UCLouvain

LOUVAIN
School of Management

Louvain School of Management

Développement d'un solveur d'optimisation pour scheduling

Amélioration du système de planification de production dans OMP à Avery Dennison

Auteur : Pierre-André HENNEBERT
Promoteur : Jean-Sébastien TANCREZ
Année académique 2024-2025
Mémoire-projet en vue d'obtenir le titre d'ingénieur de gestion, majeure Business Analytics
Horaire de jour

Résumé

Dans ce mémoire, un problème de scheduling est étudié. Le scheduling, ou ordonnancement, est le problème correspondant à la question : « Quel est l'ordre optimal de production ? »

Ce mémoire-projet est réalisé en collaboration avec Avery Dennison. Avery Dennison est un groupe industriel spécialisé dans la fabrication de supports autocollants. Sur le site de Soignies, le système OMP est utilisé pour tirer profit de solveurs d'optimisation et pour automatiser une partie des tâches de la planification de production.

Dans la littérature en anglais, le problème d'optimisation résolu dans ce mémoire-projet est classifié comme un *« single-machine job scheduling under early/tardy constraints with family-based setups »*.

Pour ce problème, ce mémoire propose une formulation pour résolution par Branch and Bound. Malheureusement, les essais computationnels réalisés sur des données de production ont montré qu'une approche par énumération n'est pas appropriée en pratique.

Dès lors, le cœur de ce mémoire réside dans l'implémentation d'une heuristique constructive avec des coûts locaux et son paramétrage par une optimisation. Les coûts locaux sont des coûts de réglage machine et des coûts liés aux échéances de production. Les coûts locaux sont les paramètres du solveur constructif.

Les paramètres optimaux sont obtenus par une méthodologie de fine-tuning. Le fine-tuning est l'optimisation globale des paramètres pour résoudre au mieux les 112 instances de problèmes de scheduling récoltées au sein d'Avery Dennison.

Deux algorithmes d'optimisation globale sont utilisés et comparés : Direct et Basin-Hopping. Les paramètres optimaux obtenus par la méthode Direct sont meilleurs car ils conservent le sens physique par les bornes sur leur valeur.

Pour les coûts liés aux échéances de production, deux fonctions d'activations – linéaire et exponentielle – sont comparées. Les résultats expérimentaux de la méthodologie de fine-tuning et backtesting montrent que l'activation linéaire donne de meilleurs résultats que l'activation exponentielle.

Le solveur développé avec les paramètres optimaux est validé par backtesting sur les données historiques récoltées. Les performances du solveur sont comparées aux performances du scheduling manuel et du scheduling par défaut. Cette analyse montre que le solveur est bien meilleur que le scheduling par défaut et est comparable au scheduling manuel. Cette analyse montre aussi le compromis entre le respect des contraintes et la minimisation des setups.

Une fois le fine-tuning et le backtesting réalisés, une méthodologie de développement logiciel a été employée. Ainsi, dans OMP, le solveur constructif a été développé, implémenté, testé, déployé, validé, et enfin documenté pour les utilisateurs finaux.

Remerciements

Je tiens tout d'abord à remercier Lionel Louis et Quentin Polet pour leur suivi régulier lors du stage et pour la confiance qu'ils m'ont accordée.

Je remercie également Jean-Sébastien Tancrez pour avoir accepté d'être promoteur de ce mémoire et pour ses conseils.

J'adresse mes sincères remerciements à Peter Kluge pour son aide précieuse lors de la prise en mains d'OMP sous l'angle de la programmation et pour les réponses à mes nombreuses questions.

Je tiens aussi à remercier Marie-Louise Kamphuis pour ses explications sur le fonctionnement de la centrale Supply Chain.

Enfin, un tout grand merci à Marc-André Bonge, Alain Briolo, Sabrina Cannuyer, Nathalie Canonne, Ricardo Meresse et Pejman Touni pour leurs réponses à mes questions, pour leur participation active à ce mémoire-projet et surtout pour l'excellente ambiance au sein du département Supply Chain à Soignies.

Table des matières

Table des figures

Introduction générale

Dans ce mémoire, un problème de scheduling est étudié. Il s'agit du problème d'ordre optimal de production.

L'intérêt d'étudier les problèmes de scheduling est considérable, puisqu'il s'agit de problèmes récurrents au sein de multiples productions industrielles. De plus, les problèmes de scheduling sont connus pour être computationnellement lourds.

Ce mémoire est un mémoire-projet lié à la gestion de la Supply Chain. Ce mémoire a pour objectif final d'améliorer le système OMP, servant pour la planification de production au sein d'une entreprise partenaire : Avery Dennison.

OMP est un système informatique pour la planification de production centrée autour de solveurs d'optimisation et d'une interface entre les planneurs et le plan de production. OMP est une solution pour remplacer complètement la méthode MRP (« Material Requirement Planning ») pour la planification de production.

Avery Dennison est une industrie chimique spécialisée dans les supports autocollants. Il y a quelques années, OMP a été implémenté au sein d'Avery Dennison pour automatiser le planning de production et d'achats de matières premières.

Avant mon stage réalisé à Avery Dennison, l'implémentation d'OMP ne comprenait aucun solveur d'optimisation pour le scheduling. L'importance du développement d'un solveur pour le scheduling réside dans le fait que chaque jour, plusieurs centaines d'étapes de production sont ordonnancées manuellement, en plusieurs heures.

Ce mémoire, en 4 parties, a donc pour but de détailler le développement d'un solveur d'optimisation dans OMP pour le scheduling.

Dans la première partie, le contexte et la problématique seront détaillés. A l'issue de cette première partie, les caractéristiques de l'entreprise Avery Dennison, du système OMP et du cahier des charges pour le problème d'optimisation auront été expliqués.

La deuxième partie sera constituée d'une revue scientifique et d'une revue technique. La revue de littérature scientifique aura pour objectif de classifier le problème d'optimisation à résoudre et de lister les approches classiques pour la résolution. La revue technique servira à introduire OMP sous l'angle de la programmation, afin de lister les outils à disposition pour la résolution.

La troisième partie contiendra les modèles et méthodologies que j'ai développés pour ce mémoire-projet. Une formulation MILP sera construite pour résolution par Branch and Bound. Après un essai computationnel, les heuristiques et relaxations utilisées pour accélérer la résolution du problème seront expliquées. Enfin, la méthodologie pour le fine-tuning et le backtesting du solveur sera illustrée.

Dans la quatrième partie de ce mémoire, je détaillerai mes résultats et analyses. Les résultats concerneront principalement la démarche de fine-tuning et de backtesting. Enfin, le déploiement du solveur dans OMP sera expliqué.

1. Contexte et problématique

1.1. Présentation de l'entreprise

Avery Dennison [1] est une entreprise multinationale américaine présente dans plus de 50 pays et employant plus de 35000 employés à travers le monde. Le groupe Avery Dennison est spécialisé dans la production de produits autocollants.

Parmi les applications les plus fréquentes de ces produits autocollants, citons entre autres les étiquettes d'emballages, les panneaux publicitaires, les films pour applications automobiles.

Les businesses principaux d'Avery Dennison sont : « label and packaging materials » pour les étiquettes d'emballages, et « graphic solutions » pour les autocollants destinés à la communication visuelle tels que les films pour véhicules, la signalétique et les affichages publicitaires.

Avery Dennison est présent en Belgique sur le site de Soignies, anciennement Mactac. Le site de Soignies se concentre sur le business « graphic solutions » et emploie 550 équivalents temps pleins pour un chiffre d'affaires de plus de 75 millions d'euros selon le bilan pour l'exercice de l'année 2023.

Sur le site de Soignies, la production peut être résumée en deux étapes majeures : le couchage et la finition. Les matières premières sont au nombre de quatre : la face est le papier autocollant qui est apposé sur la surface, l'adhésif à base de solvants ou à base d'eau, le papier liner est le papier qui est jeté une fois l'autocollant apposé, et le silicone est utilisé pour que le liner ne colle pas à l'adhésif. Lors de l'étape de couchage, les matières premières sont transformées pour créer des master rolls. Un master roll est un énorme rouleau de plusieurs kilomètres d'autocollant. Le master roll est un produit intermédiaire. Lors de l'étape de finition, les master rolls sont découpés pour créer des plus petits rouleaux de longueurs correspondant aux demandes des clients.

La Figure 1 ci-dessous illustre la production en deux étapes.

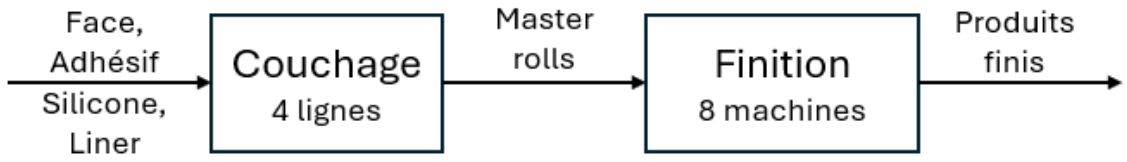

Figure 1 : Schéma simplifié de la production à Avery Dennison Soignies

En tout, le site de Soignies compte 4 lignes de couchage et 8 machines de finition. Les machines de finition sont 3 entités automatiques et 5 machines manuelles.

Les rouleaux de papier adhésif produits à Soignies peuvent être envoyés directement aux clients (« Sales Order »), au centre de distribution à Tournai (« Distribution Order »), ou bien à d'autres sites du groupe Avery Dennison (« Inter-Site Distribution Order »).

Les indicateurs clés principaux fixés par le groupe Avery Dennison pour le site de Soignies sont la disponibilité de service et la fiabilité. La disponibilité de service correspond au pourcentage de commandes clients pour lesquelles la commande a été remplie, l'objectif est de 92%. La fiabilité est la métrique pour le respect des échéances, l'objectif est de 96%.

Sur le site de Soignies, 4 stratégies de production coexistent pour les différents produits proposés.

- STO (« Ship To Order ») : les produits finis sont gardés en stocks,
- FTO (« Finish To Order ») : les master rolls sont gardés en stocks,
- CTO (« Coat To Order ») : les matières premières sont gardées en stocks,
- PTO (« Purchase To Order ») : rien n'est gardé en stocks.

L'existence de ces différentes stratégies de production implique que certaines productions sont MTO (« Make To Order ») et d'autres MTS (« Make To Stock ») sur les machines de couchage et de finition. En fait, les productions MTO et MTS cohabitent avec des frontières différentes selon les produits finis.

Pour chaque produit et selon leur stratégie de production, les stocks de sécurité sont calculés en fonction de la variance, de l'objectif de disponibilité de service et du lead time, avec une hypothèse de distribution normale de la demande. Le dimensionnement des stocks de sécurité incombe à la centrale Supply Chain située au Pays-Bas pour la gestion sur le continent européen.

Le département Supply Chain à Soignies est responsable des achats de matières premières et de la planification de production. La planification de production est réalisée dans le système OMP. Nous présenterons le système OMP dans la section suivante.

1.2. Présentation du système OMP

Dans cette section, nous allons présenter OMP sous l'angle de ses fonctionnalités. Il est important de comprendre le fonctionnement du système OMP car c'est dans ce système que sera développé le solveur pour le scheduling.

OMP est une entreprise belge développant des logiciels intégrés pour l'optimisation de la Supply Chain. L'implémentation du système OMP sur le site d'Avery Dennison à Soignies a débuté en 2020 dans l'objectif d'automatiser une grande partie de la planification de production, qui était initialement réalisée manuellement par les planneurs du département Supply Chain.

Le système OMP est ici utilisé comme logiciel pour optimiser la planification de production. Le système OMP est un « Advanced Planning System ».

Le système OMP reçoit les données de ventes, d'ordres de distributions, de demandes intersites, et de niveaux de stocks depuis le système ERP (« Enterprise Resource Planning », ici QAD). Les prévisions de vente sont réalisées dans Oracle Demand Management, exportées et chargées mensuellement dans OMP via des fichiers Excel. Les étapes de production validées dans OMP sont envoyées vers Magma, le logiciel pour la gestion de la production. Le statut des étapes en production est renvoyé par Magma vers OMP.

La Figure 2 ci-dessous illustre les interactions entre le système OMP et les autres systèmes informatiques.

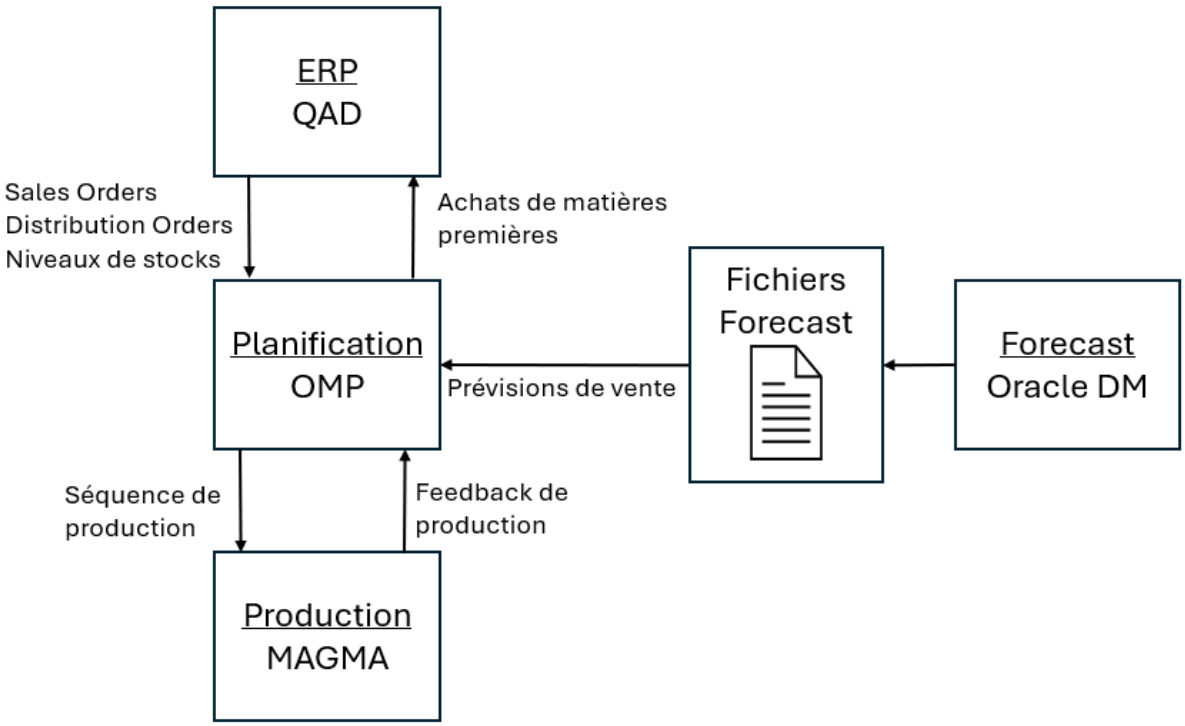

Figure 2 : Interactions entre le système OMP et les autres systèmes informatiques

Dans le système OMP, des solveurs d'optimisation proposent une planification optimale par résolution d'un problème d'optimisation sous contraintes.

