HAL
open science

LORH : outil pour la planification du parcours patient dans le milieu hospitalier

Olivier Gérard

To cite this version:

Olivier Gérard. LORH : outil pour la planification du parcours patient dans le milieu hospitalier. Modélisation et simulation. Université de Picardie Jules Verne, 2022. Français. NNT : 2022AMIE0060 . tel-04104057

HAL Id: tel-04104057

https://theses.hal.science/tel-04104057

Submitted on 23 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Thèse de Doctorat

Mention Informatique
Spécialité Recherche Opérationnelle

présentée à *l'Ecole Doctorale en Sciences Technologie et Santé (ED 585)*

de l'Université de Picardie Jules Verne

par

Olivier Gérard

pour obtenir le grade de Docteur de l'Université de Picardie Jules Verne

LORH : Outil pour la planification du parcours patient dans le milieu hospitalier

Soutenue le 24 novembre 2022, après avis des rapporteurs, devant le jury d'examen :

Mme. Sandra Bringay, Professeure, Université Paul Valéry - Montpellier III	Rapporteur
M. Antoine Jouglet, Professeur, Université de Technologie de Compiègne	Rapporteur
M. Gilles Dequen, Professeur, Université de Picardie Jules Verne	Président du jury & Examinateur
M. François Delbot, Maître de Conférences, Université Paris Ouest Nanterre	Examinateur
Mme. Corinne Lucet-Vasseur, Maître de Conférences HDR, Université de Picardie Jules Verne	Directrice de thèse
Mme. Laure Brisoux-Devendeville, Maître de Conférences, Université de Picardie Jules Verne	Co-directrice de thèse
M. Sylvain Darras, Docteur, Evolucare Technologies	Invité

Table des matières

Introduction **1**

1 Contexte industriel et problématique **7**

1.1 Présentation de l'entreprise Evolucare Technologies 8

1.1.1 Historique et chiffres 8

1.1.2 Evolucare Labs 10

1.1.3 Contexte économique 11

1.1.4 Gamme de produits 14

1.2 Problème de planification dans le milieu médical 15

1.2.1 Définition informelle 15

1.2.2 Contraintes 16

1.2.3 Notion de parcours de soin 16

1.2.4 Méthode de résolution actuelle 17

1.3 Le projet LORH 20

1.3.1 Cadre scientifique du projet 21

1.3.2 Verrous technologiques 22

1.3.3 Opportunités 27

1.3.4 Étude de la concurrence 28

2 Domaine de l'étude **31**

2.1 Optimisation combinatoire 31

2.1.1 Définition 32

2.1.2 Complexité 34

2.2 Problèmes de planification dans le domaine de la santé 37

2.2.1 Généralités 37

2.2.2 Types de problèmes 42

2.3 Resource Constraint Project Scheduling Problem 47

2.3.1 Définition 47

2.3.2 Extensions du RCPSP 49
2.3.3 Méthodes de résolution 55

3 Description du problème et formalisation 63
3.1 Formulation du problème 64
3.1.1 Données du problème 64
3.1.2 Contraintes 66
3.1.3 Solution & Fonction objectif 68
3.2 Modèle mathématique 71
3.3 Instances & Résultats 74
3.3.1 Cas d'utilisation concrets 74
3.3.2 Instances de l'entreprise 78
3.3.3 Instances PSPLIB 87

4 Recherche locale 93
4.1 Méthode de construction aléatoire 94
4.2 Mouvements 102
4.2.1 Destruction aléatoire et reconstruction simple 103
4.2.2 Destruction aléatoire et reconstruction optimale 104
4.2.3 Destruction basée sur la difficulté et reconstruction optimale . 108
4.2.4 Destruction ciblée 109
4.2.5 Échange de date de début 115
4.2.6 Décalage à gauche 117
4.3 Adaptive Large Neighborhood Search 118
4.3.1 Principe général 120
4.3.2 Formule d'ajustement des poids 121
4.4 Expérimentations & Résultats 123
4.4.1 Paramétrage 123
4.4.2 Instances de l'entreprise 126
4.4.3 Instances PSPLIB 132

5 Algorithme génétique 139
5.1 Principe de l'algorithme génétique 140
5.1.1 Principe général 140

5.1.2 Codage des solutions 141
5.1.3 Reproduction 143
5.1.4 Mutation 153
5.1.5 Sélection 158
5.1.6 Algorithme général 163
5.2 Métriques et convergence de l'algorithme 165
5.2.1 Métriques 166
5.2.2 Calcul de diversité 172
5.2.3 Algorithme génétique avec calcul de distance 180
5.3 Expérimentations & Résultats 185
5.3.1 Paramétrage 186
5.3.2 Résultats sur les instances de l'entreprise 189
5.3.3 Résultats sur les instances PSPLIB 194

Conclusion générale **201**

Remerciements **207**

Introduction

L'optimisation est une notion présente dans tous les domaines de notre vie. Nous la pratiquons tous au quotidien sans forcément nous en rendre compte : nous tentons d'organiser nos tâches et nos actions pour être les plus efficaces possibles, que ce soit pour le travail ou dans notre vie de tous les jours. Elle est aujourd'hui pour nous une question de survie : nous allons devoir trouver les moyens de faire mieux avec moins, d'optimiser l'utilisation de nos ressources pour faire face aux défis qui nous attendent et tenter de nous assurer un meilleur avenir. Telle est l'essence de l'optimisation : chercher des moyens plus intelligents, plus efficaces, de mener certaines actions afin d'obtenir de meilleurs résultats, ou de consommer moins de ressources dans l'accomplissement de ces actions.

