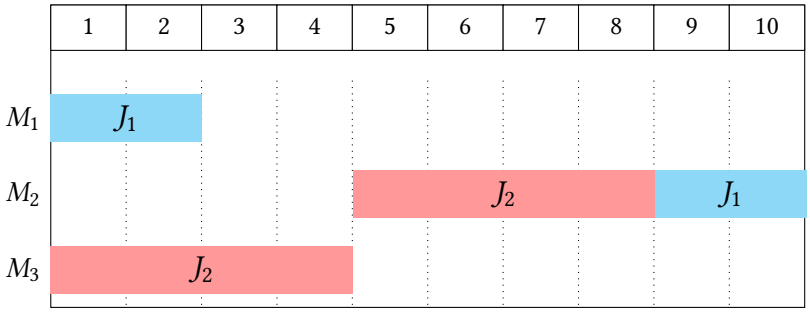


FIGURE 1.2 – Exemple d'ordonnancement semi-actif

Ordonnancement actif : un ordonnancement est dit actif si aucune tâche ne peut être exécutée plus tôt sans retarder l'exécution d'une autre sur une machine et sans violer une contrainte. En d'autres termes, dans un ordonnancement actif, aucune opération ne peut être placée dans un espace de temps libre plus tôt tout en préservant la faisabilité. La figure 1.2 n'est pas un ordonnancement actif car la 2^e opération de J_1 pourrait commencer en 3 plutôt qu'en 9 sans devoir déplacer d'autres tâches ni violer de contraintes. La figure 1.3 est un exemple d'ordonnancement actif pour un problème avec des durées différentes de celles des 2 exemples précédents. Tout ordonnancement actif est également semi-actif. Quelque soit le problème, en ce qui concerne la minimisation de la $C_{\max}$ au moins un des ordonnancements optimaux y afférant est actif [Bierwirth and Mattfeld, 1999].

Ordonnancement sans délai : un ordonnancement est dit sans délai si à aucun moment, une machine n'est laissée inactive pendant qu'il existe des tâches en attente. Les machines n'attendent pas avant d'exécuter leurs tâches. La figure 1.3 n'est pas un ordonnancement sans délai. En effet, à l'instant 3 la 2^e opération de J_1 pourrait démarrer sur la machine M_2. La figure 1.4 est un exemple d'ordonnancement sans délai. Tout ordonnancement sans délai est également actif. Pour un problème, il est possible que l'ensemble des ordonnancements sans délai ne contienne aucun ordonnancement optimal.

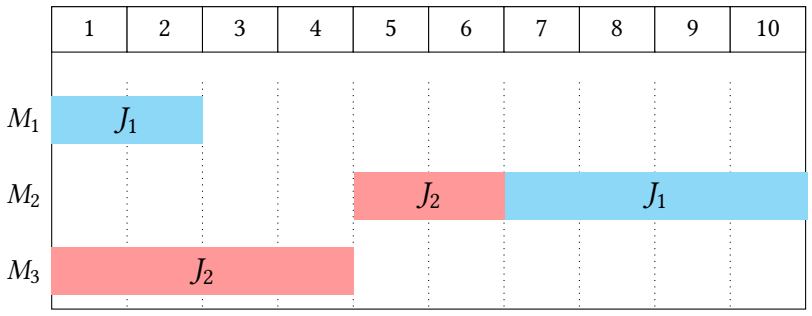


FIGURE 1.3 – Exemple d'ordonnancement actif

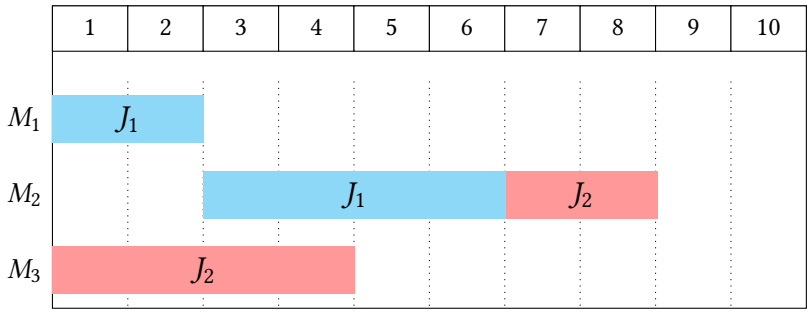


FIGURE 1.4 – Exemple d'ordonnancement sans délai

1.1.3 Classification des problèmes d'atelier

Il existe différents types d'organisation d'atelier, chacun suivant une configuration donnée de ressources. Les problèmes d'ordonnancement liés aux ateliers de production impliquent généralement l'utilisation de ressources renouvelables et l'affectation disjonctive de ces ressources. Les différentes classes de problèmes d'ordonnancement dépendent essentiellement de la manière dont ces ressources sont organisées et gérées au sein de l'atelier et de la diversité des produits (figure 1.5). La notation de [Graham et al., 1979], permet d'identifier rapidement la variante de problème étudiée.

Atelier à ressource unique : elle se rapporte souvent à la notion de ressource critique ; elle regroupe les problèmes dits à une machine. Dans cette catégorie de problème, il y a une seule machine disponible pour effectuer un ensemble de tâches. Chaque tâche a une durée d'exécution spécifique, et l'objectif est de déterminer l'ordre dans lequel les tâches doivent être exécutées sur la machine pour optimiser une certaine mesure de performance (temps total d'exécution, temps total d'attente).

Atelier à ressources parallèles : un atelier peut contenir plusieurs machines en parallèle, ce qui signifie que plusieurs tâches peuvent être traitées simultanément pour un type d'opération donné. Les machines peuvent être identiques ou différentes.

Atelier à ressources multiples : plusieurs ressources exécutent plusieurs tâches dans ce type d'organisation. Le type de cheminement des tâches conduit aux trois sous-organisations d'atelier suivantes :

— Atelier à cheminement unique ou *flow shop* : il existe un seul itinéraire ou séquence fixe pour traiter les tâches. Cela signifie que les produits suivent un chemin prédéfini à travers l'atelier, passant

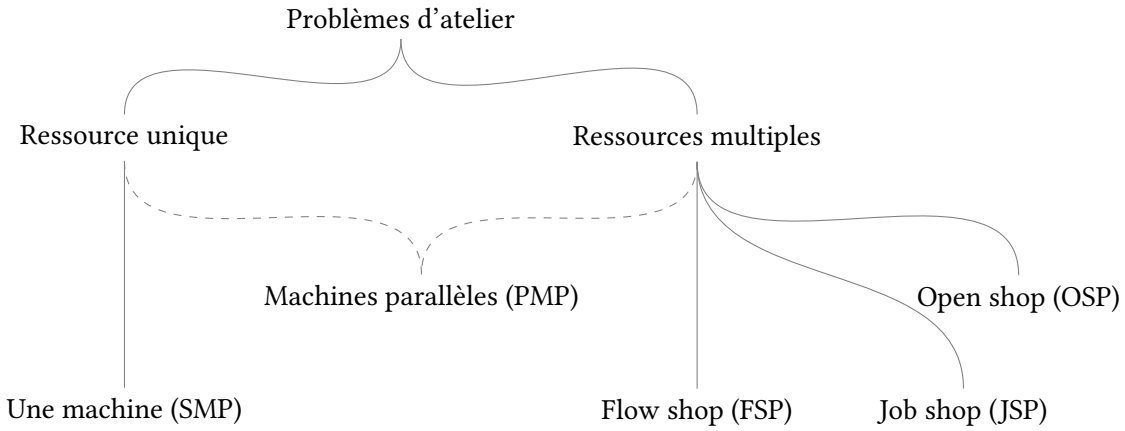


FIGURE 1.5 – Typologie des problèmes d'ordonnancement d'atelier tirée de [Esquirol and Lopez, 1999; Zhang et al., 2012]

par une série d'étapes ou de stations dans un ordre strict. Le processus de confection (ou gamme) des produits est dit linéaire car les opérations d'une gamme sont liées par une relation d'ordre total. Cette configuration est courante dans les environnements de production où les produits sont standardisés et où le processus de fabrication est hautement répétitif.

— Atelier à cheminement multiple ou *job shop* : il correspond à un environnement qui s'occupe de la production de divers types de produits, nécessitant l'utilisation de différentes machines dans des séquences variées. Dans ce cas, la gamme reste linéaire mais est spécifique à chaque produit.

— Atelier à cheminement libre ou quelconque ou *open shop* : il représente un environnement de production dans lequel chaque produit à fabriquer doit passer par une série d'opérations effectuées sur un ensemble de machines, mais l'ordre de réalisation de ces opérations est quelconque. Ici les gammes de fabrication ne sont plus linéaires et peuvent être de type arborescent. On peut citer l'exemple d'un processus de montage d'un produit dans lequel les opérations d'assemblage de deux sous-ensembles de ce produit sont indépendantes l'une de l'autre et peuvent donc être réalisées dans un ordre quelconque.

Il est possible de trouver différents types d'organisations coexistant au sein d'un même atelier de production, chacun intervenant à une étape particulière du processus. De surcroît, l'introduction de contraintes et/ou de paramètres supplémentaires aux configurations précédemment évoquées donne lieu à de nouvelles variantes. Un exemple concret serait l'émergence d'un *flow shop hybride* lorsque, dans un atelier de type flow shop, une étape de fabrication peut être assurée par plusieurs machines en parallèle.

Après avoir exploré la variété des problèmes d'atelier, englobant des configurations allant des ateliers à ressource unique, à ressources parallèles et à ressources multiples, notre attention, dans la présente thèse, se tourne désormais vers une catégorie particulière qui a suscité et continue de susciter l'intérêt des chercheurs en optimisation : le problème d'ordonnancement d'atelier de type job shop. Ce problème représente un défi complexe où différentes tâches doivent être exécutées par des machines dans un ordre spécifique. La flexibilité dans la séquence des tâches crée un défi supplémentaire en termes d'ordonnancement.

1.1.4 Problème d'ordonnancement job shop et extensions

Dénommé *Job shop Scheduling Problem* (JSP), ce problème d'ordonnancement consiste à ordonnancer des travaux (jobs) qui ont des séquences d'opérations préétablies dans un environnement multi-machines. Pour chaque job, les opérations s'exécutent dans un ordre respectant des exigences de gamme opératoire. Le traitement d'une opération ne peut débuter que lorsque l'opération précédente dans la gamme du job est terminée ; on parle de contrainte de précédence. La gamme varie d'un job à un autre créant ainsi un flux multidirectionnel [French, 1982; Choi and Choi, 2002; Esquirol and Lopez, 2008]. Le JSP est un problème reconnu comme étant NP-difficile [Garey et al., 1976]. Il admet un ensemble de n jobs $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ et un ensemble de m machines $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Le job i, aussi noté J_i, est composé de N_i opérations $\{O_{i,1}, O_{i,2}, \ldots, O_{i,N_i}\}$. En d'autres termes, une opération $O_{i,j}$ désigne la j^{ieme} opération de production de la gamme opératoire du job J_i devant s'exécuter sur une machine k. Le temps de traitement de l'opération est noté $p_{i,j,k}$. Le nombre total d'opérations à ordonnancer est obtenu par $\sum_{i=1}^{n} N_i$.

Les configurations des systèmes de production diffèrent d'un atelier à un autre. Il en résulte plusieurs extensions pour le JSP. L'ajout ou la suppression des hypothèses et/ou des contraintes d'une variante permet d'en obtenir une autre. Une des extensions très connue du JSP est le *Flexible Job shop Scheduling Problem* (FJSP). On parle de FJSP lorsque plusieurs machines sont capables de réaliser la même opération. [Pinedo, 2012] le définit comme étant une généralisation des problèmes de job shop et d'environnements de machines parallèles. Afin de rendre le FJSP plus réaliste, des contraintes additionnelles sont parfois rajoutées pour la définition de problèmes particuliers. L'intégration des ressources de transport entre machines, de l'empreinte écologique et énergétique du transport ou encore des temps de configuration des machines est caractéristique des éventuelles ramifications dont le FJSP peut faire l'objet. Dans le cadre de la présente étude, il sera étudié en plus du JSP classique, le FJSP avec ressources de transport (FJSPT). Lorsque les ressources de traitement des tâches sont éloignées les unes des autres, des tâches de transport viennent s'ajouter aux tâches de traitement, qu'il convient d'ordonnancer également. La présence de ressources de transport génère des contraintes associées, par exemple liées à la disponibilité de ces ressources, et certaines intègrent des temps de transport non négligeables devant les durées opératoires. Notons que dans de nombreuses variantes de base, ou quand les temps de transport sont faibles devant les temps de traitement, les tâches et contraintes de transport sont négligées. Si plusieurs ressources de transport partagent une même zone de déplacement, des risques de collision existent et cela induit des contraintes dites spatiales ; elles peuvent aussi induire des contraintes de synchronisation avec d'autres ressources.

De l'anglicisme *Flexible Job shop Scheduling Problem with Transportation*, le FJSPT étend le JSP en incluant la flexibilité des ressources lors du processus d'ordonnancement. Aux ensembles de jobs et de machines, s'ajoutent un ensemble $\mathcal{R}$ de r ressources de transport (AGV, grues, robots) $\mathcal{R} = \{R_1, R_2, \ldots, R_r\}$ et une station de chargement/déchargement (load/unload en abrégé LU) à l'entrée et/ou à la sortie du système, qui peut être assimilée à une machine. Le sous-ensemble $\mathcal{M}_{i,j} \subseteq \mathcal{M}$, dont l'effectif est noté $m_{i,j}$, regroupe l'ensemble des machines capables de réaliser l'opération $O_{i,j}$. En outre, une fois l'opération $O_{i,j-1}$ achevée sur la machine $M_{k'}$, le job J_i est transféré par une ressource de transport R_h vers la machine suivante dans la gamme M_k, pour y effectuer l'opération $O_{i,j}$. Cette tâche de transport est notée $T_{i,j}$. Toute ressource de transport peut effectuer deux types de déplacements : un déplacement à vide de son emplacement courant vers l'emplacement où le job doit être récupéré et, à partir de ce deuxième emplacement, un déplacement en charge pour transférer le job vers son emplacement suivant. Les temps de transport à vide et en charge (respectivement $t^E_{i,j}$ et $t^L_{i,j}$) sont parfois indépendants du job mais ils dépendent le plus souvent des machines. Les temps $t^E_{i,j}$ et $t^L_{i,j}$ ne sont pas des données prédictibles car ils dépendent de l'état

du système. En particulier, $t^L_{i,j}$ dépend des machines $M_{k'} \in \mathcal{M}_{i,j-1}$ et $M_k \in \mathcal{M}_{i,j}$ tandis que $t^E_{i,j}$ dépend de $M_{k'}$ et de la machine sur laquelle se trouve la ressource de transport affectée à $T_{i,j}$.

Compte tenu de la complexité inhérente à ces problèmes d'ordonnancement d'atelier, la quête de solutions optimales implique nécessairement l'application de méthodes de résolution appropriées.

1.1.5 Résolution des problèmes d'ordonnancement d'atelier

En recherche opérationnelle, il existe deux grandes catégories de méthodes de résolution pour les problèmes d'optimisation combinatoire, y compris les problèmes d'ordonnancement d'atelier. Il s'agit des méthodes exactes et des méthodes approchées. La catégorie est communément choisie en fonction de la complexité du problème à résoudre.

1.1.5.1 Méthodes exactes

Les méthodes exactes sont des approches algorithmiques qui garantissent l'obtention de la solution optimale pour un problème d'optimisation combinatoire. Elles sont globalement des méthodes énumératives pouvant être considérées comme des algorithmes de recherche arborescente. La recherche est effectuée sur l'ensemble de l'espace de recherche intéressant, et le problème est résolu en le subdivisant en problèmes plus simples. La programmation linéaire en nombres entiers (avec les algorithmes branch and bound, branch and cut, branch and price), la programmation dynamique et la programmation par contraintes sont des exemples de méthodes exactes couramment utilisées en optimisation combinatoire [Hillier and Lieberman, 2001; Borne et al., 2013]. Les méthodes exactes peuvent être appliquées à de petites instances de problèmes difficiles en raison du temps calculatoire qui est à même d'augmenter exponentiellement sur les instances de grande taille. Pour pallier cette faiblesse, le recours est fait aux méthodes dites approchées.

Dans cette thèse, notre focus se dirige spécifiquement vers des problèmes d'ordonnancement soumis à différents types de contraintes. L'accroissement du nombre de contraintes dans une approche exacte entraîne une complexité croissante, rendant la recherche d'une solution optimale plus difficile dans des délais raisonnables. Les solveurs (logiciels informatiques capables de résoudre des problèmes d'optimisation) basés sur des méthodes exactes, tels que *CPLEX* et *Gurobi*, tendent à s'interrompre après un certain laps de temps d'exécution lorsqu'ils sont confrontés à des instances de grande taille, dépassant souvent les limites de ressources telles que le temps CPU, la mémoire disponible, ou le nombre d'itérations autorisé. Afin d'assurer la robustesse de notre démarche et de la rendre adaptable à des instances de toutes tailles, nous orientons notre choix vers les méthodes approchées, une direction que nous explorerons tout au long de ce manuscrit.

1.1.5.2 Méthodes approchées

Les méthodes approchées permettent de trouver une solution de bonne qualité en un temps de calcul raisonnable sans garantir son optimalité. Ces méthodes sont utilisées lorsque le problème combinatoire est NP-difficile, ce qui signifie qu'il n'existe pas d'algorithme efficace pour trouver la solution optimale en temps polynomial. En d'autres termes, les techniques de résolution approchées sont utilisées pour des problèmes de grande taille ou complexes, où la recherche de la solution optimale est difficile en raison du grand espace de recherche et/ou du type de contraintes.

Au sein de la classe des méthodes approchées se trouvent des algorithmes de type heuristique. En général, ces algorithmes peuvent se montrer sous trois différentes formes :

- *heuristiques spécifiques* qui sont conçues pour résoudre exclusivement un problème et/ou un cas spécifique (sui generis) ;
- *algorithmes d'approximation* qui sont des heuristiques à partir desquelles il est possible de prouver théoriquement la qualité de la solution dans le pire des cas ;
- *métaheuristiques* qui sont des algorithmes génériques applicables à presque n'importe quel problème d'optimisation [Talbi, 2009].

Dans cette thèse, nous nous intéressons en particulier à cette dernière famille de méthodes d'optimisation que représentent les métaheuristiques.

[Sörensen and Glover, 2013] proposent une définition d'une métaheuristique en tant que structure algorithmique de haut niveau, indépendant du problème et qui fournit un ensemble de lignes directrices ou de stratégies pour développer des algorithmes d'optimisation heuristiques. Ils précisent que le terme est également utilisé pour désigner la mise en œuvre d'un algorithme heuristique spécifique à un problème, conformément aux lignes directrices exprimées dans la structure algorithmique en question. Deux grandes classes de métaheuristiques peuvent être identifiées dans la littérature : les métaheuristiques basées sur une solution unique et les métaheuristiques à base de population. Il existe tout de même une approche constructive souvent rattachée à la première classe mais parfois considérée comme une classe à part entière.

Une métaheuristique basée sur une solution unique, comme son nom l'indique, améliore une seule solution à la fois dans un processus itératif au sein de l'espace de recherche. Partant d'une solution initiale, à chaque itération, une solution est sélectionnée en remplacement de la solution courante dans le voisinage de celle-ci. Le processus se termine lorsque le critère d'arrêt prédéfini est réalisé. Le critère d'arrêt implique généralement une limite de nombre d'itérations ou une limite de temps de traitement (temps CPU). Ce type de méthode de résolution est également appelé *métaheuristique à voisinage* ou *métaheuristique de recherche locale* ; le recuit simulé *(simulated annealing)*, la recherche taboue *(tabu search)*, la recherche locale itérée *(iterated local search)*, la recherche par voisinage variable *(variable neighborhood search)* et recherche locale guidée *(guided local search)* sont des exemples courants de métaheuristique à voisinage.

Une métaheuristique à base de population quant à elle améliore de façon itérative une population de solutions. Un ensemble de solutions désigné population est d'abord initialisé. Ensuite, à chaque itération et jusqu'à ce qu'un critère d'arrêt prédéfini ne soit atteint, une nouvelle population est obtenue en remplacement de l'ancienne par une sélection, une modification d'une solution et/ou une combinaison de solutions courantes. Les procédures qui permettent de modifier les populations sont communément appelées des *opérateurs de recherche*. À chaque itération, la meilleure solution de la population est retenue si elle améliore la meilleure existante. L'algorithme génétique *(genetic algorithm)*, l'optimisation par essaims particulaires *(particle swarm optimization)*, les colonies de fourmis/abeilles *(ant colony optimization, bee colony optimization)*, la recherche par dispersion *(scatter search)* et les algorithmes immunitaires artificiels *(artificial immune algorithms)* sont entre autres des métaheuristiques à base de population. Il faut relever que les algorithmes génétiques sont les méthodes les plus répandues de cette catégorie de métaheuristiques.

Les méthodes ci-dessus présentées dans leur diversité sont utilisées suivant différentes configurations tant en fonction du problème traité qu'en fonction des objectifs poursuivis. Ces dernières années, une tendance d'hybridation se matérialisant par la combinaison d'idées issues de méthodes variées a fait l'objet de plusieurs travaux. Par exemple, dans [Zhang et al., 2012], une combinaison d'algorithme génétique et

de recherche taboue a été adoptée pour résoudre un problème d'ordonnancement d'atelier. Une démarche similaire a été entreprise dans [Ibrahim and Tawhid, 2023], où un algorithme d'algues artificielles *(artificial algae algorithm)* a été hybridé avec un algorithme d'évolution différentielle *(differential evolution)*. Par ailleurs, les méthodes approchées sont hybridées non seulement entre elles mais aussi avec les méthodes exactes ; dans ce dernier cas, cette approche est connue sous le nom de *matheuristique* (voir [Boschetti et al., 2009]). En outre, des techniques d'intelligence artificielle avec des algorithmes d'apprentissage sont de plus en plus intégrées aux méthodes classiques de la recherche opérationnelle, comme illustré dans [Stanković et al., 2022].

1.1.6 Composants classiques des métaheuristiques

L'utilisation d'une métaheuristique se caractérise par l'exploration d'un espace de recherche et l'exploitation des bonnes solutions à l'aide de techniques de diversification et d'intensification. Ceci passe par une définition de l'espace des solutions et une stratégie d'exploration incluant la valorisation des solutions.

1.1.6.1 Définition de l'espace des solutions

Définir l'espace des solutions possibles nécessite la clarification des concepts de solution et de codage.

Définition 1.1 (Solution)**.** *En optimisation combinatoire, une solution fait référence à une configuration spécifique qui satisfait aux contraintes d'un problème donné.*

Dans le contexte spécifique d'un problème d'ordonnancement d'atelier, une solution se réfère à un ensemble d'allocations spécifiques de temps et de ressources pour les différentes tâches du problème. Ces allocations respectent les contraintes imposées par le problème et vise généralement à optimiser un ou plusieurs critères, comme la minimisation du temps total d'exécution, la maximisation de l'utilisation des ressources, ou d'autres critères selon le contexte. À l'intérieur des métaheuristiques, chaque solution est exprimée par une "écriture contractée" qui la représente et à partir de laquelle l'ensemble des allocations peut être reconstituée ; on parle de codage de solution ou de représentation de solution.

Définition 1.2 (Codage de solution)**.** *Le codage de solution fait référence à la représentation formelle d'une solution dans le contexte d'un problème d'optimisation. Il définit la manière dont les informations nécessaires à la description d'une solution sont structurées et encodées pour être utilisées par un algorithme d'optimisation.*

Si on se réfère au vocabulaire usuel dans le domaine des algorithmes génétiques, le codage de solution définit le *génotype* et la résultante de la solution (ordonnancement) désigne le *phénotype*. Deux approches de codages sont à distinguer : l'approche directe (codage job, clé aléatoire, etc.) et l'approche indirecte (codage basé sur des règles de priorité, graphe disjonctif, etc.) [Cheng et al., 1996]. Dans l'approche directe, l'enchaînement des opérations d'un ordonnancement est encodé dans le codage tandis que dans l'approche indirecte, le codage contient des informations sur les préférences de décision pour arbitrage à chaque étape de l'ordonnancement. Prenons l'exemple d'une petite instance de JSP avec 2 jobs et 3 machines, avec les gammes opératoires (durées des opérations entre parenthèses) décrites comme suit :

Job 1 : Machine 1 (4) $\longrightarrow$ Machine 2 (2) $\longrightarrow$ Machine 3 (1)
Job 2 : Machine 2 (7) $\longrightarrow$ Machine 3 (3) $\longrightarrow$ Machine 1 (2)

La figure 1.6 illustre la représentation et l'ordonnancement déduit d'une solution pour cette instance avec le codage job de [Bierwirth, 1995], qui est un codage direct. Ce codage représente une solution par une liste ordonnée de jobs, chaque job apparaissant autant de fois qu'il contient d'opérations. L'ordonnancement est construit en suivant impérativement l'ordre des éléments de la liste. Pour chaque job, les opérations sont placées au plus tôt sur les machines correspondantes dans l'ordonnancement, suivant l'ordre normal des opérations de la gamme du job. Dans cet exemple, l'opération 1 du job 1 ($O_{1,1}$) est placée en premier sur M1. Ensuite, l'opération 1 du job 2 ($O_{2,1}$) est placée sur M2, puis l'opération 2 du même job ($O_{2,2}$) sur M3, et ainsi de suite.

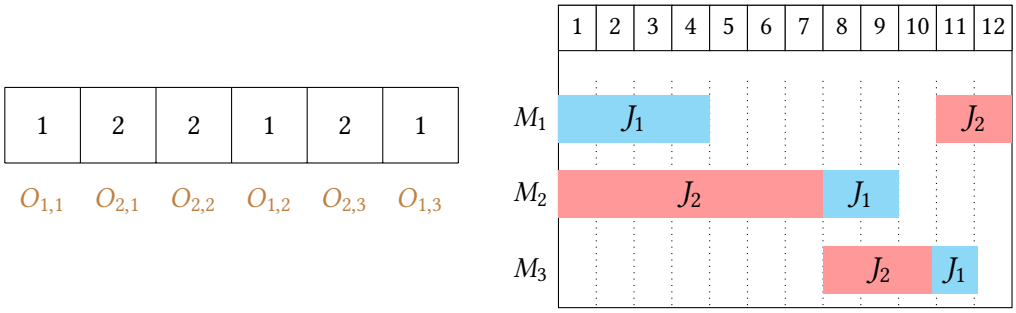


FIGURE 1.6 – Exemple de solution représentée par un codage direct (codage job) et ordonnancement déduit

Le codage basé sur des règles de priorité *(priority rule-based encoding)* de [Storer et al., 1992; Cheng et al., 1996] est un exemple de codage indirect qui représente une solution sous la forme d'une liste de règles de répartition des priorités pour l'affectation des opérations. La taille de la liste correspond au nombre total d'opérations et chaque élément de la liste indique la règle de priorité à utiliser à chaque itération du processus d'ordonnancement. Un ordonnancement est donc construit avec une heuristique basée sur la séquence de règles de répartition. À chaque itération de l'heuristique, une opération est sélectionnée parmi les opérations pouvant être ordonnancées. Un élément p_i de la liste représente une règle de l'ensemble des règles de répartition des priorités prédéfinies. L'élément en i^e position indique qu'un conflit dans la i^e itération de l'heuristique doit être résolu en utilisant la règle de priorité p_i. Plus précisément, une opération, parmi celles en conflit, doit être sélectionnée par la règle p_i ; les ex æquo sont départagés par un choix aléatoire. Cette approche, connue sous le nom de *méthode de Giffler & Thompson* [Giffler and Thompson, 1960], est considérée comme le fondement de toutes les heuristiques qui reposent sur des règles de priorité. Soient :

- PS_t un ordonnancement partiel contenant t opérations ordonnancées,
- S_t l'ensemble des opérations pouvant être ordonnancées à l'itération t,
- ρ_i la date au plus tôt à laquelle l'opération $i \in S_t$ pourrait être démarrée,
- ϕ_i la date au plus tôt à laquelle l'opération $i \in S_t$ pourrait être terminée,
- C_t l'ensemble des opérations conflictuelles à l'itération t.

La procédure d'ordonnancement suit les étapes suivantes :

Étape 1 : Soit $t = 1$, l'ordonnancement partiel PS_t étant vide et S_t comprenant toutes les opérations qui n'ont pas de prédécesseurs.

Étape 2 : Déterminer $\phi_i^* = \min_{i \in S_t}\{\phi_i\}$ et la machine m^* associée. S'il existe plus d'une machine de ce type, l'égalité est départagée par un choix aléatoire.

Étape 3 : Former l'ensemble conflictuel C_t, qui comprend toutes les opérations $i \in S_t$ avec $\rho_i < \phi_i^*$ qui

nécessitent la machine m^*. Sélectionner une opération dans C_t, selon la règle de priorité p_t, et ajouter cette opération à PS_t le plus tôt possible, créant ainsi un nouvel ordonnancement partiel PS_{t+1}. S'il existe plus d'une opération selon la règle de priorité p_t, l'égalité est départagée par un choix aléatoire.

Étape 4 : Mettre à jour PS_{t+1} en supprimant l'opération sélectionnée de S_t, et en ajoutant le successeur direct de l'opération à S_t. Incrémenter t de un.

Étape 5 : Revenir à l'étape 2 jusqu'à ce qu'un ordonnancement complet soit généré.

Tableau 1.1 – Règles de priorité tirées de [Cheng et al., 1996]

Règle	Description
`SPT`	Délai de traitement le plus court *(Shortest Processing Time)*
`LPT`	Délai de traitement le plus long *(Longest Processing Time)*
`MWR`	Job dont le temps de traitement total restant est le plus long *(Most Work Remaining)*
`LWR`	Job dont le temps de traitement total restant est le plus court *(Least Work Remaining)*

La figure 1.7 présente une solution avec le codage à base de règles de priorités et l'ordonnancement correspondant pour la même instance JSP. Les règles de priorités utilisées sont décrites dans le tableau 1.1. À l'étape initiale, nous avons $S_1 = \{O_{1,1}, O_{2,1}\}$, $\phi_1^* = \min\{4, 7\} = 4$, $m^* = 1$, $C_1 = \{O_{1,1}\}$. L'ensemble C_1 étant un singleton, le recours à la première règle (`LPT`) de la liste n'est pas nécessaire ; l'opération $O_{1,1}$ est donc ordonnancée, retirée de S_1 et ajoutée à PS_1. S_1 est alors mis à jour avec l'ajout de l'opération $O_{1,2}$. La mise à jour des données conduit à l'itération suivante avec $S_2 = \{O_{2,1}, O_{1,2}\}$, $\phi_2^* = \min\{7, 6\} = 6$, $m^* = 2$, $C_2 = \{O_{2,1}, O_{1,2}\}$. À cette itération, la règle `SPT` permet de retenir l'opération $O_{1,2}$ car il finit plus tôt que $O_{2,1}$ sur M_2. La troisième itération a pour données : $S_3 = \{O_{2,1}, O_{1,3}\}$, $\phi_3^* = \min\{13, 7\} = 7$, $m^* = 3$, $C_3 = \{O_{1,3}\}$. Une fois de plus, l'application de la règle de priorité n'est pas nécessaire. Le reste de l'ordonnancement est construit de la même façon.

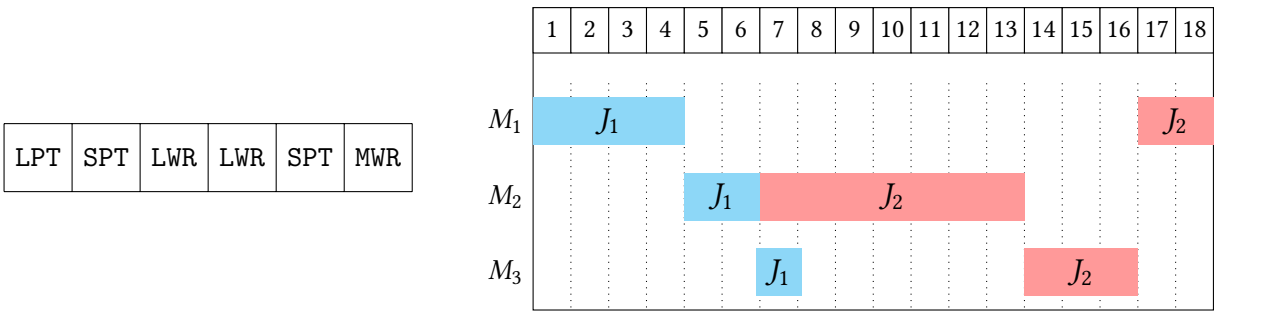


Figure 1.7 – Exemple de solution représentée par un codage indirect (codage à base de règles de priorités) et ordonnancement déduit

1.1.6.2 Stratégie d'exploration des solutions

Les métaheuristiques étant généralement des méthodes itératives, à chaque itération une ou plusieurs solutions sont modifiées pour obtenir de nouvelles solutions. Pour ce faire, des opérateurs de recherche sont appliqués sur les représentations de solutions. En fonction du type de métaheuristique utilisé, les opérateurs de recherche peuvent être des opérateurs de voisinage (échange, insertion, k-opt, etc.) [Schiavinotto and Stützle, 2007], des opérateurs de sélection [Jebari and Madiafi, 2013], des opérateurs de variation (mutation, croisement, etc.) [Kramer, 2017]. Les opérateurs de voisinage regroupent les opérateurs de recherche utilisés au sein de métaheuristique à voisinage.

Définition 1.3 (Voisinage)**.** *Soit S l'ensemble de solutions, pour toute solution $s \in S$, il existe un sous-ensemble $\mathcal{V}_s \subseteq S$ nommé voisinage de s. La fonction de voisinage est la fonction $\mathcal{V} : S \rightarrow 2^S$. Tout élément $s' \in \mathcal{V}_s$ est appelé voisin de s.*

Définition 1.4 (Espace de recherche)**.** *Un espace de recherche désigne l'ensemble des solutions potentiellement aptes à être considérées (visitées) lors d'une exploration. Il est très souvent assimilé à l'espace des solutions.*

[Talbi, 2009], quant à lui, définit l'espace de recherche comme étant un graphe orienté $G = (S, E)$, où l'ensemble des sommets S correspond aux solutions du problème qui sont définies par la représentation utilisée pour résoudre le problème, et l'ensemble des arêtes E correspond aux opérateurs de recherche utilisés pour générer de nouvelles solutions. Les espaces de recherche varient non seulement en fonction du problème mais aussi de la représentation de solution (codage) utilisée. Et la conception de toute métaheuristique itérative nécessite fondamentalement un codage. Selon le codage, les solutions constitutives d'un espace de recherche peuvent être faisables ou non. Le cas échéant, l'espace de recherche peut être restreint à l'espace des solutions faisables. Cependant, autoriser la recherche à explorer des solutions infaisables est parfois souhaitable [Osogami and Imai, 2000; Gendreau and Potvin, 2005; While and Hingston, 2013]. L'espace des solutions faisables regroupe les solutions qui satisfont les contraintes dites strictes du problème. Un exemple de contrainte stricte pour le problème de job shop est la contrainte de précédence induite par les gammes opératoires des jobs. La violation d'une contrainte stricte mène à une solution infaisable. L'ensemble des solutions infaisables d'un espace de recherche constituent l'espace de solutions infaisables [Burke and Kendall, 2014].

Définition 1.5 (Fonction de fitness)**.** *Dans le contexte d'un problème d'optimisation, une fonction de fitness représente la fonction d'évaluation de la qualité d'une solution candidate.*

En d'autres termes, la fonction de fitness accorde une valeur numérique au génotype en fonction de la qualité du phénotype. Cette valeur est généralement appelée *fitness*. La fonction de fitness sert à établir une correspondance entre une solution et un ordonnancement en associant à chaque solution une valeur rendant compte de la qualité de la solution qu'elle représente. De plus, elle permet à l'algorithme d'arbitrer sur le choix des solutions lors de la recherche, au regard de la fonction objectif à optimiser. La fonction de fitness est souvent utilisée à raison pour désigner la fonction objectif. Car une bonne définition de la fonction de fitness doit refléter les objectifs et les contraintes du problème (voir sous-section 1.1.1). Dans le cadre d'un problème d'ordonnancement, la fonction de fitness est associée à ce qu'on peut globalement appelé un moteur d'ordonnancement. Le terme moteur d'ordonnancement n'est pas souvent mentionné lorsqu'il s'agit d'un ordonnancement linéaire. Les moteurs spécifiques construisent les ordonnancements de différentes manières. Autrement dit, à partir d'un même génotype (solution en termes de codage), chaque

moteur peut construire un phénotype (ordonnancement) distinct. La méthode de Giffler & Thompson (les tâches pouvant se terminer le plus tôt, par exemple, sont placées prioritairement) et la méthode FIFO (First In First Out ; les tâches sont placées le plus tôt possible dans l'ordre exact de la permutation d'opérations) sont des exemples de moteurs d'ordonnancement. Il est essentiel de noter que les propriétés des ordonnancements (quelconques, semi-actifs, actifs ou sans délai) sont inhérentes aux moteurs respectifs.

Définition 1.6 (Politique de mouvement)**.** *La politique de mouvement, également appelée stratégie de mouvement est un concept utilisé dans les algorithmes d'optimisation et de recherche locale pour déterminer comment sélectionner et effectuer des modifications sur une solution courante.*

La stratégie de mouvement définit les règles selon lesquelles les solutions sont mises à jour afin de trouver une solution meilleure ou optimale. Elle peut inclure des décisions telles que le choix des voisins à retenir, l'ordre d'application des procédures de mouvement, la fréquence des mouvements, etc. Dans le cas du choix des voisins, une politique de mouvement fournit une description claire des dispositions observées pour la sélection d'un voisin parmi ses pairs lors de l'exploration de l'espace de recherche. Il peut s'agir de choisir, en fonction de la fitness, un voisin strictement meilleur, meilleur ou égal, sensiblement égal, aléatoire, premier améliorant, etc. La politique de mouvement est souvent implicite pour les méthodes dites méthodes de descente (Hill Climbing, Netcrawler, ALNS, etc.) [Barnett, 2001; Al-Betar, 2017].

Les métaheuristiques démarrent invariablement par une *solution initiale* ou une *population initiale*. Dans une dynamique de recherche locale, la solution initiale peut être générée de différentes façons. Néanmoins, il existe deux approches usuelles. La première consiste à générer aléatoirement un génotype de solution. Cette approche permet de démarrer la recherche de tout point de l'espace de recherche de manière équiprobable, ce qui favorise une certaine diversité des solutions aléatoires générées pour chaque exécution. Bien que rapide, avec cette démarche, aucune garantie n'est acquise sur la qualité de la solution initiale. La seconde approche consiste à générer la solution de manière constructive ou gloutonne pour l'obtention d'une solution initiale de qualité relativement bonne. Cette approche requiert un coût calculatoire significativement plus élevé que la première. Elle réduit la diversité des solutions initiales et réduit donc potentiellement les zones atteignables de l'espace de recherche, surtout lorsqu'il s'agit de générer plusieurs solutions pour former une population initiale. Par conséquent, l'utilité de cette démarche réside grandement dans la capacité de l'heuristique à s'orienter vers les bonnes zones de l'espace. Quelquefois, les méthodes à base de population se servent d'une recherche locale pour générer la population initiale. Ce procédé permet de démarrer la recherche avec une population de meilleure qualité qu'avec une génération aléatoire.

1.1.6.3 Relation entre les composants et l'efficacité d'une métaheuristique

Considérant les sous-sections précédentes, les métaheuristiques, qu'elles soient fondées sur l'amélioration itérative d'une solution unique ou d'une population de solutions, mettent en évidence plusieurs composants dont certains peuvent être qualifiés de traditionnels. Parmi ces derniers, nous comptons les codages, les opérateurs de recherche, les fonctions objectifs, les moteurs et les politiques de mouvement (figure 1.8).

Les performances d'une métaheuristique dépendent ainsi du problème et de l'ensemble des composants classiques identifiés précédemment ou d'autres facteurs. Toutefois, plusieurs études soulignent l'importance particulière des codages de solution et des opérateurs de recherche. Par exemple, [Goldberg, 1989] a montré que les codages influencent le comportement et la performance des algorithmes génétiques.

Dans la même dynamique [Radcliffe, 1994] a élaboré un mécanisme permettant d'exploiter les informations sur un espace de recherche dans le but de générer des représentations de solutions et des opérateurs efficaces. Soulignant cette influence significative des codages, [Rothlauf, 2006] a examiné comment différents codages de solution peuvent théoriquement affecter les performances des algorithmes génétiques, fournissant des directives pour cette analyse. De manière plus générale, selon [Talbi, 2009], la définition du codage constitue une étape cruciale dans la conception d'une métaheuristique en raison de son rôle central dans l'efficacité des algorithmes. Il a également fait remarqué que l'efficacité d'un codage est liée aux opérateurs de recherche qui lui sont associés. Ces différentes études mettent en exergue l'influence significative sur les métaheuristiques du choix des codages et de leur association avec des opérateurs, conférant ainsi à ces deux éléments une place prépondérante parmi les autres composants.

Chaque espace de recherche d'une métaheuristique est déterminé par un codage de solution définissant l'ensemble des solutions et par un opérateur de recherche qui établit le lien entre toute solution et ses voisins. Ainsi, pour caractériser la combinaison d'un codage et d'un opérateur, il suffit de caractériser l'espace de recherche qu'ils induisent. Pour une métaheuristique, l'espace de recherche est assimilable à un *paysage de fitness* [Marmion, 2013]. L'analyse du paysage de fitness, en tant qu'outil permettant de comprendre la dynamique d'un espace de recherche, se révèle également comme un moyen efficace pour caractériser les codages de solutions et les opérateurs de recherche.

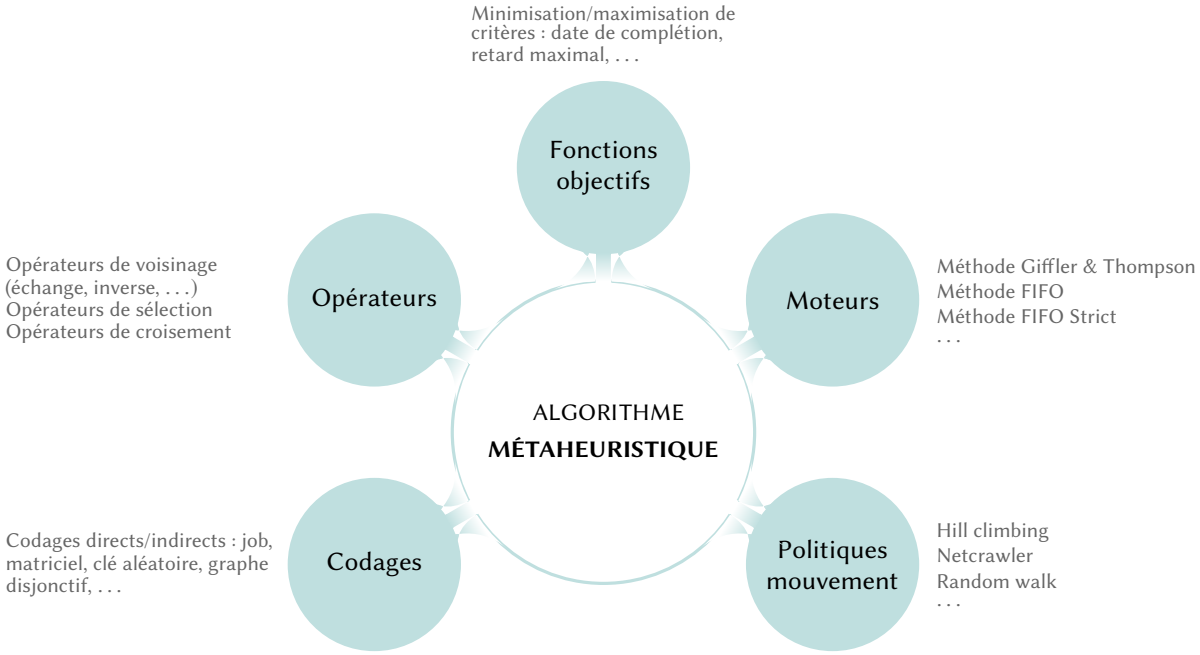


Figure 1.8 – Composants classiques des métaheuristiques

1.2 Analyse de paysage de fitness

L'analyse de paysage de fitness tire ses sources des travaux de [Wright, 1932] sur l'évolution biologique. Le concept s'est élargi au fil des années à de nombreux domaines scientifiques tels que la physique, la biophysique, la chimie, l'optimisation stochastique, l'évolution artificielle, etc [Verel, 2016]. L'analyse de paysage de fitness est un outil analytique utilisé en optimisation combinatoire pour représenter la structure d'un espace de recherche. Outre le fait que les paysages de fitness permettent d'appréhender la complexité des problèmes, [Malan, 2021; Zou et al., 2022] ont évoqué d'autres applications dignes d'intérêt pour les algorithmes de résolution des problèmes d'optimisation. En l'occurrence, il s'agit de l'explication du comportement des algorithmes, de la prédiction de leurs performances, du guidage de leur conception et du pilotage de leurs paramétrages.

Définition 1.7 (Paysage de fitness)**.** *Un paysage de fitness, en optimisation combinatoire, est défini par un triplet* $(S, \mathcal{V}, f)$*. S représente l'ensemble des solutions potentielles au problème; encore appelé espace de solutions. $\mathcal{V} : S \rightarrow 2^S$ est une relation de voisinage entre les solutions qui associe à chaque solution un ensemble de solutions dites voisines. $f : S \rightarrow \mathbb{R}$ est la fonction d'évaluation à optimiser.*

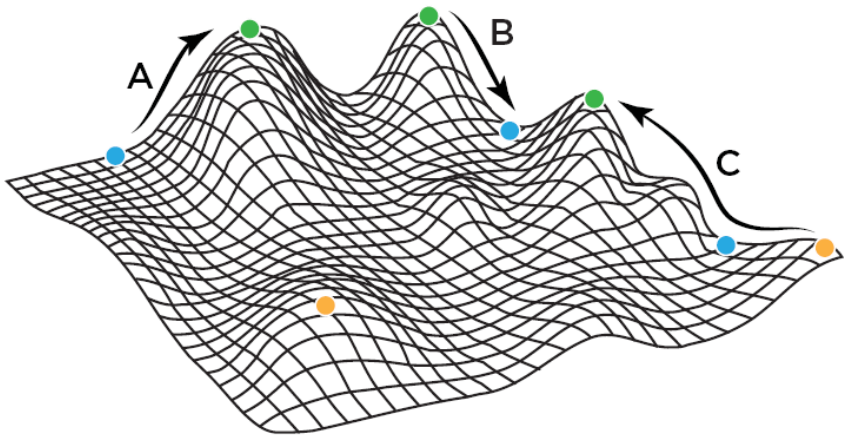


FIGURE 1.9 – Représentation de paysage de fitness [Olson, 2013]

L'illustration géographique de [Olson, 2013] présentée à la figure 1.9 propose une description pratique du paysage de fitness, réalisée au moyen d'un maillage carré. Sur la grille, chaque maille représente une solution et les mailles adjacentes représentent les solutions voisines. La fitness d'une solution est représentée par sa hauteur dans le paysage. Les vallées indiquent de faibles fitness à l'opposé de celles des sommets qui sont élevées. En considérant un objectif de maximisation de la fitness, l'algorithme d'optimisation a tendance à se déplacer vers le sommet le plus proche (A) autour duquel toutes les variations mènent vers le bas. Ce sommet est très probablement un optimum local, qui n'est pas nécessairement le sommet le plus élevé du paysage (optimum global). Cela s'explique par le fait que l'algorithme pousse la fitness vers des sommets proches (bassin d'attraction), mais il n'a pas la prévoyance nécessaire pour avancer vers l'optimum global avec de probables régressions intermédiaires. Pour pallier cette insuffisance, l'algorithme peut tolérer de légères régressions afin d'améliorer la fitness sur le long terme (C). Par ailleurs, il est possible qu'une solution voisine fasse un grand bond positif ou négatif en termes de fitness (B) par rapport à la solution courante.

Un paysage de fitness encore appelé *paysage adaptatif* présente des caractéristiques variées à l'image de celles d'un relief. En gardant la métaphore géographique, un paysage peut être composé entre autres de vallées, de plaines, de plateaux, de bassins, de pics, de zones non praticables comme l'illustre la figure 1.10.

Pour décrire ces caractéristiques, des *propriétés* de paysage avec des métriques associées sont utilisées. Plus d'une trentaine de techniques d'analyse de paysage ont été passées en revue et explicitées dans [Malan and Engelbrecht, 2013] et [Malan, 2021]. Ces techniques concernent pour certaines les espaces de recherche des problèmes discrets et pour d'autres les espaces de recherche des problèmes continus. Il convient également de relever que différentes formes de paysage, telles que les paysages mono-objectif, les paysages multi-objectifs, les paysages dynamiques et autres, sont visées par les métriques répertoriées. Les propriétés de paysage ciblées par les techniques d'analyse en question sont susceptibles d'influencer la performance des algorithmes stochastiques. Ces propriétés sont multiples ; d'aucunes se rapportent aux fitness des solutions prises individuellement et d'autres s'intéressent aux voisinages des solutions au sein du paysage. Bien que distinctes de part leurs définitions, nombre de ces propriétés sont liées les unes aux autres. La modalité, la distribution de fitness, la rugosité, la neutralité, la symétrie, la structure globale et l'évolutivité sont des exemples de propriétés du paysage de fitness. Avant de revenir sur le détail des propriétés de paysage et les techniques de mesure associées, il est essentiel de définir quelques notions préliminaires.

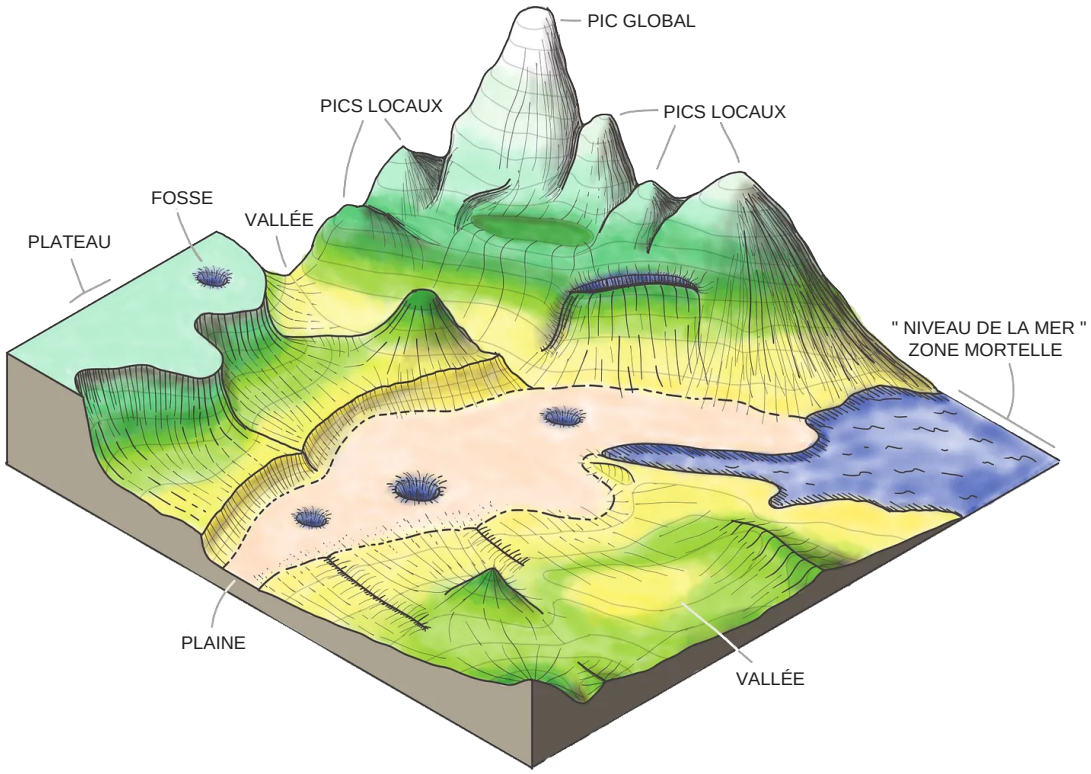


Figure 1.10 – Caractéristiques d'un paysage de fitness [Olson, 2020]

1.2.1 Notions préliminaires et outils de mesure

Les évaluations des paysages de fitness sont principalement réalisées soit par un échantillonnage des solutions de l'espace de recherche, soit par un mouvement itératif au sein de l'espace à l'aide de la fonction de voisinage. Le choix entre ces approches dépend de la technique de calcul utilisée. La première approche peut impliquer un *tirage aléatoire uniforme* parmi l'ensemble des solutions, mais cela n'est pas toujours réalisable pour tous les codages. La seconde approche peut recourir à une *marche aléatoire* parmi les solutions de l'espace de recherche, en intégrant les concepts de pas et de distance. Dans cette sous-section, nous définissons les notions associées aux outils de mesure de l'analyse de paysage de fitness.