Toutes les nuits et pour les produits MTS, le « Cost Solver » du système OMP calcule un planning pour les prochaines semaines, tout en minimisant les coûts en fonction des stocks, sous contraintes de capacités, de compatibilité machines, etc. Tous les jours, les planneurs reçoivent une liste de conflits non-résolus par le solveur, et fixent leurs modifications dans le planning. A ce titre, le système OMP fonctionne comme un *« rolling horizon inventory planning system »* avec intervention humaine entre deux résolutions par le solveur d'optimisation.

Pour les produits MTO, le *« Replenish Demand Solver »* du système OMP modifie le planning à chaque nouveau Sales Order.

Le système OMP a pour utilisateurs principaux les planneurs du département Supply Chain, dont le travail consiste à affiner le planning et le scheduling proposés, prendre en compte les contraintes non-modélisées par simplification, réagir aux évènements inattendus, valider et suivre le plan proposé.

Les planneurs travaillent sur 3 niveaux de planification imposés par la structure d'OMP :

- Le « Volume Planning » correspond à la planification à moyen et long termes pour les prochaines semaines. Le « Cost Solver » tourne exclusivement à ce niveau.
- L'« Order Master Planning » est le niveau intermédiaire pour fixer manuellement des décisions de planification.
- Le « Detailed Scheduling » est la planification à court terme pour les prochains jours. Le scheduling sous contraintes est réalisé manuellement, avant d'être envoyé vers Magma.

Notons que le système OMP comporte d'autres fonctionnalités que la planification de production. Citons par exemple la gestion de la demande, l'optimisation des réseaux Supply Chain, la planification du transport, etc. Ce travail n'abordera pas ces modules du système OMP.

Maintenant que le système OMP a été décrit sous l'angle de ses fonctionnalités, nous pouvons aborder le cahier des charges pour le solveur d'optimisation à développer dans OMP.

1.3. Elaboration du cahier des charges

Introduction

Avant ce mémoire-projet, les solveurs implémentés dans OMP concernaient uniquement le planning pour le couchage et l'achat des matières premières en fonction de la demande et des niveaux de stocks. En effet, une hypothèse simplificatrice considérait que la finition était pilotée par le couchage.

De plus, le scheduling était réalisé intégralement à la main par les planneurs. Pour le couchage, cela ne pose pas de problème puisque sur le court terme, les planneurs du couchage séquencent une dizaine de master rolls à produire. Pour la finition cependant, il y a plusieurs centaines d'étapes de découpes à séquencer quotidiennement pour les quelques prochains jours.

Ce mémoire-projet traite donc du développement d'un solveur pour automatiser le scheduling pour la finition. J'ai construit le cahier des charges pour cette optimisation en collaboration avec les planneurs de la finition, qui sont les utilisateurs finaux du solveur développé.

L'objectif de cette section est de présenter le cahier des charges de l'optimisation, tel que je l'ai construit avec les planneurs.

Dans cette section, nous présentons la manière dont les planneurs effectuent manuellement le scheduling. De plus, nous définissons ce qui doit être automatisé et ce qui est laissé en manuel aux planneurs.

A l'issue de ce cahier des charges, il sera possible de construire le modèle du problème d'optimisation à résoudre.

Cahier des charges

Les étapes de finition correspondent principalement à une découpe d'un master roll. Les étapes sont aussi appelées Work Orders (WOs), et proviennent directement de la planification du couchage.

Chaque WO a pour caractéristiques une longueur de bobine, une largeur de bobine, un master roll, et une boite pour l'expédition. Chaque changement d'une de ces 4 caractéristiques entre deux WOs nécessite un setup. Le setup est constitué du déchargement du master roll précédent, du chargement du nouveau master roll, de la reconfiguration de la machine, et du changement de la boite dans laquelle les rouleaux finis sont emballés.

Pour chaque WO, le master roll à découper est soit en stock soit en train d'être produit. Dans le premier cas, il n'y a pas de contrainte à considérer. Dans le second cas, la matière est disponible dès que l'étape de couchage est terminée. Cette information est disponible dans OMP.

Certains WOs présentent une contrainte qualité liée à leur durée de vie. Pour ces produits et une fois le master roll sorti de l'étape de couchage, le master roll doit être découpé avant un certain nombre d'heures (12, 24, 48 ou 72 heures).

Certains WOs sont reliés à un rainbow roll. Pour ces produits, le master roll présente plusieurs couleurs avec un WO par couleur. Le master roll doit impérativement être découpé en une fois ; les WOs d'un même rainbow roll doivent être groupés ensemble.

Chaque WO a une deadline suggérée par OMP lors de la première planification, appelée WODD (« Work Order Due Date »). En l'absence d'autre information, la WODD peut être utilisée pour trier les WOs. Cependant, il est important de noter que la WODD n'est pas la meilleure indication de l'urgence de production d'un WO. En effet, la WODD est fixée lors de la première planification sans être revue par le système OMP par la suite.

Les WOs associés à des produits MTO ont une deadline appelée SODD (« Sales Order Due Date »). Cette deadline correspond à la date de livraison sans retard au client.

Les WOs associés à des produits MTS n'ont pas de deadline explicite. Cependant, il est possible de calculer dans OMP la première date pour laquelle le produit est en rupture de stock.

Lors du scheduling manuel, les planneurs changent l'ordre et le statut des WOs.

Chaque WO a un statut dans OMP :

- Pour le statut « Busy » ou « Production », le WO est en train d'être produit. La position du WO dans la séquence ne peut plus être modifiée. Le statut n'est pas modifiable.
- Pour le statut « Launched », la position du WO dans la séquence est fixée. Le statut est modifiable.
- Pour le statut « Planned », la position du WO dans la séquence n'est pas encore fixé dans la séquence. Le statut est modifiable.

L'affectation aux 8 machines de la finition est gérée dans OMP selon le routing préférentiel. Le routing préférentiel dicte pour chaque produit fini la machine compatible permettant le plus haut niveau de productivité, en tenant compte de l'équilibrage de la charge de travail.

Très rarement, les planneurs modifient manuellement les affectations aux différentes machines pour équilibrer la charge de travail. Dans un premier temps, il n'est pas nécessaire que le solveur gère l'affectation des WOs aux machines. Ainsi, le problème de scheduling doit être résolu pour chaque machine individuellement.

De la même manière, le changement du statut des WOs ne devrait pas être laissé au choix du solveur. En effet, les planneurs restent responsables de la planification.

L'objectif est de fournir un outil aux planneurs pour faciliter le scheduling. Le solveur doit donc être rapide. Idéalement, nous souhaitons une exécution en moins d'une minute pour plusieurs centaines de WOs.

L'objectif n'est pas de remplacer le planneur par un outil automatisé mais bien de leur fournir une aide de prise à la décision.

A titre d'exemple, la Figure 3 ci-dessous illustre la complexité du scheduling dans OMP pour les 8 machines de finition. Nous observons que le scheduling concerne plusieurs centaines de WOs, pour un horizon de scheduling de plus ou moins 3 jours.

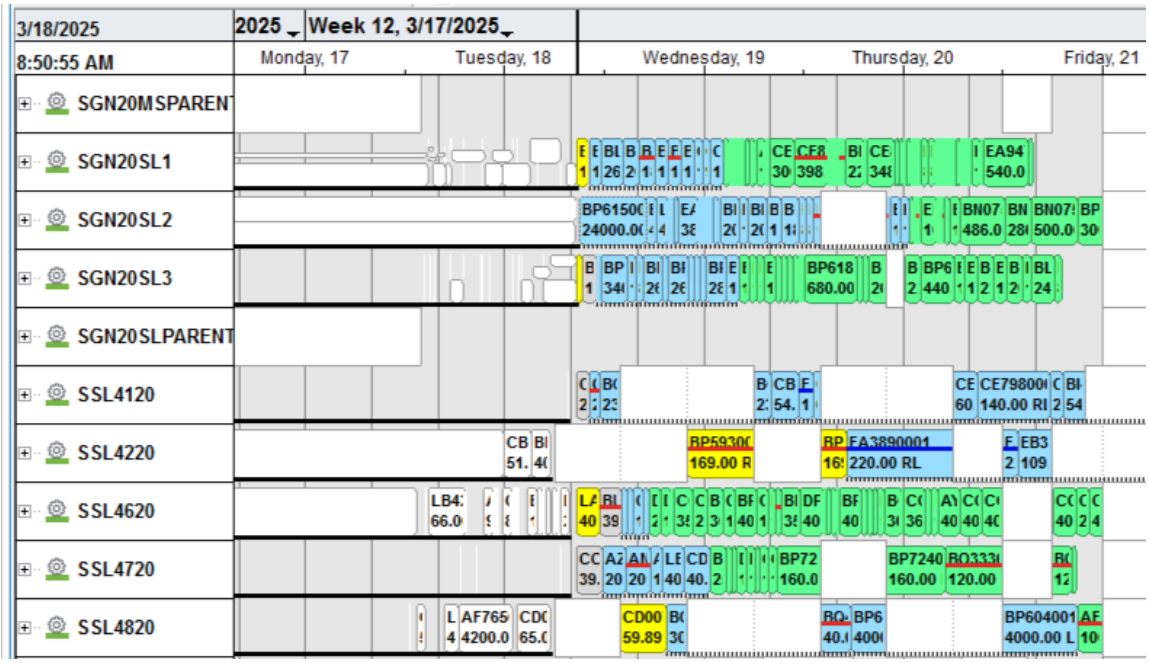

Figure 3 : Scheduling sur les 8 machines de finition dans OMP

Conclusion intermédiaire

A ce stade, nous avons situé le contexte en présentant les activités de l'entreprise et en expliquant le système OMP sous l'angle de ses fonctionnalités. Dans cette section, nous avons également construit le cahier des charges pour le problème d'optimisation à résoudre.

Nous avons donc tous les éléments essentiels pour aborder la revue scientifique et technique en nous concentrant sur nos besoins.

Dans le cahier des charges, nous avons repris la terminologie propre à l'entreprise. Pour une revue scientifique efficace, il faudra lier les concepts et contraintes de notre problème avec la terminologie scientifique.

2. Revue de littérature scientifique et technique

2.1. Revue de littérature scientifique

Introduction et classification de la problématique

L'objectif de cette revue de littérature scientifique est de parcourir les méthodes de résolution classiquement employées pour des problèmes d'optimisation similaires.

A partir du cahier des charges, nous pouvons classifier le problème pour lequel le solveur doit être développé sous : *« single-machine job scheduling under early and tardy constraints with family-based setups »*.

Pour reprendre la terminologie de la littérature scientifique, le problème étudié est le scheduling de WOs : *« job scheduling »*. Le scheduling est réalisé pour chaque machine de finition considérée séparément : *« single-machine »*. Le problème présente des contraintes de deadlines et de disponibilité matière : *« early and tardy constraints »*. Enfin, le problème comprend des setups dépendant des caractéristiques du job précédent : *« family-based setups »*, cas particulier de *« sequence-dependent setups »*.

Plusieurs machines

Le scheduling sur une seule machine est une particularisation du scheduling sur plusieurs machines. Commençons par les généralités sur le scheduling pour plusieurs machines, tout en gardant à l'esprit que notre problème est à particulariser au cas *« single-machine »*.

Si le scheduling avait été réalisé sur plusieurs machines, le problème aurait été dérivé du *« job-shop scheduling »* [2] [3] [4] [5] [6] [7] [8] [9]. Plusieurs variantes du *« job-shop scheduling »* existent. Par exemple, dans le *« flexible job-shop »*, la production peut tirer profit de compatibilités machines-produits multiples. Dans *le « customer-order job-shop »*, une commande par un client peut concerner plusieurs jobs à livrer ensemble. Dans le *« dynamic job-shop »*, les caractéristiques peuvent changer dynamiquement : nouvelles commandes urgentes, maintenance nécessaire des machines, etc.

Dans la littérature, les algorithmes étudiés pour résoudre des problèmes de scheduling sur plusieurs machines sont tous basés sur une recherche de voisinage. Des solveurs par énumération ne fonctionnent pas en pratique car le *« job-shop scheduling »* prend trop de temps pour une résolution complète. Les algorithmes de recherche de voisinage comprennent des heuristiques définies en fonction des caractéristiques du problème et une structure pour l'acceptation des schedules-solutions.

Programmation dynamique

Dans notre cas, le scheduling est réalisé plusieurs fois individuellement sur chaque machine de finition. Nous devons donc résoudre plusieurs instances de *« single-machine job scheduling »*.

Pour certaines variantes de ce problème, il existe des formulations en programmation dynamique pour obtenir la solution exacte. Ces formulations reviennent à une énumération implicite des solutions par relation de récurrence.

La formulation générale en programmation dynamique suppose un makespan constant. Le makespan désigne l'intervalle de temps entre le début du premier job et la fin du dernier job dans le schedule. Dans notre cas, les setups cassent l'hypothèse de makespan constant car le temps de setup dépend de la séquence.