Durant cette thèse CIFRE, nous nous sommes intéressés au domaine médical, qui doit faire face à de nouveaux défis et qui nécessite donc de nombreuses améliorations sous tous ses aspects. Ces dernières années ont révélé que les systèmes de soins de nombreux pays, développés ou non, n'étaient pas infaillibles et que la résilience des équipes soignantes était de plus en plus mise à mal. La demande de soins est de plus en plus élevée, que ce soit en terme de quantité ou de qualité : qu'il s'agisse de lutter contre des pandémies ou simplement garantir des soins suffisants à la population, l'ensemble des pays est concerné. Il est donc nécessaire, en France comme ailleurs, d'améliorer nos systèmes de santé, et les entreprises éditrices de logiciels pour la santé telles qu'Evolucare joueront un rôle central dans la modernisation des infrastructures numériques des établissements de santé. Dans nos travaux au sein de cette entreprise, nous nous sommes intéressés à l'organisation des soins. Plus précisément, à la planification du parcours de soins des patients et à l'optimisation de l'utilisation des différentes ressources au sein d'un établissement de santé.

Ce travail de planification consiste à prévoir les ressources nécessaires (patients, personnels soignants, équipements, salles, etc.) pour les actes médicaux à venir des patients de l'établissement, puis à fixer une date pour le rendez-vous en respectant

les disponibilités de chacun. C'est une tâche ardue qui incombe soit aux équipes de planificateurs pour les établissements qui en disposent, soit à certaines personnes expérimentées ayant une compréhension profonde du fonctionnement de leur service. Dans tous les cas, c'est un travail long et difficile de par le nombre important d'actes à planifier et de données à prendre en compte. Les emplois de temps qu'ils construisent doivent respecter un ensemble de contraintes visant à garantir l'intégrité d'un planning : respect des ressources requises et des disponibilités de chaque intervenant, par exemple. Ces plannings sont essentiels au bon fonctionnement d'un service, afin que les enchaînements de rendez-vous se fassent de la manière la plus fluide possible, et que les patients puissent être soignés dans de bonnes conditions, sans oublis ou reports. Pour les assister dans ce travail, les planificateurs ne disposent que de peu d'outils et ont donc exprimé le besoin d'un logiciel permettant la gestion d'agendas et la planification automatique des demandes de rendez-vous qu'ils ont à traiter. Pour répondre à ce besoin, l'entreprise Evolucare a monté le projet LORH (Logiciel d'Optimisation des Ressources Hospitalières) avec pour objectif de vendre un logiciel s'intégrant dans le système d'information des établissements de santé et permettant aux planificateurs d'obtenir en un temps raisonnable des plannings de qualité.

Les problèmes de planification et d'ordonnancement de tâches sont des sujets récurrents de la recherche opérationnelle depuis les années 1960, et sont toujours sources de nombreux sujets d'articles et de thèses. Le problème RCPSP (Resource Constraint Project Scheduling Problem) semble particulièrement proche de celui que nous traitons. Les bases de sa définition ont été posées en 1969 par Pritsker (Pritsker et al., 1969), et de nouvelles variantes apparaissent régulièrement pour décrire certains problèmes plus spécifiques ou plus complexes. Dans sa version classique, il s'agit de planifier un ensemble d'activités nécessitant diverses ressources pour être réalisées et étant reliées par des relations de précédence. C'est un problème NP-difficile qui mobilise les chercheurs depuis des décennies. En premier lieu des méthodes exactes capables de fournir les solutions optimales sur des instances de taille modeste, puis de nombreuses heuristiques et méta-heuristiques capable de traiter des problèmes bien plus grand, au prix d'une qualité de solution qui ne sera pas forcément optimale.

Dans le cadre de cette thèse, nous nous sommes intéressés à trois approches pour résoudre les problèmes présentés à l'entreprise. Notre première approche est un modèle de programmation linéaire en 0-1. Pour élaborer ce modèle, nous avons avant tout décrit formellement la problématique de planification proposée par les planificateurs, ce qui inclut l'ensemble des variables et des contraintes intrinsèques au problème. Nous avons également élaboré la fonction objectif, en collaboration avec l'entreprise et en accord avec les souhaits des planificateurs de divers établissements en France. Cette fonction, indispensable à la suite de nos travaux, permet d'évaluer la qualité d'une solution. Pour tester l'efficacité de ce modèle et de toute autre méthodes élaborées ultérieurement, nous avons créé un ensemble d'instances tirées de quelques scénarios d'usage typique d'un logiciel de planification automatique. Ces scénarios ont été élaborés en s'inspirant de situations et de cas décrits par des planificateurs, et représentent des problèmes qu'ils rencontrent régulièrement et qu'ils souhaiteraient voir résolus par un tel logiciel. Ce modèle de programmation linéaire en 0-1, implémenté sous CPLEX, nous a permis d'obtenir des solutions optimales sur certaines instances et des bornes inférieures et supérieures de la fonction objectif sur les autres. Ces travaux ont fait l'objet de deux publications : à la conférence nationale de la ROADEF 2020 (Gérard et al., 2020) et à la conférence internationale PMS (Gérard et al., 2021b).

