Ordonnancement de tâches sous contraintes sur des métiers à tisser

Mémoire

Alexandre Mercier-Aubin

Maîtrise en informatique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Ordonnancement de tâches sous contraintes sur des métiers à tisser

Mémoire

Alexandre Mercier-Aubin

Sous la direction de :

Jonathan Gaudreault, directeur de recherche
Claude-Guy Quimper, codirecteur de recherche

Résumé

Dans une usine de production de textile, il y a des métiers à tisser. Ces métiers à tisser peuvent être configurés de différentes façons. Des tâches doivent être exécutées sur ces métiers à tisser et le temps d'exécution d'une tâche est fonction du métier sur lequel elle est effectuée. De plus, chaque tâche est seulement compatible avec les métiers à tisser étant configurés de certaines façons. Un temps de mise en course peut permettre de configurer ou préparer un métier à tisser pour l'exécution d'une tâche. Le temps de mise en course est dépendant de la tâche qui précède et de celle qui suit. Nous souhaitons alors créer un horaire pour minimiser les temps de fabrication et les retards. Toutefois, certaines contraintes doivent être respectées. Lorsque des préparations surviennent sur des métiers différents en même temps, le nombre d'employés doit être suffisant. Un métier ne peut faire qu'une seule action à la fois. L'ordonnancement d'une seule machine est un problème NP-Difficile. Dans ce projet, il faut ordonnancer environ 800 tâches sur 90 machines dans un horizon de deux semaines, tout en respectant les contraintes de personnel. Des évènements stochastiques doivent être pris en compte pour obtenir un meilleur horaire. Le bris d'un fil n'étant pas un évènement rare, l'occurrence des bris est donnée sous la forme d'une loi de Poisson. Nous proposons alors une approche de résolution utilisant une heuristique de branchement basée sur le problème du commis voyageur. Cette approche permet d'obtenir de bonnes solutions pour le problème d'ordonnancement exploré. Les solutions trouvées sont 5 à 30% meilleures en termes de fonction objectif qu'une heuristique semblable à celle utilisée par l'équipe de planification de notre partenaire industriel. Nous présentons aussi un algorithme pour garantir la robustesse d'un horaire. Notre algorithme permet de générer des horaires plus réalistes et qui résistent bien aux évènements imprévus. La combinaison de ces deux pratiques mène à l'intégration et l'utilisation du produit final par notre partenaire industriel.

Abstract

In a textile factory, there are looms. Workers can configure the looms to weave different pieces of textiles. A loom can only weave a piece of textiles if the piece of textiles is compatible with its loom configuration. To change its configuration, a loom requires a setup. The setups are performed manually by workers. There are also sequence-dependent setups to prepare a loom for the upcoming piece of textiles. We wish to minimize the setups duration and the lateness. A solution must satisfy some constraints. The problem is subject to cumulative resources. The quantity of workers simultaneously configuring machines can't exceed the total number of employees. A loom can only weave a piece of textiles at a time. Scheduling tasks on a single loom is an NP-Hard problem. In this project, we must schedule tasks an average of 800 tasks on 90 looms with a two-week horizon. Stochastic events might occur and must be accounted for. We must design an algorithm to create robust schedules under uncertainty. As a thread breaking during the weaving process isn't a rare occurrence, a better schedule could greatly impact the performances of a company when applying the schedule to a real situation. We formulate that the number of breaks per task follows a Poisson distribution. First, we propose a branching heuristic based on the traveling salesperson problem in order to leverage computation times. The solutions found are 5 to 30% better according to their objective function than the ones of a greedy heuristic similar to what our industrial partner uses. We also present a filtering algorithm to guarantee robustness of solutions in respect to a confidence level. This algorithm improves robustness and creates more realist schedules. The algorithm is also efficient in computation time by achieving bound consistency in linear time. Combining both these techniques leads to the integration of our research in the decision system of our industrial partner.

Table des matières

Liste des tableaux

Liste des figures

Liste des abbréviation et sigles

— SAT - Satisfiabilité

— CP - Programmation par contraintes

— LNS - Recherche locale à grand voisinage

— TSP - Commis voyageur

L'intelligence dans tous ses artifices
brille au ciel des sciences de
l'information.

Mon père

Remerciements

Je profite de cette section pour remercier Philippe Marier et Ludwig Dumetz d'avoir contribué à diverses parties du projet pour ainsi me permettre de me concentrer sur les éléments pertinents à ma recherche. Ce fut une belle expérience de travailler en collaboration avec ces modèles de professionnalisme.

Je remercie personnellement mes directeurs de recherche, Jonathan Gaudreault et Claude-Guy Quimper de m'avoir assisté tout au long de ma recherche en fournissant de précieuses idées. Grâce à eux, j'ai eu la chance de développer plusieurs compétences amorçant mon parcours pour devenir un meilleur chercheur.

Nous remercions notre partenaire industriel pour les jeux de données, la motivation de la recherche, ainsi que le support financier reçu en collaboration avec Mitacs. Cette recherche a reçu le soutien de Mitacs dans le cadre du programme Mitacs Accélération.

Nous remercions aussi les évaluateurs du mémoire pour leur temps et implication dans le processus de soumission du mémoire.

Avant-propos

Nous présentons le résultat d'une recherche réalisée en collaboration avec une entreprise québécoise. Cette recherche a mené à deux articles acceptés et publiés dans des conférences de programmation par contraintes.

Le premier article *Leveraging Constraint Scheduling : A Case Study to the Textile Industry* sera publié le 21 septembre 2020 dans la conférence : 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR). Aucune modification n'a été effectuée dans le mémoire par rapport à l'article. J'ai le statut de premier auteur. Dans cet article, j'ai apporté des idées, implémenté les solutions, fait des bancs d'essai standards, ainsi que la rédaction de l'article. Jonathan Gaudreault et Claude-Guy Quimper sont mes professeurs en co-supervision.

Le second article *The Confidence Constraint : A Step Towards Stochastic CP Solvers* sera publié le 7 septembre 2020 dans la conférence : 26th International Conference on Principles and Practice of Constraint Programming (CP). Aucune modification n'a été effectuée dans le mémoire par rapport à l'article. J'ai le statut de premier auteur. Dans cet article, j'ai apporté des idées, conçu l'algorithme, fait les bancs d'essai, ainsi que la rédaction de l'article. Le premier coauteur, Ludwig Dumetz, a conçu le modèle de simulation utilisé lors des bancs d'essai. Jonathan Gaudreault et Claude-Guy Quimper sont mes professeurs en co-supervision.

Introduction

Pour assurer la performance d'une entreprise de production, plusieurs décisions doivent être prises. Parmi les décisions importantes, il y a la planification et l'ordonnancement de la production. Ces décisions sont cruciales pour maximiser les profits.

Selon le nombre de tâches à planifier et ordonnancer, il peut être difficile et fastidieux d'obtenir de bonnes solutions à la main. Trouver de bonnes solutions peut aussi s'avérer difficile pour un ordinateur. Pour certains problèmes, le temps de calcul des solutions peut augmenter de façon exponentielle en fonction de l'augmentation du nombre de paramètres en entrée. Nous parlons alors d'explosion combinatoire.

Il existe des logiciels et des modèles mathématiques qui modélisent des problèmes d'ordonnancement. Toutefois, ces outils ne sont pas adaptés à n'importe quels problèmes des entreprises. En effet, certains problèmes spécifiques peuvent nécessiter des approches spécifiques pour gérer l'explosion combinatoire.

Notre partenaire industriel a alors contacté le Consortium de Recherche en Ingénierie des systèmes industriels 4.0 dans l'optique de formuler un modèle correspondant à leurs besoins.

Nous présentons une façon d'automatiser les décisions de planification et d'ordonnancement, tout en conservant une certaine qualité de résultats et des délais raisonnables pour notre partenaire industriel. Nous souhaitons obtenir un plan et un horaire de qualité comparable à l'humain dans des délais plus courts. Nous explorerons la problématique proposée par notre partenaire industriel : la planification et l'ordonnancement des tâches sur des métiers à tisser. Dans ce mémoire, nous allons principalement nous concentrer sur l'ordonnancement.

Prenons une usine de textile ayant 90 métiers à tisser. Ces métiers ont été achetés sur plusieurs dizaines d'années. Il y a alors différents métiers de différentes marques et différentes technologies plus ou moins récentes. Nous catégorisons alors les métiers par *type*.

Chaque métier à tisser peut être configuré de certaines façons. L'ensemble des *configurations* possibles peut varier selon le métier à tisser. Une pièce de textile peut seulement être tissée sur un métier à tisser configuré de la bonne façon pour ce textile. Selon la façon dont un métier est configuré, le temps de tissage varie. En tout temps, un métier à tisser peut uniquement tisser une pièce de textile à la fois.

Entre deux pièces de textiles, un métier doit être préparé. La durée pour préparer le métier se nomme un *temps de mise en course* (en anglais : *setup*). Il est possible de changer la configuration d'un métier, ceci requiert un temps de mise en course de longue durée. Nous appelons ces temps, des *temps de mise en course majeurs*. Il est à noter qu'un métier n'a pas plus d'un temps de mise en course majeur par deux semaines. Pour la transition entre des pièces de textile qui ne nécessitent pas de changement de configuration, les temps de mise en course sont plus courts. Nous les appelons alors des *temps de mise en course mineurs*. Les temps de mise en course mineurs sont variables selon l'ordre des pièces de textile à tisser. Il est alors impératif de fournir un bon ordre de tissage pour réduire les temps de mise en course. Ces temps sont composés de plusieurs interventions de courte durée par des employés spécialisés. Il doit donc y avoir l'intervention d'un tisserand, suivi d'un attacheur et potentiellement d'un mécanicien. Cet ordre est toujours respecté, toutefois certaines professions ne sont pas toujours nécessaires. De plus, nous pourrons utiliser les caractéristiques de motifs et de couleurs pour déterminer que deux pièces de textiles sont les mêmes.

Nous souhaitons produire un bon horaire de production. L'*horaire* est un résultat de la planification et de l'ordonnancement des tâches. Cet horaire doit contenir des informations quant à la date de début et la durée de tissage de chaque pièce de textile, ainsi que des temps de mise en course. L'assignation des pièces de textiles aux métiers doit aussi s'y retrouver. Un horaire valide doit prendre en compte la quantité d'employés et s'assurer ne jamais la dépasser. Pour notre partenaire industriel, les retards sont un enjeu. Un bon horaire minimise les retards en fonction de la priorité des pièces de textiles à tisser. Plus généralement, en réduisant les temps de mise en course, nous améliorons aussi la productivité de l'entreprise.

Nous pouvons voir la conception de cet horaire comme deux étapes distinctes. L'étape de *planification* consiste à choisir quelle pièce de textile est tissée sur quel métier à tisser. Par la suite, l'étape d'*ordonnancement* permet de choisir l'ordre des tâches, ainsi que leur date de début de tissage. Séparer la conception d'horaires en deux étapes permet généralement de réduire le temps de calcul, ce qui peut faire la différence entre trouver des solutions ou non.

Pour obtenir un bon horaire, il faut considérer les évènements imprévus pendant l'étape d'ordonnancement. Lors de la confection d'une pièce de textile, il y a deux types de fils. Sur le métier à tisser, les fils sont placés dans des sens différents. Un *fil de trame* est placé dans le sens de la largeur, tandis qu'un *fil de chaîne* est en longueur. Les *évènements stochastiques* du problème sont les bris des différents fils. Les probabilités de bris des deux types de fils sont dépendantes de la pièce de textile à tisser et ont des chances de survenir différentes selon le type de fil brisé.

Généralement, le bris d'un fil de trame a une haute probabilité de survenir, mais est rapide à réparer, même parfois automatique. Le bris d'un fil de chaîne est plus rare, mais est plus long et requiert l'intervention manuelle d'un employé. Plusieurs bris d'un même type peuvent survenir lors du tissage d'une pièce de textile.

Le partenaire industriel embauche quatre employés à temps plein pour concevoir les horaires manuel-

lement. Les employés n'ont généralement pas le temps de planifier et d'ordonnancer la production des pièces de textiles au-delà d'une semaine. Beaucoup de tâches doivent être ordonnancées et peu d'employés sont disponibles pour préparer les métiers à tisser. Il est donc difficile pour les humains de trouver un bon horaire. Pour les ordinateurs, il est aussi difficile de trouver le meilleur horaire. Aucun modèle déjà écrit dans la littérature ne modélise le problème de notre partenaire industriel. Avoir un modèle adapté peut permettre de trouver de meilleures solutions comparées à des solutions génériques ou à ce que les planificateurs peuvent faire.

En tant qu'équipe, nous avons décidé de résoudre le problème en le séparant en plusieurs parties. Chaque partie se spécialise dans un aspect précis de la résolution. Concevoir un modèle se chargeant de la planification et de l'ordonnancement impliquerait une grande quantité de variables et de contraintes. Comme le problème est sujet à l'explosion combinatoire, cela signifie de longs temps de calcul. Nous avons décidé de séparer les décisions de la planification et d'ordonnancement afin de réduire la taille des problèmes et ainsi d'atténuer la courbe de temps de calcul.

Avant le début de ma maîtrise, Philippe Marier a conçu un modèle de planification. J'ai dû réécrire le modèle mathématique dans un nouveau langage de modélisation et l'adapter aux besoins changeants de notre partenaire industriel. Par la suite, mon projet de maîtrise a débuté concernant principalement le modèle mathématique d'ordonnancement. Pour valider les horaires, Ludwig Dumetz a conçu un modèle de simulation. J'ai donc assuré le lien et le transfert des données entre les trois modèles.

Grâce au projet, notre partenaire aura en main un ensemble d'outils d'aide à la décision, adaptés à sa réalité, qui facilite grandement le travail de ses planificateurs tout en guidant mieux les actions de son équipe d'amélioration continue. La compagnie aura de plus la possibilité d'exploiter de tels outils pour tester de nouvelles réalités d'affaires et en évaluer leurs effets. Notre système permet l'automatisation de décisions opérationnelles qui était auparavant faite par des employés. Ces employés peuvent alors se concentrer sur des tâches plus importantes quant à l'amélioration du rendement de l'entreprise.

Nous apporterons au domaine plusieurs contributions. En ordonnancement, nous décrivons un nouveau problème spécifique et nous proposons une approche de résolution. Nous proposons aussi une nouvelle contrainte pour résoudre les problèmes de nature stochastique, ainsi qu'un algorithme de filtrage en temps linéaire pour la résoudre. Un algorithme de filtrage a comme but de retirer des valeurs du domaine d'une ou plusieurs variables. Nous expliquerons davantage le concept de filtrage dans la revue de littérature.

Ce mémoire décrit les travaux réalisés. La suite du mémoire est organisée comme suit. Dans le chapitre 1, nous montrons l'approche générale de résolution, comportant entre autres les différents modèles, leurs fonctions, une description sommaire de nos stratégies de résolution et l'échange de données entre chaque partie. Par la suite, le chapitre 2 contient les informations nécessaires à la compréhension de la recherche effectuée lors de la maîtrise, ainsi que la revue de littérature. Comme le modèle de planification est important au problème, mais ne fait pas partie du sujet principal de ma recherche, il est situé en annexe A. Dans le premier article, au chapitre 3, nous présenterons les techniques pour

résoudre le problème sous forme déterministe, ainsi que la modélisation mathématique du problème d'ordonnancement. Dans le second article, au chapitre 4,nous montrerons un nouvel algorithme pour obtenir des solutions plus résistantes aux évènements imprévus. Nous présenterons les contributions qui ont été faites en entreprise au chapitre 5 en décrivant l'intégration du contenu de la recherche dans l'entreprise, ainsi que l'évolution du projet depuis la soumission des deux articles. Finalement, j'ai rédigé un document technique de référence pour notre partenaire industriel quant à l'utilisation du système issue de la recherche. Ce document se retrouve dans l'annexe B.

Chapitre 1

Cadre général de résolution

Globalement, le projet proposé par notre partenaire industriel est divisé en plusieurs parties ayant différents contributeurs. Ces contributeurs sont des membres du Consortium de Recherche en Ingénierie des Systèmes Industriels 4.0. En tant qu'équipe, nous avons conçu un système pour résoudre le problème de notre partenaire industriel. Malgré mes contributions importantes dans les autres modèles, le sujet principal de ma maîtrise est le modèle d'ordonnancement. Cette partie est abordée plus en détail dans les deux articles intégrés au mémoire. Nous présentons les équations mathématiques du modèle sous leur forme déterministe dans le chapitre 3. La version prenant en compte les divers évènements stochastiques est présentée dans le chapitre 4.

Dans ce chapitre, nous présentons un schéma des interactions entre les différentes parties du système, une interaction typique et nous expliquons plus en détail chaque partie du cadre de résolution.

1.1 Interactions

Dans une exécution normale du système (tel que présenté dans la figure 1.1), le procédé de conception d'horaires devrait se produire dans l'ordre décrit ci-dessous. Notre partenaire industriel (1) nous envoie un jeu de données (2) qui contient les données nécessaires à la planification (3) et les données nécessaires à l'ordonnancement (4).

Une fois le jeu de données reçu, il faut résoudre le modèle de planification (5). Ce modèle agit comme une boîte noire auprès de l'entreprise. Les résultats de la planification (6) sont enregistrés.

Quand les résultats de la planification (6) sont écrits, c'est le tour du modèle d'ordonnancement (7) d'être exécuté. Encore une fois, ce modèle agit comme une boîte noire pour l'entreprise. Le modèle reçoit en entrée le jeu de données complet (3,4), ainsi que les résultats de la planification (6). Une fois son exécution terminée, le modèle d'ordonnancement offre des résultats (8) qui sont extraits sous forme d'horaire (8). Cet horaire peut être envoyé à notre partenaire industriel (13) ou analysé par le modèle de simulation (10).

Le modèle de simulation (10) reprend l'horaire (8) et valide que son application en entreprise est judicieuse. Le modèle de simulation produit des métriques et lignes du temps avec aléas (11). Ces métriques peuvent être extraites (12) et analysées par le partenaire industriel (13).

Dans la figure 1.1, les flèches pointillées représentent l'utilisation de données par un modèle. Par exemple, le modèle d'ordonnancement (7) utilise les données de sortie de la planification (6). Les flèches pleines représentent un flux d'information. Par exemple, le modèle de planification (5) produit les sorties de la planification (6).

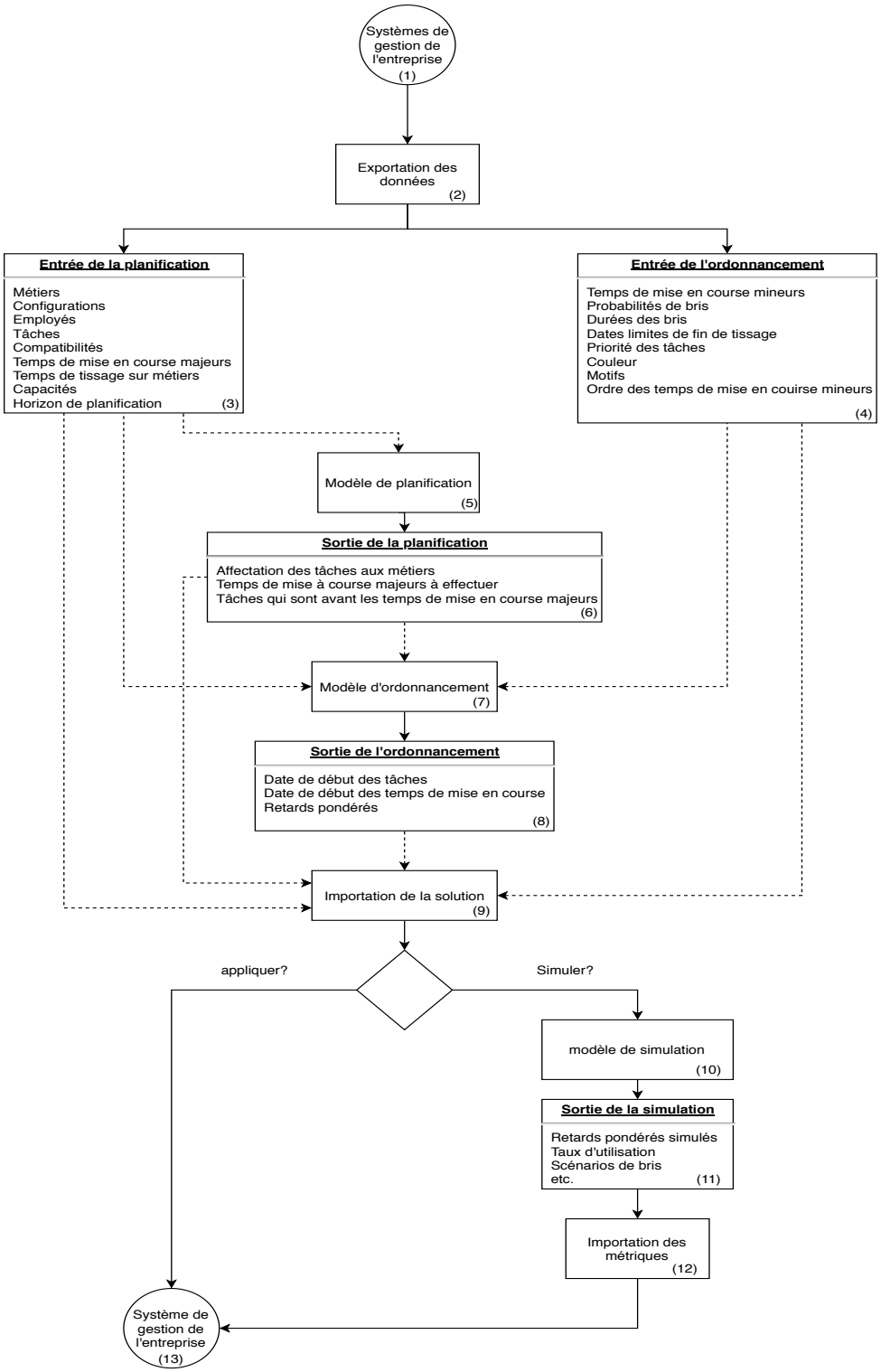


FIGURE 1.1 – Schéma des interactions

1.2 Description sommaire de l'approche

Le système développé comporte trois modèles principaux. Le modèle de planification (5) et le modèle d'ordonnancement (7) sont constitués d'équations mathématiques. Résoudre ces équations donne des horaires respectant les contraintes proposées par le partenaire industriel. Comme un modèle comportant la planification et l'ordonnancement comporterait beaucoup de variables et de contraintes, nous avons décidé de séparer les décisions de la planification et d'ordonnancement afin de réduire la taille des problèmes et ainsi d'atténuer la courbe de temps de calcul. Toutefois, cette décomposition restreint l'espace de solutions accessibles. Ainsi, en théorie, on ne peut plus prétendre à l'optimalité. On fait cependant le pari que pour un temps de calcul fixe, on obtiendra de meilleures solutions.

Le modèle de planification (5) choisit quelles pièces de textiles devront être fabriquées sur quels métiers à tisser, ainsi que les temps de mise en course majeurs. L'objectif de ce modèle est de minimiser la durée des tâches et diminuer les temps de mise en course majeurs. Ce modèle ignore les temps de mise en course mineurs et les dates limites. Toutefois, nous prenons le risque que si l'étape de planification (5) offre de mauvais résultats, alors l'ordonnancement (7) soit moins performant. Supposons que la planification (5) obtient un choix de métier optimal pour chaque tâche. Comme nous ne considérons pas les dates limites de fin de tissage dans la planification, il n'y a aucune différence entre une planification optimale où les tâches sont bien réparties sur les métiers et une planification optimale où les tâches sont condensées sur un même métier. Par contre, lors de l'ordonnancement (7), le plan où les tâches sont condensées est plus sujet aux retards.

Par la suite, le modèle d'ordonnancement (7) récupère l'affectation des tâches aux métiers et agit en conséquence. Le modèle d'ordonnancement permet de choisir l'ordre de tissage, les moments dans le temps où les machines tissent les différentes pièces de textiles et les moments où sont effectuées les mises en courses. L'horaire résultant doit minimiser les retards pondérés en fonction de la priorité de la tâche en retard.

Le modèle de simulation (10) nous permet d'évaluer la qualité des algorithmes qui génèrent les horaires. Ce modèle est différent, car il est constitué d'un environnement 3D à l'échelle de l'usine. Le modèle de simulation génère des aléas et fournit des métriques de performances sur la robustesse de l'horaire en contexte réaliste. Le nombre d'employés, les distances à parcourir, le nombre de métiers à tisser et tous autres éléments qui peuvent influencer l'horaire sont pris en compte. Non seulement le modèle permet de valider l'horaire, mais il peut aussi être utilisé comme un outil d'aide à la décision pour les décisions stratégiques de l'entreprise. Ces résultats peuvent être interprétés avant d'utiliser l'horaire. Si les résultats de l'analyse des métriques ne sont pas convenables selon l'équipe de notre partenaire, certaines variables peuvent être modifiées lors de la conception de l'horaire pour générer un horaire plus adapté à leurs besoins. Ces variables incluent entre autres le nombre d'employés, le nombre de métiers, etc. Dans les contextes où le modèle d'ordonnancement ne trouve pas la solution optimale, le temps limite de calcul devient aussi un paramètre à choisir. En effet, il est possible d'exécuter les trois modèles en changeant le nombre de métiers, le nombre d'employés, etc. Cela permet

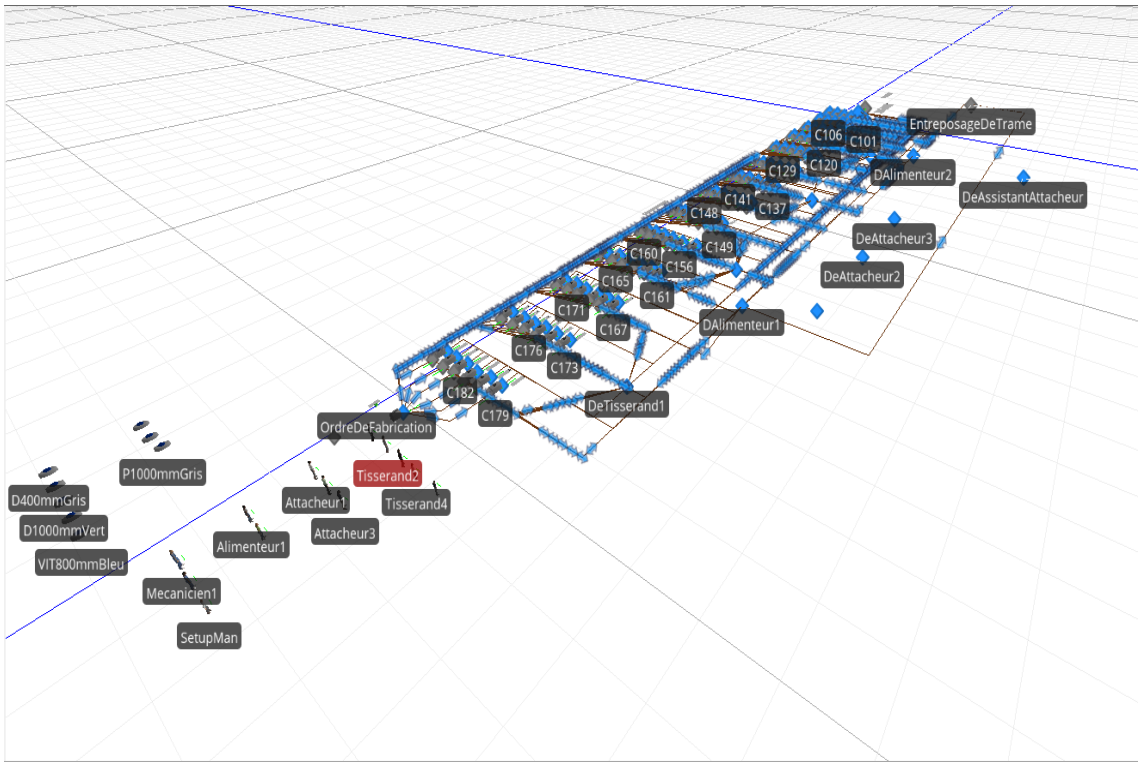


FIGURE 1.2 – Maquette 3D

alors de voir directement l'impact des choix de l'entreprise sur la production des textiles.