Il existe des formulations en programmation dynamique pour différentes variantes du single-machine job scheduling. Par exemple, pour attribuer des deadlines aux différents jobs afin de regrouper les jobs en une même date de livraison, ou encore pour permettre de rejeter certains jobs.

Koulamas et al. (2022) [10] présentent une classification de plus de 200 formulations en programmation dynamique. Parmi les formulations présentées dans cet article, aucune ne répond à l'entièreté des exigences de notre problème. En effet, les « family-based setups » ne sont pas modélisés, alors qu'ils modifient le makespan en fonction de la séquence. Seuls trois types de setups ont une formulation décrite dans cet article : les past-sequence-dependent setups pour modéliser un effet de fatigue par un setup proportionnel à la durée totale des jobs précédemment séquencés, les setups lors d'un arrêt machine, et les setups au début de chaque batch prédéfini et indivisible.

Dans un article plus récent, Hu et al. (2025) [11] proposent une formulation en programmation dynamique pour résoudre un problème très similaire à celui qui nous préoccupe : « single machine scheduling with sequence-dependent setups ». La grande idée décrite dans ce papier est d'utiliser une heuristique spécialisée pour le scheduling à l'intérieur des familles et d'utiliser la programmation dynamique pour les changements de familles (cf. *« family-based setups »*). Cette approche est vite limitée lorsque le nombre de familles est grand, ce qui est notre cas puisqu'il y a autant de familles que de combinaisons de boites, largeurs, longueurs et de master rolls. De plus, les complexités algorithmique et spatiale présentées dans cet article ne donnent pas envie de se lancer dans une implémentation en programmation dynamique : $O(n^5 \log(n))$ pour l'heuristique spécialisée et complexité spatiale limitée à $2^n + 1$ pour la gestion des familles par programmation dynamique.

Les formulations en programmation dynamique ont l'avantage de fournir une solution exacte mais ont l'inconvénient d'être complexes à implémenter et à maintenir. En effet, ces formulations sont fort dépendantes du type de problème à résoudre, et sont difficilement adaptables si le cahier des charges est modifié dans le futur. De plus, la résolution risque fort de prendre beaucoup de temps pour un grand nombre de jobs. Pour une résolution rapide, nous étudierons maintenant les approches heuristiques.

Approches heuristiques

Une grande partie des articles traitant de single-machine job scheduling avec une composante early/tardy et/ou des family-based setups utilisent une approche par heuristique pour résoudre le problème de scheduling.

Yano et al. (1991) [12] comparent les performances de 5 heuristiques pour le problème early/tardy sans setups. Les heuristiques de cet article sont les suivantes : tri par deadline au plus tôt, tri par deadline modifiée par une règle de dominance, tri par première disponibilité matière, tri par règle de précédence localisée, et échanges. Une critique à apporter à ce papier est le manque de clarté dans l'explication du fonctionnement de ces heuristiques. Les heuristiques plus complexes de ce papier (précédence localisée et échanges) ne semblent pas facilement transposables à un contexte dans lequel le solveur doit également considérer les setups.

De Athayde Prata et al. (2020) [13] ajoutent une contrainte pour l'indisponibilité de la machine lors de périodes de maintenance. Les heuristiques proposées permettent de remplir des périodes prédéfinies, ce qui correspond à une logique de bin-packing. Cette idée est intéressante pour inciter des groupements mais nécessiterait de connaitre à l'avance l'ordre dans lequel les groupements devraient être réalisés.

Almeida et al. (1998) [14] proposent une heuristique pour le problème early/tardy sans setups. Cette heuristique réalise des permutations de jobs adjacents ou non et des insertions aléatoires dans le schedule. Chaque schedule-solution obtenu est évalué globalement et un algorithme est utilisé pour chercher l'optimum global. Comme algorithmes de recherche d'optimum global, le papier propose le recuit simulé et le tabu search.

Costa et al. (2024) [15] utilisent une heuristique constructive pour générer le schedule en sélectionnant successivement les jobs les moins chers au regard d'une fonction spécifique. Ensuite, des heuristiques avec permutations, insertions et critère de sélection de schedule sont appliquées. Même si cet article traite de la minimisation du retard total, qui n'est pas notre seul objectif, les idées proposées sont intéressantes et pourraient être transposées à notre problème.

Suriyaarachchi et al. (2004) [16] étudient les caractéristiques du problème de single-machine job early/tardy scheduling avec family-based setups, qui est précisément notre problème. Ce papier liste et prouve des propriétés des schedules optimaux. Ce papier propose une manière d'évaluer une borne sur la solution. Pour résoudre le problème, ce papier utilise une heuristique constructive (algorithme glouton) suivie d'un algorithme génétique (permutations et insertions aléatoires).

Morais et al. (2023) [17] étudient le problème de « makespan minimization pour single-machine job scheduling with release dates (disponibilité matière) and sequence-dependent setup times ». Minimiser le makespan pour un tel problème revient à minimiser la somme des temps de setups. Pour résoudre le problème, cet article utilise une approche exacte par Branch and Price ainsi qu'une approche heuristique, constituée d'une Beam Search (recherche par construction d'un arbre) suivie d'une

recherche de voisinage (permutations et insertions). Sans surprise, les résultats expérimentaux de cet article montrent le gain de temps de l'approche heuristique par rapport à l'approche exacte.

De Weerdt et al. (2020) [18] étudient le problème de « maximize accepted (non-tardy) jobs with release times, deadlines, setup times and rejection ». La seule différence majeure concerne la réjection de jobs, qui dans notre cas n'est pas acceptable. Dans cet article, la solution exacte est obtenue par programmation dynamique. Pour la formulation en programmation dynamique, l'article montre une complexité algorithmique $O(n^2w^2 2^w)$ avec w le nombre maximum de jobs avec des time-windows superposées. Dans notre cas, un grand nombre de jobs ont leur time-window superposées, ce qui signifie que w est grand. Cet article utilise également une approche par heuristique : permutations et insertions avec un algorithme génétique pour la recherche de schedule optimum.

Mazzini et al. (2001) [19] résolvent le problème single-machine job scheduling with early/tardy costs sans setups par une heuristique constructive spécialisée suivie d'une recherche de voisinage (permutation et insertions). Cette approche est intéressante. Malheureusement, l'heuristique constructive est difficilement transposable à notre cas avec setups.

Schaller et al. (2006) [20] résolvent le problème de single-machine job scheduling avec family-based setups pour minimiser l'earliness et la tardiness totales. La différence principale avec notre problème est la fonction objectif, puisque nous essayons ici aussi de minimiser les setups. Ce papier propose une manière d'évaluer une borne sur la solution. Ce papier résout le problème par une approche exacte (Branch and Bound) et par une approche heuristique : heuristique constructive spécialisée (tri par due date et algorithme de timetable) suivie d'une procédure d'amélioration (permutations et insertions).

Ronconi et al. (2010) [21] proposent des règles de pruning spécialisées par des bornes propres au problème pour accélérer une résolution par Branch and Bound du problème de minimisation des pénalités early/tardy.

Lo (2022) [22] étudie le problème early/tardy scheduling avec setups. Une solution exacte est obtenue par Branch and Bound. Une solution approchée (heuristique) est obtenue par construction simple d'un schedule (choix aléatoire parmi une liste restreinte de candidats) suivi d'une procédure d'amélioration par permutations et insertions.

Poursheikh et al. (2012) [23] présentent une approche heuristique pour minimiser la plus grande earliness/tardiness sans setups. La grande idée de ce papier est de précalculer une approximation de la position des jobs, d'utiliser cette information lors de l'application de l'algorithme hongrois (adapté aux problèmes d'affectation) pour générer le schedule, et enfin de tenter d'améliorer le schedule par permutations.

Conclusion de la revue de littérature scientifique

Cette revue de littérature scientifique avait pour objectif de parcourir les manières dont des problèmes d'optimisation similaires sont résolus.

Dans un premier temps, nous avons consulté des articles qui utilisent une formulation pour programmation dynamique, pour Branch and Bound, ou pour Branch and Price afin d'obtenir une solution exacte par énumération. Cette approche n'est valable que pour un nombre de jobs relativement petit (moins de 100 jobs). Les articles consultés utilisent les résultats de l'approche exacte comme une solution de référence.

Dans un second temps, nous avons consulté des articles qui développent extensivement des heuristiques pour approcher rapidement la meilleure solution pour un grand nombre de jobs à séquencer. Nous distinguons deux types d'heuristiques dans les papiers : les heuristiques constructives pour construire un schedule proche de l'optimum rapidement, et les heuristiques d'améliorations par permutations et insertions. Les heuristiques développées dans les articles consultés tendent à être spécifiques à chaque problème résolu.

Pour réaliser des expériences computationnelles, la majorité des articles consultés génèrent aléatoirement les données, ce qui pourrait réduire la confiance envers l'implémentation d'un algorithme complexe promettant un gain marginal de temps de calcul ou de qualité de solution.

Pour résoudre notre problème de job scheduling, une formulation sera écrite pour résolution par Branch and Bound afin de vérifier que le problème puisse être résolu pour un grand nombre de jobs. Ensuite, un solveur sera développé par approche heuristique pour une résolution rapide. C'est probablement ce solveur par heuristique qui sera implémenté dans OMP.

2.2. Revue technique

L'objectif de cette revue technique est de décrire les outils de programmation disponibles pour implémenter un solveur dans OMP pour résoudre le problème de single-machine job scheduling.

Le système OMP est une application hébergée sur un serveur. Les utilisateurs se connectent à OMP par protocole RDP (« Remote Desktop Protocol »). Les utilisateurs se connectent au serveur de production (OMP PROD, production) pour réaliser la planification de production. Un serveur de développement et de validation (OMP UAT, « User Acceptance and Testing ») permet de tester les modifications au système sans impacter la planification de production.

Les utilisateurs se connectent à un même environnement de production. En effet, OMP est un système à mémoire partagée. Grâce à cette caractéristique, les impacts des modifications apportées à la planification de production par l'un des planneurs est visible par tous les utilisateurs. Cette caractéristique signifie également qu'implémenter une validation d'un solveur sur base de données historiques n'est pas immédiatement possible dans OMP. Le fine-tuning et le backtesting d'un solveur doivent être réalisés en dehors d'OMP. Le développement du solveur est cependant directement testable dans OMP sur le serveur de développement et de validation (OMP UAT) ou sur les données réelles de production (OMP PROD).

La programmation dans OMP est réalisée dans des macros OPAL. OPAL est un langage orienté-objet spécifique à OMP. Une macro est simplement une routine avec un point d'entrée et une valeur de sortie définis. Chaque macro OPAL peut être exécutée manuellement, appelée par une autre macro, ou encore affectée à un « setting » pour être appelée lors d'une action sur l'interface utilisateur. Les macros OPAL peuvent être compilées dans des fichiers PAK.

Les macros OPAL peuvent faire référence à la mémoire partagée par leur point d'entrée. Par exemple, les données d'un WO sont stockées dans un objet Step, caractérisant une étape de production. Parmi les données d'un objet Step, citons la durée de production, la date planifiée de début de production ou encore le produit fabriqué. La section « data model » de la documentation fournie par OMP [24] permet de parcourir les descriptions des objets.

Le système OMP est centré autour d'algorithmes d'optimisation de la planification de production.

Pour les problèmes de planning, un « mathematical programming solver » est disponible pour résoudre par énumération une formulation « Mixed Integer Linear Programming » (MILP).

Pour les problèmes de scheduling, la documentation fournie par OMP décrit les deux types de « neighbourhood search solvers » disponibles. Un solveur constructif génère un schedule en sélectionnant successivement les WOs les moins chers selon une fonction coût à définir. Un solveur incrémental améliore le schedule en réalisant des permutations et insertions aléatoires et en sélectionnant les meilleurs schedules par « tabu search », par « recuit simulé » ou par « late-acceptance hill-climbing ».

Notons que dans la version d'OMP installée (R06_08), la macro OPAL interne au système OMP s'occupant du solveur incrémental est hors service. Il est possible d'implémenter l'amélioration de schedules par permutations et insertions dans une macro OPAL à partir de zéro. Cette piste nécessiterait néanmoins des développements significatifs.

Il est également possible d'implémenter une résolution par programmation dynamique dans une macro OPAL.

Pour conclure cette revue technique, le développement d'un solveur dans OMP pour résoudre le problème de single-machine job scheduling nécessite l'écriture de macros OPAL, la paramétrisation du solveur, le test dans le serveur de développement (OMP UAT), et pour terminer le déploiement en environnement de production (OMP PROD).

3. Modélisation et méthodologie

3.1. Modélisation MILP

Dans cette section, une modélisation « Mixed Integer Linear Programming » (MILP) est présentée pour résoudre notre problème de *« single-machine job scheduling under early and tardy constraints with family-based setups »*.

J'ai développé cette modélisation spécifiquement pour le problème de scheduling décrit dans la section « Elaboration du cahier des charges ».

Dans les explications du modèle, nous utiliserons le terme « job » (cf. revue scientifique) de manière interchangeable à « work order » (cf. cahier des charges).

Indices et ensembles

Ensembles	**Explications**
$i, j \in S_J = \{1..N\}$	Ensemble des jobs à ordonnancer. N est le nombre de jobs à ordonnancer. Pour ordre de grandeur, il y a régulièrement plusieurs centaines de jobs à ordonnancer.
$i \in S_{Fixed} = \{1..NumFixed\}$	Ensemble des jobs fixés. Les jobs sont fixés au début du schedule lorsque leur statut dans OMP est « busy » ou « production » et ne peuvent plus être modifiés.
$r \in S_{RB}$	Ensemble des master rolls de type rainbow rolls. Pour rappel, les rainbow rolls doivent être groupés ensemble lors de la découpe. La taille de l'ensemble S_{RB} correspond au nombre de codes rainbow rolls uniques et donc au nombre de groupements rainbow rolls.
$i, j \in S_{RB}^{r}$	Ensemble des jobs appartenant au groupement de rainbow rolls r. Les ensembles pour groupement de rainbow rolls sont précalculés.
$k \in S_{Setups} = \{1..4\}$	Ensemble des types de setups. Dans notre cas, nous avons 4 types de setups pour les changements de : boites, largeurs, longueurs et master rolls.