La seconde approche que nous avons étudiée est une méthode incomplète basée sur une recherche locale et reposant sur différentes modifications très localisées sur une solution permettant d'obtenir une nouvelle solution proche de la précédente. Ces modifications, plus communément appelées « mouvements », sont les composants essentiels des méthodes de recherche locale, qui explorent l'espace de recherche en se déplaçant de solution en solution. Chaque mouvement définit un voisinage de la solution courante, et le choix du voisin vers lequel s'orienter se fait en fonction de divers critères spécifiques au problème traité. Pour obtenir les solutions initiales de ces recherches, nous proposons l'algorithme de construction $LORH_ARC$, qui attribuera les premières ressources disponibles à chaque demande de rendez-vous du problème. L'ordre dans lequel ces demandes sont traitées est calculé en fonction de certaines caractéristiques : la priorité de la demande, son importance ou encore les rendez-vous qui devront la précéder. Cet algorithme nous permet d'obtenir rapidement des

solutions que nous tentons ensuite d'améliorer par application successive de mouvements. Nous avons défini quatre mouvements de destruction et de construction, qui consistent à retirer des rendez-vous d'une solution pour permettre d'en planifier d'autres selon certains critères de choix tels que la difficulté à placer certains rendez-vous, ou la surexploitation de certaines ressources. Nous avons défini deux autres mouvements : l'échange de rendez-vous et le décalage à gauche, qui ne sont pas basés sur des opérations de destruction et de reconstruction mais permettent aussi l'amélioration des solutions et l'exploration de l'espace de recherche. Ces six mouvements ont été utilisés au sein d'une ANLS (Adaptive Large Neighborhood Search), une méthode définie par Shaw (1998) et basée sur la recherche locale à voisinage large. À chaque itération de l'algorithme, un mouvement est sélectionné parmi un ensemble défini au préalable et appliqué selon des probabilités initialement équivalentes pour l'ensemble des mouvements. Selon la qualité des solutions obtenues, les probabilités évoluent pour favoriser les plus performants. Les résultats obtenus sur nos instances par cette méthode, nommée *LORH_ALNS*, ont été comparés aux solutions optimales et aux bornes trouvées avec notre approche précédente, et nous avons obtenu de nouvelles solutions optimales sur les instances d'Evolucare. Nous avons aussi démontré l'efficacité de la couche adaptative de l'algorithme sur notre problématique, en comparant les résultats obtenus avec et sans l'adaptation des poids associés aux mouvements. Ce travaux ont été soumis à la ROADEF 2021 (Gérard et al., 2021c) et à la conférence internationale APMS (Gérard et al., 2021a).

La troisième et dernière approche que nous présentons dans ce manuscrit simule le processus d'évolution théorisé par Charles Darwin (Darwin, 1859) pour façonner et améliorer des solutions. Cette méthode, démocratisée entre autres par John H. Holland (Holland John, 1975) et David E. Goldberg (Golberg, 1989) s'est révélée particulièrement efficace sur des problèmes de planification similaires à ceux que nous traitons. Elle repose essentiellement sur trois opérateurs génétiques utilisés sur les populations de solutions : la reproduction, la mutation et la sélection. Nous décrirons donc l'adaptation de ces opérateurs à notre problème, ainsi que le codage et la représentation de nos solutions dans l'algorithme génétique. Si notre algorithme, appelé *LORH_GA*, a effectivement fourni de bons résultats sur nos scénarios, nous avons aussi constaté une convergence rapide des populations de solutions

vers certains points de l'espace de recherche, appelés optimums locaux. Pour mieux comprendre et maîtriser ce phénomène, nous avons adapté deux métriques de la littérature : la distance de Hamming (Hamming, 1950) et la distance de Kendall-Tau (Kendall, 1938) à notre problème, afin d'estimer une distance entre deux solutions. Ces distances nous permettent de calculer un score de diversité de la population, et ainsi d'avoir une idée plus précise du déroulement de la recherche. Pour remédier à des chutes trop brutales ou trop précoces de cette diversité, nous avons développé une autre approche à base d'algorithme génétique qui nous permet de mieux contrôler les phases d'exploration et d'exploitation de l'algorithme : $LORH_GADM$. Nous avons comparé les résultats obtenus par $LORH_GA$ et $LORH_GADM$ avec nos approches précédentes sur nos instances. Les deux algorithmes génétiques surpassent nos méthodes précédentes sur la majorité des instances de l'entreprise, et $LORH_GADM$ se montre aussi plus performant que $LORH_GA$ sur la plupart des instances. Nous avons également évalué les deux versions de l'algorithme génétique sur les instances de la littérature RCPSP, de la bibliothèque PSPLIB (Kolisch and Sprecher, 1997), regroupant plusieurs ensembles d'instances régulièrement utilisés dans la littérature. Les travaux sur l'algorithme génétique et les distances ont été présentés à la conférence ROADEF 2022 (Gérard et al., 2022).

Ce manuscrit est organisé en cinq chapitres. Dans le premier chapitre, nous introduisons le contexte industriel de la thèse, en présentant l'entreprise Evolucare Technologies, la problématique que nous tentons de résoudre et le projet LORH qui représente la réponse de l'entreprise à cette problématique. Nous nous intéressons dans le deuxième chapitre à la littérature concernant notre problème de planification : sa complexité, ses caractéristiques et la place qu'il occupe au sein de la littérature des problèmes de planification dans le domaine médical. Nous faisons également le lien entre notre problème et le RCPSP avant de présenter les méthodes de résolution utilisées pour résoudre le RCPSP. Le troisième chapitre présente notre formulation pour représenter tous les éléments de la problématique et le modèle de programmation linéaire en 0-1 que nous avons conçu. Nous y expliquons aussi la création des instances dérivées des scénarios obtenus par l'entreprise. Dans le quatrième chapitre nous détaillons l'algorithme de construction des solutions initiales $LORH_ARC$ ainsi que les différents mouvements et leur intégration au sein

de *LORH_ALNS*. Enfin, le chapitre cinq présente nos travaux sur l'adaptation de l'algorithme génétique à notre problème : les opérateurs de mutation, de reproduction et de sélection que nous avons mis en œuvre. Nous décrivons aussi dans ce chapitre les métriques et les modifications de l'algorithme génétique pour tirer profit des mesures rendues possibles de la diversité de la population. Nous terminerons ce manuscrit par la conclusion et les perspectives sur ces travaux.