À la figure 1.2, une représentation en trois dimensions de l'usine est présentée. Il est possible d'apercevoir les employés en bas à gauche. La disposition des métiers à tisser est située dans le centre. Comme l'image est condensée, les étiquettes montrant les noms des métiers à tisser ne sont pas toutes affichées. Une fois le modèle exécuté, il est possible de voir les employés se déplacer et interagir avec les métiers. Cette simulation permet à l'équipe de notre partenaire industriel de visionner le comportement des employés. L'aspect visuel peut être visionné en temps réel, à l'accéléré ou même totalement retiré afin d'éviter des temps de calculs pour le rendu graphique. Alors que notre partenaire industriel peut visionner les rendus graphiques, nous souhaitons analyser le comportement des horaires en contexte stochastique. Nous choisissons alors de ne pas afficher les rendus graphiques lors de nos bancs d'essai standards du chapitre 4. L'utilisation du simulateur permet de répondre à certaines questions comme les suivantes. Quelle technique de réparation est meilleure ? Est-ce que l'on devrait répartir les machines à réparer entre employés ou tous les employés devraient pouvoir réparer n'importe quel métier à tisser ? Connaître ces informations permet d'améliorer le rendement de l'entreprise en conservant le même nombre d'employés et de métiers.

Les modèles doivent être appelés séquentiellement, car ils utilisent les résultats produits par les autres modèles dans un ordre précis. Dans la figure 1.1, nous présentons de haut en bas l'ordre de résolution des modèles, ainsi que les interactions sous forme de données entre les différents modèles. Ces

interactions sont aussi décrites dans les sous-sections plus bas.

Notre partenaire industriel nous fournit des données qui sont utilisées par le modèle de planification et d'ordonnancement, ainsi que des données utilisées uniquement par le modèle d'ordonnancement.

1.3 Entrées et sorties du modèle de planification

Un jeu de données fourni par notre partenaire industriel doit avoir les informations (3) suivantes pour résoudre le modèle de planification (5). Il doit y avoir une liste de tous les métiers disponibles, ainsi que les configurations que peuvent prendre ces métiers. Il doit y avoir une liste des pièces de textiles à tisser (tâches) et la compatibilité de ces tâches avec les différentes configurations. Pour connaître les temps de tissage, il faut obtenir une matrice en deux dimensions comportant les temps de tissage des pièces sur les différents métiers. Il faut aussi une liste des employés, des professions des employés, la capacité en temps des métiers et la capacité en temps des employés. Pour connaître les temps de mise en course majeurs, il faut avoir une matrice en deux dimensions présentant pour chaque métier, le temps que peut prendre le passage vers une autre configuration que celle initiale. Un paramètre d'horizon est requis. Ce paramètre représente la ligne du temps sur laquelle la planification est effectuée. Malgré qu'il puisse être changé, il a été décidé que ce paramètre représente toujours deux semaines en heures ouvrables, soit 240 heures de travail. Par détail technique, nous convertissons toutes les unités en minutes afin de les encoder en valeurs entières et garder une précision raisonnable.

En sortie (6), une solution comporte quelle tâche devra être exécutée sur quel métier à tisser, les temps de mise en course majeurs à effectuer et le partitionnement des tâches avant ou après les temps de mise en course majeurs des métiers. Ces informations sont utilisées par le modèle d'ordonnancement.

1.4 Entrées et sorties du modèle d'ordonnancement

Pour le modèle d'ordonnancement (7), nous prenons en entrée les données d'entrées de la planification (3), les sorties de la planification (6), ainsi que les nouvelles entrées (4) décrites ci-dessous. Comme l'ordre des tâches influence les temps de mise en course mineurs, nous avons besoin d'une matrice carrée donnant leur durée. Nous notons que la matrice n'est pas symétrique. Par exemple, couper des fils est plus rapide que de les attacher. La durée est fonction de la pièce tissée avant et celle après le temps de mise en course mineurs. Il peut y avoir jusqu'à trois employés de professions différentes. Nous utilisons comme entrée un tuple, soit une liste d'éléments finie pour encoder le lien entre les professions et le temps requis par cette profession pour un temps de mise en course mineur. En d'autres mots, nous avons une matrice carrée ayant en x et en y des pièces de textiles et dont les valeurs sont des tuples. Comme nous souhaitons minimiser le retard pondéré, il faut avoir des informations sur les dates de livrable des pièces de textiles et leur priorité. Nous savons aussi que l'ordre dans lequel les employés de différentes professions interagiront avec un métier à tisser est toujours le même dans le cadre d'un temps de mise en course mineur. Nous devons alors avoir cet ordre sous forme de liste. Afin d'éliminer

des solutions équivalentes en termes de retards pondérés, nous avons aussi accès à la couleur et les motifs des pièces. En effet, il n'y a pas de différence entre deux pièces ayant ces caractéristiques en commun. Par la suite, nous avons besoin d'une liste probabilités de bris et de leur durée lorsqu'ils surviennent selon le métier à tisser et le tissu confectionné. Cette liste doit être filtrée en fonction des résultats de la planification pour conserver uniquement les entrées qui correspondent à l'assignation des métiers. Les données relatives au bris permettent de retirer des horaires considérés risqués selon notre partenaire industriel.

En sortie (8), nous avons un horaire répondant aux contraintes du problème. Cet horaire est donné sous forme d'une liste des dates de début de tissage des pièces de textiles, ainsi qu'une liste des dates de début de mises en course mineurs et majeurs.

1.5 Entrées et sorties du modèle de simulation

Le modèle de simulation (10) reprend les entrées et sorties des deux autres modèles (3, 4, 6, 8). Ces données permettent à la simulation de savoir quand commencer la confection d'un tissu et sur quel métier à tisser. Pour les aléas, les probabilités sont extraites de l'ordonnancement. Une moyenne des probabilités de bris est faite par métier. Nous perdons alors un peu de précision, mais ceci rend les taux compatibles avec le logiciel de simulation utilisé pour la simulation.

Les sorties du modèle de simulation (11) sont diverses métriques de performances. Par exemple, dans le chapitre 4, nous utilisons le retard pondéré moyen simulé. Ce retard est essentiellement la moyenne des retards pondérés de chaque itération de la simulation.

Nous définissons un scénario de bris comme un horaire avec bris faisant partie de l'ensemble des horaires avec bris étant possible. Le modèle de simulation génère un scénario de bris par itération. Nous pouvons choisir le nombre d'itérations manuellement. Nous prenons 100 itérations pour nous assurer une qualité des résultats. Tous ces scénarios peuvent être utilisés pour des analyses de performances.

Chapitre 2

Concepts préliminaires

Dans ce chapitre, nous expliquons plusieurs concepts nécessaires à la compréhension des contributions scientifiques apportées dans ce mémoire. Nous commençons par un résumé des techniques utilisées pour l'ordonnancement de la production de textiles et comment notre problème diverge des contributions déjà existantes. Par la suite, nous expliquons ce qu'est la classe non polynomiale déterministe dont fait partie notre problème et comment cette classe influence les techniques à utiliser pour résoudre le problème. Nous présentons ensuite, de façon générale, la forme d'un problème d'ordonnancement. Nous faisons le lien entre les problèmes d'ordonnancement et le problème du commis voyageur. Ce lien est principalement utilisé dans notre premier article pour obtenir rapidement des solutions de qualité. Nous présentons aussi la programmation par contraintes, soit la technique de résolution qui est présentement l'état de l'art en ordonnancement [21]. Nous expliquons la technique de recherche à grand voisinage qui offre généralement de bons résultats en ordonnancement. Finalement, nous expliquons sommairement les différentes techniques d'optimisation stochastique.

2.1 L'ordonnancement en production de textiles

Le premier auteur à s'attaquer à un problème de production de textile sur plusieurs métiers fut Serafini [39]. Cet auteur propose un algorithme permettant de résoudre le problème d'ordonnancement des tâches sur les métiers à tisser en temps polynomial, si tous les métiers sont du même type, si les tâches sont préemptives et indépendantes. Une tâche est considérée comme préemptive si elle peut être arrêtée et reprise plus tard ou ailleurs. Malheureusement, le problème de notre partenaire industriel ne respecte pas ces contraintes.

Wang *et al.* [46] soulignent les nombreuses erreurs pouvant survenir lorsqu'une équipe planifie et ordonnance manuellement les tâches. Ces erreurs incluent : manquer d'employés à un instant donné, sous-estimer le temps que vont prendre les tâches et ne pas avoir assez de fils pour tisser. Ils soulèvent des difficultés dans la résolution de problème par un ordinateur. Ce problème apporte plusieurs difficultés de résolution pour un ordinateur. Ces difficultés incluent les optimums locaux que nous expliquons dans la section 2.5.7 et le nombre de solutions possibles au problème que nous expliquons

dans la section 2.2.

D'autres auteurs se sont penchés sur le même problème, mais en souhaitant minimiser les temps de mise en course mineurs [44] ou minimiser les délais de livraison [19]. Ces méthodes reposent sur des métaheuristiques pour obtenir des solutions en temps raisonnables. Certaines heuristiques de recherche locales ont aussi été explorées [41].

Aucune de ces contributions ne modélise exactement le problème de notre partenaire industriel, notamment en ce qui concerne l'aspect stochastique qui n'est pas exploré dans ces articles. Toutefois, certains auteurs se sont attaqués à des problèmes similaires du point de vue stochastique [36, 47]. Nous souhaitons alors adapter à notre problème certains concepts comme les contraintes de chance que nous expliquons dans la sous-section 2.7.4.

2.2 La classe non déterministe polynomiale

2.2.1 Définition

La *classe non déterministe polynomiale* (NP) est l'ensemble des problèmes de décision ayant un algorithme permettant de vérifier leur solution en temps polynomial. Un *problème de décision* est un problème pouvant être répondu par oui ou non.

Cette classe inclut alors des problèmes faciles à résoudre, mais aussi des problèmes n'ayant pas d'algorithme en temps polynomial pour trouver des solutions [5]. Ces problèmes sont très communs, incluant dans le milieu industriel.

2.2.2 Problème de satisfiabilité

Il existe un problème très connu en informatique nommé le problème de satisfiabilité booléenne (SAT). Ce problème fut initialement énoncé par Stephen A. Cook lorsqu'il souhaitait trouver des failles dans des circuits électriques [6]. Une *variable booléenne* a un domaine pouvant prendre les valeurs vrai ou faux. Une *clause booléenne* est une disjonction de litéraux. Un *litéral* est une variable ou sa négation. Voici un exemple de clause : $A \vee \neg B \vee C$ où $A, B, C \in \{Vrai, Faux\}$. Dans un problème SAT, l'objectif est de trouver une affectation aux variables qui satisfait toutes les clauses. SAT est NP-Complet. Ce fut le premier problème prouvé NP-Complet. Il est possible d'utiliser les avancées faites sur le problème SAT pour améliorer la programmation par contraintes. Nous expliquerons comment dans la section 2.5.6.

2.2.3 Explosion combinatoire

Le phénomène d'*explosion combinatoire* survient lorsque l'ajout d'un petit nombre d'éléments au problème cause l'augmentation exponentielle de la complexité de calcul, et ce jusqu'au manque de ressource de calculs, soit la limite de Bremermann [1]. Les problèmes n'ayant pas d'algorithme en temps polynomial pour trouver des solutions sont sujets à l'explosion combinatoire. Les problèmes

sujets à ce phénomène sont nombreux et retrouvés dans divers domaines. Voici quelques exemples : le problème du commis voyageur qui sera introduit dans la section 2.4, la conception d'horaires pour les tâches de métiers à tisser de notre partenaire industriel ou même certains jeux de tables. Tous les problèmes NP-Complets sont sujets à l'explosion combinatoire.

2.2.4 Méthodes de résolutions

Il existe plusieurs méthodes pour résoudre les problèmes sujets à l'explosion combinatoire. Dans ce mémoire, nous en utiliserons deux : la programmation par contraintes et les solveurs de satisfiabilité (SAT). Chaque approche a des avantages et inconvénients. En ordonnancement, l'état de l'art réside dans la programmation par contraintes [21]. Nous aborderons la programmation par contraintes dans la section 2.5.

2.3 Ordonnancement de tâches dans le temps

Nous utilisons la définition suivante de l'ordonnancement : « l'affectation temporelle de tâches à des ressources pour satisfaire des objectifs en respectant des contraintes »[18]. Nous sommes particulièrement intéressés à l'ordonnancement de tâches dans le temps. Nous avons alors un ensemble de tâches $i \in \mathcal{I}$. Ces tâches ont une date de départ le plus tôt (earliest starting time) est_i et une date de fin au plus tard (latest completion time) lct_i. Une tâche doit commencer après est_i et finir avant lct_i. Les tâches ont aussi un temps d'exécution, p_i tel que $p_i \leq \text{lct}_i - \text{est}_i$. Les problèmes d'ordonnancement ont parfois une date de fin prévue d_i qui peut permettre de calculer des indicateurs de performances avec des pénalités.

Une tâche peut demander un certain nombre de ressources pour être exécutée. On appelle la hauteur h_i la consommation en ressources d'une tâche pour tout point dans le temps. Par exemple, une tâche pourrait nécessiter trois employés pendant toute sa durée.

Les problèmes d'ordonnancement sont généralement des problèmes NP-Complets [45]. Il faut alors utiliser des outils spécialisés comme ceux mentionnés dans la section 2.2.4. Pour les problèmes d'ordonnancement, Beck *et al*. ont effectué des tests empiriques pour comparer différents outils et la programmation par contrainte offre les meilleures performances [21].

2.4 Commis voyageur

Un problème NP-Complet célèbre est le commis voyageur (en anglais : *traveling salesperson problem*). Un commis voyageur doit vendre des produits à un certain nombre de villes. La ville de départ est aussi la ville de fin de trajet. Ici, nous pouvons représenter ces données en entrées sous forme d'une matrice décrivant la distance entre chaque paire de villes. La figure 2.1 contient un graphe présentant un problème de commis voyageur. Chaque noeud représente une ville et chaque arc une distance. Les arcs

en bleu représentent le chemin optimal pour visiter toutes les villes. L'objectif est de trouver le circuit le plus court passant à travers toutes les villes.

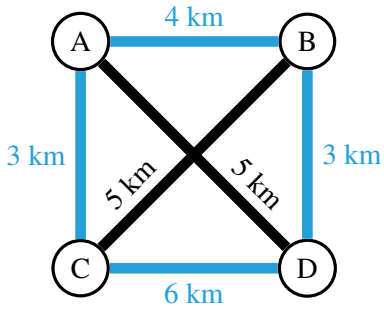


FIGURE 2.1 – Exemple de commis voyageur

Une approche rapide pour résoudre un problème de commis voyageur est d'utiliser le solveur Concorde qui est spécialement conçu pour le commis voyageur [7] . Ce solveur est particulièrement bon quand le nombre de villes est inférieur à 1000. Ce solveur arrive toutefois à trouver et prouver la solution optimale à des problèmes ayant 85 000 villes. Le solveur utilise des approximations pour débuter la recherche du meilleur circuit en étant déjà proche du circuit optimal. Ces approximations rendent le solveur très performant, mais au coût de la polyvalence. En effet, ce solveur peut résoudre qu'un seul problème NP-Complet : le commis voyageur. Il permet de représenter un problème du commis voyageur, mais pas de modéliser d'autres contraintes.

Il est parfois possible de faire un lien entre le commis voyageur et l'ordonnancement [3], c'est le cas pour notre problème. Remplaçons les villes par des tâches et les arcs par des temps de mise en course. Une nouvelle ville est rajoutée pour représenter quelle tâche sera la première. Cela nous permet de retirer la distance entre la première et dernière ville du problème d'optimisation. L'ordre dans lequel le commis voyageur visite les villes devient alors l'ordre des tâches dans le temps. Nous obtenons un problème d'ordonnancement de tâches non préemptives avec des temps de mises en course. Dans la figure 2.2, nous transformons le graphe de la figure 2.1 en problème d'ordonnancement.

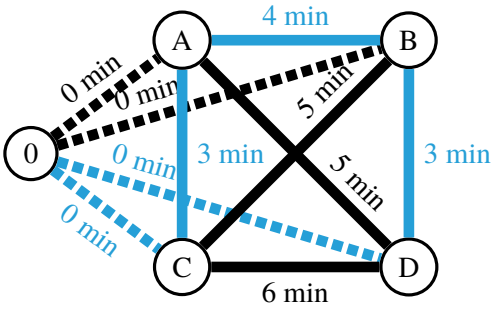


FIGURE 2.2 – Exemple de transformation du commis voyageur en problème d'ordonnancement

2.5 Programmation par contraintes

Comparé à d'autres langages de programmation comme le C++, les langages de programmation par contraintes (en anglais : *constraint programming*) utilisent la définition mathématique d'une variable. Une *variable* est un inconnu que nous souhaitons identifier et qui peut prendre une valeur de n'importe quel élément de son domaine. Un *domaine* est un ensemble de valeurs. Une variable X est un inconnu que nous souhaitons identifier et qui peut prendre une valeur de n'importe quel élément de son domaine dom(X). Nous notons $X \in dom(X)$.

En programmation par contraintes, nous souhaitons appliquer des contraintes sur les domaines des variables, ce qui réduit généralement le nombre de valeurs que peuvent prendre les variables. La *portée* d'une contrainte est l'ensemble des variables sous cette contrainte. Avec la programmation par contraintes, il est possible de modéliser plusieurs problèmes complexes et potentiellement les résoudre en temps raisonnable.

Il existe différents types de problèmes à résoudre. Parmi ceux-ci, il existe les problèmes de satisfaction. Dans un *problème de satisfaction*, il suffit de trouver une solution qui respecte toutes les contraintes. Ensuite, il existe les *problèmes de maximisation* et de *minimisation*. Dans ces problèmes, une fonction-objectif doit être minimisée ou maximisée. La *fonction-objectif* est encodée dans une contrainte qui met généralement en relation les autres variables. La fonction-objectif est une fonction qui contraint une variable à prendre une valeur (la *valeur de la fonction-objectif*) laquelle sera l'objet de l'optimisation. Par exemple, en ordonnancement, une fonction-objectif pourrait être la somme des retards. Pour chaque solution trouvée, une contrainte peut être rajoutée sur la valeur de la fonction-objectif afin de trouver uniquement des solutions meilleures. Dans le cas d'un problème de minimisation, plus la valeur de la fonction-objectif est petite, meilleure est la solution. Dans un problème de maximisation, plus la valeur que prend la fonction-objectif est grande, meilleure est la solution.

2.5.1 Arbre de recherche

Pour trouver des solutions, les techniques de programmation par contraintes représentent l'espace de recherche d'un problème sous forme d'arbre de recherche. L'*espace de recherche* représente toutes les solutions possibles à un problème. Un *arbre de recherche* est un graphe dirigé. Chaque noeud contient des données. Chaque noeud peut aussi pointer vers d'autres noeuds qui sont appelés *enfants*. Chaque noeud peut uniquement être pointé par un noeud. Ce noeud est appelé *parent*. Le noeud n'ayant pas de parent est appelé la *racine*. Lorsqu'un arbre est représenté graphiquement, la racine est généralement le noeud affiché le plus haut [8].

Chaque noeud est associé à une solution partielle. Une *solution partielle* est une solution où seules quelques variables ont des valeurs assignées. Chaque noeud enfant E doit avoir des domaines de variables étant des sous-ensembles des domaines de variables de son parent. L'action du choix de valeur à assigner dans le domaine d'une variable dans une solution partielle est appelée un *branchement* [35]. Une variable ayant une valeur d'assignée est dite fixée, tandis qu'une variable qui n'est pas assignée

est dite libre. Lorsqu'une solution ne satisfait pas les contraintes ou que le domaine d'une variable est vide, nous sommes en situation d'*échec*. En cas d'échec, il est nécessaire de remonter dans l'arbre afin d'explorer d'autres solutions.

Un algorithme de programmation par contraintes très simple peut représenter tous les choix de variables dans un arbre. Cet arbre aura une taille potentiellement exponentiellement grande en fonction de la cardinalité du domaine des variables et du nombre de variables. Il est possible de générer l'arbre de recherche au fur et à mesure que nous explorons les solutions.

Pour atténuer le problème de taille, il est possible de ne pas générer certaines parties de l'arbre. Sur certaines solutions partielles, il est possible d'inférer qu'aucune solution satisfaisante ne pourra être trouvée. Une solution satisfaisante est une solution valide. En retirant des parties de l'arbre, nous diminuons l'espace de recherche, soit le nombre de solutions partielles à explorer. Si le problème est formulé sous forme d'un problème de minimisation ou de maximisation, une solution satisfaisante est meilleure que celles déjà trouvées.

Il existe plusieurs techniques pour couper des branches d'un arbre de recherche. Nous expliquerons la vérification anticipée dans la section 2.5.2, le filtrage dans la section 2.5.3 et la propagation en 2.5.4.

À la figure 2.3, nous présentons un exemple d'arbre de recherche sans vérification anticipée, filtrage ou propagation. Chaque noeud présente une solution partielle. L'arbre est généré à partir d'un modèle mathématique simple ayant une contrainte et trois variables. La variable X peut prendre la valeur 4. La variable Y peut prendre la valeur 2 ou 3. La variable Z peut prendre la valeur 5 ou 6. La contrainte est $X + Y \leq Z$. Ici, nous avons un problème de satisfaction où toutes les solutions valides doivent être trouvées.

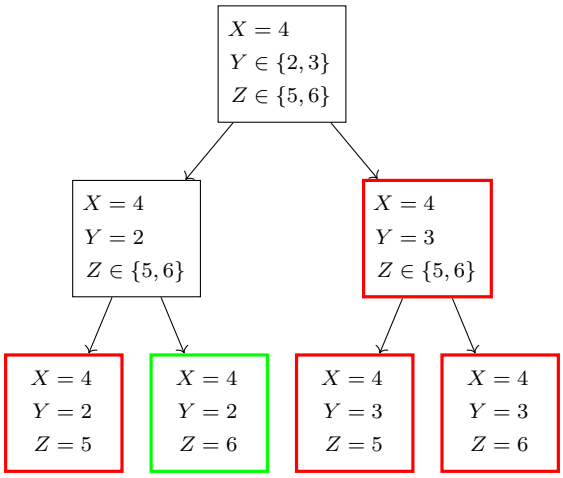


FIGURE 2.3 – Exemple d'arbre de recherche

$$X + Y \leq Z \tag{2.1}$$
$$X \in \{4\} \tag{2.2}$$
$$Y \in \{2, 3\} \tag{2.3}$$
$$Z \in \{5, 6\} \tag{2.4}$$

Nous commençons par créer le noeud racine en haut de l'image. Ce noeud contient les variables avec leurs domaines initiaux. Selon le solveur de contraintes et l'heuristique de branchement choisie, l'arbre sera amplement différent. Ici nous choisissons arbitrairement de brancher sur la plus petite valeur du domaine de Y, suivi de la plus petite valeur du domaine de Z. Pour explorer le second noeud, nous brancherons alors sur $Y = 2$. Dans l'image, le noeud se situe au centre, du côté gauche. Nous pouvons ensuite brancher sur $Z = 5$. La solution ne satisfait pas la contrainte (2.1), car $4+2 > 5$. Nous pouvons alors retirer la valeur 5 du domaine de Z dans le noeud parent. Par la suite, nous pouvons explorer $Z = 6$. Ici, la solution est valide, car $4+2 \leq 6$. Nous conservons alors la solution. Comme le problème demande de trouver toutes les solutions, il faut remonter dans l'arbre et retirer 6 de la solution partielle du noeud parent. Comme il n'y a plus de solution à explorer dans le noeud parent, nous remontons encore dans l'arbre. Nous revenons alors à la racine. Par la suite, nous explorons les solutions où $Y = 3$. Intuitivement, les solutions où $X = 4, Y = 3$ sont toutes plus grandes que la plus grande valeur de Z. L'algorithme explore toutefois tous les noeuds. Voici alors l'intérêt des algorithmes de vérification anticipée, de filtrage et de propagation. Ces différents algorithmes permettent de retirer plusieurs des solutions partielles inutiles affichées en rouge et converger plus rapidement vers les solutions valides en vert.

2.5.2 Vérification anticipée

La vérification anticipée (en anglais : forward checking) [13] nécessite un algorithme de vérification pour chaque contrainte. L'*algorithme de vérification* permet de valider si une affectation de variables à des valeurs satisfait la contrainte. La *vérification anticipée* peut retirer des valeurs du domaine d'une variable lorsque toutes les autres variables sont fixées. Nous utilisons l'algorithme de vérification sur chacune des valeurs du domaine de la variable non fixe afin de savoir quelles valeurs retirer [35]. Un appel à l'algorithme de vérification est donc fait pour chaque valeur dans le domaine de la variable non fixe. Par exemple, dans la branche gauche de la figure 2.3, il est possible de retirer la valeur 5 de la variable Z, tandis qu'il est possible de retirer toutes les valeurs de Z dans la branche de droite. Nous avons alors un arbre plus simple à la figure 2.4.

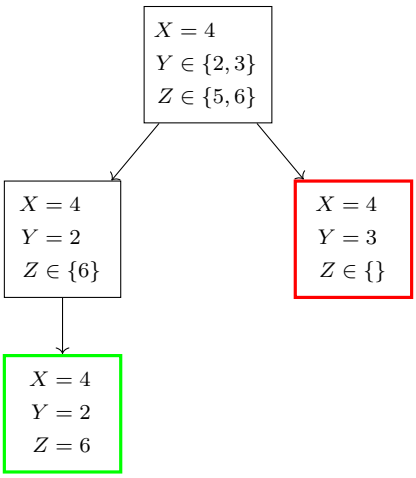


FIGURE 2.4 – Exemple vérification anticipée

2.5.3 Filtrage

Un *algorithme de filtrage* permet le retrait de valeurs du domaine des variables. Pour effectuer le filtrage, il faut un algorithme de filtrage spécifique à la contrainte à filtrer. Ces algorithmes sont généralement moins rapides à exécuter que les algorithmes de vérification. Toutefois, il permet de retirer des valeurs même quand il y a plus d'une variable n'étant pas fixée. Certains algorithmes de filtrage retirent plus de valeurs que d'autres. Il est souvent nécessaire de faire un compromis entre le temps d'exécution et le nombre de valeurs retirées du domaine.

Les algorithmes de filtrages sont séparés en catégories. Ceux offrant la cohérence de domaine, la cohérence d'intervalle, la cohérence de valeur [25] et ceux n'appartenant à aucune de ces catégories.