Paramètres

Paramètres	**Explications**
$Load_i$	Durée du job i. Cette durée est la durée de découpe pour l'étape de finition hors temps de setup. Toutes les durées sont exprimées en heures.
$PrevStepEnd_i$	Temps de la fin de l'étape de couchage du master roll découpé lors du job i. Chaque temps est exprimé par rapport au référentiel 0, début de l'horizon de scheduling. Le début de l'horizon est le temps de début du premier job.
$MaxHoursToFinish_i$	Nombre maximal d'heures avant perte de qualité pour le job i. Ce nombre d'heures est compté entre la fin de l'étape de couchage et la fin de l'étape de finition.
$WODueDate_i$	Deadline prévue pour la fin de découpe du job i. Ce paramètre correspond à l'échéance suggérée par OMP lors de la première planification. Chaque job a une échéance suggérée.
$SODueDate_i$	Deadline de livraison pour la fin de découpe du job i. Seuls les jobs liés à une livraison chez un client (Sales Order) ont une deadline de livraison.
$SetupTime_k$	Temps de setup de type k. Estimation du temps pour changement de longueur, largeur, boite et master roll.
$MSetup_{ijk}$	Binaire valant 1 s'il y a un setup de type k lorsque le job i précède directement le job j et valant 0 sinon. Ce paramètre est équivalent à « le job i et le job j ont-ils des caractéristiques k différentes ? »
N	Nombre de jobs à ordonnancer.
$NumFixed$	Nombre de jobs à fixer, ayant un statut « busy » ou « production ».
$FTMax$	Facteur de coût pour le temps de fin de schedule. Le temps de fin de schedule correspond à la somme des temps de production et des temps de setups.
$FSetup_k$	Facteur de coût d'un changement nécessitant un setup de type k.
$FDelay$	Facteur de coût d'un retard d'une heure sur l'échéance suggérée pour la fin d'un job.
$FMaxDelay$	Facteur de coût du plus grand retard sur les échéances suggérées pour la fin des jobs.
M	Un très grand nombre. Dans notre cas, $M = 100$ car nous pouvons considérer que le temps de fin de schedule ne dépassera pas 100 heures (4 jours d'horizon de scheduling).

Paramètres utilisés pour prétraitements

Paramètres	**Explications**
$Width_i$	Largeur du master roll découpé lors du job i.
$Length_i$	Longueur du master roll découpé lors du job i.
Box_i	Boite pour emballer les bobines découpées lors du job i.
$RainbowRoll_i$	Rainbow roll du job i.
MR_i	Master roll du job i.

Variables

Variables	**Explications**
t_i	Temps du début du job i.
x_{ij}	Binaire de précédence, vaut 1 si le job i précède directement le job j, vaut 0 sinon.
$setup_i$	Temps de setup du job i, calculé en fonction du job directement précèdent.
$makespan$	Temps de la fin du dernier job. Le $makespan$ correspond à la somme des temps de process et des temps de setups.
$delay_i$	Retard sur la fin suggérée pour le job i, cette variable est utilisée pour la relaxation de la contrainte de respect des échéances suggérées.
$maxDelay$	Le plus grand retard sur la fin suggérée d'un job.

Fonction objectif

L'optimisation est multi-objective. Nous minimisons :

- Le $makespan$, ce qui revient à minimiser la somme des temps de setups.
- Le nombre de setups de chaque type : boite, largeur, longueur et master roll.
- Le plus grand retard sur les deadlines suggérées.
- Le retard total sur les deadlines suggérées lors de la première planification.

$$\min z = FTMax * makespan + \sum_{k \in S_{Setups}} FSetup_k * \sum_{j \in S_J} \sum_{i \in S_J, i \neq j} x_{ij} * MSetup_{ijk}$$
$$+ FMaxDelay * maxDelay + FDelay * \sum_{i \in S_J} delay_i$$

Notons que cette formulation considère que les contraintes peuvent être respectées. En pratique, cette hypothèse n'est pas nécessairement vérifiée. Par exemple, une deadline de livraison peut être dans le passé (backlog). Pour une implémentation pratique dans OMP, ces contraintes devront être relaxées et mises dans l'objectif.

Contraintes

Calcul des temps de setups à partir de la matrice de setup si précédence et des binaires de précédences. Si l'on retranscrit directement « temps de setup du job j s'il est précédé du job i », nous obtenons le temps de setup du job j :

$$setup_j = \sum_{k \in S_{Setups}} \sum_{i \in S_J, i \neq j} x_{ij} * SetupTime_k * MSetup_{ijk} \; ; \forall j \in S_J \; ; (1)$$

La fin du dernier job est la plus grande fin de job :

$$makespan \geq t_i + setup_i + Load_i \; ; \forall i \in S_J \; ; (2)$$

La contrainte de précédence implémente une élimination des sous-tours :

$$t_i + setup_i + Load_i \leq t_j + M(1 - x_{ij}) \; ; \forall i \in S_J, j \in S_J \; tq \; j \neq i \; ; (3)$$

Puisque les jobs sont séquencés sur une même machine, les jobs se succèdent. Chaque job est suivi par au plus un autre job :

$$\sum_{j \in S_J, j \neq i} x_{ij} \leq 1 \; ; \forall i \in S_J \; ; (4)$$

Chaque job suit au plus un autre job :

$$\sum_{i \in S_J, i \neq j} x_{ij} \leq 1 \; ; \forall j \in S_J \; ; (5)$$

En dehors du dernier job, tous les jobs ont un successeur :

$$\sum_{i \in S_J} \sum_{j \in S_J, j \neq i} x_{ij} = N - 1 \; ; (6)$$

La contrainte de disponibilité matière est implémentée en vérifiant que le temps de fin de l'étape de couchage est antérieur au temps de début de l'étape de finition :

$$t_i \geq PrevStepEnd_i \; ; \forall i \in S_J \; ; (7)$$

La contrainte de qualité assure que le nombre d'heures entre la fin de l'étape de couchage et la fin de l'étape de finition ne dépasse pas le nombre maximal d'heures avant perte de qualité :

$$t_i + setup_i + Load_i - PrevStepEnd_i \leq MaxHoursToFinish_i \; ; \forall i \in S_J \; ; (8)$$

La contrainte pour le respect des échéances suggérées est relaxée au travers de la variable $delay_i$, représentant le retard sur cette échéance :

$$t_i + setup_i + Load_i \leq WODueDate_i + delay_i \; ; \forall i \in S_J \; ; (9)$$

Pour les jobs liés à une livraison chez un client (Sales Order), la contrainte de livraison assure de ne pas dépasser l'échéance de livraison :

$$t_i + setup_i + Load_i \leq SODueDate_i \; ; \forall i \in S_J \; ; (10)$$

La contrainte pour les groupements de rainbow rolls assure que chaque ensemble de jobs appartenant au même rainbow roll sont découpés l'un à la suite de l'autre. Dans la contrainte suivante, $card(S_{RB}^r)$ correspond au cardinal de l'ensemble S_{RB}^r, c'est-à-dire le nombre d'éléments contenus dans l'ensemble S_{RB}^r. La contrainte de groupement rainbow rolls (11) reprenant la même structure que la contrainte de structure (6).

$$\sum_{i \in S_{RB}^r} \sum_{j \in S_{RB}^r, j \neq i} x_{ij} = card(S_{RB}^r) - 1 \; ; \forall r \in S_{RB} \; ; (11)$$

Les premiers jobs sont fixés (statuts « busy » ou « production »). Il y a toujours au moins un job fixé. Le premier job est fixé :

$$\sum_{i \in S_J, i \neq 1} x_{i,1} = 0 \; ; (12)$$

L'enchainement des jobs fixés (statuts « busy » ou « production ») est imposé. Les jobs fixés se suivent au début du schedule :

$$x_{i,i+1} = 1 \; ; \forall i \in S_{Fixed} \; ; (13)$$

Le délai max est le plus grand retard sur les deadlines suggérées :

$$maxDelay \geq delay_i \; ; \forall i \in S_J \; ; (14)$$

Les contraintes de binarité et de positivité sont définies :

$$x_{ij} \in \{0,1\} \; ; \forall i \in S_J, j \in S_J \; tq \; j \neq i \; ; (15)$$

$$delay_i \geq 0 \; ; \forall i \in S_j \; ; (16)$$

$$t_i \geq 0 \; ; \forall i \in S_J \; ; (17)$$

$$makespan \geq 0 \; ; (18)$$

Résultats par solveur Branch and Bound

La modélisation MILP a été implémentée dans Xpress IVE pour résolution par algorithme Branch and Bound. Les données pour cet essai ont été récoltées le 13/02/2025. Les données sont des données réelles de production.

Les essais computationnels concernent 5 jobs sur la machine SSL4120 et 36 jobs sur la machine SL3. La fonction objective cherche à la fois à minimiser le nombre total de setups, le temps total de setup et le retard total sur les deadlines planifiées.

$$FTMax = 1\,; FDelay = 1\,; FMaxDelay = 0\,;\ FSetup_k = 1\ \forall k \in S_{Setups}\,;$$

Pour un très petit nombre de jobs (5 jobs), la résolution par énumération donne la solution exacte (gap=0%) en moins d'une seconde.

La Figure 4 ci-dessous montre l'évolution du gap d'optimalité en fonction du nombre de jobs pour un temps de calcul limité à 60 secondes. La résolution par Branch and Bound est optimale pour moins de 22 jobs. Les instances de scheduling résolues ont été créées à partir de l'instance ayant 36 jobs. Pour 36 jobs, nous remarquons que le gap dépasse 20%, ce qui n'est pas acceptable.

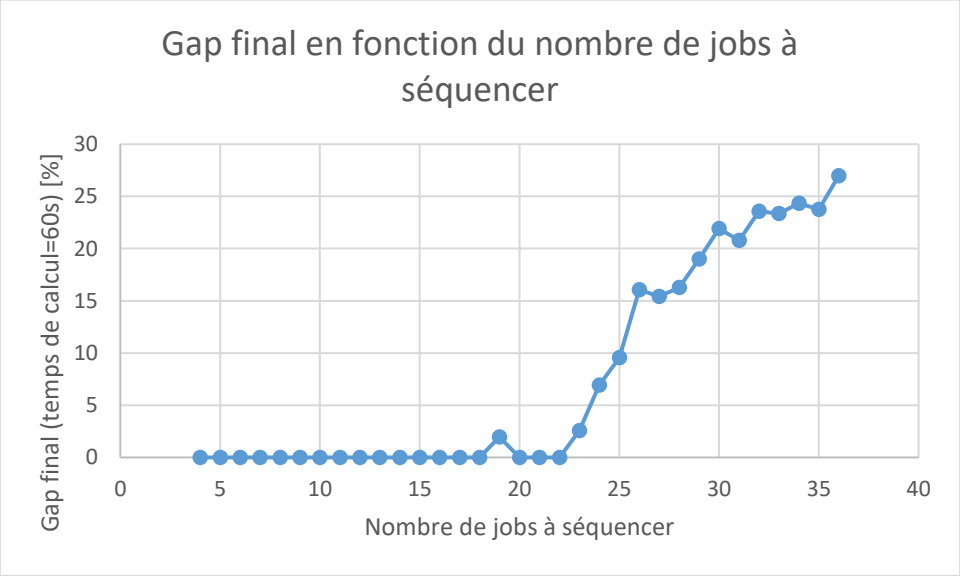

Figure 4 : Gap final en fonction du nombre de jobs – résolution par Branch and Bound

Les schedules construits à partir du modèle MILP ont été proposés aux planneurs de la finition pour validation. Le schedule avec 5 jobs a été validé mais le schedule avec 36 jobs a été jugé inacceptable par les planneurs.

Puisqu'en pratique le nombre de jobs dépasse régulièrement 36 (70, 150, et plus), il est nécessaire de développer un solveur par heuristique, pour approcher la solution optimale rapidement.

3.2. Relaxations et heuristiques

Comme nous l'avons vu dans la section précédente, lorsque le nombre de jobs est grand, le problème ne peut pas être résolu dans un temps raisonnable par énumération. Pour cette raison, il est nécessaire de développer un solveur avec heuristiques pour approximer rapidement la solution optimale.

Dans OMP, deux algorithmes solveurs sont disponibles pour résoudre un problème de scheduling par heuristiques (« neighbourhood search solvers »). Le premier algorithme disponible est le solveur incrémental et le second est le solveur constructif.

Solveur incrémental

La Figure 5 ci-dessous illustre le fonctionnement d'un solveur incrémental pour la résolution d'un problème de scheduling.

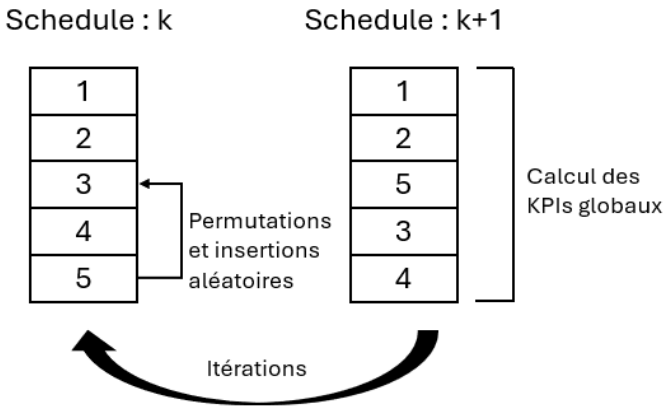

Figure 5 : Fonctionnement d'un solveur incrémental

Dans le solveur incrémental, le voisinage (« neighbourhood search solvers ») d'un schedule est défini comme l'ensemble des schedules obtenus par permutations et insertions de WOs dans ce schedule.