Chapitre

1

Contexte industriel et problématique

Contents

= 1.1 Présentation de l'entreprise Evolucare Technologies

1.1.1 Historique et chiffres

1.1.2 Evolucare Labs

1.1.3 Contexte économique

1.1.4 Gamme de produits

1.2 Problème de planification dans le milieu médical

1.2.1 Définition informelle

1.2.2 Contraintes

1.2.3 Notion de parcours de soin

1.2.4 Méthode de résolution actuelle

1.3 Le projet LORH

1.3.1 Cadre scientifique du projet

1.3.2 Verrous technologiques

1.3.3 Opportunités

1.3.4 Étude de la concurrence

Dans cette partie nous présentons le contexte industriel de cette thèse CIFRE. Pour commencer nous décrirons brièvement l'entreprise Evolucare, son historique et nous donnerons quelques chiffres importants la concernant. Nous introduirons ensuite la problèmatique de planification traitée dans ce manuscrit, puis le projet LORH qui est la réponse d'Evolucare à cette problématique.

1.1 Présentation de l'entreprise Evolucare Technologies

Le groupe Evolucare Technologies est une société familiale présente dans l'édition de logiciels de santé depuis plus de trente ans, et propose une des plus larges gammes de logiciels métiers dédiés aux établissements de santé. Aujourd'hui, Evolucare est une Entreprise de Taille Intermédiaire (ETI) présente dans le monde entier avec 16 agences réparties sur tous les continents, mais majoritairement basée en Europe et en France, avec une agence principale se trouvant à Villers-Bretonneux en Picardie.

1.1.1 Historique et chiffres

À l'origine du groupe connu aujourd'hui sous le nom d'Evolucare Technologies, a été fondée la société Corwin. Corwin a été créée en 1996 et ne comptait à l'époque que six collaborateurs et une seule agence. Ces personnes expérimentées dans le domaine de l'édition de logiciels de santé proposaient déjà des solutions de qualité pour les premiers clients de la société. L'entreprise a connu de nombreux changements au fil du temps, que ce soit dans le nombre de collaborateurs et d'agences ou dans la création des différentes filiales du groupe. La figure 1.1 détaille les moments marquants dans la vie de l'entreprise.

FIGURE 1.1: Dates importantes dans l'histoire d'Evolucare

Depuis sa création en 1996 avec six collaborateurs, la société n'a cessé de s'agrandir et compte aujourd'hui plus de 320 employés, avec l'intention de doubler ce chiffre dans les années à venir. Cette croissance s'est évidemment accompagnée d'un agrandissement des locaux, notamment avec l'inauguration de l'agence principale à Villers-

Bretonneux. D'autres agences ont ensuite été créées dans différentes villes de France (Paris, Vannes, Aix-en-Provence et d'autres) mais aussi en Europe : en Belgique, en Allemagne, en Turquie 1.2. D'autres agences ont depuis vu le jour dans le monde entier pour se rapprocher des clients sur tous les continents. En 2009, le groupe Evolucare Technologies est créé afin de regrouper les différentes sociétés au sein d'une même entité et ainsi renforcer l'image de l'entreprise. Par la suite, de nombreuses sociétés seront créées ou rachetées pour étoffer la gamme de produits du groupe. Fin 2016, le service Innovation (nommé Evolucare Labs) voit le jour pour enrichir la gamme de produits de l'entreprise avec des solutions innovantes nécessitant un travail de recherche, des outils plus innovants et des compétences diverses.

FIGURE 1.2: Localisation des différentes agences du groupe Evolucare

À l'image de l'entreprise, le nombre et le panel de clients d'Evolucare s'est lui aussi étoffé avec le temps. À ses débuts, les 20 clients de la société étaient principalement des centres médico-sociaux qui cherchaient à poser les premières briques logicielles de ce que serait leur système d'information bien plus tard. Avec les années, les besoins de création ou d'amélioration de logiciels de gestion se sont accentués et en élargissant son offre, Evolucare a attiré d'autres types de clients : des centres de Soins de Suite et de Rééducation (SSR), des centres hospitaliers et des cliniques MCO (Médecine, Chirurgie et Obstétrique, prenant en charge les soins de courte durée), etc. Le nombre de clients a ainsi augmenté de manière exponentielle depuis 20 ans : 90 clients en 2000, 1000 clients en 2010 et 4500 clients en 2021 répartis

principalement en Europe mais présents aussi sur les autres continents. À travers les logiciels du groupe, ce sont aujourd'hui 1,2 million de patients pris en charge, 3,8 millions de prescriptions médicales et 70 millions de rendez-vous qui sont gérés chaque année. L'entreprise Evolucare réalise en 2020 un chiffre d'affaire de 30 millions d'euros, et en dépit de la crise sanitaire actuelle, les perspectives de croissance sont suffisamment rassurantes pour envisager d'agrandir encore le groupe dans les années à venir.