Cohérence de domaine

Pour assurer la *cohérence de domaine* (en anglais : *generalized arc consistency*), il faut trouver un support de domaine pour toutes les valeurs du domaine de toutes les variables dans la portée d'une contrainte. Un *support de domaine* est une assignation possible de valeurs aux variables telles que la contrainte est respectée.

Cohérence d'intervalle

Pour assurer la *cohérence de domaine* (en anglais : *range consistency*), il faut trouver un support d'intervalle pour toutes les valeurs du domaine de toutes les variables dans la portée d'une contrainte. Une variable a deux bornes, la *borne inférieure* qui est la plus petite valeur de son domaine et la variable et la *borne supérieure* qui est la plus grande. Un *support d'intervalle* est une assignation de valeur entre les bornes d'une variable pour chacune des variables telles que la contrainte est respectée.

Dans l'exemple de la figure 2.3, $X = 4, Y = 2, Z = 6$ serait un support de domaine et d'intervalle pour la contrainte (2.1), car chaque valeur fait partie du domaine de sa variable correspondante et est donc aussi entre les bornes du domaine de sa variable.

Comme second exemple, prenons les variables $X \in \{4, 5, 6\}$ $Y \in \{2, 6\}$ et la contrainte $2Y = X$. L'assignation $X = 4$ permet le support d'intervalle $X = 6, Y = 3$, car 3 est entre les bornes du domaine de Y. Nous avons alors un support d'intervalle qui n'est pas un support de domaine.

Cohérence de borne

La *cohérence de borne* (en anglais : *bounds consistency*) valide uniquement qu'il existe un support d'intervalle pour les bornes supérieure et inférieure du domaine des variables.

Voici un exemple pour illustrer le concept. Prenons les variables $X \in \{4, 5, 6\}$ $Y \in \{2, 6\}$ et la contrainte $2Y = X$. Pour assurer la cohérence de borne, des supports d'intervalles doivent être trouvés pour $X = 4, X = 6, Y = 2, Y = 6$. L'assignation $X = 4$ permet le support d'intervalle $X = 4, Y = 2$. L'assignation $X = 6$ permet le support d'intervalle $X = 6, Y = 3$. Le support d'intervalle $X = 4, Y = 2$ peut être réutilisé pour valider la borne $Y = 2$. Malheureusement, il n'existe pas de support d'intervalle pour $Y = 6$, cette valeur est alors retirée et le processus peut être recommencé.

Il est possible de réduire la taille de l'arbre de la figure 2.4 en appliquant un filtrage. En appliquant la cohérence de borne, il est possible de retirer totalement la partie droite de l'arbre. Nous trouvons alors directement sur la solution de ce problème simple, comme illustré dans la figure 2.5.

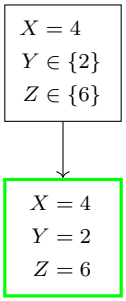


FIGURE 2.5 – Exemple filtrage avec cohérence de borne

Comparaison sommaire des cohérences

Les différents types de cohérences vont filtrer un nombre de solutions différent. Toute valeur filtrée par la cohérence de borne le sera nécessairement par la cohérence d'intervalle et toute valeur filtrée par la cohérence d'intervalle sera filtrée aussi par la cohérence de domaine. Toutefois, l'inverse n'est pas vrai. Un compromis doit être fait entre le filtrage et la rapidité d'exécution.

2.5.4 Propagation

Lorsque le niveau de cohérence désiré est atteint pour toutes les contraintes, on peut dire que nous avons atteint la *cohérence locale*. Atteindre la cohérence locale peut toutefois être difficile. Supposons que nous avons deux contraintes, appliquer la cohérence locale sur une contrainte et ensuite appliquer la cohérence sur l'autre contrainte peut invalider la cohérence appliquée sur la première contrainte si certaines variables dans la portée des contraintes sont les mêmes. Afin d'aborder ce problème, il existe la technique de propagation. Pour effectuer la propagation [35], il faut suivre les étapes suivantes. Lors d'un branchement ou d'un changement de domaine d'une variable dû à un filtrage, toutes les contraintes ayant cette variable dans leur portée doivent être filtrées. La propagation doit continuer jusqu'à cohérence locale pour chaque noeud exploré dans l'arbre de recherche.

$$X < Y \tag{2.5}$$

$$Y \mod 2 = 1 \tag{2.6}$$

$$X, Y \in \{1, 2, 3, 4\} \tag{2.7}$$

Nous montrons un exemple de propagation dans la table 2.1. Chaque ligne correspond à une étape dans la propagation. Soit les variables X et Y tels que $X, Y \in \{1, 2, 3, 4\}$. Prenons les contraintes (2.5) et (2.6) et appliquons les cohérences de domaines. Le choix de la première contrainte à propager est arbitraire au solveur de contraintes choisi. Nous choisissons $X < Y$. Uniquement deux valeurs peuvent être filtrées des domaines. Il est possible d'enlever 4 au domaine de X et 1 au domaine de Y, car il n'y a pas de support de domaine disponible pour les deux valeurs. Comme le domaine de Y a changé et Y est dans la portée de la contrainte (2.6), c'est donc la prochaine contrainte que nous propageons. Il faut alors enlever les valeurs paires du domaine de Y. Comme le domaine de Y a changé, il faut propager la contrainte (2.5), car la variable est dans sa portée. Nous pouvons alors retirer 3 du domaine de X. Il faut alors propager (2.6), toutefois le domaine ne change pas, nous avons alors atteint la cohérence locale.

Contrainte propagée	Domaines
$X < Y$	$X \in \{1, 2, 3\}, Y \in \{2, 3, 4\}$
$Y \mod 2 = 1$	$X \in \{1, 2, 3\}, Y \in \{3\}$
$X < Y$	$X \in \{1, 2\}, Y \in \{3\}$
$Y \mod 2 = 1$	$X \in \{1, 2\}, Y \in \{3\}$

TABLE 2.1 – Exemple de propagation

2.5.5 Heuristique de branchement

Une *heuristique de branchement* permet de diriger les choix de branchement lors de la génération d'un arbre de recherche. Pour un même problème, l'utilisation de deux heuristiques de branchement différentes créera possiblement deux arbres de recherche totalement différents. Comme une heuristique

de branchement permet de diriger l'exploration des solutions partielles, en encodant des éléments de connaissance du problème, il est possible de converger plus rapidement vers une meilleure solution. Une heuristique de branchement parfaite trouverait alors toujours la meilleure solution sans devoir remonter dans l'arbre de recherche. Une telle heuristique serait toutefois aussi difficile à calculer que son problème. Il est donc plus raisonnable de chercher une heuristique de branchement qui offre de bons résultats rapidement qu'une heuristique de branchement parfaite.

Notons que peu importe l'heuristique de branchement utilisée, l'exploration complète de l'arbre de recherche sera effectuée. L'heuristique de branchement permet principalement de converger plus rapidement vers une solution optimale. L'exploration complète de l'arbre de recherche peut être effectuée et ainsi garantir de trouver la solution au problème. Ceci permet alors de distinguer l'heuristique de branchement qui offre une garantie d'une heuristique qui n'en offre pas.

Il existe deux types principaux d'heuristique de branchement, soit celles choisissant sur quelle variable brancher et celles qui choisissent sur quelle valeur du domaine d'une variable il faut brancher. Nous présentons alors quelques heuristiques de branchement couramment utilisées en programmation par contrainte.

Choix de variable

Échouer rapidement. La variable choisie est celle avec la plus petite cardinalité de domaine. Cette approche peut donner de bons résultats dans les solveurs apprenant de leurs erreurs en réduisant énormément l'espace de recherche.

Recherche orientée sur l'impact. L'*impact* mesure la réduction des domaines causée par l'assignation d'une valeur à une variable. L'impact est appris au cours de la conception de l'arbre de recherche. Plus une variable cause de changement à la valeur de la fonction-objectif, plus l'impact devrait être grand. Selon le problème, il peut être pertinent de choisir la valeur la plus grande ou la plus petite d'une variable ayant un grand impact. [32]

Recherche orientée sur l'activité. La recherche orientée sur l'activité (en anglais : activity-based search) utilise le concept de clauses paresseuses qui sera présenté dans la section 2.5.6. Le principe derrière cette heuristique de branchement est d'assigner en premier des valeurs aux variables plus susceptibles de causer des échecs. L'*activité* mesure combien de fois le domaine d'une variable est affecté lors de la recherche. La mesure d'activité doit être mise à jour pour chaque noeud de l'arbre de recherche. Cette heuristique fait partie des meilleures en ce moment et dépasse généralement les performances de la recherche orientée sur l'impact. [29]

Choix de valeur

Plus petite valeur. La plus petite valeur du domaine d'une variable est utilisée en premier pour le branchement.

Plus grosse valeur. La plus grosse valeur du domaine d'une variable est utilisée en premier pour le branchement.

Médiane. La valeur médiane du domaine est utilisée lors du branchement. Comme le domaine peut changer, la médiane doit être recalculée à chaque fois.

2.5.6 Clauses paresseuses

L'heuristique de branchement peut se tromper et mener la recherche là où il n'y a pas de solution. Quand cette situation se produit, il est intéressant de comprendre quelle erreur a faite l'heuristique afin de ne pas la répéter. Il est donc intéressant de laisser le solveur apprendre de ses erreurs lors de l'exploration d'une instance afin d'éviter de refaire les mêmes erreurs. La *génération de clauses paresseuses* (en anglais : lazy clause generation) [42] vise ce but.

Au fur et à mesure que le solveur plonge dans l'arbre de recherche, les variables se font filtrer et les branchements réduisent davantage les domaines des contraintes. Il est alors possible de créer un *graphe de dépendance* qui permet d'expliquer comment les domaines ont été modifiés [30]. Pour chaque filtrage, on crée un noeud contenant un littéral. Le litéral représente une borne du domaine d'une variable dans une solution partielle. Prenons $X \in \{3, 4, 5, 6\}$, le litéral $[\![X \leq 6]\!]$ représente alors la borne supérieure.

L'algorithme de filtrage doit non seulement filtrer, mais aussi expliquer son filtrage. L'explication est encodée dans le graphe par des arrêtes. Nous obtenons alors un graphe dirigé montrant les filtrages effectués. L'explication prend la forme d'une implication où l'impliquant est une conjonction de litéraux et l'impliqué est un seul litéral. Chaque litéral est un noeud dans le graphe d'implication. Les noeuds de l'impliquant ont chacun une arrête sortante vers le noeud de l'impliqué.

Prenons $X \in \{1, 2\}, Y \in \{0, 1, 2\}, Z \in \{0, 1, 2\}$ sous la contrainte $X + Y \leq Z$. Nous avons alors les litéraux $[\![X \geq 1]\!], [\![Z \leq 2]\!]$. Avec ces litéraux, il est possible de réduire le domaine de Y en expliquant comme suit : $[\![X \geq 1]\!] \wedge [\![Z \leq 2]\!] \Rightarrow [\![Y \leq 1]\!]$. Dans la figure 2.6, nous représentons graphiquement cette explication.

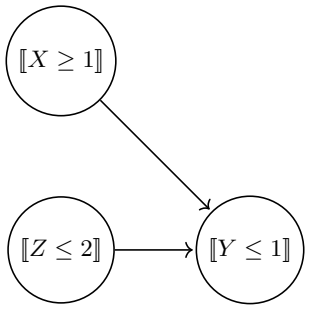


FIGURE 2.6 – Explication de $[\![X \geq 1]\!] \wedge [\![Z \leq 2]\!] \Rightarrow [\![Y \leq 1]\!]$

En plus des explications provenant du filtrage, d'autres contraintes implicites sont rajoutées par le

solveur afin de permettre la création de litéraux plus générique. Voici quelques contraintes implicites pouvant être utilisées.

$$[\![X \leq v]\!] \wedge [\![X \geq v]\!] \implies [\![X = v]\!] \tag{2.8}$$
$$[\![X = v]\!] \implies [\![X \leq v]\!] \tag{2.9}$$
$$[\![X = v]\!] \implies [\![X \geq v]\!] \tag{2.10}$$
$$[\![X \leq v]\!] \implies [\![X \leq v + 1]\!] \tag{2.11}$$
$$[\![X \leq v]\!] \wedge \neg[\![X = v]\!] \implies [\![X \leq v - 1]\!] \tag{2.12}$$
$$\neg[\![X \leq v - 1]\!] \wedge \neg[\![X = v]\!] \implies \neg[\![X \leq v]\!] \tag{2.13}$$

Lors d'un échec, nous devons apprendre de nos erreurs. C'est alors à ce moment que la clause paresseuse est générée. Une *clause paresseuse* est une disjonction de variables booléennes ou de leur négation comme expliquée dans la section 2.2.2. Le qualificatif « paresseuses » provient de la façon dont une clause est générée. Ces clauses sont apprises au besoin lors de l'exécution du solveur de contraintes. Les clauses paresseuses peuvent être vues comme des contraintes redondantes ajoutées en cours d'exécution. En d'autres mots ces contraintes ne changent pas l'ensemble de solutions valides. Les clauses paresseuses permettent de rapidement filtrer des valeurs du domaine des variables. Dans ces clauses, chaque litéral est un noeud du graphe d'implication ayant mené à l'échec. Lorsqu'un échec survient, une clause doit être générée à partir des explications menant à l'échec.

Lors d'un échec, nous souhaitons comprendre l'explication minimale ayant mené à cet échec. Ce ne sont pas nécessairement tous les branchements qui ont été de mauvais choix. Le choix des litéraux à inclure dans la clause est propre au solveur de contraintes. Les litéraux de la clause sont calculés à l'aide d'une *coupe*, car dans un graphe, le choix peut être visuellement représenté comme un partitionnement des noeuds représentant des litéraux. Dans cette représentation visuelle, la coupe passe à travers des arcs. Chaque litéral associé à l'origine d'un arc coupé fait partie de la clause. Par convention, les noeuds de branchements sont à gauche de la coupe. Plus les litéraux sont proches de l'échec, plus ils sont précis par rapport à l'échec produit. Il y a un compromis à faire entre choisir des litéraux plus généraux et des litéraux plus précis. Plusieurs techniques existent pour choisir la coupe, notamment la coupe 1UIP [10]. Cette technique de coupe choisit les valeurs booléennes au premier point d'implication unique, soit le premier noeud par lequel passent tous les chemins et qui est situé sur le chemin vers l'échec. La coupe est donc faite directement après ce noeud.

Dans la figure 2.7, nous présentons un exemple de coupe, ainsi que la clause générée. Il y a deux variables entières, soient V et M. Ces deux variables sont soumises à la contrainte $M \geq V + 4$. Dans le graphique, nous supposons que le solveur de contrainte a fait un branchement sur $M = 5$ et que le domaine de V était défini dans une clause comme $V \geq 2$ grâce à des branchements précédents. Afin de généraliser la valeur booléenne du branchement, le solveur transforme la clause $[\![M = 5]\!]$ en $[\![M \leq 5]\!]$. Ceci aura comme impact de créer une clause coupant plus de solutions invalides. En appliquant la contrainte au booléen $[\![V \geq 2]\!]$, nous obtenons $[\![M \geq 6]\!]$. Comme M est un entier et

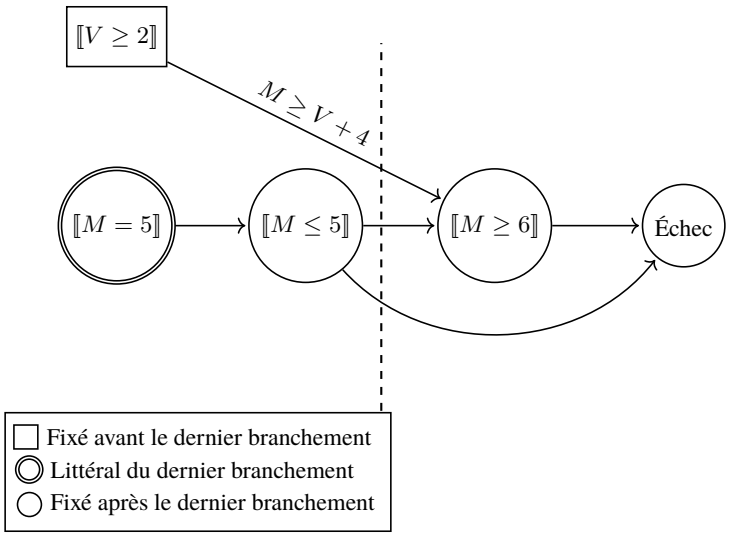


FIGURE 2.7 – Exemple de coupe

qu'aucune valeur ne peut être plus petite que 6 et plus grande que 5 en même temps, alors nous avons un échec. Dans la figure, nous avons choisi arbitrairement de couper au centre. Ceci nous laisse alors les valeurs booléennes suivantes pour former notre clause : $[\![V \geq 2]\!]$ et $[\![M \leq 5]\!]$. Nous pouvons alors former la clause suivante : $\neg[\![M \leq 5]\!] \vee \neg[\![V \geq 2]\!]$. Cette clause pourra ensuite être utilisée par le solveur pour ne pas explorer les noeuds susceptibles de causer cet échec.

L'implémentation des clauses paresseuses dans un solveur de programmation par contraintes permet alors de chercher les forces des solveurs SAT et les forces des solveurs de programmation par contraintes. Le solveur Chuffed conçu par Geofrey Chu [4] est considéré comme l'état de l'art dans ce domaine.

2.5.7 Redémarrage

Les solutions dans la même portion de l'arbre de recherche se ressemblent puisqu'elles sont construites à partir des mêmes solutions partielles. Le *redémarrage* permet de cesser l'exploration d'une portion de l'arbre de recherche afin d'aller explorer une autre portion [15]. Une méthode de redémarrage réinitialise les domaines des variables et recommence l'exploration de l'espace de recherche. Il est alors possible que certaines solutions partielles soient revisitées.

Il est alors généralement pertinent d'utiliser une heuristique de branchement qui n'explore pas les mêmes solutions. Une heuristique de branchement utilisant des valeurs aléatoires peut donc être bénéfique [14]. Toutefois, dans des solveurs de contraintes avec clauses paresseuses, il est non seulement possible de conserver les clauses paresseuses afin de réduire plus rapidement la taille de l'arbre de recherche en ne reproduisant pas certains échecs, mais aussi d'éviter de revisiter une portion de l'arbre de recherche ayant déjà été explorée avant le redémarrage [23].

Le moment de redémarrage peut être fonction du nombre de noeuds explorés, du temps écoulé, du nombre de solutions trouvées, etc. Un exemple notable de redémarrage est Luby [26] qui a été prouvé optimal pour certains problèmes.

2.5.8 Contraintes globales

Une *contrainte globale* (en anglais : global constraint) est une contrainte dont l'arité (le nombre de variables dans la portée) est un paramètre. L'avantage des contraintes globales est de prendre en compte plusieurs sous-contraintes [2]. Ceci permet alors d'utiliser des algorithmes spécifiques et performants pour le filtrage de solutions. En permettant aux contraintes globales d'être spécialisées en des problèmes précis, il est possible d'utiliser les particularités des problèmes et l'interaction entre les variables pour optimiser le filtrage des valeurs du domaine des variables. L'utilisation de contraintes globale cause généralement une amélioration des performances lors de la résolution de problèmes.

Dans un solveur de contraintes avec génération de clauses paresseuses, il est possible de créer des clauses spécifiques à ces contraintes globales. Ces clauses seront alors plus performantes.

Nous présentons alors quelques contraintes globales issues de la littérature.

Contrainte AllDifferent

Le concept des *contraintes globales* a pris son envole suite à la conception d'algorithme pour la contrainte AllDifferent [37]. Cette contrainte empêche les variables passées en argument de prendre les mêmes valeurs. Toutes les variables dans la portée de la contrainte AllDifferent doivent prendre une valeur différente.

Formellement, la contrainte $\text{AllDifferent}([X_1, \ldots, X_n])$ où X_i est une variable qui doit satisfaire :

$$X_i \neq X_j \qquad \forall i, j \in [1, n] \text{ tels que } i \neq j \tag{2.14}$$

Contrainte Cumulative

La contrainte $\text{Cumulative}([S_1, \ldots, S_n], [p_1, \ldots, p_n], [h_1, \ldots, h_n], c)$ contraint des tâches $i \in \mathcal{I}$ à respecter une quantité de ressources maximale c . Les tâches ont une durée p_i et un temps de départ S_i. Les tâches ont une quantité de ressource nécessaire h_i. La quantité de ressource est validée pour tout point dans le temps t. Notons aussi l'horizon H.

Formellement, une solution respectant la contrainte cumulative [31, 38] satisfait l'équation suivante :

$$\sum_{i \in \mathcal{I} : S_i \leq t < S_i + p_i} h_i \leq c \qquad \forall t \in [0, H-1] \tag{2.15}$$

Contrainte Circuit

La contrainte $\text{Circuit}([X_1, \ldots, X_n])$ force les variables à former un circuit [20]. Comme le tableau doit former un circuit, toutes les valeurs doivent être différentes. Il est possible de représenter graphi-

quement un tableau passé en paramètre où les variables sont assignées sous la forme d'un graphe. La position i de la variable dans le tableau représente le noeud. Le noeud i a un arc sortant vers X_i.

Dans la figure 2.8, nous représentons un tel graphe pour le tableau $[3, 5, 4, 2, 6, 1]$. Comme nous pouvons voir, la première entrée du tableau contient la valeur 3. Il y a donc un lien du noeud 1 au noeud 3.

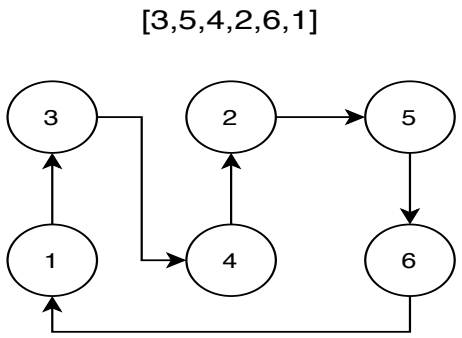


FIGURE 2.8 – Exemple de circuit

Contrainte TABLE

La contrainte TABLE$([X_1, \ldots, X_n], T)$ permet de forcer les variables X_i à prendre des valeurs extraites d'une rangée du tableau T [24, 28]. Le tableau T doit alors avoir une rangée de longueur n.

Cette contrainte est simple, mais elle peut être très efficace pour réduire les domaines de plusieurs variables à la fois.

2.6 Recherche locale à grand voisinage

Certaines techniques pour rechercher des solutions sont inspirées du redémarrage. C'est le cas pour la recherche locale à grand voisinage (en anglais : large neighborhood search) [40]. Cette technique a comme principe que réduire la taille de l'instance permet de trouver plus rapidement une solution optimale. Séparer un problème en plus petites parties peut être bénéfique. Pour effectuer la recherche locale à grand voisinage, il faut choisir une à plusieurs variable devant faire l'objet de la recherche locale à grand voisinage.

Dans la figure 2.9, nous présentons un organigramme de programmation représentant l'enchaînement des opérations lors d'une recherche locale à grand voisinage. Pour une itération, certaines assignations de variable sont libérées. C'est le processus de destruction. Le problème est alors réduit à trouver les valeurs optimales pour une portion du problème. La résolution de ce sous-problème est l'étape de réparation. Un solveur de contraintes peut être utilisé pour cette étape. Une fois la valeur optimale trouvée pour le sous-problème, l'assignation des valeurs aux variables est conservée. Il peut alors

y avoir une condition d'arrêt. Cette condition peut être un temps maximal d'exécution, un nombre d'itérations, etc. Ainsi s'achève une itération de la recherche locale à grand voisinage

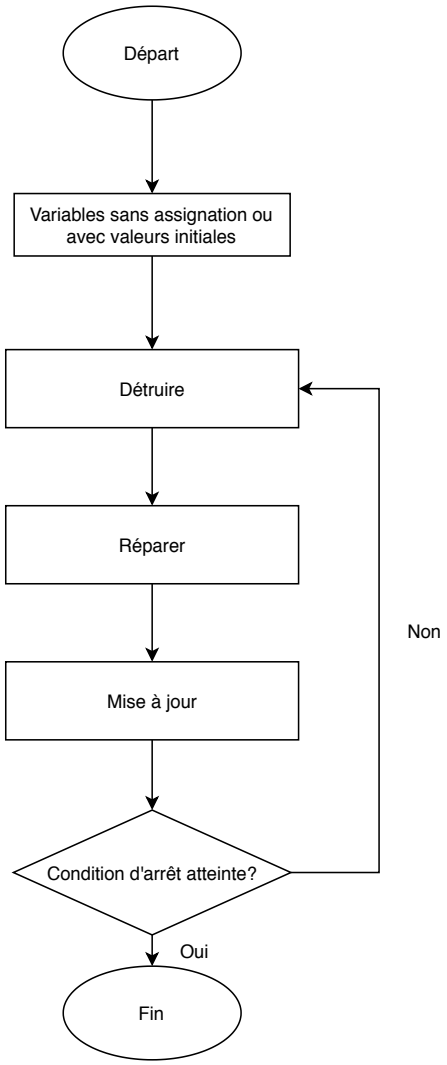


FIGURE 2.9 – Organigramme de recherche locale à grand voisinage

Il existe plusieurs façons de déterminer quelles variables seront oubliées lors de la destruction. Le choix aléatoire est simple et souvent efficace [12]. Il faut uniquement choisir aléatoirement les valeurs à oublier. Une autre méthode est de retirer des groupes de variables ayant un lien entre elles. Le choix des groupes peut encore être aléatoire [27].

2.7 Optimisation stochastique

L'*optimisation stochastique* est une branche de l'optimisation qui se concentre sur les problèmes avec aléas. Ces problèmes contiennent une ou plusieurs variables stochastiques, soit des variables aléatoires suivant des lois probabilistes. Une variable stochastique tirée d'une loi de Poissonserait alors notée ainsi $Y \sim \text{Poisson}(\gamma)$ où γ est le paramètre de la loi de Poissonet Y est une valeur aléatoire pigée à partir de la distribution Poisson(γ).

En optimisation stochastique, un des objectifs est souvent de trouver des solutions robustes. Une *solution robuste* est une solution qui réagit bien aux aléas et évènements imprévus. Ignorer ces évènements peut causer des pertes financières importantes pour une compagnie. L'optimisation stochastique permet de s'approcher de la réalité en modélisant les aléas et donc de réduire l'impacte des évènements imprévus sur une compagnie.

Nous présentons, dans les prochaines sous-sections, certaines méthodes utilisées en optimisation stochastique tirées de la littérature.

2.7.1 Optimisation par scénarios

Chaque scénario représente une suite d'évènements probables. Un *scénario* encode alors une valeur possible pour toutes les variables stochastiques. Il est possible de représenter tous les scénarios ou seulement une partie. Une optimisation est faite où les variables stochastiques prennent les valeurs d'un scénario. Cette optimisation doit être faite pour tous les scénarios sélectionnés par l'utilisateur.

La fonction-objectif peut être une somme des fonctions objectif pour chaque scénario avec une pondération [43]. Une autre méthode est de résoudre optimalement tous les scénarios et de pénaliser les différences entre les valeurs assignées aux variables [34]. Il existe d'autres méthodes dont certaines seront décrites dans les prochaines sous-sections.