Le solveur incrémental améliore un schedule existant par itérations successives. A chaque itération, le solveur réalise aléatoirement des permutations et des insertions. Ensuite, les KPIs globaux sont calculés sur le schedule généré. Dans notre cas, les KPIs globaux sont le nombre de contraintes non-respectées et le nombre de setups.

Le solveur que je développerai dans OMP ne sera pas un solveur incrémental.

En effet, la macro OPAL interne à OMP pour implémenter les permutations et insertions aléatoires ainsi que les algorithmes de choix de solution est hors service dans la version R06_08. Implémenter un solveur incrémental nécessiterait donc d'écrire l'algorithme complet dans des macros OPAL, qui ne seraient alors pas maintenables par les planneurs après la fin de mon stage.

Solveur constructif

Puisque nous ne pouvons pas développer un solveur incrémental, nous allons plutôt développer un solveur constructif. Dans cette section, nous allons expliquer en détail le solveur constructif tel que je l'ai développé dans OMP.

Dans le solveur constructif, le voisinage (« neighbourhood search solvers ») d'un schedule est défini comme l'ensemble des WOs candidats à ajouter à la séquence construite.

Le solveur constructif construit un schedule à partir d'une séquence contenant initialement les WOs en statut « Busy » ou « Production ». A chaque itération, le solveur liste les WOs candidats pouvant être ajoutés à la séquence. Pour chaque candidat, le solveur calcule une fonction coût à définir. Le solveur sélectionne le candidat ayant le plus petit coût et le place à la suite de la séquence construite.

La Figure 6 ci-dessous illustre le fonctionnement d'un solveur constructif pour la résolution d'un problème de scheduling. Lors de la première itération, le WO numéro 1 fait partie du schedule et des WOs candidats sont évalués. Le WO candidat ayant le plus petit coût est sélectionné (WO numéro 4, coût valant 0.2). Lors de la deuxième itération, des WOs candidats sont à nouveaux évalués.

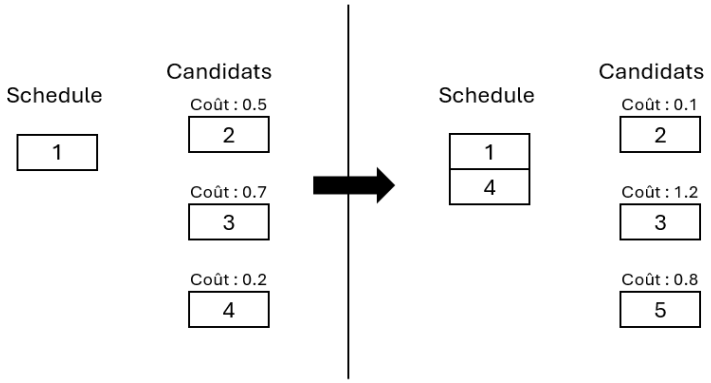

Figure 6 : Fonctionnement d'un solveur constructif

L'algorithme est glouton, et réalise autant d'itérations que de WOs à placer. A l'itération i et pour n jobs, le solveur calcule la fonction coût pour au plus $n - i$ candidats. Le nombre d'évaluations de la fonction coût est donc bornée par $\sum_{i=0}^{n}(n-i) = \sum_{i=0}^{n} i = \frac{1}{2}n(n+1)$. En effet, il y a au plus n candidats à la première itération, $n-1$ candidats à la deuxième itération, $n-2$ candidats à la troisième itération, etc. La complexité algorithmique du solveur constructif est donc $O(n^2)$.

La fonction coût à définir est au cœur du fonctionnement du solveur constructif. Les coûts locaux sont fonction des caractéristiques des candidats et de la séquence déjà construite. Par les coûts locaux, nous modélisons les objectifs et les contraintes relaxées de l'optimisation. Par les coûts locaux, nous incitons le solveur à respecter les échéances de production (deadlines) et à minimiser les setups.

Dans la construction de la liste des WOs candidats à chaque itération du solveur constructif, nous pouvons implémenter certaines contraintes.

Par exemple, la contrainte de disponibilité matière peut être implémentée en considérant dans la liste des candidats uniquement les WOs ayant un temps de fin de couchage antérieur au temps de fin du dernier WO dans la séquence construite. Ceci est vrai puisque le temps de fin du dernier WO dans la séquence construite est le temps de début du candidat sélectionné.

De la même manière, les groupements de rainbow rolls peuvent être garantis par une contrainte dans la sélection des candidats : si le dernier WO dans la séquence a un code rainbow roll et s'il reste au moins un WO à séquencer avec ce même code rainbow roll alors les seuls candidats à évaluer sont ceux avec ce code rainbow roll.

Cependant, les contraintes de deadlines (contraintes qualité et de livraison client) ne peuvent pas être modélisées par une règle dans la liste des candidats. En effet, les WOs avec une contrainte de deadline sont dans la liste des candidats jusqu'à ce que leur deadline soit dépassée, et n'apparaitront plus dans les prochaines listes de candidats. Une échéance est l'inverse logique d'une disponibilité matière.

La fonction coût est la somme pondérée des coûts locaux reprenant chacun des objectifs et des contraintes propres au problème. Nous distinguons 2 types de coûts locaux utiles pour notre problème.

- Des coûts relatifs à chaque setup entre le dernier WO de la séquence et le candidat incitent les groupements. De cette manière, la contrainte de groupement des rainbow rolls est relaxée.
- Des coûts relatifs à chaque contrainte deadline (qualité, livraison au client, deadline suggérée lors de la planification) incitent le solveur à sélectionner un candidat urgent malgré un setup potentiel. Les coûts pour les contraintes de deadline sont d'autant plus négatifs que la deadline est proche, afin de compenser le coût positif causé par un setup.

La Figure 7 ci-après illustre la manière dont les coûts sont calculés pour les candidats en fonction de leur boite et de leur deadline. Dans cet exemple, le facteur de coût local pour changement de boite vaut 1,2. Dans cet exemple, une deadline proche entraine un coût plus négatif qu'une deadline lointaine.

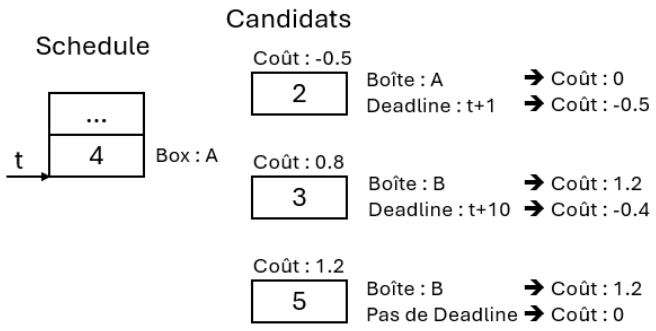

Figure 7 : Calcul des coûts locaux dans un solveur constructif

Il existe plusieurs manières de calculer les coûts locaux liés au respect des deadlines. Les coûts de deadlines sont fonction de :

- $slack$: nombre d'heures restantes avant la deadline,
- $activation$: paramètre désignant le nombre d'heures à partir duquel la deadline doit être considérée
- $facteur$: paramètre pour pondérer le respect de cette deadline par rapport aux autres objectifs de l'optimisation.

La Figure 8 ci-après illustre les deux types de fonction coût pour respecter les deadlines. L'axe des abscisses représente le nombre d'heures restantes par rapport à la deadline ($slack$). Pour cet exemple, le paramètre $activation$ est fixé à 72 heures et le paramètre $facteur$ est unitaire.

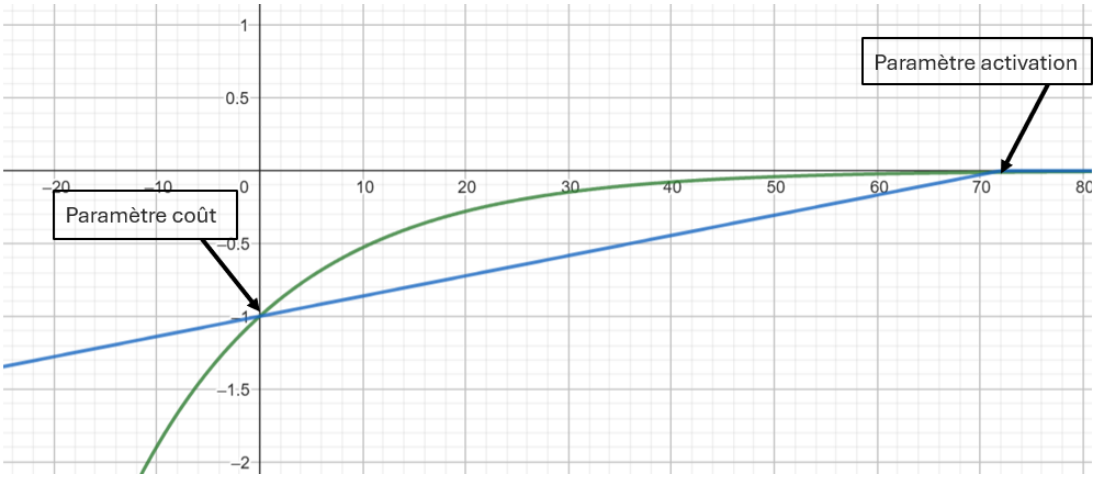

Figure 8 : Fonctions d'activation pour respect des deadlines

Pour une fonction d'activation exponentielle, le coût est donné par la formule suivante :

$$f(slack) = -facteur\,.\exp\left(-\tau * \frac{slack}{activation}\right)$$

- $slack \gg 0$: la contribution est proche de 0 lorsque la deadline est lointaine,
- $slack > 0$: la contribution est négative lorsque la deadline est proche,
- $slack < 0$: la contribution est très négative lorsque la deadline est dépassée.

Le paramètre τ est fixé arbitrairement par la règle suivante : pour $activation$ heures restantes, la fonction vaut 1% de la valeur à la deadline (0 heure restante) ; $\tau = -\ln(0.01) = 4.6$.

Pour une fonction d'activation linéaire, le coût est donné par la formule suivante :

$$f(slack) = \begin{cases} facteur\,.\dfrac{slack - activation}{activation} ; si\ slack \leq activation \\ 0\ ; sinon \end{cases}$$

Actuellement, nous ne pouvons pas savoir quelle fonction d'activation sera la plus appropriée pour résoudre notre problème de scheduling. Il faudra tester et comparer les résultats obtenus avec chaque type d'activation : linéaire et exponentielle.

3.3. Méthodologie : fine-tuning et backtesting

A ce stade, les questions suivantes se posent :

- « Quels paramètres coûts locaux génèrent des schedules ayant de meilleures performances globales ? »
- « Quel type de fonction d'activation pour les deadlines (linéaire ou exponentielle) donne de meilleurs résultats ? »
- « Quelles auraient été les performances historiques du solveur, par rapport au scheduling manuel et à un tri par défaut ? »

Pour répondre à ces 3 questions, je dois développer et appliquer une méthodologie de fine-tuning pour trouver les meilleurs paramètres et une méthodologie de backtesting pour valider l'outil sur les données historiques.

Au total, le solveur constructif personnalisé que j'ai développé dans OMP contient 13 paramètres de recherche de voisinage :

1) Facteur coût : setup boite
2) Facteur coût : setup largeur
3) Facteur coût : setup longueur
4) Facteur coût : setup master roll
5) Facteur coût : groupement rainbow roll
6) Facteur coût : deadline contrainte qualité (tardiness)
7) Facteur coût : deadline livraison client (tardiness)
8) Facteur coût : deadline suggérée (tardiness)
9) Facteur coût : deadline suggérée (earliness)
10) Paramètre d'activation : deadline contrainte qualité (tardiness)
11) Paramètre d'activation : deadline livraison client (tardiness)
12) Paramètre d'activation : deadline suggérée (tardiness)
13) Paramètre d'activation : deadline suggérée (earliness)

La Figure 9 ci-dessous illustre la méthodologie de fine-tuning et de backtesting que j'ai développée pour résoudre mon problème de scheduling.

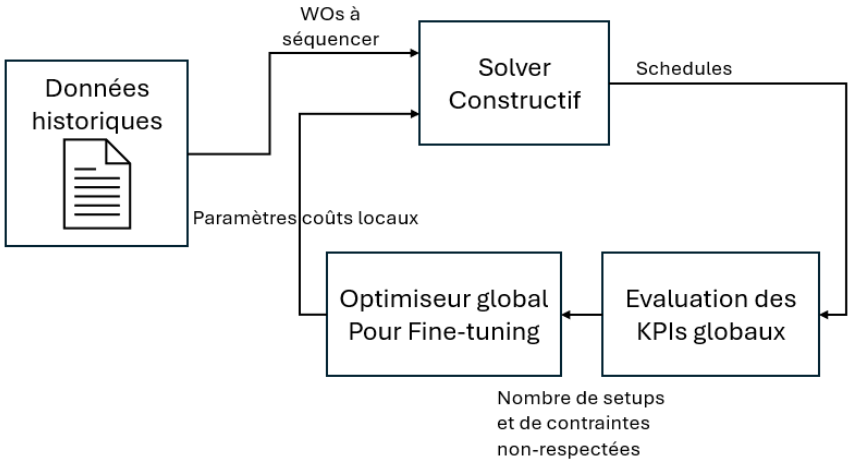

Figure 9 : Méthodologie de fine-tuning et de backtesting

Fondamentalement, le fine-tuning est un problème d'optimisation. Nous cherchons à trouver l'ensemble de 13 paramètres coûts locaux qui minimisent les KPIs globaux : nombre de setups et nombre de contraintes non-respectées.

Notons que nous aurions pu également minimiser le temps total de setups.

Pour ce problème d'optimisation, la fonction à minimiser est une fonction vectorielle non-linéaire de 13 variables et comporte de nombreux minima locaux. La fonction est :

$$fonction(paramètres) = (nombre\ de\ setups, nombre\ de\ contraintes\ pas\ respectées)$$

Rendre la fonction scalaire est nécessaire pour la rendre compatible avec des algorithmes d'optimisation globale. La fonction vectorielle a été rendue scalaire en considérant qu'une contrainte non-respectée a le même impact que 100 setups. Ce choix est tout aussi arbitraire que le choix de facteurs $(FTMax\ ; FDelay\ ; FMaxDelay\ ;\ FSetup_k)$ dans la modélisation MILP.