1.1.2 Evolucare Labs

Evolucare Labs (E-Labs), créé fin 2016, est le laboratoire de recherche et développement d'Evolucare permettant d'ajouter de nouvelles technologies et des fonctionnalités innovantes au sein des projets du groupe. En cela E-Labs crée de la valeur ajoutée par l'innovation dite de rupture, des innovations destinées à remplacer la technologie en place actuellement. E-Labs conçoit des prototypes opérationnels qui pourront ensuite être réutilisés par les différentes entreprises du groupe Evolucare Technologies. L'équipe s'entoure de grands laboratoires de recherche tels que le laboratoire Modélisation, Information et Systèmes (MIS) de l'Université Picardie Jules Verne (UPJV), le Laboratoire de Traitement de l'Information Médicale (LaTIM) de l'Université de Bretagne Occidentale (UBO), ou encore le Centre de Recherche en Psychologie, Cognition, Psychisme et Organisations (CRP CPO) faisant également parti de l'UPJV. Elle travaille également avec des établissements de professionnels du domaine médical, de nombreux CHU en France notamment. Par cette démarche d'innovation collaborative, les nouvelles technologies sont au cœur des prototypes développés par l'équipe.

L'objectif principal du service est la création de solutions innovantes, notamment au travers de nouvelles technologies comme les objets connectés et l'intelligence artificielle, qui seront indispensables pour la pérennité de l'entreprise dans les années à venir. Mais le Labs a aussi pour objectif de communiquer largement à destination des acteurs de l'écosystème, afin de se positionner comme élément moteur dans le domaine de la santé et de nouer des collaborations stratégiques dans le cadre de projets transdisciplinaires de grande ampleur.

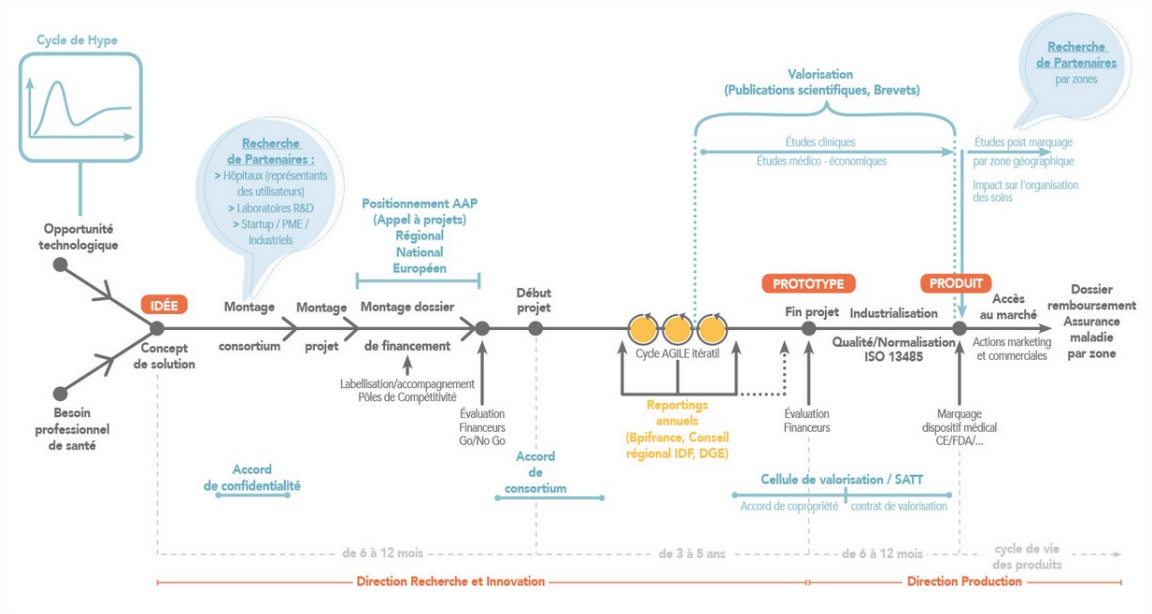


FIGURE 1.3: La chaîne de création de valeur

Le cycle de vie d'un projet innovant au sein d'Evolucare est dépeint dans la figure 1.3. Ce schéma nous montre que l'idée d'un projet naît de la convergence d'un besoin dans le domaine de la santé et d'une opportunité technologique. Dans notre cas, l'opportunité technologique est l'intelligence artificielle, qui est impliquée dans bon nombre de projets du Labs. S'ensuit une période de montage de dossier, de demande de financement et de recherche de partenaire qui, si elle aboutit, mène au développement à proprement parler de l'idée de base. Ce développement s'étale sur plusieurs années et conduit à la production d'un prototype. Les projets sont valorisés par des publications dans des revues scientifiques, des demandes de brevets ou des études cliniques. Quand le projet est terminé, le prototype passe par une phase d'industrialisation afin de le transformer en produit, qui sera alors en mesure d'être vendu.

1.1.3 Contexte économique

L'amélioration des systèmes de soins est un enjeu dans le monde entier, que ce soit pour les pays développés ou pour les pays en voie de développement. Les premiers doivent gérer une demande de soin croissante tandis que les seconds font face à des environnements difficiles et à un manque de moyens financiers ou de ressources médicales. De plus, la crise sanitaire provoquée par l'épidémie de Covid-19 a mis

en lumière à la fois la résilience des équipes médicales et la nécessité de moderniser infrastructures et systèmes d'information pour les aider à surmonter les défis actuels et à venir. Les entreprises impliquées dans l'édition de logiciels pour ce milieu vont donc avoir un rôle important dans les années qui viennent pour développer des logiciels et des produits pouvant épauler toutes les personnes travaillant dans le milieu médical.