L'optimisation par scénario (en anglais : scenario-based optimization) est facile d'accès et déjà offerte dans plusieurs logiciels connus [33]. Toutefois, dans certains problèmes, le nombre de scénarios possibles est infini ou presque. Il n'est alors pas judicieux de générer tous les scénarios. Utiliser seulement une partie des scénarios peut aussi créer des biais dans la robustesse d'une solution, une suroptimisation à des scénarios spécifiques.

2.7.2 Supermodèles

Un (a,b)-super modèle (en anglais : (a,b)-super model) est un modèle générant des (a,b)-super solutions. Ces super solutions sont des solutions où pour a évènements stochastiques, il est possible de trouver une solution en faisant b modifications ou moins [11]. La technique permet alors de créer des solutions qui s'adaptent aux changements causés par des évènements imprévus. Cette technique permet de limiter le nombre de changements à faire pour conserver une solution valide.

En effet, le nombre d'évènements pouvant survenir est exponentiel en fonction de a tel que $O(n^a)$. Le temps de calcul des modifications possibles est exponentiel en fonction de b tel que $O(n^b)$. Comme les problèmes à résoudre sont dans la classe NP ou encore plus complexes, trouver de super solutions reste dans la même classe de complexité [11]. Pratiquement, le temps de résolution augmente rapidement. Cette technique fonctionne particulièrement bien pour les problèmes ayant une petite valeur de b [16, 17].

2.7.3 Optimisation à deux étapes

Le concept d'*optimisation à deux étapes* (en anglais : two-stage optimization) [9] est fondé sur le principe que des décisions devraient être prises uniquement sur des données connues.

Cette méthode consiste à séparer un problème de nature stochastique en deux. La première étape optimise la fonction-objectif déterministe de la première étape, ainsi que l'espérance des fonctions objectifs de la seconde étape. Dans la seconde étape, une optimisation est faite où les variables stochastiques ont une valeur d'assignée. Les valeurs sont sélectionnées par scénarios. La seconde étape est exécutée pour chaque scénario jusqu'à ce que la valeur optimale soit trouvée.

Le calcul de la fonction-objectif aura une forme comme la suivante :

$$q + \mathop{\mathbb{E}}_{x \in \xi} Q(x) \tag{2.16}$$

Où q est la valeur de la fonction-objectif déterministe et $Q(x)$ est la fonction-objectif de la partie stochastique pour le scénario x provenant de l'ensemble des scénarios ξ.

2.7.4 Contraintes de chance

Les *contraintes de chance* (en anglais : chance constraint) permettent de contraindre des probabilités. Une contrainte de chance est une contrainte qui doit être satisfaite avec un certain niveau de probabilité. Cette contrainte peut être n'importe quelle contrainte ayant un élément de probabilité intégré. Les contraintes de chance peuvent entre autres contenir un seuil devant être respecté par les variables stochastiques. Le seuil peut être une probabilité [49] minimum, une probabilité maximum, un niveau [36], etc. Ces contraintes sont souvent accompagnées de distributions desquelles seront extraites les diverses probabilités. Ce concept provient initialement de la communauté de programmation linéaire. Malgré que ce concept ne soit pas linéaire de façon inhérente, il y a peu d'algorithmes utilisant les contraintes de chance en programmation par contraintes. Nous utilisons alors cette notion comme la base pour une nouvelle contrainte de chance en programmation par contraintes dans l'article du chapitre 4.

2.7.5 Simulation

La simulation [22] permet de valider des solutions dans un contexte stochastique. Des probabilités d'évènements sont données à un simulateur. Des aléas sont générés à partir des différentes probabilités.

Certains simulateurs permettent de modéliser une maquette exacte d'un lieu de travail dans un environnement 3D [48]. Ceci peut alors permettre de prendre en compte certains facteurs comme le temps de déplacement, la distance à parcourir, la position des employés ou de l'équipement, etc.

Il est alors possible d'offrir une représentation précise de ce qui peut survenir en situation réelle lors de l'application d'une solution en entreprise.

Chapitre 3

Leveraging Constraint Scheduling : A Case Study to the Textile Industry

Material from : 'Alexandre Mercier-Aubin, Gaudreault Jonathan and Claude-Guy Quimper, Leveraging Constraint Scheduling : A Case Study to the Textile Industry, In proceedings of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, published in 2020, 2020 Springer Nature Switzerland AG'. [1]

3.1 Résumé

Même avec les récents progrès en ordonnancement, les problèmes industriels avec plusieurs centaines de tâches restent difficiles à résoudre sous certaines formes. Nous présentons des techniques qui permettent d'améliorer les performances de la programmation par contraintes pour un problème industriel de 600 tâches sans préemption, 90 ressources machines et des temps mise en course influencés par la séquence. Notre méthode utilise le problème du commis voyageur comme une simplification d'un problème d'ordonnancement et utilise la solution comme départ pour sa recherche. Nous explorons aussi une technique de recherche locale. Les expérimentations conduites sur un jeu de données fourni par notre partenaire industriel provenant de l'industrie du textile montrent que nous n'obtenons pas des solutions optimales, mais satisfaisantes.

3.2 Abstract

Despite the significant progress made in scheduling in the past years, industrial problems with several hundred tasks remain intractable for some variants of the scheduling problems. We present techniques that can be used to leverage the power of constraint programming to solve an industrial problem with 600 non-preemptive tasks, 90 resources, and sequence-dependent setup times. Our method involves solving the traveling salesperson problem (TSP) as a simplification of the scheduling problem and using the simplified solution to guide the branching heuristics. We also explore large neighborhood

search. Experiments conducted on a dataset provided by our partner from the textile industry show that we obtain non-optimal but satisfactory solutions.

3.3 Introduction

Nowadays, most textiles are mass-produced using automated looms. In the context of the fourth Industrial Revolution [8, 9], the textile industry seeks to expand the automation to planning and scheduling tasks. While recent progress made in constraint programming allows tackling many NP-Hard scheduling problems, the size of the scheduling instances that can be solved for some variants of the problem remain small compared to what the industry needs. It is common to observe industrial instances with 800 tasks and limited resources. In this context, constraint programming can still be used to obtain good, but not optimal, schedules. However, extra work on branching heuristics and the use of local search is often required to obtain satisfactory results.

We present a study case of a scheduling problem encountered by our industrial partner, a textile company. More than 800 tasks need to be scheduled over 90 automated looms. A team of technicians needs to set up the looms before starting new tasks. The duration of each setup depends on the tasks that precede and succeed the setup and no more setups than technicians should be simultaneously scheduled.

We explain how we succeed in obtaining good, but possibly non-optimal, solutions for large instances. We achieve this goal by solving a simplification of the problem and using the simplification solution to find better solutions to the non-simplified problem. We also use a large neighborhood search to improve the solution. Note that similar problems were studied in the literature [5], but the approach to solve the problem relies on different optimization techniques such as Mixed Integer Programs.

The paper is divided as follows. Section 3.4 presents the preliminary concepts about constraint scheduling, the connection between scheduling with setup times and the traveling salesman problem (TSP), and the large neighborhood search. Section 3.5 formally introduces the industrial scheduling problem. Section 3.6 presents the mathematical models. Section 3.7 shows how to solve the models. Section 3.8 presents the experimental results. Finally, Section 3.9 concludes this work.

3.4 Background

3.4.1 Constraint Scheduling

We present the main components of common scheduling problems. The actual industrial scheduling problem we will solve is formally defined in Section 3.5.

A *scheduling problem* is composed of a set of tasks $\mathcal{I}$ (or activities) that need to be positioned on a time line. A task $i \in \mathcal{I}$ has for parameters its *earliest starting time* est_i, its *latest completion time* lct_i, a *processing time* p_i, a *due date* d_i, and a resource consumption rate h_i also called *height*. We consider non-preemptive tasks, i.e. that when a task starts, it executes for exactly p_i units of time

without interruption. A task must start no earlier than its earliest starting time and complete no later than its latest completion time. A task that completes after its due date incurs a penalty that depends on the objective function.

A *cumulative resource* r has capacity C_r. Multiple tasks can execute on a cumulative resource r as long as the sum of their heights is no greater than C_r. By fixing $C_r = h_i$ for all tasks i, we obtain a *disjunctive resource* that can only execute one task at a time. On such a resource, it could happen that a setup needs to be performed between the execution of two tasks. The *setup time* $t_{i,j}$ is a minimum lapse of time that must occur between the completion of task i and the starting time of task j, should task i executes before task j. The setup times satisfy the triangle inequality $t_{i,j} + t_{j,k} \geq t_{i,k}$.

Constraint programming can be used to solve scheduling problems. One can define a *starting time* variable S_i with domain $\text{dom}(S_i) = [\text{est}_i, \text{lct}_i - p_i]$.

The constraints $\textsc{Cumulative}([S_1, \ldots, S_n], [p_1, \ldots, p_n], [h_1, \ldots, h_n], C_r)$ or $\textsc{Disjunctive}([S_1, \ldots, S_n], [p_1, \ldots, p_n])$ ensures that the cumulative or disjunctive resource is not overloaded. Extra constraints can easily be added to the model such as a *precedence constraint* $S_i + p_i + t_{i,j} \leq S_j$ that forces a task i to complete before a task j can start.

Combining the concepts presented above results in a large variety of scheduling problems. For instance, the Resource-Constrained Project Scheduling Problem (RCPSP) contains cumulative resources and non-preemptive tasks subject to precedence constraints. Constraint solvers can find optimal solutions to instances of the RCPSP with 120 tasks [16]. However, with setup times, constraint solvers can only solve instances up to 120 tasks but never to optimality [2]. In order to improve the performances, search strategies and branching heuristics can be provided to the solver. It is also possible to use the constraint solver within a large neighborhood as explained in Section 3.4.2.

3.4.2 Local Search

A *local search* is a heuristic method that starts from a suboptimal and possibly unfeasible solution and tries to improve its feasibility and its objective value. At each iteration, an *operator* is applied, and the solution is modified. If the modified solution becomes more feasible or more optimal, it becomes the current solution. The operator modifies the values of a subset of the variables in the problem. When a large number of variables are modified, we say that the heuristic method is a *large neighborhood search*.

A constraint solver can be used as a large neighborhood search operator. One takes the current solution and forces a subset of variables to take the same values as the current solution. The remaining variables are let free. The result is a constraint satisfaction problem (CSP) with fewer variables to assign that is generally easier to solve. The solution of this CSP becomes the new current solution that can be further improved by selecting a different subset of variables.

There exists various strategies to select which variables to reassign in a scheduling problem. One could

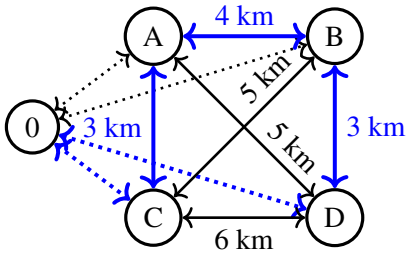


FIGURE 3.1 – Example of a scheduling problem with setup times reduced to a TSP problem. The node 0 is the dummy node. Dotted lines have null costs. The blue lines represent the optimal schedule : C, A, B, D.

randomly select a fixed percentage of the tasks to reschedule [10]. It is also possible to select a time window from which all tasks whose execution is contained in this window are rescheduled [15]. Another option is to fix some precedences observed in the current solution between a subset of pairs of tasks and let the remaining pairs of tasks free from any precedence [14]. When minimizing the makespan in scheduling problems of 300 tasks, these techniques can significantly improve the objective function [6].

3.4.3 The Traveling Salesperson Problem

The traveling salesperson problem (TSP) is a classic optimization problem. In its directed variant, we have n cities and a distance matrix D such that $D_{i,j}$ is the distance to travel from i to j. The matrix satisfies the triangle inequality $D_{i,j}+D_{j,k} \geq D_{i,k}$. The salesperson plans on finding the shortest circuit visiting each city exactly once.

There is a strong connection between the TSP and the scheduling problem with setup times. For a scheduling problem with n tasks $\mathcal{I}$ and setup times $t_{i,j}$, one creates an instance of the TSP with $n+1$ cities and a distance matrix $D_{i,j} = t_{i,j}$ for $i, j \in \mathcal{I}$ and $D_{i,j} = 0$ otherwise. The extra city marks the beginning (and the end) of the schedule. The salesperson visits the cities (tasks) in order to minimize the sum of the distances (setup times). Figure 3.1 illustrates the reduction.

The solver Concorde [4] represents the state-of-the-art for solving the TSP. It can find and prove optimal solutions to instances with 85,000 cities. As it was designed solely for this type of problem, it cannot handle additional constraints. For example, if the tasks have earliest starting times and latest completion times, the scheduling problem reduces to a TSP with time windows [12]. The solver Concorde cannot solve such problems.

Solving the TSP can help solving scheduling problems. For instance, Tran and Beck [18] use the TSP as the slave problem in a Benders decomposition to solve a problem with resource allocation and setup times.

3.5 Problem Description

We describe the industrial problem specified by our industrial partner. The parameters introduced in this section are summarized in Table 3.1. A task consists of weaving a textile on a loom. We therefore have a set of tasks $\mathcal{I}$ and a set of looms L. Each task $i \in \mathcal{I}$ is pre-assigned to a loom l_i and has for processing time p_i. A loom $l \in L$ becomes available at time a_l. Prior to this time, the loom is busy terminating a task not in $\mathcal{I}$ that can neither be interrupted nor rescheduled. Each task $i \in \mathcal{I}$ has a style z_i, a due date d_i, and a priority r_i. We wish to minimize the total tardiness weighted by priority, i.e. $\sum_{i \in \mathcal{I}} r_i \cdot \max(0, S_i + p_i - d_i)$ where S_i is the starting time of the task. The scheduling horizon spans from time 0 to time H. In practice, we have a horizon of 240 hours with a time step of 15 minutes resulting in $H = 240 \times \frac{60}{15} = 960$.

Major setups : Each loom $l \in L$ has an initial configuration c_l^{init} and a final configuration c_l^{final}. If $c_l^{\text{init}} \neq c_l^{\text{final}}$ then there is a *major setup* of duration p_l^{major} to change the configuration of the loom l. Only one major setup is possible during the scheduling horizon. A *specialized worker* is selected from a pool W to achieve this major setup. A task $i \in \mathcal{I}$ needs to be executed on a loom l_i when it has configuration $c_i \in \{c_{l_i}^{\text{init}}, c_{l_i}^{\text{final}}\}$. If $c_i = c_{l_i}^{\text{init}}$, the task needs to be executed prior to the major setup and after the major setup if $c_i = c_{l_i}^{\text{final}}$.

Minor setups : A *minor setup* needs to be performed between two consecutive tasks $i, j \in \mathcal{I}$ on a loom. While a major setup entails a configuration change, a minor setup only gets the loom ready for its next job. This setup is decomposed into several steps, each executed by a person of a different profession in P which is disjoint from the set of workers W. The professions are sorted in order of execution and labeled with integers from 1 to $|P|$, i.e. that the first step of a minor setup is executed by profession 1, the second step is executed by profession 2, and so on. The order of execution is the same for all minor setups. The person of the profession $p \in P$ needs $t_{i,j,p}$ time to execute his/her part of the minor setup between task $i \in \mathcal{I}$ and $j \in \mathcal{I}$. There are q_p people of the profession p. Consequently, no more than q_p minor setup steps can be simultaneously executed by the people from the profession p.

Figure 3.2 shows a schedule on two looms with a worker of each category. We see conflicts delaying minor setups on the second loom.

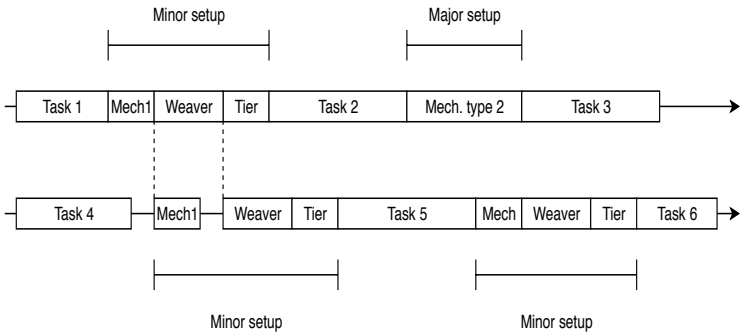


FIGURE 3.2 – Example of scheduling on two looms with one of each resource.

$\mathcal{I}$	: Set of tasks.
L	: Set of looms.
P	: Set of professions for minor setups.
W	: Set of specialized workers for major setups.
r_i	: Priority of task i.
d_i	: Due date of task i.
z_i	: Style of task i.
l_i	: Loom assigned to task i.
c_l^{init}	: Initial configuration of loom l.
c_l^{final}	: Final configuration of loom l.
c_i	: Required configuration for task i.
p_i	: Processing time of task i.
p_l^{major}	: Major setup time of loom l.
$t_{i,j,p}$	: Minor setup time between tasks i and j for the profession p.
a_l	: Earliest available time of loom l.
q_p	: Number of workers of the profession p available for the minor setups.

TABLE 3.1 – Parameters of the problem

3.6 Models

We present an optimization model that is later submitted to a constraint solver. The variables and domains are summarized in Table 3.2. Constraints are stated from (3.1) to (3.11).

There are three types of events to schedule : tasks, major setups, and minor setups between a task and its successor. Let $S_i \in [a_l, H)$, $S_l^{\text{major}} \in [a_{l_i}, H)$, and $S_{i,p}^{\text{minor}} \in [a_{l_i}, H)$ be their starting time variables for tasks $i \in \mathcal{I}$, loom $l \in L$, and profession $p \in P$. Their domains prevent the events from starting before their respective loom l becomes available.

The variable F_l encodes the first task to execute on loom l. The variable N_i encodes the task that succeeds task i on the loom. If i is the last task, its value is set to a sentinel. There is one sentinel per loom : $\sigma = \{\sigma_1, \ldots, \sigma_{|L|}\}$. The variable N_i is also defined when $i \in \sigma$ is a sentinel. The next task of

Variable	Domain	Description
S_l^{major}	$[a_l, H)$	Start of the major setup on loom i
S_i	$[a_{l_i}, H)$	Start of task i
$S_{i,p}^{\text{minor}}$	$[a_{l_i}, H)$	Start of the minor setup for profession p between task i and its successor
N_i	$\{j \in \mathcal{I} \mid l_j = l_i\} \cup \{\sigma_{l_i}\}$	Next task after task i
F_l	$\{i \in \mathcal{I} \mid l_i = l\}$	First task on loom l
T	$[0, \infty)$	Total tardiness weighted by priority

TABLE 3.2 – Variables and their domains

a sentinel is the first task on the next loom. Consequently, the vector N is a permutation of $\mathcal{I} \cup \sigma$ with a single cycle.

The model contains the constraints (3.1) to (3.11). The objective function (3.1) minimizes the tardiness of the tasks, weighted by priority. Constraints (3.2) and (3.3) ensure that a task requiring its loom's initial configuration executes before the major setup or else waits after the major setup to execute. Constraint (3.4) ensures that the first step of the minor setup following task i starts once task i is completed. Constraint (3.5) is a precedence constraint over the different steps of a minor setup. Constraint (3.6) makes the task N_i start immediately after the minor setup is completed. Indeed, once the loom is ready, there is no need to postpone the task. Constraints (3.7) and (3.8) ensure that the last task of a sentinel on a loom is the first task on the next loom. The loom that succeeds the last loom is the first loom. That creates a circuit visiting each task exactly once. This idea of a circuit is inspired from Focacci et al. [7] and led to the addition of constraint (3.9) to the model. This constraint [11] offers a strong filtering on the next variables N.

The model contains global constraints specialized for scheduling problems. We use the notation $[f(x) \mid x \in X]$ to represent the vector $[f(x_1), \ldots, f(x_n)]$ for $X = \{x_1, \ldots, x_n\}$. The constraint (3.10) limits to q_p the number of simultaneous minor setups accomplished by a person of the profession p. The constraint (3.11) limits to $|W|$ the number of simultaneous major setups. The constraint (3.12) breaks symmetry by forcing tasks producing the same product style on the same loom to execute in order of due dates.

Minimize T subject to :

$$\text{Minimize} \quad \sum_{\mathrm{i}\in\mathcal{I}} \mathrm{r_i} \cdot \max(0, \mathrm{S_i} + \mathrm{p_i} - \mathrm{d_i}) \tag{3.1}$$

$$c_i = c_{l_i}^{\text{init}} \implies S_i + p_i \leq S_{A_i}^{\text{major}} \qquad \forall i \in \mathcal{I} \tag{3.2}$$

$$c_i \neq c_{l_i}^{\text{init}} \implies S_{A_i}^{\text{major}} + p_l^{\text{major}} \leq S_i \qquad \forall i \in \mathcal{I} \tag{3.3}$$

$$S_i + p_i \leq S_{i,1}^{\text{minor}} \qquad \forall i \in \mathcal{I} \tag{3.4}$$

$$S_{i,p+1}^{\text{minor}} \geq S_{i,p}^{\text{minor}} + t_{i,N_i,p} \qquad \forall i \in \mathcal{I}, \forall p \in P \setminus \{|P|\} \tag{3.5}$$

$$S_{N_i} = S_{i,|P|}^{\text{minor}} + t_{i,N_i,|P|} \qquad \forall i \in \mathcal{I} \tag{3.6}$$

$$N_{\sigma_l} = F_{l+1} \qquad \forall l \in [1, |L| - 1] \tag{3.7}$$

$$N_{\sigma_{|L|}} = F_1 \tag{3.8}$$

$$\textsc{Circuit}(N) \tag{3.9}$$

$$\textsc{Cumulative}([S_{i,p}^{\text{minor}} \mid i \in \mathcal{I}], [t_{i,N_i,p} \mid i \in \mathcal{I}], 1, q_p) \qquad \forall p \in P \tag{3.10}$$

$$\textsc{Cumulative}([S_l^{\text{major}} \mid l \in L], [p_l^{\text{major}} \mid l \in L], 1, |W|) \tag{3.11}$$

$$S_a \leq S_b \qquad \forall a, b \in \mathcal{I}, z_a = z_b \wedge l_a = l_b \wedge d_a \leq d_b \tag{3.12}$$

3.7 Resolution

We present four methods to solve the model from the previous section. Some are pure heuristics or are *rules of thumb* used in the industry. These methods are either used as a point of comparison or are integrated as a branching heuristics in the constraint solver.

3.7.1 The Greedy Method Based on Due Dates

The first method, denoted Greedy, consists of executing the tasks on a loom in non-decreasing order of due dates. Ties are arbitrarily broken. If a resource is unavailable to either execute the minor or major setup that precedes a task, it delays the execution of the task until the resource becomes available and has time to complete the setup.

While this method might return a sub-optimal solution, it is nevertheless a rule of thumb used by people in the industry to generate an initial schedule that can be improved later. It is also a point of comparison for other methods.

3.7.2 The Circuit Method

The next approach denoted Circuit focuses on the Circuit constraint. We want to find the circuit that minimizes the sum of the setup times. While this objective function is not the weighted tardiness, it is correlated. Indeed, shorter setups lead to shorter idle times and therefore earlier completion times.

We solve the TSP instance induced by the Circuit constraint using the solver Concorde [4]. Concorde is quick to solve TSP instances, especially for instances with fewer than 1000 cities. We sort the looms

in non-decreasing amount of time available to execute the minor setups, i.e. for each loom $l \in L$, we compute $H - a_l - \sum_{i|l_i=l} p_i - p_l^{\text{major}}$. Loom by loom in the sorted order, we assign their tasks in the same order found by Concorde. We delay the minor and major setups until the resource is available. The resulting schedule minimizes the amount of time spent by the workers on the setups without considering tardiness.

3.7.3 The CP Method

The next approach denoted CP consists in coding the model with the MiniZinc [13] language and submitting the model to the constraint solver Chuffed [3].

As a branching heuristics, we generate a *template solution* before the search with either the method Greedy or Circuit. During the search, we choose a next variable N_i that can be assigned to the same value as in the template solution. That was implemented by declaring a vector B of Boolean variables connected to the model with the constraints $B_i = 1 \iff N_i = N_i^t$ where N_i^t is the successor of task i in the template solution. The heuristic branches on the vector B by setting the variables to the value 1. This has for effect to set $N_i = N_i^t$. In case the value 0 is selected for B_i (for instance, after a backtrack), that imposes the constraint $N_i! = N_i^t$ but this does not fix the variable N_i. Once the variables in B are assigned, the solver chooses the starting variables of a task, a minor setup, or a major set up with the smallest value in its domain and assigns this variable to this smallest value. This has for effect to set all next variables that were not already assigned to a value.

3.7.4 The LNS Method

The large neighborhood search, denoted LNS, starts with an initial solution that could be, for instance, the solution obtained from the methods Greedy and Circuit. It iteratively improves this solution by randomly selecting looms. The CP method is then called to reschedule the tasks on these looms while leaving the tasks on unselected looms untouched.

A note about our implementation. For each iteration, we generate a MiniZinc data file that contains the execution time of the unselected tasks. The model has unary constraints of the form $S_i = v$ to fix the variables to a value. A constraint states that the objective value must improve over the best solution found so far. We solve for satisfaction, i.e. we stop the search once we found an improving solution. An iteration of the LNS is given a timeout (we use 5 minutes) after which, if no solution is found, we pass to the next iteration without changing the current solution but by resetting 10% fewer looms. Whenever the solver returns unsatisfiable, we reset 10% more looms until a solution is found or infeasibility is proven (which implies that the current solution is optimal).

Since the time for compiling the MiniZinc code is significant and could be avoided if the local search was directly implemented in C++ calling the Chuffed solver, we do not count the MiniZinc compilation time in the solving time. The search stops when the computation time, that excludes MiniZinc compilation time, reaches a timeout.

In a context of a local search, we do not use the branching heuristics described in Section 3.7.3 since this heuristic aims at finding a solution similar to a template solution. In a local search, one rather wants to find a solution that is different from the current one. We simply randomly assign the next variables N.

3.8 Experiments

3.8.1 Instances

Our industrial partner shared 4 instances with 571, 592, 756, and 841 tasks. From each instance, we create a dataset of 10 instances by randomly selecting 10 %, 20 %, …, 100 % of the tasks from the original instance. Once the tasks are selected, this selection is used for all the tests and all the solving methods. This allows us to see how our algorithms scale with the number of tasks. The number of looms $|L| = 90$, the number of professions $|P| = 3$ with quantities $[q_1, q_2, q_3] = [5, 3, 2]$, and the number of workers $|W| = 1$ for the major setups remains constant for all instances of all datasets.

3.8.2 Experimental Setup

The CP model was written in the MiniZinc language [13]. We use the solver Chuffed [3] with the free search parameter [17]. The LNS method is implemented in Python. We ran the experiments on a computer with the following configuration : Ubuntu 19.10, 16 GB ram, Processor Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, 4008 Mhz, 4 Cores, 8 Logical Processors.

3.8.3 Methodology

We solved all instances using the Greedy and Circuit method. For the CP method, we tried three different branching heuristics : CP+Greedy assigns the next variables according to the solution of Greedy, CP+Circuit uses the solution from Circuit, and CP+Random randomly assigns the next variable and uses restarts with a Luby sequence with a scale of 250.

For the methods based on CP, we tried the three configurations with LNS and without LNS. At each iteration, 50 % of the looms are rescheduled. A specific iteration is given a timeout of 5 minutes to improve the solution.

All methods are given a timeout of 15 minutes. As the search goes, the CP methods (with or without LNS) keeps improving their best solution. We keep track of the objective value and time of the solutions as we find them during the search.