La fonction comprend le calcul de schedules sur les données historiques par un solveur constructif paramétré ainsi que l'évaluation de leurs KPIs globaux.

La fonction comporte de nombreux minima locaux avec effets plateaux (discontinuités) car modifier un paramètre coût d'une valeur suffisamment petite n'aura aucun impact sur les schedules générés. Ainsi, tout algorithme d'optimisation nécessitant un calcul de dérivée partielle ne fonctionnera pas.

Pour résoudre le problème de fine-tuning, deux algorithmes d'optimisation seront utilisés : Direct et Basin-Hopping. La pertinence du choix de ces algorithmes sera vérifiée dans la section analyses et résultats.

Direct [25] [26] (Dividing Rectangles) est un algorithme d'optimisation déterministe capable de minimiser une fonction boite noire. La capacité à minimiser une fonction boite noire est importante car nous n'avons aucune information sur les dérivées partielles. L'algorithme Direct utilise des bornes sur chaque variable. L'algorithme divise l'espace de recherche en rectangles et évalue la fonction au centre de ces rectangles. L'algorithme utilisé est rendu disponible dans la librairie SciPy [27].

La Figure 10 créée par Deng et al. (2007) [28] ci-dessous montre le fonctionnement de l'algorithme Direct en deux dimensions.

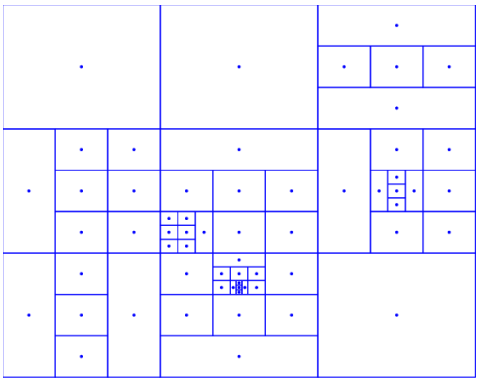

Figure 10 : Fonctionnement de l'algorithme Direct (Deng et al.)

Basin-Hopping est un algorithme d'optimisation probabiliste. A partir d'une estimation initiale, Basin-Hopping cherche l'optimum global en simulant la manière dont des groupes d'atomes réagissent à une perturbation par leur niveau d'énergie. Cet algorithme est intéressant car la fonction à minimiser comporte des plateaux de minima locaux et des discontinuités. Basin-Hopping comprend deux niveaux d'optimisation : une exploration d'une région autour d'un minimum local et une procédure pour sélectionner la prochaine région à explorer. L'algorithme utilisé est rendu disponible dans la librairie SciPy [29].

La Figure 11créée par Ferreiro-Ferreiro et al. (2019) [30] ci-dessous illustre le fonctionnement de Basin-Hopping pour une fonction à une dimension.

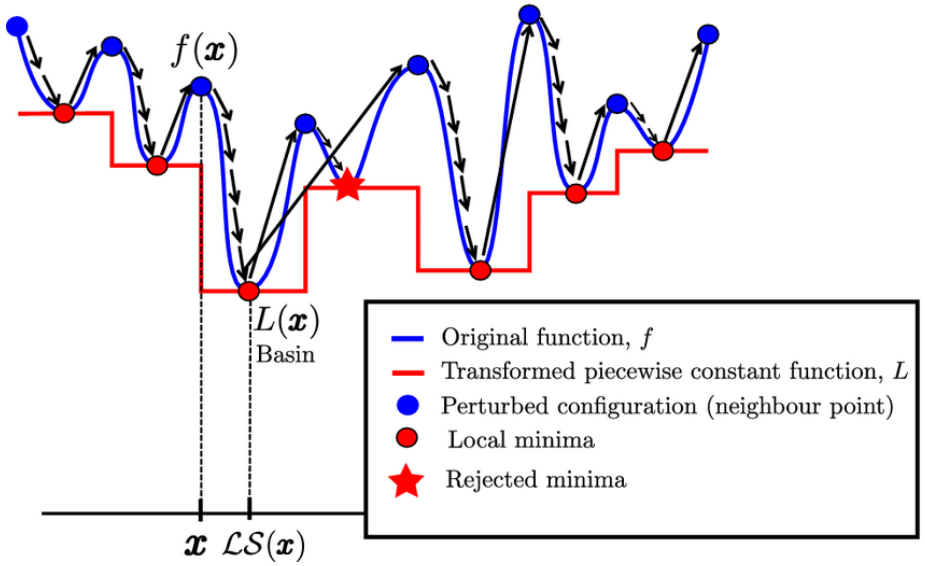

Figure 11 : Fonctionnement de l'algorithme Basin-Hopping (Ferreiro-Ferreiro et al.)

Le fine-tuning et le backtesting sont réalisés en dehors d'OMP, en Python (Google Colab notebook), pour traiter les données historiques.

Le fine-tuning est le calcul des paramètres locaux donnant lieu à un optimum global.

Le backtesting est la validation du meilleur solveur développé sur les données historiques.

Les données historiques ont été récoltées sur une période de 3 semaines à partir du serveur OMP PROD avant que mon solveur n'y soit déployé. De cette manière, il est possible de comparer les schedules générés par le solveur et ceux générés manuellement par les planneurs.

Dans le notebook en Python, j'ai implémenté l'algorithme du solveur constructif. J'ai également implémenté le fine-tuning en faisant appel aux algorithmes d'optimisation globale (Direct et Basin-Hopping). Enfin, j'ai implémenté le backtesting, tel qu'expliqué dans cette section.

4. Analyses et résultats

4.1. Données pour le backtesting

Dans cette section, nous présentons les données historiques collectées et utilisées pour la méthodologie de fine-tuning et de backtesting.

Les données pour le backtesting ont été récoltées chaque jour sur une période allant du 13/02/2025 jusqu'au 04/03/2025 avant le déploiement du solveur sur le serveur OMP PROD. Les données sont issues du serveur OMP PROD. Les données sont les données réelles de production.

Les données ont été récoltées durant 14 jours et sur 8 machines de finition. Au total, cela correspond aux données pour 112 instances de « single-machine scheduling », avec 2490 WOs en tout.

La Figure 12 ci-dessous montre le nombre de WOs à séquencer chaque jour et sur chaque machine. Durant la période de récolte de données, les machines de finition avaient un relatif petit nombre de WOs à séquencer.

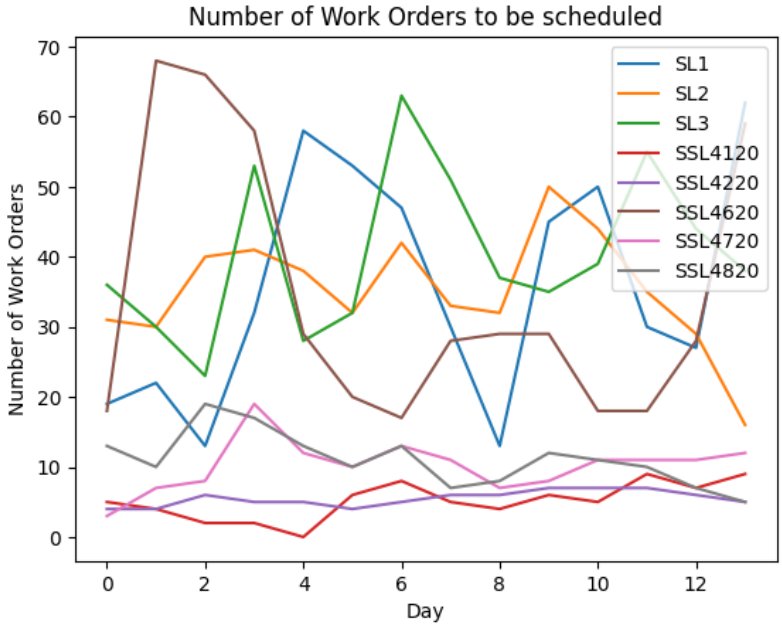

Figure 12 : Données de backtesting - nombre de WOs à séquencer

4.2. Résultats du fine-tuning

Dans cette section, nous présentons les résultats de l'optimisation globale des paramètres du solveur constructif lors du fine-tuning.

Comme Direct est un algorithme déterministe, il n'y a pas de graphique détaillant la résolution à montrer.

Basin-Hopping est un algorithme probabiliste. A chaque itération, une région caractérisée par un minimum local est visitée. Pour 100 itérations, nous obtenons donc 100 minima locaux successifs.

Les graphes ci-après montrent les minima locaux successifs trouvés par l'algorithme Basin-Hopping. Le fine-tuning utilise 100 itérations à partir d'une estimation initiale. Nous comparons les deux types d'activation de deadlines : linéaire et exponentielle.

Pour la fonction d'activation linéaire, la Figure 13 ci-dessous montre que Basin-Hopping parvient à trouver des minima locaux inférieurs à l'estimation initiale.

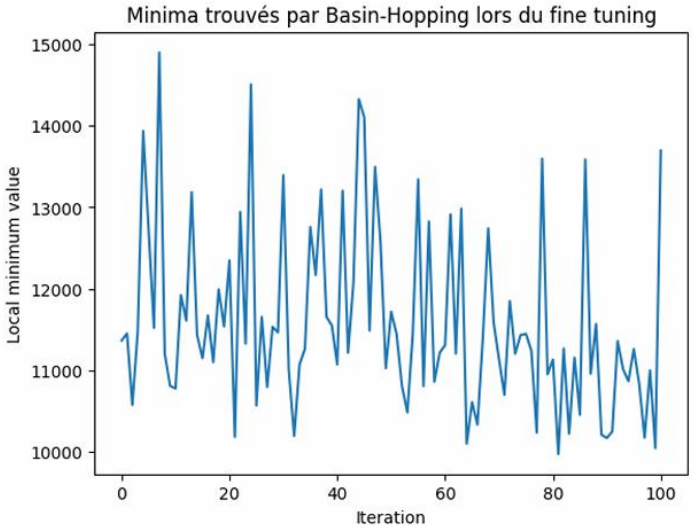

Figure 13 : Basin-Hopping pour fine-tuning – fonction d'activation linéaire

Pour la fonction d'activation exponentielle, la Figure 14 ci-dessous montre que Basin-Hopping trouve un minimum local inférieur à l'estimation initiale avant de rester bloqué entre des plateaux bien supérieurs à l'estimation initiale.

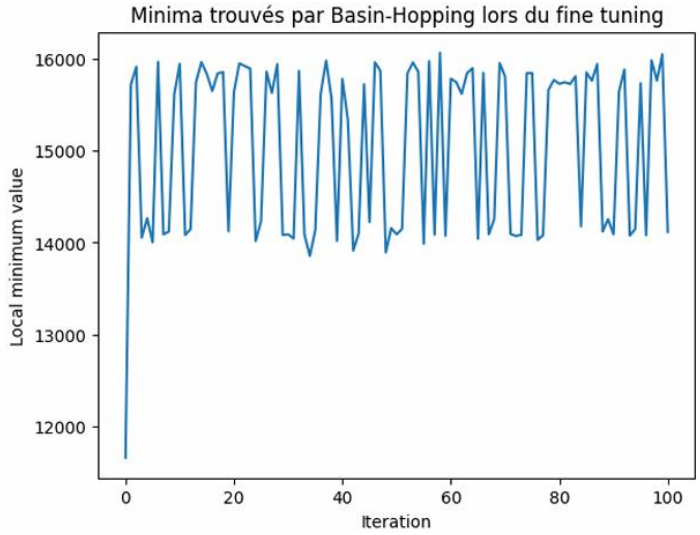

Figure 14 : Basin-Hopping pour fine-tuning – fonction d'activation exponentielle

En comparant la facilité de Basin-Hopping à trouver de meilleurs optima locaux, nous concluons que la fonction d'activation linéaire est meilleure que la fonction d'activation exponentielle. En effet, le minimum global trouvé pour activation linéaire est inférieur à celui pour activation exponentielle.

La Figure 15 ci-dessous résume les résultats du fine-tuning pour les deux fonctions d'activation (linéaire et exponentielle) et pour les trois méthodes de fine-tuning (estimation initiale, Direct et Basin-Hopping). Nous comparons les méthodes par le nombre de contraintes non-respectées et le nombre de setups totaux sur les 3 machines de finition automatiques.

#Contraintes non-respectées - #Setups	Linéaire	Exponentielle
Estimation initiale	99 - 1665	103 - 1672
Direct optimisation	83 - 1787	101 - 1896
Basin-Hopping optimisation	81 - 1870	98 - 1859

Figure 15 : Résultats obtenus lors du fine-tuning

Nous remarquons que l'activation linéaire est meilleure que l'activation exponentielle pour chaque méthode de fine-tuning. Nous remarquons également qu'il y a un compromis entre le nombre de contraintes non-respectées et le nombre de setups.

Les paramètres sélectionnés pour le solveur sont ceux obtenus par méthode avec activation linéaire et par algorithme Direct. En effet, dans le meilleur optimum trouvé par Basin-Hopping, certains facteurs coûts sont négatifs (ex. setup largeur). Ce facteur coût négatif n'a pas de sens physique car il encourage les setups de largeur.

4.3. Résultats du backtesting

Introduction

Dans cette section, nous comparons les performances du solveur sur les données historiques par rapport aux performances du scheduling manuel, et par rapport au scheduling par défaut. Le scheduling par défaut est obtenu par un tri par WODD croissantes et représente la situation dans laquelle il n'y a ni solveur ni planneur.

D'abord, les performances sont évaluées dans le temps, avec les schedules construits pour les 8 machines de finition. Ensuite, les performances sont agrégées, pour montrer les résultats globaux du solveur constructif développé et fine-tuné.

Nombre de contraintes non-respectées

Nous comparons le nombre de contraintes non-respectées. Les contraintes non-respectées sont celles de disponibilité matière, livraison client, qualité, et groupement rainbow roll.