Définition 1.8 (Distance)**.** *Dans le cadre d'un espace de recherche, la distance désigne une mesure de la proximité ou de la séparation entre différentes solutions ou points présents dans cet espace.*

La distance peut être définie de diverses manières en fonction du problème spécifique et de la nature de l'espace de recherche. La distance de Hamming et la distance euclidienne sont des exemples courants de mesure de distance. La distance de Hamming, souvent utilisée en optimisation avec les paysages binaires (paysages induits par un codage binaire), sert à comparer des chaînes de bits de longueurs égales en comptant le nombre d'endroits où les bits diffèrent ; tandis que la distance euclidienne est une mesure géométrique classique correspondant à la longueur du segment de droite qui relie deux points dans un espace euclidien.

Définition 1.9 (Pas)**.** *Dans un espace de recherche discret, un pas représente un mouvement individuel effectué dans l'espace. Lors de l'exploration de l'espace, à chaque itération, un pas est franchi pour passer d'une solution à une autre.*

Définition 1.10 (Marche aléatoire)**.** *Une marche aléatoire est un processus aléatoire qui consiste à effectuer une série de pas aléatoires consécutifs. Soit S_N la somme de chaque pas aléatoire consécutif X_i, S_N forme une marche aléatoire où X_i est un pas aléatoire tiré d'une distribution aléatoire.*

À cette définition de [Yang, 2010], il ajoute que cette relation peut également être exprimée sous forme d'une formule récursive, ce qui signifie que l'état suivant S_N dépendra uniquement de l'état existant actuel S_{N-1} et du mouvement ou de la transition X_N de l'état existant à l'état suivant.

$$S_N = \sum_{i=1}^{N-1} + X_N = S_{N-1} + X_N$$

Dans le contexte d'optimisation, une marche aléatoire est un mécanisme d'exploration à travers lequel les déplacements d'une solution à une autre sont choisis de manière aléatoire. Cela signifie que le choix d'un voisin lors de l'exploration se fait au hasard plutôt que selon un schéma défini.

Définition 1.11 (Isotropie)**.** *L'isotropie se réfère à l'homogénéité des propriétés ou des caractéristiques dans toutes les directions à l'intérieur d'un espace de recherche. Un espace de recherche est considéré comme isotropique (ou isotrope) s'il ne présente pas de variation significative de ses propriétés selon l'orientation de l'exploration.*

Un paysage de fitness isotropique implique que les propriétés du paysage, telles que la rugosité ou la neutralité, sont similaires indépendamment de la direction dans laquelle l'on regarde dans l'espace de recherche [Malan and Engelbrecht, 2013].

Définition 1.12 (Tirage aléatoire uniforme)**.** *Un tirage aléatoire uniforme désigne la sélection aléatoire d'une valeur à partir d'une distribution uniforme, où chaque valeur possible a une probabilité égale d'être choisie.*

Dans le contexte d'optimisation, le tirage aléatoire uniforme est souvent utilisé pour générer des nombres aléatoires qui peuvent être appliqués à des opérations telles que des mutations, des sélections, ou d'autres processus aléatoires dans des algorithmes d'optimisation. L'utilisation d'un tirage aléatoire uniforme garantit que chaque option a une chance égale d'être choisie, contribuant ainsi à l'exploration équitable de l'espace de recherche. Il peut également être utilisé pour un échantillonnage au sein d'un ensemble de solutions. Le cas échéant, il faut que le codage de solution utilisé y soit favorable.

1.2.2 Propriétés et métriques

[Malan and Engelbrecht, 2013], dans leur revue, ont présenté 22 techniques pour la caractérisation des paysages de fitness des problèmes d'optimisation discrets. Les auteurs présentent plusieurs techniques pour mesurer diverses propriétés de paysages. Ces techniques dans leur variété sont par endroit non applicables ou difficilement applicables aux espaces de recherche des problèmes d'ordonnancement de type job shop que nous étudions dans ce document. Nous présentons, dans le tableau 1.2, l'ensemble des techniques proposées par [Malan and Engelbrecht, 2013] regroupées par propriété et nous expliquons en quoi les techniques dites non applicables ne sont pas adaptées à notre contexte d'étude.

Tableau 1.2 – Techniques d'analyse de paysage de fitness de [Malan and Engelbrecht, 2013]

Propriété	Technique	Applicable à notre étude
Rugosité	Autocorrélation *(Autocorrelation function)*	Oui
	Distance de corrélation *(Correlation length)*	Oui
	1re et 2e mesures entropiques *(First and second entropic measures)*	Oui
Neutralité	Marche neutre *(Neutral walk)*	Oui
	Mesures relatives aux réseaux de neutralité *(Measures on neutral networks)*	Non *Cette métrique a été proposée et testée, à la base, avec les problèmes de parité booléens* (voir [Vanneschi et al., 2006]). *Les mesures sont calculées à partir d'un ensemble de réseaux de neutralité qu'il faut déterminer au préalable. L'identification des réseaux de neutralité est difficile, surtout sur les instances de moyenne et grande taille.*

Suite à la page suivante ...

Tableau 1.2 – (Suite)

Propriété	Technique	Applicable à notre étude
Evolutivité	Portraits de l'évolutivité de fitness *(Fitness evolvability portraits)*	Oui
	Nuage de fitness *(Fitness cloud)*	Oui
	Coefficient de pente négative *(Negative slope coefficient)*	Oui
	Nuage de probabilité de fitness *(Fitness-probability cloud)*	Oui
	Probabilité d'échappement cumulée *(Accumulated escape probability)*	Oui
Distribution de fitness	Densité des états *(Density of states)*	Oui
	Variance de forme *(Formae variance)*	Non *La variance de fitness est calculée pour les différentes formes (schémas généralisés) échantillonnées à partir de l'ensemble de solutions $\mathcal{S}$; une forme étant un sous-ensemble de $\mathcal{S}$ dont les éléments présentent au moins une caractéristique commune* (pour plus de détails, voir [Radcliffe, 1994]). *La définition des formes représente la principale difficulté pour l'application de cette métrique.*
	Distance de Hamming à/entre un niveau *(Hamming Distance in/between a Level)*	Non *Cette mesure est basée sur la distance de Hamming, qui est spécifique aux paysages générés par le codage binaire.*
Leurre	Leurre d'un algorithme génétique *(GA-deception)*	Non *Technique propre aux algorithmes génétiques avec une connaissance préalable des optima globaux.*

Suite à la page suivante ...

Tableau 1.2 – (Suite)

Propriété	Technique	Applicable à notre étude
	Distance de corrélation de fitness *(Fitness distance correlation)*	Non *Cette métrique suppose l'existence d'une mesure de la distance entre les solutions : distance de Hamming. Une connaissance préalable des optima globaux est également requis. De plus, le calcul du leurre n'a de sens qu'en référence à un algorithme de recherche particulier.*
	Métrique statique-∅ *(Static-∅ metric)*	Non *Cette mesure nécessite une connaissance des optima globaux et se limite aux codages binaires.*
	Difficulté du paysage *(Landscape hardness measure)*	Non *La connaissance préalable des optima globaux est requise.*
Épistasie	Variance d'épistasie *(Epistasis variance)*	Non *Métrique adaptée à un codage binaire, l'épistasie étant calculée sur la base d'une composition linéaire d'une solution à partir de ses bits.*
	Épistasie en bits *(Bit-wise epistasis)*	Non *Métrique adaptée à un codage binaire.*
Structure globale	Métrique de dispersion *(Dispersion metric)*	Non *Cette mesure implique le calcul des distances dans l'espace des solutions. Le calcul de distance est difficile lorsqu'il s'agit d'un paysage induit par un codage non-binaire.*

En complément des propriétés mesurables avec les techniques adaptées à notre contexte, telles que proposées dans [Malan and Engelbrecht, 2013] comprenant la rugosité, la neutralité, l'évolutivité et la distribution de fitness, nous intégrons la propriété *dimension*, extraite de [Tari, 2019]. Ces propriétés et les métriques qui leurs sont associées sont présentées en détail dans la suite de cette section.

1.2.2.1 Rugosité

La rugosité est une caractéristique générale des paysages, elle n'a pas de définition formelle unique. Elle s'intéresse au nombre et à la distribution des optima locaux ainsi qu'à la taille de leur bassin d'attraction. Elle peut également être décrite comme la fréquence des changements de pente entre "montée" et "descente" [Pitzer and Affenzeller, 2012]. Le bassin d'attraction d'un optimum local correspond à l'ensemble des

solutions qui peuvent mener à cet optimum en empruntant uniquement des arcs améliorants dans le graphe de voisinage. Un grand bassin d'attraction pour un optimum local signifie qu'il existe de nombreuses solutions à partir desquelles cet optimum peut être atteint lors d'un processus de descente. La probabilité d'atteindre un optimum donné est donc corrélée avec la taille de son bassin d'attraction [Tari, 2019]. Un paysage est dit rugueux si les solutions voisines ont des valeurs de fitness très différentes. À l'opposé, un paysage doux est celui dans lequel les voisins ont presque la même valeur de fitness [Kauffman, 1993]. Un paysage doux est aussi caractérisé par un faible nombre d'optima locaux et la présence de grands bassins d'attraction [Vassilev et al., 2003]. Ainsi, les algorithmes ont généralement du mal à optimiser les paysages très rugueux en raison de la quantité importante d'optima locaux parmi lesquels ils peuvent se retrouver piégés. La figure 1.11 est une illustration de la rugosité dans un paysage de fitness.

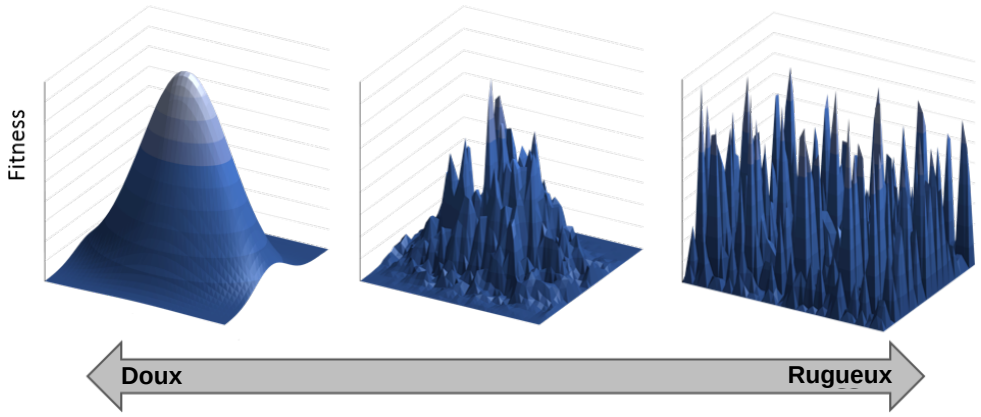


FIGURE 1.11 – Illustration d'un paysage doux (à gauche), d'un paysage rugueux (au milieu) et d'un paysage extrêmement rugueux (à droite) [Shafee, 2014]

Il existe plusieurs techniques pour mesurer la rugosité d'un paysage (marches adaptatives, mesure d'entropie, etc). Une marche adaptative, par exemple, est une marche qui part d'une solution initiale et se déplace à travers un système de voisinage, par un algorithme ou une politique de mouvement, vers de meilleures solutions [Kauffman and Levin, 1987]. L'une des techniques les plus utilisées pour la rugosité est la mesure d'*autocorrélation* de [Weinberger, 1990], originellement testée sur les paysages des modèles NK de Kauffman. En supposant que les paysages sont discrets et *isotropiques*, l'autocorrélation est calculée sur la base d'un ensemble de marches aléatoires. Cette mesure correspond à la corrélation entre les fitness des solutions rencontrées au cours de la marche en considérant la distance qui les sépare. En d'autres termes, à partir d'une solution choisie au hasard dans l'espace de recherche, à chaque pas, une solution voisine est choisie au hasard. Une série temporelle de valeurs de fitness est ainsi générée. La fonction d'autocorrélation est alors utilisée pour déterminer la structure de corrélation de cette série temporelle. Elle donne une estimation de l'influence de la distance sur la variation des fitness entre des paires de solutions rencontrées au cours de la marche. Soient :

$\mathbb{E}[X]$: l'espérance mathématique de X,
$Var(X)$: la variance de X,
f_t : la fitness de la t^e solution d'une marche aléatoire,
f_{t+n} : la fitness de la solution après n pas comptés à partir de la t^{ieme} solution.

La fonction d'autocorrélation $\rho(n)$ concerne la fitness de deux solutions qui sont distantes de n pas.

L'autocorrélation pour n pas des fitness $f_t, t = 1, \ldots, \ell$, est définie comme suit [Hordijk, 1996] :

$$\rho(n) = \frac{\mathbb{E}[f_t \times f_{t+n}] - \mathbb{E}[f_t] \times \mathbb{E}[f_{t+n}]}{Var(f_t)}$$

où $\rho(n) \in [-1, 1]$. Avec $\overline{f} = \frac{1}{\ell} \sum_{t=1}^{\ell} f_t \,;\, \ell > 0$, la fonction d'autocorrélation $\rho(n)$ d'un paysage de fitness peut être estimée par $r(n)$ en échantillonnant les données le long d'une marche aléatoire.

$$r(n) = \frac{\sum_{t=1}^{\ell-n}(f_t - \overline{f})(f_{t+n} - \overline{f})}{\sum_{t=1}^{\ell}(f_t - \overline{f})^2}$$

$|\rho(n)| = 1$ indique une corrélation maximale et une valeur proche de 0 indique une quasi absence de corrélation. À partir de la fonction d'autocorrélation, une *distance de corrélation* τ est déduite. La distance de corrélation représente la distance au-delà de laquelle la majorité des points ne sont plus corrélés. Une petite valeur de τ dénote un paysage plus rugueux.

$$\tau = \frac{-1}{\ln(|\rho(1)|)}$$

1.2.2.2 Neutralité

Dans un paysage de fitness, on parle de neutralité lorsque deux solutions voisines possèdent la même valeur de fitness. La neutralité est donc la proportion de solutions connectées de qualité égale que contient un paysage. Un paysage est considéré comme neutre si une quantité importante de paires de solutions voisines est neutre [Reidys and Stadler, 2001]. Un paysage neutre ne désigne pas un paysage plat mais plutôt il indique la présence d'une neutralité successive, qui peut se manifester par des caractéristiques telles que des plateaux et des crêtes dans le paysage [Malan and Engelbrecht, 2013]. Au cours d'une recherche, traverser une zone neutre du paysage peut être interprété à tort par un algorithme comme une convergence vers un optimum local. Étant donné que les fitness ne changent pas, l'algorithme peut avoir l'impression de stagner lors de son passage dans une zone neutre.

Le *taux de neutralité* d'un paysage peut être obtenu à l'aide du dégré de neutralité moyen issu d'une population de solutions qui échantillonnent l'espace de recherche [Marmion, 2013]. Un voisin neutre d'une solution s est une solution telle que $s' \in \mathcal{V}(s)$ et $f(s) = f(s')$. Le dégré de neutralité $\mathcal{N}_s$ (en %) de la solution s indique sa proportion de voisins neutres.

$$\mathcal{N}_s = \frac{Card(s' \in \mathcal{V}(s), f(s') = f(s)) \times 100}{Card(\mathcal{V}(s))}$$

$\mathcal{N}_s = 0\%$ induit $\forall s' \in \mathcal{V}(s), f(s) \neq f(s')$ tandis que pour $\mathcal{N}_s = 100\%, \forall s' \in V(s), f(s) = f(s')$. Subséquemment, avec n l'effectif d'un échantillon de solutions, le taux neutralité $\mathcal{N}$ du paysage est obtenu par :

$$\mathcal{N} = \frac{1}{n} \sum_{i=1}^{n} \frac{Card(s'_i \in \mathcal{V}(s_i), f(s'_i) = f(s_i)) \times 100}{Card(\mathcal{V}(s_i))}$$

Le taux de neutralité peut être également mesuré avec des marches neutres [Malan and Engelbrecht, 2013]. Une marche neutre est une variante de marche aléatoire opérée à l'intérieur d'une zone neutre en tâchant de maximiser la distance par rapport au point de départ. À partir d'un point de départ aléatoire s_0 dans l'espace de recherche, une marche neutre consiste à générer des voisins neutres successifs avec l'idée de trouver, à chaque itération, un voisin neutre pour lequel la distance totale par rapport à s_0 augmentera avec le pas. Ce processus s'arrête lorsqu'il n'y a plus de voisins neutres pour lesquels la distance totale augmente.

1.2.2.3 Évolutivité

Le terme évolutivité est à la base utilisé en biologie évolutive pour définir la propension au changement des séquences génétiques au fil du temps. Cette propriété dépend, dans un premier temps, de la vitesse de modification de populations dans le paysage de fitness et dans un second temps de la structure du paysage qui lui-même peut limiter l'accessibilité aux optima locaux. Une évolutivité nulle pour un caractère particulier, par exemple, indique que l'évolution de caractère est impossible [Shafee, 2014]. En optimisation, le terme est souvent utilisé en rapport avec les algorithmes évolutionnaires, par analogie avec la biologie. Il y désigne la capacité d'un algorithme de recherche à converger vers une meilleure zone du point de vue de la valeur de fitness dans un paysage. Comme le relève [Marmion, 2013; Malan and Engelbrecht, 2013], cette définition élargit le champ d'application de l'évolutivité au-delà des algorithmes évolutionnaires pour englober n'importe quelle méthode de recherche. Une métaheuristique peut tirer parti d'un paysage à forte évolutivité pour obtenir plus rapidement les meilleures solutions.

Il est également manifeste que la structure d'un paysage adaptatif en termes de rugosité peut influencer la capacité d'une recherche à identifier l'optimum global, comme le montre la figure 1.12. Pour (A), les trajectoires blanche et noire aboutissent toutes les deux à l'optimum global car le paysage est unimodal (ne contient qu'un seul optimum). Par contre, dans un paysage rugueux comme (B), certaines trajectoires se terminent à un optimum local.

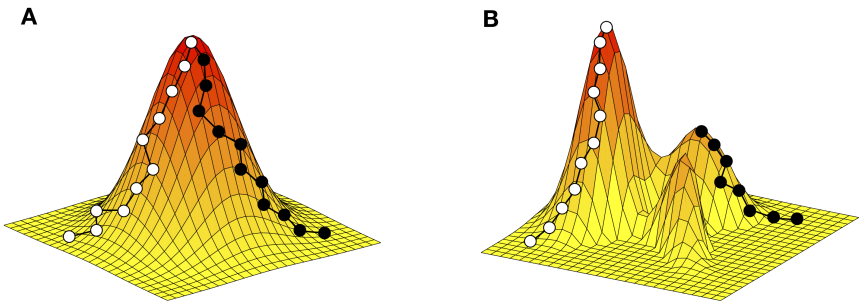


FIGURE 1.12 – Influence de la rugosité sur l'évolutivité dans un paysage de fitness [Payne and Wagner, 2019]

Parmi les quelques mesures qui traitent de l'évolutivité, la probabilité d'échappement cumulée (en anglais *accumulated escape probability*, technique 22 dans [Malan and Engelbrecht, 2013]) semble être la plus pratique pour les expérimentations à suivre dans le présent document. Pour chaque solution tirée d'un échantillon, des voisins sont générés. Ensuite, la proportion de voisins améliorants est calculée dans le but de former un ensemble de probabilités de fitness associant à chaque solution une probabilité d'amélioration. La probabilité d'échappement cumulée est alors définie comme la moyenne de toutes les probabilités de cet ensemble.

1.2.2.4 Distribution de fitness

La distribution de fitness fait partie des propriétés qui peuvent caractériser un espace de recherche à la fois sur l'aspect global et local. Elle a donc différentes définitions en relation avec l'objectif poursuivi. Pour mesurer par exemple la fréquence d'apparition de chaque valeur de fitness, la distribution de fitness

consiste en une analyse statistique qui quantifie le nombre de solutions pour les différentes valeurs de fitness à l'image de la *densité des états* de [Rosé et al., 1996] telle que décrite par [Malan and Engelbrecht, 2013]. Une autre approche consiste à mesurer la répartition des fitness en tenant compte de la position des valeurs de fitness dans le paysage à travers des techniques basées sur la distance de Hamming.

[Hoos and Stützle, 2005] propose une mesure de la distribution de fitness avec un regard sur le voisinage immédiat des solutions. Dénommée *distribution des types de point*, cette technique permet de classer une solution (point) en fonction des différences de fitness présentes dans son voisinage. Le type de point désigne la position d'une solution en termes de fitness par rapport à ses voisins. Sept types de point ont été identifiés : `SLMIN`, `SLMAX`, `LMIN`, `LMAX`, `SLOPE`, `LEDGE` et `IPLAT`. Un point p appartient au type `IPLAT` si tous ses voisins ont la même valeur de fitness que lui. Le type de point est `SLMAX` (`SLMIN`) s'il est strictement un maximum (minimum) local, tandis que les types `LMAX` et `LMIN` désignent respectivement des maxima et minima locaux non-stricts. p est un `SLOPE` si certains de ses voisins ont une valeur de fitness plus grande et d'autres une valeur plus petite que p. Un point `LEDGE` a des voisins ayant la même valeur de fitness que lui et d'autres voisins avec des valeurs de fitness plus grandes et plus petites que lui. Les types de point sont illustrés par la figure 1.13.

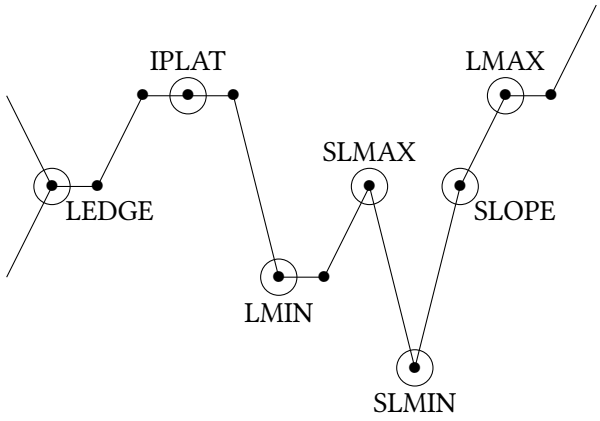


Figure 1.13 – Types de points d'un paysage de fitness [Hoos and Stützle, 2005]

La distribution des types de point est une mesure plus ou moins transversale qui vient en complément aux autres propriétés qui caractérisent un paysage de fitness. Plus le pourcentage de `IPLAT`, `LMIN`, `LMAX` et `LEDGE` est élevé dans un paysage, plus la neutralité de ce paysage est forte et inversement [Zhao et al., 2019]. Calculer cette métrique revient à déterminer le pourcentage de chaque type de point en cumulant la classification d'un ensemble de solutions recueillies par échantillonnage de l'espace de recherche.

1.2.2.5 Dimension

[Tari, 2019] définit la dimension d'un paysage en tant qu'une mesure déterminée par la taille de l'espace de recherche et le taux de connexion entre les solutions. L'espace de recherche est ici assimilé à l'ensemble des solutions possibles, qu'elles soient faisables ou non. La taille de cet espace est induit par le codage de solutions sur la base d'une formulation mathématique du problème. Étant donné que la navigation au sein de l'espace de recherche est aussi fonction du nombre de solutions, l'atteinte de l'optimal global en particulier peut logiquement être influencée par la taille de l'espace. En ce qui concerne la connectivité des solutions entre elles, la relation de voisinage caractérise l'espace de recherche en définissant le nombre de

voisins associés à chaque solution. Ce nombre est assez souvent constant pour l'ensemble des solutions. Seulement, avec la présence de solutions infaisables, il n'est pas toujours possible de déterminer à l'avance le nombre de voisins faisables et infaisables pour l'ensemble des solutions. Car ces deux effectifs varient d'une solution à une autre. Les techniques combinatoires de dénombrement et la théorie des graphes font partie des moyens qui permettent de déterminer la dimension d'un paysage de fitness.

1.3 Bilan

L'ordonnancement d'atelier est un problème d'optimisation qui s'inscrit dans la famille des problèmes combinatoires. Ce type de problème donne généralement lieu à un grand nombre de solutions possibles, et l'idéal est de trouver la meilleure en fonction d'un ou plusieurs objectifs bien définis. Pour ce faire, bien que les méthodes dites exactes soient très efficaces pour trouver les meilleures solutions, elles sont limitées pour les grandes instances en raison du temps de calcul astronomique qu'elles requièrent. Cette faiblesse des méthodes exactes rend d'autant plus pertinente l'utilisation des méthodes approchées, telles que les métaheuristiques. Cette deuxième catégorie de méthodes ne garantit pas l'optimalité, mais elle permet de trouver des solutions relativement bonnes. Le recours à une métaheuristique implique l'exploration d'un espace de recherche analogue à un paysage de fitness. Les métaheuristiques sont faites de divers éléments pouvant influencer leurs performances, parmi lesquels les codages de solution et les opérateurs de recherche. Ces deux composants sont communs à l'ossature des métaheuristiques et à la définition des paysages de fitness. Ainsi, la compréhension de la structure d'un paysage pourrait potentiellement permettre de prédire la performance de la métaheuristique employée sur ledit paysage. À travers différentes mesures, l'analyse de paysage de fitness permet de caractériser les composants de métaheuristiques par le truchement des paysages induits. Cette caractérisation serait fondamentale pour affiner la sélection des composants pertinents pour une conception *«optimisée»* des métaheuristiques.

L'objectif de cette thèse est de caractériser les codages de solution et les opérateurs de voisinage dans le cadre de la résolution approchée de problèmes d'ordonnancement soumis à des contraintes. L'étude a pour but d'apporter un éclairage sur les facteurs qui contribuent à l'efficacité des métaheuristiques, avec un focus sur les deux composants mentionnés. Bien que les problèmes liés à l'ordonnancement de tâches dans des cas contraints soient divers, nous avons restreint notre étude aux problèmes d'ordonnancement d'atelier en général, et plus particulièrement aux problèmes de type job shop. L'idée est de développer une approche conceptuelle pouvant potentiellement s'étendre à d'autres problèmes d'ordonnancement fortement contraints. Nous avons choisi le problème de job shop en raison de sa pertinence, tant dans la recherche scientifique que dans les environnements de production industrielle. Comme mentionné précédemment dans ce chapitre, le problème de job shop existe sous plusieurs variantes. Notre démarche implique une étude graduée du problème en deux étapes. Nous commencerons par examiner le problème de job shop dans sa variante classique (JSP), puis nous aborderons le job shop flexible avec ressources de transport (FJSPT). Cette approche vise à évaluer l'impact de l'ajout de contraintes sur le problème, en déterminant si les résultats obtenus pour la variante de base peuvent conduire à une généralisation des caractéristiques des codages et des opérateurs.

Le prochain chapitre propose une revue de la littérature qui situe notre travail de recherche par rapport aux études existantes sur les métaheuristiques appliquées à la résolution des problèmes d'ordonnancement d'ateliers, en particulier ceux de type job shop.

Chapitre 2
État de l'art

Ce chapitre a pour objectif de positionner notre étude par rapport à la littérature sur la résolution des problèmes d'ordonnancement d'ateliers de type job shop à l'aide de métaheuristiques. Il définit le contexte du présent travail de recherche à travers une revue bibliographique. Tout d'abord, il expose un aperçu des méthodes de résolution utilisées pour les problèmes de type job shop. Ensuite, les représentations de solutions souvent utilisées sont examinées, pour les problèmes d'atelier en général et pour le JSP en particulier, ainsi que pour sa variante flexible avec ressources de transport (FJSPT). Des opérateurs de voisinage dédiés à la recherche locale sont également détaillés. Pour finir, un état de l'art sur l'analyse de paysage de fitness pour les problèmes d'ordonnancement est présenté.

Contenu

2.1 Résolution des problèmes de type job shop . . . 32
2.1.1 Métaheuristiques pour le job shop classique . . . 32
2.1.2 Métaheuristiques pour les variantes de job shop . . . 34
2.1.3 Synthèse des métaheuristiques présentées pour les problèmes de type job shop . . . 36
2.2 Codages et opérateurs de voisinage en ordonnancement d'atelier . . . 39
2.2.1 Codages de solution . . . 39
2.2.2 Opérateurs de voisinage . . . 55
2.3 Paysages de fitness et problèmes d'ordonnancement . . . 58
2.4 Bilan . . . 61

2.1 Résolution des problèmes de type job shop

Plusieurs approches ont été utilisées dans la littérature de la recherche opérationnelle pour la résolution des problèmes d'ordonnancement de type job shop. Indépendamment de nouvelles approches telles que l'intelligence artificielle et la simulation, les modèles exacts, les métaheuristiques et d'abondantes hybridations ont connu une application étendue au cours de la dernière décennie.

Les méthodes exactes, comme indiqué dans la sous-section 1.1.5.1, regroupent des modèles d'optimisation réputés efficaces sur les problèmes dont la combinatoire est modérée. Pour les problèmes de type job shop, elles peuvent prendre les formes de programmation par contraintes [Ham, 2020; Winklehner and Hauder, 2022] ou de programmation linéaire en nombres entiers mixtes [Homayouni and Fontes, 2021; Al-Ashhab et al., 2023; Meng et al., 2023].

L'intelligence artificielle, à travers sa branche d'apprentissage automatique, fait partie des nouvelles techniques intégrées à la recherche opérationnelle classique pour capitaliser sur les connaissances acquises principalement lors de l'exécution des métaheuristiques [Du et al., 2022; Pan et al., 2022; Song et al., 2023; Yuan et al., 2023; Zhu et al., 2023]. A coté des méthodes d'apprentissage, l'usage des systèmes multi-agents [Xu et al., 2022; Kamali et al., 2023], des environnements de simulation [Zaidi et al., 2021; Xu et al., 2022; Zhu et al., 2023] et des jumeaux numériques [Zhang et al., 2022; Chen et al., 2023] n'est pas négligeable. Parfois, ces techniques sont mises à profit de concert dans la quête de solutions optimales [Liu et al., 2023].

Parmi les méthodes approchées, les métaheuristiques et diverses hybridations de méthodes de résolution ont été largement exploitées pour résoudre les problèmes d'ordonnancement de type job shop. Dans la suite, nous présentons quelques travaux de référence sur ce sujet.

2.1.1 Métaheuristiques pour le job shop classique

Comme pour d'autres problèmes d'optimisation, la littérature montre que les problèmes d'ordonnancement de type job shop sont souvent résolus au moyen de métaheuristiques diverses, qu'elles soient à base de solution unique ou à base de population. D'après notre étude bibliographique, principalement des 10 dernières années, il semble que les algorithmes génétiques soient majoritairement utilisés pour le job shop, que ce soit seuls ou hybridés avec d'autres approches [Bierwirth, 1995; Meeran and Morshed, 2012; Salido et al., 2016; Liao and Wang, 2019; Yang et al., 2019].

En 1995, afin de surmonter les problèmes d'infaisabilité des solutions rencontrés avec les techniques de codage standard de l'époque (telles que le codage binaire, la permutation), [Bierwirth, 1995] a introduit un codage basé sur une permutation avec répétition pour le JSP. Cette représentation, ultérieurement nommée codage de Bierwirth ou codage par job, était initialement conçue pour les algorithmes génétiques, mais elle a également été adoptée dans d'autres méthodes de résolution du JSP. Le même codage a été utilisé par [Salido et al., 2016] pour construire leur algorithme génétique visant à traiter le JSP en minimisant le makespan et la consommation d'énergie. Dans une autre étude menée par [Liao and Wang, 2019], un algorithme génétique a été employé pour minimiser les coûts de production et de livraison d'un problème JSP de la vie réelle. Pour se rapprocher davantage des environnements de production réels du JSP, [Yang et al., 2019] ont introduit des nombres flous *(fuzzy numbers)* triangulaires et semi-trapézoïdaux pour exprimer les incertitudes de temps opératoires, les temps d'assemblage et les temps de livraison. Les

auteurs ont également développé un algorithme génétique adapté à ce modèle. Concernant l'hybridation de méthodes incluant les algorithmes génétiques pour le JSP, [Meeran and Morshed, 2012] fournissent une belle illustration. Leur approche combine une recherche taboue et un algorithme génétique pour obtenir la meilleure séquence de tâches à travers les machines, qui minimise le temps total d'exécution. L'algorithme proposé a été utilisé pour des problèmes pratiques en environnement réel, montrant des améliorations tangibles en termes de réduction des délais. Les tests effectués sur 51 instances carrées tirées de la littérature du JSP et 3 instances réelles ont démontré de bonnes performances pour cette approche, avec l'obtention de la solution optimale pour 48 instances sur 51 issues de la littérature.

D'autres métaheuristiques ont également été développées pour le job shop dans sa variante de base, notamment la recherche taboue et le recuit simulé. S'appuyant sur les succès préalables de la recherche taboue dans le contexte du JSP, comme démontré par [Taillard, 1994] et [Nowicki and Smutnicki, 1996], [Ponsich and Coello, 2013] ont décidé de la combiner avec une méthode de recherche globale. Cette décision visait à comparer les performances de l'approche hybride avec celles de l'algorithme de recherche taboue itérée avec sauts en arrière présenté par [Nowicki and Smutnicki, 1996]. La méthode de recherche globale adoptée a été basée sur un algorithme d'évolution différentielle, reconnu pour ses très bonnes performances dans le cadre des problèmes d'optimisation continue. Lors des tests sur 104 instances classiques de littérature, l'efficacité globale de la méthode hybride s'est avérée assez proche de celle de la recherche taboue itérée, sans toutefois la surpasser. Les auteurs ont apporté une nuance à ces résultats en soulignant que la recherche taboue itérée utilise des stratégies d'initialisation avancées, qui dans certains cas conduisent à des solutions sous-optimales, voire à l'optimum, avant l'application de l'heuristique principale. Dans une autre approche axée sur la minimisation du $C_{\max}$ pour le JSP, [Peng et al., 2015] ont développé un algorithme incorporant une recherche taboue dans une méthode de recomposition de chemin (path relinking). Les résultats ont indiqué que cette heuristique a amélioré les bornes supérieures de 49 instances classiques de littérature et a résolu avec succès l'instance `swv15`, qui était restée sans solution optimale connue pendant plus de deux décennies.

L'un des premiers algorithmes efficaces pour la résolution du JSP a été proposé par [van Laarhoven et al., 1992]. Cette heuristique, basée sur un recuit simulé, a introduit un nouvel opérateur de voisinage consistant à échanger la position de deux opérations adjacentes présentes sur le chemin critique. Depuis son introduction, cet opérateur a connu plusieurs variantes, dont celle proposée par [Nowicki and Smutnicki, 1996]. L'efficacité du recuit simulé de [van Laarhoven et al., 1992] pour minimiser le makespan a été démontrée à travers des tests effectués sur 43 instances, dont 3 issues de [Fisher and Thompson, 1963] et 40 de [Lawrence, 1984]. Par la suite, [Zorin and Kostenko, 2015], dans leur quête d'un temps d'exécution optimal respectant les contraintes de délais et d'intégrité d'un système de production en environnement dynamique de JSP, ont également mis en œuvre un recuit simulé. Les tests ont été réalisés sur cet algorithme avec la comparaison de quatre stratégies heuristiques (réduction des délais, réduction des temps morts, stratégie mixte, sélection aléatoire). De ces travaux, une convergence asymptotique a été établie entre l'algorithme proposé et l'évaluation expérimentale. La stratégie mixte a montré une convergence plus rapide comparativement aux autres. D'autre part, dans le but d'accélérer le flux de production et de définir la priorité des jobs dans les systèmes de type job shop, [Güçdemir and Selim, 2017] ont combiné des règles de répartition des lots basées sur les jobs avec des règles de répartition basées sur les machines. Neuf règles de répartition ont été utilisées avec une approche d'optimisation de simulation basée sur un recuit simulé. Les résultats de cette approche ont montré une utilisation plus efficace des ressources, avec un objectif centré sur la satisfaction des clients, visant à minimiser l'écart moyen pondéré du pourcentage des segments d'attente des clients.

Parmi les travaux exploitant d'autres méthodes, nous pouvons citer les exemples suivants. [Huang and Yu, 2017] ont présenté un algorithme hybride principalement basé sur l'optimisation des colonies de fourmis pour résoudre un JSP réel avec un fractionnement de lots en taille égale *(equal-size lot splitting)*. Cet algorithme a été renforcé par cinq techniques d'optimisation visant à minimiser le temps total d'exécution, le retard total et le coût total du fractionnement de lots. Ces techniques impliquent l'utilisation d'un nouveau type de phéromone, l'introduction de nouvelles fonctions de règles de transition d'état, l'intégration d'un algorithme de recherche locale agile, l'incorporation d'un mécanisme de mutation élargissant la portée de la recherche tout en évitant les pièges des minima locaux, et l'ajustement adaptatif des paramètres de l'algorithme. Un autre exemple est l'algorithme d'algues artificielles hybridé avec évolution différentielle par [Ibrahim and Tawhid, 2023] pour l'optimisation du makespan dans le contexte du JSP. L'évolution différentielle a inspiré la conception d'opérateurs de mutation et de croisement, chacun ciblant respectivement l'intensification et la diversification de la recherche de solutions.

2.1.2 Métaheuristiques pour les variantes de job shop, flexible et/ou avec ressources de transport

Selon que la flexibilité des ressources, en particulier celle des machines, est prise en compte et/ou que l'utilisation des ressources de transport est considérée, le problème de job shop se décline en plusieurs variantes, notamment le job shop avec ressources de transport (JSPT), le job shop flexible (FJSP), et le job shop flexible avec ressources de transport (FJSPT). Ces différentes variantes ont été explorées dans plusieurs travaux récents, que nous présentons ci-dessous.

À l'instar du JSP, le JSPT fait également l'objet de différentes métaheuristiques. En partant d'un GRASP *(Greedy Randomized Adaptive Search Procedure)* qui utilise un codage job, comme proposé par [Afsar et al., 2016], et en passant par un VNS *(Variable Neighborhood Search)* appliqué par [Abderrahim et al., 2022], [Fontes et al., 2023] ont abordé le JSPT en hybridant un essaim particulaire avec un recuit simulé. Dans cet algorithme, une représentation de solution basée sur des nombres réels a été adoptée pour encoder la séquence des opérations et l'affectation des ressources de transport. Pour décoder les solutions avec cette représentation, les auteurs ont utilisé une procédure introduite par [Bean, 1994], permettant de convertir les nombres réels en une permutation d'opérations. Les opérateurs de voisinages employés dans le cadre de ces travaux sur le JSPT comprennent l'échange et l'inverse (ou inversion).

Le FJSP a été abordé à travers l'utilisation d'algorithmes génétiques et diverses hybridations, similairement au JSP. Dans l'étude menée par [Zhang et al., 2011], une hybridation avec une heuristique de recherche locale a été réalisée, tandis que dans les travaux de [Li and Gao, 2016] et [Xu et al., 2021] ont pour leur part intégré une recherche taboue. Une nouvelle variante d'algorithme génétique hybride a été élaborée par [Momenikorbekandi and Abbod, 2023]. Cette métaheuristique combine un algorithme parthénogénétique (PGA) et un algorithme génétique ethnique (EGA). Le PGA utilise des opérateurs de recombinaison et de sélection pour générer une descendance, tandis que l'EGA se base sur une combinaison des différentes populations générées à l'aide de diverses méthodes de sélection. Selon les résultats des tests comparatifs réalisés par les auteurs, cette approche présente des avantages tels que l'amélioration de la vitesse de convergence et de l'efficacité globale de la recherche. Outre les algorithmes génétiques, les essaims particulaires ont également été employés pour la résolution du FJSP, comme le montrent les travaux de [Nouiri et al., 2018] et [Sana et al., 2023]. Des approches moins conventionnelles ont également été explorées pour cette variante du JSP, incluant un *Shuffled Frog-Leaping Algorithm* par [Lei et al., 2017], un *Backtracking Search Algorithm* par [Caldeira et al., 2020], ainsi qu'une combinaison d'apprentissage par

renforcement avec des règles de répartition proposée par [Du et al., 2022]. La plupart de ces algorithmes reposent sur un codage de solutions qui encapsule la séquence des opérations ainsi que l'affectation des machines pour ces opérations.

Pour la résolution du FJSPT, on note des métaheuristiques incluant principalement des algorithmes génétiques. [Zhang et al., 2019a] ont mis en oeuvre une version améliorée d'un algorithme génétique générique dans le but de minimiser le temps total d'exécution pour le FJSPT. Ils ont proposé une méthode d'insertion d'opérations avec décalage à gauche en mettant à profit les intervalles d'inactivité sur les machines. Cette technique permet d'obtenir des ordonnancements actifs. D'autres variantes d'algorithmes génétiques ont également été utilisées par [Dai et al., 2019], [He et al., 2021], [Yan et al., 2021] et [Stanković et al., 2022].

De leur côté, [Zhang et al., 2012] ont développé une approche hybride combinant une recherche taboue avec un algorithme génétique pour résoudre le problème du job shop flexible, intégrant des contraintes de transport et des temps de traitement bornés. L'objectif était de minimiser à la fois le temps total d'exécution et le temps total d'attente avant et après chaque machine tout au long du processus de production. Dans un contexte d'environnement multi-agents, [Nouri et al., 2016b] ont également opté pour une hybridation similaire afin de minimiser le makespan pour le problème de job shop flexible, intégrant le transport avec des robots. Leur étude a conduit à la découverte d'une trentaine de nouvelles bornes supérieures sur les 65 instances testées. Par le biais d'une combinaison entre un algorithme génétique et un recuit simulé, [Huang and Yang, 2019] ont démontré l'amélioration des performances de leur algorithme, tant en termes de qualité que de distribution des solutions, à mesure que le nombre de jobs et la flexibilité des machines augmentent. Une hybridation du *Nondominated Sorting Genetic Algorithm* avec une recherche à voisinage variable a également été présentée par [Han et al., 2022].

Une adaptation de l'algorithme compétitif impérialiste (ICA) couplée à une recherche locale basée sur le recuit simulé, a été élaborée par [Karimi et al., 2017] dans le contexte de la résolution du FJSPT, avec la minimisation du makespan comme fonction objectif. En comparant cette heuristique à deux algorithmes de programmation linéaire en nombres entiers mixtes et à deux algorithmes évolutionnaires, les résultats ont montré que l'ICA mis en œuvre a produit les meilleurs résultats sur un ensemble de 80 instances générées. De même, [Peng et al., 2022], travaillant sur une instance réelle, ont adopté une approche similaire pour résoudre une variante du FJSPT, confirmant l'efficacité de cette approche hybride. L'algorithme compétitif impérialiste a été employé dans une autre étude menée par [Li and Lei, 2021] visant à minimiser simultanément le makespan, le retard et la consommation en énergie, démontrant ainsi la polyvalence de cette méthode dans la résolution de divers objectifs.

Plusieurs autres types de métaheuristique ont eu de bons résultats pour la variante flexible avec ressources de transport du job shop. Alors que [Kumar et al., 2011] ont exploité un algorithme d'évolution différentielle, [Liu et al., 2013] ont adopté une approche basée sur une colonie artificielle d'abeilles pour aborder le problème. En introduisant des contraintes de temps de réglages (setup times) séparables sur les machines dans le contexte du FJSPT, [Rossi, 2014] ont opté pour l'utilisation d'un système de colonie de fourmis avec un apprentissage par renforcement dans le but de minimiser le temps total d'exécution. Pour les mêmes fonctions objectifs que [Li and Lei, 2021] vu précédemment, [Li et al., 2020] ont adopté une approche hybride en intégrant une heuristique basée sur le recuit simulé dans un algorithme Jaya pour résoudre le problème du job shop avec des temps de transport et des temps de configuration de machines. D'après les auteurs, leur étude comparative a révélé l'efficacité de leur algorithme par rapport à six autres méthodes

existantes considérées comme performantes. Une autre contribution vient de [Chen et al., 2020], qui ont présenté une métaheuristique hybride combinant le recuit simulé et un algorithme immunitaire artificiel pour optimiser le makespan et la consommation d'énergie dans le contexte du job shop, tout en tenant compte des contraintes de temps de transport et de la consommation d'énergie. L'efficacité de leur algorithme a été démontrée par des comparaisons avec un algorithme impérialiste compétitif, un algorithme de recherche à voisinage variable, ainsi que l'algorithme génétique de [Dai et al., 2019] mentionné précédemment. En outre, les méthodes proposées dans [Zhang et al., 2021], [Homayouni and Fontes, 2021] et [Xu et al., 2022] illustrent la diversité des approches de résolution du FJSPT.

2.1.3 Synthèse des métaheuristiques présentées pour les problèmes de type job shop

Le tableau 2.1 présente une synthèse des métaheuristiques recensées ci-dessus. Il est énuméré, pour chaque méthode, les différentes approches avec lesquelles elle a été hybridée, les variantes de job shop traitées de même que les objectifs ciblés par les travaux. Il est évident qu'aucune métaheuristique ou hybridation quelconque ne surpasse incontestablement l'autre en termes de performance. Car, par endroit et en fonction de la conception et/ou des paramétrages, l'une ou l'autre obtient les meilleurs résultats. Par conséquent, les facteurs qui influent sur l'efficacité de ces méthodes sont des éléments de fond plutôt que de forme, justifiant ainsi l'intérêt de suivre la recommandation de [Sörensen, 2015] sur l'analyse des éléments de fond qui entrent dans la conception des métaheuristiques. L'efficacité d'une métaheuristique est potentiellement liée au problème traité ainsi qu'aux composants de la métaheuristique. Comme expliqué précédemment dans la section 1.1.6, les codages de solution et les opérateurs de recherche occupent une place prépondérante parmi les différents composants classiques des métaheuristiques. Ainsi, le choix de ces composants doit faire l'objet d'une attention particulière.

Dans la suite de ce chapitre, nous présentons différents codages et opérateurs de voisinage, principalement issus de travaux récents sur le problème de job shop et ses variantes.

Tableau 2.1 – Approches métaheuristiques pour les problèmes de type job shop

Méthode	Hybridation	Variantes	Objectifs	Références
Algorithme génétique *(GA)*	*SA, TS, VNS*	JSP, FJSP, FJSPT+	$C_{\max}$, *attente, retard, charge, énergie, coût*	[Bierwirth, 1995; Zhang et al., 2011; Meeran and Morshed, 2012; Zhang et al., 2012; Nouri et al., 2016b; Li and Gao, 2016; Salido et al., 2016; Yang et al., 2019; Huang and Yang, 2019; Zhang et al., 2019a; Liao and Wang, 2019; Dai et al., 2019; Yan et al., 2021; Xu et al., 2021; He et al., 2021; Stanković et al., 2022; Han et al., 2022; Momenikorbekandi and Abbod, 2023]
Recuit simulé *(SA)*	*ABC, AIA, GA, ICA, JA, PSO*	JSP, JSPT, FJSPT+	$C_{\max}$, *charge, énergie, bruit, priorité, temps CPU*	[van Laarhoven et al., 1992; Zorin and Kostenko, 2015; Güçdemir and Selim, 2017; Karimi et al., 2017; Huang and Yang, 2019; Chen et al., 2020; Li et al., 2020; Peng et al., 2022; Fontes et al., 2023]
Recherche taboue *(TS)*	*ACO, DE, GA, PR, PSO*	JSP, FJSP, FJSPT+	$C_{\max}$, *attente, coût, retard*	[Nowicki and Smutnicki, 1996; Meeran and Morshed, 2012; Zhang et al., 2012; Ponsich and Coello, 2013; Peng et al., 2015; Nouri et al., 2016b; Li and Gao, 2016; Huang and Yu, 2017; Xu et al., 2021]
Essaim particulaire *(PSO)*	*ACO, PSO, TS*	JSP, JSPT, FJSP, FJSPT	$C_{\max}$, *retard, coût, temps CPU*	[Huang and Yu, 2017; Nouiri et al., 2018; Stanković et al., 2022; Fontes et al., 2023; Sana et al., 2023]
Evolution différentielle *(DE)*	*AAA, TS*	JSP, FJSPT	$C_{\max}$	[Kumar et al., 2011; Ponsich and Coello, 2013; Ibrahim and Tawhid, 2023]
Algorithme compétitif impérialiste *(ICA)*	*SA*	FJSPT+	$C_{\max}$ *, retard, énergie, bruit*	[Karimi et al., 2017; Li and Lei, 2021; Peng et al., 2022]
Recherche à voisinage variable *(VNS)*	*GA*	JSPT, FJSPT+	$C_{\max}$, *énergie, coût*	[He et al., 2021; Han et al., 2022; Abderrahim et al., 2022]
Colonie artificielle d'abeilles *(ABC)*	–	FJSPT	$C_{\max}$, *charge*	[Liu et al., 2013; Stanković et al., 2022]

Suite à la page suivante ...

Tableau 2.1 – (Suite)

Méthode	Hybridation	Variantes	Objectifs	Références
Colonie de fourmis *(ACO)*	*PSO, TS*	JSP, FJSPT+	$C_{\max}$, *retard, coût*	[Rossi, 2014; Huang and Yu, 2017]
Algorithme mémétique *(MA)*	–	FJSPT+	$C_{\max}$, *charge, énergie*	[Luo et al., 2020; He et al., 2021]
Algorithme immunitaire artificiel *(AIA)*	*SA*	FJSPT+	$C_{\max}$, *énergie*	[Chen et al., 2020]
Algorithme Jaya *(JA)*	*SA*	JSPT+	$C_{\max}$, *énergie*	[Li et al., 2020]
Algorithme d'algues artificielles *(AAA)*	*DE*	JSP	$C_{\max}$	[Ibrahim and Tawhid, 2023]
Recomposition de chemin *(PR)*	*TS*	JSP	$C_{\max}$	[Peng et al., 2015]
GRASP / Shuffled Frog-Leaping Algorithm / Backtracking Search Algorithm / Hill Climbing / Heuristique	–	JSPT, FJSP, FJSPT+	$C_{\max}$, *charge, énergie*	[Afsar et al., 2016; Lei et al., 2017; Caldeira et al., 2020; Homayouni and Fontes, 2021; Zhang et al., 2021]

(+) désigne des contraintes additionelles

2.2 Codages et opérateurs de voisinage en ordonnancement d'atelier

2.2.1 Codages de solution

Pour un même problème d'optimisation, plusieurs espaces de recherche peuvent être générés, chacun reposant sur un choix de représentation de solution bien défini [Rothlauf, 2006]. Dans la suite de cette section, des codages tirés des métaheuristiques proposées pour les problèmes d'ordonnancement d'atelier en général et pouvant être adaptés aux différentes variantes du problème de job shop sont présentés. Les notations utilisées pour les problèmes de job shop abordés dans ce document sont récapitulées et expliqués dans le tableau 2.2.