3.8.4 Results

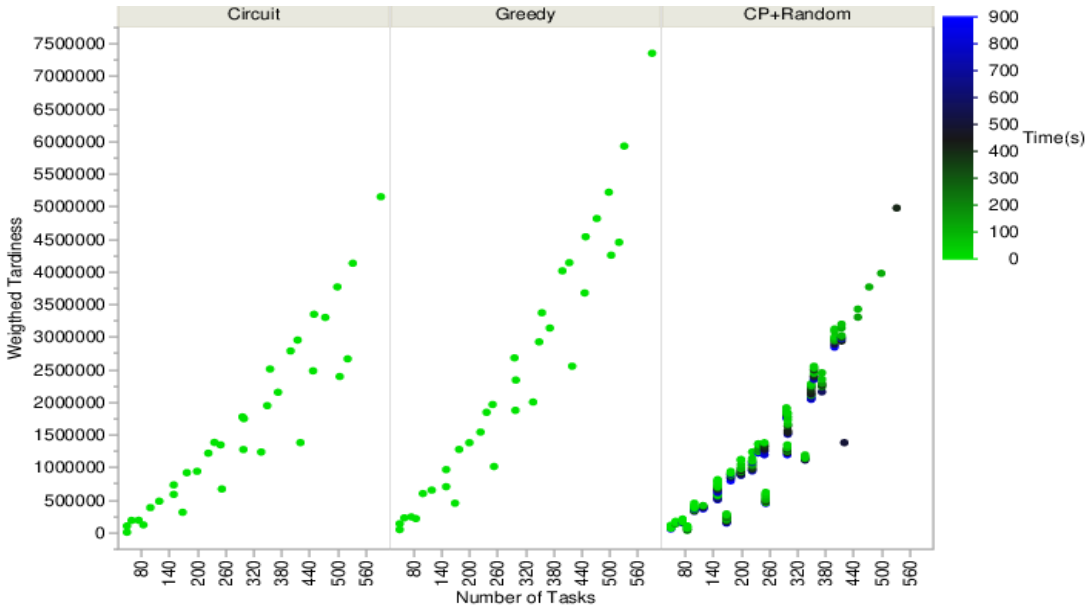


FIGURE 3.3 – Comparison between CIRCUIT, GREEDY, and one of the methods based on CP

Figure 3.3 shows the performances of the methods GREEDY and CIRCUIT. Each point represents a solution obtained for an instance : on the x-axis is the number of tasks in the instance and on the y-axis is the objective value of the solution returned by the method. The color indicates after how much time (in seconds) the solution was found.

As expected, instances with a larger number of tasks get a larger weighted tardiness. Indeed, since the resources and the due dates remain the constant among all instances, it gets harder to deliver products on time if we increase the number of orders without increasing the resources or delaying the due dates.

Both the methods GREEDY and CIRCUIT solve every instance in less than a second. The quality of the solution is not competitive with any method using CP. However, we will see that GREEDY and CIRCUIT are nevertheless useful to guide the branching heuristics of the CP solver.

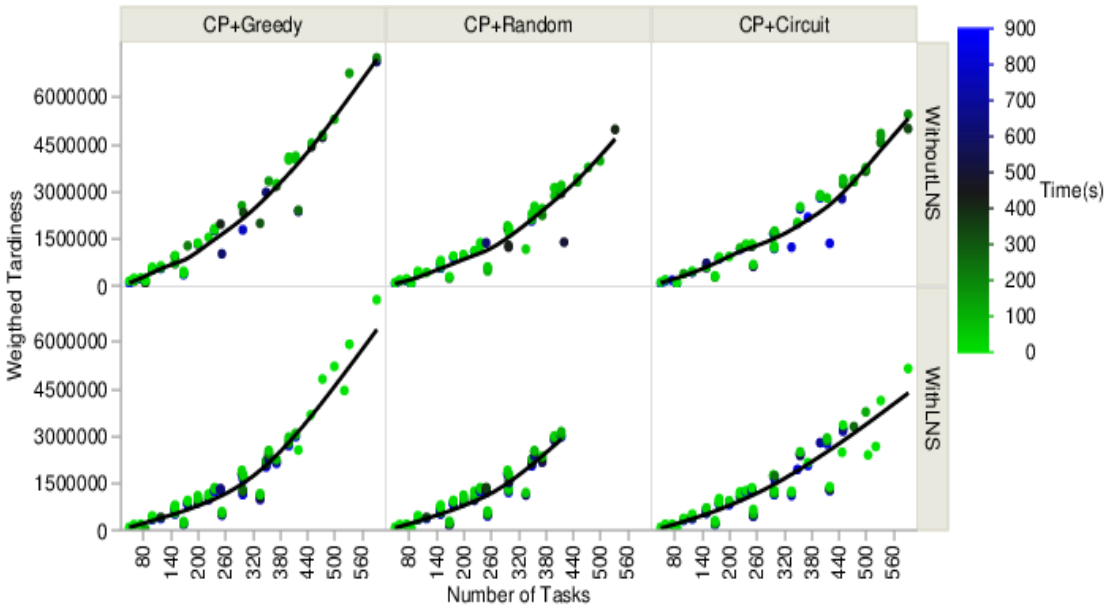


FIGURE 3.4 – Comparison of methods based on CP

Figure 3.4 presents a comparison of the six methods based on CP. The left, middle, and right graphs make the branching heuristics vary. It is either based on the CIRCUIT, GREEDY, or the random heuristics. The graphs on the top use the standard CP search while the graphs at the bottom use the LNS. For the LNS, the initial solutions are also based on CIRCUIT, GREEDY, and RANDOM. For CP+CIRCUIT+LNS and CP+GREEDY+LNS, the next variables N_i are set in the initial solution according to the precedences in the solutions generated by CIRCUIT and GREEDY. For CP+RANDOM+LNS, the initial solution is completely random.

The first thing to analyze on Figure 3.4 is how the methods behave on instances with more than 500 tasks. The methods CP+RANDOM CP+RANDOM+LNS are unable to solve all instances while other methods do. The CP+GREEDY method is more stable than CP+RANDOM while CP+RANDOM obtains better solutions. CP+CIRCUIT and CP+CIRCUIT+LNS outperform CP+RANDOM and CP+RANDOM+LNS in both stability and objective value.

Globally, CP+CIRCUIT and CP+CIRCUIT+LNS offer the solutions with the smallest weighted tardiness. This might look surprising at first sight because CIRCUIT aims at minimizing the amount of setups while GREEDY directly minimizes tardiness. However, the solution that CIRCUIT generates is optimal according to the amount of setup while the GREEDY algorithm only returns an approximation for the weighted tardiness. Guiding the search towards a solution that is optimal, even according to a different but correlated criteria, provides the best solution.

The third observation is that CP+CIRCUIT+LNS generally outputs better solutions than CP+CIRCUIT. The same goes for CP+GREEDY+LNS compared to CP+GREEDY. While other methods perform better with our homemade LNS, the random heuristic offers poor results.

Finally, we observe that using LNS is beneficial. The poor results of the LNS on the random heuristic is most likely due to a bad initial solution.

3.9 Conclusion

We presented a model to solve an industrial instance presented by our partner. We showed how a solution from a simplification (the TSP) can guide the search to obtain better solutions. Our model is now able to find solutions to the expected range of tasks. The integration of our program to the operations planning team is in progress.

3.10 Bibliographie

[1] Alexandre Mercier-Aubin, J. G., and Quimper, C.-G. Leveraging constraint scheduling : A case study to the textile industry. to appear in proceeding of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2020.

[2] Chakrabortty, R. K., Sarker, R. A., and Essam, D. L. Resource constrained project scheduling with uncertain activity durations. *Computers & Industrial Engineering 112* (2017), 537 – 550.

[3] Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. Chuffed, a lazy clause generation solver, 2018.

[4] Cook, W. *In Pursuit of the Traveling Salesman : Mathematics at the Limits of Computation*. Princeton University Press. Princeton University Press, 2011.

[5] Costa, A., Cappadonna, F. A., and Fichera, S. A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology 69* (2013), 2799—2817.

[6] Danna, E., and Perron, L. Structured vs. unstructured large neighborhood search : A case study on job-shop scheduling problems with earliness and tardiness costs. In *Principles and Practice of Constraint Programming – CP 2003* (Berlin, Heidelberg, 2003), F. Rossi, Ed., Springer Berlin Heidelberg, pp. 817–821.

[7] Focacci, F., Laborie, P., and Nuijten, W. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (2000), pp. 92–101.

[8] Hermann, M., Pentek, T., and Otto, B. Design principles for industrie 4.0 scenarios. In *49th Hawaii International Conference on System Sciences (HICSS)* (2016), pp. 3928-–3937.

[9] Kagermann, H., Helbig, J., Hellinger, A., and Wahlster, W. Recommendations for implementing the strategic initiative industrie 4.0 : Securing the future of german manufacturing industry ; final report of the industrie 4.0 working group. Tech. rep., Forschungsunion, 2013.

[10] Kajgard, E. *Route Optimisation for Winter Road Maintenance using Constraint Modelling*. PhD thesis, Uppsala Universitet, 2015.

[11] Lauriere, J.-L. A language and a program for stating and solving combinatorial problems. *Artificial intelligence 10*, 1 (1978), 29–127.

[12] López-Ibáñez, M., Blum, C., Ohlmann, J. W., and Thomas, B. W. The travelling salesman problem with time windows : Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing 13*, 9 (2013), 3806–3815.

[13] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. Minizinc : Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming – CP 2007* (2007), pp. 529–543.

[14] Psaraftis, H. N. k-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research 13*, 4 (1983), 391–402.

[15] Savelsbergh, M. W. P. Local search in routing problems with time windows. *Annals of Operations Research 4*, 1 (Dec 1985), 285–305.

[16] Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. Why cumulative decomposition is not as bad as it sounds. In *Principles and Practice of Constraint Programming - CP 2009* (Berlin, Heidelberg, 2009), I. P. Gent, Ed., Springer Berlin Heidelberg, pp. 746–761.

[17] Shishmarev, M., Mears, C., Tack, G., and Garcia de la Banda, M. Learning from learning solvers. In *Principles and Practice of Constraint Programming* (Cham, 2016), M. Rueher, Ed., Springer International Publishing, pp. 455–472.

[18] Tran, T. T., and Beck, J. C. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence ECAI'12* (2012), pp. 774–779.

Chapitre 4

The Confidence Constraint : A Step Towards Stochastic CP Solvers

Material from : 'Alexandre Mercier-Aubin, Ludwig Dumetz, Jonathan Gaudreault and Claude-Guy Quimper, The Confidence Constraint : A Step Towards Stochastic CP Solvers, In proceedings of the 26th International Conference on Principles and Practice of Constraint Programming, published in 2020, 2020 Springer Nature Switzerland AG' [2].

4.1 Résumé

Nous introduisons une nouvelle contrainte chance nommée CONFIDENCE qui assure avec probabilité γ qu'un ensemble de variables ne sera pas plus petit que des variables aléatoires pour lesquelles leur distribution sont connues. Cette contrainte est utile en optimisation stochastique, car elle offre une garantie de robustesse dans les solutions par rapport aux évènements probabilistes externes. Cela permet de contrôler le compromis entre l'optimization de la fonction-objectif et d'assurer la satisfaction des solutions sous l'influence d'évènements stochastiques. Nous présentons un algorithme de filtrage pour cette contrainte avec explications. Nous appliquons cette contrainte au projet de notre partenaire industriel. Dans ce problème d'ordonnancement industriel, les tâches ont des durées aléatoires variant selon le nombre de bris pendant leur exécution. Nous évaluons nos solutions grâce à un simulateur et nous montrons que cette contrainte permet des solutions robustes en un temps de calcul satisfaisant.

4.2 Abstract

We introduce the CONFIDENCE constraint, a chance constraint that ensures, with probability γ, that a set of variables are no smaller than random variables for which the probability distribution is given. This constraint is useful in stochastic optimization to ensure that a solution is robust to external random events. It allows to control the trade-off between optimizing the objective function and ensuring the satisfiability of the solution under random parameters. We present a filtering algorithm for this

constraint with explanations. We apply the constraint to a case study, an industrial scheduling problem where tasks have random processing times due to possible breakdowns during their execution. We evaluate our solutions with simulations and show that this new constraint allows robust solutions in decent computation time.

4.3 Introduction

Decisions in an organization are taken at different levels : strategic, tactical, operational, and execution. In some situations, it is important that decisions taken at the operation level take into account what could happen at the execution level. For instance, one can schedule tasks while taking into account that breakdowns might occur during the execution and that the schedule might not be followed as expected. Stochastic optimization allows taking decisions at one level while coping with random events at the next level. One way to achieve this goal is by using *chance constraints* in the optimization model to guarantee that the solution holds beyond a given threshold probability.

We introduce a new chance constraint called Confidence that forces variables to take sufficiently large values so that random variables, following a known distribution, are unlikely to be greater. Let $X_1, \ldots, X_n$ be decision variables, let $D_1, \ldots, D_n$ be statistical distributions, and let γ be the confidence threshold (a constant parameter). Each distribution D_i must have a well-defined cumulative distribution function cdf_i. The Confidence constraint is defined as follows.

$$\textsc{Confidence}([X_1, \ldots, X_n], [\text{cdf}_1, \ldots, \text{cdf}_2], \gamma) \iff \prod_{i=1}^{n} \text{cdf}_i(X_i) \geq \gamma \tag{4.1}$$

In other words, let Y_i be an independent random variable following a distribution D_i. The chance that at least one random variable Y_i takes a value greater than X_i must be less than γ.

Such a constraint is particularly useful in stochastic scheduling where tasks can execute for a longer time than expected. In order to let sufficient time for the tasks to execute in a schedule, one can assign processing times to the tasks that are sufficiently long γ percent of the time. We will show that such an approach has many advantages and is particularly well suited for constraint programming.

The rest of the paper is divided as follows. Section 4.4 presents background information about stochastic optimization. Section 4.5 introduces the Confidence constraint and its filtering algorithm. Section 4.6 presents a case study, an industrial scheduling problem that led to (and financed) this research. Section 4.7 presents our simulator that will be used in our experiments in Section 4.8. We present and analyze the results in Section 4.8.3. We conclude in Section 4.9.

4.4 Background

Stochastic problems are optimization problems for which part of the input is given with random variables. One aims at a solution of *good quality* over all possible values that these random variables can

take. Usually, one wants to optimize the expected objective value and/or to ensure feasibility occurs with a given probability.

Stochastic linear programs are linear programs $\min\{c^T x \mid Ax \leq b, x \geq 0\}$ for which some parameters in c, A, or b are replaced by random variables. These linear problems are particularly well studied [11]. They can, for instance, encode scheduling problems where processing times are subject to random delays [5].

There are several approaches to stochastic optimization. One important approach is called scenario-based optimization. It consists, in one way or another, in achieving deterministic optimization on sample data called scenarios. A scenario is a possible outcome of stochastic events. The number of possible scenarios grows exponentially with the number of stochastic variables. Chance constraints [21] can achieve scenario-based optimization by constraining the solution to satisfy a scenario with a given probability. The concept emerged in stochastic linear programming, but is not intrinsically linear and can be applied to any constraint satisfaction problem. The formulae described in [21] inspired the ones stated in Section 4.5. However, as the they sum probabilities over possible scenarios, some formulae might take too much time to compute for problems with hundreds of stochastic variables.

In constraint programming, Walsh [19] proposes an extension of a constraint satisfaction problem by modeling decision problems with uncertainty. In this extension, stochastic variables based on probabilistic distributions coexist with common constraint programming decision variables. The solver computes a policy, i.e. a solution for each possible outcome of the stochastic variables. While this suits the needs for many problems, for problems with hundreds of random variables, one cannot state (not to mention computing) a complete policy.

The modeling language MiniZinc [13, 18] is also adapted to stochastic optimization [14] keeping the modeling process independent from the solving technique. It requires to encode a sample of scenarios in a vector. As the number of random variables grows, the cardinality of the sample scenarios set become insignificant compared to the number of possibilities and the quality of the solutions decays.

Stochastic problems is one way to obtain solutions robust to change. However, there exist other robust optimization approaches that do not rely on probabilities. For instance, *super solutions* are solutions that adapt themselves in case of a change [10]. Such solutions are, however, hard to compute and are often restricted to small instances. The FlexC constraint [7, 8] can model cumulative scheduling problems where tasks have two processing times : a normal processing time and a delayed one. The constraint ensures that if any subset of k tasks is delayed, the cumulative resource is not overloaded. However, the constraint does not support delays of variable duration.

The outreach of chance constraints goes far beyond scheduling problems. Rossi et al. [15] present an algorithm designed to compute an optimal policy for inventory levels. Based on current demand, chance constraints are used in such a way that one can predict more accurate future demand. While we do not aim at solving the same problem, this is the closest chance constraint algorithm we could find

in constraint programming to what we propose.

4.5 The CONFIDENCE Constraint

4.5.1 Description

The CONFIDENCE constraint defined in (4.1) has for scope a vector of integer variables $[X_1, \ldots, X_n]$ and has for parameters a vector of random distributions $[D_1, \ldots, D_n]$. A collection of independent random variables $Y_1, \ldots, Y_n$ following these distributions i.e. the random variable Y_i follows the distribution D_i. Each distribution D_i is fully described by its *cumulative distribution function* $\text{cdf}_i(v) = P[Y_i \leq v]$ that computes the probability that the random variable Y_i takes a value smaller than or equal to a value v. We also consider its inverse function called the *quantile function* $Q_i(p) = \min\{v \mid P[Y_i \leq v] \geq p\}$ that computes the smallest value v for which variable Y_i takes a value no greater than v with probability p. The distributions can be common distributions (e.g. Poisson, Uniform, ...) or can be custom much like the one shown in Section 4.6. The distributions can be different for every variable. Finally, the constraint takes a parameter γ that is a confidence threshold, a probability between 0 and 1. Note that even if this constraint handles probabilities, it constrains integer variables and is therefore compatible with solvers that only handle this type of variable. Moreover, if the solver does not handle floating point parameters, it is possible to express γ in percents and to give it an integer value.

The constraint is satisfied when, with probability at least γ, all random variables Y_i take a value smaller than or equal to their corresponding threshold X_i. Since the random variables Y_i are independent, the constraint is satisfied when

$$P[\bigwedge_{i=1}^{n} Y_i \leq X_i] \geq \gamma. \tag{4.2}$$

Since the random variables are independent, we obtain

$$\prod_{i=1}^{n} P[Y_i \leq X_i] \geq \gamma. \tag{4.3}$$

For numerical stability reasons, we use this form

$$\sum_{i=1}^{n} \ln(P[Y_i \leq X_i]) \geq \ln(\gamma). \tag{4.4}$$

The CONFIDENCE constraint is useful in problems where unknown events might arise. In a scheduling problem, it is common to have tasks that take longer to execute than expected. One usually plans more time than necessary for these tasks in a schedule. Let X_i be the processing time of task i and D_i be a distribution on the observed processing times in the past. The CONFIDENCE constraint allows planning sufficient times for the execution of the tasks in say, $\gamma = 95\%$ of the time. This confidence threshold

allows making schedules that can be executed without modification 95% of the time without over estimating the duration of the tasks.

In a production problem where we want to produce sufficient goods to satisfy an unknown demand, we let X_i be the amount of good i that is produced and Y_i be the amount that needs to be produced to fulfill the demand. We want $X_i \geq Y_i$ which is equivalent to $-X_i \leq -Y_i$. The CONFIDENCE constraint can restrict the quantities $-X_i$ to be smaller than the random variable $-Y_i$ with probability γ.

4.5.2 Filtering Algorithm

Algorithm 1 is the filtering algorithm directly derived from applying interval arithmetic on inequality (4.4). Line 1 computes α, the log of the highest probability that can be reached on the left-hand side of the inequality. It is computed from the random variables' cumulative distribution functions. If this probably is too low, line 2 triggers a failure. Otherwise, the lower bound of each variable domain is tested for consistency. The test on line 3 is equivalent to testing $\sum_{j \neq i} \ln(P[Y_j \leq \max(\text{dom}(X_j))]) + \ln(P[Y_i \leq \min(\text{dom}(X_i))]) < \ln(\gamma)$. If the test is positive, then $\min(\text{dom}(X_i))$ does not have a support and should be filtered out from the domain. In order to evaluate what is the smallest value in $\text{dom}(X_i)$ with a support, we use the quantile function on line 4. We search for the smallest value $v \in \text{dom}(X_i)$ such that $\beta + \ln(P[Y_i \leq v]) \geq \ln(\gamma)$ which is equivalent to $P[Y_i \leq v] \geq \gamma e^{-\beta}$. This is directly provided by the quantile function $Q_i(\gamma e^{-\beta})$.

The algorithm accepts any distribution for which the functions cdf_i and Q_i can be computed. Such functions for common distributions are already implemented in libraries like Boost [16]. This includes Pareto, Gaussian, Poisson, Laplace, Uniform, etc. It is also possible to create custom distributions.

Algorithme 1 : CONFIDENCE Filtering($[\text{dom}(X_1), \ldots, \text{dom}(X_n)], [D_1, \ldots, D_n], \gamma$)

Let Y_i be a random variable following distribution D_i for $i \in \{1, \ldots, n\}$;
1 $\alpha \leftarrow \sum_{i=1}^{n} \ln(\text{cdf}_i(\max(\text{dom}(X_i))))$;
2 **if** $\alpha < \ln(\gamma)$ **then**
  Fail with explanation $\bigwedge_{i=1}^{n} [\![X_i \leq \max(\text{dom}(X_i))]\!] \implies \text{False}$;
for $i \in \{1, \ldots, n\}$ **do**
  $\beta \leftarrow \alpha - \ln(\text{cdf}_i(\max(\text{dom}(X_i))))$;
3 **if** $\beta + \ln(\text{cdf}_i(\min(\text{dom}(X_i)))) < \ln(\gamma)$ **then**
4 $v \leftarrow Q_i(\gamma e^{-\beta})$;
    Filter X_i with explanation $\bigwedge_{j \neq i} [\![X_j \leq \max(\text{dom}(X_j))]\!] \implies [\![X_i \geq v]\!]$;

From the arithmetic of intervals, the algorithm enforces bounds consistency on the CONFIDENCE constraint, but also domain consistency. Indeed, the algorithm tests on line 3 the assignment $[\max(\text{dom}(X_1)), \ldots, \max(\text{dom}(X_{i-1})), \min(\text{dom}(X_i)), \max(\text{dom}(X_{i+1})), \ldots, \max(\text{dom}(X_n))]$. If this assignment satisfies the CONFIDENCE constraint, so does $[\max(\text{dom}(X_1)), \ldots, \max(\text{dom}(X_{i-1})), v, \max(\text{dom}(X_{i+1})), \ldots, \max(\text{dom}(X_n))]$

for any value $v \in \text{dom}(X_i)$. Hence, all values in $\text{dom}(X_i)$ are domain consistent. Otherwise, the value $\min(\text{dom}(X_i))$ is filtered out from the domain of X_i and the quantile function guarantees that the new lower bound on the domain satisfies the constraint.

The running time complexity is dominated by the number of calls to the cumulative distribution functions and the quantile functions. In the worst case, the algorithm performs $3n$ calls to the cumulative distribution functions and at most n calls to the quantile functions. This leads to a running time complexity of $O(n)$.

4.6 Case Study

4.6.1 The deterministic version

We have an industrial partner in the textile industry whose needs motivate the theoretical contribution and whose data allow to empirically evaluate this contribution on an industrial problem.

The textile manufacturer has to schedule a set of tasks $\mathcal{I}$ on looms L. The tasks represent textile pieces to weave or setups to prepare a loom for the next textile. A piece of textile i has a style z_i which is a number that encodes the width of the textile, the type of thread, the weaving pattern, etc. A task i is pre-assigned to loom $l_i \in L$ in order to simplify the problem. In order to process the task, this loom needs to be set up with configuration c_i. Each task $i \in \mathcal{I}$ has a due date d_i and a priority r_i. The higher the priority, the more urgent the task is. In the deterministic version of the problem, every piece of textile i has a predefined weaving duration p_i.

Looms are disjunctive resources [4] that can only weave one piece of textile at the time or being subject to one set up at the time. A loom $l \in L$ becomes available at time a_l. Prior to this time point, the loom completes tasks that were started and that cannot be changed. A loom l is initially in configuration c_l^{init} and upon a *major setup* operation of duration p_l^{major}, its new configuration becomes c_l^{final}. There is only one possible major setup per loom. A loom l can only execute a task i if it is assigned to that loom ($l_i = l$). It executes the task before its major setup if $c_i = c_l^{\text{init}}$ or after if $c_i = c_l^{\text{final}}$. A major setup is performed by a worker from the set W. Hence, at most $|W|$ major setups can be performed simultaneously.

A *minor setup* takes place between two consecutive weaving tasks on a loom, but does not change the configuration of the loom. There are up to 3 types of employees $p \in P$ interacting with a loom during a minor setup. The order is always as follows : weavers, beam tiers, and mechanics. Therefore, a minor setup is decomposed into an ordered sequence of 3 tasks, one associated for each profession. The minor setup duration is sequence-dependent in the sense that the duration $t_{i,j,p}$ for the profession p is a function of the task i before the setup and the task j after. Employees are cumulative resources. Since there are q_p employees of profession $p \in P$, up to q_p minor setup tasks associated to profession p can be simultaneously executed across the looms.

$$\text{Minimize} \quad \sum_{\mathrm{i}\in\mathcal{I}} \mathrm{r_i} \cdot \max(0, \mathrm{S_i} + \mathrm{p_i} - \mathrm{d_i}) \tag{4.5}$$

$$c_i = c_{l_i}^{\text{init}} \implies S_i + p_i \leq S_{l_i}^{\text{major}} \qquad \forall i \in \mathcal{I} \tag{4.6}$$

$$c_i \neq c_{l_i}^{\text{init}} \implies S_{l_i}^{\text{major}} + p_l^{\text{major}} \leq S_i \qquad \forall i \in \mathcal{I} \tag{4.7}$$

$$S_i + p_i \leq S_{i,1}^{\text{minor}} \qquad \forall i \in \mathcal{I} \tag{4.8}$$

$$S_{i,p+1}^{\text{minor}} \geq S_{i,p}^{\text{minor}} + t_{i,N_i,p} \qquad \forall i \in \mathcal{I}, \forall p \in P \setminus \{|P|\} \tag{4.9}$$

$$S_{N_i} = S_{i,|P|}^{\text{minor}} + t_{i,N_i,|P|} \qquad \forall i \in \mathcal{I} \tag{4.10}$$

$$N_{\sigma_l} = F_{l+1} \qquad \forall l \in [1, |L| - 1] \tag{4.11}$$

$$N_{\sigma_{|L|}} = F_1 \tag{4.12}$$

$$\textsc{Circuit}(N) \tag{4.13}$$

$$\textsc{Cumulative}([S_{i,p}^{\text{minor}} \mid i \in \mathcal{I}], [t_{i,N_i,p} \mid i \in \mathcal{I}], 1, q_p) \qquad \forall p \in P \tag{4.14}$$

$$\textsc{Cumulative}([S_l^{\text{major}} \mid l \in L], [p_l^{\text{major}} \mid l \in L], 1, |W|) \tag{4.15}$$

$$S_a \leq S_b \qquad \forall a, b \in \mathcal{I}, z_a = z_b \wedge l_a = l_b \wedge d_a \leq d_b \tag{4.16}$$

FIGURE 4.1 – Model equations directly taken from our previous paper [1]

In the deterministic version of the problem, the unknowns are the starting time S_i of each task $i \in \mathcal{I}$, the starting time $S_{i,p}^{\text{minor}}$ of the minor setup succeeding a task $i \in \mathcal{I}$ for each profession $p \in P$, and the starting time S_l^{major} of the major setup of loom $l \in L$. The objective function is the weighted tardiness where the weights are the task priorities. We use this metric since the tight deadlines make it unreasonable to search for a schedule without tardiness.