1.1.3.1 Évolution de la Consommation de Soins et Biens Médicaux

En France, les dépenses en biens et en services de santé sont représentées par la Consommation de Soins et Biens Médicaux (CSBM). Cette somme regroupe les dépenses en soins hospitaliers et ambulatoires, en transports sanitaires, en médicaments et tout autre bien médical mais ne couvre pas les dépenses des personnes handicapées ou des personnes âgées en institution. Ces dépenses n'ont cessé d'augmenter ces dernières années, passant de 8.3% du PIB en 2008 à 9.1% en 2020, ce qui représente 209 millions d'euros (Gonzalez et al., 2021). D'après certaines projections (Geay and de Lagasnerie, 2013), ce chiffre va continuer à croître, pour atteindre 11.5% du PIB à l'horizon 2060.

Cette augmentation de la demande et des dépenses en matière de soin ne concerne pas que la France. De nombreux pays européens et une bonne partie des pays développés font face aux mêmes phénomènes. Les facteurs influençant ces tendances sont multiples, mais parmi eux certains sont cités plus fréquemment dans tous ces pays. Ces facteurs sont :

— Le vieillissement de la population dans la plupart des pays développés, qui entraîne des besoins en soins plus conséquents en raison de la fragilité de nos aînés. Ces derniers nécessitent des prises en charge de plus en plus fréquentes et plus coûteuses au fil du temps.

— L'augmentation des revenus moyens et la hausse du niveau de vie, qui pousse les habitants à demander des services de soins plus performants et plus évolués qui s'avèrent être plus coûteux.

— Les progrès de manière générale dans le domaine médical. Que ce soit des avancées médicales, de nouveaux traitements ou des progrès techniques, ces nouvelles méthodes sont souvent plus onéreuses.

La demande de soin des habitants s'accroît donc et les populations souhaitent accéder à des soins de meilleure qualité, plus efficaces et innovants. Pour répondre à cette demande, les structures existantes devront être améliorées : extension et aménagement des locaux, recrutement de plus de personnel, optimisation des procédés, les pistes d'amélioration sont nombreuses. Pour couvrir les déserts médicaux, il sera aussi nécessaire de construire de nouveaux établissements. Dans tous les cas, ces aménagements et nouvelles structures devront être accompagnés de meilleurs systèmes d'information, de logiciels pouvant à la fois alléger les charges administratives et organisationnelles des équipes et d'améliorer la qualité des soins prodigués aux patients. De ce fait, les opportunités seront nombreuses pour les entreprises spécialisées dans l'édition de logiciels pour le milieu de la santé.

1.1.3.2 Ségur de la Santé

Le Ségur de la Santé est une consultation des acteurs du système de soins en France qui avait pour but d'améliorer le secteur de la santé en traitant de différents sujets : le salaire, les soins, la gouvernance des hôpitaux, etc. Durant cette consultation d'environ deux mois, les représentants syndicaux, les professionnels de santé et le gouvernement français ont discuté et négocié pour construire un plan d'investissement permettant de répondre à la demande croissante de soins et permettant aussi d'améliorer les conditions de travail de toute personne travaillant dans le milieu médical. Cette consultation ayant eu lieu en pleine épidémie n'était pas seulement liée au Covid-19, qui, encore une fois, a mis en relief certaines des difficultés rencontrées. Elle vise à répondre à des problèmes antérieurs auxquels notre système de soin doit trouver des réponses.

Les conclusions du Ségur de la Santé font état de différentes décisions (recrutements, ouverture de lits, développement de la télé-santé) et d'investissements financiers dans le système de santé. Dans ces conclusions, nous relevons en particulier le plan d'investissement de 19 milliards d'euros sur 10 ans dans le système de santé pour l'amélioration de la prise en charge des patients et du quotidien des soignants. Ces 19 milliards d'euros seront répartis de la manière suivante :

— 9 milliards pour les grands projets et l'amélioration du quotidien dans les établissements de santé ;

— 6,5 milliards pour le désendettement des hôpitaux ;

— 1,5 milliards pour la modernisation des EHPAD ;

— 2 milliards pour le développement du numérique ;

Ces investissements encourageront la construction de nouveaux établissements et la modernisation des structures existantes, en passant notamment par la modernisation de leurs logiciels de gestion. Cela instaure un contexte favorable pour les entreprises telles qu'Evolucare, avec des opportunités pour acquérir de nouveaux clients et de vendre plus de solutions logicielles aux clients actuels de l'entreprise. Les 2 milliards d'investissement pour le développement du numérique ont pour objectif de faciliter les échanges de données entre particuliers et professionnels du domaine. C'est un soutien important, car certaines structures sont réticentes à l'idée d'installer de nouveaux logiciels à cause des incompatibilités et les échanges difficiles qu'il peut y avoir entre les différents logiciels présents dans ce secteur. Le personnel est souvent attaché ou habitué à certains systèmes historiquement utilisés, et sont donc plus réticents pour adopter de nouveaux logiciels qui cohabitent difficilement avec leurs logiciels habituels. La facilitation des échanges entre les différentes solutions des divers éditeurs encouragera donc la mise à niveau des systèmes d'information dans tout le secteur de la santé.