Tableau 2.2 – Description des notations utilisées pour le JSP/FJSPT

Notation	Description
J	ensemble de jobs à traiter
M	ensemble de machines (ressources de traitement)
R	ensemble de ressources de transport
n	nombre de jobs dans J
m	nombre de machines dans M
r	nombre de ressources de transport dans R
J_i	job i, $(i \in [1, n])$
N_i	nombre d'opérations de J_i
$O_{i,j}$	j^{e} opération de J_i, $j \in [1, N_i]$
$T_{i,j}$	tâche de transport conduisant à $O_{i,j}$
$M_{i,j}$	sous-ensemble de machines dans M capables de traiter $O_{i,j}$
$m_{i,j}$	nombre de machines dans $M_{i,j}$
M_k	machine k, $(k \in [1, m])$
R_h	ressource de transport h, $(h \in [1, r])$
LU	station de chargement/déchargement
r_i	date de disponibilité de J_i
$s_{i,j,k}$	date de début de $O_{i,j}$ sur M_k
$p_{i,j,k}$	temps de traitement de $O_{i,j}$ sur M_k
$t^E_{i,j}$	temps de transport à vide de $T_{i,j}$
$t^L_{i,j}$	temps de transport en charge de $T_{i,j}$
C_i	date de complétion de J_i
$C_{\max}$	date de complétion totale, makespan défini par le max des C_i

Le job shop classique répond à un problème de séquencement des opérations, tandis que le job shop flexible avec ressources de transport peut être décomposé en quatre sous-problèmes distincts.

1. Séquencement des opérations (OS) : il s'agit de déterminer dans quel ordre les opérations de chaque job seront effectuées.
2. Affectation des machines (MA) : il s'agit de décider quelle machine sera utilisée pour effectuer chaque opération.

3. Séquencement des tâches de transport (TS) : il s'agit de déterminer l'ordre dans lequel les tâches de transport seront exécutées par les ressources de transport.
4. Affectation des ressources de transport (TA) : il s'agit de décider quelles ressources de transport utiliser pour convoyer les jobs entre les machines.

Afin d'illustrer les différences entre les codages, une solution à l'instance présentée dans le tableau 2.3 sera représentée par chacun des codages. L'instance décrit 2 jobs à effectuer sur 3 machines avec une station LU (chargement/déchargement) et 2 ressources de transport. Pour chaque opération, une seule machine est requise et affectée parmi les machines possibles. De même chaque transport ne mobilise qu'une ressource de transport. Les temps de transport sont indiqués dans le tableau 2.4 en supposant que les temps de transport à vide et en charge sont identiques.

Tableau 2.3 – Durées opératoires de l'instance illustrative

Jobs	Operations	Machines		
		M_1	M_2	M_3
J_1	$O_{1,1}$	8	-	4
	$O_{1,2}$	8	4	-
	$O_{1,3}$	10	14	4
J_2	$O_{2,1}$	-	6	10
	$O_{2,2}$	6	4	8

Tableau 2.4 – Temps de transport de l'instance illustrative

	LU	M_1	M_2	M_3
LU	0	1	2	2
M_1	5	0	4	4
M_2	2	5	0	3
M_3	2	4	2	0

2.2.1.1 Codage job (`job`)

Le *codage job*, désigné dans la suite par `job`, représente une solution par une liste d'entiers. La taille de la liste est égale au nombre d'opérations et chaque élément correspond à un numéro de job. Au sein de cette liste, le numéro i d'un job ($i = 1$ à n) apparaît autant de fois qu'il comporte d'opérations, soit N_i fois. Ce codage se caractérise par une génération probable de redondance. Toute solution générée avec le codage job est faisable, car les opérations sont représentées par les numéros des jobs qui leur correspondent, et le moteur d'ordonnancement prend en compte les contraintes de gamme lors du décodage de la solution. En substance, la première apparition de J_1 correspond à l'opération $O_{1,1}$, sa deuxième apparition à l'opération $O_{1,2}$ et ainsi de suite. Il en est de même pour les autres jobs. L'avantage de cette représentation,

introduite par [Bierwirth, 1995], réside dans le fait qu'elle nécessite seulement un simple constructeur d'ordonnancement (plaçant les opérations dans leur ordre d'apparition, au plus tôt sur les machines), couvrant ainsi l'espace de tous les ordonnancements semi-actifs possibles dans un contexte de job shop classique. Soit S l'ensemble des solutions possibles, le nombre de solutions est obtenu par :

$$Card(S) = \frac{(\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$$

Dans un job shop classique, le problème d'affectation de machines ne se pose pas. En revanche, dans le cas d'un *job shop flexible*, ce codage qui ne résout que le problème de séquencement des jobs n'est pas suffisant pour construire un ordonnancement ; parce qu'elle n'indique pas les machines sur lesquelles les opérations doivent s'exécuter. Il sera donc nécessaire d'adjoindre une règle d'affectation de machines.

Pour l'instance exemple, 10 solutions sont possibles. La figure 2.1 illustre l'une de ces solutions. La première opération de J_1 est ordonnancée en premier, ensuite la deuxième opération du même job puis la première opération de J_2 suivi de la troisième et dernière opération de J_1. L'ordonnancement est clôturé avec la dernière des opérations de la gamme opératoire de J_2.

1	1	2	1	2
$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{1,3}$	$O_{2,2}$

FIGURE 2.1 – Solution représentée avec le codage `job`

2.2.1.2 Codage opération (`ope`)

Le *codage opération*, désigné dans la suite par `ope`, représente une solution par une liste ordonnée d'opérations dans laquelle chaque opération apparaît exactement une fois. La liste dont la taille est égale au nombre total d'opérations est une permutation de l'ensemble des opérations. Ce codage a été utilisé dans [Kumar et al., 2011] pour les systèmes de fabrication flexibles. Contrairement au codage `job`, certaines des solutions générées par `ope` sont infaisables. Les permutations d'opérations indépendamment des gammes engendrent un grand nombre de solutions qui violent les contraintes de précédence. Parmi les solutions possibles, le nombre de solutions faisables est égal au nombre de solutions possibles pour le codage `job`. Il existe une bijection entre l'ensemble des solutions faisables de `ope` et l'ensemble des solutions possibles de `job`, en ce sens que les ordonnancements produits par les deux ensembles sont identiques. Le nombre de solutions possibles, pour `ope`, est $(\sum_{i=1}^{n} N_i)!$, ce qui correspond à l'ensemble des permutations des opérations. Soit S_{inf} l'ensemble des solutions infaisables, sa taille est obtenue par :

$$Card(S_{inf}) = \frac{(\sum_{i=1}^{n} N_i)! \times (\prod_{i=1}^{n} N_i! - 1)}{\prod_{i=1}^{n} N_i!}$$

L'affectation de machines n'est pas pris en compte dans `ope`. Dans un contexte flexible, une règle d'affectation est indispensable. L'ordonnancement consiste donc en une succession d'opérations sur les différentes machines suivant l'ordre dans lequel elles apparaissent dans le codage.

Pour notre exemple, sur 120 solutions possibles, l'instance compte 110 solutions infaisables. Sur la figure 2.2, la solution (a) est une solution faisable correspondant à la solution de la figure 2.1 dans le codage `job`. La solution (b) est une solution infaisable, car elle viole une contrainte de précédence vis à vis des opérations $O_{1,3}$ et $O_{1,2}$ de J_1.

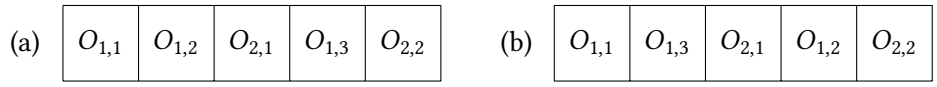


FIGURE 2.2 – Solutions représentées avec le codage `ope`

2.2.1.3 Codage machine (`mch`)

Une solution avec le *codage machine*, noté `mch`, est une liste de m listes d'opérations assignées à une même machine, m étant le nombre de machines de l'atelier. La taille des listes varie en fonction du nombre d'opérations N_k pour chaque machine k. [van Laarhoven et al., 1992] a utilisé cette représentation pour le JSP, et [Vallada and Ruiz, 2011] pour le problème de machines non-reliées parallèles. Soit S l'ensemble des solutions possibles, la taille de S est obtenue par :

$$Card(S) = \prod_{k=1}^{m} N_k!$$

Les permutations sur les différentes listes sont susceptibles d'occasionner des violations de contraintes de gamme, d'où la présence de solutions infaisables. À l'opposé de `ope` où le nombre de solutions infaisables est quantifiable avec précision, les effectifs de solutions faisables et infaisables pour `mch` dépendent intrinsèquement de l'instance traitée. Ce codage définit les séquences d'opérations sur chaque machine ; il prend donc en compte l'affectation des machines et le séquencement des opérations. Dans un contexte flexible, il est nécessaire d'arbitrer en amont la machine à assigner à chaque opération.

La figure 2.3 montre deux solutions représentées par `mch` pour l'instance illustrative. La solution (b) est infaisable en raison d'une situation de blocage sur M_2 et M_3 : $O_{1,2}$ ne peut pas être ordonnancé avant $O_{2,1}$ car $O_{2,2}$ devrait venir avant $O_{1,1}$ sur M_3. La solution (a) est faisable : il suffit de réaliser toutes les opérations sur M_3 avant de commencer une opération sur M_2.

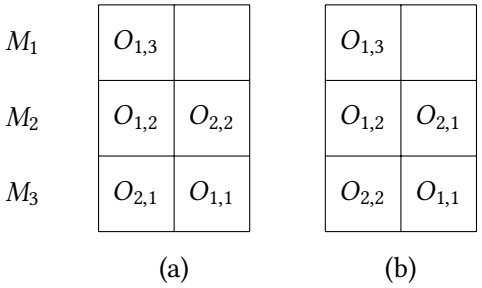


FIGURE 2.3 – Solutions représentées avec le codage `mch`

2.2.1.4 Codage matriciel (`mtx`)

Le *codage matriciel*, noté `mtx`, représente une solution par une matrice de nombres réels. Les lignes de la matrice désignent les machines tandis que les colonnes désignent les opérations. Ce codage a été utilisé à la base par [Balin, 2011] pour le problème de machines parallèles. [Vlašic et al., 2020] s'en sont également servi pour le problème de machines non-reliées. Pour ces catégories de problème d'ordonnancement, les opérations sont implicitement assimilées aux jobs. Une adaptation est donc nécessaire pour appliquer ce codage au problème de job shop. Dans le cas d'espèce, les opérations désignent effectivement les opérations des jobs et non les jobs eux-même. La matrice compte donc m lignes et $\sum_{i=1}^{n} N_i$ colonnes. Les opérations du job 1 constituent les n_1 premières colonnes. Les n_j opérations du job j ($j = 2$ à n) sont représentées par les colonnes $1 + \sum_{i=1}^{j-1} N_i$ à $\sum_{i=1}^{j} N_i$. Chaque cellule contient un nombre réel compris dans l'intervalle $[0, 1]$. Pour une cellule contenant un nombre non nul, l'opération correspondant à la colonne est exécutée sur la machine indiquée par la ligne. Si la valeur de la cellule est égale à 0, l'opération de la colonne correspondante n'est pas programmée sur la machine de la ligne concernée. Sur chaque machine, les opérations sont ordonnancées dans l'ordre croissant des valeurs des cellules qui les représentent.

Cette représentation de solution présente des similitudes avec le codage `mch`. Les affectations de machines sont résolues de la même façon avec les deux codages. En ce qui concerne le séquencement, bien que l'ensemble des valeurs aléatoires possibles que peuvent prendre les cellules non nulles d'une ligne soit infini pour le codage `mtx`, le décodage de ces valeurs se ramène à l'ensemble des permutations des opérations utilisant la même machine comme c'est le cas pour `mch`. Avec une infinité de solutions possibles, la taille d'une solution issue de `mtx` est égale à la dimension de la matrice. Cependant, le nombre total des séquences d'opérations possibles sur l'ensemble des machines est identique à celui du codage `mch`. En d'autres termes, l'effectif des solutions possibles avec décodage de `mtx` est égal à l'effectif des solutions possible de `mch`. L'intérêt de ce codage par rapport à `mch` est de lui associer des opérateurs de voisinage différents, permettant une structure différente de l'espace de recherche.

La figure 2.4 présente l'une des solutions possibles équivalente à l'exemple (a) de la figure 2.3 du `mch`. Sur la ligne M_3, les valeurs des cellules des opérations $O_{1,1}$ et $O_{2,1}$ sont différentes de 0, signifiant que ces opérations sont à exécuter sur M_3. L'opération $O_{2,1}$ ayant le réel le plus petit, elle est exécutée en premier. Les séquences d'opérations sur les machines M_1 et M_2 sont obtenues de la même manière.

	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$
M_1	0	0	0.41	0	0
M_2	0	0.11	0	0	0.69
M_3	0.85	0	0	0.36	0

FIGURE 2.4 – Solution représentée avec le codage `mtx`

2.2.1.5 Codage clé aléatoire (`rkey`)

Le *codage clé aléatoire*, noté `rkey`, représente une solution sous la forme d'une liste de nombres réels dont la taille est égale au nombre d'opérations. Cette représentation a été appliquée par [Bean, 1994] pour le problème à machines multiples et le problème d'affectation quadratique. Chaque élément de la liste a une valeur comprise dans l'intervalle $[1, m+1[$, m étant le nombre total de machines. À chaque position de la liste est associée une opération suivant l'ordre naturel d'énumération des opérations. Pour l'instance illustrative par exemple, le premier élément est associé à $O_{1,1}$, le deuxième à $O_{1,2}$ et ainsi de suite jusqu'au dernier $O_{2,2}$. Le codage par clé aléatoire encapsule l'affectation et le séquencement des opérations sur les machines. Pour chaque élément de la liste, la partie entière correspond à la machine k affectée et la partie décimale indique l'ordre de priorité de l'opération associée sur k.

Dans un contexte de job shop, pour qu'une opération soit faisable, la partie entière du réel correspondant doit faire partie de la liste des machines candidates de ladite opération. La pertinence de cette structure est manifeste pour les cas qui intègrent une flexibilité totale des machines, ce qui marque la différence avec `mtx`. Il existe une infinité de solutions possibles avec le `rkey`. Cependant, le nombre de solutions faisables est identique à celui de `mch`. La figure 2.5 montre un équivalent de la solution faisable de `mch` (figure 2.3 (a)) pour `rkey`. Après la déduction des séquences d'opérations de l'ensemble des machines, l'ordonnancement est construit à l'image de celui issu du codage `mch` avec nécessité d'une règle de sélection.

$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$
3.77	2.05	1.34	3.18	2.96
M_3	M_2	M_1	M_3	M_2

FIGURE 2.5 – Solution représentée avec le codage `rkey`

2.2.1.6 Codage séquence d'opérations et affectation de machines (`osma`)

Le *codage séquence d'opérations et affectation de machines*, noté `osma`, est une représentation à deux couches largement utilisée pour le job shop flexible (FJSP) et ses extensions [Zhang et al., 2011; Karimi et al., 2017; Huang and Yang, 2019]. Une solution avec ce codage se compose d'un vecteur de séquence d'opérations (OS) et d'un vecteur d'affectation de machine (MA). Le vecteur OS est équivalent à la liste de jobs avec le codage `job` pendant que celui de MA comprend les indices des machines candidates pour les opérations dédiées. L'interprétation de la liaison des éléments entre les deux vecteurs peut se faire de deux façons différentes. Pour [Chen et al., 2020] comme pour beaucoup d'autres, les éléments du vecteur MA sont liés aux opérations suivant l'ordre ordinaire des opérations des jobs. Toutefois, dans le cadre de la présente étude, nous retenons l'interprétation de [Han et al., 2022]. Elle stipule que chaque élément du vecteur MA est l'indice d'une machine parmi les candidates correspondant à l'opération à la même position dans le vecteur OS. Une solution infaisable survient lorsqu'un élément du vecteur MA est supérieur au nombre de machines candidates correspondantes. La taille du codage est donc égale au double de celle de `job`, soit $2 \times \sum_{i=1}^{n} N_i$. Soit S l'ensemble des solutions possibles, le nombre de solutions est déduit de la formule suivante :

$$Card(S) = \frac{m^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$$

Soient $m_{i,j}$ le nombre de machines candidates de l'opération $O_{i,j}$ et S_{inf} l'ensemble des solutions infaisables, l'effectif de S_{inf} est obtenu comme suit :

$$Card(S_{inf}) = (m^{\sum_{i=1}^{n} N_i} - \prod_{i=1}^{n} \prod_{j=1}^{N_i} m_{i,j}) \times \frac{(\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$$

Le nombre de solutions faisables se déduit de la différence de ces deux valeurs. Par ailleurs, dans [Durasević and Jakobović, 2016], une variante de ce codage a été utilisée pour un environnement de machines non-reliées. L'avantage du `osma` réside dans le fait qu'il définit explicitement non seulement l'ordre d'exécution des opérations mais aussi la machine de traitement de chaque opération. Pour un JSP classique, la liste de machines est implicite car une opération possède exactement une machine candidate. En revanche, dans un système flexible, la génération de chaque élément du vecteur MA nécessite un choix parmi plusieurs machines éligibles.

La figure 2.6 présente l'une des 720 solutions faisables sur les 2 430 solutions possibles pour l'instance exemple avec `osma`. $O_{1,1}$ est exécutée sur M_3, car M_3 vient en deuxième position dans l'ensemble des machines candidates pour $O_{1,1}$, d'après le tableau 2.3.

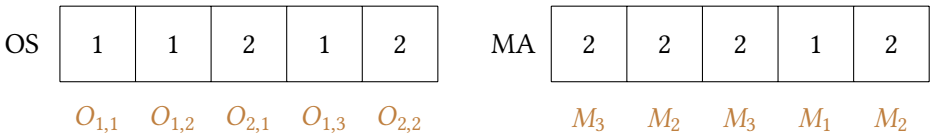


FIGURE 2.6 – Solution représentée avec le codage `osma`

2.2.1.7 Codage séquence d'opérations et comptage machine (`osmc`)

Le *codage séquence d'opérations et comptage machine*, noté `osmc`, est une représentation de solution initialement conçue par [Carter and Ragsdale, 2006] pour le problème du voyageur de commerce (TSP). Il a été adapté au problème de machines non-reliées dans [Vlašic et al., 2020]. Ce codage comporte deux parties. La première est une permutation des opérations (villes pour le TSP). La seconde est une liste d'entiers dont la taille est égale au nombre de machines (vendeurs pour le TSP). Chaque élément de la seconde liste représente le nombre d'opérations associées à la machine de l'indice correspondant. Ainsi, le premier élément de la liste indique le nombre d'opérations affectées à la première machine M_1, à partir du début de la permutation des opérations. Le second élément indique le nombre d'opérations attribuées à M_2 sur le restant des permutations, et ainsi de suite.

Pour augmenter la faisabilité des solutions, une adaptation de ce codage au problème de job shop, pourrait consister à considérer la permutation des opérations comme étant une solution du codage job, l'équivalent du vecteur OS avec `osma`. La taille d'une solution avec `osmc` est égale à $m + \sum_{i=1}^{n} N_i$. Déterminer l'effectif des solutions possibles pour ce codage revient à multiplier le nombre de combinaisons de la partie OS par le nombre de combinaisons possibles pour la seconde partie du codage `osmc` qui traite le comptage machine. Il s'agit de trouver toutes les combinaisons d'additions possibles pour obtenir le nombre $\sum_{i=1}^{n} N_i$ en utilisant exactement m opérandes non négatifs et non nuls, soit $\binom{\sum_{i=1}^{n} N_i - 1}{m-1}$ possibilités [Carter and Ragsdale, 2006]. L'effectif de l'ensemble de solutions S avec ce codage est obtenu par :

$$Card(S) = \frac{(\sum_{i=1}^{n} N_i)! \times (\sum_{i=1}^{n} N_i - 1)!}{\prod_{i=1}^{n} N_i! \times (m-1)! \times (\sum_{i=1}^{n} N_i - m)!}$$

Pour l'instance d'illustration, le nombre de possibilités de la partie comptage machine est 21. La figure 2.7 illustre une solution faisable pour l'instance exemple avec le codage `osmc`. Les deux premières opérations à ordonnancer sont $O_{1,1}$ et $O_{1,2}$; elles sont affectées à la machine M_1. Ensuite, suivent $O_{2,1}$ et $O_{1,3}$ sur M_2 après quoi $O_{2,2}$ clôture l'ordonnancement sur M_3.

En l'absence de flexibilité ou en cas de flexibilité partielle sur les machines, des situations d'infaisabilité sont susceptibles d'advenir. Par exemple, si M_1 ne faisait pas partie des machines candidates de $O_{1,1}$, la solution présentée à la figure 2.7 serait infaisable.

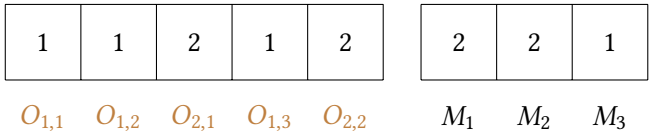


FIGURE 2.7 – Solution représentée avec le codage `osmc`

2.2.1.8 Codage séquence d'opérations et affectation de ressources de transport (`osta`)

Le *codage séquence d'opérations et affectation de ressources de transport*, noté `osta`, représente une solution avec deux vecteurs de même taille : un vecteur de séquence d'opérations (OS) identique à celui de `osma` et un vecteur d'affectation des ressources de transport (TA). [Abderrahim et al., 2022] a utilisé ce codage pour le problème d'ordonnancement de type job shop avec transport. En faisant correspondre les vecteurs OS et TA par leurs indices, TA représente les ressources de transport affectées aux tâches de transport associées aux opérations de OS. De cette manière, les tâches de transport $T_{i,j}$ représentent le transport du job i entre les machines réalisant respectivement les opérations $j-1$ et j (avec les temps de $T_{i,1}$ sont considérés nuls car les auteurs ne prennent pas en compte les tâches de chargement à la station LU). En général, lorsqu'on prend en considération le transport de jobs dans le problème, des opérations fictives sont incorporées aux données des instances pour gérer le chargement et le déchargement des jobs. Pour représenter le chargement d'un job de la station LU sur sa première machine, une opération fictive avec un temps de traitement nul est insérée au début de la séquence des opérations du job. De manière similaire, pour le déchargement à la sortie de la dernière machine, une opération fictive est ajoutée à la fin de la séquence d'opérations. Toutes les solutions sont faisables avec ce codage. L'effectif de l'ensemble S des solutions est obtenu par la formule suivante (en guise de rappel, r est le nombre de ressources de transport) :

$$Card(S) = \frac{r^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$$

Avec `osta`, l'instance exemple compte 320 solutions dont celle présentée dans la figure 2.8. Cette figure indique que R_2 assure $T_{1,2}$ en transportant J_1 vers la machine affectée à $O_{1,2}$. Cette représentation de solution nécessite une règle d'affectation pour le choix de machine.

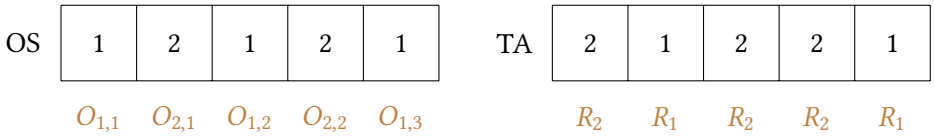


FIGURE 2.8 – Solution représentée avec le codage `osta`

2.2.1.9 Codage séquence d'opérations et affectation de ressources de transport en réels (`rosta`)

Le *codage séquence d'opérations et affectation de ressources de transport en réels*, noté `rosta`, est similaire au codage `osta`, à la différence que les vecteurs d'entiers sont remplacés par des vecteurs de réels [Fontes et al., 2023]. Les éléments du vecteur OS sont initialement générés de manière aléatoire dans l'intervalle $]0, 1[$ et les éléments du vecteur TA dans l'intervalle $]0, r[$. Pour décoder le vecteur OS, une méthode nommée *smallest position value* conçue par [Bean, 1994] est utilisée pour convertir les nombres réels en une permutation d'opérations. Cette conversion consiste à ranger les opérations par ordre croissant au regard des valeurs des réels qui leurs sont nommément assignées. Ensuite, la permutation d'opérations est transformée en une permutation des jobs correspondants ; ceci afin de garantir la faisabilité. En ce qui concerne le vecteur TA, les nombres réels sont convertis en numéros de ressources de transport en les arrondissant par excès en entiers. Ce codage a fondamentalement une infinité de possibilités en terme de solutions. Cependant, son décodage ramène l'ensemble des solutions à celui du `osta`. Une solution avec ce codage est détaillée dans la figure 2.9.

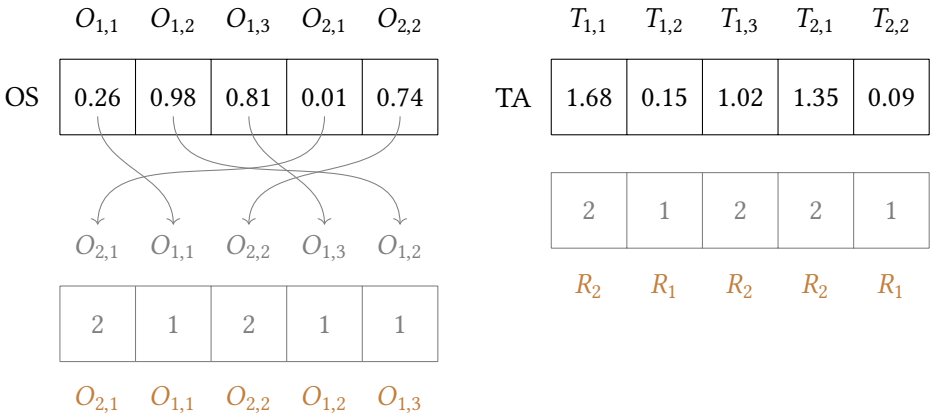


FIGURE 2.9 – Solution représentée avec le codage `rosta`

2.2.1.10 Codage affectation de transport et de job (`tja`)

Le *codage affectation de transport et de job*, noté `tja`, utilisé dans [Zhang et al., 2012] pour le job shop flexible avec transport (FJSPT) en considérant des temps de traitement bornés, comporte deux parties. D'une part, l'affectation des machines (MA) aux opérations de job (générée comme pour `osma`) et d'autre part l'affectation des ressources de transport (TA) aux tâches de transport (générée comme pour `osta`). Cette représentation de solution n'induit aucun séquencement des opérations. À cet effet, pour construire l'ordonnancement de chaque solution, une séquence aléatoire d'opérations est générée. L'effectif de l'ensemble S des solutions possibles est de $(m \times r)^{\sum_{i=1}^{n} N_i}$. La flexibilité des machines est susceptible d'être partielle comme c'est le cas dans la table 2.3 où toutes les machines ne peuvent pas réaliser toutes les opérations. Dans ces conditions, le codage peut générer des solutions infaisables. La séquence d'opérations générée doit a priori respecter les gammes opératoires afin d'éviter l'infaisabilité liée aux contraintes de précédence. Soit S_{inf} l'ensemble des solutions infaisables, son effectif est obtenu comme suit :

$$Card(S_{inf}) = (m^{\sum_{i=1}^{n} N_i} - \prod_{i=1}^{n} \prod_{j=1}^{N_i} m_{i,j}) \times r^{\sum_{i=1}^{n} N_i}$$

L'instance illustrative génère 2 304 solutions faisables contre 5 472 infaisables. La figure 2.10 présente une solution avec `tja`. Quelle que soit sa position dans l'ordonnancement, l'opération $O_{1,3}$ doit être convoyée à la machine M_1 par la ressource de transport R_2.

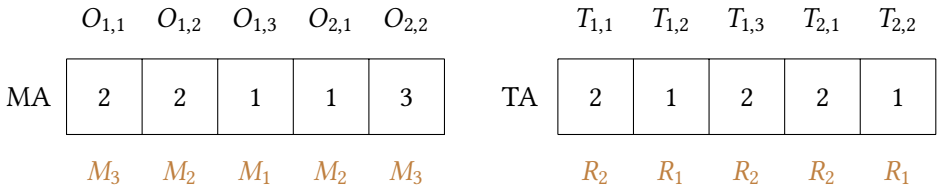


FIGURE 2.10 – Solution représentée avec le codage `tja`

2.2.1.11 Codage séquence d'opérations et affectation de ressources (`osra`)

Le *codage séquence d'opérations et affectation de ressources*, noté `osra`, est une représentation de solution à trois couches [Homayouni and Fontes, 2021]. La première couche encode la séquence d'opérations (OS) dans le même format que `job`, tandis que la deuxième et la troisième encodent respectivement l'affectation des machines (MA) et l'affectation des ressources de transport (TA). `osra` étend `osma` en ajoutant une couche TA qui indique les ressources de transport qui doivent être utilisées pour acheminer les jobs suivant le vecteur OS. En ajoutant le vecteur TA, la taille du codage revient à $3 \times \sum_{i=1}^{n} N_i$ et l'effectif de l'ensemble des solutions possibles S est obtenu par :

$$Card(S) = \frac{(m \times r)^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$$

Tout comme pour le codage `osma`, une flexibilité partielle sur les machines produira potentiellement des solutions infaisables. L'instance exemple compte 77 760 solutions possibles dont 54 720 infaisables. La figure 2.11 présente une solution faisable avec ce codage.

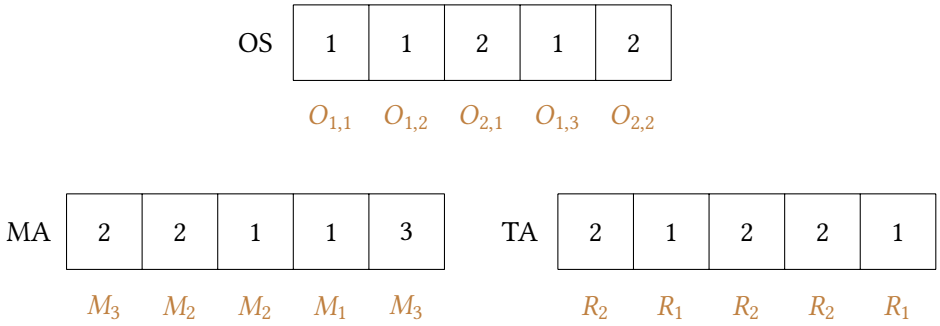


FIGURE 2.11 – Solution représentée avec le codage `osra`

2.2.1.12 Codage affectation des tâches de transport (`tta`)

Originellement utilisé par [Xu et al., 2022], le *codage affectation des tâches de transport*, noté `tta`, est une représentation de solution à trois volets avec des réels entre 0 et 1. Une solution est composée d'un vecteur de séquence de tâches de transport (TS), d'un vecteur d'affectation des machines (MA) et d'un vecteur d'affectation des ressources de transport (TA). Ce codage suppose qu'une tâche de transport $T_{i,j}$ induit l'opération $O_{i,j}$ correspondante. Pour décoder les éléments de chaque vecteur, la fonction $I(x) = \lfloor x \times y + 1 \rfloor$ est appliquée, où $\lfloor \rfloor$ est la fonction "floor" (prend un réel en entrée et donne en sortie le plus grand entier inférieur ou égal à), x est l'élément et y le nombre d'éléments disponibles de $I(x)$. Pour le vecteur TS, $I(x)$ donne un indice de job en considérant les jobs dont le traitement n'est pas terminé. Pour le vecteur MA, $I(x)$ détermine un indice de machine en fonction des machines candidates pour l'opération correspondante du vecteur TS. Pour le vecteur TA, il s'agit de l'indice d'une ressource de transport. En raison de l'usage de réels dans la constitution de solutions avec ce codage, il existe une infinité de solutions possibles. Toutefois, le décodage permet de réduire le nombre de solutions exactement à celui des solutions faisables de `osra`.

La figure 2.12 montre une solution avec `tta`. Le premier élément du vecteur TS, $I(0.23) = \lfloor 0.23 \times 2 + 1 \rfloor = 1$, conduit à $J_1 \longrightarrow T_{1,1}$. Après $0.14 \longrightarrow J_1 \longrightarrow T_{1,3}$, toutes les opérations de J_1 sont terminées. Il ne reste donc que J_2 dans l'ensemble des jobs disponibles à planifier, de sorte que $I(0.35) = \lfloor 0.35 \times 1 + 1 \rfloor = 1$ conduit à $J_2 \longrightarrow T_{2,2}$.

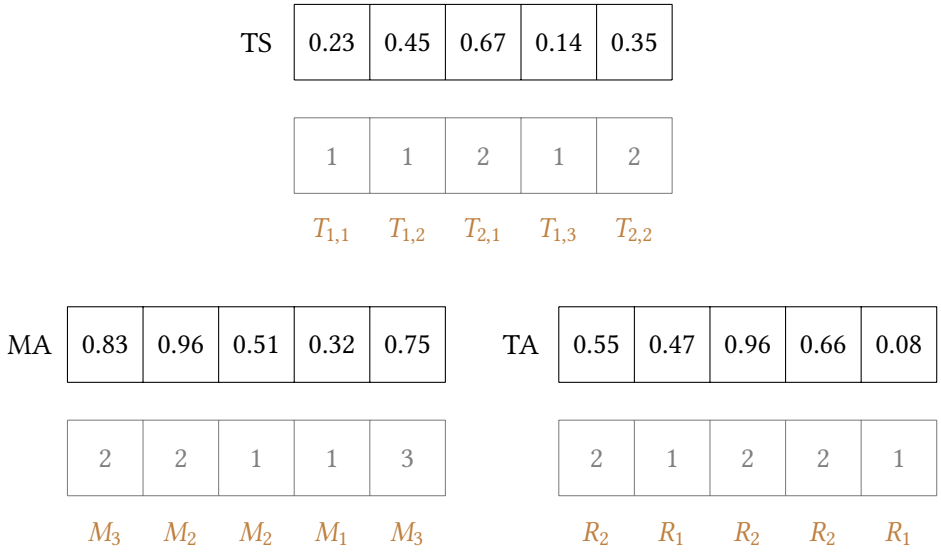


Figure 2.12 – Solution représentée avec le codage `tta`

2.2.1.13 Codage affectation de ressources et séquence d'opérations job-transport (`rajts`)

Le *codage affectation de ressources et séquence d'opérations job-transport*, noté `rajts`, dont s'est servi [Nouri et al., 2016b], répond aussi aux quatre sous-problèmes du FJSPT, avec cette fois un codage explicite des affectations et des séquencements. À l'aide de 3 vecteurs, cette représentation encode l'affectation des machines (MA), l'affectation des ressources de transport (TA) et la séquence des opérations des jobs couplées avec les tâches de transport (OTS). Le vecteur OTS se compose de $2 \times n$ numéros de job désignant alternativement $T_{i,j}$ et $O_{i,j}$. La première occurrence de chaque numéro de job est associée à une tâche

de transport, la suivante à une opération, et le schéma se répète, passant successivement d'une tâche de transport à une opération. Les vecteurs MA et TA sont similaires à ceux utilisés par `osra`. Une solution avec `rajts` a une taille de $4 \times \sum_{i=1}^{n} N_i$. L'effectif de l'ensemble des solutions possibles S et celui des solutions infaisables S_{inf} sont obtenus comme suit :

$$Card(S) = \frac{(m \times r)^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} 2N_i)!}{\prod_{i=1}^{n} 2N_i!}$$

$$Card(S_{inf}) = \frac{(m^{\sum_{i=1}^{n} N_i} - \prod_{i=1}^{n} \prod_{j=1}^{N_i} m_{i,j}) \times r^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} 2N_i)!}{\prod_{i=1}^{n} 2N_i!}$$

A l'égard de l'instance exemple, sur 1 632 960 solutions possibles, seulement 483 840 sont faisables. La figure 2.13 présente l'une de ses solutions faisables. Les 6 premiers éléments de la solution (1–2–1–1–2–1) sont ordonnancés de la manière suivante : $(T_{1,1}, R_2) \rightarrow (T_{2,1}, R_1) \rightarrow (O_{1,1}, M_3) \rightarrow (T_{1,2}, R_2) \rightarrow (O_{2,1}, M_3) \rightarrow (O_{1,2}, M_1)$.

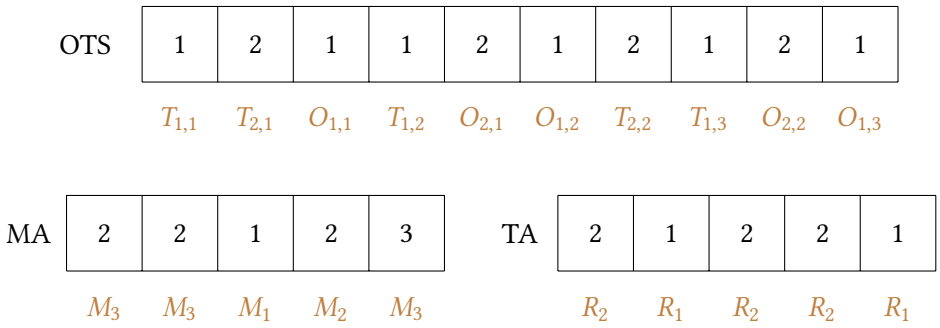


FIGURE 2.13 – Solution représentée avec le codage `rajts`

2.2.1.14 Synthèse des codages présentés

La liste des codages présentés ci-dessus et dont la synthèse est proposée dans les tableaux 2.5 et 2.6 n'est pas exhaustive. Le tableau 2.5 récapitule les informations encapsulées par chaque codage en termes de séquencement d'opérations (OS), d'affectation des machines (MA), de séquencement des tâches de transport (TS), d'affectation de ressources de transport (TA) et de présence de solutions infaisables (Inf.). Il inclut également les références des travaux intégrant ces divers codages. En complément, le tableau 2.6 rappelle les formules de calcul du nombre de solutions possibles pour chaque codage. Pour l'instance illustrative utilisée tout au long de la présentation des codages, il présente le nombre de solutions possibles ainsi que le nombre de solutions infaisables.

Il existe d'autres représentations de solutions pouvant être adaptées aux problèmes d'ordonnancement de la classe job shop. Le codage binaire de [Nakano and Yamada, 1991], le codage par temps d'exécution des opérations de [Yamada and Nakano, 1992], les codages job-machine de [Bierwirth et al., 1996] et de [Lei and Xiong, 2008], le codage à base de nombres réels de [Durasević and Jakobović, 2016] et le codage factoriel de [Zhao et al., 2019] en sont quelques exemples.

Tableau 2.5 – Synthèse des caractéristiques des codages

Codage	OS	MA	TS	TA	Inf.	Références
`job`	Oui	Non	Non	Non	Non	[Bierwirth, 1995; Afsar et al., 2016; Salido et al., 2016; Liao and Wang, 2019]
`ope`	Oui	Non	Non	Non	Oui	[Kumar et al., 2011; Lei et al., 2017; Li and Lei, 2021]
`mch`	Oui	Oui	Non	Non	Oui	[van Laarhoven et al., 1992; Vallada and Ruiz, 2011]
`mtx`	Oui	Oui	Non	Non	Oui	[Balin, 2011; Momenikorbekandi and Abbod, 2023]
`rkey`	Oui	Oui	Non	Non	Oui	[Bean, 1994; Ponsich and Coello, 2013]
`osma`	Oui	Oui	Non	Non	Oui	[Zhang et al., 2011; Karimi et al., 2017; Huang and Yang, 2019; Caldeira et al., 2020; Chen et al., 2020; Luo et al., 2020; Xu et al., 2021; Du et al., 2022]
`osmc`	Oui	Oui	Non	Non	Oui	[Carter and Ragsdale, 2006; Vlašic et al., 2020]
`osta`	Oui	Non	Non	Oui	Non	[Abderrahim et al., 2022]
`rosta`	Oui	Non	Non	Oui	Non	[Fontes et al., 2023]
`tja`	Non	Oui	Non	Oui	Oui	[Zhang et al., 2012]
`osra`	Oui	Oui	Non	Oui	Oui	[Liu et al., 2013; He et al., 2021; Homayouni and Fontes, 2021]
`tta`	Non	Oui	Oui	Oui	Non	[Xu et al., 2022]
`rajts`	Oui	Oui	Oui	Oui	Oui	[Nouri et al., 2016b]

2.2.1.15 Études comparatives de codages dans la littérature

Dans leur diversité, les représentations de solutions ont fait l'objet d'examen et de comparaison pour plusieurs problèmes d'atelier avec des objectifs plus ou moins variés. Pour le problème de job shop, déjà en 1996, [Cheng et al., 1996] ont recensé les codages utilisés au sein des algorithmes génétiques. L'étude a couvert une quinzaine de publications (1985-1995) à partir desquelles les neuf codages suivants ont été inventoriés.

- Représentation basée sur les opérations (`job` de [Bierwirth, 1995])
- Représentation basée sur les jobs (variante de [Holsapple et al., 1993])
- Représentation basée sur une liste de préférences (une adaptation de `mch` avec des séquences d'opérations non strictes sur les machines)
- Représentation basée sur les relations entre paires de job (variante de `mtx`)
- Représentation basée sur des règles de priorité
- Représentation basée sur un graphe disjonctif
- Représentation basée sur le temps d'achèvement
- Représentation basée sur la machine (une solution est codée sous la forme d'une séquence de machines dont le décodage requiert une heuristique de type *shifting bottleneck* [Adams et al., 1988])
- Représentation basée sur clé aléatoire (`rkey`)

TABLEAU 2.6 – Nombre de solutions générées par les différents codages avec un exemple pour l'instance des tableaux 2.3 et 2.4 ($n \times m \times r = 2 \times 3 \times 2$, $N_{i=1} = 3$ et $N_{i=2} = 2$)

Codage	Formule	Exemple	
		Possibles	Infaisables
`job`	$\frac{(\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$	10	0
`ope`	$\sum_{i=1}^{n} N_i!$	120	110
`mch`	$\prod_{k=1}^{m} N_k!$	574	340
`mtx`	$+\infty$	$+\infty$	–
`rkey`	$+\infty$	$+\infty$	–
`osma`	$\frac{m^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$	2 430	1 710
`osmc`	$\frac{(\sum_{i=1}^{n} N_i)! \times (\sum_{i=1}^{n} N_i - 1)!}{\prod_{i=1}^{n} N_i! \times (m-1)! \times (\sum_{i=1}^{n} N_i - m)!}$	60	39
`osta`	$\frac{r^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$	320	0
`rosta`	$+\infty$	$+\infty$	0
`tja`	$(m \times r)^{\sum_{i=1}^{n} N_i}$	7 776	5 472
`osra`	$\frac{(m \times r)^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} N_i)!}{\prod_{i=1}^{n} N_i!}$	77 760	54 720
`tta`	$+\infty$	$+\infty$	0
`rajts`	$\frac{(m \times r)^{\sum_{i=1}^{n} N_i} \times (\sum_{i=1}^{n} 2N_i)!}{\prod_{i=1}^{n} 2N_i!}$	1 632 960	1 149 120

Ces codages ont été comparés à travers le prisme de quatre propriétés : la propriété lamarckienne des chromosomes, la complexité du décodeur, le mappage entre l'espace des solutions codées et l'espace des ordonnancements, les exigences en matière de mémoire. En perspective, les auteurs ont souligné l'importance d'analyser ces codages dans des conditions expérimentales standardisées, afin d'établir un jugement juste à leur sujet. Suivant ces recommandations, [Ponnambalam et al., 2001] ont procédé à une étude comparative des temps CPU d'exécution et des makespan minimaux de quatre des codages présentés dans [Cheng et al., 1996]. Des algorithmes génétiques basés sur ces codages ont été confrontés dans le cadre du JSP. Le codage par liste de préférence est arrivé en tête en termes de temps CPU, tandis qu'à l'égard de la minimisation du makespan, c'est plutôt le codage `job` de [Bierwirth, 1995]. Les deux autres sont le codage job de [Holsapple et al., 1993] et le codage basé sur les règles de priorité. Il est à noter qu'il existe des disparités dans les moteurs d'ordonnancement selon le codage utilisé. Par exemple, le codage par liste de préférence ne fixe pas un ordre impératif des opérations sur les machines comme sous-entendu

pour le codage par opérations. Quant au codage job de [Holsapple et al., 1993], les intervalles vides sur les machines sont mis à profit pour l'obtention d'ordonnancements actifs. [Abdelmaguid, 2010] a également examiné, pour le JSP, six des représentations de solutions préalablement introduites par [Cheng et al., 1996]. Cette évaluation s'est basée sur deux métriques principales : l'écart d'optimalité moyen et le temps CPU moyen divisé par le nombre d'opérations. Les résultats de cette étude ont révélé que la représentation basée sur les machines, qui implique l'utilisation d'une heuristique de décodage, a présenté le plus faible écart d'optimalité en moyenne. Cependant, elle a également nécessité le temps CPU le plus élevé par rapport aux autres représentations. Par ailleurs, les représentations basées sur clé aléatoire et sur liste de préférences se sont avérées moins performantes que les autres représentations. À l'opposé des deux études comparatives précédentes sur les codages répertoriés par [Cheng et al., 1996] pour le job shop, [Jorapur et al., 2014] ont mis en évidence de meilleures performances pour le codage job de [Holsapple et al., 1993] lors de l'utilisation d'un algorithme génétique.

La variation des résultats entre les trois études peut en partie s'expliquer par les différences dans les instances considérées lors des expérimentations. Dans l'étude de [Ponnambalam et al., 2001], 20 instances carrées ont été utilisées, tandis que [Abdelmaguid, 2010] a employé 40 instances rectangulaires et carrées, et [Jorapur et al., 2014] ont eu recours à 68 instances rectangulaires et carrées. De plus, bien que les opérateurs de recherche aient été énumérés dans les travaux, les modalités de couplage avec les différents codages n'ont pas été explicitées. Par exemple, les opérateurs de croisement peuvent varier d'une étude à l'autre, de même que la politique régissant le choix des opérateurs (sélection, croisement, mutation). À cet égard, [Abdelmaguid, 2010] ont explicitement mentionné que, étant donné que la caractérisation des opérateurs n'était pas incluse dans leur étude, l'algorithme sélectionnait aléatoirement un opérateur, avec une probabilité égale pour l'ensemble des opérateurs. Sachant que les opérateurs de recherche influent également sur la structure des espaces de recherche, les variations de performance observées dans les trois études peuvent également être dues aux opérateurs de recherche employés.

Dans l'article de [Şahman and Korkmaz, 2022], une étude comparative de trois codages (clé aléatoire (`rkey`), codage réel de [Tasgetiren et al., 2004] utilisant la méthode *smallest position value* (partie OS de `rosta`), *ranked-over value*) a été entreprise. Cette analyse a impliqué l'utilisation de huit algorithmes d'optimisation à base de population pour résoudre le JSP. Sur les 48 instances classiques testées, les meilleurs résultats en termes de valeur de makespan ont été obtenus en utilisant le codage réel de [Tasgetiren et al., 2004] (qui est une adaptation d'un codage indirect initialement proposé pour les algorithmes d'optimisation continue). Ces résultats concernent la variante de l'algorithme d'algues artificielles (AAA) introduite par les auteurs. Étant donné que les opérateurs de recherche utilisés sont typiquement des concepts issus de l'optimisation continue, il devient difficile de comparer ces résultats avec ceux énoncés précédemment.

Les analyses comparatives de codages ont également porté sur les variantes du job shop de base. En utilisant un algorithme génétique, le codage `job` de [Bierwirth, 1995] a été confronté au codage job de [Holsapple et al., 1993] et au codage par clé aléatoire dans le contexte d'un job shop flexible distribué (DFJSP). Deux règles ont été introduites pour l'affectation des cellules et des machines. L'efficacité de l'algorithme utilisant le codage `job` de [Bierwirth, 1995] a permis aux auteurs de souligner l'importance de l'utilisation de codages appropriés lors de la conception des métaheuristiques. Dans une étude préliminaire portant sur un JSP combiné avec des robots collaboratifs, [Kinast et al., 2022] ont développé un algorithme génétique hybride pour comparer deux codages de solutions. Ces codages comprenaient un codage basé sur des nombres entiers et un codage basé sur des clés aléatoires biaisées. Testés sur une instance réelle, les résultats obtenus avec le codage basé sur clés aléatoires biaisées se sont avérés être meilleurs de 9.7% par

rapport à ceux obtenus avec le codage basé sur des nombres entiers. [Xuewen et al., 2020] ont également investigué cinq types de codage avec un algorithme génétique pour le problème de job shop flexible. Les codages étudiés sont nommément le *MSOS-I/MSOS-II* (`osma` dans le cadre de ce document), le *triple encoding* de [Kacem, 2003] (`ope` combiné à un vecteur d'affectation de machine), le *table representation* et le *matrix representation* (qui sont tous les deux des formes de `mtx`) et enfin le *A/B string* où la couche *A* traite l'affectation machine et la couche *B* correspond au `mch`. Au regard de la simplicité de la structure chromosomique, de la faisabilité des solutions et de l'espace de stockage, le codage *MSOS-I* a été présenté comme la meilleure alternative pour le FJSP. Toutefois, la faisabilité des solutions avec le *MSOS-I* n'est garantie qu'en cas de flexibilité totale ou avec des opérateurs de recherche améliorés. Le *A/B string* quant à lui présente une redondance d'information dans sa structure. Les couches *A* et *B* encodent tous deux l'affectation machine. Cette situation, source potentielle de conflits, nécessite l'incorporation de décisions supplémentaires dans le moteur d'ordonnancement ou dans les opérateurs de recherche afin d'aider à arbitrer l'affectation des machines.

En dehors du job shop, des comparaisons de codages ont été menées pour d'autres problèmes d'ordonnancement d'atelier, tels que les problèmes de flow shop [Fernandez-Viagas et al., 2019], d'open shop [Tsujimura et al., 1997], et de machines non-reliées [Durasević and Jakobović, 2016; Vlašic et al., 2020]. Comme dans les travaux précédents, ces comparaisons se concentrent généralement sur deux aspects principaux. D'une part, l'évaluation est basée sur la qualité des solutions produites, mesurée en termes de makespan et de temps CPU. D'autre part, elle repose sur des caractéristiques intrinsèques des codages, telles que l'espace de solutions codées, la faisabilité des solutions, la complexité de décodage, ou encore la compatibilité avec des opérateurs de recherche existants. Cependant, certains travaux ont également pris en compte d'autres éléments. Par exemple, [Durasević and Jakobović, 2016], en plus du makespan, ont comparé le temps total d'écoulement *(total flowtime)*, le total des retards pondérés *(total weighted tardiness)*, et le nombre pondéré des jobs en retard. Pour leur part, [Fernandez-Viagas et al., 2019] ont mis l'accent sur la définition d'un éventuel lien entre les espaces de solutions générés par les codages et la qualité des ordonnancements produits. Une de leurs contributions majeures révèle que, pour le problème étudié, des solutions de bonne qualité peuvent être obtenues en utilisant des codages qui n'explorent qu'une petite partie de l'espace complet des ordonnancements semi-actifs.