The constraint model is similar to the one of a resource-constrained project scheduling problem (R-CPSP) [3]. In a RCPSP, there are tasks to schedule and cumulative resources. Our problem differs from the RCPSP in the sense that minor setups have sequence-dependent processing times.

Figure 4.1 presents the constraint model of the deterministic problem as published in [1]. The variable F_l encodes the first task on loom l. The order of the tasks is encoded in an array of variables N such that for any piece of textile i, N_i is the next piece of textile to weave. The dummy task σ_l acts as a sentinel to encode the last task to execute on loom l. The task following the last task on a loom is the first task on the next loom (see constraints (4.11) and (4.12)). The vector N forms a circuit visiting all the tasks hence Constraint (4.13). We model the disjunctive resources using less than or equal to constraints. Two CUMULATIVE constraints model the different setup types (minor and major).

4.6.2 The stochastic version

The solver acts as a black box and returns a solution that satisfies the constraints. The resulting schedule is valid, yet this solution does not consider the probabilities of different stochastic events. Since a loom weaves many threads, each job has a high chance of having multiple breakdowns. Each task is subject to

two types of breakdowns with different probabilities and repair time. A warp thread has a high chance of breaking, but is quick to repair and the repair process is often automated. However, a weft thread rarely breaks, but when it does, an employee gets involved. Since both types of breakdowns are observable on a day-to-day basis, we must be able to consider them in a single distribution. In our case, the breakdowns follow Poisson distributions. We suppose the independence of both types of breakdowns (warp thread and weft thread). Since the sum of two independent random Poisson variables is a random Poisson variable [9], we can sum the λ-parameters of both breakdown distributions. Therefore, the duration of the breakdowns in our problem is a random variable that follows the sum of two Poisson distributions, which is equivalent to a single Poisson distribution. Depending on the task $i \in \mathcal{I}$ and the loom $l \in L$, the weft thread has an average number of breakdowns $\beta_{i,l}^{\text{weft}}$ between 0 to 10 times and when it breaks, it takes 10 minutes to repair. The warp thread has an average number of breakdowns $\beta_{i,l}^{\text{warp}}$ between 0 and 40 times and it takes 2 or 4 minutes to repair. The average numbers of breakdowns $\beta_{i,l}^{\text{weft}}$ and $\beta_{i,l}^{\text{warp}}$ are proportional to the duration of task i on loom l. The λ-parameter of the Poisson distribution for the delay caused by the breakdowns is therefore $10 \cdot \beta_{i,l}^{\text{weft}} + 2 \cdot \beta_{i,l}^{\text{warp}}$ or $10 \cdot \beta_{i,l}^{\text{weft}} + 4 \cdot \beta_{i,l}^{\text{warp}}$. Our case study is about solving the stochastic version of this problem when weaving tasks might take longer to execute than expected.

A solution with fixed processing times : One way to solve the problem could be to artificially increase the duration of the weaving tasks by a given percentage. This prevents the unplanned events from starving the resource pool available for setups which causes ripple effects. The drawback of this technique happens when we overestimate the duration and the number of delays. The delays must be chosen in a smart and intuitive manner. Giving the solver a choice of where to take risks can lead to a better solution. This is where our constraint comes in.

A solution with flexible processing times : As a second solution, we rather modify the constraint model with an additional variable P_i to represent the duration of a weaving task, including breakdowns, and a variable B_i that includes only the breakdowns. The task duration P_i cannot be shorter than the deterministic version of the problem, but could be longer if we plan time for breakdowns. We therefore insert this constraint.

$$P_i = p_i + B_i \tag{4.17}$$

We impose a Confidence constraint on the duration of the breakdowns to ensure that we plan for sufficiently long breakdowns with probability γ.

$$\textsc{Confidence}([B_i \mid i \in \mathcal{I}], [cdf_i \mid i \in \mathcal{I}], \gamma) \tag{4.18}$$

The Confidence constraint can guarantee that a breakdown does not alter the subsequent tasks with probability γ. That is, if the breakdown is shorter than what is computed by the solver, all tasks can still start at the expected time. If the breakdown is longer than what was planned, it might cause a ripple effect and delay the execution of the future tasks, but this only happens with probability $1 - \gamma$. Therefore, a valid solution to our problem is a solution that causes no disturbances, γ % of the time.

The rest of the new model with variable processing time is similar to [12]. The constraints 4.6 and 4.8 are modified by replacing p_i with P_i.

4.7 Simulation

The simulation model is used to evaluate the quality of the solutions. This implies measuring robustness by emulating scenarios and how well the solutions cope with stochastic events such as breakdowns. The simulator produces an in-depth three-dimensional depiction of the work environment. It considers the number of employees, their positions, the time each employee takes to move between looms, the number of looms, etc. By using a simulator, we can compare different solutions without trying them in practice. Since the company might be reluctant to directly apply the results to their schedules, this also acts as a way to alleviate doubts. The simulator was designed to test scenarios and to answer questions that are much beyond the scope of this paper and even beyond scheduling in general. It can be used to see the effect of strategic decisions on textile production and the effect of modifying the number of resources on the performance of the plant.

For our purpose, the simulation model receives as input the same data as our optimization model (configuration of the looms, availability of the resources, duration of the tasks) and the output of the optimization (the starting time of the tasks). One point that is worth mentioning is that the optimization model uses, for each weaving task, a distinct distribution for the duration of the breakdowns. However, the simulator averages out these distributions per loom and uses the same distribution for all weaving tasks on a given loom.

4.8 Experiments

4.8.1 Methodology

Three different methods were compared. First, in the Deterministic model, we ignore the stochastic events. The Deterministic model consists of using the model from Figure 4.1 without any change. In the Fixed model, we artificially increase the tasks' processing times by the sum of the average breakdown duration multiplied by the average number of breakdowns. This corresponds to the first solution presented in Section 4.6.2. Finally, the Confidence method is the model that uses our chance constraint with $\gamma \in \{0.01, 0.05, 0.1, 0.2, 0.3, \ldots, 0.8, 0.9, 0.95, 0.99\}$. The processing time of a task becomes a variable equal to the sum of the deterministic processing time (a constant) and the duration of breakdowns (another variable). These breakdown duration variables are subject to the Confidence constraint using the Poisson distributions. This corresponds to the second solution presented in Section 4.6.2.

The CP model was written in the MiniZinc 2.4.2 language [13]. We use the solver Chuffed [6] with the free search parameter [17]. The simulation is modeled using the academic version of SIMIO [20]. We ran the experiments on a computer with the configuration : Ubuntu 19.10, 64 GB ram, Proces-

sor Intel(R) Core(TM) i7-6700K CPU @ 4.00GHz, 4 Cores, 8 Logical Processors. All optimization processes were given a timeout of 30 minutes and 10 hours.

To compare the quality of the schedules provided by the different methods, we use the mean simulated weighted tardiness. Let E_i^s be the ending time of task i in simulation s. The mean simulated weighted tardiness $\bar{T}$ is calculated as follows :

$$\bar{T} = \frac{1}{n} \sum_{s=1}^{n} \sum_{i \in \mathcal{I}} r_i \max(0, E_i^s - d_i) \tag{4.19}$$

The inner sum computes the weighted tardiness of a simulation s while the outer sum averages over n simulations. We use $n = 100$ simulations. A simulation is a randomly generated scenario based on probabilities, i.e. that the duration of the breakdowns are randomly drawn according to the Poisson distributions we established.

During the simulation of a schedule, tasks can be delayed by breakdowns that were not planned by the solver. The task on a loom can only start after its predecessor, on the same loom, is completed. An idle technician always start working on a setup when the loom gets ready. If s/he has the choice between setting up two looms, the priority is given to the setup that was planned earlier in the original schedule. A loom that completes a task earlier than expected must wait before starting the next task as we suppose that the material for this next task is not necessarily ready before its planned start time. Indeed, when a task is planned, it instructs people on the floor to prepare the material in order for the task to start on time. This preparation is not part of the optimization since it adds too much complexity to the decision process. The resources in charge of preparing the material are voluntary omitted from the problem definition. However, if, during the execution of the plan, we inform at the last minute that a task can start earlier, we cannot assume that the preparation will be ready. In other words, the tasks are subject to precedence constraints with tasks that are external to the problem.

4.8.2 Instances

To compare the methods, we had access to four datasets. In these datasets, there are 5 workers for minor setups, 1 for major setups, and a total of 81 looms available. The scheduling horizon is 14,400 minutes which is roughly two weeks of work time. Table 4.1 shows the characteristics of each instance, including the number of pieces of textiles to weave. It also reports the resource usage rate $r = \frac{1}{H|L|} \sum_i p_i$ where H is the scheduling horizon, $|L|$ is the number of looms and p_i is the processing time (without breakdowns) of task i. We also report the usage rate when processing times are augmented with the expected breakdown duration. These breakdowns approximately add 4% to the resource usage in the last three datasets.

Dataset 1 consists of a reduced dataset where pieces of textiles with a due date of 0 (these tasks are already late) and some others with due dates larger than the horizon are removed. This leads to an easier dataset to solve for which optimal solutions are obtained within 30 minutes of computation time with every method.

Datasets 2, 3, and 4 are not modified, but cannot be optimally solved even after 10 hours. These datasets are more realistic with respect to what is usually observed in the industry where one has to work with suboptimal solutions. Those are directly extracted from our industrial partner's database.

In every instance, there is $|W| = 1$ mechanic available for major setups, 2 mechanics, 5 weavers, and 3 beam tiers available at all time for minor setups.

TABLE 4.1 – Dataset descriptions

Dataset	1	2	3	4
Textile pieces	448		480	
Resource usage (%) without / with average breakdowns	47.23/51.37	48.06/52.33	52.94/57.79	48.07/48.30

4.8.3 Results

In Figures 4.2, 4.4, 4.6, and 4.8, we present the results of our models with different methods on the x-axis. The CONFIDENCE method was tested with different confidence thresholds γ. The y-axis is the weighted tardiness. There are two colors of dots and lines. Red presents the prediction value. The prediction is essentially the objective value of the solution returned by the solver, i.e. the weighted tardiness if the plan executes as expected by the CONFIDENCE constraint. Blue shows the mean simulated weighted tardiness of the 100 runs with its 95 % confidence interval.

Figure 4.2 shows the results for the reduced dataset solvable to optimality. Notice that the red curve is non-decreasing, since an optimal schedule that satisfies the CONFIDENCE constraint with threshold γ is a feasible, but potentially suboptimal, schedule for a smaller threshold. For this instance, which is rather trivial to solve since few tasks were competing for the resources, the best strategy was to plan no extra time for the breakouts. In case of a breakout, the next tasks are simply postponed until the delayed task is completed. At this time, the resource is always available to perform the setup to execute the next task and no further delay is caused. This strategy is achieved by the DETERMINISTIC method and the CONFIDENCE constraint with $\gamma = 0.01$. However, the CONFIDENCE predicted value is closer to reality. At the opposite, for $\gamma = 0.99$, the weighted tardiness is at its highest since the duration of nearly all tasks is set to the longest possible duration. Notice how the FIXED method offers a performance similar to the CONFIDENCE method with $\gamma = 0.5$.

For the datasets 2, 3, and 4, we clearly see on Figures 4.3 to 4.8 that the solver does not produce optimal solutions even within 10 hours. Indeed, the red curve is non-monotonic. Moreover, results of the dataset 2 with $\gamma = 95\%$ are not reported as no solution was obtained within 10 hours. This happens with even lower confidence thresholds when using a timeout of 30 minutes. However, notice that the simulation follows the predictions of the model for the higher values of γ. The gap between the objective value obtained by the solver (red) and the simulation (blue) decreases by increasing γ. In every case, the best solution, as evaluated by the simulator (blue), comes from the CONFIDENCE method. In Dataset 4, the best solution is obtained at $\gamma = 20\%$. The CONFIDENCE model trades objective value for robustness.

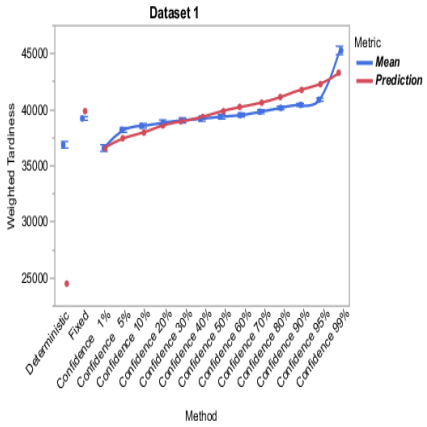


FIGURE 4.2 – Dataset with 448 textile pieces to weave

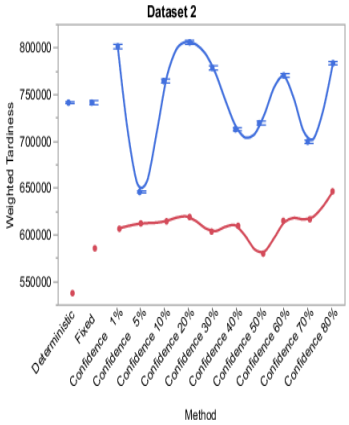


FIGURE 4.3 – Dataset with 602 textile pieces to weave. 30 minutes timeout.

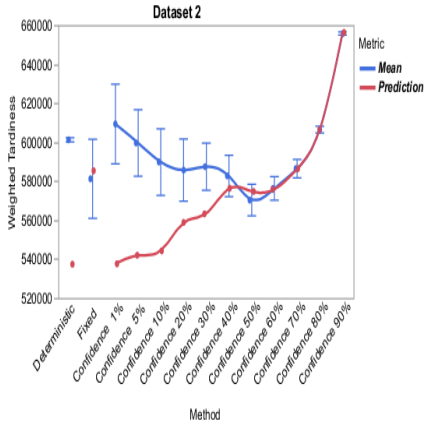


FIGURE 4.4 – Dataset with 602 textile pieces to weave. 10 hours timeout.

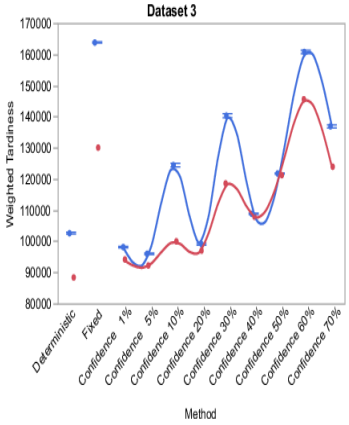


FIGURE 4.5 – Dataset with 480 textile pieces to weave. 30 minutes timeout

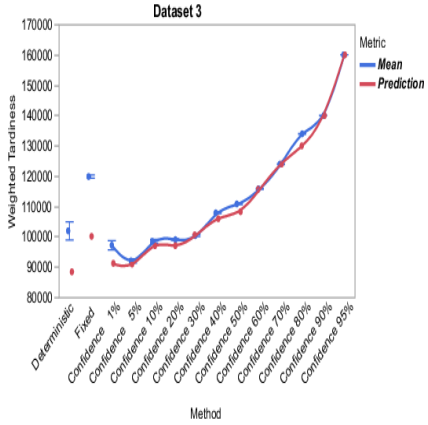


FIGURE 4.6 – Dataset with 480 textile pieces to weave. 10 hours timeout.

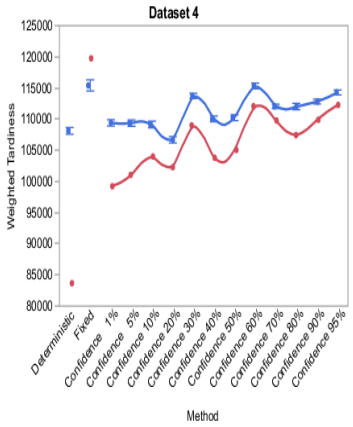


FIGURE 4.7 – Dataset with 525 textile pieces to weave. 30 minutes timeout.

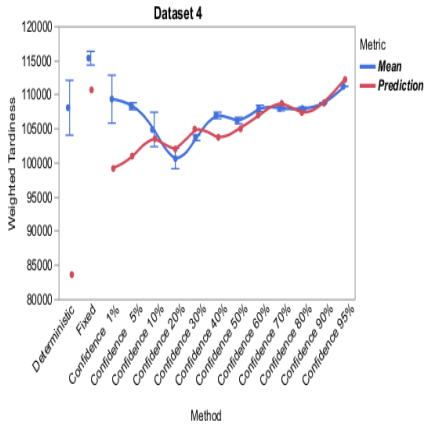


FIGURE 4.8 – Dataset with 525 textile pieces to weave. 10 hours timeout.

The CONFIDENCE model outperforms the FIXED model in both objective value and robustness. We can also notice that choosing a bad γ value can make or break the quality of a solution. The solutions for $\gamma = 1\%$ and $\gamma = 95\%$ are worse than the DETERMINISTIC model solution.

With the confidence intervals, we notice that the mean is usually quite precise with 100 runs of the simulation model. The lengthy events are rarer, therefore, increasing the number of simulations decreases the variability. Yet, using a higher value of γ also reduces the variability. With CONFIDENCE the prediction expects more tardiness, but the guarantee of not causing ripple effects makes it so the mean simulated weighted tardiness is better.

The DETERMINISTIC method has no stochastic optimization involved and completely ignores potential breakdowns. We would have thought that during the simulation, this method would most likely lag behind the planned schedule and finally execute a non-optimize schedule. However, it seems that in a context where resources are not scarce (dataset 1), repairing the plan as it executes is not a bad way to proceed. This pattern gradually vanishes with bigger instances.

On the harder datasets (datasets 2, 3, and 4), the objective value adopts a wavy form. Our main hypothesis is that varying γ does not only affect the objective value, it also affects the difficulty to solve the problem. So decreasing γ would normally decrease the objective value, but in some situations, the search heuristic gets trapped and the solver returns a worse solution once the timeout is reached. This is especially apparent with a timeout of 30 minutes.

The harder datasets (2, 3, and 4) have tighter schedules. A ripple effect from a breakdown can produce increasingly more lateness. We assume that the usefulness of the CONFIDENCE scales with the risk of the ripple effect. In Figures 4.4, 4.6, and 4.8, the curves are respectively centered around $\gamma = 50\%, 5\%, 20\%$. The red curve seems to show that $\gamma = 100\%$ cannot be achieved. This can be explained by the difficulty of satisfying the higher percent of a Poisson cdf function and the tendency of probability products to tend towards 0.

When we compare the different solutions graphs with the Table 4.1, we can notice the following. The more resource usage there is, the harder it is to solve the problem with a higher γ value. The instances also get harder as the number of tasks increases. While we don't include computation times, we noticed that in the dataset 1, the computation times were similar (less than a second of difference) to the deterministic model until 99% where it spiked. The computation times for the other datasets where inconclusive since no optimal solution was found.

The FIXED method often overestimates the breakdown times needed to create robustness and produces schedules where tasks are planned to be executed late. The FIXED method is more rigid compared to CONFIDENCE because it cannot allow to choose where the risk should be taken in order to minimize the weighted tardiness. More often than not, the FIXED approach gives results similar to the CONFIDENCE method with $\gamma = 0.5$, but slightly worse. In a context where resources are abundant, it would be interesting to compute the processing times by using a value shorted than the average breakdown time.

The CONFIDENCE method shows that the weighted tardiness computed by the solver is a good indicator of the weighted tardiness obtained in the simulation. It would be interesting to push further the evaluation with instances, not necessarily industrial, for which we can control the scarcity of the resources. A search heuristic adapted to stochastic optimization could also help with this type of problem and reduce the computation times.

4.9 Conclusion

We presented the CONFIDENCE constraint, a new global chance constraint that provides robust solutions in stochastic contexts. We presented a linear time filtering algorithm with explanations. We compared the CONFIDENCE constraint to a DETERMINISTIC and FIXED approach and determined that the new global chance constraint offers good prediction about the weighted tardiness obtained in the simulation. It is planned that our industrial partner sets the γ-parameter according to how much risk they are willing to take and readjusts it every week. The deterministic method is implemented and being deployed by our industrial partner, while the stochastic optimization is still under evaluation. More stochastic events need to be taken into account such as the arrival of new orders.

4.10 Bibliographie

[1] ALEXANDRE MERCIER-AUBIN, J. G., AND QUIMPER, C.-G. Leveraging constraint scheduling : A case study to the textile industry. to appear in proceeding of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2020.

[2] ALEXANDRE MERCIER-AUBIN, LUDWIG DUMETS, J. G., AND QUIMPER, C.-G. The confidence constraint : A step towards stochastic cp solvers. to appear in proceeding of 26th International Conference on Principles and Practice of Constraint Programming, 2020.

[3] Baptiste, P., and Le Pape, C. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints 5*, 1 (2000), 119—139.

[4] Baptiste, P., Pape, C. L., and Nuijten, W. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[5] Chakrabortty, R. K., Sarker, R. A., and Essam, D. L. Resource constrained project scheduling with uncertain activity durations. *Computers & Industrial Engineering 112* (2017), 537 – 550.

[6] Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. Chuffed, a lazy clause generation solver, 2018.

[7] Derrien, A., Petit, T., and Zampelli, S. A declarative paradigm for ro- bust cumulative scheduling. In *Principles and Practice of Constraint Programming - 20th International Conference (CP 2014)* (2014), pp. 298–306.

[8] Fahimi, H. *Efficient algorithms to solve scheduling problems with a variety of optimization criteria*. PhD thesis, Université Laval, 2016.

[9] Grimmett, G., and Welsh, D. *Probability : An Introduction*. Oxford University Press, 2014.

[10] Hebrard, E., and Walsh, T. Improved algorithm for finding (a,b)-super solutions. In *Principles and Practice of Constraint Programming - CP 2005* (Berlin, Heidelberg, 2005), P. van Beek, Ed., Springer Berlin Heidelberg, pp. 848–848.

[11] Kall, P., and Mayer, J. *Stochastic Linear Programming*. Springer US, 2011.

[12] Lombardi, M., and Milano, M. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In *Principles and Practice of Constraint Programming - CP 2009* (2009), pp. 569–583.

[13] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. Minizinc : Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming – CP 2007* (2007), pp. 529–543.

[14] Rendl, A., Tack, G., and Stuckey, P. J. Stochastic minizinc. In *Principles and Practice of Constraint Programming* (2014), pp. 636–645.

[15] Rossi, R., Tarim, S., Hnich, B., and Pestwich, S. A global chance-constraint for stochastic inventory systems under service level constraints. *Constraints 13*, 4 (Dec. 2008), 490–517.

[16] Schling, B. *The Boost C++ Libraries*. XML Press, 2011.

[17] Shishmarev, M., Mears, C., Tack, G., and Garcia de la Banda, M. Learning from learning solvers. In *Principles and Practice of Constraint Programming* (Cham, 2016), M. Rueher, Ed., Springer International Publishing, pp. 455–472.

[18] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. The minizinc challenge 2008–2013. *AI Magazine 35*, 2 (June 2014), 55–60.

[19] Walsh, T. Stochastic constraint programming. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-2002)* (2009), pp. 111–115.

[20] Zaayman, G., and Innamorato, A. The application of simio scheduling in industry 4.0. In *2017 Winter Simulation Conference (WSC)* (2017), pp. 4425–4434.

[21] Zghidi, I., Hnich, B., and Rebaï, A. Modeling uncertainties with chance constraints. *Constraints 23*, 2 (Apr. 2018), 196–209.

Chapitre 5

Résumé des contributions industrielles

Depuis la publication des articles, la compréhension du problème et le développement de nos approches de résolution ont continué d'évoluer. Nous présentons ces quelques changements et ces tableaux des résultats finaux. Dans cette section, nous présentons des changements effectués au système pour améliorer la performance de la planification, réduire la mémoire vive utilisée lors de l'ordonnancement, la réduction du temps de génération des fichiers pour le solveur et ajouter la modélisation de la fluctuation du nombre d'employés. Par la suite, nous analysons les nouveaux jeux de données reçus avec les modifications effectuées. Notons que cette technologie est présentement chez notre partenaire industriel et que l'implémentation des détails techniques est en cours. Cette contribution offre un grand bénéfice en permettant des solutions en temps de calcul raisonnables, alors que l'équipe de planification de notre partenaire industriel devait manuellement faire l'horaire grâce à quatre employés engagés à temps plein.

5.1 Regroupement des pièces de textile sans attacheur

Un des problèmes pouvant survenir avec la séparation en modèle de planification et en modèle d'ordonnancement est que le modèle de planification ne tient pas compte des temps de mise en course mineurs. Après l'article du chapitre 3, nous avons utilisé une approche, dans la planification, pour tenir compte d'une partie des temps de mise en course mineurs.

Il est parfois possible de mettre des tâches sur le même métier afin d'éviter l'intervention des attacheurs lors des temps de mise en course mineurs. Comme le modèle de planification ne tient pas compte les temps de mise en course mineurs, nous avons décidé de regrouper les tâches par type (couleur et motifs). Des blocs de tâches ayant un même type sont alors assignés à des métiers. Ceci a aussi pour effet de diminuer la variabilité des temps de mise en course mineurs. En effet, comme les pièces de textiles seront plus souvent les mêmes, il y aura moins de temps de mise en course mineurs qui auront des durées différentes. Le problème étant maintenant différent, la solution au problème l'est aussi.

5.2 Réduction de l'utilisation en mémoire vive

La consommation en mémoire vive étant aussi trop grande pour certains ordinateurs plus modestes (moins de 16 gigaoctets de mémoire vive), nous avons fait certaines modifications techniques ont permis de réduire la consommation. En supposant que les professions resteront toujours les mêmes, il est possible de les encoder directement dans le modèle d'ordonnancement. Cette manipulation a permis de déclarer individuellement certaines contraintes au lieu d'utiliser l'assignation de tableau par compréhension de MiniZinc. En choisissant manuellement les domaines initiaux des variables, la quantité de mémoire utilisée semble baisser. Nous supposons alors que MiniZinc encode dans un bitset les variables entières n'ayant pas un domaine défini préalablement.

5.3 Réduction du temps de génération

Lors de l'appel à MiniZinc, il y a un temps de génération de données sous un format compréhensible pour un solveur de contraintes. Cette étape doit être faite pour chaque jeu de données. Le temps de génération était relativement élevé (environ 3 minutes). Nous avons remarqué que la matrice de transition pour les temps de mise en course mineurs était clairsemée. Afin de réduire sa taille, nous avons encodé le tableau dans une liste des transitions ayant la forme suivante pour chaque entrée : $i, j, t_{i,j,p}$ où i et j sont des pièces de textiles, $t_{i,j,p}$ est le temps de mise en course mineur pour ces tâches et p est le type de profession. Cette liste est ensuite passée dans une contrainte TABLE où une nouvelle variable prend la valeur des transitions pour chaque pièce de textile et chaque profession. Grâce à cette modification, le temps de compilation est maintenant de moins d'une seconde.