La Figure 16 ci-dessous montre le nombre total de contraintes non-respectées sur les 8 machines pour chaque jour. Le solveur constructif est en bleu, le scheduling manuel est en orange et le tri par défaut (WODD) est en vert. Nous remarquons que le solveur est meilleur que le scheduling manuel. Sans surprise, le tri par défaut est beaucoup moins bon que le scheduling manuel ou que le solveur.

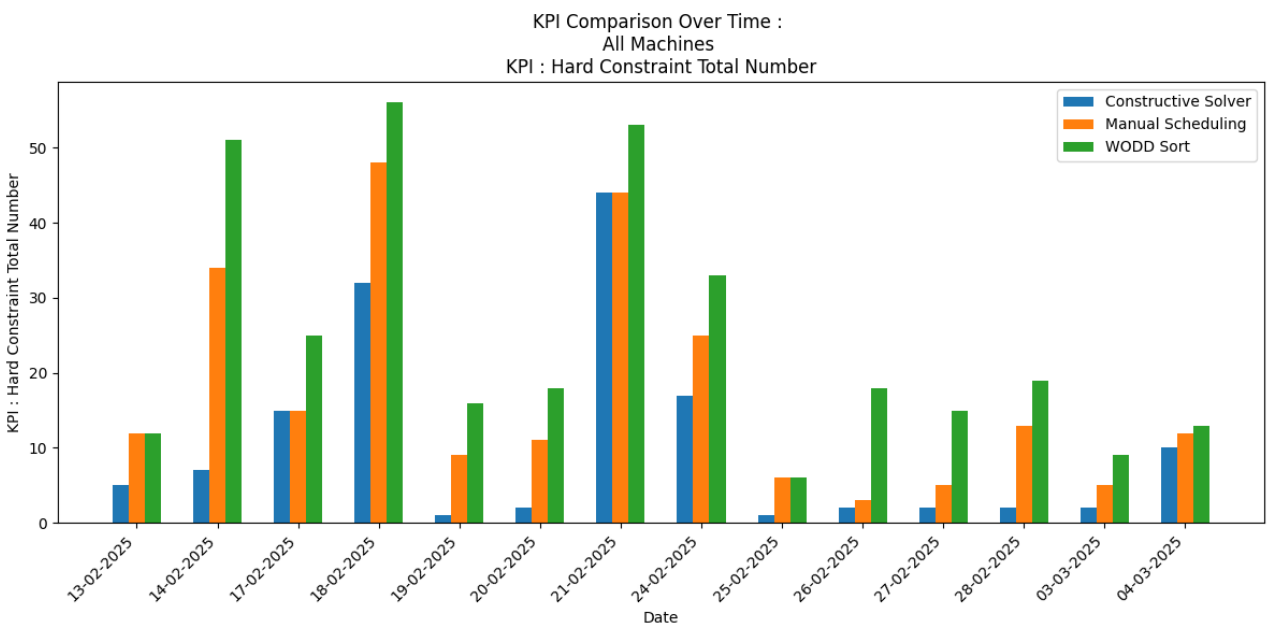

Figure 16 : Résultats détaillés du backtesting – contraintes non-respectées

La Figure 17 ci-dessous montre le nombre de contraintes non-respectées au total pour toutes les machines et pour tous les jours. Les contraintes sont : la disponibilité matière, les deadlines de livraison aux clients, les contraintes qualités, et les groupements de rainbow rolls. La tendance reste la même : le solveur respecte plus de contraintes que le scheduling manuel ou que le tri par défaut.

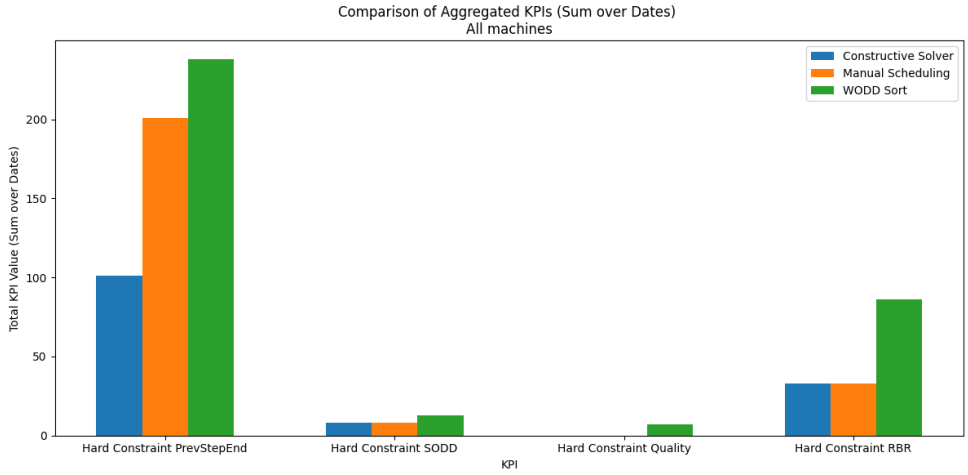

Figure 17 : Résultats agrégés du backtesting – contraintes non-respectées

Nombre de setups

Nous comparons également le nombre de setups. Les setups correspondent à un changement de master roll, boite, largeur, et longueur.

La Figure 18 ci-dessous montre le nombre total de setups sur les 8 machines pour chaque jour. Nous remarquons que le solveur est moins bon que le scheduling manuel. Cela est causé par le compromis entre le respect des contraintes et la minimisation des setups. Sans surprise, le tri par défaut est beaucoup moins bon que le scheduling manuel ou que le solveur.

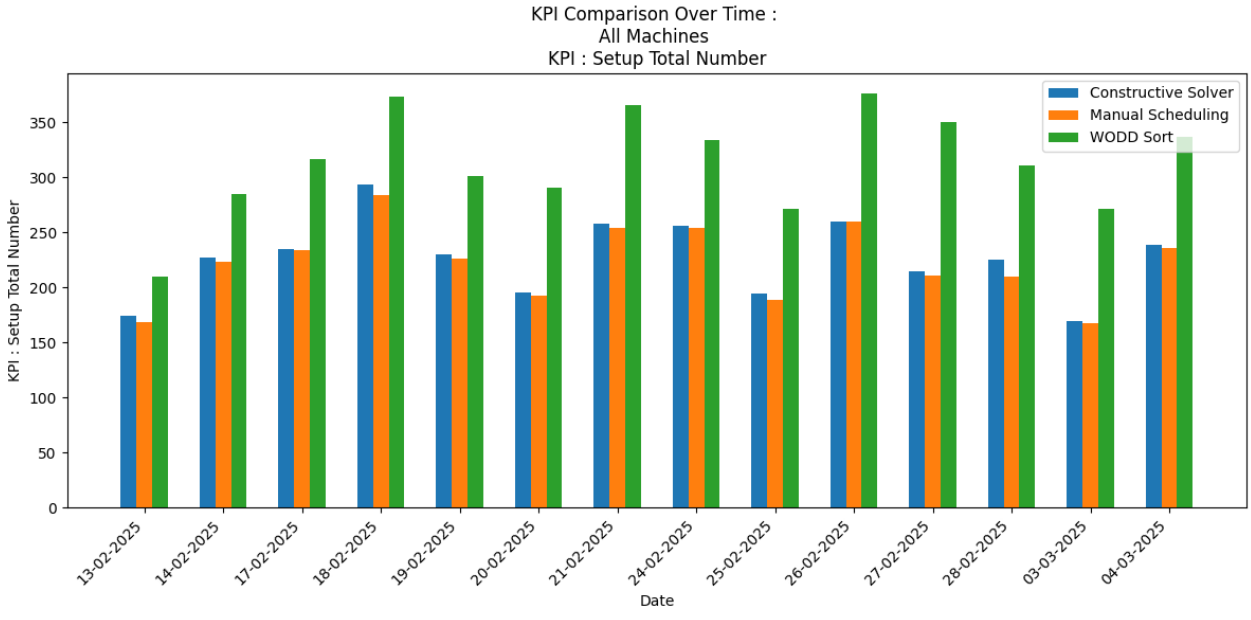

Figure 18 : Résultats détaillés du backtesting – setups

La Figure 19 ci-dessous montre le nombre de setups au total pour toutes les machines et pour tous les jours. La tendance reste la même, pour minimiser le nombre de setups : le solveur est meilleur que le tri par défaut mais est moins bon que le scheduling manuel.

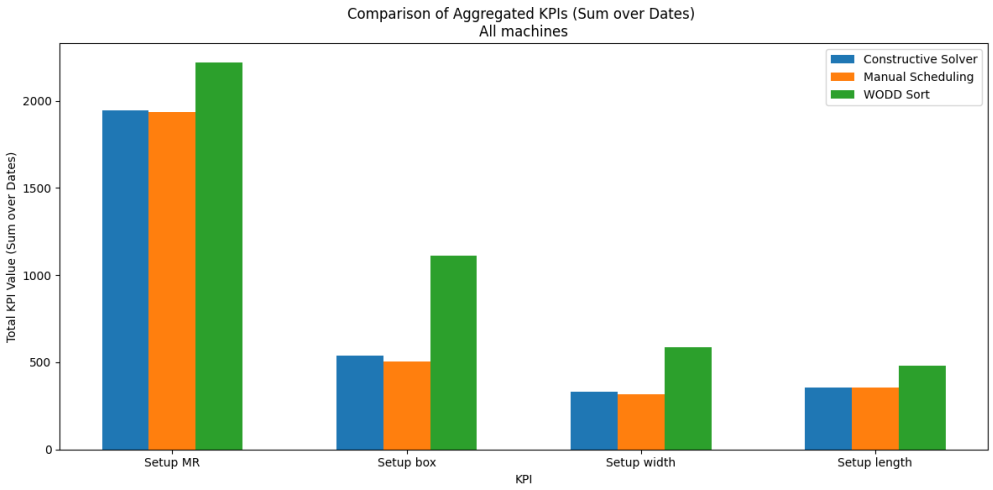

Figure 19 : Résultats agrégés du backtesting – setups

Conclusion intermédiaire

Pour conclure, les résultats du backtesting montrent que le solveur développé est bien meilleur qu'un tri par défaut. En effet, au total, le solveur mène à 59% de contraintes non-respectées en moins et à 72% de setups en moins par rapport à un tri par défaut.

De plus, le solveur développé est comparable au scheduling manuel au regard de sa qualité. En effet, au total, le solveur mène à moins de contraintes non-respectées (41% en moins) mais à plus de setups (2%). Cependant, le solveur construit les schedules en quelques secondes, beaucoup plus rapidement que le scheduling manuel.

Nous ne comparons pas les résultats sortis du modèle MILP avec ceux du solveur constructif car le modèle MILP ne peut pas être implémenté dans OMP pour le scheduling. L'objectif n'est pas d'obtenir les mêmes conclusions que les papiers consultés dans la revue scientifique avec nos données pratiques. L'objectif est de développer et d'implémenter un solveur pour scheduling qui sera utilisé par les planneurs dans OMP.

4.4. Implémentation dans OMP

Dans cette section, nous expliquons comment le solveur constructif a été implémenté et déployé dans OMP pour résoudre notre problème de scheduling.

Pour implémenter le solveur constructif dans OMP, des macros OPAL ont été écrites pour chaque coût local. Au total, j'ai écrit 17 macros OPAL pour calculer les coûts locaux. Après des tests dans l'environnement de développement OMP UAT, chaque macro OPAL a été compilée en fichier PAK et rendue disponible sur le serveur OMP PROD.

Les macros OPAL sont appelées à chaque évaluation d'un job candidat et implémente les coûts locaux du solveur constructif développé dans ce mémoire. Le job candidat sélectionné est le job ayant le plus petit coût local total pondéré.

La Figure 20 ci-dessous montre les noms et résumés de chaque macro OPAL implémentée pour les fonctions coûts.

CostBoxChange	Returns a unit cost if there is a box change
CostLengthChange	Returns a unit cost if there is a length change
CostLengthChangeCustom	Returns a unit cost if there is a length change that follows a custom logic
CostMRChange	Returns a unit cost if there is a Master Roll change
CostQualityExp	For products with a quality constraint, returns a negative cost linked exponentially with remaining hours to discourage tardiness
CostRainbowRollChange	Returns a unit cost if there is a rainbowroll change
CostSOTardinessExp	For products with a SO duedate, returns a negative cost linked exponentially with remaining hours to discourage tardiness
CostUrgencyTardyExp	Returns a negative cost linked exponentially with remaining hours to first out of stock duedate to discourage tardiness
CostWOEarlinessExp	Returns a positive cost linked exponentially with remaining hours to WO duedate to discourage earliness
CostWOTardinessExp	Returns a negative cost linked exponentially with remaining hours to WO duedate to discourage tardiness
CostWidthChange	Returns a unit cost if there is a width change
CostAvailability	Returns a unit cost if the component availability constraint is not respected
CostQualityLin	For products with a quality constraint, returns a negative cost linked linearly with remaining hours to discourage tardiness
CostSOTardinessLin	For products with a SO duedate, returns a negative cost linked linearly with remaining hours to discourage tardiness
CostUrgencyTardyLin	Returns a negative cost linked linearly with remaining hours to first out of stock duedate to discourage tardiness
CostWOEarlinessLin	Returns a positive cost linked linearly with remaining hours to WO duedate to discourage earliness
CostWOTardinessLin	Returns a negative cost linked linearly with remaining hours to WO duedate to discourage tardiness

Figure 20 : Programmation dans OMP – Macros OPAL pour coûts locaux

L'implémentation de chaque coût local dans une macro OPAL séparée permet ensuite de paramétrer le solveur constructif dans OMP sans modifier la moindre ligne de code. Le solveur constructif a été implémenté dans OMP PROD pour chaque profil de planneur finition. L'implémentation a été réalisée en suivant les étapes décrites dans la documentation fournie par OMP.

La Figure 21 ci-dessous montre l'onglet « Cost/Profit » de la fenêtre de paramétrisation du solveur constructif dans OMP.