De ces études comparatives de codages pour le job shop classique, émerge des résultats divergents, explicables par les différences de conditions expérimentales, telles que les instances testées ou les opérateurs de recherche utilisés. En conséquence, sur la base de ces travaux, il est difficile de désigner un codage comme étant le meilleur pour le JSP classique à partir de la littérature. En ce qui concerne la variante flexible du job shop (FJSP), on observe la réutilisation du codage `job` initialement conçu pour la variante de base. Toutefois, puisque `job` ne code que le séquencement des opérations, l'affectation de machine, indispensable dans le FJSP, est gérée par une règle d'affectation. De la même façon que pour la variante de base, l'usage et la contribution des opérateurs de recherche sont passés sous silence. Quant au FJSPT, à notre connaissance, aucune étude comparative des codages n'a encore été menée.

Étant établi que les opérateurs ont un impact sur la structure des espaces de recherche (voir section 1.1.6), il devient nécessaire d'étendre la caractérisation des codages en les associant à des opérateurs. Aussi est-il important de standardiser les conditions d'expérimentation pour garantir une évaluation objective.

2.2.2 Opérateurs de voisinage

2.2.2.1 Choix des opérateurs

La notion de voisinage précédemment définie (définition 1.3) est représentée dans une métaheuristique à travers un ou plusieurs opérateurs de recherche appliqués au codage d'une solution. Indépendamment du paysage, le nombre de solutions voisines pour chaque solution est déterminé par le choix de l'opérateur de recherche. Ainsi, l'opérateur de recherche exerce une influence directe sur la densité du paysage, comme souligné par [Tari, 2019].

Dans la littérature des métaheuristiques, divers opérateurs de recherche plus ou moins sophistiqués ont été introduits pour les problèmes d'ordonnancement d'ateliers. Par exemple, [Abdelmaguid, 2015] a développé une fonction de voisinage randomisée et gloutonne pour le problème de job shop flexible avec des temps de préparation dépendant de la séquence. Cette fonction se caractérise par le retrait et l'insertion à une nouvelle position d'une opération sur le chemin critique. En parallèle, [Shen et al., 2018] ont proposé un algorithme de recherche taboue intégrant quatre variantes de l'opérateur d'insertion d'opérations critiques appliqués sur un graphe disjonctif pour le FJSP. De même, [Wang et al., 2021] ont développé deux types de structures de voisinage pour les opérateurs de recherche locale dans un contexte de recherche taboue, spécifiquement conçues pour un problème d'ordonnancement de type job shop sous incertitude avec deux fonctions objectifs. La première correspond à l'opérateur d'échange d'opérations sur le chemin critique de [Nowicki and Smutnicki, 1996], tandis que la seconde en est une adaptation.

L'ensemble de ces contributions témoigne d'un effort continu visant à développer des opérateurs de recherche plus performants pour les problèmes d'ordonnancement d'ateliers. En examinant les métaheuristiques présentées dans la section 2.1, on constate une prédominance des opérateurs de voisinage d'échange d'opérations sur le chemin critique et d'échange d'opérations quelconques pour le job shop classique, tandis que pour les FJSPT, les opérateurs de voisinage les plus fréquents incluent ceux utilisés couramment pour le JSP, ainsi que l'insertion, l'inversion et la réaffectation.

La diversité des opérateurs de recherche disponibles reste plutôt limitée, car on observe que presque les mêmes types d'opérateurs, parfois avec quelques variations, sont couramment utilisés pour résoudre divers types de problèmes d'ordonnancement. Cette tendance se retrouve tant dans les algorithmes de recherche locale que dans les approches évolutionnaires. Il existe des études comparatives visant à évaluer l'efficacité de ces opérateurs de recherche. En l'occurrence, pour résoudre le problème de job shop flexible en utilisant une approche de recherche locale, [Mastrolilli and Gambardella, 2000] comparent deux fonctions de voisinage, basées sur l'insertion d'opérations sur le chemin critique en utilisant un graphe comme représentation de la solution. Les auteurs ont également proposé une recherche taboue pour évaluer l'efficacité de ces opérateurs. Leur expérience a permis de découvrir 120 nouvelles bornes supérieures ainsi que 116 solutions optimales pour les 221 instances testées. Dans l'étude de [Kuhpfahl and Bierwirth, 2016], une méthode basée sur l'utilisation de graphes disjonctifs a été introduite pour appréhender la structure générale d'une dizaine de voisinages, qui sont principalement des variantes (voire des combinaisons) de deux voisinages connus (notés dans la suite de la section `cas`, `ins`).

En tenant compte des opérateurs de voisinage fréquemment utilisés dans la résolution approchée des problèmes de type job shop ainsi que de la diversité des variantes proposées, nous choisissons d'examiner cinq opérateurs de voisinage dans leur forme classique dans la suite de cette étude.

2.2.2.2 Opérateur insertion (`ins`)

L'*opérateur insertion*, noté `ins`, retire un élément choisi au hasard d'une solution et l'insère à un indice différent choisi au hasard. Chaque élément situé entre l'ancien et le nouvel emplacement est décalé d'une place en direction de l'ancien emplacement. Cet opérateur de voisinage que l'on retrouve dans divers travaux comme [Schiavinotto and Stützle, 2007; Zhang et al., 2012; Zhao et al., 2019; Li et al., 2020] peut être appliqué à tout codage ayant la forme d'une permutation. Pour les codages composés de différentes couches (OS, TS, MA et/ou TA), l'opérateur `ins` peut être appliqué à n'importe quelle couche à conditions que celle-ci ait la forme d'une permutation. En revanche, pour les codages à l'instar de `mch` et `mtx`, une adaptation est nécessaire pour l'emploi de l'opérateur ; le cas échéant, l'opérateur de voisinage peut-être appliqué à une liste d'opérations choisie au hasard. La figure 2.14 montre une illustration de `ins` dans laquelle l'élément 5 est déplacé vers la deuxième position faisant décaler vers la droite les éléments 2,3 et 4 de leurs positions initiales.

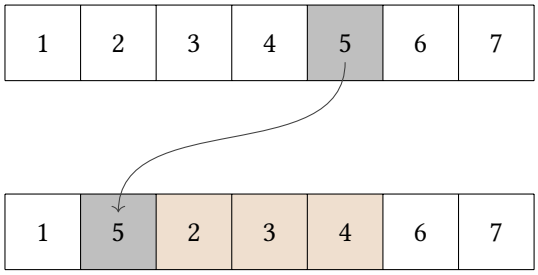


Figure 2.14 – Illustration de l'opérateur de voisinage `ins`

2.2.2.3 Opérateur échange (`swp`)

L'*opération échange*, noté `swp`, échange deux éléments choisis aléatoirement dans une solution de sorte d'avoir une nouvelle solution différente de l'initiale. Cet opérateur est fréquemment appliqué dans le domaine de l'optimisation en général, ainsi que dans le contexte des problèmes de type job shop, comme en attestent les travaux de [Peng et al., 2015; Nouri et al., 2016b; Yang et al., 2019; Fontes et al., 2023]. Les modalités d'application de cet opérateur aux divers codages sont identiques à celles de `ins`. La figure 2.15 schématise le procédé de `swp`.

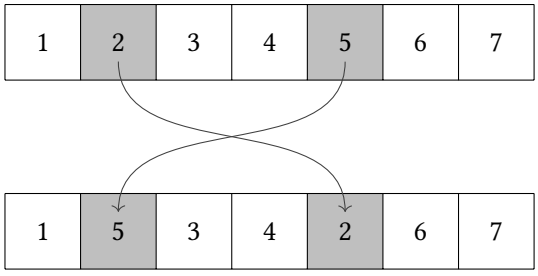


Figure 2.15 – Illustration de l'opérateur de voisinage `swp`

2.2.2.4 Opérateur inverse (`rev`)

L'*opérateur inverse* ou *inversion*, noté `rev`, sélectionne aléatoirement deux indices distincts dans une solution et inverse tous les éléments situés entre ces deux indices. Par endroit, cet opérateur est également appelé *2-opt* et fait partie de la famille des opérateurs *k-opt*, fréquemment utilisée dans des problèmes de tournées, notamment [Lin, 1965]. Toutefois, il est également employé dans le contexte des problèmes de type job shop, comme en témoignent les travaux de [Li et al., 2020; Peng et al., 2022; Abderrahim et al., 2022; Momenikorbekandi and Abbod, 2023]. L'opérateur `rev` peut être appliqué aux différents codages dans les mêmes conditions que `ins` et `swp`. Une illustration de `rev` est présentée à la figure 2.16.

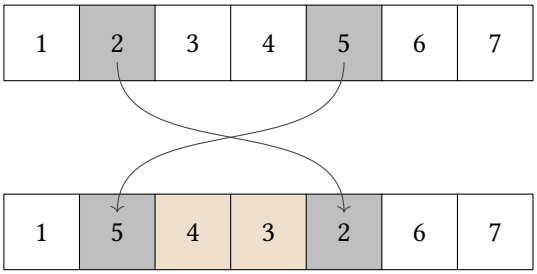


FIGURE 2.16 – Illustration de l'opérateur de voisinage `rev`

2.2.2.5 Opérateur échange adjacent (`adj`)

L'*opérateur échange adjacent*, noté `adj`, est un cas particulier des trois opérateurs `ins`, `swp` et `rev`. Il échange deux éléments adjacents choisis au hasard dans une solution [Anderson et al., 2003]. Cet opérateur de voisinage peut être appliqué aux codages dans les mêmes conditions que précédemment.

2.2.2.6 Opérateur échange adjacent critique (`cas`)

L'*opérateur échange adjacent critique*, noté `cas`, échange deux opérations adjacentes dans un bloc critique, c'est-à-dire deux opérations adjacentes qui se trouvent sur la même machine et sur un chemin critique de l'ordonnancement. Cet opérateur garantit que si une solution est faisable, ses voisines le sont également. À l'origine proposé par [van Laarhoven et al., 1992] pour le JSP, cet opérateur a connu plusieurs variantes au fil du temps, parmi lesquelles la variante de [Nowicki and Smutnicki, 1996] est l'une des plus connues. D'autres variantes existent, comme celles présentées dans les travaux de [Meeran and Morshed, 2012; Kuhpfahl and Bierwirth, 2016]. Dans le reste du document, `cas` fait référence à la variante de base introduite par [van Laarhoven et al., 1992]. Cet opérateur est pratique pour les codages `ope` et `mch`.

2.2.2.7 Synthèse sur les opérateurs de voisinage

Au sein des algorithmes de résolution, généralement constitués d'hybridations entre des métaheuristiques à base de population et des métaheuristiques de recherche locale, divers opérateurs de recherche, incluant des opérateurs de sélection, de recombinaison et de voisinage, interviennent alternativement pour converger vers la meilleure solution. Dans le cadre de notre étude bibliographique sur les récentes métaheuristiques

dédiées à la résolution de problèmes de type job shop, nous avons constaté que les opérateurs de voisinage `cas` et `swp` sont les plus fréquemment utilisés pour le JSP. En ce qui concerne le FJSPT, ce binôme est étendu pour inclure les opérateurs `ins`, `rev` et `réaffectation`.

Les études comparatives des opérateurs de voisinage dans la littérature s'appuie toujours sur un unique codage spécifique dont le choix a priori n'est pas justifié. De plus, les recommandations formulées à la suite de l'analyse des résultats ne prennent pas en compte la possible invalidité des conclusions lorsque d'autres représentations de solution sont utilisées. Comme discuté dans la section 2.2.1.15 sur les études comparatives des codages, il y a un réel intérêt à évaluer le couplage entre codage et opérateur de voisinage plutôt que de les traiter de manière indépendante. L'analyse du paysage fitness apparaît comme un outil potentiellement pertinent pour caractériser les couples codage et opérateur de voisinage, et nous présentons dans la suite de ce chapitre des travaux s'inscrivant dans ce contexte, notamment pour les problèmes d'ordonnancement.

2.3 Paysages de fitness et problèmes d'ordonnancement

Comme indiqué dans la section consacrée aux paysages du précédent chapitre (voir section 1.2), l'analyse d'un paysage revient à étudier la topologie d'un espace de recherche de solutions pour un problème donné. Elle permet d'acquérir une meilleure compréhension de la complexité inhérente aux problèmes d'optimisation et de clarifier le comportement des algorithmes d'optimisation à l'égard de ces problèmes. Il est important de noter qu'aucune propriété ne peut à elle seule fournir une description exhaustive d'un espace de recherche hétérogène de grande dimension. Cependant, une idée émerge du travail de [Vassilev et al., 2003], suggérant que la structure d'un paysage peut être pleinement appréhendée en considérant simultanément sa douceur *(smoothness)*, sa rugosité et sa neutralité. Cependant, comme argumenté par [Malan and Engelbrecht, 2013], même si cette approche peut s'avérer pertinente, il peut toujours être bénéfique de considérer un problème sous un angle différent afin d'éclairer sa nature sous-jacente. Par conséquent, d'autres propriétés du paysage, telles que la modalité, le bruit, l'évolutivité, la symétrie, et bien d'autres, ont été ciblées et évaluées à l'aide de diverses mesures largement discutées dans les études sur l'analyse de paysage de fitness.

À la suite des travaux de [Pitzer and Affenzeller, 2012] et de [Malan and Engelbrecht, 2013], qui ont proposé des métriques adaptées aux problèmes d'optimisation discrets, [Malan, 2021] a récemment introduit onze nouvelles métriques spécifiques à de nouveaux types de paysages émergents, notamment les paysages dynamiques. Parallèlement, [Zhu et al., 2022] ont évalué le degré de dépendance à la dimension du problème et à la taille de l'échantillon de 89 mesures de paysage de fitness, identifiant ainsi 39 mesures avec une faible dépendance. Bien que cette étude soit intéressante pour identifier les métriques applicables aux problèmes de grande dimension, il est à noter que les mesures utilisées ne peuvent être calculées que pour des problèmes d'optimisation continue, excluant ainsi certains types de problèmes tels que le nôtre.

Des analyses du paysage de fitness ont été réalisées pour diverses finalités, couvrant une gamme variée de problèmes d'optimisation. Cependant, en ce qui concerne le job shop, le nombre d'études sur ce sujet demeure limité. Dans leur analyse visant à caractériser les différences de paysage entre des instances de référence considérées comme difficiles et faciles pour le JSP, [Mattfeld et al., 1999] ont utilisé diverses mesures, telles que les distances entre solutions, les distributions d'entropie, la douceur (inverse de la rugosité) et la distance de corrélation. Ces mesures ont été appliquées pour évaluer les paysages générés à

l'aide d'une recherche locale. Pour les deux types d'instances, les optima locaux sont répartis sur l'ensemble du paysage de fitness. De plus, en ce qui concerne la distance de corrélation, les valeurs plus élevées observées pour les instances difficiles indiquent des paysages beaucoup plus plats, tandis que les instances faciles présentent une plus grande dispersion de pics dans le paysage. De là, certaines implications ont été déduites pour les opérateurs de recherche : pour que les opérateurs de voisinage fonctionnent efficacement, un certain degré de douceur du paysage est nécessaire. Les auteurs en ont conclu qu'une recherche avec les opérateurs de voisinage est plus favorable pour les instances difficiles du JSP qu'une recherche avec des opérateurs de recombinaison et de sélection.

D'autre part, [Bierwirth et al., 2004] ont analysé le paysage généré par 10 000 solutions de l'instances JSP de taille 10×10 de [Fisher and Thompson, 1963] (`ft10`). Ils ont opté pour une représentation de solution basée sur un graphe disjonctif, accompagnée d'un voisinage qui se base sur l'opérateur d'échange adjacent (`adj`). L'analyse a révélé que la connectivité des solutions varie le long de l'espace de recherche, créant ainsi un paysage irrégulier. Cependant, ils ont observé que les solutions de bonne qualité tendent à avoir des niveaux de connectivité élevés. Par conséquent, ils ont formulé l'hypothèse selon laquelle cette irrégularité dans le paysage pourrait en fait faciliter la recherche de solutions de bonne qualité par des méthodes basées sur des marches aléatoires et des algorithmes de recherche locale stochastique. Autrement dit, cette irrégularité pourrait orienter la recherche vers les régions du paysage contenant des solutions optimales.

Dans une étude menée par [Streeter and Smith, 2006], l'influence du rapport entre le nombre de jobs (n) et le nombre de machines (m) sur le paysage d'instances de JSP générées aléatoirement a été examinée. Pour les 11 combinaisons de $\frac{n}{m}$, 100 petites instances et 1 000 grandes instances ont été générées. L'algorithme de [van Laarhoven et al., 1992] a été utilisé pour obtenir 100 valeurs de makespan pour chacune des petites instances et 2 valeurs pour chacune des grandes instances. Ensuite, pour chaque makespan obtenu, un facteur ρ, qui représente l'écart le séparant du makespan optimal de l'instance, est calculé. Enfin, des clusters sont définis sur la base d'une estimation de la distance entre les solutions ayant un facteur ρ donné. Les résultats ont révélé que, pour de faibles valeurs du ratio $\frac{n}{m}$, les ordonnancements ayant un faible makespan sont concentrés dans un cluster dans une petite zone de l'espace de recherche et partagent de nombreuses caractéristiques communes. Cependant, cette tendance ne se manifeste pas pour des valeurs élevées du ratio. À partir de ces résultats, les auteurs ont suggéré deux règles, l'une concernant les faibles $\frac{n}{m}$ (≈ 1) et l'autre concernant les valeurs élevées du ratio (≥ 3). Respectivement, dans ces deux cas, la première règle stipule que l'algorithme doit tenter de localiser le groupe (cluster) unique d'optima globaux et de l'exploiter, tandis que la seconde règle suggère que l'algorithme doit tenter d'isoler un ou plusieurs groupes d'optima globaux et de traiter chacun d'entre eux séparément. Bien qu'intéressantes, ces règles semblent difficiles à implémenter car les groupes d'optima globaux ne peuvent être connus avant le début de la recherche. De plus, l'étude de [Zhang, 2004] a permis à Streeter M. et Smith S. de constater la violation de leur deuxième règle.

Ces travaux mettent en évidence, tout d'abord, que les optima locaux sont dispersés sur l'ensemble du paysage d'une instance de JSP, quelle que soit sa taille. En outre, l'étude menée par [Streeter and Smith, 2006] souligne que les instances de JSP présentant un ratio $\frac{n}{m} \approx 1$ tendent à regrouper des solutions avec un faible makespan dans une petite région de leurs paysages, ce qui est en accord avec les observations d'une précédente étude de [Bierwirth et al., 2004] sur l'instance `ft10`. Il convient toutefois de noter que les structures de voisinage utilisées dans ces deux travaux, notamment `adj` et `cas`, sont relativement proches en comparaison avec les voisinages `ins` et `rev`. De plus, la représentation de la solution adoptée dans ces

deux études repose sur un graphe disjonctif. Étant donné que les opérateurs de voisinage influent sur la configuration des paysages de fitness concomitamment avec les représentations de solutions, il est probable que ces résultats ne soient pas généralisables à d'autres choix de représentation de solution et de voisinage.

En dehors du job shop, l'analyse du paysage de fitness a été appliquée à divers autres problèmes d'optimisation. Dans l'étude menée par [Czogalla and Fink, 2012], le paysage de fitness du problème d'ordonnancement flow-shop sans attente (CFSP) a été analysé en examinant la rugosité du paysage et la relation entre la qualité d'une solution et sa distance avec une solution optimale. Cette analyse s'est appuyée sur des mesures telles que la distance de corrélation de fitness et la distribution des types de points. Les expériences ont été menées en utilisant les opérateurs de voisinage d'échange (swap) et de décalage (shift) Les résultats ont confirmé la présence d'une structure en forme de *grande vallée*, typique à celle que l'on trouve dans les problèmes d'optimisation combinatoire telle que l'affectation quadratique. Le paysage de fitness du CFSP a semblé être favorable à l'utilisation de méthodes de recherche locale, mais les résultats également suggèrent que les algorithmes évolutionnaires devraient être capables de repérer des solutions de qualité pour le CFSP. Pour le même problème, [Zhao et al., 2019] a également procédé à l'analyse de paysage de fitness avec quasiment les mêmes métriques mais en une utilisant une réprésentation de solution dite factorielle. Leur analyse a permis de vérifier la présence d'une structure caractérisée par plusieurs grandes vallées dans le paysage de fitness. Cette conclusion repose sur l'observation du graphique de distance entre les solution et sur une analyse fondée sur la théorie du codage factoriel. Ils ont également mis en évidence la diversité des optima locaux et la complexité du paysage de fitness grâce à des résultats statistiques issus des distributions de types de points. Enfin, cette étude a débouché sur des recommandations à prendre en compte lors de la conception d'un algorithme évolutif utilisant le codage factoriel pour la résolution de ce type de problème.

Sur le problème de parité booléenne, [Collard et al., 2006] ont exploré de nouvelles caractéristiques liées à la neutralité dans les paysages de fitness. Leur étude a comparé les paysages de fitness générés par deux ensembles d'opérateurs, à savoir `NAND` et `XOR, NOT`. Parmi les constatations notables, il est ressorti que l'utilisation de l'ensemble d'opérateurs `NAND` semblait favoriser une plus grande neutralité au sein des réseaux ayant de mauvaises valeurs de fitness par rapport à ceux ayant de bonnes valeurs de fitness. Cette étude a éclairé les raisons pour lesquelles la programmation génétique semble être plus efficace pour résoudre le problème de parité paire lorsque les opérateurs `XOR, NOT` sont employés, tandis qu'elle rencontre davantage de difficultés avec l'ensemble d'opérateurs `NAND`. [Kovács et al., 2020] ont travaillé sur une approche visant à exploiter le paysage de fitness dans le contexte d'un problème complexe de routage de véhicules, également dans le but de comparer des opérateurs de recherche. Cette analyse essentiellement basée sur des notions de distance a comparé les opérateurs suivants : 2-opt, croisement par ordre, croisement partiellement appairé et croisement en cycle. Les résultats ont indiqué que l'opérateur 2-opt était le plus performant. Sur la base de cette analyse du paysage de fitness, une nouvelle variante d'algorithme génétique pour le problème du voyageur de commerce a été proposée.

[Verel et al., 2007] ont conduit une étude portant sur les automates cellulaires chargés de réaliser la tâche de la majorité computationnelle qui est un problème de calcul distribué. Ils se sont penchés sur les raisons qui rendent le paysage de fitness de ce problème particulièrement complexe. Grâce à une quantification statistique des caractéristiques du paysage, ils ont identifié un sous-espace significatif baptisé *Olympe* au sein du paysage du problème de la majorité computationnelle, en exploitant les similitudes entre ces automates cellulaires et les structures symétriques présentes dans le paysage global. Dans ce sous-espace, ils ont examiné la dynamique et les performances de trois algorithmes génétiques afin de confirmer

les conclusions tirées de l'analyse du paysage et de trouver des automates cellulaires efficaces pour la résolution de la tâche de la majorité computationnelle, tout en minimisant la charge computationnelle.

[Pavelski et al., 2019] se sont intéressés à l'application de l'apprentissage méta *(meta-learning)* pour suggérer des stratégies de recherche locale destinées à résoudre plusieurs instances du problème de flow shop de permutation. Dans cette recherche, les caractéristiques des instances ont principalement été extraites par le biais d'une analyse du paysage de fitness. En plus des caractéristiques fondamentales du problème, telles que le détail des instances, l'objectif et les critères d'arrêt, les auteurs ont enrichi les métadonnées en intégrant des mesures de l'analyse du paysage de fitness. Un total de vingt-neuf mesures de paysage a été utilisé, comprenant huit mesures basées sur des marches aléatoires (telles que l'autocorrélation, l'entropie, la densité du bassin, etc.) et vingt-et-une basées sur des marches adaptatives (comme les corrélations entre fitness et sa distance par rapport à l'optimum), dans le but de fournir des informations supplémentaires lors de la phase de sélection des recommandations.

Plusieurs éléments en termes de méthodologie peuvent être tirés de ces travaux. Tout d'abord, une approche consiste à analyser les paysage de fitness afin de formuler des recommandations pour la conception d'algorithmes d'optimisation, comme illustré par les travaux de [Czogalla and Fink, 2012], [Zhao et al., 2019], et [Kovács et al., 2020]. [Czogalla and Fink, 2012] a aussi étendu le répertoire des métriques d'analyse du paysage en introduisant la distribution des types de points. Une autre approche consiste à expliquer les performances des composants d'un algorithme à travers l'analyse du paysage de fitness, comme démontré par [Collard et al., 2006]. Ensuite, la démarche consistant à confronter les résultats d'une analyse de paysage (non orientée par un algorithme) aux performances réelles d'un algorithme [Verel et al., 2007]. Une extension a été apportée par [Pavelski et al., 2019], qui a proposé un modèle d'apprentissage permettant d'exploiter les caractéristiques de paysage dans un algorithme d'optimisation. Néanmoins, nous estimons que sans une validation de la pertinence des caractéristiques à intégrer, le modèle risque de ne pas atteindre les objectifs escomptés.

Malgré l'intérêt grandissant pour l'analyse de paysage de fitness au cours des dernières décennies, il convient de noter qu'à l'exception du problème d'ordonnancement *flow shop* (FSP), la recherche dans ce domaine reste relativement limitée en ce qui concerne les problèmes d'ordonnancement d'atelier. Cette observation est confirmée par la revue récente menée par [Zou et al., 2022] sur l'analyse de paysage de fitness pour divers problèmes d'optimisation. De plus, en ce qui concerne l'étude du paysage de fitness, nous n'avons pas identifié, dans la littérature existante, d'études comparatives portant sur les différentes représentations de solutions associées à des opérateurs de voisinage pour le problème de job shop. Par conséquent, il est à la fois pertinent et novateur de mettre en lumière l'analyse du paysage de fitness dans ce contexte particulier pour ce problème d'ordonnancement.

2.4 Bilan

La résolution des problèmes d'optimisation d'atelier de production a connu une évolution marquée ces dernières décennies, caractérisée par l'utilisation de diverses métaheuristiques. Parmi celles-ci, l'algorithme génétique a suscité un grand intérêt dans le contexte des métaheuristiques à base de populations, tandis que des techniques telles que la recherche taboue et le recuit simulé ont été privilégiées pour les métaheuristiques de recherche locale. Cependant, il est à noter que malgré l'émergence de nombreuses méthodes dites "nouvelles", bon nombre d'entre elles ne sont en réalité que des adaptations des anciennes,

sans apport révolutionnaire majeur. Comme souligné par [Sörensen, 2015], il est essentiel de consolider les connaissances existantes en effectuant une analyse critique des composants des métaheuristiques.

Parmi les composants, les codages de solutions et les opérateurs de recherche sont identifiés comme des éléments cruciaux qui entrent dans la définition des espaces de recherche. Toutefois, force est de constater que de nombreuses recherches portant sur des métaheuristiques se concentrent davantage sur des hybridations de méthodes et des paramétrages, souvent en omettant de justifier le choix de leur codage de solutions. Selon les travaux exposés dans la section 2.1, et synthétisés dans le tableau 2.7, les codages les plus récurrents pour le JSP sont, par ordre de fréquence, `job` et `mch`, pour le JSPT, ce sont `osta` et `rosta`, pour le FJSP, on retrouve `osma`, et pour le FJSPT, les codages dominants sont `osma` et `osra`. Bien entendu, en plus des prédominants, d'autres codages sont également utilisés de manière sporadique dans la littérature pour ces différentes variantes du JSP. À titre d'exemple, `job` est employé à la fois pour le JSP classique et le JSPT, tout comme `osma` qui est utilisé pour le FJSP et le FJSPT, suggérant ainsi une certaine transversalité des codages entre les diverses variantes du problème. Cette observation est encore plus marquée en ce qui concerne les opérateurs de voisinage.

Le tableau 2.7 fournit une synthèse des codages et des opérateurs de voisinage associés les plus fréquemment rencontrés dans la littérature des métaheuristiques dédiées au JSP et à ses variantes. Une observation récurrente est la présence systématique de l'opérateur de voisinage `swp` dans le groupe des opérateurs fréquents pour l'ensemble des variantes du problème. Pour le FJSP et le FJSPT, un schéma d'association `osma`–`swp` se dégage de manière constante, auquel s'ajoutent d'autres opérateurs. Cela soulève la question de savoir s'il est plus avantageux de maintenir un codage efficace pour la variante de base avec transposition, plutôt que de développer des algorithmes métaheuristiques hautement spécifiques pour chaque variante. Il est nécessaire de noter que ces algorithmes spécifiques peuvent potentiellement masquer les faiblesses des codages et des opérateurs de voisinage qu'ils utilisent. Ainsi, notre démarche vise à formaliser et à confirmer, ou infirmer, la validité des schémas identifiés, plutôt que de se fier à des intuitions.

En plus de leurs propriétés intrinsèques, ces deux composants (codage et opérateur) peuvent être caractérisés en fonction des espaces de recherche qu'ils engendrent, ce qui peut être réalisé à travers des analyses de paysage de fitness. En raison de la notion de voisinage inhérente à la définition du paysage de fitness, l'analyse de paysage de fitness est syntaxiquement appropriée pour les méthodes de recherche locale. Dans ce sens, deux interrogations majeures émergent. Tout d'abord, est-ce que l'analyse du paysage de fitness peut conduire à des résultats ou des propriétés significatives, voire à des recommandations pour le choix judicieux de couples codage–voisinage lors de la conception de métaheuristiques pour le problème du job shop ? Ensuite, dans quelle mesure les résultats obtenus pour la variante de base peuvent-ils être généralisés à des variantes plus complexes, impliquant davantage de contraintes ?

Afin de répondre à ces interrogations, nous avons décidé d'appliquer cette démarche d'analyse à la fois pour le JSP classique et pour le FJSPT. Nous procéderons ensuite à une comparaison des résultats afin d'établir d'éventuels rapprochements. Le prochain chapitre est dédié à l'étude des couples de codage et d'opérateur de voisinage pour le JSP classique.

Tableau 2.7 – Codages et opérateurs de voisinage associés les plus courants dans la littérature

Variante	Sous-problèmes				Codages		Opérateurs de voisinage	
	OS	MA	TS	TA	Fréquents	Autres	Fréquents	Autres
JSP	×	–	–	–	`job`, `mch`	`rkey`, graphe, règles de priorité	`cas`, `swp`	décalage *(shift)*, mélange *(shuffle)*, adaptation `rev`, tirage aléatoire
JSPT	×	–	×	×	`osta`, `rosta`	`job`	`swp`, `rev`	graphe disjonctif
FJSP	×	×	–	–	`osma`	`mtx`	`ins`, `swp`, réaffectation *(reassign)*	`rev`, permutation d'éléments, sélection de vitesse *(speed)*, règles de répartition
FJSPT	×	×	×	×	`osma`, `osra`	`tja`, `tta`, `rajts`, graphe disjonctif, `ope`	réaffectation, `swp`, `ins`, `cas`, `rev`	mélange, permutation, `SPT`, sélection de vitesse

OS *: séquencement des opérations*, MA *: affectation des machines*, TS *: séquencement des tâches de transport*, TA *: affectation des ressources de transport*

× *: est un sous-problème de la variante*, – *: n'est pas un sous-problème de la variante*

Chapitre 3
Caractérisation de codages et voisinages pour le job shop classique

Ce chapitre est consacré à l'étude de codages en association avec des opérateurs de voisinage dans le contexte du job shop classique. L'influence des deux composants sur les espaces de recherche est analysée au moyen de métriques d'analyse de paysage de fitness. Ensuite, les liens possibles entre les résultats de l'analyse du paysage et les performances d'une méthode de recherche locale, telle que la recherche taboue, utilisant ces codages et opérateurs pour un grand nombre d'instances JSP, sont examinés.

Contenu

3.1 Introduction ... 66
3.2 Description du problème traité ... 66
3.3 Codages et opérateurs de voisinage ... 67
3.3.1 Nouvelle proposition de codage ... 67
3.3.2 Les codages et opérateurs de voisinage retenus ... 69
3.3.3 Moteurs d'ordonnancement ... 69
3.4 Utilisation des espaces de recherche ... 70
3.4.1 Marche aléatoire ... 70
3.4.2 Tirage aléatoire uniforme ... 71
3.4.3 Recherche locale ... 71
3.5 Instances JSP ... 74
3.6 Caractérisation des paysages pour le JSP ... 75
3.6.1 Distance de corrélation ... 76
3.6.2 Taux de neutralité ... 79
3.6.3 Probabilité d'échappement cumulée ... 80
3.6.4 Distribution des types de points ... 81
3.6.5 Synthèse sur la caractérisation des paysages ... 82
3.7 Performance d'une métaheuristique : recherche taboue ... 83
3.7.1 Résultats des tests ... 83
3.7.2 Synthèse sur les performances ... 88
3.8 Bilan ... 88

3.1 Introduction

Pour les problèmes d'optimisation en général, et d'ordonnancement en particulier, les artisans de métaheuristiques se focalisent plus sur l'hybridation de méthodes approchées à solution unique et à base de population, sur le choix des opérateurs de recherche et sur le paramétrage des éléments stochastiques tels que la valeur des probabilités de sélection, de croisement et de mutation (dans un algorithme génétique par exemple), le nombre d'itérations et la taille de population. De plus, les efforts se concentrent le plus souvent sur le développement d'opérateurs dédiés aux instances à résoudre, et beaucoup moins sur le choix du meilleur codage. D'ailleurs, le codage n'est même pas spécifié dans quelques-uns des articles de la littérature analysés au chapitre précédent. Or le codage est un composant déterminant pour la définition d'un espace de recherche (voir section 1.1.6 concernant les composants classiques des métaheuristiques), d'où notre contribution dans ce chapitre.

De manière plus détaillée, l'analyse de la littérature synthétisée au chapitre 2 a montré que :

1. peu d'études ont été effectuées concernant l'impact du choix de représentation des solutions sur l'efficacité des métaheuristiques utilisées pour résoudre des problèmes d'ordonnancement d'atelier, même si quelques tentatives existent pour leurs différentes classes ;
2. des points sont à améliorer sur ces études concernant notamment les conditions expérimentales (nombre d'instances, standardisation des moteurs d'ordonnancement) ;
3. peu de résultats probants sont sortis de ces études ;
4. l'impact du choix de codage conjointement avec le choix des opérateurs de voisinage n'a pas fait l'objet d'études poussées.

C'est ce qui nous a motivés à investiguer cet impact conjoint, dans le domaine d'optimisation des problèmes d'ordonnancement d'atelier. En raison de leur pertinence intemporelle, nous avons choisi comme champ d'application les problèmes de type job shop contraints, en particulier la variante flexible avec transport, qui nous semble représentative de nombreux problèmes industriels concrets. Néanmoins, compte tenu des observations concernant le manque de résultats de la littérature, il nous a semblé pertinent de commencer notre travail de recherche par l'étude de la variante de base du job shop, afin d'extraire si possible des schémas de codage-voisinage efficaces et de vérifier si ceux-ci peuvent être ensuite transposés à des variantes plus complexes.

3.2 Description du problème traité

Le problème d'ordonnancement de type job shop a été présenté dans la sous-section 1.1.4. Néanmoins, il est important d'apporter des précisions sur le contexte dans lequel le JSP est abordé dans ce chapitre. Ainsi, les hypothèses à considérer pour le job shop classique traité sont les suivantes :

- les jobs sont disponibles dès le début de l'ordonnancement ($\forall i \in [1, n], r_i = 0$) ;
- les temps de traitement $p_{i,j}$ des opérations sont connus à l'avance ;
- les temps de montage et de démontage sont inclus dans les temps de traitement ;
- les temps de transport sont négligeables (sans transport), donc les ressources (et contraintes associées) de transport ne sont pas prises en compte ;
- les produits peuvent attendre dans les stocks de capacité illimitée (sans contrainte de stock) ;
- chaque machine ne peut réaliser qu'une opération à la fois (ressource disjonctive) ;

— chaque opération est réalisée sans interruption (non-préemption) ;
— une opération ne peut être traitée que par une machine connue à l'avance (problème non flexible) ;
— un job ne peut être traité qu'au plus par une machine à la fois ;
— un job peut être traité sur la même machine plus d'une fois (avec réentrance).

La fonction objectif consiste à minimiser le temps total d'exécution de l'ordonnancement $C_{\max}$. Formellement, $C_{\max} = \max(C_1, C_2, \ldots, C_n)$, où C_i représente la date de complétion du job J_i. Ce critère est un des plus couramment utilisés, d'où notre choix (voir tableau 2.1)

3.3 Codages et opérateurs de voisinage

3.3.1 Nouvelle proposition de codage

Parmi les codages à tester, il nous semblait pertinent d'inclure a minima un codage lié au temps puisque l'ordonnancement fait intervenir des contraintes temporelles. Il existe, pour les problèmes d'ordonnancement, diverses formes de codage par temps comme celles de [Yamada and Nakano, 1992; Mouret et al., 2011; Stefansson et al., 2011]. Le codage de [Yamada and Nakano, 1992], par exemple, est un codage qui représente une solution par l'ensemble des dates de complétion des opérations. Ces dates de complétion sont obtenues à partir de l'algorithme de Giffler & Thompson.

Le *codage par temps* que nous proposons et appelons `tim` représente une solution par une liste de $\sum_{i=1}^{n} N_i$ temps. Chaque temps correspond à une opération $O_{i,j}$ et désigne le délai minimum entre le début de l'opération et la fin de l'opération précédente du même job (ou le début de l'ordonnancement pour la première opération). Pour construire l'ordonnancement d'une solution, dans un premier temps, les opérations sont disposées chronologiquement sur les machines en suivant les délais minimums spécifiés par la solution. Étant donné qu'il s'agit d'un délai minimum, si la date de début d'une opération x sur une machine coïncide avec la période d'exécution d'une opération y, le début de l'opération x est décalé à la fin de l'opération y. À l'inverse, si la période d'exécution de l'opération x coïncide avec la date de début d'une opération y, l'opération x conserve la priorité, et toutes les opérations qui suivent l'opération y, y compris elle-même, sont décalées par translation pour que l'opération y débute à la fin de l'opération x. Ainsi, il existe toujours un ordonnancement valide qui satisfait aux contraintes de délai avec ce codage. Cependant, l'ordonnancement obtenu n'est probablement pas semi-actif. Alors, la seconde étape consiste à déduire un ordonnancement semi-actif équivalent par l'élimination des créneaux horaires inactifs tout en respectant les contraintes de précédence des opérations. Ce codage par temps a un nombre infini de solutions possibles, qui sont toutes faisables.

$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$
2	3	8	10	1	3

FIGURE 3.1 – Solution représentée avec le codage `tim`

La figure 3.1 montre une solution représentée par le codage `tim` pour l'instance décrite ci-après :

Job 1 : Machine 1 (4) ⟶ Machine 2 (2) ⟶ Machine 3 (1)
Job 2 : Machine 2 (3) ⟶ Machine 3 (3) ⟶ Machine 1 (2)

La première phase de la construction de l'ordonnancement, décrite précédemment, génère le premier diagramme de Gantt illustré dans la figure 3.2. Conformément à la solution, $O_{2,1}$ doit respecter un écart minimum de 10 unités de temps par rapport au début de l'ordonnancement sur la machine M_2. Placer $O_{2,1}$ à la date 11 aurait pour conséquence d'interrompre $O_{1,2}$, mais étant donné que les opérations sont non-préemptives, $O_{2,1}$ est décalée pour être positionnée à la date 12, juste après $O_{1,2}$. Une fois toutes les opérations placées, la deuxième phase consiste à éliminer les créneaux horaires inactifs en ajustant à gauche les opérations. Cela donne lieu à l'ordonnancement semi-actif représenté par le deuxième diagramme de Gantt dans la figure 3.2.

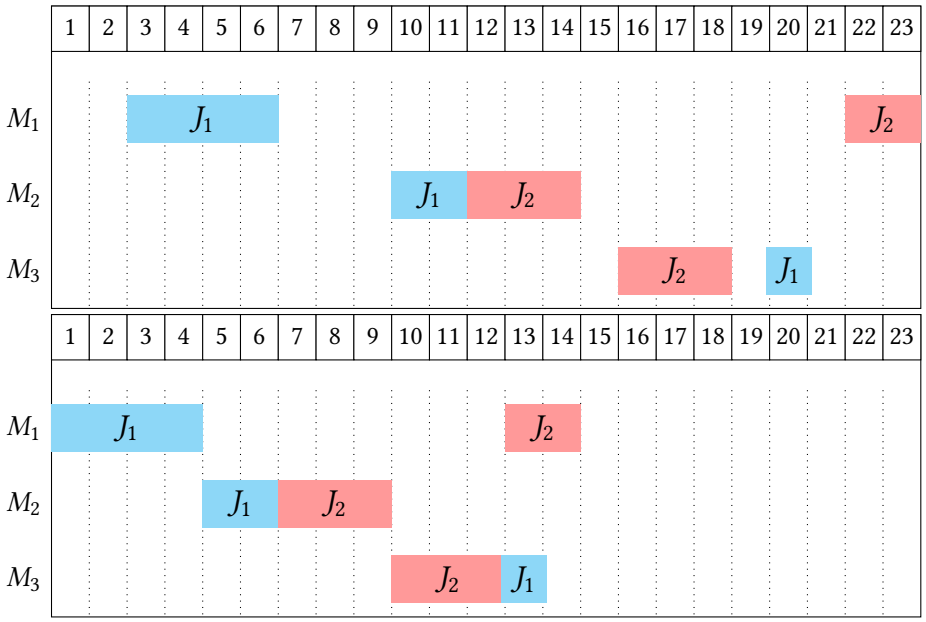


FIGURE 3.2 – Diagrammes de Gantt associés à l'exemple de solution avec le codage `tim` avant et après élimination des crénaux horaires inactifs

En raison de la particularité du codage par temps, nous lui avons spécifiquement associé un opérateur de voisinage portant le même nom `tim`. Avec cet opérateur, un sous-ensemble d'éléments est modifié (15% dans notre cas) de la manière suivante : un élément dont la valeur t peut être nulle est modifié par une nouvelle valeur choisie aléatoirement selon une distribution binomiale négative avec les paramètres $r = 4 \times \max(1, t)$, $p = 0.8$. Si t est différente de zéro, cette distribution assure une valeur moyenne égale à t mais une valeur médiane inférieure à la moyenne : la valeur des éléments tend à diminuer (ce qui devrait permettre d'obtenir un meilleur ordonnancement) mais parfois elle augmente beaucoup (pour obtenir des ordonnancements très différents).

3.3.2 Les codages et opérateurs de voisinage retenus

Dans le but d'examiner les codages en association avec les opérateurs de voisinage dans le contexte du problème de job shop, nous avons composé des couples codage-voisinage en nous basant sur l'analyse réalisée au chapitre 2. Les codages et les opérateurs de voisinage concernés sont indiqués dans le tableau 3.1. Comme discuté au chapitre 2 (voir le tableau 2.5) nous avons établi que seuls 3 des 13 codages recensés étaient directement adaptés à la variante de base de notre problème de job shop : `job`, `ope` et `mch`, auquel nous ajoutons `tim`. Notons que les codages `mtx` et `rkey`, bien qu'applicables pour le job shop classique, ne sont pas pris en compte dans nos expérimentations en raison de leur similitude structurelle et de leur convertibilité directe avec `mch`. Nous considérons dans la suite les 5 opérateurs de voisinage présentés au chapitre précédent : `adj`, `cas`, `ins`, `rev`, `swp`, auxquels nous ajoutons l'opérateur `tim`.

Tableau 3.1 – Codages et opérateurs de voisinages investigués pour le JSP

Codages	job (`job`) machine (`mch`) opération (`ope`) par temps (`tim`)
Voisinages	échange adjacent (`adj`) échange adjacent critique (`cas`) insertion (`ins`) inverse (`rev`) échange (`swp`) temps (`tim`)

Parmi les 24 combinaisons possibles des 4 codages et 6 opérateurs retenus, seules 15 ont été testées, nommément : `job-adj`, `job-ins`, `job-rev`, `job-swp`, `mch-adj`, `mch-cas`, `mch-ins`, `mch-rev`, `mch-swp`, `ope-adj`, `ope-cas`, `ope-ins`, `ope-rev`, `ope-swp`, `tim-tim`.

En effet, le voisinage `tim` ne s'applique qu'au codage `tim`, parce-qu'il ne peut pas s'appliquer à une permutation, contrairement aux autres opérateurs de voisinage. Par ailleurs, le couplage `job-cas` n'a pas de sens dans la mesure où `cas` s'applique à des opérations.

Il est à noter que, du fait que le codage `mch` est structuré en plusieurs couches de séquences d'opérations, l'application des opérateurs `adj`, `ins`, `rev`, et `swp` se fera sur l'une de ces séquences, sélectionnée de manière aléatoire.

3.3.3 Moteurs d'ordonnancement

Les moteurs d'ordonnancement sont des instruments qui permettent de transformer une solution, exprimée par un codage, en un ordonnancement. Afin d'appréhender véritablement les qualités des codages testés, nous utilisons des moteurs d'ordonnancement simples, minimisant autant que possible toute sophistication. Pour l'ensemble des codages, les moteurs construisent des ordonnancements semi-actifs à travers une approche gloutonne, en programmant les opérations au plus tôt sur les différentes machines.

Pour les codages `job` et `ope`, les opérations sont intégrées dans l'ordonnancement dans l'ordre de leur apparition dans la permutation (solution). Lorsqu'une opération $O_{i,j}$ est sélectionnée, le moteur identifie la machine M_k correspondante, et $O_{i,j}$ est positionnée à une date déterminée par la valeur maximale entre la date de complétion de la dernière opération sur M_k et la date de complétion C_i de la dernière opération de J_i. En ce qui concerne `job`, les opérations sont déduites de l'ordre d'apparition des numéros de jobs. C'est-à-dire, la première occurrence du numéro 1, par exemple, correspond à l'opération $O_{1,1}$, la deuxième à $O_{1,2}$, et ainsi de suite. Ce procédé garantit naturellement le respect des contraintes de précédence. En revanche, pour `ope`, la violation d'une contrainte de précédence dans la séquence des opérations conduit à l'infaisabilité de la solution.

Le moteur d'ordonnancement du codage `mch` traite les opérations prêtes à être programmées au plus tôt sur chaque machine. En d'autres termes, avec `mch` constitué d'une séquence d'opérations pour chaque machine, le moteur procède itérativement à la programmation de n'importe quelle opération éligible. À chaque itération, les opérations éligibles sont les premières opérations, non encore ordonnancées des séquences sur les machines, qui ne violent pas de contrainte de précédence. Jusqu'à l'épuisement des opérations, l'opération choisie à chaque itération est programmée au plus tôt sur la machine concernée, suivant le même principe que pour `job` et `ope`. Si des opérations restent non ordonnancées, mais qu'à une itération aucune opération n'est éligible, la solution est considérée comme infaisable. Le fonctionnement du moteur d'ordonnancement du codage `tim` a été précédemment expliqué dans la section 3.3.1.

Ces moteurs d'ordonnancement attribuent à chaque solution une valeur de fitness, qui dans notre contexte, correspond simplement au maximum des temps de complétion C_i des jobs J_i, i allant de 1 à n. Les solutions, ainsi déterminées par un codage spécifique et interconnectées par une notion de voisinage, constituent un espace de recherche.

3.4 Utilisation des espaces de recherche

Les indicateurs retenus pour les expérimentations appartiennent à deux catégories distinctes :

- l'analyse de paysage de fitness à travers les mesures de rugosité, de neutralité, d'évolutivité et de distribution de fitness ;
- l'évaluation des performances de convergence avec une métaheuristique de recherche locale.

Pour calculer ces indicateurs, nous avons utilisé trois méthodes pour explorer l'espace de recherche, pour chaque codage et ses opérateurs de voisinage, ainsi que pour chaque instance : marche aléatoire, sélection aléatoire, recherche locale. La phase d'expérimentation a nécessité plusieurs mois de temps CPU et généré un total de 17 Go de données brutes.

3.4.1 Marche aléatoire

Une marche aléatoire est un modèle mathématique qui décrit une séquence de pas aléatoires successifs le long d'un espace mathématique discret ou continu [Motwani and Raghavan, 1995]. Les marches aléatoires sont couramment utilisées pour modéliser des phénomènes aléatoires, des processus stochastiques, ou pour résoudre des problèmes d'optimisation. C'est un concept largement utilisé en mathématiques, en statistiques et en sciences de l'informatique. Dans notre cas, la marche se fait sur un ensemble discret de points (solutions). Chaque étape consiste à choisir de manière aléatoire parmi les voisins possibles du point actuel comme le montre notre algorithme 1.

Vingt marches aléatoires de 1 000 pas partant de 20 solutions faisables sont effectuées pour chaque couple codage-voisinage. Nous faisons l'hypothèse que les paysages sont statistiquement isotropes. À travers ces marches aléatoires, nous calculons la *distance de corrélation* (rugosité) et le *taux de neutralité* (neutralité).

Algorithme 1 Marche aléatoire

Entrée : `instance`, `codage`, fonction de voisinage $\mathcal{V}$, nombre d'itérations `nbItérations`
Sortie : liste de solutions S

```
1:  s ← GénérerSolutionInitiale(instance, codage)
2:  AjouterÀListe(S, s)
3:  pour i de 1 à nbItérations faire
4:      s' ← V(s)
5:      tant que EstInfaisable(s') faire
6:          s' ← V(s)
7:      fin tant que
8:      s ← s'
9:      AjouterÀListe(S, s)
10: fin pour
11: retourne S
```

3.4.2 Tirage aléatoire uniforme

Le tirage aléatoire uniforme, également appelée échantillonnage aléatoire, fait référence à une stratégie où une ou plusieurs solutions candidates sont choisies de manière uniforme parmi un ensemble de solutions possibles à un problème donné [Hoos and Stützle, 2005]. Cette approche est souvent utilisée pour explorer l'espace de recherche de manière aléatoire, pour diversifier la recherche ou encore pour rechercher des solutions dans un contexte où la connaissance sur la structure du problème est limitée. L'algorithme 2 présente le pseudo-code du tirage aléatoire uniforme utilisé.

Dans les expériences, nous générons uniformément 1 000 solutions faisables et pour chaque solution nous générons 100 voisins faisables avec possibilité de redondance. La génération uniforme de solutions n'est possible que pour le codage job parce qu'il n'y a aucun infaisable dans ce codage. Pour le codage `ope`, malgré la présence d'infaisables, on utilise la bijection des faisables avec le codage `job` pour choisir uniformément un codage faisable. En revanche, il est très difficile de faire un tirage aléatoire uniforme parmi les solutions faisables du codage `mch`. En effet, pour tirer de manière uniforme, on procède par rejet mais le taux très important d'infaisables rend cette méthode inutilisable en pratique. De même pour le codage `tim`, les durées entre la date de début d'une opération et la fin de l'opération précédente n'étant pas bornée par valeur supérieure dans ce codage, il est impossible d'effectuer un tirage uniforme.

Le tirage aléatoire uniforme permet de calculer la *probabilité d'échappement cumulée* (évolutivité) et la *distribution des types de points* (distribution de fitness).