1.1.4 Gamme de produits

À ses débuts, Evolucare Technologies proposait principalement des outils permettant le suivi informatique du dossier patient (appelé DPI, Dossier Patient Informatisé) de ses différents établissements clients. Actuellement, la gamme de produits dont dispose Evolucare touche quasiment toutes les étapes du parcours de soins d'un patient ainsi que tous les services des établissements les accueillant. De son admission dans l'établissement à la facturation des soins, en passant par la gestion des ordonnances, de l'anesthésie ou encore de l'imagerie, les logiciels vendus par Evolucare permettent une gestion complète du patient et de son parcours de soins. De fait, tous les services (administration, urgences, pharmacie, radiologie, etc.) sont aujourd'hui concernés par les solutions proposées par Evolucare Technologies.

La convergence des besoins croissants en soins de la population et de l'investissement massif de l'état dans le système de santé représente une occasion de développer

de nouvelles solutions, plus innovantes et capables de répondre à des problématiques nouvelles ou anciennes et qui ne disposent toujours pas de solutions adéquates. C'est de l'une de ces problématiques, justement ancienne mais persistante dont il est question dans ce mémoire : la planification des actes médicaux et rendez-vous dans le domaine de la santé.

1.2 Problème de planification dans le milieu médical

La planification est un problème complexe et récurrent présent dans de nombreux domaines. Elle est décrite comme étant l'action d'organiser dans le temps une succession d'activités nécessitant un nombre défini de ressources en respectant les contraintes liées au problème, aux ressources et au contexte dans lequel elle s'inscrit. On la retrouve par exemple dans tous nos établissements scolaires, de la primaire à l'université où nous devons planifier les cours de nos écoliers, lycéens et étudiants en prenant en considération les nombreuses contraintes du milieu scolaire. Elle est aussi présente dans nos usines, où il nous faut prévoir et optimiser toutes les étapes de fabrication d'un produit afin d'améliorer les rendements et réduire les coûts au maximum. C'est aussi un domaine de recherche très actif au sein de la communauté de la Recherche Opérationnelle, en raison de sa complexité et des nombreux cas d'application existants dans le secteur industriel.

1.2.1 Définition informelle

La problématique de planification existe également dans le domaine de la santé, et c'est précisément ce dernier qui nous intéresse dans le cadre de cette thèse. Elle prend alors de nombreuses formes et varie selon les établissements et les services : centres de rééducation, établissements hospitaliers, blocs opératoires, cabinets médicaux ; chaque problème a ses spécificités et des difficultés qui lui sont propres. La planification, dans ce domaine particulier, consiste à organiser les différents rendez-vous entre les ressources de ces établissements. Généralement, cela consiste à prévoir les dates, le personnel médical et les ressources matérielles requises pour un acte médical ou un rendez-vous avec un ou plusieurs patients.

1.2.2 Contraintes

Naturellement, comme dans tout domaine, la planification dans le domaine médical s'accompagne de besoins et de contraintes particuliers. Un établissement hospitalier est organisé en plusieurs services spécialisés dans diverses disciplines de la médecine qui peuvent faire intervenir plusieurs praticiens : médecin généraliste, Infirmier Dimplômé d'État (IDE), kinésithérapeute, anesthésiste, radiologue, aide-soignant, etc. Les actes médicaux sont réalisés dans des salles dédiées : bloc opératoire, salle d'auscultation, plateau technique ou salle de réveil par exemple. De plus, certains actes médicaux ou de rééducation nécessitent l'utilisation de matériel spécifique qui va être soumis à des règles précises : délai minimum entre deux utilisations, nombre maximum d'utilisations par jour, etc.

Il faut également prendre en considération les désirs des patients concernant leur date d'admission, le choix des médecins qui vont s'occuper d'eux ou encore leurs préférences horaires pour certains actes médicaux. Les patients sont aussi soumis à des contraintes médicales : certains actes sont urgents tandis que d'autres font partie d'un parcours de soins et doivent respecter un ordre précis. La planification des actes et des rendez-vous impliquant ces différents acteurs et ressources tout en prenant en compte les différentes contraintes qui leur sont liées et leurs disponibilités est une tâche ardue qui demande un savoir-faire rare et énormément de temps.

1.2.3 Notion de parcours de soin

Pour le patient, une bonne planification de son parcours de soins est essentielle pour son bien-être et celui des équipes qui vont l'accueillir et lui prodiguer les soins dont il a besoin. Son séjour dans l'établissement se fait en plusieurs étapes menant au traitement de la ou des pathologies dont il souffre, étapes qu'il faut planifier en respectant toutes les contraintes évoquées précedemment.

Avant le début de son séjour au sein d'un établissement, ce dernier traite la reception de sa demande de prise en charge. Cette demande peut provenir d'un médecin adresseur auprès de l'établissement, du patient lui-même via des plateformes telles que Doctolib, ou venant d'autres plateformes telles que ViaTrajectoire ou ROR, par exemple. Régulièrement, une commission médicale d'admission se réunit pour étudier les demandes de prise en charge en attente. Elle sélectionne les patients

qui seront pris en charge en fonction des capacités de l'établissement, des soins à apporter et des ressources disponibles.

Si la commission rend un avis favorable, l'arrivée du patient est programmée et son Dossier Patient Informatisé est complété. Son arrivée au sein de l'établissement mobilise plusieurs personnes afin de l'accueillir dans le service où il sera traité et de régler les détails administratifs. La prise en charge du patient sera rythmée par tous les rendez-vous et les actes médicaux prévus dans son parcours de soins. Les actes médicaux pour un patient peuvent être nombreux et répartis sur plusieurs jours, plusieurs semaines et plusieurs établissement. Certains de ces rendez-vous doivent avoir lieu avant les autres, ou entraîner des périodes de repos pendant lesquelles le patient ne peut pas subir d'autres actes médicaux. Le parcours de soins impose donc certaines contraintes dans sa planification, et il peut aussi changer en fonction de l'état de santé du patient et de l'évolution de sa maladie.