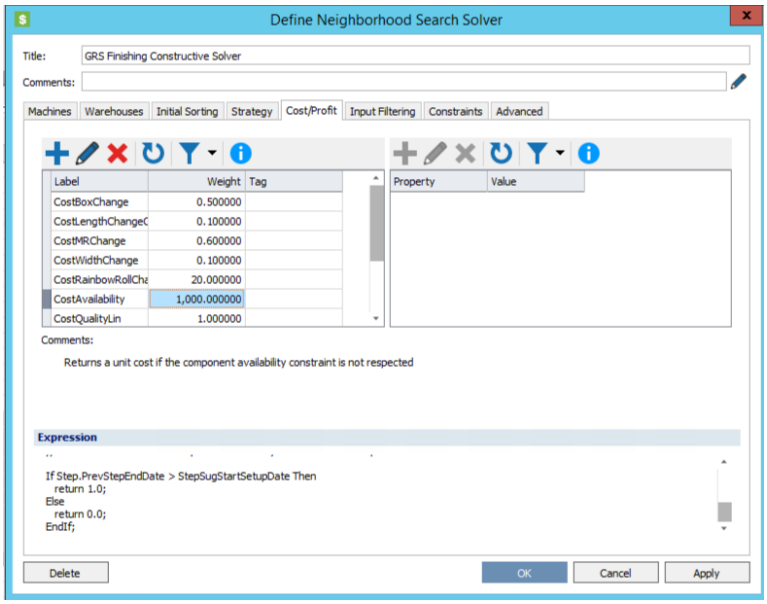

Figure 21 : Paramétrisation du solveur dans OMP

Une fois le solveur constructif déployé dans OMP, les paramètres coûts locaux obtenus par le fine-tuning ont été insérés. Le solveur déployé dans OMP PROD a été testé en collaboration avec les planneurs finition. Cette étape de test a permis un retour d'expérience, des modifications mineures, et surtout des explications supplémentaires. L'objectif de ce retour d'expérience était également d'augmenter la confiance des planneurs finition envers l'outil solveur.

Enfin, une documentation interne a été écrite à destination des planneurs finition.

Le solveur implémenté dans OMP résout le scheduling pour l'ensemble des WOs sélectionnés ayant un statut « planned » dès que les planneurs cliquent sur un bouton. Le solveur est rapide. En effet, il lui faut 20 secondes pour résoudre un schedule de 138 WOs en statut « planned » sur une machine.

La Figure 22 ci-dessous montre le solveur développé qui calcule un schedule optimal.

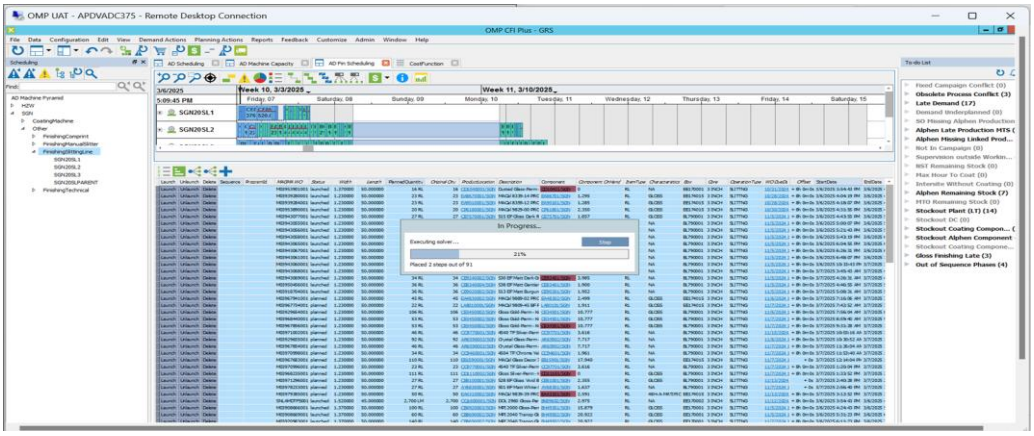

Figure 22 : Calcul d'un schedule optimal dans OMP

Conclusion générale

L'objectif de ce mémoire-projet était l'amélioration du système de planification OMP implémenté à Avery Dennison. L'amélioration exposée dans ce mémoire est le développement d'un solveur d'optimisation pour automatiser le scheduling de l'étape de finition.

D'abord, nous avons décrit la problématique en présentant l'entreprise, le système OMP, et le cahier des charges construit en collaboration avec les planneurs finition. Le problème d'optimisation résolu est un « single-machine job scheduling under early/tardy constraints with family-based setups ».

A partir de cette caractérisation, la revue de littérature scientifique a montré les approches classiquement utilisées pour résoudre ce problème d'optimisation. Dans la revue technique, les outils de programmation disponibles dans OMP ont été décrits.

Ensuite, un modèle MILP a été proposé pour résoudre le problème d'optimisation décrit dans le cahier des charges par un algorithme Branch and Bound. Pour un nombre de jobs modeste (36 jobs), la résolution par énumération n'a malheureusement pas donné de résultats concluants.

Pour résoudre rapidement le problème de scheduling, une approche heuristique constructive a été développée avec des coûts locaux pour la minimisation des setups et des contraintes non-respectées.

Une méthodologie fine-tuning et backtesting a été employée pour déterminer les paramètres locaux optimaux et pour valider le solveur. Les données de backtesting ont été récoltées pendant 3 semaines dans l'environnement de production d'OMP. Lors du fine-tuning, les paramètres locaux optimaux ont été obtenus par les algorithmes d'optimisation Direct et Basin-Hopping.

Les résultats de backtesting ont montré que le solveur fine-tuné était meilleur que le scheduling manuel pour respecter les contraintes mais moins bon pour minimiser les setups. Le scheduling manuel et le solveur sont tous les deux meilleurs que le tri par défaut.

Enfin, le solveur constructif a été implémenté dans OMP. Le solveur a ensuite été testé, validé et documenté lors d'une phase de retour d'expérience avec les planneurs finition. A titre d'exemple, le solveur dans OMP génère un schedule pour 138 jobs en 20 secondes.

Comme piste d'amélioration, un solveur par permutations et insertions pourrait être développé pour améliorer incrémentalement un schedule. Cependant, cela nécessiterait des développements supplémentaires pour des gains marginaux puisque le solveur constructif répond aux attentes des planneurs finition.

Une autre perspective serait le déploiement de cet outil sur les autres sites « graphic solutions » du groupe Avery Dennison. De la même manière, cet outil pourrait inspirer d'autres projets d'amélioration continue au sein de l'entreprise.

Bibliographie

1. *Home | Avery Dennison*. (s. d.). https://www.averydennison.com/en/home.html
2. Tamssaouet, K., & Dauzère-Pérès, S. (2023). A general efficient neighborhood structure framework for the job-shop and flexible job-shop scheduling problems. *European Journal Of Operational Research*, *311*(2), 455-471. https://doi.org/10.1016/j.ejor.2023.05.018
3. Adibi, M., Zandieh, M., & Amiri, M. (2009). Multi-objective scheduling of dynamic job shop using variable neighborhood search. *Expert Systems With Applications*, *37*(1), 282-287. https://doi.org/10.1016/j.eswa.2009.05.001
4. Xie, J., Li, X., Gao, L., & Gui, L. (2022). A new neighbourhood structure for job shop scheduling problems. *International Journal Of Production Research*, *61*(7), 2147-2161. https://doi.org/10.1080/00207543.2022.2060772
5. Zheng, Q., Dai, W., Peng, C., Wang, J., & Zhao, Y. (2024). A novel neighborhood structure for flexible job shop scheduling problem considering Quality-Efficiency coupling effect. *Computers & Industrial Engineering*, 110735. https://doi.org/10.1016/j.cie.2024.110735
6. Vanchipura, R., Sridharan, R., & Babu, A. S. (2013). Improvement of constructive heuristics using variable neighbourhood descent for scheduling a flow shop with sequence dependent setup time. *Journal Of Manufacturing Systems*, *33*(1), 65-75. https://doi.org/10.1016/j.jmsy.2013.07.003
7. Kumar, D. S., Kumar, P., & Nalini, S. (2024). Job scheduling using variable neighborhood search method. *AIP Conference Proceedings*, *3075*, 020101. https://doi.org/10.1063/5.0217197
8. Beck, J. C. (2007). Solution-Guided Multi-Point Constructive Search for Job Shop Scheduling. *Journal Of Artificial Intelligence Research*, *29*, 49-77. https://doi.org/10.1613/jair.2169
9. Hoffmann, J., Neufeld, J. S., & Buscher, U. (2024). Minimizing the earliness–tardiness for the customer order scheduling problem in a dedicated machine environment. *Journal Of Scheduling*, *27*(6), 525-543. https://doi.org/10.1007/s10951-024-00814-z
10. Koulamas, C., & Kyparisis, G. J. (2022). A classification of dynamic programming formulations for offline deterministic single-machine scheduling problems. *European Journal Of Operational Research*, *305*(3), 999-1017. https://doi.org/10.1016/j.ejor.2022.03.043
11. Hu, T., Tseng, S., & Allen, T. T. (2025). Dynamic programming-based exact and heuristic algorithms for single machine scheduling with sequence-dependent setups. *Expert Systems With Applications*, 126866. https://doi.org/10.1016/j.eswa.2025.126866
12. Yano, C. A., & Kim, Y. (1991). Algorithms for a class of single-machine weighted tardiness and earliness problems. *European Journal Of Operational Research*, *52*(2), 167-178. https://doi.org/10.1016/0377-2217(91)90078-a
13. De Athayde Prata, B., De Abreu, L. R., & Lima, J. Y. F. (2020). Heuristic methods for the single-machine scheduling problem with periodical resource constraints. *Top*, *29*(2), 524-546. https://doi.org/10.1007/s11750-020-00574-x
14. Almeida, M. T., & Centeno, M. (1998). A composite heuristic for the single machine early/tardy job scheduling problem. *Computers & Operations Research*, *25*(7-8), 625-635. https://doi.org/10.1016/s0305-0548(97)00097-x
15. Costa, A., Corsini, R. R., Pagano, D., & Fernandez-Viagas, V. (2024). A new composite heuristic to minimize the total tardiness for the single machine scheduling problem with

variable and flexible maintenance. *Computers & Operations Research*, 106849. https://doi.org/10.1016/j.cor.2024.106849
16. Suriyaarachchi, R. H., & Wirth, A. (2004). Earliness/tardiness scheduling with a common due date and family setups. *Computers & Industrial Engineering*, *47*(2-3), 275-288. https://doi.org/10.1016/j.cie.2004.07.006
17. Morais, R., Bulhões, T., & Subramanian, A. (2023). Exact and heuristic algorithms for minimizing the makespan on a single machine scheduling problem with sequence-dependent setup times and release dates. *European Journal Of Operational Research*, *315*(2), 442-453. https://doi.org/10.1016/j.ejor.2023.11.024
18. De Weerdt, M., Baart, R., & He, L. (2020). Single-machine scheduling with release times, deadlines, setup times, and rejection. *European Journal Of Operational Research*, *291*(2), 629-639. https://doi.org/10.1016/j.ejor.2020.09.042
19. Mazzini, R., & Armentano, V. A. (2001). A heuristic for single machine scheduling with early and tardy costs. *European Journal Of Operational Research*, *128*(1), 129-146. https://doi.org/10.1016/s0377-2217(99)00345-8
20. Schaller, J. E., & Gupta, J. N. (2006). Single machine scheduling with family setups to minimize total earliness and tardiness. *European Journal Of Operational Research*, *187*(3), 1050-1068. https://doi.org/10.1016/j.ejor.2006.06.061
21. Ronconi, D. P., & Kawamura, M. S. (2010). The single machine earliness and tardiness scheduling problem : lower bounds and a branch-and-bound algorithm. *Computational And Applied Mathematics*, *29*(2). https://doi.org/10.1590/s1807-03022010000200002
22. Lo, V. (2022, 1 juin). *Decision Diagrams and Large Neighbourhood Search for Earliness Tardiness Single Machine Scheduling with Sequence Dependent Setups*. http://hdl.handle.net/1807/123451
23. Poursheikh Ali, S., & Bijari, M. (s. d.). Minimizing Maximum Earliness and Tardiness on a Single Machine using a Novel Heuristic Approach. Dans *IEOM Society*. 3rd International Conference On Industrial Engineering And Operations Management (IEOM 2012), Turquie. https://ieomsociety.org/ieom2012/pdfs/266.pdf
24. *Welcome to the OMP Portal*. (s. d.). https://portal.omp.com/Login
25. *What is Direct Search ?* (s. d.). https://www.mathworks.com/help/gads/what-is-direct-search.html
26. Jones, D. R., & Martins, J. R. R. A. (2020). The DIRECT algorithm : 25 years Later. *Journal Of Global Optimization*, *79*(3), 521-566. https://doi.org/10.1007/s10898-020-00952-6
27. *direct — SciPy v1.15.2 Manual*. (s. d.). https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.direct.html#scipy.optimize.direct
28. Deng, G., & Ferris, M. C. (2007). Extension of the direct optimization algorithm for noisy functions. *2008 Winter Simulation Conference*, *49*, 497-504. https://doi.org/10.1109/wsc.2007.4419640
29. *basinhopping — SciPy v1.15.2 Manual*. (s. d.). https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.basinhopping.html#scipy.optimize.basinhopping
30. Ferreiro-Ferreiro, A. M., García-Rodríguez, J. A., Souto, L. A., & Vázquez, C. (2019). Efficient Model Points Selection in Insurance by Parallel Global Optimization Using Multi CPU and Multi GPU. *Business & Information Systems Engineering*, *62*(1), 5-20. https://doi.org/10.1007/s12599-019-00626-y

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management

Place des Doyens, 1 bte L2.01.01, 1348 Louvain-la-Neuve
Boulevard Emile Devreux 6, 6000 Charleroi, Belgique
Chaussée de Binche 151, 7000 Mons, Belgique

www.uclouvain.be/lsm