3.4.3 Recherche locale

En optimisation combinatoire, un processus de recherche locale peut être conceptualisé comme une exploration séquentielle d'un graphe orienté. Dans ce graphe, les sommets représentent les différentes

Algorithme 2 Tirage aléatoire uniforme

Entrée : `instance`, `codage`, fonction de voisinage $\mathcal{V}$, nombre d'itérations `nbItérations`, nombre de voisins faisables `nbVoisinsFsbl`
Sortie : liste de solutions S, liste des voisins V_S de l'ensemble des solutions de S

pour `i` **de** 1 **à** `nbItérations` **faire**
    `s` ← GénérerSolutionFaisable(`instance`, `codage`)
    AjouterÀListe(S, `s`)
    `nbFsblVisités` ← 0
    tant que `nbFsblVisités` < `nbVoisinsFsbl` **faire**
        `s'` ← $\mathcal{V}$(`s`)
        si EstFaisable(`s'`) **alors**
            AjouterÀListe(V_S, `s'`)
            `nbFsblVisités` ← `nbFsblVisités` + 1
        fin si
    fin tant que
fin pour
retourne S, V_S

solutions possibles de l'espace de recherche, tandis que les arcs indiquent les relations entre chaque solution et ses voisins, comme évoqué par [Aarts and Lenstra, 2003]. La recherche locale, en tant que métaheuristique à solution unique, opère comme une heuristique classique où les voisins d'une solution courante sont considérés comme des alternatives potentielles. Lorsqu'un voisin est choisi, la recherche se déplace vers cette nouvelle solution et examine à son tour les voisins de cette dernière. Ce cycle continue jusqu'à la satisfaction d'un critère d'arrêt. Le choix du voisin à explorer à chaque étape est dicté par une politique de mouvement (voir section 1.1.6).

Afin de tester les différents couples codage-voisinage et de trouver des corrélations avec les résultats de l'analyse du paysage, nous avons mis en place une recherche taboue comme méthode de recherche locale. Nous avons opté pour une métaheuristique à solution unique car nos opérateurs de voisinage génèrent un unique voisin. Quant au choix de la recherche taboue, il s'explique par son utilisation antérieure, notamment par [Taillard, 1994], et plus récemment par [Strassl and Musliu, 2022], qui ont utilisé une paire codage-voisinage similaire à `mch-cas`, ce qui en fait une base de comparaison pertinente. Nous avons adapté la recherche taboue de [Taillard, 1994], en la rendant plus générique à l'égard des codages et des opérateurs de voisinage. Dans notre algorithme, la longueur de liste taboue (`LongueurTaboue`), exprimée en nombre d'itérations, est celle définie par E. Taillard. L'algorithme 3 montre l'implémentation de notre recherche taboue générique. GénérerSolutionFaisable génère une solution faisable en partant généralement d'une solution du codage job et en la convertissant dans le codage considéré.

Pour chaque couple codage-voisinage, la recherche taboue a été exécutée 10 fois avec une durée totale de recherche limitée à 10 secondes pour chacune des exécutions *(runs)*. L'exploration est arbitrairement bornée à 20 voisins au total (faisables ou non) par itération, afin d'économiser le temps CPU d'évaluation des voisinages. Si l'algorithme ne trouve pas de candidat (par exemple, aucun voisin n'est faisable), il redémarre la procédure depuis le début. Cette situation peut restreindre ses performances, surtout lorsque les opérateurs rencontrent des difficultés à trouver un voisin faisable. Sur les résultats des 10 exécutions de la recherche taboue, seule la meilleure est conservée. Les résultats sont classés par rang et par instance.

Algorithme 3 Recherche taboue générique inspirée de [Taillard, 1994]

Entrée : `instance`, `codage`, fonction de voisinage $\mathcal{V}$, délai d'exécution `délai`, nombre de voisins `nbVoisins`
Sortie : meilleure solution `meilleure`

`début` ← RécupérerTemps( )
LongueurTaboue ← $\left(n + \frac{m}{2}\right) e^{-\frac{n}{5m}} + \frac{N}{2} e^{-\frac{5m}{n}}$
`s` ← GénérerSolutionFaisable(`instance`, `codage`)
`meilleure` ← `s`
tant que TempsNonEcoulé(`début`, `délai`) **faire**
    `candidate` ← Null
    pour `i` **de** 1 **à** `nbVoisins` **faire**
        `voisin` ← $\mathcal{V}$(`s`)
        si EstInfaisable(`voisin`) **alors**
            continuer
        fin si
        si `candidate` = Null **ou** C_{max}(`voisin`) < C_{max}(`candidate`) **alors**
            si EstTaboue(`voisin`) **alors**
                si C_{max}(`voisin`) ≥ C_{max}(`s`) **alors**
                    continuer
                fin si
            fin si
            `candidate` ← `voisin`
        fin si
    fin pour
    ActualiserListeTaboue(LongueurTaboue)
    si `candidate` ≠ Null **alors**
        `s` ← `candidate`
        AjouterÀListeTaboue(`candidate`)
    sinon
        ViderListeTaboue( )
        `s` ← GénérerSolutionFaisable(`instance`, `codage`)
    fin si
    si C_{max}(`s`) < C_{max}(`meilleure`) **alors**
        `meilleure` ← `s`
    fin si
fin tant que
retourne `meilleure`

En particulier, pour `ins`, `swp` et `rev` s'appliquant à `ope`, nous avons modifié les opérateurs afin d'éviter les solutions totalement ou partiellement infaisables. Plus précisément, après avoir obtenu un premier indice aléatoire i_1, nous vérifions la chaîne des indices possibles autour de i_1 qui n'ont pas le même numéro de job que i_1 et nous obtenons le deuxième indice dans cet intervalle. La figure 3.3 illustre, pour une solution `ope` et l'opérateur de voisinage `ins`, l'utilité de prendre en compte la modification décrite pour éviter les solutions infaisables. Dans cet exemple, l'indice $i_1 = 2$ correspond à l'opération $O_{1,2}$. Pour un $i_2 = 5$,

on observe qu'une opération de J_1 ($O_{1,3}$) est située entre les indices 2 et 5, ce qui conduit à une situation d'infaisabilité à partir de l'indice 4 (a). En revanche, pour un $i_2 = 3$, la solution obtenue demeure faisable car il n'y a aucune autre opération de J_1 entre les indices 2 et 3 (b). Cette modification ne change pas la structure du paysage mais améliore considérablement les performances de la recherche taboue.

Cependant, cette adaptation n'est pas applicable lorsque les mêmes opérateurs sont utilisés sur `mch`. En effet, étant donné que les opérations sont réparties sur les séquences des différentes machines avec `mch`, il n'est pas envisageable d'anticiper l'ensemble des opérations qui seront affectées par l'application de l'opérateur de voisinage sur l'une des séquences.

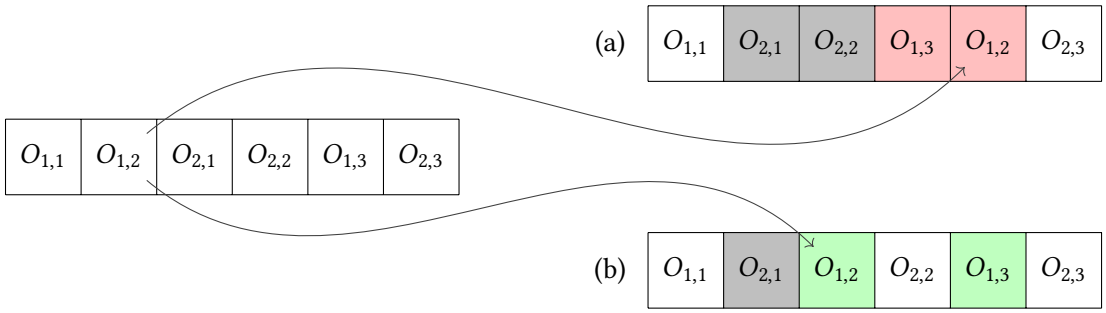


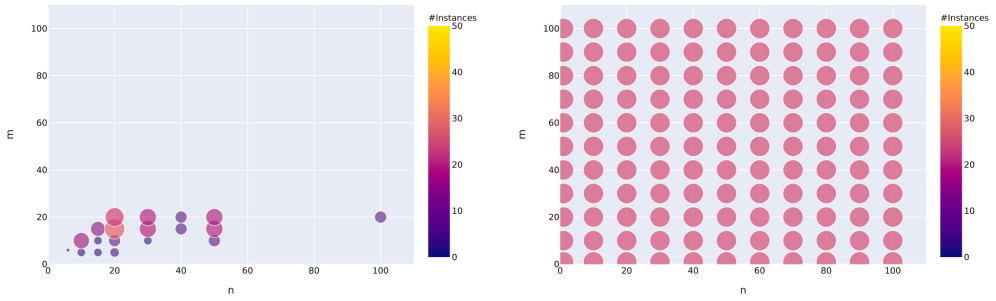
FIGURE 3.3 – Illustration de la méthode permettant d'éviter les solutions infaisables pour le codage `ope` avec l'opérateur de voisinage `ins`

3.5 Instances JSP

Pour les expérimentations rapportées dans ce chapitre, nous avons utilisé deux ensembles d'instances. Le premier ensemble regroupe 242 instances classiques tirées des benchmarks de job shop [1]. Ces instances de taille et de complexité variées sont décrites en détail par [van Hoorn, 2018]. Le tableau 3.2 en présente les principales caractéristiques.

Le deuxième ensemble est un groupe de 2500 instances générées [2] créées par [Strassl and Musliu, 2022]. Strassl et Musliu constatent en effet que les instances classiques souffrent de deux problèmes : d'une part, les temps p_{ij} sont tous générés de la même manière ; d'autre part, les valeurs des couples (n, m) ne sont pas bien réparties et ne couvrent pas l'ensemble des possibilités (figure 3.4). Les auteurs proposent donc de générer des instances pour n et m variant de 10 à 100 de 10 en 10, et d'utiliser d'autres distributions de probabilité pour générer les p_{ij}. Le tableau 3.3 fournit les caractéristiques de ces instances, en particulier la distribution utilisée pour générer les temps de traitement des opérations. `gen-const` indique une distribution constante égale à 1, `gen-uniform-99` est une distribution uniforme sur l'intervalle $[1, 99]$, `gen-uniform-200` une distribution uniforme sur l'intervalle $[1, 200]$, `gen-binom` est une distribution binomiale avec $n = 98$ et $p = 0.5$ et `gen-nbinom` représente la distribution binomiale négative avec $r = 1$ et $p = 0.5$.

1. `http://jobshop.jjvh.nl/`
2. `https://doi.org/10.5281/zenodo.4081658`



(a) Instances existantes (`lit`) (b) Instances générées (`gen`)

FIGURE 3.4 – Nombre d'instances par job et par machine [Strassl and Musliu, 2022]

TABLEAU 3.2 – Instances JSP classiques

Préfixe	Nombre d'instances	n	m	Référence
`abz`	5	10,20	10,15	[Adams et al., 1988]
`dmu`	80	20,30,40,50	15,20	[Demirkol et al., 1998]
`ft`	3	6,10,20	5,6,10	[Fisher and Thompson, 1963]
`la`	40	10,15,20,30	5,10,15	[Lawrence, 1984]
`orb`	10	10	10	[Applegate and Cook, 1991]
`swv`	20	20,50	10,15	[Storer et al., 1992]
`ta`	80	15,20,30,50,100	15,20	[Taillard, 1993]
`yn`	4	20	20	[Yamada and Nakano, 1992]

TABLEAU 3.3 – Instances JSP générées (`gen`) de [Strassl and Musliu, 2022]

Distribution	Nombre d'instances	n	m
`gen-const`	500	10,20,...,100	10,20,...,100
`gen-uniform-99`	500	10,20,...,100	10,20,...,100
`gen-uniform-200`	500	10,20,...,100	10,20,...,100
`gen-binom`	500	10,20,...,100	10,20,...,100
`gen-nbinom`	500	10,20,...,100	10,20,...,100

3.6 Caractérisation des paysages pour le JSP

Dans cette section, nous présentons les résultats de l'analyse de paysage de fitness pour les différents couples de codage et d'opérateur de voisinage appliqués aux 2742 instances de JSP définies ci-dessus. Pour cette analyse, nous utilisons les 4 métriques identifiées au chapitre 1 comme applicable à notre problème d'optimisation (voir tableau 1.2) : la distance de corrélation, le taux de neutralité, la probabilité d'échappement cumulée et la distribution des types de points.

Dans la suite, les résultats sont présentés sous forme de diagrammes en boîte (`box-plot`), aussi appelé boîtes à moustache. Un `box-plot` est un graphique simple composé d'un rectangle duquel deux droites

sortent afin de représenter certains éléments des données (figure 3.5 [3]). La valeur centrale du graphique est la médiane (il existe autant de valeurs supérieures qu'inférieures à cette valeur dans l'échantillon). Les bords du rectangle sont les quartiles (pour le bord inférieur, un quart des observations ont des valeurs plus petites et trois quart ont des valeurs plus grandes, le bord supérieur suit le même raisonnement). Les extrémités des moustaches sont calculées en utilisant 1.5 fois l'espace interquartile (la distance entre le 1er et le 3^{e} quartile). On peut remarquer que 50% des observations se trouvent à l'intérieur de la boîte. Les valeurs à l'extérieur des moustaches sont représentées par des points. On ne peut pas dire que si une observation est à l'extérieur des moustaches alors elle est associée à une valeur aberrante. Par contre, cela indique qu'il faut étudier plus en détail cette observation.

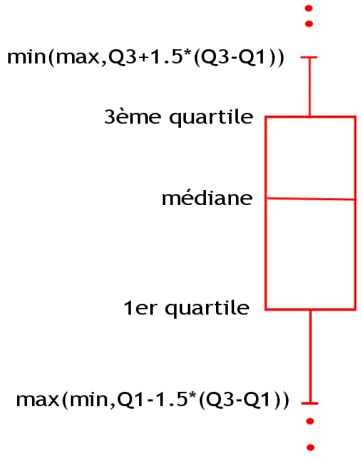


FIGURE 3.5 – Description d'un `box-plot`[3]

3.6.1 Distance de corrélation

Les figures 3.6 et 3.7 indique la distance de corrélation en fonction du nombre de jobs n, pour les instances classiques et les instances générées, respectivement. Les figures sont organisées par paire codage-voisinage. La distance de corrélation présente une plage allant de 2 à plusieurs milliers, tout en maintenant un ordre de grandeur similaire pour la même paire de codage-voisinage.

Dans la plupart des cas, la distance de corrélation augmente avec n. Cette tendance s'explique par la nature des opérateurs de voisinage, qui ont un impact relativement moindre sur les solutions lorsque la taille de l'instance augmente (en particulier le nombre d'opérations), entraînant ainsi des solutions voisines présentant une plus grande corrélation. En revanche, l'opérateur de voisinage `tim` modifie une portion significative de la solution, ce qui se traduit par une légère diminution de la distance de corrélation.

La paire `job-rev` présente une distance de corrélation comprise entre 2 et 3, ce qui implique que les voisins dont la distance est supérieure à 3 ne sont pas corrélés. En réalité, effectuer deux opérations `rev` successives sur un codage `job` revient à obtenir une solution de manière aléatoire.

Pour certaines paires codage-voisinage, les distances de corrélation se situent dans une fourchette moyenne, allant de 10 à 80 : `job-ins`, `job-swp`, `ope-ins`, `ope-rev`, `ope-swp`. Dans ces cas, le paysage n'est ni

3. `https://www.stat4decision.com/fr/le-box-plot-ou-la-fameuse-boite-a-moustache/`

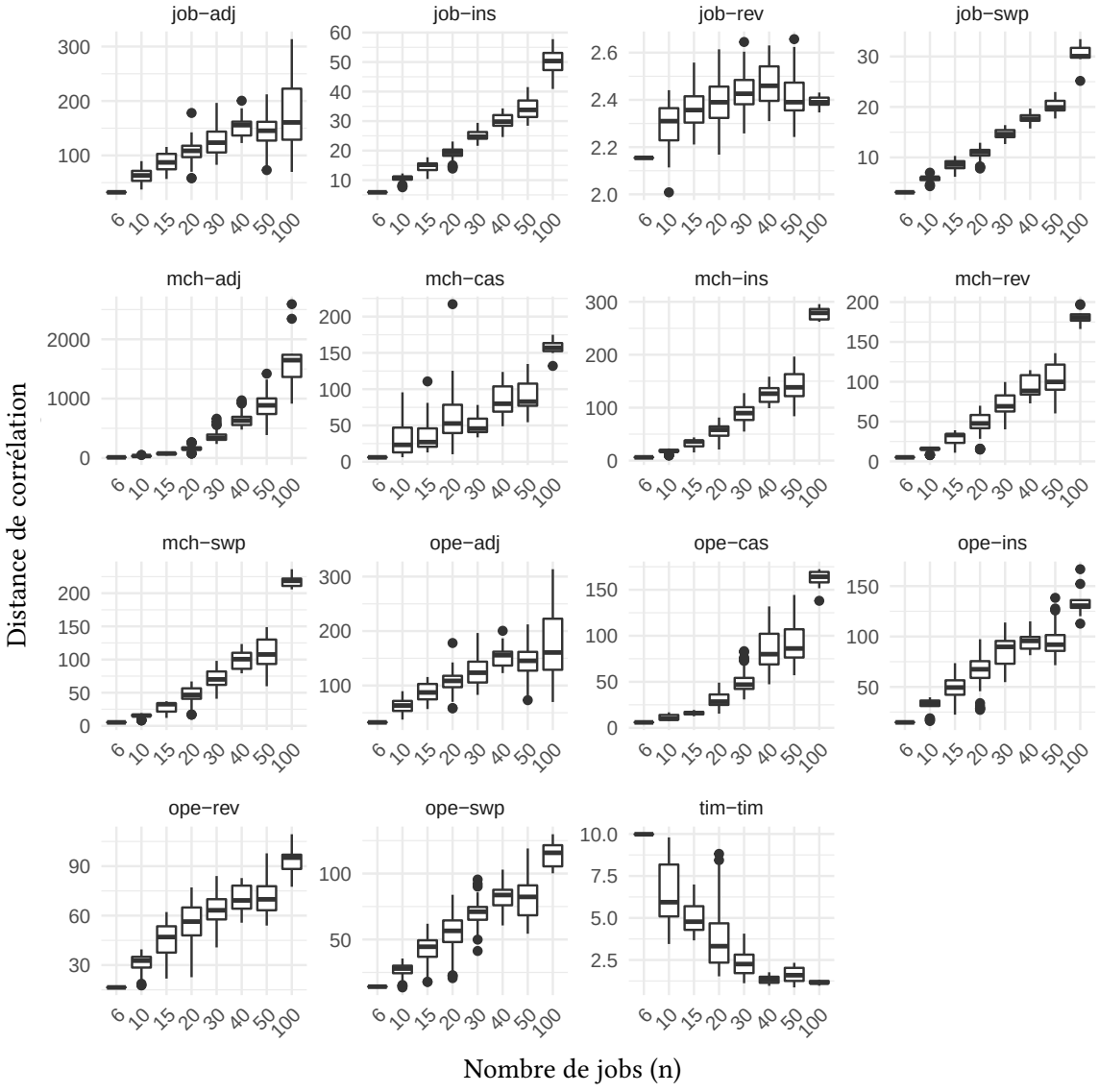


FIGURE 3.6 – Distance de corrélation pour les instances classiques regroupées par codage-voisinage

excessivement rugueux ni trop doux. Les opérateurs de voisinage modifient les solutions de manière significative, mais pas trop, de sorte que la distance de la corrélation corresponde à ce que l'on peut attendre afin d'éviter un trop grand nombre d'optima locaux, tout en obtenant assez rapidement des solutions non corrélées.

Un autre groupe se caractérise par d'importantes distances de corrélation, comprises entre 80 et 200, ce qui traduit un paysage plutôt doux : `mch-cas`, `mch-ins`, `mch-rev`, `mch-swp`, `ope-cas`. L'opérateur `cas` restreint le nombre de voisins possibles, et la plupart de ces voisins ne diffèrent pas considérablement de la solution d'origine, ce qui induit une corrélation notable même après plusieurs itérations. De manière analogue, le codage `mch` présente la même caractéristique, car les opérateurs de voisinage autres que

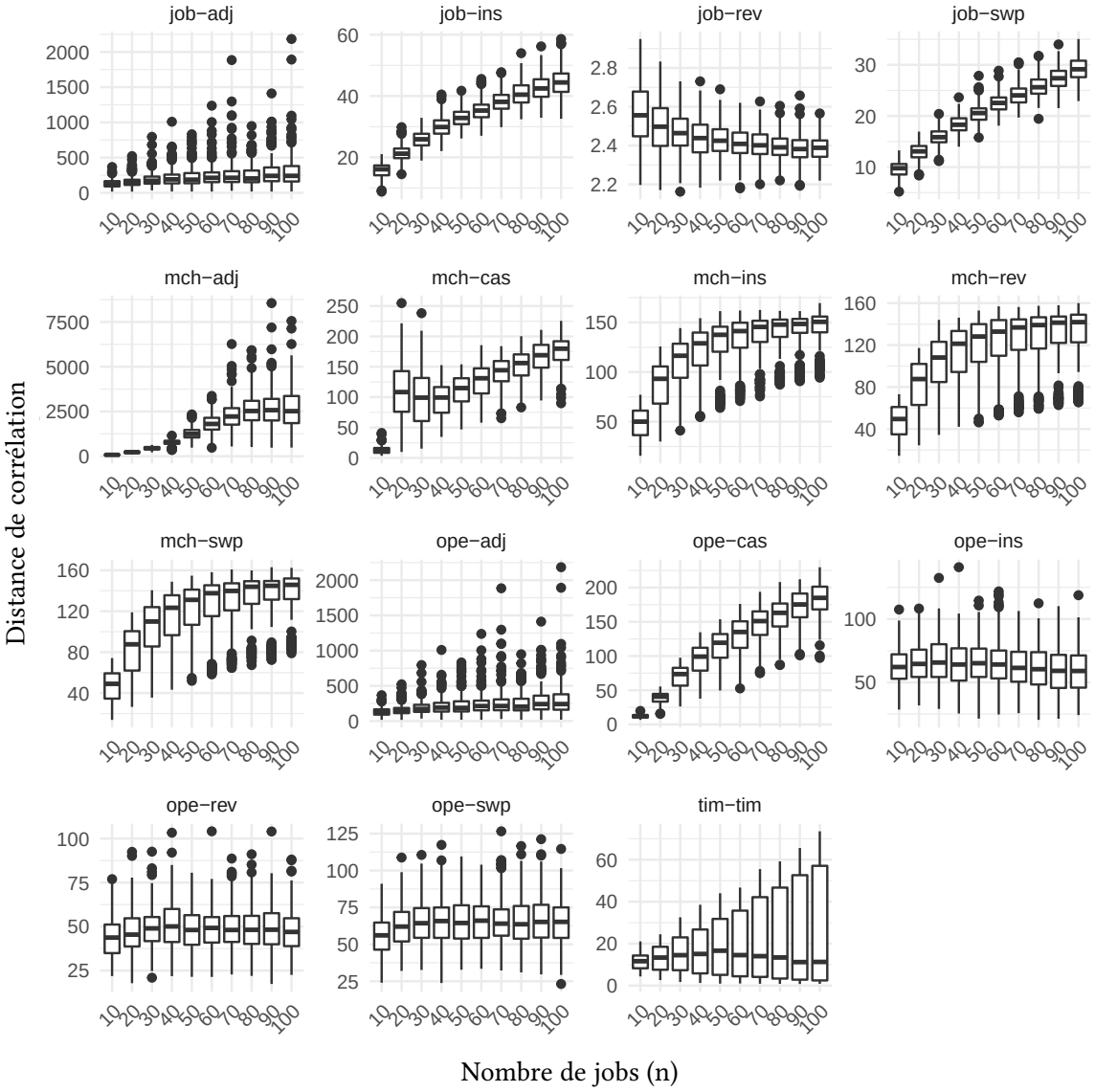


FIGURE 3.7 – Distance de corrélation pour les instances générées regroupées par codage-voisinage

`cas` ne modifient les opérations que sur une seule machine. En conséquence, il n'est pas aisé d'obtenir rapidement des solutions qui ne sont pas corrélées.

L'opérateur de voisinage `adj` génère des distances de corrélation extrêmement élevées, allant jusqu'à plusieurs milliers. Comme mentionné précédemment, cet opérateur n'apporte pas de modifications suffisamment significatives à la solution, ce qui implique que de nombreuses itérations sont requises pour obtenir une solution non corrélée.

Enfin, en ce qui concerne la paire `tim-tim`, elle présente une dispersion significative de la distance de corrélation dans le contexte des instances générées. La figure 3.8 expose la distance de corrélation en

relation avec la distribution de probabilités utilisée pour la génération des temps de traitement de ces instances. Une observation importante est que la distance de corrélation est influencée par la distribution de probabilités utilisée pour la génération des temps de traitement. Deux groupes d'instances se dégagent clairement : l'un affiche une distance de corrélation moyenne (20-50) pour `gen-const` et `gen-nbinom`, tandis que l'autre présente une distance de corrélation faible (5-20) pour `gen-binom`, `gen-uniform-200`, et `gen-uniform-99`. Ces résultats concordent avec ceux obtenus avec les instances classiques où les temps de traitement sont générés de manière uniforme.

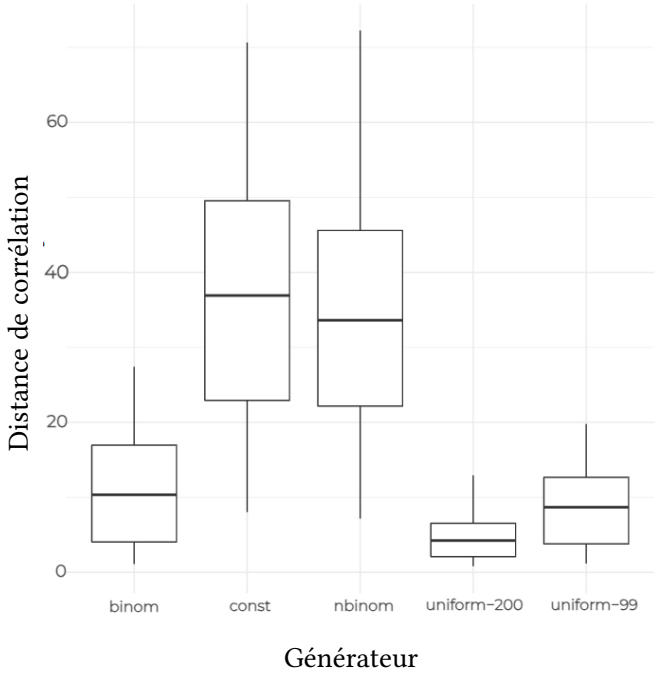


FIGURE 3.8 – Distance de corrélation pour les instances générées avec `tim-tim`

3.6.2 Taux de neutralité

La figure 3.9 montre le taux de neutralité respectivement pour les instances classiques et les instances générées. Le taux de neutralité présente des similitudes notables entre les instances classiques et les instances générées. Sa valeur dépend principalement de la combinaison spécifique de codage et de voisinage utilisée.

Deux combinaisons de codage-voisinage se distinguent par un taux de neutralité particulièrement élevé, avoisinant les 100% : `job-adj` et `ope-adj`. Dans ces cas, l'échange de deux éléments adjacents a très peu d'incidence sur l'ordonnancement. Dans la plupart des cas, soit l'ordonnancement de la nouvelle solution demeure identique à celui de la précédente, soit l'ordonnancement obtenu présente la même valeur de $C_{\max}$, ce qui explique le taux de neutralité exceptionnellement élevé.

Trois autres combinaisons montrent une neutralité élevée, se situant entre 80 et 95% : `ope-ins`, `ope-rev`, et `ope-swp`. Cette situation découle de la difficulté à trouver des voisins faisables avec le codage `ope`. La plupart des voisins possibles ne se distinguent pas de manière significative, conduisant ainsi à des ordonnancements assez similaires avec des valeurs de fitness similaires.

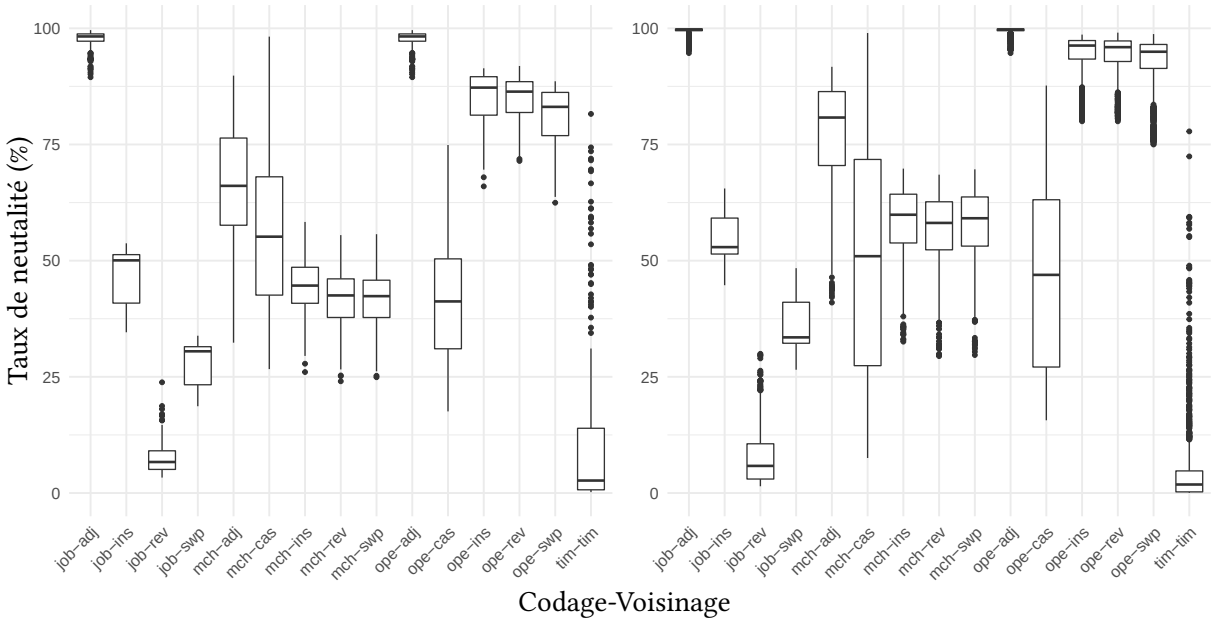


FIGURE 3.9 – Taux de neutralité pour les instances classiques (à gauche) et les instances générées (à droite)

Un grand nombre de combinaisons présentent une neutralité moyenne, se situant entre 25 et 75% : `job-ins`, `job-swp`, `mch-adj`, `mch-cas`, `mch-ins`, `mch-rev`, `mch-swp`, `ope-cas`. Comme avec le codage `ope`, le codage `mch` associé aux voisinages `ins`, `rev`, et `swp` présente des caractéristiques similaires en raison de la difficulté à trouver des voisins faisables. Une fois de plus, l'opérateur `adj`, qui se trouve à l'intersection des trois autres, présente une neutralité plus élevée.

Pour finir, deux combinaisons de codage-voisinage se distinguent par leur neutralité faible, voire très faible : `job-rev` et `tim-tim`. Avec `job-rev`, l'inversion de nombreux éléments de la solution peut conduire à une solution radicalement différente, avec un ordonnancement et un makespan totalement différents. Quant à `tim-tim`, l'opérateur de voisinage modifie suffisamment d'éléments de la solution pour engendrer une valeur de makespan différente.

3.6.3 Probabilité d'échappement cumulée

La figure 3.10 montre la probabilité d'échappement cumulée pour les instances classiques et générées respectivement. Les résultats obtenus sont plutôt similaires pour les deux jeux d'instances. Les paires `job-adj` et `ope-adj` affichent une probabilité d'échappement très faible. Cela est directement lié au taux de neutralité très élevé de leur paysage : il est difficile de trouver un meilleur voisin parmi les voisins ayant la même fitness. De manière similaire, `ope-ins`, `ope-rev`, et `ope-swp` présentent une probabilité d'échappement faible en raison de leur taux de neutralité élevé. Quant à `job-ins` et `job-swp`, elles affichent une probabilité d'échappement moyenne, ce qui est également étroitement lié à leur taux de neutralité moyen.

Globalement, pour ce problème, en moyenne, la moitié des voisins non neutres s'avèrent être de meilleurs voisins, quelle que soit la combinaison de codage et de voisinage. Cela n'apporte pas nécessairement d'indi-

cations cruciales pour la sélection d'une paire de codage-voisinage optimale. Cela signifie simplement que, du point de vue statistique, ces paires présentent des comportements prévisibles en termes d'évolutivité.

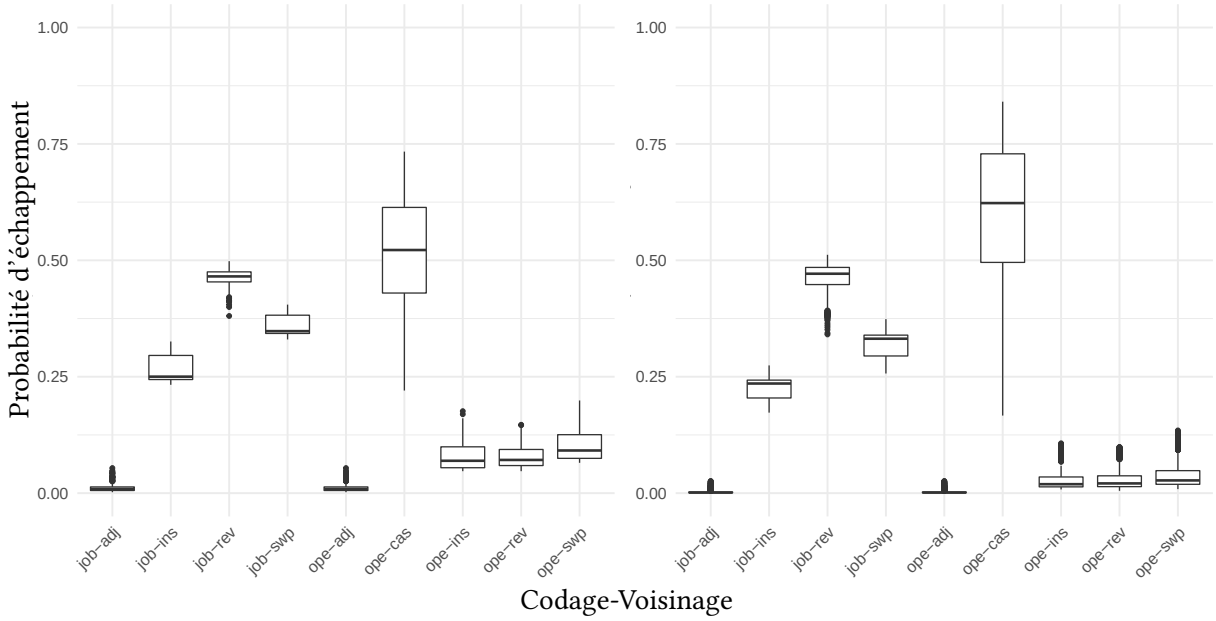


FIGURE 3.10 – Probabilité d'échappement cumulée pour les instances classiques (à gauche) et les instances générées (à droite)

3.6.4 Distribution des types de points

La figure 3.11 illustre la distribution des types de points pour les instances classiques et générées, respectivement. Nous rappelons que le couple `job-cas` n'a pas été considéré, d'où l'absence de graphe dans la figure, et que les codages `mch` et `tim` ne peuvent pas non plus faire l'objet de cette analyse.

Pour cette expérimentation, nous classons les voisins en trois catégories distinctes : ceux qui présentent une meilleure fitness que la solution courante, ceux qui ont une fitness inférieure à la solution courante, et enfin, ceux dont la fitness est équivalente à celle de la solution courante. Chacun de ces groupes est considéré comme significatif s'il représente plus de 10% de l'ensemble des voisins. Ensuite, à partir de ces groupes identifiés, nous déterminons les différents types de points.

Sans surprise, les paires `job-adj` et `ope-adj` montrent un taux très élevé de solutions de type `IPLAT` en raison de leur paysage extrêmement neutre. Elles ne présentent même aucun type de point où l'égalité de fitness est absente (`SLMAX`, `SLOPE`, `SLMIN`).

Pour `ope-cas`, on observe un profil asymétrique, caractérisé par un grand nombre de solutions `LMIN`, ainsi qu'une petite proportion de solutions `SLMIN`, mais très peu de solutions `LMAX` et encore moins de solutions `SLMAX`. Le reste des solutions se compose principalement de solutions `LEDGE`. Les paires `ope-ins`, `ope-rev`, et `ope-swp` affichent des profils similaires, caractérisés par un taux élevé de solutions de type `IPLAT` en raison de leur grande neutralité. Ces paires présentent également des taux significatifs de solutions `LMIN`, `LEDGE`, et `LMAX`.

Pour `job-ins` et `job-swp`, une similitude se dégage également, avec un taux très élevé de solutions `LEDGE`, et des taux très faibles de solutions `LMIN` et `LMAX`.

Enfin, `job-rev` se caractérise par un taux très élevé de solutions de type `SLOPE`, ainsi qu'un taux élevé de solutions `LEDGE`, confirmant la nature très rugueuse de son paysage.

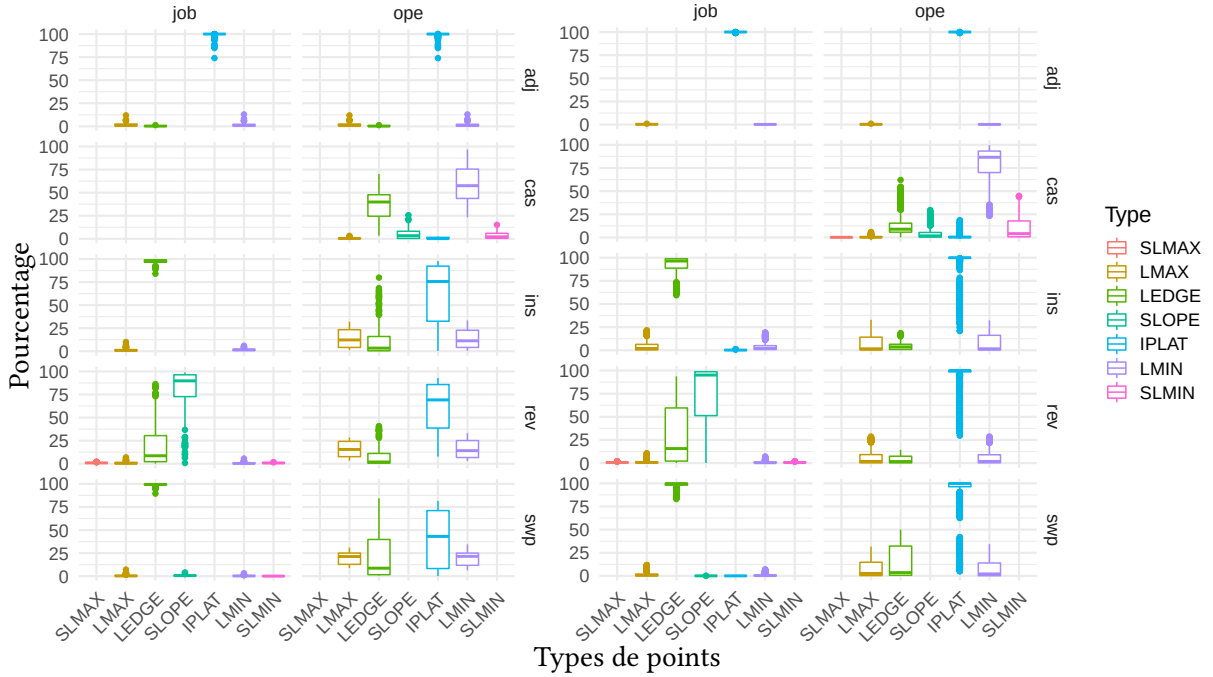


FIGURE 3.11 – Distribution des types de points pour les instances classiques (à gauche) et les instances générées (à droite)

3.6.5 Synthèse sur la caractérisation des paysages

La première phase de notre expérimentation a porté sur la caractérisation des espaces de recherche à travers l'analyse du paysage. À cette étape, notre objectif est de formuler des prédictions sur les performances de chaque paire de codage-voisinage en nous basant exclusivement sur l'analyse du paysage de fitness. Ces prédictions devraient servir à anticiper les résultats lorsque chaque paire sera mise en œuvre avec une métaheuristique. Lorsque le codage est fixé, quel opérateur de voisinage est susceptible d'offrir les meilleures performances ? Et lorsqu'un opérateur de voisinage est fixé, quel type de codage semble produire les meilleurs résultats ? Le tableau 3.4 synthétise les résultats obtenus en utilisant diverses métriques d'analyse du paysage. Une conclusion partielle qui se dégage est que le paysage dépend principalement de la paire de codage-voisinage sélectionnée. L'instance du problème de job shop semble jouer un rôle relativement mineur pour certaines métriques, et dans certains cas, la méthode de génération de l'instance peut avoir un impact. D'autres mesures (qui ne sont pas présentées ici) révèlent que parfois les instances classiques se regroupent par famille lorsque les résultats d'une métrique sont plus dispersés que d'habitude. Cependant, cela reste marginal et il est clair que le facteur prédominant dans les résultats est la combinaison spécifique de codage et d'opérateur de voisinage.

Tableau 3.4 – Synthèse de l'analyse des paysages JSP avec les couples codage-voisinage

	Neutralité	Rugosité	Evolutivité	Distribution de fitness
`job-adj`	Maximale	Faible	Minimale	IPLAT
`job-ins`	Moyenne	Moyenne	Moyen	LEDGE
`job-rev`	Faible	Maximale	Forte	SLOPE
`job-swp`	Moyenne-Faible	Moyenne	Moyenne	LEDGE
`mch-adj`	Moyenne-Forte	Très Faible	–	–
`mch-cas`	Moyenne	Moyenne-Faible	–	–
`mch-ins`	Moyenne	Faible	–	–
`mch-rev`	Moyenne	Faible	–	–
`mch-swp`	Moyenne	Faible	–	–
`ope-adj`	Maximale	Faible	Minimale	IPLAT
`ope-cas`	Moyenne	Faible	Forte	LMIN
`ope-ins`	Forte	Moyenne	Faible	IPLAT
`ope-rev`	Forte	Moyenne	Faible	IPLAT
`ope-swp`	Forte	Moyenne	Faible	IPLAT
`tim-tim`	Minimale	Moyenne/Faible	–	–

3.7 Performance d'une métaheuristique : recherche taboue

3.7.1 Résultats des tests

Les tableaux 3.5 et 3.6 présentent les classements obtenus par chaque paire de codage-voisinage lors de l'exécution de la recherche taboue sur l'ensemble des instances classiques et des instances générées, respectivement. Le rang 1 correspond aux paires qui obtiennent le meilleur $C_{\max}$. Par exemple, dans le tableau 3.5, la paire `job-ins` a obtenu la première place 211 fois sur 242 instances. Ces tableaux fournissent également, pour chaque paire, le rang moyen (colonne Moyenne), le rang médian (colonne Médiane) et la moyenne des écarts relatifs de fitness avec les meilleures valeurs de fitness obtenues pour les instances (colonne R-Écart). Les couples sont présentés dans l'ordre de leur rang moyen croissant. Pour les instances classiques (tableau 3.5), le nombre de fois où la recherche a atteint la meilleure solution connue est également indiqué (colonne Best).

Dans les deux jeux d'instances, la paire `job-ins` se distingue comme la meilleure combinaison de codage-voisinage en termes de rang moyen. Elle performe légèrement mieux dans le cas des instances classiques, avec un rang moyen de 1.19, que dans le cas des instances générées, où elle affiche un rang moyen de 2.59. Ce résultat est plutôt surprenant. Pour rappel, la paire `job-ins` présentait un taux de neutralité médian de 50%, une distance de corrélation médiane de 20 à 50, une probabilité d'échappement médiane de 25%, et se caractérisait par un taux élevé de solutions de type LEDGE. Le codage `job` est un codage pour lequel de nombreuses solutions conduisent au même ordonnancement. Sa principale caractéristique est que toutes ces solutions sont faisables. L'opérateur de voisinage `ins` modifie l'ordre relatif d'un seul élément.

Dans le cas des instances classiques, à l'exception de la paire `job-ins`, les paires de codage-voisinage se regroupent en trois catégories distinctes en fonction de leurs rangs moyens. Le premier groupe comprend quatre paires ayant des rangs moyens compris entre 3.11 et 3.81 : `mch-adj`, `ope-ins`, `job-swp`, et `ope-swp`. Le deuxième groupe rassemble cinq paires avec des rangs moyens allant de 6.33 à 8.99 : `tim-tim`, `mch-cas`, `ope-rev`, `ope-cas`, et `job-rev`. Enfin, le troisième groupe se compose de cinq paires ayant des rangs moyens compris entre 10.81 et 13.36 : `mch-ins`, `ope-adj`, `job-adj`, `mch-swp`, et `mch-rev`.

Dans le cas des instances générées, les groupes sont légèrement différents. Le premier groupe regroupe quatre paires ayant des rangs moyens compris entre 3.06 et 3.74 : `mch-cas`, `tim-tim`, `ope-cas`, et `job-swp`. Le deuxième groupe comprend cinq paires avec des rangs moyens allant de 6.05 à 9.35 : `ope-ins`, `ope-swp`, `mch-adj`, `job-rev`, et `ope-rev`. Enfin, le troisième groupe se compose de cinq paires ayant un rang moyen compris entre 12.31 et 13.66 : `mch-ins`, `mch-rev`, `mch-swp`, `job-adj`, et `ope-adj`.

Il y a quelques tendances communes qui se dégagent de ces résultats. Tout d'abord, pour n'importe quel codage donné, l'opérateur `ins` est généralement meilleur que `swp`, qui est lui-même généralement meilleur que `rev` (sauf pour `mch`). Cet ordre de performance correspond également au nombre de changements dans l'ordre relatif des éléments dans le voisinage (1 pour `ins`, 2 pour `swp` et 1 à $\sum_{i=1}^{n} N_i$ pour `rev`).

Deuxièmement, pour les opérateurs de voisinage `ins`, `swp` et `rev`, le codage `job` donne généralement de meilleurs résultats que `ope`, qui à son tour donne généralement de meilleurs résultats que `mch`. Ces résultats semblent refléter la complexité des codages, c'est-à-dire la quantité de contraintes codées.

Un troisième point commun notable est que le groupe le moins performant reste généralement le même, avec une grande marge de moins bons résultats. Dans ce groupe, on retrouve `job-adj` et `ope-adj`, qui présentent une neutralité proche de 100% et une distance de corrélation très importante. Également inclus dans ce groupe sont `mch-ins`, `mch-swp`, et `mch-rev`, qui génèrent de nombreuses solutions infaisables, ce qui n'est pas favorable pour une mise en œuvre de la recherche taboue, comme cela a été expliqué précédemment.

Il y a quelques différences notables entre les performances des paires de codage-voisinage pour les deux ensembles d'instances. Tout d'abord, les performances de `mch-cas` et `tim-tim` sont très bonnes pour les instances générées, avec un rang moyen de 3.06 pour les deux, tandis que pour les instances classiques, leurs performances sont moyennes, avec un rang moyen de 6.51 et 6.33 respectivement. À l'inverse, `mch-adj` obtient de bons résultats pour les instances classiques, avec un rang moyen de 3.11, tandis que pour les instances générées, ses performances sont moyennes, avec un rang moyen de 7.24.

Il est important de noter que la différence de performance entre ces deux ensembles d'instances ne semble pas être due à la distribution aléatoire utilisée pour les temps de traitement des instances générées. La principale raison de cette différence est le nombre de machines, qui ne dépasse jamais 20 dans les instances classiques, mais varie de 10 à 100 dans les instances générées. Ce changement dans la taille du problème peut influencer les performances des paires codage-voisinage.

La Figure 3.12 montre comment le rang moyen de chaque paire codage-voisinage varie en fonction du nombre de machines m et du nombre de jobs n pour les instances générées.