5.4 Fluctuation du nombre d'employés

La fluctuation du nombre d'employés selon les différents quarts de travails n'était pas représentée. Nous rajoutons alors de fausses tâches pour réduire artificiellement la quantité d'employés à certains moments de la journée. Ces tâches n'occupent pas de métier, mais diminuent le nombre d'employés.

5.5 Nouveaux jeux de données

Nous avons reçu de nouveaux jeux de données au fil du temps. Les fonctions objectifs pour les méthodes GREEDY et CIRCUIT du premier article ont été réévaluées dans des bancs d'essai standards en incluant les nouveaux jeux de données et les nouvelles contraintes.

Dans le tableau 5.1, nous présentons les diverses fonctions objectifs en effectuant les mêmes bancs d'essai standards que ceux effectués dans le premier article. Nous présentons une comparaison des diverses fonctions objectifs obtenues pour ces jeux de données ainsi que les nouveaux. Ces fonctions objectifs sont accompagnées d'un pourcentage pour mettre en valeur l'amélioration ou la perte en

valeur de la fonction-objectif comparée à la valeur initiale de l'heuristique de choix de branchement GREEDY.

Comme la nature du problème a changé avec les nouvelles contraintes imposées, la solution du problème peut changer. Or, ça explique pourquoi la méthode GREEDY est rendue aussi bonne. Comme les pièces de tissu tissées sur un métier à tisser sont plus souvent les mêmes, il y a moins de temps de mise en course mineurs ayant des durées différentes. Comme il y a moins de différences dans les durées, il y a moins de gain en qualité de solution lorsque la taille du circuit est réduite au plus petit possible. Le gain causé par la diminution en temps de mise en course mineurs ne surpasse plus le gain d'ordonnancer les tâches par date limite de fin de tissage.

En effet, en analysant le tableau, CP+GREEDY semble être le meilleur choix dans tous les cas. Ceci nous mène à croire que la perte des clauses paresseuses causée par le redémarrage dans l'algorithme de recherche locale à grand voisinage nuit lorsque la variabilité est plus faible. Nous remarquons aussi que l'utilisation de la recherche locale à grand voisinage est pertinente avec CP+CIRCUIT. Comme le circuit généré par CP+CIRCUIT est moins bon, il est plus facile de trouver un meilleur circuit en faisant moins de modifications et donc cette technique apporte de meilleures solutions.

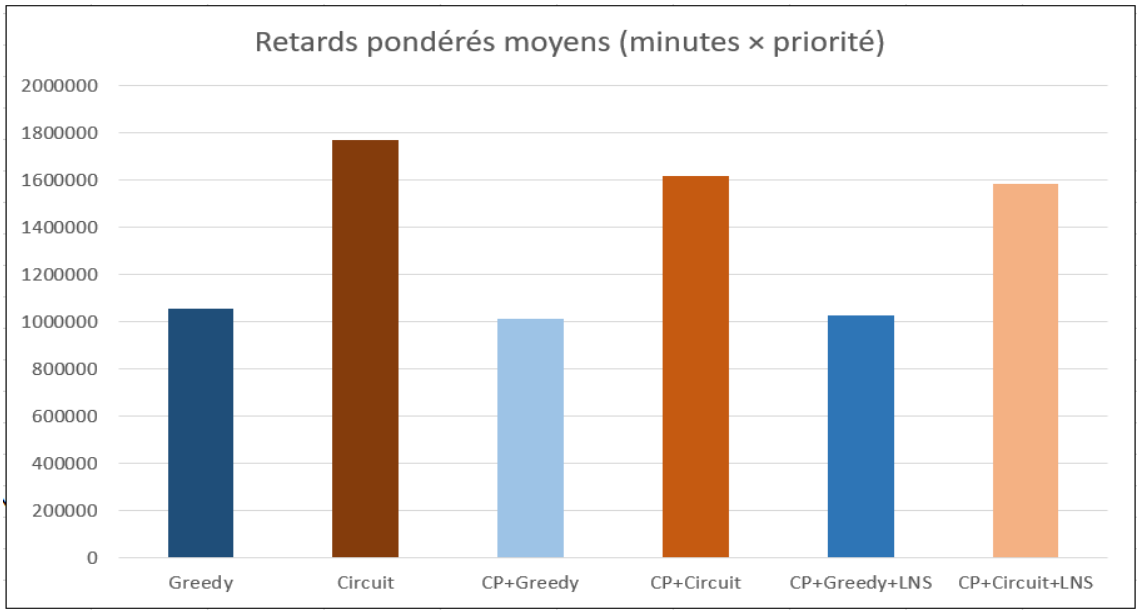


FIGURE 5.1 – Comparaison de la moyenne des retards pondérés (minutes × priorité) moyens pour chaque méthode. Cette figure est extraite de la dernière rangée de la table 5.1

Dans la figure 5.1, nous présentons un graphique résumé comparant les différentes méthodes. Une teinte rouge implique l'utilisation de CIRCUIT, tandis qu'une teinte bleue implique l'utilisation de GREEDY. En effet, plus la couleur est pâle pour une teinte donnée, plus la méthode est performante sur les jeux de données présentés dans le tableau 5.1. La méthode CP+GREEDY est la plus performante et est suivie de très près par celle avec lns.

Jeu de données	Méthodes											
	Greedy		Circuit		CP+ Greedy		CP+ Circuit		CP+ Greedy+ LNS		CP+ Circuit+ LNS	
	obj	%	obj	%	obj	%	obj	%	obj	%	obj	%
2020-04-08	767 306	100	935 613	121.93	**665 684**	86.756	813 590	106.03	703 016	91.62	819 802	106.84
2020-02-17	233 65	100	387 425	165.81	**222 445**	95.20	318 597	136.35	223 421	95.62	357 935	153.19
2020-02-19	258 106	100	376 052	145.70	**236 439**	91.61	303 020	117.40	252 466	97.82	300 961	116.60
2020-02-25	149 942	100	221 702	147.86	**82 824**	55.24	154 604	103.11	134 217	89.51	141 975	94.69
2020-03-04	153 273	100	393 091	256.47	**108 562**	70.83	237 982	155.27	140 743	91.83	244 437	159.48
2020-03-12	77 328	100	258 703	334.55	**74 180**	95.93	220 555	285.22	**74 180**	95.93	218 922	283.11
2020-03-23	862 214	100	1 269 546	147.24	**778 071**	90.24	1 085 150	125.86	837 353	97.12	1 023 200	118.67
2020-03-24	380 504	100	714 434	187.76	**355 200**	93.35	629 544	165.45	360 342	94.70	537 888	141.36
2020-03-25	361 586	100	655 126	181.18	**331 674**	91.73	472 034	130.55	346 043	95.70	395 553	109.39
2019-08-08	2 219 951	100	3 836 764	172.83	**2 158 428**	97.23	3 697 443	166.56	2 166 705	97.60	3 749 023	168.88
2019-08-12	2 833 792	100	4 897 233	172.82	2 785 927	98.31	4 701 019	165.89	**2 783 187**	98.21	4 705 090	166.04
2019-09-02	2 199 734	100	3 531 169	160.53	**2 160 868**	98.23	3 279 600	149.09	2 164 046	98.38	3 388 958	154.06
2019-09-16	2 619 230	100	4 011 752	153.17	**2 565 858**	97.96	3 840 343	146.62	2 568 623	98.07	3 418 510	130.52
2019-10-16	1 274 526	100	2 200 437	172.65	**1 250 566**	98.12	2 106 279	165.26	1 267 709	99.47	2 003 027	157.16
2019-10-24	1 092 954	100	2 250 791	205.94	**1 026 592**	93.93	1 933 471	176.90	1 037 171	94.90	2 005 357	183.48
2019-09-16	1 421 776	100	2 415 201	169.87	**1 356 899**	95.44	2 072 545	145.77	1 394 258	98.07	2 034 941	143.13
Moyenne	1 056 617	100	1 772 190	167.72	1 010 014	95.59	1 616 611	153.00	1 028 343	97.32	1 584 099	149.92

Table 5.1 – Bancs d'essai standards renouvelés des retards pondérés (minutes $\times$ priorité) pour chaque méthode et jeu de données. Chaque valeur de la fonction-objectif est accompagnée d'un pourcentage de comparaison (méthode / Greedy).

Conclusion

Dans ce mémoire, nous avons présenté un système de résolution pour trouver des solutions au problème apporté par notre partenaire industriel. Ce système comporte trois étapes : la planification, l'ordonnancement et la simulation. Notre partenaire industriel a maintenant à sa disposition des outils d'aide à la décision.

Nous avons repris le modèle de planification de Philippe Marier que nous avons adapté aux besoins courants de notre partenaire industriel. Nous avons ensuite présenté une première version déterministe du modèle d'ordonnancement pour résoudre le problème. Nous montrons que le modèle d'ordonnancement apporte des résultats compétitifs en termes d'ordonnancement. Ce modèle a mené à la publication d'un article scientifique dans la conférence CPAIOR2020.

Une fois le modèle déterministe conçu, nous avons défini une nouvelle contrainte de chance en programmation par contrainte pour filtrer les solutions risquées. Cette contrainte permet de chercher une garantie de robustesse lors de l'application des solutions. Bien que cette nouvelle contrainte a été appliquée au problème de notre partenaire industrielle, sa portée va beaucoup plus loin. La contrainte peut modéliser les différents problèmes de chance ayant des distributions définies. Nous assurons la cohérence de bornes en temps linéaire. Nous entendons rendre publics le code source de la contrainte et son implémentation dans le solveur utilisé. Cette contrainte a mené à la publication d'un article scientifique dans la conférence CP2020.

Nous avons alors la chance d'effectuer des contributions pratiques et théoriques significatives. Ces avancées techniques mènent à deux contributions dans des conférences reconnues en programmation par contraintes.

Cette technologie est présentement chez notre partenaire industriel et l'implémentation des détails techniques est en cours. Cette contribution offre un grand bénéfice en permettant des solutions en temps de calcul raisonnables, alors que l'équipe de planification de notre partenaire industriel devait manuellement faire l'horaire grâce à quatre employés engagés à temps plein.

Nous ouvrons alors les portes à de nouvelles avancées en programmation par contraintes pour ce qui est de la gestion d'évènements stochastique et la robustesse de solutions.

Bibliographie

5.6 Chapitre 2

[1] Baofu, P. *The Future of Complexity*. WORLD SCIENTIFIC, 2007.

[2] Beldiceanu, N., and Contejean, E. Introducing global constraints in chip. *Mathematical and Computer Modelling 20*, 12 (1994), 97 – 123.

[3] Bigras, L.-P., Gamache, M., and Savard, G. The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization 5*, 4 (2008), 685 – 699.

[4] Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. Chuffed, a lazy clause generation solver, 2018.

[5] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1971), STOC '71, Association for Computing Machinery, p. 151–158.

[6] Cook, S. A. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing* (New York, NY, USA, 1971), STOC '71, Association for Computing Machinery, p. 151–158.

[7] Cook, W. *In Pursuit of the Traveling Salesman : Mathematics at the Limits of Computation*. Princeton University Press. Princeton University Press, 2011.

[8] Cormen, T., Leiserson, C., and Rivest, R. *Introduction à l'algorithmique*. Dunod, Paris, France, 1994.

[9] Dantzig, G. B. Linear programming under uncertainty. *Management Science 1*, 3-4 (1955), 197–206.

[10] Dershowitz, N., Hanna, Z., and Nadel, A. A clause-based heuristic for sat solvers. In *Theory and Applications of Satisfiability Testing* (Berlin, Heidelberg, 2005), F. Bacchus and T. Walsh, Eds., Springer Berlin Heidelberg, pp. 46–60.

[11] Ginsberg, M. L., Parkes, A. J., and Roy, A. Supermodels and robustness. In *In Proceedings AAAI'98* (1998), p. 334–339.

[12] Godard, D., Laborie, P., and Nuijten, W. Randomized large neighborhood search for cumulative scheduling. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling* (01 2005), pp. 81 – 89.

[13] Golomb, S. W., and Baumert, L. D. Backtrack programming. *J. ACM 12*, 4 (Oct. 1965), 516–524.

[14] Guddeti, V. P., and Choueiry, B. Y. Characterization of a new restart strategy for randomized backtrack search. In *Recent Advances in Constraints* (Berlin, Heidelberg, 2005), B. V. Faltings, A. Petcu, F. Fages, and F. Rossi, Eds., Springer Berlin Heidelberg, pp. 56–70.

[15] Harvey, W. D. *Nonsystematic Backtracking Search*. PhD thesis, Stanford University, 1995.

[16] Hebrard, E., Hnich, B., and Walsh, T. Super solutions in constraint programming. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Berlin, Heidelberg, 2004), J.-C. Régin and M. Rueher, Eds., Springer Berlin Heidelberg, pp. 157–172.

[17] Hebrard, E., and Walsh, T. Improved algorithm for finding (a,b)-super solutions. In *Principles and Practice of Constraint Programming - CP 2005* (Berlin, Heidelberg, 2005), P. van Beek, Ed., Springer Berlin Heidelberg, pp. 848–848.

[18] Hoc, J.-M., Mebarki, N., and Cegarra, J. L'assistance à l'opérateur humain pour l'ordonnancement dans les ateliers manufacturiers. *Le travail humain Vol. 67*, 2 (2004), 181–208.

[19] Hsu, H.-M., Hsiung, Y., Chen, Y.-Z., and Wu, M.-C. A ga methodology for the scheduling of yarn-dyed textile production. *Expert Systems with Applications 36*, 10 (2009), 12095 – 12103.

[20] Kaya, L. G., and Hooker, J. N. A filter for the circuit constraint. In *Principles and Practice of Constraint Programming - CP 2006* (Berlin, Heidelberg, 2006), F. Benhamou, Ed., Springer Berlin Heidelberg, pp. 706–710.

[21] Ku, W.-Y., and Beck, J. C. Mixed integer programming models for job shop scheduling : A computational analysi. *Computers & Operations Research 73* (2016), 165 – 173.

[22] Law, A. M. *Simulation Modeling & Analysis*, 4 ed. McGraw-Hill, New York, NY, USA, 2007.

[23] Lecoutre, C. *Restarts and Nogood Recording*. John Wiley & Sons, Ltd, 2010, ch. 10, pp. 431–458.

[24] Lecoutre, C., and Szymanek, R. Generalized arc consistency for positive table constraints. In *Principles and Practice of Constraint Programming - CP 2006* (Berlin, Heidelberg, 2006), F. Benhamou, Ed., Springer Berlin Heidelberg, pp. 284–298.

[25] Lopez-Ortiz, A., Quimper, C.-G., Tromp, J., and Van Beek, P. A fast and simple algorithm for bounds consistency of the all different constraint. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence* (San Francisco, CA, USA, 2003), IJCAI'03, Morgan Kaufmann Publishers Inc., p. 245–250.

[26] Luby, M., Sinclair, A., and Zuckerman, D. Optimal speedup of las vegas algorithms. *Information Processing Letters 47*, 4 (1993), 173 – 180.

[27] Lutz, R. *Adaptive Large Neighborhood Search*. Universität Ulm. Fakultät für Ingenieurwissenschaften und Informatik, 2014.

[28] Mairy, J.-B., Deville, Y., and Lecoutre, C. The smart table constraint. In *Integration of AI and OR Techniques in Constraint Programming* (Cham, 2015), L. Michel, Ed., Springer International Publishing, pp. 271–287.

[29] Michel, L., and Van Hentenryck, P. Activity-based search for black-box constraint programming solvers. In *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimzation Problems* (2012), pp. 228–243.

[30] Ohrimenko, O., Stuckey, P. J., and Codish, M. Propagation = lazy clause generation. In *Principles and Practice of Constraint Programming – CP 2007* (Berlin, Heidelberg, 2007), C. Bessière, Ed., Springer Berlin Heidelberg, pp. 544–558.

[31] Ouellet, P., and Quimper, C.-G. Time-table extended-edge-finding for the cumulative constraint. In *Principles and Practice of Constraint Programming* (Berlin, Heidelberg, 2013), C. Schulte, Ed., Springer Berlin Heidelberg, pp. 562–577.

[32] Refalo, P. Impact-based search strategies for constraint programming. In *Principles and Practice of Constraint Programming – CP 2004* (Berlin, Heidelberg, 2004), M. Wallace, Ed., Springer Berlin Heidelberg, pp. 557–571.

[33] Rendl, A., Tack, G., and Stuckey, P. J. Stochastic minizinc. In *Principles and Practice of Constraint Programming* (2014), pp. 636–645.

[34] Rockafellar, R. T., and Wets, R. J.-B. Scenarios and policy aggregation in optimization under uncertainty. *Mathematics of Operations Research 16*, 1 (1991), 119–147.

[35] Rossi, F., Beek, P. v., and Walsh, T. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., USA, 2006.

[36] Rossi, R., Tarim, S. A., Hnich, B., and Prestwich, S. A global chance-constraint for stochastic inventory systems under service level constraints. *Constraints 13* (12 2008), 490–517.

[37] Régin, J.-C. Filtering algorithm for constraints of difference in csps. In *Proceedings of the Eleventh National Conference on Artificial Intelligence* (07 1994), vol. 1, AAAI Press, p. 362 – 367.

[38] Schutt, A., Feydy, T., Stuckey, P., and Wallace, M. Explaining the cumulative propagator. *Constraints 16* (07 2011), 250–282.

[39] Serafini, P. Scheduling jobs on several machines with the job splitting property. *Operations Research 44*, 4 (1996), 617–628.

[40] Shaw, P. A new local search algorithm providing high quality solutions to vehicle routing problems, 1997.

[41] Silva, C. Combining ad hoc decision-making behaviour with formal planning and scheduling rules : a case study in the synthetic fibre production industry. *Production Planning & Control 20*, 7 (2009), 636–648.

[42] Stuckey, P. J. Lazy clause generation : Combining the power of sat and cp (and mip ?) solving. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* (Berlin, Heidelberg, 2010), A. Lodi, M. Milano, and P. Toth, Eds., Springer Berlin Heidelberg, pp. 5–9.

[43] Tarim, S. A., Manandhar, S., and Walsh, T. Stochastic constraint programming : A scenario based approach. *Journal of Constraints 11* (01 2006), 53–80.

[44] Tucci, M., and Rinaldi, R. From theory to application : Tabu search in textile production scheduling. *Production Planning & Control 10*, 4 (1999), 365–374.

[45] Ullman, J. Np-complete scheduling problems. *Journal of Computer and System sciences 10*, 3 (1975), 384–393.

[46] Wang, J., Pan, R., Gao, W., and Wang, H. An automatic scheduling method for weaving enterprises based on genetic algorithm. *The Journal of The Textile Institute 106* (01 2015), 1–11.

[47] Wang, L., Huang, H., and Ke, H. Chance-constrained model for rcpsp with uncertain durations. *Journal of Uncertainty Analysis and Applications 3* (12 2015).

[48] Zaayman, G., and Innamorato, A. The application of simio scheduling in industry 4.0. In *2017 Winter Simulation Conference (WSC)* (2017), pp. 4425–4434.

[49] Zghidi, I., Hnich, B., and Rebaï, A. Modeling uncertainties with chance constraints. *Constraints 23*, 2 (Apr. 2018), 196–209.

5.7 Chapitre 3

[1] Alexandre Mercier-Aubin, J. G., and Quimper, C.-G. Leveraging constraint scheduling : A case study to the textile industry. to appear in proceeding of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2020.

[2] Chakrabortty, R. K., Sarker, R. A., and Essam, D. L. Resource constrained project scheduling with uncertain activity durations. *Computers & Industrial Engineering 112* (2017), 537 – 550.

[3] Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. Chuffed, a lazy clause generation solver, 2018.

[4] Cook, W. *In Pursuit of the Traveling Salesman : Mathematics at the Limits of Computation*. Princeton University Press. Princeton University Press, 2011.

[5] Costa, A., Cappadonna, F. A., and Fichera, S. A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology 69* (2013), 2799—2817.

[6] Danna, E., and Perron, L. Structured vs. unstructured large neighborhood search : A case study on job-shop scheduling problems with earliness and tardiness costs. In *Principles and Practice of Constraint Programming – CP 2003* (Berlin, Heidelberg, 2003), F. Rossi, Ed., Springer Berlin Heidelberg, pp. 817–821.

[7] Focacci, F., Laborie, P., and Nuijten, W. Solving scheduling problems with setup times and alternative resources. In *Proceedings of the Fifth International Conference on Artificial Intelligence Planning Systems* (2000), pp. 92–101.

[8] Hermann, M., Pentek, T., and Otto, B. Design principles for industrie 4.0 scenarios. In *49th Hawaii International Conference on System Sciences (HICSS)* (2016), pp. 3928-–3937.

[9] Kagermann, H., Helbig, J., Hellinger, A., and Wahlster, W. Recommendations for implementing the strategic initiative industrie 4.0 : Securing the future of german manufacturing industry ; final report of the industrie 4.0 working group. Tech. rep., Forschungsunion, 2013.

[10] Kajgard, E. *Route Optimisation for Winter Road Maintenance using Constraint Modelling*. PhD thesis, Uppsala Universitet, 2015.

[11] Lauriere, J.-L. A language and a program for stating and solving combinatorial problems. *Artificial intelligence 10*, 1 (1978), 29–127.

[12] López-Ibáñez, M., Blum, C., Ohlmann, J. W., and Thomas, B. W. The travelling salesman problem with time windows : Adapting algorithms from travel-time to makespan optimization. *Applied Soft Computing 13*, 9 (2013), 3806–3815.

[13] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. Minizinc : Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming – CP 2007* (2007), pp. 529–543.

[14] Psaraftis, H. N. k-interchange procedures for local search in a precedence-constrained routing problem. *European Journal of Operational Research 13*, 4 (1983), 391–402.

[15] Savelsbergh, M. W. P. Local search in routing problems with time windows. *Annals of Operations Research 4*, 1 (Dec 1985), 285–305.

[16] Schutt, A., Feydy, T., Stuckey, P. J., and Wallace, M. G. Why cumulative decomposition is not as bad as it sounds. In *Principles and Practice of Constraint Programming - CP 2009* (Berlin, Heidelberg, 2009), I. P. Gent, Ed., Springer Berlin Heidelberg, pp. 746–761.

[17] Shishmarev, M., Mears, C., Tack, G., and Garcia de la Banda, M. Learning from learning solvers. In *Principles and Practice of Constraint Programming* (Cham, 2016), M. Rueher, Ed., Springer International Publishing, pp. 455–472.

[18] Tran, T. T., and Beck, J. C. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. In *Proceedings of the 20th European Conference on Artificial Intelligence ECAI'12* (2012), pp. 774–779.

5.8 Chapitre 4

[1] Alexandre Mercier-Aubin, J. G., and Quimper, C.-G. Leveraging constraint scheduling : A case study to the textile industry. to appear in proceeding of the 17th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research, 2020.

[2] Alexandre Mercier-Aubin, Ludwig Dumets, J. G., and Quimper, C.-G. The confidence constraint : A step towards stochastic cp solvers. to appear in proceeding of 26th International Conference on Principles and Practice of Constraint Programming, 2020.

[3] Baptiste, P., and Le Pape, C. Constraint propagation and decomposition techniques for highly disjunctive and highly cumulative project scheduling problems. *Constraints 5*, 1 (2000), 119—139.

[4] Baptiste, P., Pape, C. L., and Nuijten, W. *Constraint-Based Scheduling*. Kluwer Academic Publishers, 2001.

[5] Chakrabortty, R. K., Sarker, R. A., and Essam, D. L. Resource constrained project scheduling with uncertain activity durations. *Computers & Industrial Engineering 112* (2017), 537 – 550.

[6] Chu, G., Stuckey, P. J., Schutt, A., Ehlers, T., Gange, G., and Francis, K. Chuffed, a lazy clause generation solver, 2018.

[7] Derrien, A., Petit, T., and Zampelli, S. A declarative paradigm for ro- bust cumulative scheduling. In *Principles and Practice of Constraint Programming - 20th International Conference (CP 2014)* (2014), pp. 298–306.

[8] Fahimi, H. *Efficient algorithms to solve scheduling problems with a variety of optimization criteria*. PhD thesis, Université Laval, 2016.

[9] Grimmett, G., and Welsh, D. *Probability : An Introduction*. Oxford University Press, 2014.

[10] Hebrard, E., and Walsh, T. Improved algorithm for finding (a,b)-super solutions. In *Principles and Practice of Constraint Programming - CP 2005* (Berlin, Heidelberg, 2005), P. van Beek, Ed., Springer Berlin Heidelberg, pp. 848–848.

[11] Kall, P., and Mayer, J. *Stochastic Linear Programming*. Springer US, 2011.

[12] Lombardi, M., and Milano, M. A precedence constraint posting approach for the rcpsp with time lags and variable durations. In *Principles and Practice of Constraint Programming - CP 2009* (2009), pp. 569–583.

[13] Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. Minizinc : Towards a standard cp modelling language. In *Principles and Practice of Constraint Programming – CP 2007* (2007), pp. 529–543.

[14] Rendl, A., Tack, G., and Stuckey, P. J. Stochastic minizinc. In *Principles and Practice of Constraint Programming* (2014), pp. 636–645.

[15] Rossi, R., Tarim, S., Hnich, B., and Pestwich, S. A global chance-constraint for stochastic inventory systems under service level constraints. *Constraints 13*, 4 (Dec. 2008), 490–517.

[16] Schling, B. *The Boost C++ Libraries*. XML Press, 2011.

[17] Shishmarev, M., Mears, C., Tack, G., and Garcia de la Banda, M. Learning from learning solvers. In *Principles and Practice of Constraint Programming* (Cham, 2016), M. Rueher, Ed., Springer International Publishing, pp. 455–472.

[18] Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. The minizinc challenge 2008–2013. *AI Magazine 35*, 2 (June 2014), 55–60.

[19] Walsh, T. Stochastic constraint programming. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI-2002)* (2009), pp. 111–115.

[20] Zaayman, G., and Innamorato, A. The application of simio scheduling in industry 4.0. In *2017 Winter Simulation Conference (WSC)* (2017), pp. 4425–4434.

[21] Zghidi, I., Hnich, B., and Rebaï, A. Modeling uncertainties with chance constraints. *Constraints 23*, 2 (Apr. 2018), 196–209.

Annexe A

Modèle de planification

A.1 Auteurs

Ce modèle fut initialement écrit par Philippe Marier. Le modèle fut ensuite adapté aux besoins changeant du partenaire industriel et réécrit en MiniZinc par Alexandre Mercier-Aubin.