Tableau 3.5 – Rangs de recherche taboue de codage-voisinage pour 242 instances classiques

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Moyenne	Médiane	R-Écart	Best
job-ins	211	25	2	2	1		1									1.19	1.00	1.002	36
mch-adj	49	62	47	35	29	7		11	1					1		3.11	3.00	1.056	35
ope-ins	26	56	60	51	29	8	2	3	3		1		1	1	1	3.44	3.00	1.049	25
job-swp	44	23	39	33	86	14	3									3.61	4.00	1.038	36
ope-swp	30	22	45	73	33	25	10	1	1	2						3.81	4.00	1.054	28
tim-tim	28	6		2	8	83	44	30	29	3	4	4		1		6.33	6.00	1.096	23
mch-cas	31	7	11	7	8	26	55	50	23	11	1	3	3	4	2	6.51	7.00	1.160	25
ope-rev	21				2	36	36	44	91	3	1	3	4	1		7.49	8.00	1.153	21
ope-cas	14		1	2	7	16	54	60	49	14	10	3	1	3	8	7.92	8.00	1.178	13
job-rev	19	1		5	3	3	3	4	15	166	23					8.99	10.00	1.191	18
mch-ins	16		1	1	3	7	5	10	7	14	55	16	83	19	5	10.81	12.00	1.323	16
ope-adj	13			1					1	4	69	82	33	27	12	11.55	12.00	1.396	13
job-adj	13						1		1		50	80	51	30	16	11.83	12.00	1.400	13
mch-swp	15					1	1		5	7	6	29	27	102	49	12.69	14.00	1.401	15
mch-rev	15							1	1	3	10	12	20	43	137	13.36	15.00	1.425	15

Tableau 3.6 – Rangs de recherche taboue de codage-voisinage pour 2500 instances générées

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Moyenne	Médiane	R-Écart
job-ins	710	492	525	653	119	1										2.59	3.00	1.086
mch-cas	677	446	561	224	376	43	76	64	25	2	4	2				3.06	3.00	1.099
tim-tim	1164	244	179	205	285	110	107	99	59	39	6	3				3.06	2.00	1.039
ope-cas	253	738	341	410	259	155	169	124	32	8	5	4	2			3.72	3.00	1.116
job-swp	284	258	471	544	778	94	50	21								3.74	4.00	1.104
ope-ins	183	21	97	84	131	741	802	412	28		1					6.05	6.00	1.224
ope-swp	174	11	44	77	127	706	864	364	133							6.27	7.00	1.225
mch-adj	197	67	70	97	132	180	101	670	572	361	32	12	9			7.24	8.00	1.276
job-rev	17		4	44	90	378	157	571	722	508	9					8.11	8.00	1.282
ope-rev	43		2	2	10	27	26	89	853	1411	24	7	5	1		9.35	10.00	1.382
mch-ins	1						1	3	8	47	724	545	847	162	162	12.31	12.00	1.496
mch-rev									5	45	761	680	512	220	277	12.37	12.00	1.505
mch-swp							1	1	8	40	673	657	622	323	175	12.40	12.00	1.505
job-adj	2		1		1	1			4	6	256	296	192	942	799	13.66	14.00	1.579
ope-adj						1	1	1	2	11	255	294	205	942	788	13.66	14.00	1.578

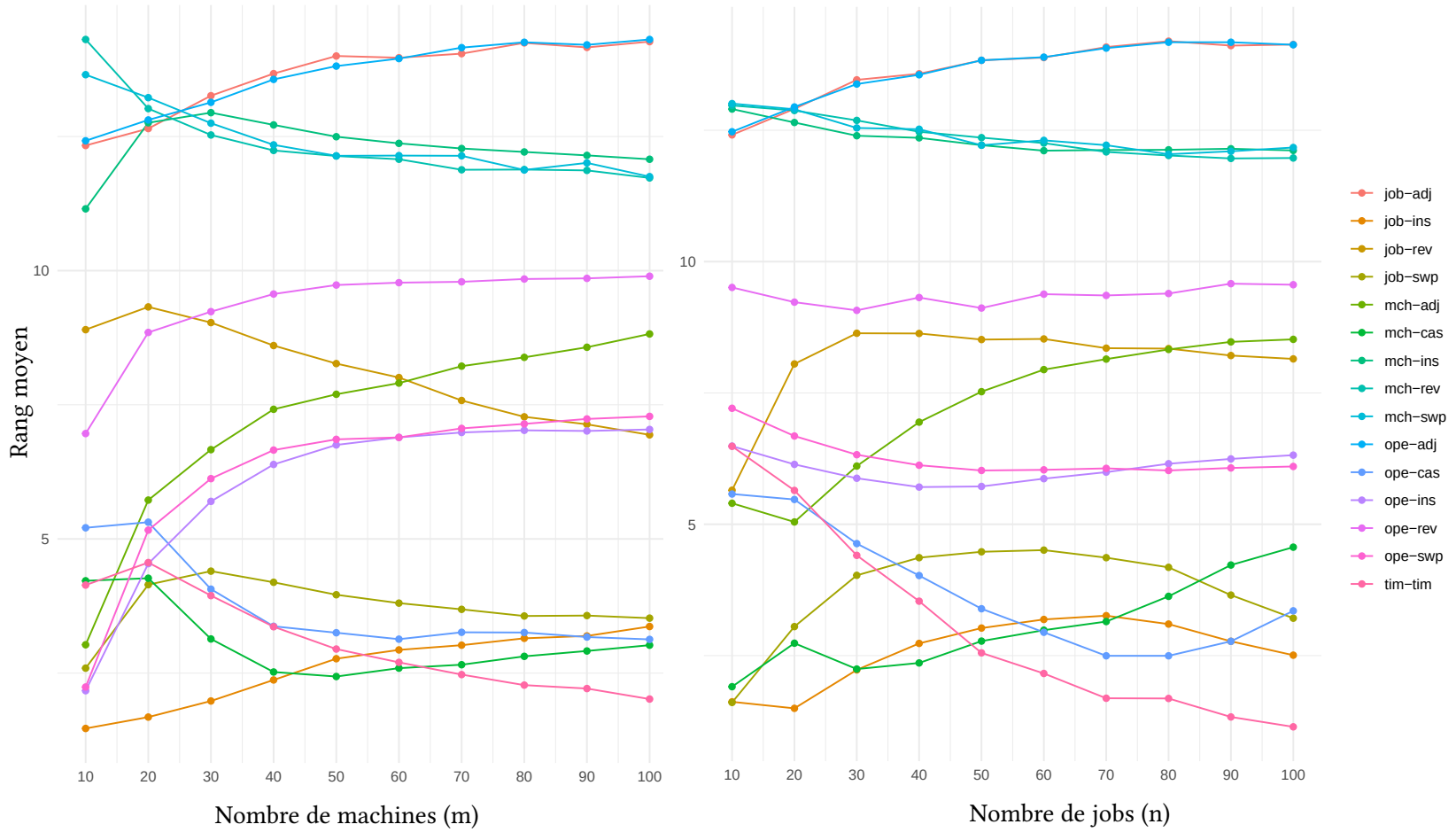


FIGURE 3.12 – Rangs moyens de codage-voisinage en fonction du nombre de machines (à gauche) et du nombre de jobs (à droite) pour les instances générées

Dans la première figure, nous pouvons observer les trois groupes que nous avons identifiés précédemment. De plus, il est intéressant de noter que pour un faible nombre de machines, les rangs peuvent différer considérablement de ceux obtenus pour un nombre élevé de machines ($m \geq 40$). Par exemple, `mch-cas` se trouve parmi les deux meilleures paires pour $m = 30$ à $m = 100$, mais il est seulement septième pour $m = 10$ et troisième pour $m = 20$. En revanche, `job-ins` reste généralement performant, ce qui explique ses bons résultats globaux. `tim-tim` devient la meilleure paire à partir de $m \geq 70$ et présente un comportement globalement satisfaisant. Dans la deuxième figure, nous observons à nouveau les trois groupes que nous avons identifiés. `mch-cas` affiche de bonnes performances pour un petit nombre de jobs ($n \leq 50$), tandis que `ope-cas` s'améliore à mesure que le nombre de jobs augmente. Le rang de `job-ins` reste relativement faible, bien qu'il soit surclassé par trois autres paires entre $n = 60$ et $n = 70$. Enfin, `tim-tim` devient la première paire pour $n \geq 50$. Ces observations mettent en évidence comment les performances des couples codage-voisinage peuvent varier en fonction des caractéristiques spécifiques des instances générées.

3.7.2 Synthèse sur les performances

Dans la seconde phase de notre expérimentation, nous avons effectué des tests de performance en évaluant la convergence de la valeur de $C_{\max}$ pour différentes instances du JSP en utilisant différentes paires de codage-voisinage. Nous avons observé que les classements des paires varient entre les instances classiques et les instances générées. Cependant, une démarcation générale positive a émergé pour le codage `job` et l'opérateur de voisinage `ins`, que ces éléments soient associés ou non. En d'autres termes, cette paire codage-voisinage a montré de bonnes performances dans la convergence de $C_{\max}$. En revanche, le codage `mch` et l'opérateur `adj` ont été associés aux paires de codage-voisinage ayant obtenu les moins bons résultats en termes de convergence. Ces résultats confirment que, dans le contexte du JSP, le choix du codage et de l'opérateur de voisinage peut avoir un impact significatif sur les performances de l'algorithme en termes de convergence vers des solutions de meilleure qualité.

3.8 Bilan

Dans ce chapitre, nous avons tout d'abord établi un protocole expérimental rigoureux pour pallier les lacunes identifiées dans les travaux antérieurs portant sur la caractérisation des codages de solution pour le problème de job shop (JSP) :

- nous avons standardisé les conditions expérimentales pour l'ensemble des tests effectués en limitant et en simplifiant les décisions prises dans les moteurs d'ordonnancement ;
- nous avons considéré 2 742 instances afin d'obtenir des résultats significatifs, sachant que les études similaires réalisées dans la littérature considèrent en général peu d'instances (la plupart du temps moins d'une soixantaine d'instances, à part [Strassl and Musliu, 2022] avec 242 instances) ;
- nous avons comparé non pas des codages et opérateurs de voisinage indépendamment mais en association, afin de mesurer leur impact respectif et conjoint.

Les résultats des expérimentations nous ont permis de tirer des conclusions intéressantes. Si dans la littérature, des codages tels que `job` et `mch` sont identifiés comme particulièrement adaptés, leur performance en association avec des voisinages n'a pas été étudiée. Pour notre part, nous constatons par exemple que le codage `job` est sans doute celui que nous pourrions préconiser s'il est associé à `ins`, alors qu'il est peu performant avec le voisinage `rev`. Donc notre étude tendrait à montrer que le fait de préconiser un codage seul dans une métaheuristique n'est pas pertinent.

A l'inverse, pour les opérateurs de voisinage, une tendance a été observée : de meilleure performance de `ins` sur `swp` et `swp` sur `rev`, quel que soit le codage associé. Néanmoins, le comportement de l'opérateur `adj` est fortement dépendant du codage qui lui est associé, aussi nous nous garderons ici d'émettre des préconisations quant au choix d'un opérateur de voisinage seul.

La littérature sur les paysages de fitness nous avait laissé espérer que les métriques dédiées aux problèmes d'optimisation, tels que les problèmes d'ordonnancement, pourraient nous aider à identifier des schémas récurrents sur le comportement des couples codages-opérateurs de voisinage. Néanmoins, à partir de nos expérimentations, nous constatons que l'analyse de paysage de fitness seule ne permet pas de prédire de manière fiable les performances des différentes paires codage-voisinage pour le job shop classique. Cependant, elle peut fournir des indices utiles pour expliquer les performances médiocres lorsque certaines métriques du paysage sont extrêmes, comme une neutralité très élevée. Cela suggère que certains couples, tels que `job-adj`, `ope-adj`, `mch-ins`, `mch-swp`, et `mch-rev`, devraient être évités pour la résolution du JSP. En ce qui concerne les performances des paires codage-voisinage avec l'algorithme de recherche taboue, nous avons observé que `job-ins`, `mch-cas`, et `tim-tim` étaient les paires les plus performantes. Bien qu'elles présentent des propriétés différentes dans l'analyse de leur paysage de fitness, elles partagent une caractéristique commune : elles demeurent toujours dans l'espace des solutions faisables. Cela est dû soit à l'utilisation d'un codage qui gère les contraintes lors du calcul de l'ordonnancement, soit à l'utilisation d'un voisinage qui garantit la production de solutions faisables.

Compte tenu de ces premiers résultats obtenus sur l'utilisation et la performance des couples codage-opérateur de voisinage pour le job shop dans sa variante classique, la question se pose maintenant quant à leur extension pour la résolution de variantes plus complexes de ce problème. Le chapitre 4 a pour objectif de répondre à cette question.

Chapitre 4

Caractérisation de codages et voisinages pour le job shop flexible avec ressources de transport

Ce chapitre étend l'étude réalisée dans le chapitre précédent au problème de job shop flexible avec ressources de transport, en utilisant la même démarche pour caractériser les espaces de recherche. Les paysages résultant d'une trentaine de combinaisons de codage et voisinage sont analysés, et les performances de convergence avec une recherche taboue sont évaluées. Les résultats sont ensuite comparés à ceux obtenus avec le problème de job shop classique. L'objectif est de déterminer si les caractéristiques de ces codages et opérateurs de voisinage sont généralisables à cette catégorie de problèmes d'ordonnancement d'atelier, auquel cas des règles d'association pertinentes en seront déduites pour les problèmes d'ordonnancement contraints de ce type.

Contenu

4.1 Introduction **92**
4.2 Description de la variante FJSPT **92**
4.3 Codages et opérateurs de voisinage pour le FJSPT **93**
4.4 Moteurs d'ordonnancement **94**
4.5 Espaces de recherche **96**
4.6 Instances FJSPT **96**
4.7 Caractérisation des paysages pour le FJSPT **98**
4.7.1 Distance de corrélation 98
4.7.2 Taux de neutralité 100
4.7.3 Probabilité d'échappement cumulée 102
4.7.4 Distribution des types de points 104
4.7.5 Synthèse sur la caractérisation des paysages 105
4.8 Performance avec la recherche taboue **108**
4.8.1 Résultats des tests 108
4.8.2 Synthèse sur les performances 112
4.9 Bilan **113**

4.1 Introduction

Le problème de job shop flexible avec ressources de transport (FJSPT) est une extension plus contrainte du problème classique de job shop, qui revêt une pertinence significative dans le domaine de la production industrielle. Son intérêt réside principalement dans le fait qu'il réflète de manière plus réaliste les défis rencontrés dans les environnements de production modernes où les machines sont polyvalentes et où le transport efficace des pièces entre les postes de travail est essentiel. En tant que problème d'optimisation combinatoire, la résolution du FJSPT contribue également au développement de la recherche opérationnelle, notamment en ce qui concerne les algorithmes d'optimisation dont les métaheuristiques.

Cette thèse s'articule autour de l'ordonnancement de tâches dans des cas contraints, d'où notre intérêt pour le FJSPT. L'objectif principal est de caractériser les codages et les opérateurs de voisinage qui définissent les espaces de recherche spécifiques à ce type de problème. Dans cette dynamique, nous avons observé que la même question reste non tranchée dans la littérature pour la variante de base qu'est le JSP. Pour mieux contextualiser notre approche, nous avons donc choisi d'initier notre étude par le JSP, comme détaillé dans le chapitre précédent, où différents couples de codages et d'opérateurs de voisinage ont été caractérisés. Le présent chapitre poursuit cette démarche en appliquant la même méthodologie à une dizaine de codages associés à trois opérateurs de voisinage classiques dans le contexte du FJSPT.

Notons qu'il n'existe, à notre connaissance, aucune étude dans la littérature examinant l'impact du choix de codages et d'opérateurs sur l'efficacité des métaheuristiques dans la résolution du problème spécifique de FJSPT. Notre étude a ainsi pour objectif d'apporter des éléments de réponse à cette question. De plus, elle vise à déterminer si les résultats obtenus dans l'étude du JSP peuvent être transposés au FJSPT, et potentiellement à d'autres variantes plus complexes.

4.2 Description de la variante FJSPT

Le problème de job shop flexible avec ressources de transport peut être vu comme une variante du FJSP, qui, à son tour, est une extension du JSP comme expliqué dans le chapitre 1. Dans le FJSPT étudié, il est utile de rappeler que la flexibilité se traduit par la possibilité pour chaque opération d'être exécutée par une machine sélectionnée parmi un ensemble de machines candidates. Les machines sont également équipées de zones de stockage tampons à l'entrée et à la sortie, ayant une capacité suffisante pour accueillir les jobs (pièces). Nous ignorons dans cette variante les temps de configuration et les interruptions du système. Une fois qu'une opération est achevée sur une machine, le job associé est acheminé vers la machine suivante dès que la ressource de transport qui lui est attribuée est disponible. Un trajet à vide est effectué par la ressource de transport depuis son emplacement courant jusqu'à la machine depuis laquelle le job doit être transporté (si l'emplacement courant diffère de la machine de départ). De là, le job est convoyé vers la prochaine machine destinée à le traiter ou vers une station de déchargement, on parle de trajet/transport en charge. Les trajets effectués à vide et en charge peuvent avoir des durées différentes. Nous partons du principe que les ressources de transport peuvent transporter n'importe quel job, mais elles ne sont pas cumulatives, ce qui signifie qu'une ressource de transport ne peut transporter qu'un seul job à la fois.

À titre de rappel, le FJSPT peut être décomposé en quatre sous-problèmes, à savoir le séquencement des opérations, l'affectation des machines, le séquencement des tâches de transport, et l'affectation des ressources de transport. L'objectif ici consiste à résoudre ces sous-problèmes de manière à obtenir

un ordonnancement faisable, c'est-à-dire un ordonnancement dans lequel toutes les contraintes sont satisfaites, tout en minimisant la date de complétion $C_{\max}$, de manière similaire à l'approche adoptée pour le JSP classique.

4.3 Codages et opérateurs de voisinage pour le FJSPT

La caractérisation des différentes paires de codage-voisinage pour le FJSPT a suivi le même protocole expérimental que celui utilisé pour le JSP. Afin de représenter les solutions dans la variante FJSPT, qui intègre la flexibilité des machines et le transport des jobs, la liste des codages a été élargie. En plus des codages issus de la littérature sur le JSP, elle a été enrichie par des codages provenant de la littérature sur le FJSP, le JSPT et le FJSPT. Ainsi, nous avons intégré 7 nouveaux codages déjà présentés au chapitre 2, en plus des 3 utilisés précédemment pour le JSP. En ce qui concerne les opérateurs de voisinage, nous avons conservé trois parmi ceux qui étaient associés aux codages dans le chapitre 3. Le tableau 4.1 rassemble les codages et les opérateurs de voisinage pour lesquels l'ensemble des combinaisons possibles a été retenu pour ce chapitre, soit un total de 30 paires de codage-voisinage.

Tableau 4.1 – Codages et opérateurs de voisinages investigués pour le FJSPT

Codages	job (`job`) opération (`ope`) machine (`mch`) séquence d'opérations et affectation de machines (`osma`) séquence d'opérations et affectation de ressources de transport (`osta`) séquence d'opérations et affectation de ressources de transport en réels (`rosta`) affectation de transport et de job (`tja`) séquence d'opérations et affectation de ressources (`osra`) affectation des tâches de transport (`tta`) affectation de ressources et séquence d'opérations job-transport (`rajts`)
Voisinages	insertion (`ins`) échange (`swp`) inverse (`rev`)

De la même manière que cela a été fait pour le JSP, bien que les codages `mtx`, `rkey` et `osmc` soient utilisables, nous préférons maintenir le codage `mch`, vers lequel les autres peuvent être convertis. Le codage `tim` pour sa part n'a pas été reconduit parce que l'introduction de la flexibilité de machines et des temps de transport rend moins pertinent ce codage (qui repose sur des délais minimaux entre les opérations successives d'un même job) pour le FJSPT comparé au JSP. En effet, d'une part on ne le retrouve pas dans la littérature du FJSPT, d'autre part, la définition du codage demande une modification de la définition de la notion d'écart minimal entre deux opérations successives. Concrètement, l'écart traduisant une attente entre deux opérations successives d'un même job peut être considéré avant ou après le transport sur une machine, auquel cas le temps de trajet s'y ajoute, ou bien il peut être considéré temps de transport compris. Dans ce dernier cas, si le temps de transport est inférieur au temps d'attente, il faut déterminer sur quelle machine le reliquat s'effectuera (attente sur la machine amont ou aval). Le choix de cette définition pour le codage `tim` induirait quatre moteurs d'ordonnancement différents et des résultats potentiellement divergents.

L'opérateur de voisinage `cas` implique la notion de chemin critique, qui devient complexe à appréhender dans le contexte du FJSPT en raison de l'incorporation des tâches de transport entre les opérations sur différentes machines. C'est pourquoi `cas` ne fait pas partie des opérateurs de voisinage retenus. En ce qui concerne l'opérateur `adj`, sa non-sélection découle principalement de son absence dans la liste des opérateurs couramment utilisés dans la littérature du FJSPT. De plus, pour le JSP, il s'est révélé pertinent en termes de performances uniquement pour le codage `mch`.

Pour les codages qui présentent plusieurs vecteurs, les opérateurs `ins`, `swp` et `rev` sont appliqués individuellement sur chaque couche afin d'obtenir un voisin.

4.4 Moteurs d'ordonnancement

Les moteurs d'ordonnancement pour les codages FJSPT génèrent des ordonnancements semi-actifs, suivant la même approche que celle décrite pour le JSP dans le chapitre précédent. Toutefois, certains codages peuvent nécessiter soit une règle d'affectation des machines, soit une règle d'affectation des ressources de transport, soit les deux, pour déterminer où placer les différentes opérations dans l'ordonnancement. Des tests préliminaires ont été conduits en utilisant trois règles d'affectation distinctes :

1. Affectation aléatoire de la machine et de la ressource de transport
2. Affectation de la première meilleure machine, puis de la première meilleure ressource de transport
3. Affectation du meilleur couple machine-ressource de transport.

La dernière règle, qui consiste à sélectionner le premier couple machine-ressource de transport qui permet à l'opération courante de se terminer le plus tôt possible, a donné les meilleurs résultats en termes de $C_{\max}$ sur l'ensemble des solutions générées pour les différentes représentations de solutions testées. En témoigne la figure 4.1 qui montre des ordonnancements pour l'instance illustrative présentée dans le chapitre 2 (tableaux 2.3 et 2.4) avec le codage `job` et les trois règles d'affectation. La séquence d'opérations de cette solution, y compris les opérations fictives de chargement et de déchargement au poste LU, est [2,1,1,2,1,2,1,1,2].

Avec la première règle d'affectation (1), pour chaque opération, la machine est choisie aléatoirement parmi les machines candidates et l'une des ressources de transport est également choisie de façon aléatoire. Ce qui explique le choix de M_3 et de R_2 pour $O_{2,1}$ avec cette règle contrairement aux règles (2) et (3). La différence entre les deux règles restantes se situe au niveau des choix pour $O_{2,2}$.

Avec la règle (2), la meilleure machine est d'abord sélectionnée, puis la meilleure ressource de transport est choisie. Dans ce cas précis, étant donné que l'opération $O_{2,1}$ se termine en 8, il serait préférable de commencer $O_{2,2}$ sur la machine M_1, qui est disponible, plutôt que sur M_2, dont la disponibilité ne survient qu'après l'achèvement de $O_{1,2}$, soit à l'unité de temps 12. Selon cette règle, dans le meilleur des cas (ce qui est souvent peu probable compte tenu des contraintes de transport), $O_{2,2}$ se terminerait en 14 si elle est attribuée à M_1, tandis qu'elle se terminerait en 16 si elle est attribuée à M_2. Cette explication justifie le choix de M_1 avec la règle (2). Ensuite le choix de la ressource de transport entraîne un décalage du début de $O_{2,2}$ qui se terminera finalement sur M_1 à la date 19.

La règle (3), bien qu'elle soit la plus coûteuse en termes de combinaisons à évaluer, sélectionne simultanément la machine et la ressource de transport qui minimisent en premier la valeur C_i du job concerné

à chaque étape de l'ordonnancement. Cette règle a été retenue pour les expérimentations futures. Cela implique que, à chaque étape du processus d'ordonnancement, le premier couple (M_k, R_h) qui termine l'opération courante au plus tôt est celui qui est sélectionné. Dans le cas des codages qui intègrent l'une des affectations dans leur structure, la règle ne traite que l'affectation qui est absente.

Parmi les dix codages examinés lors des tests, ceux qui intègrent l'affectation de machines, à l'instar de `osma`, génèrent fréquemment des solutions infaisables, quel que soit l'opérateur de voisinage utilisé. Cette infaisabilité découle du fait que les vecteurs MA sont alignés sur les vecteurs OS/TS, ce qui signifie qu'une opération dont la machine affectée ne fait pas partie des machines candidates entraîne une situation d'infaisabilité. Même si nous nous assurons que la solution initiale est réalisable, le vecteur MA est généré de manière aléatoire, ce qui limite potentiellement les mouvements faisables avec nos opérateurs de voisinage. En effet, la réaffectation des machines est restreinte aux permutations possibles du vecteur MA initial.

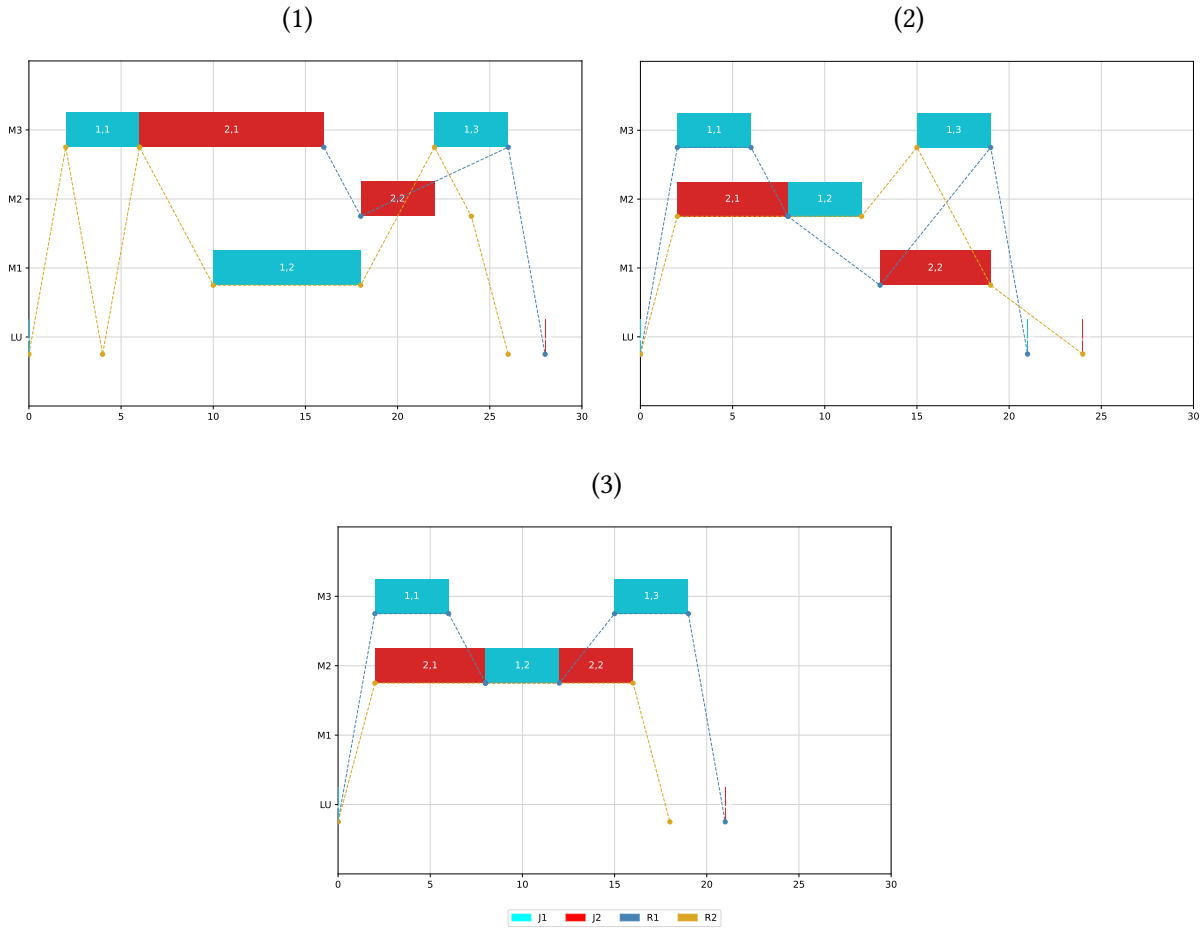


FIGURE 4.1 – Exemples d'ordonnancement avec différentes règles d'affectation

4.5 Espaces de recherche

Pour le FJSPT, nous avons conservé les mêmes indicateurs, ainsi que les trois méthodes pour explorer l'espace de recherche que nous avons utilisées pour le JSP au chapitre 3 : marche aléatoire, tirage aléatoire uniforme, recherche locale (voir section 3.4).

4.6 Instances FJSPT

Les expériences ont été menées sur 89 instances [1] issus de la littérature, comme indiqué dans le tableau 4.2. Dans toutes les instances, deux ressources de transport ont été considérées ($r = 2$). Pour prendre en compte le transport des jobs depuis/vers la station LU, deux opérations fictives sont ajoutées aux données de l'instance pour chaque job, l'une au début du job et l'autre à la fin. Ces opérations fictives ont un temps d'exécution nul et font référence à la machine virtuelle représentant la station LU (comme décrit dans la section 1.1.4).

Tableau 4.2 – Instances FJSPT

Préfixe	Nombre	n	m	Référence
`dn`	10	5,6,7,8	8	[Deroussi and Norre, 2010]
`exf`	4×7	5,6,8	4	[Kumar et al., 2011]
`sfjst, mfjst`	20	2,3,4,5,6,7,8,9,11,12	2,3,4,5,6,7,8	adapt. [Fattahi et al., 2007]
`mkt`	10	10,15,20	4,5,6,8,10,15	adapt. [Brandimarte, 1993]
`mt10t, setb4t, seti5t`	21	10,15	11,12,13,16,17,18	adapt. [Chambers and Barnes, 1996]

adapt. [*ref*] : ajout de données de transport (matrice de déplacement) à [*ref*] par [Homayouni and Fontes, 2021]

Le tableau 4.3 présente les rapports entre les temps de traitement $p_{i,j,k}$ et les temps de transport $t_{i,j,k}$ pour les instances regroupées par famille. Le ratio pour chaque instance est calculé selon la formule suivante :

$$Ratio = \frac{Moyenne\ des\ moyennes\ des\ temps\ de\ traitement\ p_{i,j,k} \times Taille\ de\ la\ matrice\ de\ transport\ sans\ la\ diagonale}{Somme\ des\ temps\ à\ vide\ et\ à\ charge\ de\ la\ matrice\ de\ transport}$$

Lorsque le ratio est égal à 1, les temps de traitement et de transport ayant le même ordre de grandeur, l'impact d'une tâche quelle qu'en soit la nature est équivalent sur la performance de l'ordonnancement. Cela correspond aux instances les plus contraintes. Lorsque le ratio est très petit ou très grand, l'impact sur les performances de l'ordonnancement est donc lié à la nature de la tâche. Cela peut amener à penser que le problème global pourrait être simplifié en prenant seulement en compte l'ordonnancement des opérations les plus longues. La variation des ratios en fonction des différentes familles d'instances requiert que les résultats à venir soient analysés également du point de vue de ces familles.

1. `https://fastmanufacturingproject.wordpress.com/2019/04/11/fjspt-instances`

Tableau 4.3 – Ratios entre les temps de traitement et de transport

Classe	Ratios
`dn`	1.20, 1.11, 1.18, 0.88, 0.82, 1.18, 0.85, 1.30, 1.26, 1.40
`exf`	0.46, 0.58, 0.55, 0.38, 0.45, 0.56, 0.53, 0.37, 0.34, 0.43, 0.40, 0.28, 0.33, 0.41, 0.39, 0.27, 0.35, 0.43, 0.41, 0.28, 0.53, 0.67, 0.63, 0.43, 0.51, 0.64, 0.60, 0.41
`sfjst`	2.26, 2.61, 4.30, 6.58, 2.36, 4.09, 3.84, 2.55, 2.76, 4.70
`mfjst`	4.11, 5.61, 5.98, 6.32, 6.17, 6.59, 7.43, 7.68, 7.86, 8.35
`mkt`	0.15, 0.14, 0.63, 0.26, 0.32, 0.25, 0.40, 0.70, 0.65, 0.56
`mt10t`	3.61, 3.50, 3.61, 3.50, 3.26, 3.50, 3.26
`setb4t`	3.64, 3.52, 3.64, 3.52, 3.51, 3.52, 3.28
`seti5t`	3.54, 3.47, 3.54, 3.47, 3.51, 3.47, 3.51

4.7 Caractérisation des paysages pour le FJSPT

Au sein de cette section, nous exposons les résultats relatifs à la rugosité, la neutralité, l'évolutivité, et la distribution de fitness pour chaque combinaison de codage et d'opérateur de voisinage. Ces résultats sont présentés à la fois de manière agrégée et de façon plus détaillée en examinant chaque famille d'instances.

4.7.1 Distance de corrélation

La figure 4.2 offre une perspective globale des distances de corrélation, triées par paire codage-voisinage, pour l'ensemble des instances. Les distances de corrélation sont plafonnées à 50 en arrondissant toutes les valeurs d'auto-corrélation supérieures à 0.98. Les distances sont réparties de façon variée en moyenne dans l'intervalle [0.5 - 15] pour les différents couples.

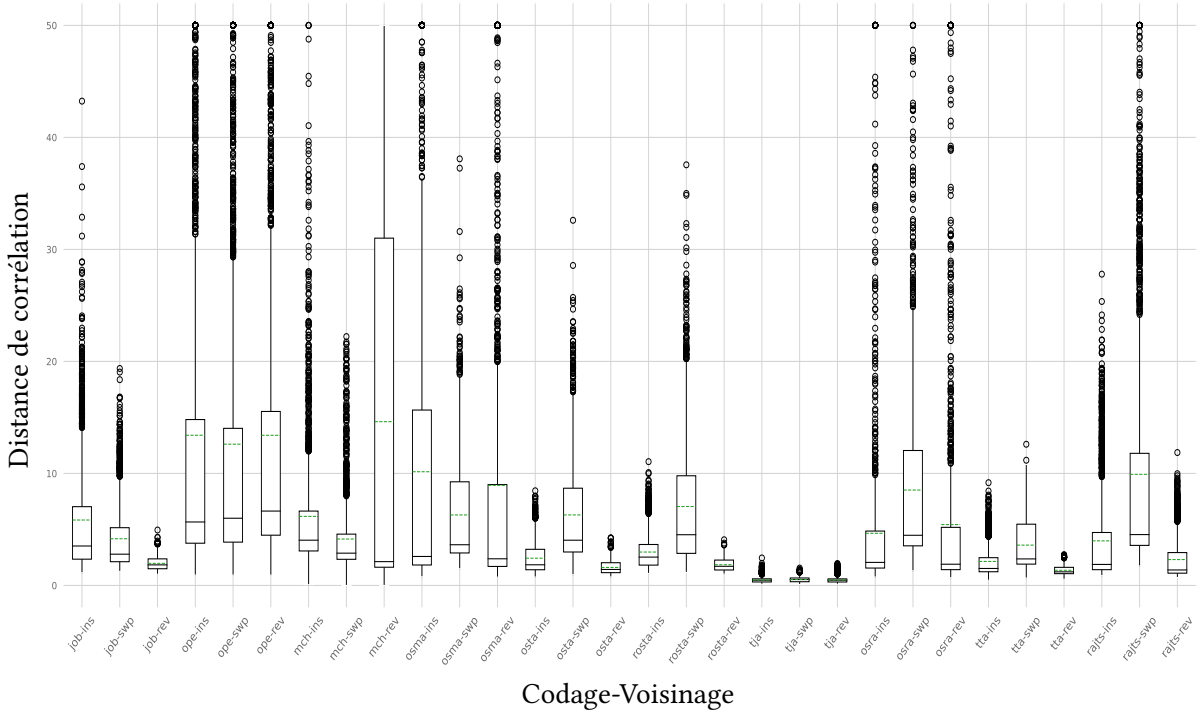


Figure 4.2 – Distance de corrélation pour les instances FJSPT

Des informations plus détaillées concernant la rugosité en relation avec les différentes familles d'instances sont fournies dans la figure 4.3. Les distances de corrélation les plus courtes sont enregistrées avec le codage `tja`, affichant un écart-type de seulement 0.25 au maximum sur les trois voisinages. En revanche, le codage `ope` présente généralement les valeurs de distances de corrélation les plus élevées, avec une variation tout aussi significative entre les familles d'instances. Ses moyennes se situent autour de 13 et les écarts-types autour de 14. Par ailleurs, le voisinage `rev` présente généralement une faible distance de corrélation par rapport à `ins` et `swp`, à l'exception des cas de `ope`, `mch` et `osma`. De plus, l'auto-corrélation augmente avec la taille de l'instance. Par rapport au JSP, nous observons une baisse notable de la distance de corrélation qui peut être expliquée par une augmentation des contraintes du problème. En effet, changer

un élément peut changer l'ordonnancement de manière significative à cause du maintien de la faisabilité liée au respect des contraintes supplémentaires.

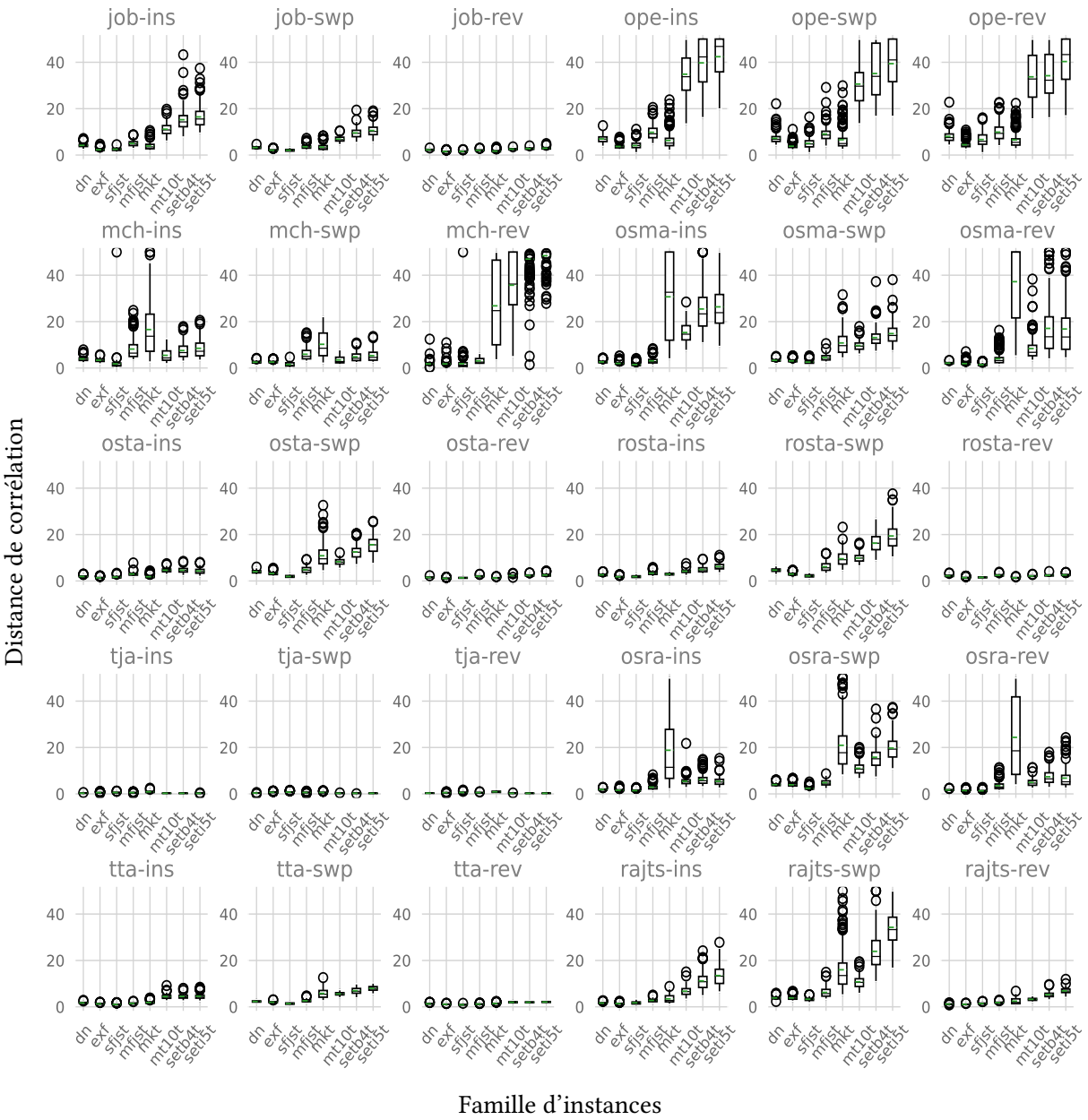


FIGURE 4.3 – Distance de corrélation regroupée par famille d'instances

4.7.2 Taux de neutralité

La figure 4.4 présente les taux de neutralité de toutes les instances sous forme de boîtes à moustaches, regroupées par les différentes paires codage-voisinage. Notablement, le codage `ope` affiche des taux de neutralité relativement élevés sur les trois voisinages, tandis que le codage `tja` enregistre les taux de neutralité les plus bas, suivi par le codage `tta`. Comme pour le JSP, la neutralité de `ope` avec le FJSPT s'explique par le nombre relativement élevé de solutions infaisables. En ce qui concerne le codage `tja`, la séquence d'opérations étant générée aléatoirement pour chaque solution lors de la marche, chaque itération produit des voisins avec des ordonnancements complètement différents. Cela met en évidence l'importance de la séquence d'opérations dans le codage au sein d'une heuristique.

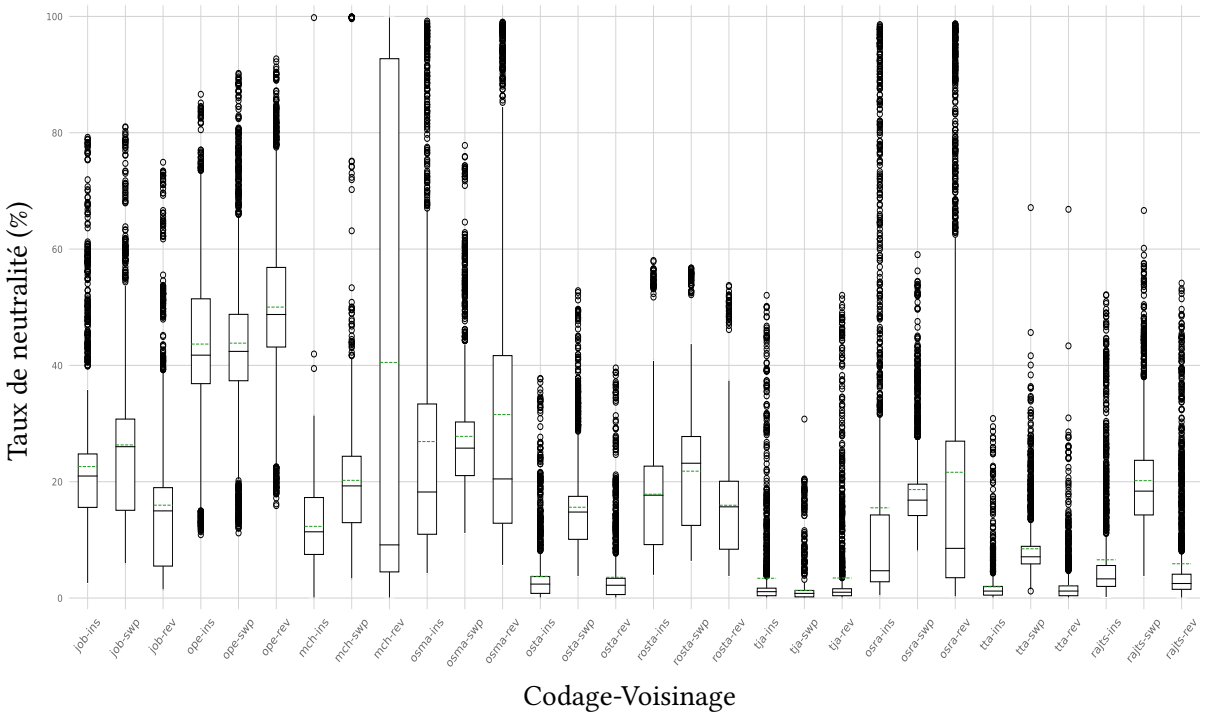


FIGURE 4.4 – Taux de neutralité pour les instances FJSPT

Afin de mettre en lumière les particularités propres aux instances d'une même famille, les résultats sont également illustrés pour les groupes d'instances provenant des différents benchmarks. Les taux de neutralité présentés dans la figure 4.5 sont regroupés en fonction des familles d'instances. Chaque sous-graphique correspond à une paire codage-voisinage spécifique. Ces taux varient en fonction du codage et de l'opérateur de voisinage appliqué. Dans les différentes combinaisons de codage-voisinage, les taux de neutralité présentent des inégalités, qui dépendent à la fois des familles d'instances et de leurs tailles. Les codages `osta`, `tja` et `tta` montrent des taux de neutralité inférieurs à 20% pour toutes les instances, à l'exception des instances `sfjst` et `mkt`, quel que soit le voisinage utilisé. En général, le codage `ope` maintient le plus grand dégré de neutralité, avec une moyenne comprise entre 43 et 50%, à l'exception des instances `mt10t`, `setb4t` et `seti5t`, où le codage `mch-rev` dépasse `ope` et atteint presque 100% dans certains cas. De plus, il apparait que la neutralité des instances `mkt` est fortement influencée par le choix des codages et des opérateurs de voisinage. Bien que certaines tendances globales puissent être observées

sur l'ensemble des instances en ce qui concerne les combinaisons codage-voisinage, il est difficile de tirer des conclusions générales en raison des variations considérables constatées pour différentes familles d'instances avec une même combinaison dans la quasi-totalité des cas.

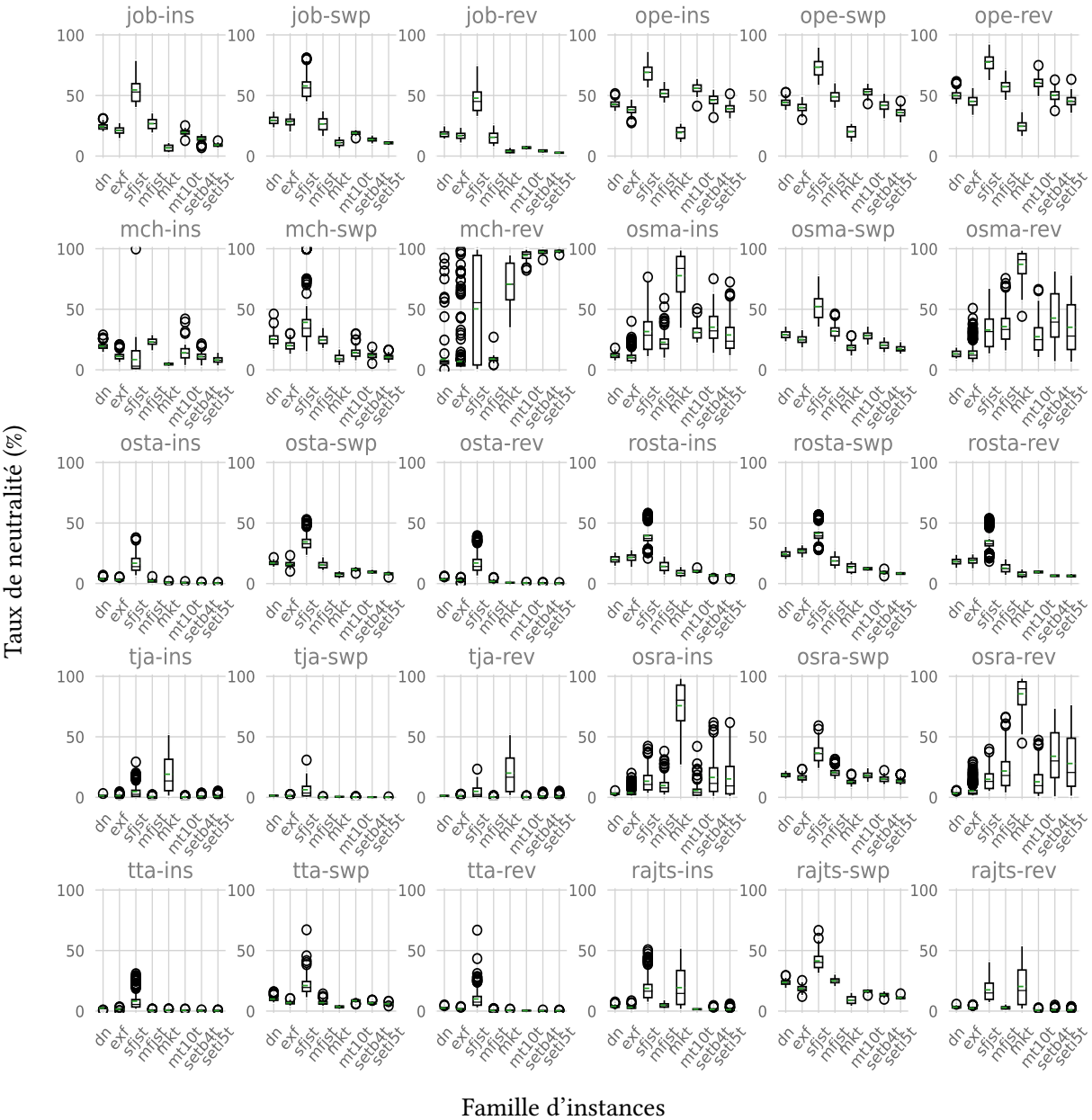


FIGURE 4.5 – Taux de neutralité regroupé par famille d'instances

4.7.3 Probabilité d'échappement cumulée

Les figures 4.6 et 4.7 montrent les probabilités d'échappement cumulées des paysages FJSPT. Tout comme pour les distances de corrélation et les taux de neutralité, sur la seconde figure, les probabilités sont regroupées par famille d'instances. Les probabilités d'échappement pour toutes les paires codage-voisinage se situent essentiellement dans une fourchette de $[0.2 - 0.5]$.

Les probabilités d'échappement pour les codages `osta`, `tja`, et `tta` sont généralement parmi les plus élevées, et les boîtes à moustaches les représentant, couvrant toutes les instances, sont compactes, avec un écart-type compris entre 0.02 et 0.04. Cette compacité est généralement due à la faible variabilité observée entre les différentes familles d'instances.

Globalement, le codage `osma` présente les probabilités les plus étalées et qui tirent plus vers le bas sur l'ensemble des trois voisinages. Cependant, lorsque l'on examine de plus près, la dispersion globale est attribuable aux variations plus ou moins importantes des probabilités d'échappement au regard des différentes familles d'instances. Pour tous les voisinages avec le codage `osma`, nous observons des valeurs relativement basses, d'environ 0.1, pour les familles `exf`, `sfjst` et `mfjst`, des valeurs moyennes pour les familles `dn` et `mkt` (entre 0.2 et 0.3), puis des valeurs plus élevées pour les familles `mt10t`, `setb4t` et `seti5t`, d'environ 0.4. Des tendances similaires sont observées avec le codage `osra`.

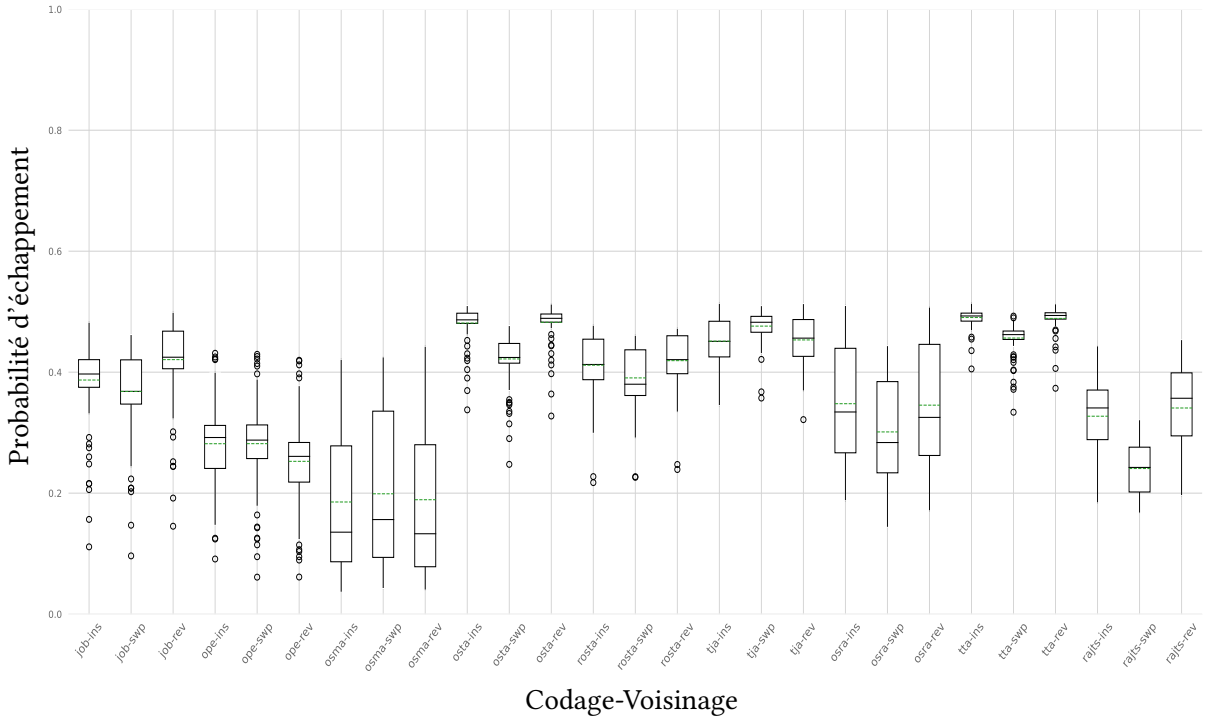


Figure 4.6 – Probabilité d'échappement cumulée pour les instances FJSPT

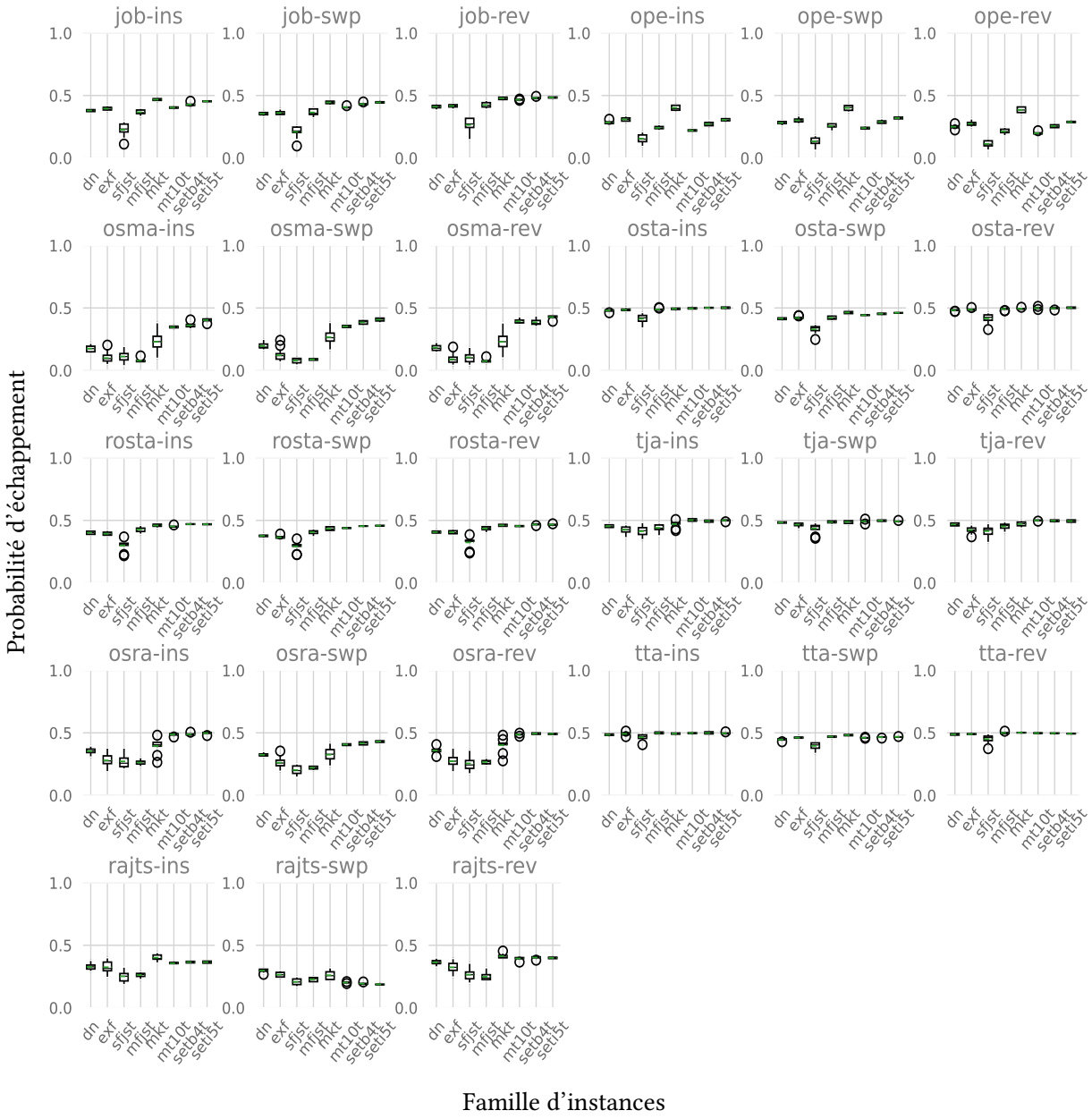


FIGURE 4.7 – Probabilité d'échappement cumulée regroupée par famille d'instances

4.7.4 Distribution des types de points

Pour chaque paire de codage et de voisinage, une synthèse de la distribution des types de points, couvrant l'ensemble des instances indépendamment de leur famille, est présentée dans le tableau 4.4. Les types de points les plus fréquents dans l'ensemble des résultats sont le SLOPE et le LEDGE. Le type SLOPE prédomine particulièrement pour les couples osta-ins, osta-rev, osra-ins, osra-rev, rajts-ins, rajts-rev, ainsi que pour les couples ayant tja ou tta comme codage. Il convient de noter que parmi ces couples, ceux utilisant le voisinage rev présentent des pourcentages légèrement plus élevés que ceux avec le voisinage ins. Les pourcentages obtenus pour le type IPLAT sont les plus marginaux pour toutes les paires, ce qui confirme les résultats obtenus pour les taux de neutralité.

Tableau 4.4 – Distribution des types de points (%)

	SLMIN	SLMAX	LMIN	LMAX	SLOPE	LEDGE	IPLAT
job-ins	0.4	0.4	6.8	6.0	18.7	67.4	0.4
job-swp	0.1	0.1	7.8	7.1	11.3	73.3	0.2
job-rev	1.2	1.1	6.0	5.8	39.2	46.6	0.0
ope-ins	0.0	0.1	10.6	9.0	0.7	78.2	1.4
ope-swp	0.0	0.1	10.2	10.1	0.6	76.6	2.5
ope-rev	0.0	0.1	12.8	13.3	0.2	70.0	3.6
osma-ins	26.6	0.0	15.8	0.2	18.7	38.3	0.4
osma-swp	0.4	0.0	37.2	0.2	2.0	60.1	0.2
osma-rev	20.8	0.0	22.6	0.1	17.5	38.8	0.1
osta-ins	5.7	5.3	0.9	0.8	81.7	5.5	0.0
osta-swp	0.2	0.1	2.3	1.8	29.8	65.8	0.0
osta-rev	4.6	3.8	0.7	0.8	84.4	5.7	0.0
rosta-ins	0.8	0.8	6.2	6.3	29.3	56.6	0.0
rosta-swp	0.0	0.0	3.0	3.0	18.2	75.8	0.0
rosta-rev	0.9	0.9	4.9	4.7	32.5	56.0	0.0
tja-ins	13.9	8.2	0.6	0.1	76.5	0.7	0.0
tja-swp	11.9	9.8	0.9	0.1	76.4	1.0	0.0
tja-rev	13.1	8.1	0.6	0.1	77.4	0.7	0.0
osra-ins	17.2	1.4	1.0	0.2	77.9	2.3	0.0
osra-swp	1.0	0.0	8.4	0.2	15.0	75.4	0.0
osra-rev	13.9	0.5	1.3	0.1	80.9	3.4	0.0
tta-ins	5.2	4.6	0.3	0.3	87.8	1.8	0.0
tta-swp	1.3	0.7	1.1	0.9	74.7	21.2	0.0
tta-rev	4.7	4.2	0.4	0.5	87.8	2.5	0.0
rajts-ins	18.1	0.8	2.2	0.4	74.3	4.3	0.0
rajts-swp	1.0	0.0	9.8	0.3	17.0	71.9	0.0
rajts-rev	14.5	0.8	1.7	0.2	79.1	3.7	0.0

Une illustration de cette répartition pour les différentes familles d'instances est fournie dans la figure 4.8. Une prédominance générale de LEDGE est observée pour un premier groupe de couples codage-voisinage sur l'ensemble des familles d'instances, tandis qu'un deuxième groupe montre une prédominance du type SLOPE. Le premier groupe comprend les couples job-ins, job-swp, ope-ins, ope-swp, ope-rev, osma-swp, osta-swp, rosta-swp, osra-swp et rajts-swp. Il est notable que swp est le voisinage le plus représentatif sur l'ensemble des couples mentionnés. Le deuxième groupe comprend les couples osta-ins, osta-rev, tja-ins, tja-swp, tja-rev, osra-ins, osra-rev, tta-ins, tta-swp, tta-rev, rajts-ins et rajts-rev. Les importantes quantités de SLOPE confirment la forte rugosité observée dans les distances de corrélation pour ces couples. En guise de rappel, LEDGE indique la présence de solutions de fitness égale, inférieure et supérieure dans le voisinage, tandis que SLOPE signifie un voisinage avec des solutions ayant des fitness supérieures et inférieures.

En dehors des deux grands groupes identifiés, certains couples présentent d'importantes variations d'une famille d'instances à une autre. C'est le cas, d'une part, des couples job-rev, rosta-ins, rosta-rev, pour lesquels les familles mkt, mt10t, setb4t et seti5t montrent une proportion plus élevée de SLOPE que de LEDGE. D'autre part, les couples osma-ins et osma-rev, malgré la prédominance de LEDGE dans les familles mt10t, setb4t et seti5t, montrent la présence des types de points SLMIN, LMIN, SLOPE, et exceptionnellement IPLAT avec une représentativité très marginale (car observée uniquement pour la famille de petites instances sfjst pour le couple osma-ins, avec 3.3%). Ces résultats indiquent la présence d'un nombre plus ou moins important d'optima locaux dans les paysages générés par les couples osma-ins et osma-rev avec les familles d'instances dn, exf, sfjst, mfjst et mkt.

De plus, nous observons également une quantité non négligeable de SLMIN et/ou SLMAX dans les paysages générés par les couples tja-ins, tja-swp, tja-rev, osra-ins, osra-rev, rajts-ins et rajts-rev, ce qui explique et confirme à nouveau la très forte rugosité et la très faible neutralité observées dans les expérimentations précédentes.

4.7.5 Synthèse sur la caractérisation des paysages

Le tableau 4.5 synthétise les résultats obtenus pour les combinaisons de codage et de voisinage en ce qui concerne la neutralité, la rugosité et l'évolutivité. Pour chaque métrique et pour chaque combinaison, les valeurs moyennes, médianes et les écarts-types sont calculés en agrégeant les données de l'ensemble des instances.

L'étude du paysage de fitness pour le FJSPT, en considérant les différentes combinaisons des 10 codages de solution et des 3 opérateurs de voisinage, a mis en évidence une fois de plus que la structure des espaces de recherche est profondément influencée à la fois par les codages et les opérateurs. Toutefois, il convient de souligner qu'avec certains codages tels que ope, tta et tja, les paysages semblent peu sensibles à l'impact de l'opérateur de voisinage, en se basant sur l'ensemble des métriques d'analyse. De plus, lorsqu'on évalue le taux de neutralité, la distance de corrélation et la probabilité d'échappement en regroupant les instances par famille, on observe souvent des résultats variés d'une famille à une autre. Cette constatation est corroborée par les écarts-types significatifs rapportés dans le tableau 4.5, ainsi que les différences marquées entre les valeurs moyennes et les valeurs médianes observées pour certaines combinaisons. Les résultats de la distribution des types de points viennent étayer davantage cette remarque. Ces observations mettent en lumière l'influence des distributions utilisées pour la génération des instances sur la structure du paysage.

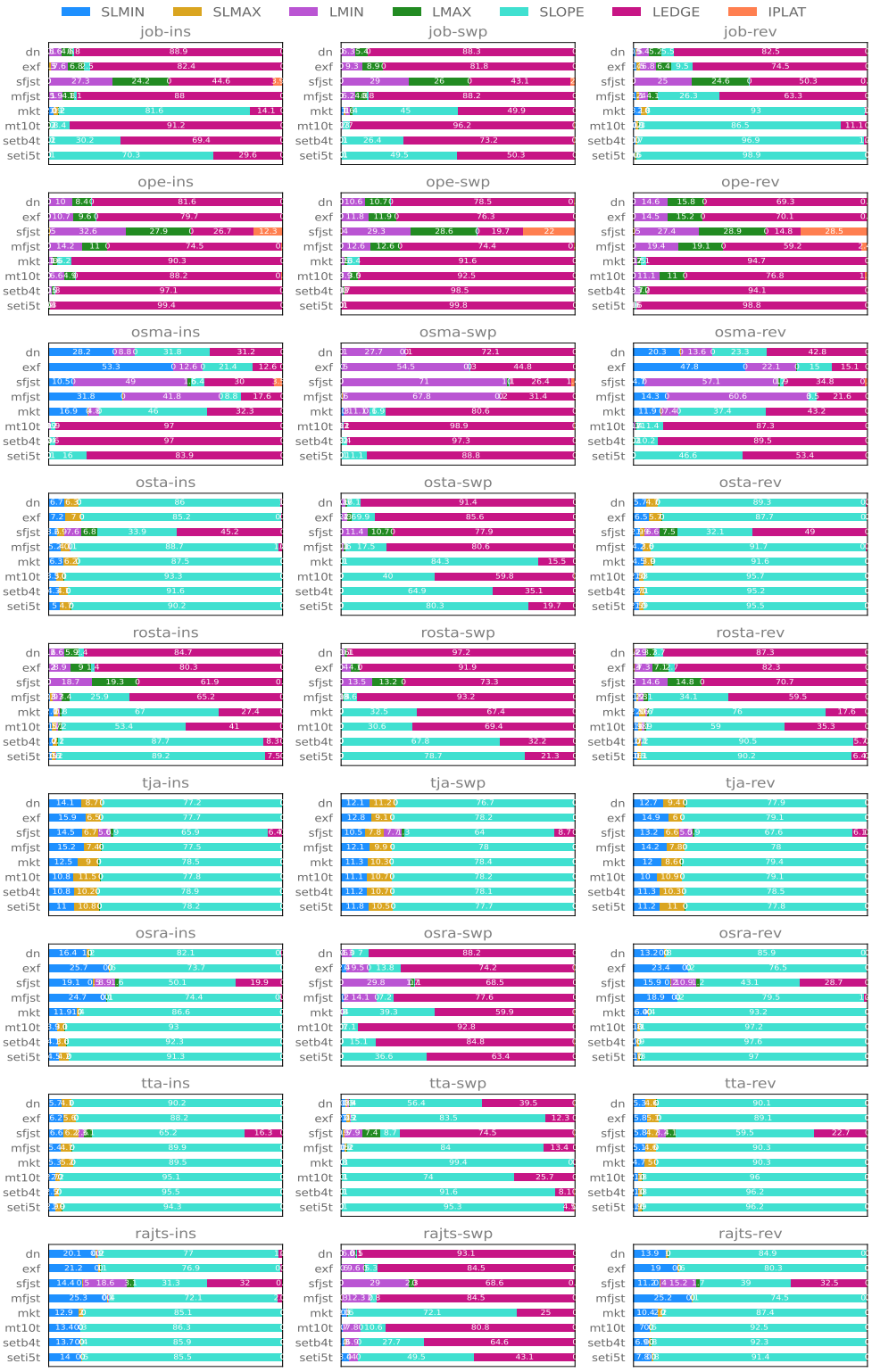


FIGURE 4.8 – Distribution des types de points regroupée par famille d'instances

Tableau 4.5 – Synthèse de l'analyse des paysages FJSPT : neutralité, rugosité, évolutivité

	Taux de neutralité			Dist. de corrélation			Prob. d'échappement		
	Moy.	Méd.	σ	Moy.	Méd.	σ	Moy.	Méd.	σ
job-ins	22.61	20.98	13.7	5.84	3.52	5.39	0.39	0.4	0.07
job-swp	26.33	26.02	14.0	4.17	2.79	3.08	0.37	0.37	0.07
job-rev	15.96	14.99	13.57	1.97	1.84	0.6	0.42	0.42	0.07
ope-ins	43.67	41.76	13.72	13.41	5.67	15.3	0.28	0.29	0.07
ope-swp	43.82	42.41	14.2	12.61	6.0	13.77	0.28	0.29	0.07
ope-rev	50.02	48.75	14.26	13.41	6.64	13.88	0.25	0.26	0.07
mch-ins	12.32	11.39	7.01	6.17	4.04	6.22	–	–	–
mch-swp	20.23	19.28	11.46	4.14	2.88	3.48	–	–	–
mch-rev	40.51	9.14	41.61	14.62	2.13	19.5	–	–	–
osma-ins	26.91	18.23	22.81	10.15	2.59	13.43	0.19	0.14	0.12
osma-swp	27.8	25.77	10.78	6.3	3.63	5.02	0.2	0.16	0.12
osma-rev	31.55	20.48	25.42	8.94	2.38	13.4	0.19	0.13	0.13
osta-ins	3.74	2.4	5.49	2.43	1.84	1.4	0.48	0.49	0.03
osta-swp	15.61	14.79	8.01	6.3	4.04	4.83	0.42	0.42	0.04
osta-rev	3.57	2.2	5.72	1.61	1.43	0.58	0.48	0.49	0.03
rosta-ins	17.85	17.68	10.29	2.98	2.53	1.53	0.41	0.41	0.05
rosta-swp	21.81	23.18	10.38	7.05	4.53	5.81	0.39	0.38	0.05
rosta-rev	15.95	15.68	9.14	1.85	1.71	0.54	0.42	0.42	0.05
tja-ins	3.4	1.1	7.88	0.49	0.47	0.25	0.45	0.45	0.04
tja-swp	1.33	0.8	2.86	0.53	0.53	0.23	0.48	0.48	0.02
tja-rev	3.45	1.0	8.24	0.49	0.47	0.25	0.45	0.46	0.04
osra-ins	15.52	4.7	23.99	4.66	2.06	7.53	0.35	0.33	0.1
osra-swp	18.65	16.83	7.58	8.52	4.48	8.02	0.3	0.28	0.08
osra-rev	21.62	8.54	27.24	5.43	1.9	9.19	0.35	0.33	0.1
tta-ins	1.93	1.2	3.31	2.14	1.52	1.44	0.49	0.49	0.02
tta-swp	8.47	7.09	5.45	3.6	2.37	2.38	0.46	0.46	0.03
tta-rev	2.13	1.2	3.94	1.35	1.21	0.44	0.49	0.49	0.02
rajts-ins	6.58	3.3	9.16	3.98	1.88	4.31	0.33	0.34	0.06
rajts-swp	20.18	18.38	9.51	9.92	4.54	10.47	0.24	0.24	0.04
rajts-rev	5.9	2.5	9.36	2.31	1.38	1.9	0.34	0.36	0.06

Dist. : Distance, Prob. : Probabilité, Moy. : Moyenne, Méd. : Médiane, σ : Ecart-type

Comparativement à l'analyse des paysages du JSP, le fait de rajouter des contraintes dans le FJSPT a tendance à tasser les valeurs des métriques calculées. En somme, en plus de l'impact des codages et des opérateurs, les caractéristiques propres aux instances jouent un rôle non négligeable dans la configuration des espaces de recherche pour le FJSPT.

4.8 Performance avec la recherche taboue

4.8.1 Résultats des tests

Pour évaluer la qualité de convergence des différentes combinaisons de codages de solution et d'opérateurs de voisinage, nous avons mis en œuvre une recherche taboue identique à celle décrite dans le chapitre précédent (chapitre 3). La figure 4.9 illustre le classement moyen obtenu par chaque paire en fonction de la valeur minimale de $C_{\max}$ calculée sur 10 exécutions de la recherche taboue pour l'ensemble des 89 instances. Ces rangs moyens sont calculés pour des durées CPU de 10s, 30s, 60s et 120s.

Bien qu'une amélioration continue ou une stabilité du rang moyen soit généralement attendue avec l'augmentation du temps CPU, nous observons, dans l'ensemble, de faibles variations d'un couple codage-voisinage à un autre. Cependant, quelques variations plus ou moins déterminantes sont à constater par endroit. Un exemple intéressant est illustré par le couple `rajts-swp`. À 10s de temps CPU, il se classe à la 9.18^{e} position en termes de rang moyen, mais au fil du temps, son classement s'améliore progressivement pour atteindre la 6.65^{e} position après 120s, dépassant ainsi les combinaisons `ope-swp` et `osma-swp`.

Le classement des 30 combinaisons de codages et de voisinages peut être divisé en quatre groupes distincts, comme détaillé dans le tableau 4.6. Le codage `job` en général obtient les meilleures performances, et la combinaison `job-ins` arrive en tête avec un rang moyen de 1.42 sur 120s de temps CPU. Le premier groupe est exclusivement composé des trois combinaisons associées au codage `job`. En privilégiant les 120s de temps CPU, le deuxième groupe rassemble des rangs moyens compris entre 3.98 et 13.69. Les combinaisons incluses sont, dans l'ordre : `osta-swp`, `ope-ins`, `rajts-swp`, `osma-swp`, `ope-swp`, `osta-ins`, `ope-rev`, `osta-rev`, `osra-swp`, `osma-ins`, `osma-rev` et `tta-swp`. Tous les voisinages avec les codages `ope`, `osta` et `osma` se retrouvent dans ce groupe, avec le voisinage `rev` systématiquement en dernière position. De la même manière, le troisième groupe englobe les combinaisons avec les codages `mch`, `tta`, `rajts`, `osra` et `rosta`, avec des rangs moyens compris entre 18.07 et 21.83. Enfin, le quatrième groupe est constitué des combinaisons avec les codages `tja` et `rosta`, incluant `tja-ins`, `tja-rev`, `tja-swp`, `rosta-ins` et `rosta-rev`, qui se classent en dernier en termes de rang moyen.

Par ailleurs, lorsque nous considérons que le codage reste fixe, nous pouvons observer les performances des opérateurs de voisinage. L'opérateur `swp` arrive en première position 6 fois sur 10, tandis que l'opérateur `ins` occupe cette place 4 fois sur 10. Quel que soit le codage utilisé, l'opérateur `rev` se classe systématiquement en dernière position. En ce qui concerne les performances des différents codages, avec un opérateur de voisinage fixe, le codage `job` est en tête, globalement suivi de par `osta` et `ope`, et ensuite `osma`. En revanche, les codages `rosta` et `tja` occupent généralement les dernières positions. Entre ces deux bords, nous retrouvons les codages `rajts`, `osra`, `tta` et `mch`.

La figure 4.10 présente les rangs moyens des paires de codages et de voisinages classées par famille d'instances. Nous constatons que, quelle que soit la famille, le codage `job`, et la combinaison `job-ins`, en particulier, affichent généralement les meilleures performances, confirmant ainsi les résultats plus

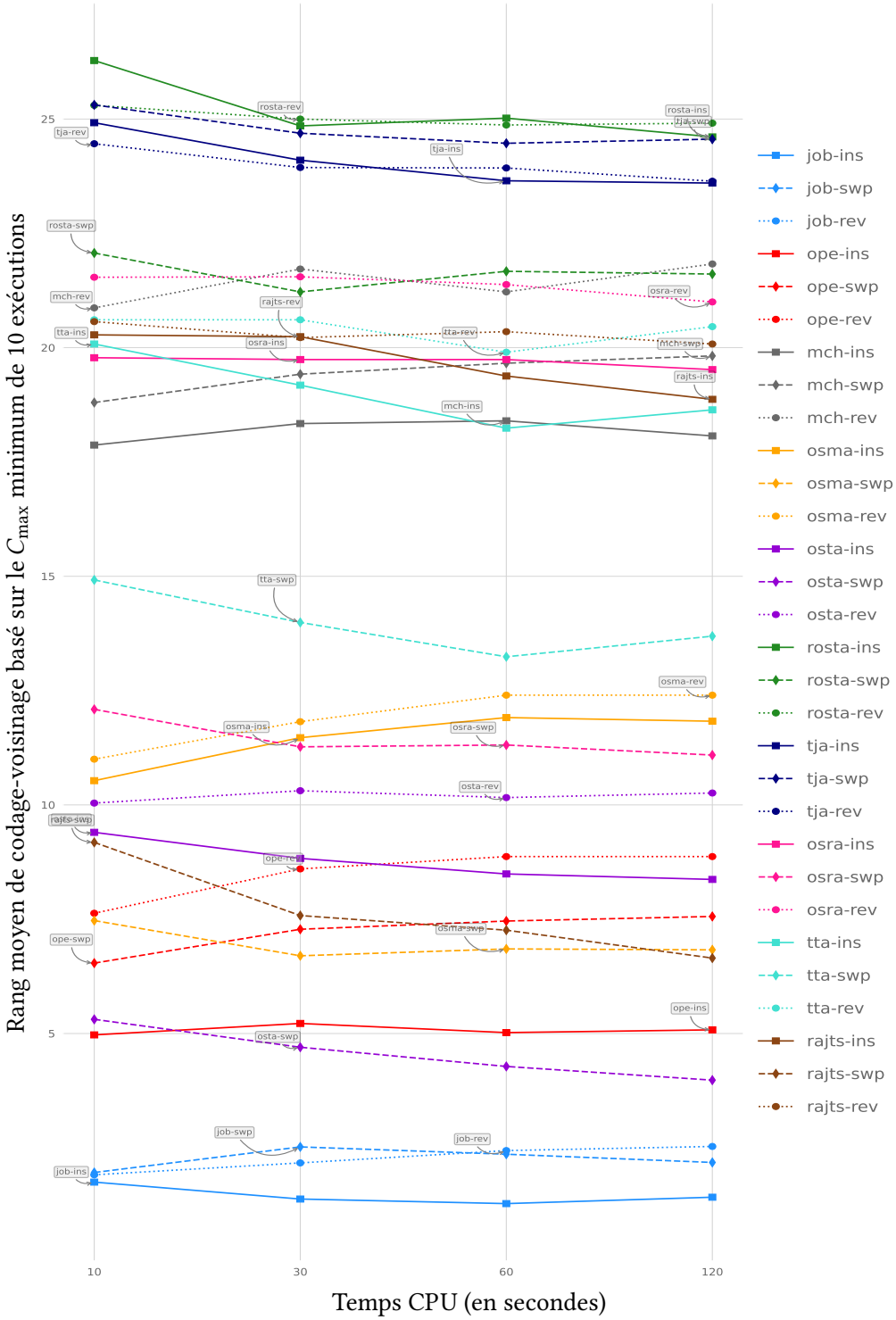


FIGURE 4.9 – Performance de codage-voisinage avec une recherche taboue pour les instances FJSPT

TABLEAU 4.6 – Rang moyen de codage-voisinage basé sur le $C_{\max}$ minimum de 10 exécutions avec une recherche taboue pour les instances FJSPT

Codage–Voisinage	10s	30s	60s	120s
job-ins	1.75	1.38	1.28	1.42
job-swp	1.96	2.52	2.36	2.18
job-rev	1.91	2.17	2.44	2.53
osta-swp	5.31	4.7	4.28	3.98
ope-ins	4.97	5.22	5.02	5.08
rajts-swp	9.18	7.58	7.26	6.65
osma-swp	7.47	6.7	6.85	6.83
ope-swp	6.54	7.28	7.46	7.56
osta-ins	9.4	8.83	8.49	8.37
ope-rev	7.63	8.6	8.87	8.87
osta-rev	10.04	10.31	10.16	10.26
osra-swp	12.09	11.27	11.31	11.09
osma-ins	10.53	11.47	11.91	11.83
osma-rev	11.0	11.82	12.4	12.4
tta-swp	14.92	13.99	13.24	13.69
mch-ins	17.87	18.34	18.4	18.07
tta-ins	20.08	19.18	18.24	18.64
rajts-ins	20.28	20.24	19.38	18.87
osra-ins	19.78	19.74	19.74	19.52
mch-swp	18.8	19.42	19.66	19.82
rajts-rev	20.57	20.22	20.35	20.08
tta-rev	20.61	20.61	19.9	20.46
osra-rev	21.54	21.55	21.38	21.0
rosta-swp	22.07	21.22	21.67	21.61
mch-rev	20.87	21.72	21.22	21.83
tja-ins	24.92	24.1	23.65	23.6
tja-rev	24.46	23.94	23.93	23.64
tja-swp	25.31	24.69	24.47	24.56
rosta-ins	26.28	24.85	25.02	24.61
rosta-rev	25.3	25.0	24.87	24.91

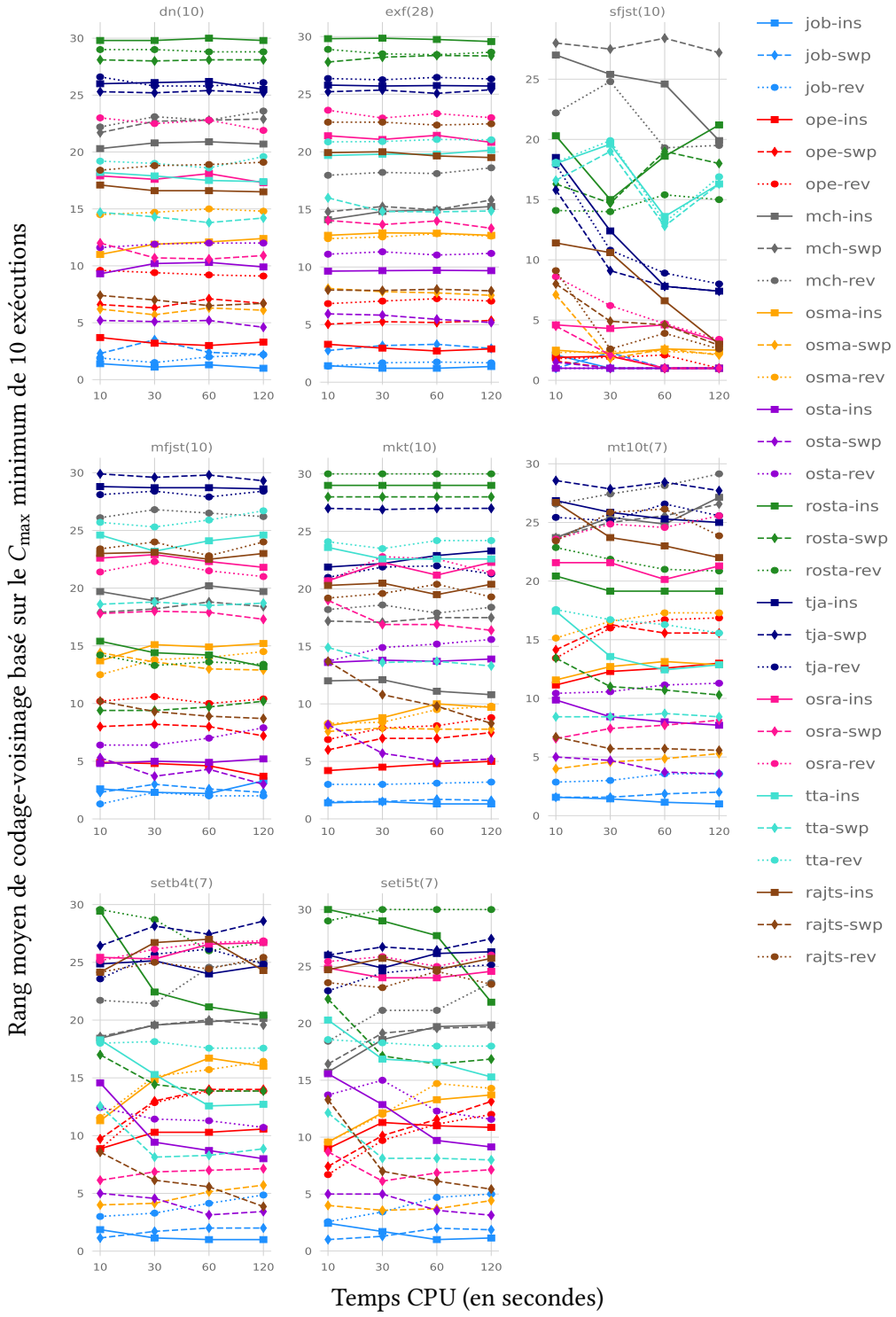


FIGURE 4.10 – Performance de codage-voisinage avec une recherche taboue pour les instances FJSPT regroupées par famille

globaux présentés précédemment. Les tendances observées dans l'approche globale, qui a été subdivisée en 4 groupes distincts, se retrouvent également dans certains classements par famille, notamment pour les instances `dn`, `exf` et `mkt`, bien que certaines variations légères soient observées. Pour les instances `mfjst`, les tendances dans le classement des différentes combinaisons suivent globalement la même trajectoire. Cependant, il est à noter que les performances des couples associés au codage `rosta` montrent des variations substantielles en termes de classement relatif. Il est intéressant de noter que les instances `mt10t`, `setb4t` et `seti5` présentent des classements quasi identiques pour les différentes combinaisons. De plus, pour ces trois familles d'instances, une segmentation claire en deux parties se dégage en regroupant le premier et le deuxième groupe, d'une part, et le troisième et le quatrième groupe, d'autre part. En ce qui concerne les instances `sfjst`, à l'exception des combinaisons associées aux codages `tja`, `tta`, `rosta` et `mch`, toutes les autres combinaisons se classent dans les quatre premiers rangs après 120s de temps CPU. Cette observation s'explique par la petite taille des instances `sfjst`, qui permet à la majorité des couples codage-voisinage d'atteindre rapidement la solution optimale.

Les valeurs de ratio entre les temps de traitement et les temps de transport dans les instances (voir tableau 4.3) peuvent expliquer les similitudes observées dans le classement des rangs moyens des couples codage-voisinage par rapport aux différentes familles d'instances. En effet, des similitudes sont observées dans la dispersion des classements pour les familles d'instances `dn`, `exf` et `mkt`, qui présentent des ratios allant de 0.14 à 1.20. D'autre part, le groupe des familles d'instances `mt10t`, `setb4t`, et `seti5t` montre des ratios compris entre 3.26 et 3.64. Quant aux familles `sfjst` et `mfjst`, les ratios sont plus dispersés, variant respectivement de 2.26 à 6.58 et de 4.11 à 8.35. Il est important de noter que les rangs moyens des paires de codage-voisinage pour ces deux dernières familles présentent des tendances différentes non seulement l'une par rapport à l'autre, mais aussi par rapport aux autres familles d'instances. Ces observations suggèrent que les différences de performances entre les huit familles d'instances semblent être liées aux rapports entre les temps de traitement et les temps de transport utilisés par ces instances.

4.8.2 Synthèse sur les performances

L'évaluation de la convergence des combinaisons codage-voisinage via une recherche taboue, sur des durées CPU allant de 10 à 120 secondes, montre que le codage `job` se révèle être le plus efficace pour résoudre les instances du FJSPT. Plus précisément, la combinaison `job-ins` se démarque en obtenant les meilleurs rangs moyens sur l'ensemble des familles d'instances au terme des 10 exécutions de l'algorithme. Sur les 89 instances testées, `job-ins` atteint un rang moyen global de 1.42 après 120s de recherche, décrochant le meilleur $C_{\max}$ 232 fois et le meilleur $C_{\max}$ moyen 70 fois (pour de plus amples détails sur les meilleurs $C_{\max}$ et les meilleurs $C_{\max}$ moyens, se référer à l'annexe A).

En ce qui concerne les rangs moyens, les combinaisons `job-swp` et `job-rev` suivent `job-ins`, et `osta-swp` occupe la quatrième position avec un rang moyen de 3.98, suivie à son tour par `ope-ins` à 5.08. Cependant, il est à noter que cette tendance entre `osta-swp` et `ope-ins` s'inverse lorsque l'on considère le nombre de fois où ces paires obtiennent le meilleur $C_{\max}$ et le meilleur $C_{\max}$ moyen. En effet, la combinaison `ope-ins` génère 161 meilleurs $C_{\max}$, comparé à 104 pour `osta-swp`. De même, en ce qui concerne les meilleurs $C_{\max}$ moyens, `ope-ins` obtient 17 fois le meilleur résultat, pendant que `osta-swp` atteint 9 occurrences. Cette inversion s'explique au regard des résultats regroupés par famille d'instance. Par exemple, sur les familles telles que `dn`, `exf`, `sfjst`, et `mfjst`, la combinaison `ope-ins` obtient un nombre plus élevé de meilleurs $C_{\max}$ que `osta-swp`. Toutefois, sur des familles d'instances telles que `mkt`, `mt10t`, `setb4t`, et `seti5t`, bien que ni `ope-ins` ni `osta-swp` n'obtiennent de meilleur $C_{\max}$ ou de

meilleur $C_{\max}$ moyen, les $C_{\max}$ enregistrés pour `osta-swp` surpassent nettement celles de `ope-ins`.

Les combinaisons impliquant les codages `tja` et `rosta` se classent parmi les moins performantes dans le cadre de l'algorithme de recherche taboue. En outre, de manière générale, l'opérateur de voisinage `rev` occupe la dernière position par rapport à `swp` et `ins`. Il atteint la deuxième place uniquement lorsqu'il est associé au codage `tja`, avec une légère différence par rapport à ses autres paires. De la même manière que pour le JSP, ces résultats confirment que le choix du codage et de l'opérateur de voisinage peut exercer un impact significatif sur les performances de l'algorithme de résolution dans le contexte du FJSPT.

4.9 Bilan

Dans ce chapitre, nous avons effectué une analyse de paysage de fitness et une étude de performance sur 89 instances du FJSPT pour 30 combinaisons de codages et d'opérateurs de voisinage.

Les résultats obtenus à partir de ces deux types d'expérimentation ne montrent pas de corrélation évidente entre les performances des couples codage-voisinage avec un algorithme de recherche taboue et les caractéristiques de leurs paysages respectifs. Un exemple pertinent de cette observation est fourni par les couples `job-rev` et `rosta-rev`. Les métriques relatives à l'analyse de paysage de fitness pour ces deux combinaisons ont généré des valeurs très similaires. Pourtant, parmi les 30 combinaisons testées, `job-rev` a décroché la troisième meilleure position en termes de rang moyen, tandis que `rosta-rev` se classe à la 30^e et dernière place.

La combinaison `job-ins`, qui s'est clairement démarquée comme la meilleure option pour le FJSPT, ne présente pas de caractéristiques particulières, si ce n'est des mesures moyennes en termes de rugosité, de neutralité et de probabilité d'échappement, avec une proportion significative de `LEDGE`, suivie de `SLOPE`. Les combinaisons `rosta-swp` et `mch-swp`, qui partagent pratiquement les mêmes caractéristiques que `job-ins` (uniquement rugosité et neutralité dans le cas de `mch-swp`), ont affiché des performances relativement médiocres. Il convient cependant de noter que, bien que les deux codages `job` et `rosta` ne génèrent que des solutions faisables, le codage `rosta` encode deux informations, à savoir le séquencement des opérations et l'affectation des ressources de transport, tandis que `job` ne code que le séquencement des opérations. En outre, nous rappelons que les codages qui génèrent exclusivement des solutions faisables sont `job`, `osta`, `rosta`, et `tta`. Toutefois, lorsque l'on considère l'ensemble des opérateurs de voisinage, ils ne se retrouvent pas systématiquement en tête de liste. Cela démontre que la simple faisabilité des solutions n'est pas suffisante pour déterminer les codages les plus performants pour le FJSPT.

Il semble également que la qualité des codages dépend davantage du type d'informations qu'ils encodent que de la quantité d'informations. Les informations encapsulées dans les représentations des solutions pour le FJSPT concernent le séquencement des opérations (OS), l'affectation des machines (MA), le séquencement des tâches de transport (TS), et l'affectation des ressources de transport (TA). Le séquencement des opérations est sans aucun doute l'information prépondérante, car dans tout moteur d'ordonnancement classique (glouton), l'OS détermine l'ordre dans lequel les opérations sont effectuées sur les machines. Les codages comme `job` et `ope`, qui se concentrent exclusivement sur l'OS, obtiennent de bonnes performances (malgré la présence d'infaisabilité avec `ope`). À l'inverse, les représentations qui encodent séquentiellement ou pas du tout l'OS, tels que `mch`, `tta` et `tja`, affichent des performances bien inférieures. Étant donné que le séquencement des opérations et le séquencement des tâches de transport peuvent se compenser en l'absence de l'un ou de l'autre, la criticité du TS est aussi clairement perceptible en l'absence de l'OS. Cette

disposition explique la supériorité des performances du codage `tta` par rapport au codage `tja`. Plus loin, en raison des problèmes d'infaisabilité causés par l'encodage de l'affectation des machines, sa présence semble plus critique que celle de l'affectation des ressources de transport. Cela est particulièrement vrai dans la variante FJSPT étudiée, où toutes les ressources de transport peuvent se déplacer vers n'importe quelle position et transporter n'importe quel job, alors que toutes les machines ne peuvent pas traiter toutes les opérations. Les résultats des codages `osma` et `osta` confirment cette observation. Cette observation montre l'intérêt que pourrait avoir l'utilisation d'un opérateur de voisinage de type réaffectation sur le vecteur MA afin de garantir l'obtention d'affectations faisables pour les machines.

Par ailleurs, il est intéressant de noter que bien que les codages `osta` et `rosta` encapsulent exactement les mêmes types d'informations, notamment l'OS et la TA, leurs performances diffèrent considérablement. Lorsque nous les examinons de plus près, nous constatons que `rosta` utilise deux vecteurs de réels pour coder une solution, tandis que `osta` le fait de manière plus directe. Ces vecteurs de réels passent par un système de décodage qui les convertit en une forme équivalente à `osta` lors du processus d'ordonnancement. Cette forme initiale du codage semble impacter négativement les performances dans ce cas particulier, possiblement en raison du système de décodage spécifique utilisé. Par analogie, si nous considérons les similitudes entre les codages `osra` et `tta`, avec une correspondance entre l'OS de `osra` et le TS de `tta`, nous observons un écart moindre entre les performances de ces deux codages par rapport à `osta` et `rosta`. Le codage `tta` utilise des vecteurs de réels tout comme `rosta`, mais le système de décodage diffère pour les deux. Il est donc concevable que le mécanisme de décodage utilisé par le codage `rosta` soit un inconvénient pour ses performances.

En somme, ces résultats montrent que l'analyse du paysage de fitness, à travers les métriques utilisées dans cette étude, ne peut pas à elle seule prédire les performances des couples codage-voisinage dans le contexte des problèmes de type job shop, que ce soit pour le JSP ou le FJSPT. Pour les couples de codage-voisinage présents dans les deux variantes, bien que les valeurs des métriques d'analyse du paysage ne présentent pas les mêmes tendances, le classement des performances est identique, du moins du point de vue des codages (`job`, `ope`, `mch`). Le couple `job-ins` demeure la meilleure option dans les deux cas. Il est néanmoins intéressant de noter que le codage `job` n'est pas à priori un choix intuitif pour le FJSPT, car il concerne uniquement le séquencement des opérations. Son efficacité est également liée à la qualité de la règle d'affectation des machines et des ressources de transport intégrée au moteur d'ordonnancement.

Conclusion générale

Au cours des dernières années, de nombreuses méthodes de résolution approchées ont été développées pour relever les défis de l'ordonnancement d'atelier de production. Cependant, peu d'études se consacrent de manière approfondie à l'analyse des composants de ces algorithmes, bien que ces derniers puissent considérablement influencer leurs performances. Ce constat est particulièrement valable pour les métaheuristiques proposées dans le cadre des problèmes de type job shop. Parmi les composants classiques des métaheuristiques, les codages de solutions et les opérateurs de voisinage s'avèrent être des plus essentiels, car ils définissent, d'une part, l'espace des solutions et, d'autre part, les relations de voisinage entre ces solutions.

Partant de cette observation, nous avons adopté une méthodologie fondée sur l'hypothèse selon laquelle des composants appropriés confèrent à une métaheuristique des meilleures performances pour un problème donné. Le job shop, dans sa variante de base, se présente comme un problème d'ordonnancement d'atelier, impliquant la programmation d'opérations de jobs avec des contraintes de précédence entre elles. Pour aligner davantage la définition du problème sur les environnements de production modernes, des contraintes supplémentaires peuvent être ajoutées pour créer d'autres variantes. Cette thèse s'est intéressée spécifiquement à la variante de base du problème de job shop (JSP) et à sa variante flexible avec ressources de transport (FJSPT). Afin de vérifier l'hypothèse de base de ce travail, nous avons d'abord mesuré la pertinence de divers couples de codages de solutions et d'opérateurs de voisinage classiques, en mettant en lumière les propriétés inhérentes aux espaces de recherche qu'ils génèrent. Ensuite, nous avons évalué les performances de ces couples codage-voisinage à l'aide d'une métaheuristique de recherche locale, cherchant ainsi à identifier d'éventuelles corrélations avec les propriétés extraites de la première étape.

Nous avons matérialisé la première étape ci-dessus mentionnée par une analyse de paysage de fitness. Cette analyse nous a permis de mesurer des propriétés de paysage telles que la rugosité, la neutralité, l'évolutivité et la distribution des fitness pour les différentes combinaisons de codages et d'opérateurs de voisinage. L'expérimentation, menée à la fois sur le JSP et le FJSPT, a confirmé que ces deux composants ont une incidence réelle sur les espaces de recherche des problèmes de type job shop. Cependant, il est à noter que, pour une même combinaison, les caractéristiques des paysages varient d'une variante à une autre. Dans le cas du FJSPT, par exemple, nous observons des niveaux de rugosité et des taux de neutralité relativement bas par rapport à ceux du JSP. Ainsi, l'introduction de la flexibilité de machines et des contraintes de transport change sensiblement la structure des espaces de recherche du job shop. De plus, nous avons remarqué que les modes de génération des instances exercent une influence plus marquée sur les paysages du FJSPT que sur ceux du JSP.

Dans la deuxième étape, nous avons évalué la qualité des $C_{\max}$ obtenus par les mêmes combinaisons de

codages et d'opérateurs de voisinage précédemment analysées, sur un algorithme de recherche taboue. Pour les deux variantes du problème, le couple associant le codage job (`job`) et l'opérateur d'insertion (`ins`) a obtenu les meilleures performances. Toutefois, sur la base de l'ensemble des résultats, aucune corrélation formelle n'a pu être déterminée de la confrontation des performances des différents couples et des propriétés de leurs paysages respectifs. L'analyse de paysage de fitness, à travers les métriques que nous avons calculées, ne permet donc pas à elle seule de prédire de manière fiable les performances des différents couples codage-voisinage pour les problèmes de job shop. Elle peut néanmoins fournir des éléments pour expliquer les performances médiocres pour les cas où les métriques obtiennent des valeurs extrêmes. Une neutralité trop élevée par exemple n'augure pas de bonnes performances pour la métaheuristique. L'inefficacité des métriques est également due au fait que la définition de ce qu'est une valeur moyenne ou une bonne valeur reste inconnue. Dans ce contexte, nous avons entrepris une analyse multidimensionnelle pour rechercher des règles d'association [Piatetsky-Shapiro, 1991; Agrawal et al., 1993], en nous basant sur les données des instances et les résultats obtenus tant sur l'analyse de paysage que sur la recherche taboue. À celà nous avons adjoint une analyse en composantes principales. Les résultats issus de l'étude préliminaire ont confirmé l'absence de corrélation ou d'anti-corrélation représentative à l'égard des métriques d'analyse de paysage utilisées.

En conclusion, avons-nous apporté une réponse aux deux questions ayant motivé nos travaux ? Pour rappel, pour la première question, il s'agissait d'établir si l'analyse du paysage de fitness peut conduire à des résultats ou des propriétés significatives. Malheureusement, malgré le nombre important de métriques de la littérature, celles utilisables pour les problèmes de type job shop, qu'ils soient de base ou contraints, semblent ne pas permettre d'apporter une réponse satisfaisante à cette question, puisque les performances des combinaisons de codages et d'opérateurs de voisinage ne peuvent pas être corrélées aux propriétés du paysage, que ce soit individuellement ou dans leur ensemble. Pour autant, et cela renvoie à la seconde question, cela nous empêche-t-il d'émettre des recommandations pour un choix judicieux de couples codage–voisinage lors de la conception de métaheuristiques pour le problème de job shop et par extension pour sa variante plus complexe FJSPT ? Certes, la comparaison des résultats entre le problème de base et sa variante plus contrainte soulève des réserves quant à une généralisation systématique des caractéristiques des codages et des opérateurs de voisinage pour cette catégorie de problème d'optimisation. Mais, à ce stade de notre étude, il nous semble pertinent de préconiser le couple `job-ins`, qui s'est révélé parmi les associations les plus performantes pour l'objectif de minimisation du makespan, avec une très bonne stabilité au changement d'instance et de variante. Pour revenir à notre analyse initiale de la littérature (chapitre 2), ce couple n'est pourtant pas le plus employé, puisque les codages les plus utilisés pour le JSP sont a priori `job` et `mch`, qui sont le plus fréquemment associés avec les opérateurs `cas` et `swp`. Ainsi, a contrario, nous déconseillerions le couple `mch-cas` qui s'est avéré intéressant mais moins stable car plus dépendant des instances. Quant au couple `mch-swp`, il est à éviter pour les deux variantes testées, ce qui est plutôt en désaccord avec la littérature du JSP.

Compte tenu des résultats que nous avons obtenus, nous avons identifié plusieurs directions envisageables pour de futurs travaux, que nous explicitons brièvement ci-après.

La première piste de recherche est que certains codages, non retenus dans notre étude, pourraient faire l'objet de nouvelles investigations. Le nouveau codage `tim` que nous avons introduit s'est révélé performant pour le JSP, ce qui a montré que les codages basés sur le temps ne sont pas à négliger, alors qu'ils sont peu utilisés dans la littérature. Tester `tim` avec les quatre moteurs d'ordonnancement qui en découlent pour le

FJSPT pourrait donner des résultats tout aussi intéressants.

De même que pour le codage, la liste des opérateurs de voisinage utilisés dans nos expérimentations concernant le FJSPT est restreinte par rapport à la littérature. En utilisant des opérateurs de voisinage adaptés à chaque couche des codages à plusieurs niveaux de décision (séquencement des opérations, affectation des machines, séquencement des tâches de transport, affectation des ressources de transport), de meilleures performances pourraient être observées pour les codages testés.

L'affinement de l'analyse du paysage de fitness pourrait donner un autre aperçu des paysages, à travers l'identification ou la création de nouvelles métriques compatibles avec les problèmes de job shop. L'analyse de paysage de fitness étant un outil analytique reposant sur l'empirisme, d'autres formes d'analyse plus théoriques telles que l'analyse numérique présentée dans [Allaire, 2012] ou encore des méthodes d'ingénierie robuste [Fowlkes and Creveling, 2000] peuvent également être considérées.

Enfin, l'infaisabilité des solutions n'a pas été exploitée dans notre thèse. Si les méthodes de la littérature cherchent souvent à ne générer que des solutions faisables en éliminant le plus en amont possible les autres solutions, il existe également des études qui montrent l'intérêt de l'infaisabilité pour l'exploration des espaces de recherche. L'exploitation des solutions infaisables sur des méthodes à base de population pourrait accroître les performances des codages qui en génèrent.

Bibliographie

[Aarts and Lenstra, 2003] Aarts, E. H. L. and Lenstra, J. K. (2003). *Local Search in Combinatorial Optimization*, chapter Introduction, pages 1–18. Princeton University Press.

[Abdelmaguid, 2010] Abdelmaguid, T. F. (2010). Representations in genetic algorithm for the job shop scheduling problem : A computational study. *Journal of Software Engineering and Applications*, 3(12) :1155–1162.

[Abdelmaguid, 2015] Abdelmaguid, T. F. (2015). A neighborhood search function for flexible job shop scheduling with separable sequence-dependent setup times. *Applied Mathematics and Computation*, 260 :188–203.

[Abderrahim et al., 2022] Abderrahim, M., Bekrar, A., Trentesaux, D., Aissani, N., and Bouamrane, K. (2022). Bi-local search based variable neighborhood search for job-shop scheduling problem with transport constraints. *Optimization Letters*, 16(1) :255–280.

[Adams et al., 1988] Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34(3) :391–401.

[Afsar et al., 2016] Afsar, H. M., Lacomme, P., Ren, L., Prodhon, C., and Vigo, D. (2016). Resolution of a job-shop problem with transportation constraints : a master/slave approach. *IFAC-PapersOnLine*, 49(12) :898–903.

[Agrawal et al., 1993] Agrawal, R., Imieliński, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2) :207–216.

[Al-Ashhab et al., 2023] Al-Ashhab, M. S., Alhejaili, A. F., and Munshi, S. M. (2023). Developing a multi-objective flexible job shop scheduling optimization model using lexicographic procedure considering transportation time. *Journal of Umm Al-Qura University for Engineering and Architecture*, 14(1) :57–70.

[Al-Betar, 2017] Al-Betar, M. A. (2017). β-hill climbing : an exploratory local search. *Neural Computing and Applications*, 28(1) :153–168.

[Allaire, 2012] Allaire, G. (2012). *Analyse numérique et optimisation : une introduction à la modélisation mathématique et à la simulation numérique*. 2 edition.