A.2 Notations

Variable	Domain	Description
Ls_m	$[0, H]$	Temps de mise en course majeure sur un métier m
P_m	$[0, H]$	Somme des temps de tissage des tâches sur le métier m
Rd_r	$[0, H]$	Somme des temps d'ouvrage pour les employés
$S_{c,m}$	$[0, 1]$	Tableau où les variables prennent la valeur 1 lorsque le changement vers la configuration c est effectuée sur le métier m
$A_{j,m}$	$[0, 1]$	Tableau où les variables prennent la valeur 1 lorsqu'une tâche j est affectée au métier m
$c^{\text{final}}_{m,c}$	$[0, 1]$	Tableau où les variables prennent la valeur 1 lorsque la configuration c est la configuration finale du métier m

TABLE A.1 – Variables et domaines du modèle de planification

$\mathcal{I}$: Ensemble des pièces de textile à tisser
L : Ensemble des métiers à tisser
C : Ensemble des configurations
R : Ensemble des employés
H : Horizon de planification
Lc : Capacité en minutes d'un métier
Rc : Capacité en minutes d'un employé
Rq : Quantité d'employés
Cp : Tableau des configurations possible pour chaque métier. Lorsque la case contient la valeur 1, alors la configuation est possible pour ce métier.
O : Temps de tissage d'une pièce selon le métier à tisser.
c^{init} : Configuration initiale d'un métier à tisser
Jl : Tableau de compatibilité des pièces de textile et des métiers à tisser. Lorsque la case contient la valeur 1, alors la pièce de textile peut être tissée sur ce métier.
Jc : Tableau de compatibilité des pièces de textiles et des configurations. Lorsque la case contient la valeur 1, alors la configuation est possible pour cette pièce de textile.
M : Temps de mise en course majeure selon le métier et la configuration finale.
Rm : Temps de mise en course majeure pour l'employé effectuant le temps de mise en course majeure selon le métier à tisser et la configuration.

TABLE A.2 – Paramètres du modèle de planification

A.3 Le modèle

$$\text{Minimiser} \qquad \sum_{\text{m}\in\text{L}} \text{Ls}_\text{m} + \text{P}_\text{m} + \sum_{\text{r}\in\text{R}} \text{Rd}_\text{r} \tag{A.1}$$

$$Lsm = \sum_{c\in C} S_{c,m} * M_{c,m} \qquad \forall m \in L \tag{A.2}$$

$$P_m = \sum_{jin\mathcal{I}} A_{j,m} * O_{j,m} \qquad \forall m \in L \tag{A.3}$$

$$Rd_r = \sum_{c\in C, m\in L} S_{c,m} * Rm_{r,m,c} \qquad \forall r \in R \tag{A.4}$$

$$Jl_{j,m} \geq A_{j,m} \qquad \forall j \in \mathcal{I}, m \in L \tag{A.5}$$

$$1 = \sum_{m\in L} A_{j,m} \qquad \forall j \in \mathcal{I} \tag{A.6}$$

$$Cp_{c,m} \geq c_{m,c}^{\text{final}} \qquad \forall m \in L, c \in C \tag{A.7}$$

$$1 = \sum_{c\in C} c_{m,c}^{\text{final}} \qquad \forall m \in L \tag{A.8}$$

$$A_{j,m} \leq \sum_{c\in C} Jc_{j,c} * (c_{m,c}^{\text{final}} + c_{m,c}^{\text{init}}) \qquad \forall j \in \mathcal{I}, m \in L \tag{A.9}$$

$$1 - c_{m,c}^{\text{init}} \geq S_{c,m} \qquad \forall c \in C, m \in L \tag{A.10}$$

$$Cp_{c,m} \geq S_{c,m} \qquad \forall c \in C, m \in L \tag{A.11}$$

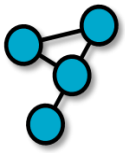
$$c_{m,c}^{\text{final}} - c_{m,c}^{\text{init}} \leq S_{c,m} \qquad \forall c \in C, m \in L \tag{A.12}$$

$$Ls_m + P_m \leq Lc_m \qquad \forall m \in L \tag{A.13}$$

$$Rc_r * Rq_r \geq Rd_r \qquad \forall r \in R \tag{A.14}$$

Annexe B

Document de référence pour notre partenaire industriel



CONSORTIUM DE RECHERCHE EN INGÉNIERIE DES SYSTÈMES INDUSTRIELS 4.0

B.1 Auteurs

Ce document a été rédigé initialement par Alexandre Mercier-Aubin. Il a ensuite été édité par Titouan Gaboriau. La collaboration entre ces deux auteurs a mené à la version finale du document de référence.

B.2 Architecture du projet

Le répertoire de projet se compose de trois sous-répertoires : les dossiers **ExemplesDatasets** (facultatif, les datasets pouvant se trouver n'importe où), **Codes** et **Doc**. On retrouve dans le dossier **Codes** les modèles d'optimisation MiniZinc, les scripts Python, les scripts Batch, et enfin le dossier **ThumbHeuristicGenerator** dans des dossiers séparés. Ce dernier un programme exécutable qui emploie une heuristique pour générer une solution de base qui sera perfectionnée par le module d'ordonnancement.

B.3 Installations préalables

Il est nécessaire pour l'exécution du programme d'installer :

— MiniZinc 2.4.3 (https ://www.minizinc.org/ide/)

— Python 3.8.2 (https ://www.python.org/downloads/release/python-382/)

Pour ce qui est des variables d'environnement PATH :

— Pour MiniZinc :

  — Dans Rechercher, lancez une recherche et sélectionnez : Système (Panneau de configuration)

  — Cliquez sur le lien **Paramètres système avancés**.

  — Cliquez sur **Variables d'environnement**. Dans la section **Variables système**, recherchez la variable d'environnement PATH et sélectionnez-la. Cliquez sur **Modifier**. Si la variable d'environnement PATH n'existe pas, cliquez sur Nouvelle.

  — Dans la fenêtre **Modifier la variable système** (ou **Nouvelle variable système**), indiquez la valeur de la variable d'environnement PATH (ici, ajouter le dossier d'installation par ex : C :\ProgramFiles\MiniZinc)

  — Cliquez sur **OK**. Fermez toutes les fenêtres restantes en cliquant sur **OK**.

  — Pour Python, il suffit de cocher la case correspondante lors de l'installation

— Si Python est déjà installé, procéder comme pour MiniZinc avec le chemin d'installation approprié.

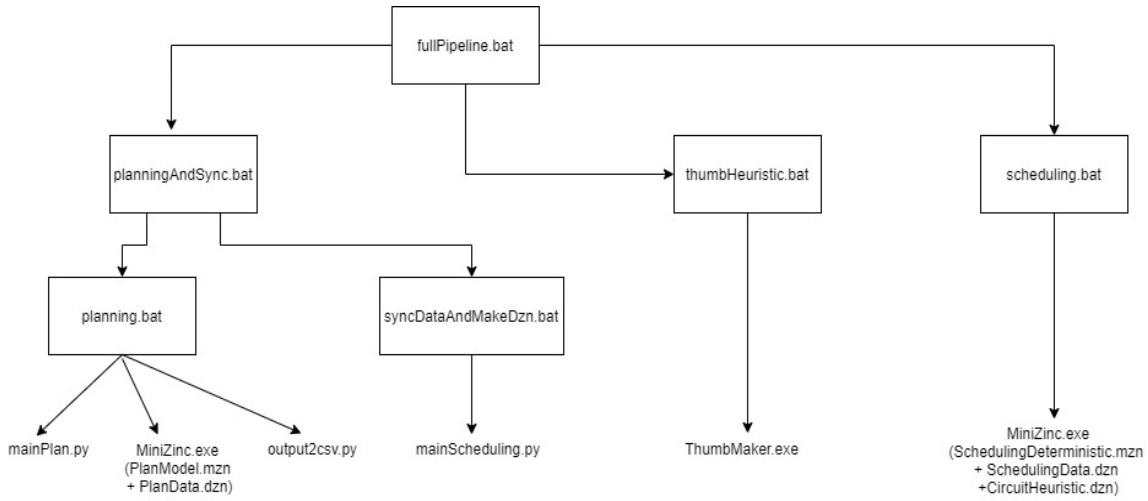
(par ex : C :\AppData\Roaming\Python\Python38\Scripts)

NB : Pour vérifier qu'une installation est complète, il est possible d'utiliser un terminal de commandes et de taper « python » ou « minizinc ».

B.4 Scripts Batch

Schéma explicatif de l'utilisation des différents fichiers

Un appel à **FullPipeline.bat** permet de réaliser de manière intégrée la planification et l'ordonnancement. Hormis à des fins de tests ou de débogage, **fullPipeline.bat** est le seul script qui devrait être appelé par l'utilisateur.



B.4.1 Pipeline complet (fullPipeline.bat)

Pour lancer le pipeline au complet (planification puis ordonnancement), il faut disposer d'un dataset complet. Tous les fichiers définissant ce dataset doivent se trouver dans un seul et même dossier.

Avant d'appeler le script **fullPipeline.bat**, il faut s'assurer que le dossier de travail du système d'exploitation est le dossier **Codes\WindowsBatch**.

Puis on peut lancer le script **fullPipeline.bat** en lui passant les paramètres appropriés.

Par exemple :

fullPipeline.bat 600000 0 0 C :\Documents_simplifie\ExemplesDatasets\20200325_8h00_COMPLET\240

Les paramètres étant :

1. Temps maximum permis pour l'exécution du modèle d'ordonnancement (en millisecondes)
2. Mode de planification/ordonnancement.

   Cela doit être **FilterFromCompany** pour utiliser uniquement les jobs que l'équipe de planification ont choisi de planifier (planificationParCompany.csv, voir page 9) ou autre pour utiliser tous les jobs du jeu de données.

3. Permet de choisir la liberté du modèle.

   FromCompany force les jobs à respecter l'ordre choisi par l'équipe de notre partenaire industriel dans la planification (planificationParCompany.csv, voir page 9). **DifferentBeam** n'exécute pas l'algorithme regroupant les tâches ayant la même ensouple sur le même métier. Une autre entrée exécute normalement le système.

4. Chemin complet du répertoire contenant le jeu de données choisi (penser à rajouter un « \ » à la fin)

   Attention, le chemin vers ce répertoire ne doit pas contenir d'espace pour un bon fonctionnement du programme.

   Les différents fichiers devant se trouver dans ce répertoire sont décrits dans la section IV, à partir de la page 6. On peut également se référer à des dossiers contenant d'anciens datasets à titre de référence.
5. Horizon de planification et d'ordonnancement (en heures)

Par la suite, **fullPipeline.bat** se chargera d'appeler les autres scripts, lesquels sont décrits ci-dessous à titre indicatif.

B.4.2 Phase de planification

Les conditions nécessaires au bon fonctionnement de la phase de planification sont identiques à celles du pipeline global, les paramètres sont également les mêmes.

Cette phase est réalisée par le script **planning.bat**.

Par exemple :

planning.bat 0 0 C :\Company_simplifie\ExemplesDatasets\20200325_8h00_COMPLET\240

Ce script va prendre les fichiers csv d'entrée, appeler le script mainPlan.py pour créer un fichier de données PlanData.dzn, puis appeler Minizinc pour exécuter le modèle PlanModel.dzn. Le script va enfin convertir les résultats renvoyés par Minizinc en fichiers csv, consultables dans le dossier du dataset.

B.4.3 Phase de préparation de l'ordonnancement

Cette étape est nécessaire pour faire le lien entre la planification et l'ordonnancement.

Le script syncDataAndMakeDzn prépare les fichiers résultats de la planification à l'utilisation par le modèle d'ordonnancement, avec comme paramètres :

1. Mode de planification/ordonnancement.

   1) Cela doit être FilterFromCompany pour utiliser uniquement les jobs que l'équipe de planification a choisi de planifier ou autre pour utiliser tous les jobs du jeu de données.

   2) Chemin complet du répertoire contenant le jeu de données choisi (penser à rajouter un « \ » à la fin)

3) Horizon de planification et d'ordonnancement

Il est possible d'effectuer la planification et cette étape de transition en une seule commande, grâce à planningAndSync.bat, avec les 4 mêmes paramètres qu'une planification simple.

Par exemple :

planningAndSync.bat 0 0 C :\Company_simplifie\ExemplesDatasets\20200325_8h00_COMPLET\240

B.4.4 Génération de l'ordonnancement initial

L'ordre initial des jobs est généré par une heuristique, grâce au script **thumbHeuristic.bat**. Celui-ci prend comme paramètres :

1. Chemin complet du répertoire contenant le jeu de données choisi (penser à rajouter un « \ » à la fin)
2. Horizon de planification souhaité (en heures)

5. Ordonnancement

Attention, il est nécessaire d'avoir effectué les étapes de planification (3) et de transition (4) pour lancer l'ordonnancement. Il faut également avoir un dataset complet (peu importe l'emplacement), puis lancer le script **scheduling.bat**, depuis le dossier **Codes\WindowsBatch** avec quelques paramètres.

Par exemple :

scheduling.bat 600000 C :\Company_simplifie\ExemplesDatasets\20200325_8h00_COMPLET

Les paramètres sont :

1. Temps maximum permis pour le modèle d'ordonnancement (en millisecondes)
2. Chemin complet du répertoire contenant le jeu de données choisi (penser à rajouter un « \ » à la fin)

Ce script va appeler le solveur Chuffed de MiniZinc, et sortir les résultats de l'ordonnancement dans le dossier du dataset, sous le nom resultScheduling.txt.

B.4.5 Nettoyage des datasets

Si avant de lancer un test sur un dataset, on souhaite vider celui-ci de ses fichiers obtenus précédemment, pour ne conserver que les csv d'entrée, il est possible de lancer le script cleanPipeline.bat avec comme paramètre unique le chemin complet vers le dataset.

B.5 Contenu des fichiers présents dans le répertoire de dataset

B.5.1 Fichiers CSV présents en entrée

Communs entre planification et ordonnancement

Compatibilité des configurations (Compatibilite.csv) Ce fichier indique, pour chaque tâche (job), les configurations permettant l'exécution de la tâche.

Job	Configuration
965385	2B2828R25
965393	2B2828R25
965686	2B2828R25
966005	1B8712R20

TABLE B.1 – Compatibilité des configurations

Capacité des métiers (PosteDeTravailCapacite.csv) Ce fichier indique la capacité de chaque métier en nombre d'heures encore disponibles sur l'horizon de temps considéré.

PosteDeTravail	Capacite
101	400
102	500
103	300

TABLE B.2 – Capacité des métiers

Temps d'opération des jobs (TempsOperation.csv). Ce fichier indique le temps d'opération requis, en nombre d'heures, pour l'exécution des tâches (job) pour chaque métier sur lesquels la tâche pourrait être exécutée. Ce fichier inclut aussi les taux de casses (bris moyen/heure) en chaine et en trame.

Job	PosteDeTravail	Duree	TauxCassesChaine	TauxCassesTrame
1	102	12	0.513	2.357
1	109	8	0.513	2.357
2	102	12	0.513	2.357
2	108	12	0.29412	1.40868
3	102	8	0.29412	1.40868
3	117	8	0.4446	2.1294
4	109	12.5	0.4446	2.1294

TABLE B.3 – Temps d'opération des jobs

Temps des setup majeurs (TempsSetupMachine.csv) Ce fichier indique, pour chaque configuration possible sur chaque métier, le temps de setup majeur requis pour passer de la configuration actuelle à une nouvelle configuration du métier. Pour chaque configuration-métier, on indique également s'il s'agit de la configuration actuelle sur le métier (colonne Initial : 1 pour oui, 0 pour non).

PosteDeTravail	Configuration	Setup	Initial
102	2B2828R25	0	1
102	3B1918C12P5R13	35	0
103	2B2828R25	0	1
108	1B8712R20	50	0
108	3B4992C4P1R1	0	1
109	1B8712R20	65	0
109	3B1918C12P5R13	0	1
109	3B4992C4P1R1	66	0
117	1B8712R20	105	0
117	3B4992C4P1R1	0	1

TABLE B.4 – Temps des setup majeurs

Planification du partenaire industriel (planificationParCompany.csv) Ce fichier csv optionnel permet de forcer certains jobs à être effectués sur un métier donné dans un ordre donné.

PosteDeTravail	Ordre	Job
2	1	969054
2	2	968759
3	1	969205

TABLE B.5 – Planification du partenaire industriel

Uniques à la planification

Capacité totale des ressources humaines pour les temps de mise en course majeure (DisponibiliteRessource.csv) Ce fichier indique la disponibilité de chaque ressource, en nombre d'heures encore disponibles sur l'horizon de temps considéré. Pour le moment, il semble seulement y avoir les mécaniciens de type 2 qui sont utiles à la phase de planification.

Ressource	NombreEmployes	DisponibiliteUnitaire
MecanicienType2	1	160

TABLE B.6 – Capacité totale des ressources humaines pour les temps de mise en course majeure

Temps des setups majeurs selon la profession (TempsSetupRessource.csv) Ce fichier indique, pour chaque ressource requise pour les setups majeurs, le temps requis de la ressource pour passer de

la configuration actuelle à la nouvelle configuration.

Ressource	PosteDeTravail	Configuration	Setup
MecanicienType2	102	1B8712R20	30
MecanicienType2	102	2B2828R25	0
MecanicienType2	104	1B8712R20	45
MecanicienType2	105	3B4992C4P1R1	0

TABLE B.7 – Temps des setups majeurs selon la profession

Unique à l'ordonnancement

Capacité des employés selon leur profession (CapaciteEmployeProfession.csv) Ce fichier indique le nombre d'employés par profession et la capacité unitaire en heure de ces employés. Ce fichier donne aussi l'ordre des setups que devraient suivre les employés pour un setup mineur donné.

Profession	NombreEmployes	DisponibiliteUnitaire	Ordre
Mécanicien	2	45	1
Tisserand	5	30	2
Attacheur	3	35	3

TABLE B.8 – Capacité des employés selon leur profession

Date de début de disponibilité des métiers à tisser (DebutMinimum.csv) Ce fichier indique le moment au plus tôt de la disponibilité de chaque métier en vue d'un setup ou d'une première tâche. Ce tableau donne aussi le type d'un métier, le si la réparation des bris est automatisée, les temps de réparation estimés, ainsi que la dernière job qui avait été exécutée sur le métier avant le nouvel ordonnancement.

PosteDeTravail	Debut Minimum	Type Métier	Reparation Automatique	Temps Reparation Chaine	Temps Reparation Trame	Derniere Job
102	1000	3	1	600	60	4
103	120	2	0	600	240	2
108	500	1	0	600	240	1
109	900	4	1	600	60	3

TABLE B.9 – Date de début de disponibilité des métiers à tisser

Date de fin des jobs (DateDeFin.csv) Ce fichier indique la date où la tâche doit être faite ainsi que la priorité de la tâche. Ici, si une date de fin dépasse l'horizon, nous considérons que cette date de fin est égale à l'horizon. Nous utiliserons alors la priorité pour accorder une pénalité aux jobs dépassant la date limite.

Job	DateFinTissage	Priorite
5	350	1
9	1000	1
11	900	1
12	900	1

TABLE B.10 – Date de fin des jobs

Transition entre les jobs (TransitionJobSituation.csv) Ce fichier indique permet d'encoder les différentes transitions entre les jobs. Ici, nous utilisons cette table transitoire pour permettre de réduire les redondances au niveau des temps de setup pour les professions. Situation représente alors une des combinaisons de temps par professions et transition de job.

Job	DateFinTissage	Priorite
5	350	1
9	1000	1
11	900	1
12	900	1

TABLE B.11 – Transition entre les jobs

Situation possible selon le métier à tisser (PosteDeTravailSituation.csv) Cette table montre quelle situation peut exister sur quel métier.

PosteDeTravail	Situation
101	x
102	y

TABLE B.12 – Situation possible selon le métier à tisser

Temps de setup des professions (TempsSetupProfession.csv) Ce fichier indique le temps de setup par profession pour passer d'un job A à un Job B. Si un mécanicien et un tisserand sont requis sur le changement de job, on aura toujours besoin des deux pour ce changement.

Situation	Profession	Temps
x	Tisserand	1
x	Attacheur	3
x	Mecanicien	3
y	Tisserand	1

TABLE B.13 – Temps de setup des professions

JobProduit (JobProduit.csv) Ce fichier indique le descriptif de la tâche à faire.

Job	NoCommande	NoClient	Style	Couleur	Quantite	Teinture	DateCreation
1	7520	HAW66	1003	9999	120	1	43614.38
2	8526	INV01	5016	1455	240	1	43614.35
3	10520	TRU75	5016	1456	300	0	43614.33

TABLE B.14 – Caractéristiques des pièces de textile à tisser

Le taux de casses des styles selon les métiers (StyleTauxCasses.csv) Ce fichier indique les taux de casses selon un style et un type de métier. Il est utilisé pour générer r_jobBreak.csv.

Style	TypeMetier	TauxCassesChaine	TauxCassesTrame
1000	1	0.294142	1.40868
1000	2	0.513	2.457
1001	1	0.29412	1.40868

TABLE B.15 – Le taux de casses des styles selon les métiers

B.5.2 Fichiers créés au fur et à mesure du programme

Dans l'ordre linéaire de l'exécution du programme, le premier fichier à apparaître sera PlanData.dzn, un fichier de données MiniZinc créé par mainPlan.py pour réaliser la planification.

Le résultat de cette planification basée sur le modèle PlanModel.mzn, avec les données PlanData.dzn, est ensuite contenu dans planResult.txt.

Ensuite, il y a conversion de ce fichier texte en 6 fichiers CSV :

Configuration finale des métiers après l'horaire (r_configFinale.csv) Ce fichier représente les configurations finales des métiers. Ça inclut aussi le temps total de setup majeur utilisé.

PosteDeTravail	Configuration	Setup
1	3B1918C12P5R13	0.0
2	2B8712R20	40.0
3	1B7200R7	0.0
101	1B8712R20	10.0

TABLE B.16 – Configuration finale des métiers après l'horaire

Matrice de transition des jobs réduite (r_filteredJobSituation.csv) Ce fichier contient uniquement les transitions de TransitionJobSituation.csv qui sont utiles selon le plan, ainsi que les transitions des jobs sentinelles du modèle d'ordonnancement. Ça permet de grandement réduire le nombre de données passées au modèle.

JobA	JobB	Situation
974174	969054	X
974174	968759	X
968759	971105	Y
971105	968759	Y

TABLE B.17 – Matrice de transition des jobs réduite

Assignation des jobs sur les métiers (r_jobAssignment.csv) Ce fichier présente le résultat de la planification quant à l'assignation des jobs aux métiers. Ce fichier inclus aussi la durée des jobs sur les métiers, la configuration pour effectuer le job, ainsi que si le job est avant ou après le setup majeur du métier sur lequel il est exécuté.

Job	PosteDeTravail	Duree	Configuration	beforeSetup
965385	131	12.566666	2B2828R25	1
965393	131	8.383333	2B2828R25	1
966119	131	11.95	2B8712R20	0
966119	150	13.5333	1B8712R20	1

TABLE B.18 – Assignation des jobs sur les métiers

Taux de bris pour les jobs (r_jobBreaks.csv) Ce fichier présente les différents taux de bris pour chaque job. Cela inclut les bris de chaîne et les bris de trame.

Job	Chaine BrisMoyen	Trame BrisMoyen	Chaine TempsReparation	Trame TempsReparation
1	6	30	0.16666	0.01666
2	4	20	0.16666	0.06666
3	6	30	0.16666	0.01666

TABLE B.19 – Taux de bris pour les jobs

Utilisation des métiers selon le plan fait avant l'étape d'ordonnancement (r_usageMetier.csv) Ce fichier contient la durée des setups majeurs ainsi que la somme des durées des jobs assignées à chaque métier.

PosteDeTravail	DureeProcess	DureeSetup
1	0.0	0.0
2	176.53	0.0
3	93.44	20.0

TABLE B.20 – Utilisation des métiers selon le plan fait avant l'étape d'ordonnancement

Utilisation des ressources de setups majeurs (r_usageRessource.csv) Ce fichier est la somme de tous les temps que les employés passent à effectuer des setups majeurs.

Ressource	TempsRessource
MecanicienType2	13.0

TABLE B.21 – Utilisation des ressources de setups majeurs

L'exécutable ThumbMaker.exe se charge ensuite de créer resultFromHeuristic.txt ainsi que Circuit-Heuristic.dzn, qui est un ordonnancement initial.

Un autre fichier de données MiniZinc, SchedulingData.dzn est généré par mainScheduling.py, et enfin la sortie finale de l'ordonnancement se situe dans resultScheduling.txt.

B.6 Fichiers Python

csv2dzn.py : classe pour gérer la conversion du format csv vers un dzn contenant toute l'information des csv.

mainPlan.py : permets de convertir les csv d'entrée en dzn compatible avec le modèle de planification.

Paramètre 1 : mode de planification/ordonnancement. Cela doit être FilterFromCompany pour utiliser uniquement les jobs que l'équipe de planification a choisi de planifier ou autre pour utiliser tous les jobs du jeu de données.

Paramètre 2 : permets de choisir la liberté du modèle. FromCompany force les jobs à respecter l'ordre choisi par l'équipe de notre partenaire industriel. DifferentBeam n'exécute pas l'algorithme regroupant les tâches ayant la même ensouple sur le même métier. Finalement, exécute normalement le système.

Paramètre 3 : chemin du jeu de données choisi

Paramètre 4 : horizon de planification et d'ordonnancement (en heures, ex : 240)

dznWriter.py : classe ayant plusieurs fonctionnalités pour écrire des fichiers en format dzn.

mainScheduling.py : permet de convertir des csv résultats de planification vers un dzn compatible avec le modèle d'ordonnancement

Paramètre 1 : permets de choisir la liberté du modèle.

FromCompany force les jobs à respecter l'ordre choisi par l'équipe de notre partenaire industriel. DifferentBeam n'exécute pas l'algorithme regroupant les tâches ayant la même ensouple sur le même métier. Finalement, exécute normalement le système.

Paramètre 2 : chemin du jeu de données choisi

Paramètre 3 : choix entre un circuit initial par règle du pouce (thumb) ou par Minizinc (MiniZincCircuit)

output2csv.py : convertis le fichier de sortie de la planification en différents fichiers csv

Paramètre 1 : chemin vers le jeu de données

createMachineDzn.py : fonction créant un fichier .dzn par machine

B.7 Fichiers Minizinc

PlanModeConstraints.mzn : selon le mode de planification choisi, des contraintes seront rajoutées à PlanModeConstraints.mzn. Ce fichier est inclus par PlanModel.mzn.

PlanForceFromCompany.mzn : Contraintes du mode qui forcent les jobs à être comme le choix fait par l'équipe de planification de notre partenaire industriel. Peut se rajouter à PlanModeConstraints automatiquement avec les paramètres des scripts.

PlanSameBeam.mzn : Contraintes qui forcent des jobs ayant la même ensouple à être sur un même métier. Peut se rajouter à PlanModeConstraints automatiquement avec les paramètres des scripts.

PlanModel.mzn : Modèle de planification principal.

SchedulingDeterministic.mzn : Inclus le modèle SchedulingMain.mzn, pour résoudre le problème de manière déterministe.

SchedulingModeConstraints.mzn : selon le mode d'ordonnancement choisi, des contraintes seront rajoutées à SchedulingModeConstraints.mzn. Ce fichier est inclus par SchedulingMain.mzn.

SchedulingMain.mzn : Contraintes de bases utilisées pour l'ordonnancement par toutes les méthodes d'ordonnancement développées.

SchedulingForceCompanyJobOrder.mzn : Contraintes du mode qui forcent les jobs à être comme le choix fait par l'équipe de planification de notre partenaire industriel. Peut se rajouter à SchedulingModeConstraints.mzn automatiquement avec les paramètres des scripts.

SchedulingStyleSymBreak.mzn : Bris de symétrie qui est effectué sur les jobs ayant le même style et produit. Peut se rajouter à SchedulingModeConstraints.mzn automatiquement avec les paramètres des scripts.

B.8 Licences

MiniZinc : https ://www.minizinc.org/license.html

CBC : https ://github.com/coin-or/Cbc/blob/master/LICENSE

Chuffed : https ://github.com/chuffed/chuffed/blob/master/LICENSE