[Anderson et al., 2003] Anderson, E. J., Glass, C. A., and Potts, C. N. (2003). *Local Search in Combinatorial Optimization*, chapter Machine scheduling, pages 361–414. Princeton University Press.

[Applegate and Cook, 1991] Applegate, D. and Cook, W. (1991). A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2) :149–156.

[Balin, 2011] Balin, S. (2011). Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications*, 38(6) :6814–6821.

[Barnett, 2001] Barnett, L. (2001). Netcrawling-optimal evolutionary search with neutral networks. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, pages 30–37. IEEE Xplore.

[Bean, 1994] Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2) :154–160.

[Bierwirth, 1995] Bierwirth, C. (1995). A generalized permutation approach to job shop scheduling with genetic algorithms. *Operations-Research-Spektrum*, 17(2) :87–92.

[Bierwirth et al., 2004] Bierwirth, C., Mattfeld, D., and Watson, J.-P. (2004). Landscape regularity and random walks for the job-shop scheduling problem. In Gottlieb, J. and Raidl, G., editors, *Evolutionary Computation in Combinatorial Optimization, 4th European Conference*, pages 21–30. Springer-Verlag Berlin Heidelber.

[Bierwirth and Mattfeld, 1999] Bierwirth, C. and Mattfeld, D. C. (1999). Production scheduling and rescheduling with genetic algorithms. *Evolutionary Computation*, 7(1) :1–17.

[Bierwirth et al., 1996] Bierwirth, C., Mattfeld, D. C., and Kopfer, H. (1996). On permutation representations for scheduling problems. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature — PPSN IV*, pages 310–318.

[Borne et al., 2013] Borne, P., Popescu, D., Filip, F.-G., and Stefanoiu, D. (2013). *Optimisation en sciences de l'ingénieur : Méthodes exactes*. Lavoisier.

[Boschetti et al., 2009] Boschetti, M. A., Maniezzo, V., Roffilli, M., and Röhler, A. B. (2009). Matheuristics : Optimization, simulation and control. In Blesa, M. J., Blum, C., Gaspero, L., Roli, A., Sampels, M., and Schaerf, A., editors, *Hybrid Metaheuristics*, pages 171–177. Springer Berlin Heidelberg.

[Brandimarte, 1993] Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3) :157–183.

[Burke and Kendall, 2014] Burke, E. K. and Kendall, G. (2014). *Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques*, chapter Introduction, pages 1–17. Springer, Boston, MA, 2 edition.

[Caldeira et al., 2020] Caldeira, R. H., Gnanavelbabu, A., and Vaidyanathan, T. (2020). An effective backtracking search algorithm for multi-objective flexible job shop scheduling considering new job arrivals and energy consumption. *Computers & Industrial Engineering*, 149 :106863.

[Carlier and Chrétienne, 1988] Carlier, J. and Chrétienne, P. (1988). *Problèmes d'ordonnancement : modélisation, complexité, algorithmes*. 1 vol. (326 p.). Masson.

[Carter and Ragsdale, 2006] Carter, A. E. and Ragsdale, C. T. (2006). A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European Journal of Operational Research*, 175(1) :246–257.

[Chambers and Barnes, 1996] Chambers, J. B. and Barnes, J. W. (1996). Flexible job shop scheduling by tabu search. *Graduate Program in Operations Research and Industrial Engineering, Department of Mechanical Engineering, The University of Texas at Austin, Technical Report Series ORP 96-09*.

[Chen et al., 2020] Chen, X.-L., Li, J.-Q., Han, Y.-Y., and Sang, H.-Y. (2020). Improved artificial immune algorithm for the flexible job shop problem with transportation time. *Measurement and Control*, 53(9–10) :2111–2128.

[Chen et al., 2023] Chen, Z., Zou, J., and Wang, W. (2023). Digital twin-oriented collaborative optimization of fuzzy flexible job shop scheduling under multiple uncertainties. *Sādhanā*, 48(2) :78.

[Cheng et al., 1996] Cheng, R., Gen, M., and Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & Industrial Engineering*, 30(4) :983–997.

[Choi and Choi, 2002] Choi, I.-C. and Choi, D.-S. (2002). A local search algorithm for jobshop scheduling problems with alternative operations and sequence-dependent setups. *Computers & Industrial Engineering*, 42(1) :43–58.

[Collard et al., 2006] Collard, P., Mauri, G., Pirola, Y., Tomassini, M., Vanneschi, L., and Verel, S. (2006). A quantitative study of neutrality in gp boolean landscapes. In *Genetic And Evolutionary Computation Conference*, pages 895–902. ACM Press.

[Czogalla and Fink, 2012] Czogalla, J. and Fink, A. (2012). Fitness landscape analysis for the no-wait flow-shop scheduling problem. *Journal of Heuristics*, 18 :25–51.

[Dai et al., 2019] Dai, M., Tang, D., Giret, A., and Salido, M. A. (2019). Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints. *Robotics and Computer-Integrated Manufacturing*, 59 :143–157.

[Dauzère-Pérès et al., 2023] Dauzère-Pérès, S., Ding, J., Shen, L., and Tamssaouet, K. (2023). The flexible job shop scheduling problem : A review (in press). *European Journal of Operational Research*.

[Demirkol et al., 1998] Demirkol, E., Mehta, S., and Uzsoy, R. (1998). Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1) :137–141.

[Deroussi and Norre, 2010] Deroussi, L. and Norre, S. (2010). Simultaneous scheduling of machines and vehicles for the flexible job shop problem. *International conference on metaheuristics and nature inspired computing*, pages 1–2.

[Dhiflaouia et al., 2018] Dhiflaouia, M., Nouri, H. E., and Driss, O. B. (2018). Dual-resource constraints in classical and flexible job shop problems : A state-of-the-art review. *Procedia Computer Science*, 126 :1507–1515.

[Du et al., 2022] Du, Y., Li, J., Li, C., and Duan, P. (2022). A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15.

[Durasević and Jakobović, 2016] Durasević, M. and Jakobović, D. (2016). Comparison of solution representations for scheduling in the unrelated machines environment. *39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1336–1342.

[Esquirol and Lopez, 1999] Esquirol, P. and Lopez, P. (1999). *L'ordonnancement*. Economica.

[Esquirol and Lopez, 2008] Esquirol, P. and Lopez, P. (2008). *Production scheduling*, volume 1, chapter 2, Basic Concepts and Methods in Production Scheduling, pages 5–32. ISTE Ltd and John Wiley & Sons, Inc., hermès science publications edition.

[Fattahi et al., 2007] Fattahi, P., Mehrabad, M. S., and Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3) :331–342.

[Fernandez-Viagas et al., 2019] Fernandez-Viagas, V., Perez-Gonzalez, P., and Framinan, J. M. (2019). Efficiency of the solution representations for the hybrid flow shop scheduling problem with makespan objective. *Computers & Operations Research*, 109 :77–88.

[Fisher and Thompson, 1963] Fisher, H. and Thompson, G. (1963). Probabilistic learning combinations of local job-shop scheduling rules. *Industrial scheduling*, pages 225–251.

[Fontes et al., 2023] Fontes, D. B. M. M., Homayouni, S. M., and Gonçalves, J. F. (2023). A hybrid particle swarm optimization and simulated annealing algorithm for the job shop scheduling problem with transport resources. *European Journal of Operational Research*, 306(3) :1140–1157.

[Fowlkes and Creveling, 2000] Fowlkes, W. Y. and Creveling, C. M. (2000). *L'ingénierie robuste, Méthodes Taguchi en conception.* 2 edition.

[French, 1982] French, S. (1982). *Sequencing and scheduling : an introduction to the mathematics of the job-shop.* Chichester, West Sussex : E. Horwood.

[Garey et al., 1976] Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2) :117–129.

[Gendreau and Potvin, 2005] Gendreau, M. and Potvin, J.-Y. (2005). Tabu search. In Burke, E. K. and Kendall, G., editors, *Search Methodologies : Introductory Tutorials in Optimization and Decision Support Techniques*, pages 165–185. Springer, Boston, MA, 1 edition.

[Giard, 2003] Giard, V. (2003). *Gestion de la production et des flux.* 3e édition. Economica.

[Giffler and Thompson, 1960] Giffler, B. and Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research*, 8(4) :487–503.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Longman Publishing Co.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. In *Annals of Discrete Mathematics*, volume 5, pages 287–326. Elsevier, discrete optimization ii edition.

[Güçdemir and Selim, 2017] Güçdemir, H. and Selim, H. (2017). Customer centric production planning and control in job shops : A simulation optimization approach. *Journal of Manufacturing Systems*, 43 :100–116.

[Ham, 2020] Ham, A. (2020). Transfer-robot task scheduling in flexible job shop. *Journal of Intelligent Manufacturing*, 31(7) :1783–1793.

[Han et al., 2022] Han, Y., Chen, X., Xu, M., An, Y., Gu, F., and Ball, A. D. (2022). A multi-objective flexible job-shop cell scheduling problem with sequence-dependent family setup times and intercellular transportation by improved nsga-ii. *Proceedings of the Institution of Mechanical Engineers, Part B : Journal of Engineering Manufacture*, 236(5) :540–556.

[He et al., 2021] He, Y., Xin, B., Lu, S., and Ding, Y. (2021). Dynamic integrated flexible job shop scheduling with transportation robot. In *The 7th International Workshop on Advanced Computational Intelligence and Intelligent Informatics (IWACIII).*

[Hillier and Lieberman, 2001] Hillier, F. S. and Lieberman, G. J. (2001). *Introduction to Operations Research.* 7th edition. McGraw-Hill.

[Holsapple et al., 1993] Holsapple, C. W., Jacob, V. S., Pakath, R., and Zaveri, J. S. (1993). A genetics-based hybrid scheduler for generating static schedules in flexible manufacturing contexts. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(4) :953–972.

[Homayouni and Fontes, 2021] Homayouni, S. M. and Fontes, D. B. M. M. (2021). Production and transport scheduling in flexible job shop manufacturing systems. *Journal of Global Optimization*, 79(2) :463–502.

[Hoos and Stützle, 2005] Hoos, H. H. and Stützle, T. (2005). Search space structure and sls performance. In Hoos, H. H. and Stützle, T., editors, *Stochastic Local Search : Foundations and Applications*, pages 203–253. Morgan Kaufmann.

[Hordijk, 1996] Hordijk, W. (1996). A measure of landscapes. *Evolutionary Computation*, 4(4) :335–360.

[Huang and Yu, 2017] Huang, R.-H. and Yu, T.-H. (2017). An effective ant colony optimization algorithm for multi-objective job-shop scheduling with equal-size lot-splitting. *Applied Soft Computing*, 57 :642–656.

[Huang and Yang, 2019] Huang, X. and Yang, L. (2019). A hybrid genetic algorithm for multi-objective flexible job shop scheduling problem considering transportation time. *International Journal of Intelligent Computing and Cybernetics*, 12(2) :154–174.

[Ibrahim and Tawhid, 2023] Ibrahim, A. M. and Tawhid, M. A. (2023). An improved artificial algae algorithm integrated with differential evolution for job-shop scheduling problem. *Journal of Intelligent Manufacturing*, 34(4) :1763–1778.

[Jebari and Madiafi, 2013] Jebari, K. and Madiafi, M. (2013). Selection methods for genetic algorithms. *International Journal of Emerging Sciences*, 3(4) :333–344.

[Jorapur et al., 2014] Jorapur, V., Puranik, V. S., Deshpande, A. S., and Sharma, M. R. (2014). Comparative study of different representations in genetic algorithms for job shop scheduling problem. *Journal of Software Engineering and Applications*, 7(7) :571–580.

[Kacem, 2003] Kacem, I. (2003). Genetic algorithm for the flexible job-shop scheduling problem. In *IEEE International Conference on Systems, Man and Cybernetics. Conference Theme - System Security and Assurance (Cat. No.03CH37483)*, volume 4, pages 3464–3469. IEEE Xplore.

[Kamali et al., 2023] Kamali, S. R., Banirostam, T., Motameni, H., and Teshnehlab, M. (2023). An immune-based multi-agent system for flexible job shop scheduling problem in dynamic and multi-objective environments. *Engineering Applications of Artificial Intelligence*, 123 :106317.

[Karimi et al., 2017] Karimi, S., Ardalan, Z., Naderi, B., and Mohammadi, M. (2017). Scheduling flexible job-shops with transportation times : Mathematical models and a hybrid imperialist competitive algorithm. *Applied Mathematical Modelling*, 41 :667–682.

[Kauffman and Levin, 1987] Kauffman, S. and Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1) :11–45.

[Kauffman, 1993] Kauffman, S. A. (1993). *The Origins of Order : Self-Organization and Selection in Evolution*. Oxford University Press.

[Kinast et al., 2022] Kinast, A., Braune, R., Doerner, K. F., Rinderle-Ma, S., and Weckenborg, C. (2022). A hybrid metaheuristic solution approach for the cobot assignment and job shop scheduling problem. *Journal of Industrial Information Integration*, 28 :100350.

[Kovács et al., 2020] Kovács, L., Agárdi, A., and Bányai, T. (2020). Fitness landscape analysis and edge weighting-based optimization of vehicle routing problems. *Processes*, 8(11) :1363.

[Kramer, 2017] Kramer, O. (2017). *Genetic Algorithm Essentials*, chapter 2, Genetic Algorithms, pages 11–19. Springer International Publishing.

[Kuhpfahl and Bierwirth, 2016] Kuhpfahl, J. and Bierwirth, C. (2016). A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, 66 :44–57.

[Kumar et al., 2011] Kumar, M. V. S., Janardhana, R., and Rao, C. S. P. (2011). Simultaneous scheduling of machines and vehicles in an fms environment with alternative routing. *The International Journal of Advanced Manufacturing Technology*, 53(1) :339–351.

[Lawrence, 1984] Lawrence, S. R. (1984). Resource constrained project scheduling. *An experimental investigation of heuristic scheduling techniques (supplement)*.

[Lei et al., 2017] Lei, D., Zheng, Y., and Guo, X. (2017). A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption. *International Journal of Production Research*, 55(11) :3126–3140.

[Lei and Xiong, 2008] Lei, D.-M. and Xiong, H.-J. (2008). Job shop scheduling with stochastic processing time through genetic algorithm. In *International Conference on Machine Learning and Cybernetics*, volume 2, pages 941–946.

[Li et al., 2020] Li, J.-Q., Deng, J.-W., Li, C.-Y., Han, Y.-Y., Tian, J., Zhang, B., and Wang, C.-G. (2020). An improved jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times. *Knowledge-Based Systems*, 200 :106032.

[Li and Lei, 2021] Li, M. and Lei, D. (2021). An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling with transportation and sequence-dependent setup times. *Engineering Applications of Artificial Intelligence*, 103 :104307.

[Li and Gao, 2016] Li, X. and Gao, L. (2016). An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. *International Journal of Production Economics*, 174 :93–110.

[Liao and Wang, 2019] Liao, W. and Wang, T. (2019). A novel collaborative optimization model for job shop production–delivery considering time window and carbon emission. *Sustainability*, 11(10) :2781.

[Lin, 1965] Lin, S. (1965). Computer solutions of the traveling salesman problem. *The Bell System Technical Journal*, 44(10) :2245–2269.

[Liu et al., 2023] Liu, R., Piplani, R., and Toro, C. (2023). A deep multi-agent reinforcement learning approach to solve dynamic job shop scheduling problem. *Computers & Operations Research*, 159 :106294.

[Liu et al., 2013] Liu, Z., Ma, S., Shi, Y., and Teng, H. (2013). Solving multi-objective flexible job shop scheduling with transportation constraints using a micro artificial bee colony algorithm. In *Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pages 427–432.

[Luo et al., 2020] Luo, Q., Deng, Q., Gong, G., Zhang, L., Han, W., and Li, K. (2020). An efficient memetic algorithm for distributed flexible job shop scheduling problem with transfers. *Expert Systems with Applications*, 160 :113721.

[Malan, 2021] Malan, K. (2021). A survey of advances in landscape analysis for optimisation. *Algorithms*, 14 :40.

[Malan and Engelbrecht, 2013] Malan, K. M. and Engelbrecht, A. P. (2013). A survey of techniques for characterising fitness landscapes and some possible ways forward. *Information Sciences*, 241 :148–163.

[Marmion, 2013] Marmion, M.-E. (2013). Neutrality in flow shop scheduling problems : Landscape structure and local search. In Jarboui, B., Siarry, P., and Teghem, J., editors, *Metaheuristics for Production Scheduling*, pages 97–125. ISTE Ltd and John Wiley & Sons, Inc.

[Mastrolilli and Gambardella, 2000] Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3.

[Mattfeld et al., 1999] Mattfeld, D. C., Bierwirth, C., and Kopfer, H. (1999). A search space analysis of the job shop scheduling problem. *Annals of Operations Research*, 86 :441–453.

[Meeran and Morshed, 2012] Meeran, S. and Morshed, M. S. (2012). A hybrid genetic tabu search algorithm for solving job shop scheduling problems : a case study. *Journal of Intelligent Manufacturing*, 23(4) :1063–1078.

[Meng et al., 2023] Meng, L., Zhang, B., Gao, K., and Duan, P. (2023). An milp model for energy-conscious flexible job shop problem with transportation and sequence-dependent setup times. *Sustainability*, 15(1) :776.

[Momenikorbekandi and Abbod, 2023] Momenikorbekandi, A. and Abbod, M. F. (2023). A novel metaheuristic hybrid parthenogenetic algorithm for job shop scheduling problems : Applying an optimization model. *IEEE Access*, 11 :56027–56045.

[Motwani and Raghavan, 1995] Motwani, R. and Raghavan, P. (1995). *Randomized Algorithms*. Cambridge University Press.

[Mouret et al., 2011] Mouret, S., Grossmann, I. E., and Pestiaux, P. (2011). Time representations and mathematical models for process scheduling problems. *Computers & Chemical Engineering*, 35(6) :1038–1063.

[Nakano and Yamada, 1991] Nakano, R. and Yamada, T. (1991). Conventional genetic algorithm for job shop problems. In Belew, R. K. and Booker, L. B., editors, *4th International Conference on Genetic Algorithms, San Diego, CA, USA*, pages 474–479. Morgan Kaufmann.

[Nouiri et al., 2018] Nouiri, M., Bekrar, A., Jemai, A., Niar, S., and Ammari, A. C. (2018). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. *Journal of Intelligent Manufacturing*, 29(3) :603–615.

[Nouri et al., 2016a] Nouri, H. E., Driss, O. B., and Ghédira, K. (2016a). A classification schema for the job shop scheduling problem with transportation resources : State-of-the-art review. In Silhavy, R., Senkerik, R., Oplatkova, Z. K., Silhavy, P., and Prokopova, Z., editors, *Artificial Intelligence Perspectives in Intelligent Systems*, pages 1–11. Springer International Publishing.

[Nouri et al., 2016b] Nouri, H. E., Driss, O. B., and Ghédira, K. (2016b). Simultaneous scheduling of machines and transport robots in flexible job shop environment using hybrid metaheuristics based on clustered holonic multiagent model. *Computers & Industrial Engineering*, 102 :488–501.

[Nowicki and Smutnicki, 1996] Nowicki, E. and Smutnicki, C. (1996). A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6) :797–813.

[Olson, 2013] Olson, M. (2013). Mental model : Fitness landscapes.

[Olson, 2020] Olson, M. (2020). Lay of the landscape.

[Osogami and Imai, 2000] Osogami, T. and Imai, H. (2000). International symposium on algorithms and computation. In ISAAC, editor, *Algorithms and Computation*, volume 1969 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag Berlin Heidelberg.

[Pan et al., 2022] Pan, Z., Wang, L., Zheng, J., fang Chen, J., and Wang, X. (2022). A learning-based multi-population evolutionary optimization for flexible job shop scheduling problem with finite transportation resources. *IEEE Transactions on Evolutionary Computation*.

[Pavelski et al., 2019] Pavelski, L. M., Delgado, M. R., and Éléonore Kessaci, M. (2019). Meta-learning on flowshop using fitness landscape analysis. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 925—-933. ACM Digital Library.

[Payne and Wagner, 2019] Payne, J. L. and Wagner, A. (2019). The causes of evolvability and their evolution. *Nature Reviews Genetics*, 20(1) :24–38.

[Peng et al., 2015] Peng, B., Lü, Z., and Cheng, T. (2015). A tabu search/path relinking algorithm to solve the job shop scheduling problem. *Computers & Operations Research*, 53 :154–164.

[Peng et al., 2022] Peng, Z., Zhang, H., Tang, H., Feng, Y., and Yin, W. (2022). Research on flexible job-shop scheduling problem in green sustainable manufacturing based on learning effect. *Journal of Intelligent Manufacturing*, 33(6) :1725–1746.

[Piatetsky-Shapiro, 1991] Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In Piatetsky-Shapiro, G. and Frawley, W. J., editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI/MIT Press.

[Pinedo, 2012] Pinedo, M. L. (2012). *Scheduling : theory, algorithms and systems*. 4th ed. Springer New York.

[Pitzer and Affenzeller, 2012] Pitzer, E. and Affenzeller, M. (2012). A comprehensive survey on fitness landscape analysis. In Fodor, J., Klempous, R., and Araujo, C. P. S., editors, *Recent Advances in Intelligent Engineering Systems*, pages 161––191. Studies in Computational Intelligence, vol 378, springer, berlin, heidelberg edition.

[Ponnambalam et al., 2001] Ponnambalam, S. G., Aravindan, P., and Rao, P. S. (2001). Comparative evaluation of genetic algorithms for job-shop scheduling. *Production Planning & Control*, 12(6) :560–574.

[Ponsich and Coello, 2013] Ponsich, A. and Coello, C. A. C. (2013). A hybrid differential evolution—tabu search algorithm for the solution of job-shop scheduling problems. *Applied Soft Computing*, 13(1) :462–474.

[Radcliffe, 1994] Radcliffe, N. J. (1994). The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4) :339–384.

[Reidys and Stadler, 2001] Reidys, C. M. and Stadler, P. F. (2001). Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2) :321–350.

[Rossi, 2014] Rossi, A. (2014). Flexible job shop scheduling with sequence-dependent setup and transportation times by ant colony with reinforced pheromone relationships. *International Journal of Production Economics*, 153 :253–267.

[Rosé et al., 1996] Rosé, H., Ebeling, W., and Asselmeyer, T. (1996). The density of states — a measure of the difficulty of optimisation problems. In *Parallel Problem Solving from Nature — PPSN IV*, pages 208–217. Springer Berlin Heidelberg.

[Rothlauf, 2006] Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms*, chapter 2, Representations for Genetic and Evolutionary Algorithms, pages 9–32. Springer-Verlag Berlin Heidelberg.

[Salido et al., 2016] Salido, M. A., Escamilla, J., Giret, A., and Barber, F. (2016). A genetic algorithm for energy-efficiency in job-shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 85(5) :1303–1314.

[Sana et al., 2023] Sana, M. U., Li, Z., Javaid, F., Hanif, M. W., and Ashraf, I. (2023). Improved particle swarm optimization based on blockchain mechanism for flexible job shop problem. *Cluster Computing*, 26(5) :2519–2537.

[Schiavinotto and Stützle, 2007] Schiavinotto, T. and Stützle, T. (2007). A review of metrics on permutations for search landscape analysis. *Computers & Operations Research*, 34(10) :3143–3153.

[Shafee, 2014] Shafee, T. (2014). *Evolvability of a viral protease : experimental evolution of catalysis, robustness and specificity*. PhD thesis, University of Cambridge, https ://doi.org/10.17863/CAM.16528.

[Shen et al., 2018] Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2) :503–516.

[Song et al., 2023] Song, W., Chen, X., Li, Q., and Cao, Z. (2023). Flexible job-shop scheduling via graph neural network and deep reinforcement learning. *IEEE Transactions on Industrial Informatics*, 19(2) :1600–1610.

[Stanković et al., 2022] Stanković, A., Petrović, G., Marković, D., and Žarko Ćojbašić (2022). Solving flexible job shop scheduling problem with transportation time based on neuro -fuzzy suggested meta-heuristics. *Acta Polytechnica Hungarica*, 19(4) :209–227.

[Stefansson et al., 2011] Stefansson, H., Sigmarsdottir, S., Jensson, P., and Shah, N. (2011). Discrete and continuous time representations and mathematical models for large production scheduling problems : A case study from the pharmaceutical industry. *European Journal of Operational Research*, 215(2) :383–392.

[Storer et al., 1992] Storer, R. H., Wu, S. D., and Vaccari, R. (1992). New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, 38(10) :1495–1509.

[Strassl and Musliu, 2022] Strassl, S. and Musliu, N. (2022). Instance space analysis and algorithm selection for the job shop scheduling problem. *Computers & Operations Research*, 141 :105661.

[Streeter and Smith, 2006] Streeter, M. J. and Smith, S. F. (2006). How the landscape of random job shop scheduling instances depends on the ratio of jobs to machines. *Journal of Artificial Intelligence Research*, 26 :247–287.

[Sörensen, 2015] Sörensen, K. (2015). Metaheuristics—the metaphor exposed. *International Transactions in Operational Research*, 22(1) :3–18.

[Sörensen and Glover, 2013] Sörensen, K. and Glover, F. W. (2013). Metaheuristics. In Gass, S. I. and Fu, M. C., editors, *Encyclopedia of Operations Research and Management Science*, pages 960–970. Springer, Boston, MA, 3 edition.

[Taillard, 1993] Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2) :278–285.

[Taillard, 1994] Taillard, E. D. (1994). Parallel taboo search techniques for the job shop scheduling problem. *ORSA journal on Computing*, 6(2) :108–117.

[Talbi, 2009] Talbi, E.-G. (2009). *Metaheuristics : From Design to Implementation.* John Wiley & Sons.

[Tari, 2019] Tari, S. (2019). *Stratégies d'exploration de paysages de fitness : application à la résolution approchée de problèmes d'optimisation combinatoire.* PhD thesis, Université d'Angers, HAL, https ://tel.archives-ouvertes.fr/tel-02469501.

[Tasgetiren et al., 2004] Tasgetiren, M. F., Liang, Y.-C., and Şevkli, M. (2004). Particle swarm optimization algorithm for makespan and maximum lateness minimization in permutation flowshop sequencing problem. In *Proceedings of the Fourth International Symposium on Intelligent Manufacturing Systems*, pages 431–441.

[Tignon, 2020] Tignon, E. (2020). Paysages de fitness pour le problème d'ordonnancement open-shop. Master's thesis.

[T'Kindt and Billaut, 2002] T'Kindt, V. and Billaut, J.-C. (2002). *Multicriteria scheduling : theory, models and algorithms.* Springer.

[Tsujimura et al., 1997] Tsujimura, Y., Gen, M., Cheng, R., and Momota, T. (1997). Comparative studies on encoding methods of ga for open shop scheduling. In *Australian Journal of Intelligent Information Processing Systems*, volume 4, page 214.

[Vallada and Ruiz, 2011] Vallada, E. and Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, 211(3) :612–622.

[van Hoorn, 2018] van Hoorn, J. J. (2018). The current state of bounds on benchmark instances of the job-shop scheduling problem. *Journal of Scheduling*, 21(1) :127—-128.

[van Laarhoven et al., 1992] van Laarhoven, P. J. M., Aarts, E. H. L., and Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. *Operations Research*, 40(1) :113–125.

[Vanneschi et al., 2006] Vanneschi, L., Pirola, Y., and Collard, P. (2006). A quantitative study of neutrality in gp boolean landscapes. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 895––902. Association for Computing Machinery.

[Vassilev et al., 2003] Vassilev, V. K., Fogarty, T. C., and Miller, J. F. (2003). *Smoothness, Ruggedness and Neutrality of Fitness Landscapes : from Theory to Application*, chapter 1, pages 3–44. Springer Berlin Heidelberg.

[Verel, 2016] Verel, S. (2016). *Apport à l'analyse des paysages de fitness pour l'optimisation mono-objective et multiobjective*. PhD thesis, Université du Littoral Côte d'Opale, HAL, https ://hal.archives-ouvertes.fr/tel-01425127.

[Verel et al., 2007] Verel, S., Collard, P., Tomassini, M., and Vanneschi, L. (2007). Fitness landscape of the cellular automata majority problem : View from the olympus. *Theoretical Computer Science*, 378(1) :54–77.

[Vlašic et al., 2020] Vlašic, I., Durasević, M., and Jakobović, D. (2020). A comparative study of solution representations for the unrelated machines environment. *Computers & Operations Research*, 123 :105005.

[Wang et al., 2021] Wang, X., Wang, B., Zhang, X., Xia, X., and Pan, Q. (2021). Two-objective robust job-shop scheduling with two problem-specific neighborhood structures. *Swarm and Evolutionary Computation*, 61 :100805.

[Weinberger, 1990] Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics volume*, 63 :325–336.

[While and Hingston, 2013] While, L. and Hingston, P. (2013). Usefulness of infeasible solutions in evolutionary search : An empirical and mathematical study. In *2013 IEEE Congress on Evolutionary Computation*, number 13672124, pages 1363–1370. IEEE.

[Winklehner and Hauder, 2022] Winklehner, P. and Hauder, V. A. (2022). Flexible job-shop scheduling with release dates, deadlines and sequence dependent setup times : a real-world case. In *3rd International Conference on Industry 4.0 and Smart Manufacturing*, volume 200, pages 1654–1663.

[Wright, 1932] Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the 6th International Congress of Genetics*.

[Xu et al., 2021] Xu, W., Hu, Y., Luo, W., Wang, L., and Wu, R. (2021). A multi-objective scheduling method for distributed and flexible job shop based on hybrid genetic algorithm and tabu search considering operation outsourcing and carbon emission. *Computers & Industrial Engineering*, 157 :107318.

[Xu et al., 2022] Xu, Y., Sahnoun, M., Abdelaziz, F. B., and Baudry, D. (2022). A simulated multi-objective model for flexible job shop transportation scheduling. *Annals of Operations Research*, 311(2) :899–920.

[Xuewen et al., 2020] Xuewen, H., Islma, S. M. N., and Zhuo, Y. (2020). Chromosome encoding schemes in genetic algorithms for the flexible job shop scheduling : A state-of-art review useful for artificial intelligence applications. In *5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*, pages 1–8.

[Yamada and Nakano, 1992] Yamada, T. and Nakano, R. (1992). A genetic algorithm applicable to large-scale job-shop problems. In *Parallel Problem Solving from Nature*, volume 2, pages 281–290.

[Yan et al., 2021] Yan, J., Liu, Z., Zhang, C., Zhang, T., Zhang, Y., and Yang, C. (2021). Research on flexible job shop scheduling under finite transportation conditions for digital twin workshop. *Robotics and Computer-Integrated Manufacturing*, 72 :102198.

[Yang et al., 2019] Yang, M., Ba, L., Zheng, H., Liu, Y., Wang, X., He, J., and Li, Y. (2019). An integrated system for scheduling of processing and assembly operations with fuzzy operation time and fuzzy delivery time. *Advances in Production Engineering & Management*, 14(3) :367–378.

[Yang, 2010] Yang, X.-S. (2010). *Nature-inspired Metaheuristic Algorithms*. Luniver Press.

[Yuan et al., 2023] Yuan, E., Cheng, S., Wang, L., Song, S., and Wu, F. (2023). Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143 :110436.

[Zaidi et al., 2021] Zaidi, L., Bettayeb, B., and Sahnoun, M. (2021). Optimisation and simulation of transportation tasks in flexible job shop with muti-robot systems. In *1st International Conference On Cyber Management And Engineering (CyMaEn)*.

[Zhang et al., 2022] Zhang, F., Bai, J., Yang, D., and Wang, Q. (2022). Digital twin data-driven proactive job-shop scheduling strategy towards asymmetric manufacturing execution decision. *Scientific Reports*, 12 :1546.

[Zhang et al., 2011] Zhang, G., Gao, L., and Shi, Y. (2011). An effective genetic algorithm for the flexible job-shop scheduling problem. *Expert Systems with Applications*, 38(4) :3563–3573.

[Zhang et al., 2019a] Zhang, G., Sun, J., Liu, X., Wang, G., and Yang, Y. (2019a). Solving flexible job shop scheduling problems with transportation time based on improved genetic algorithm. *Mathematical Biosciences and Engineering*, 16(3) :1334–1347.

[Zhang et al., 2021] Zhang, H., Xu, G., Pan, R., and Ge, H. (2021). A novel heuristic method for the energy-efficient flexible job-shop scheduling problem with sequence-dependent set-up and transportation time. *Engineering Optimization*, 54 :1–22.

[Zhang et al., 2019b] Zhang, J., Ding, G., Zou, Y., Qin, S., and Fu, J. (2019b). Review of job shop scheduling research and its new perspectives under industry 4.0. *Journal of Intelligent Manufacturing*, 30(4) :1809–1830.

[Zhang, 2012] Zhang, Q. (2012). *Contribution à l'ordonnancement d'ateliers avec ressources de transport*. PhD thesis, Université de Technologie de Belfort-Montbéliard.

[Zhang et al., 2012] Zhang, Q., Manier, H., and Manier, M.-A. (2012). A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. *Computers & Operations Research*, 39(7) :1713–1723.

[Zhang, 2004] Zhang, W. (2004). Configuration landscape analysis and backbone guided local search. : Part i : Satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1) :1–26.

[Zhao et al., 2019] Zhao, F., Xue, F., Yang, G., Ma, W., Zhang, C., and Song, H. (2019). A fitness landscape analysis for the no-wait flow shop scheduling problem with factorial representation. *IEEE*, 7(18484678) :21032–21047.

[Zhu et al., 2022] Zhu, S., Maier, H. R., and Zecchin, A. C. (2022). Identification of metrics suitable for determining the features of real-world optimisation problems. *Environmental Modelling & Software*, 148 :105281.

[Zhu et al., 2023] Zhu, X., Xu, J., Ge, J., Wang, Y., and Xie, Z. (2023). Multi-task multi-agent reinforcement learning for real-time scheduling of a dual-resource flexible job shop with robots. *Processes*, 11(1) :267.

[Zorin and Kostenko, 2015] Zorin, D. A. and Kostenko, V. A. (2015). Simulated annealing algorithm for job shop scheduling on reliable real-time systems. In Pinson, E., Valente, F., and Vitoriano, B., editors, *Operations Research and Enterprise Systems*, pages 31–46. Springer International Publishing.

[Zou et al., 2022] Zou, F., Chen, D., Liu, H., Cao, S., Ji, X., and Zhang, Y. (2022). A survey of fitness landscape analysis for optimization. *Neurocomputing*, 503 :129–139.

[Şahman and Korkmaz, 2022] Şahman, M. A. and Korkmaz, S. (2022). Discrete artificial algae algorithm for solving job-shop scheduling problems. *Knowledge-Based Systems*, 256 :109711.

Annexe A

Compléments relatifs aux performances de la recherche taboue sur les instances FJSPT

Dans cette rubrique, nous enrichissons les résultats déjà exposés concernant les rangs moyens des paires codage–voisinage présentés dans le chapitre 4. En complément de l'aperçu global que représentent les rangs moyens dérivés du classement des valeurs de $C_{\max}$ obtenues par l'exécution d'une recherche taboue, nous intégrons les informations suivantes.

1. Pour chacun des 30 couples codage-voisinage, le nombre de fois que le meilleur des $C_{\max}$ d'une instance a été obtenu, sur un total de 10 exécutions de recherche taboue et pour l'ensemble des 89 instances.
2. Pour chacun des 30 couples codage-voisinage, le nombre de fois que le meilleur des $C_{\max}$ moyens, sur un total de 10 exécutions de recherche taboue, pour chaque instance, a été obtenu.
3. Les résultats regroupés par famille d'instances.

A.1 Nombre de meilleurs makespan

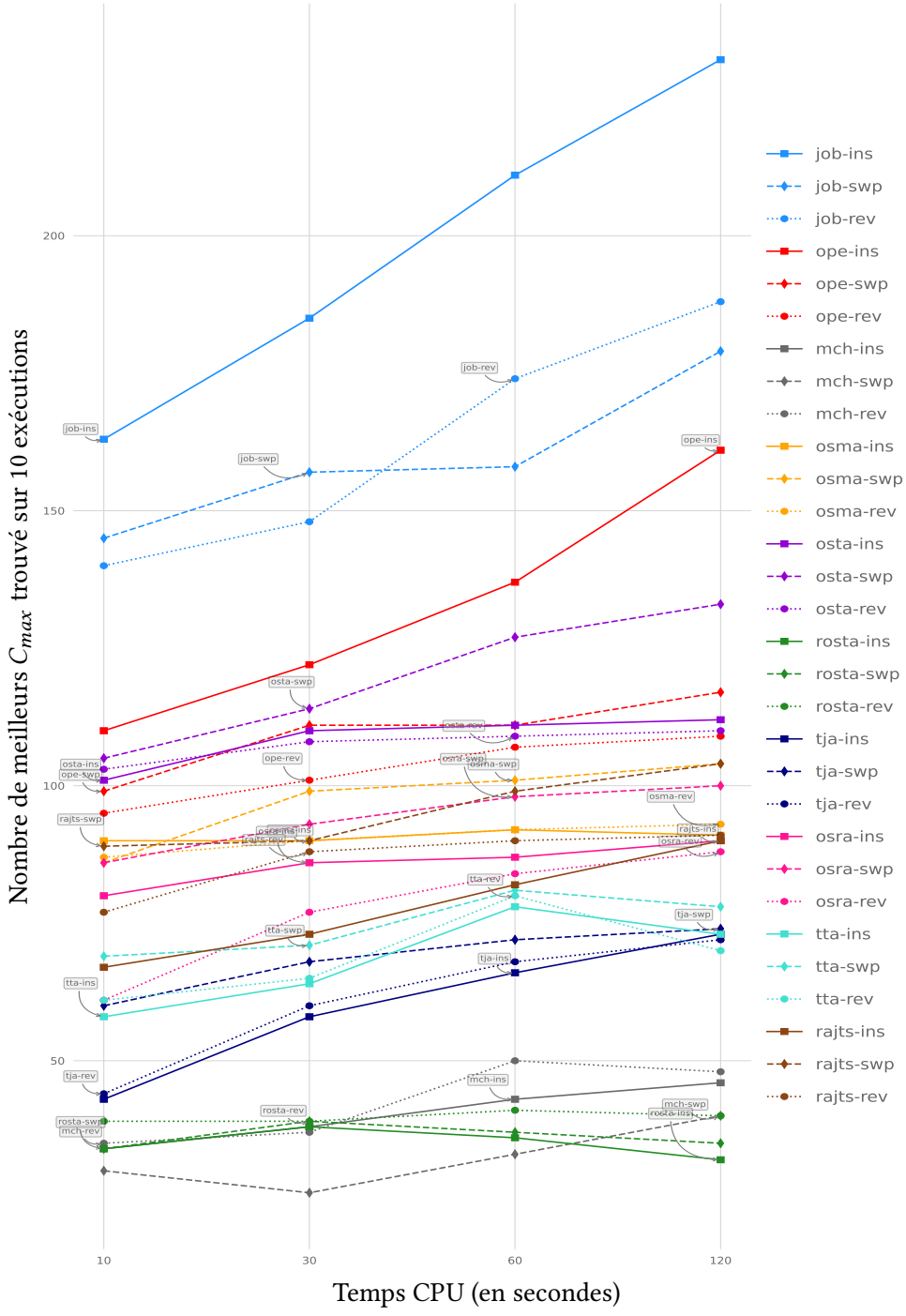


FIGURE A.1 – Nombre de meilleurs $C_{\max}$ trouvé pour chaque codage-voisinage sur 10 exécutions de recherche taboue pour les 89 instances FJSPT

A.2 Nombre de meilleurs makespan présenté par famille d'instances

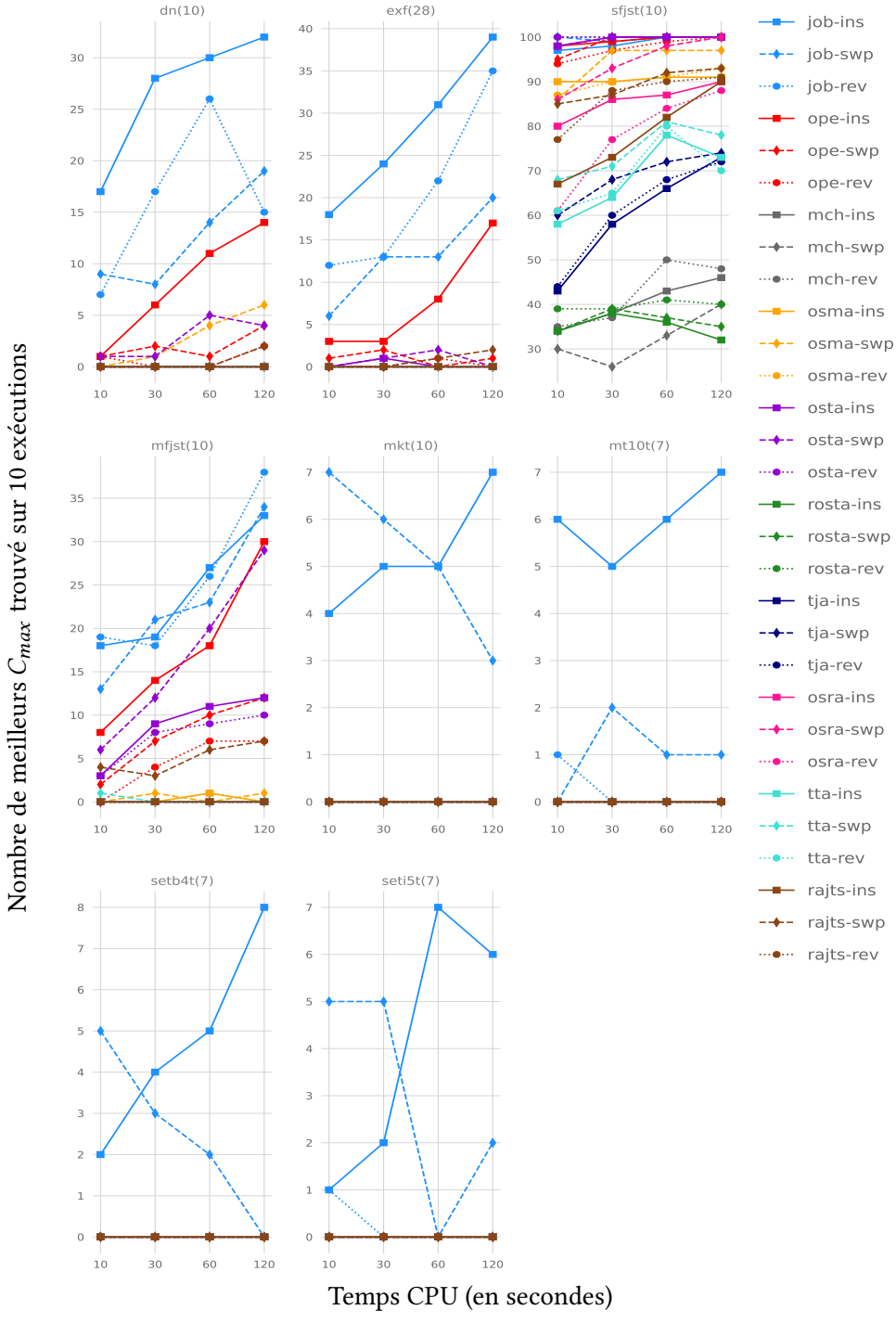


FIGURE A.2 – Nombre de meilleurs C_{max} trouvé pour chaque codage-voisinage sur 10 exécutions de recherche taboue pour les 89 instances FJSPT regroupées par famille

A.3 Nombre de meilleurs makespan moyens

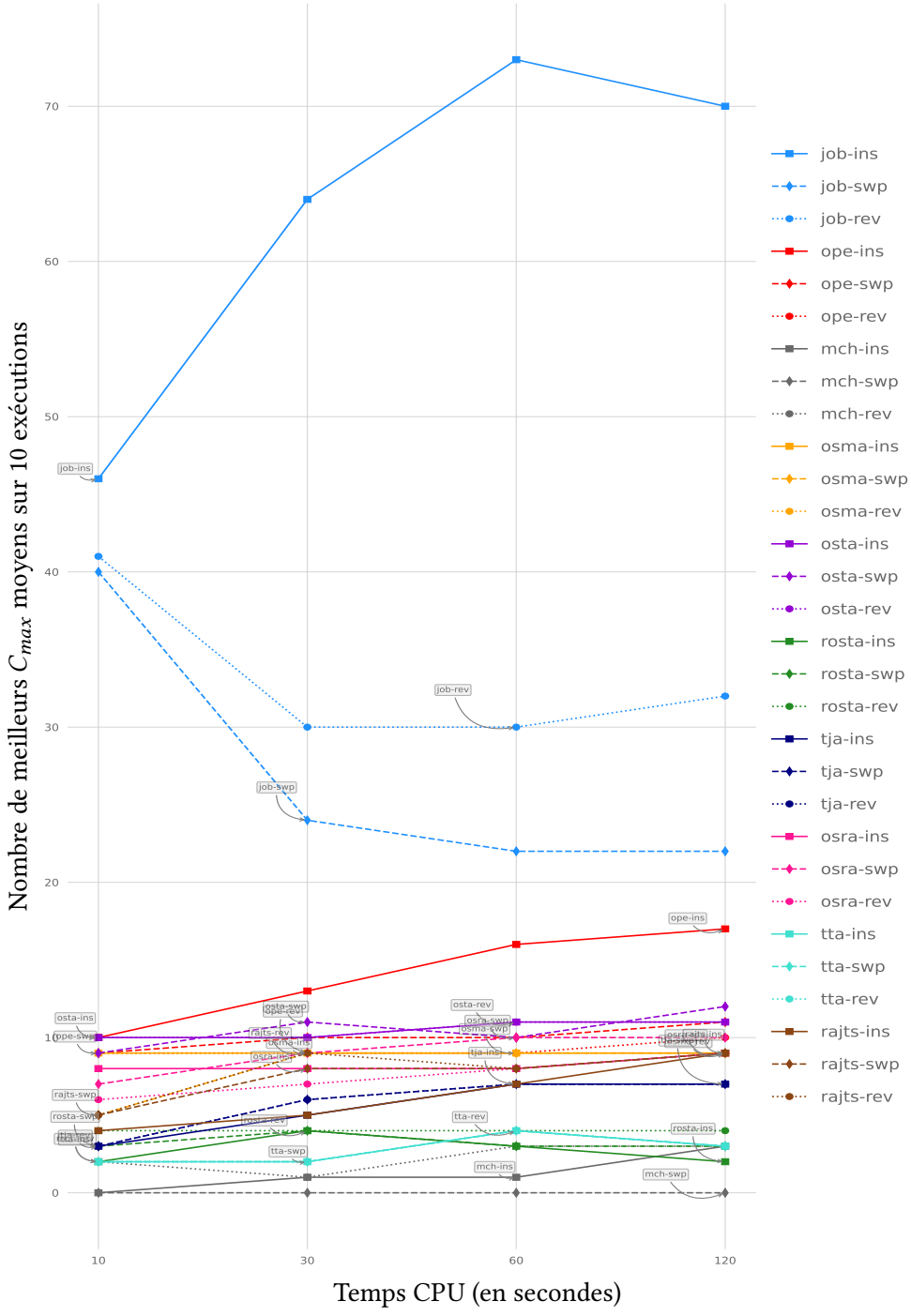


FIGURE A.3 – Nombre de meilleurs C_{max} moyens pour chaque codage-voisinage sur 10 exécutions de recherche taboue pour les 89 instances FJSPT

A.4 Nombre de meilleurs makespan moyens par famille d'instances

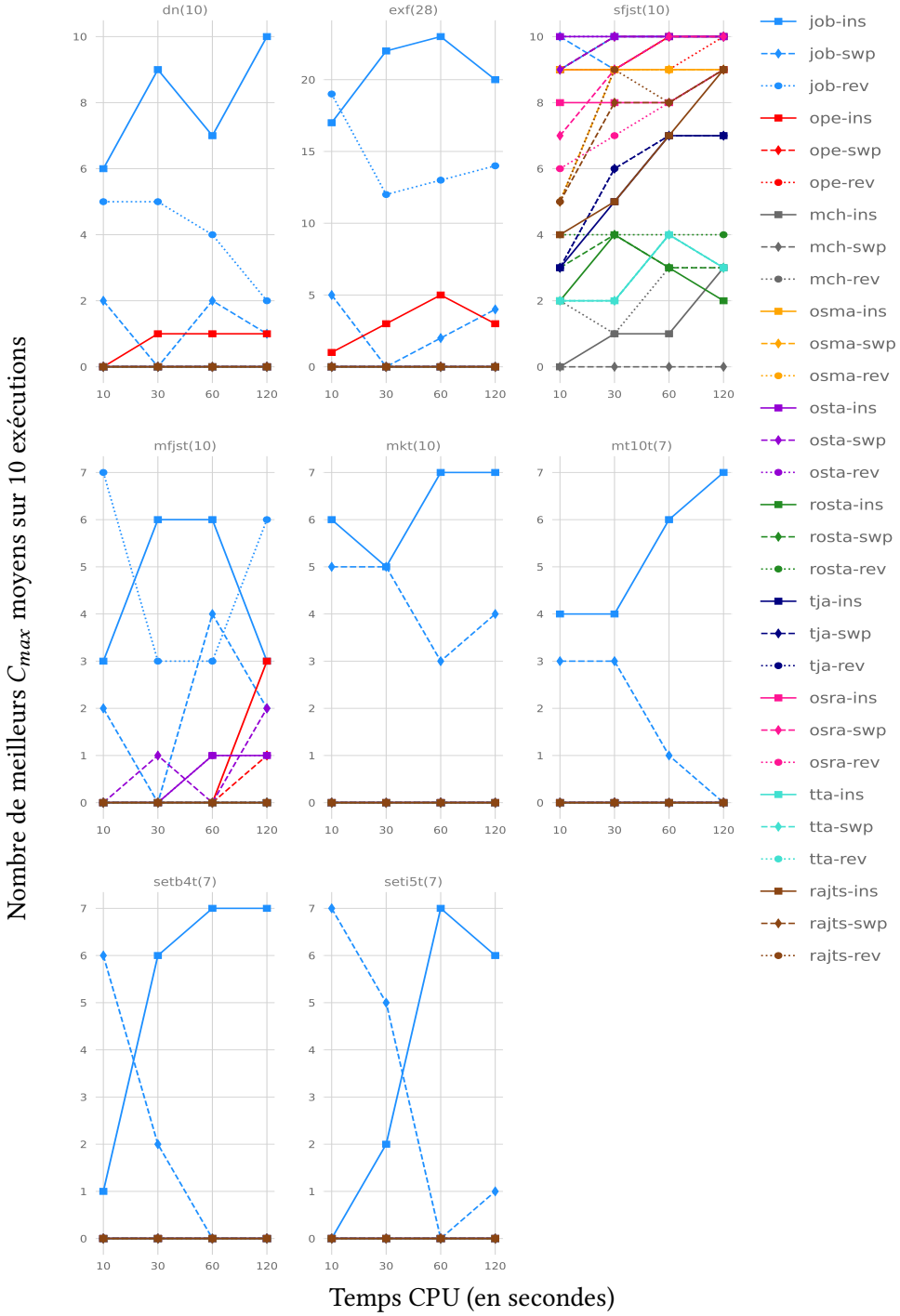


FIGURE A.4 – Nombre de meilleurs C_{max} moyens pour chaque codage-voisinage sur 10 exécutions de recherche taboue pour les 89 instances FJSPT regroupées par famille