À la fin de son parcours de soins, des documents (bilans, courriers, comptes-rendus, factures) sont rédigés pour préparer sa sortie, et sa prise en charge est clôturée une fois que toutes les factures ont été payées. Encore une fois, toutes ces étapes doivent être planifiées rigoureusement, et le problème devient extrêmement complexe si on considère qu'un établissement gère plusieurs centaines voir plusieurs milliers de patients.

1.2.4 Méthode de résolution actuelle

La planification des activités dans les établissements médicaux repose souvent sur une ou plusieurs personnes expérimentées et ayant une fine connaissance du fonctionnement de leur service. Le personnel planificateur doit traiter les demandes de rendez-vous une à une, en prenant en compte les emplois du temps de toutes les ressources nécessaires pour trouver une date respectant toutes les disponibilités et les contraintes évoquées précédemment.

1.2.4.1 Coûts humains et financiers

La construction des plannings est un travail long et complexe qui occupera les planificateurs pendant une grande partie de leur temps de travail, alors même que ce n'est pas leur cœur de métier : ce sont des médecins, des infirmiers et d'autres

spécialistes qui, par expérience, ont acquis une vision transverse des ressources et des soins. Pour assurer la continuité du travail d'organisation et de planification en cas d'absence temporaire (congés, arrêt maladie, etc.) ou prolongée (démission, départ à la retraite, etc.), le poste de planificateur est bien souvent doublé avec un second poste. Pour des établissements ou des services de plus grandes taille, ce sont des équipes qui se partagent le travail selon le domaine des membres de l'équipe.

La planification, nécessaire au bon déroulement du parcours de soin du patient, mobilise ainsi de nombreuses ressources et un temps précieux, quelle que soit la taille de la structure. De plus, cela tend à s'accentuer depuis le vote de la loi de santé de janvier 2016 instaurant entre autres les Groupements Hospitaliers du Territoire. Cette loi a pour objectif de faire fusionner plusieurs petites unités médicales en services de plus grande taille afin de mutualiser et de spécialiser les différents établissements d'une même zone géographique. Cependant, en augmentant la taille des unités, le problème de planification devient d'autant plus complexe, que ce soit pour un humain ou une machine. Cela accroît encore un peu plus le besoin pour le personnel planificateur de disposer d'un outils efficace pour planifier les parcours de soins des patients et autres rendez-vous au sein d'un établissement.

1.2.4.2 Outils et aide

Les planificateurs disposent dans certains établissements d'outils logiciels pour les assister dans leur travail d'organisation. Ces outils peuvent par exemple définir une récurrence pour la planification d'un rendez-vous, en le rendant quotidien ou hebdomadaire par exemple. Cette aide réduit la pénibilité de son travail, mais le planificateur doit tout de même choisir et affecter les ressources aux activités et actes médicaux, ces outils ne pouvant pas déterminer les meilleures affectations possibles. Il devra de plus gérer les conflits générés par cette méthode avec des rendez-vous planifiés auparavant. D'autres outils peuvent également trouver le premier créneau où un ensemble de ressources choisies sont disponibles. Là encore, c'est toujours au planificateur de choisir les ressources, et en cas de modification ultérieure, il devra relancer une recherche pour les rendez-vous impactés pour trouver de nouveaux créneaux. En dépit de l'existence de certains logiciels pour leur faciliter la tâche, une grande partie du travail de planification est toujours faite à la main.

Les équipes peuvent aussi avoir accès à des outils facilitant l'acquisition des informations qui définissent le problème à travers divers logiciels présents dans le système d'information de l'établissement. Les logiciels de gestion Ressources Humaines permettent parfois de visualiser les disponibilités et les vacations du personnel. Certains systèmes d'information contiennent des agendas, qui en plus des disponibilités des ressources indiquent quand celles-ci sont occupées par des rendez-vous ou des actes médicaux. De telles aides permettent une meilleure visualisation de toutes les informations nécessaires au traitement du problème. Néanmoins, il existe encore des établissements où elles n'existent pas sous format numérique, ce qui complique sérieusement le travail de résolution des planificateurs.

1.2.4.3 Difficultés

L'élaboration d'un emploi du temps cohérent est un processus long et complexe. Même en prenant en compte toutes les aides et outils disponibles, la quantité d'informations des différentes sources à considérer est énorme. Ces informations sont également versatiles : l'activité est souvent soutenue dans le domaine médical, et l'agenda des diverses ressources médicales est modifié régulièrement en fonction des nouvelles prescriptions, des plans de soins pour les nouveaux patients et des ajouts, modifications et suppressions de certains rendez-vous. La solution élaborée par un planificateur est donc régulièrement remise en cause au moment même de sa construction.

Une fois terminé, le meilleur des plannings peut en effet être invalidé par des impondérables. Si un membre du personnel médical est absent, qu'une salle ou qu'une machine est inutilisable ou que l'état de santé d'un patient nécessite le report ou l'annulation de certains rendez-vous, c'est l'intégralité de la planification qui peut devenir caduque. Les actes médicaux et rendez-vous impactés devront soit être affectés à d'autres ressources, soit planifiés à une autre date. Parmi l'ensemble des prescriptions, certaines, de par leur urgence ou leur priorité, devront être maintenues, parfois au prix de la perturbation d'autres prescriptions qui n'étaient pourtant pas affectées par l'imprévu initial. Chaque rendez-vous peut en plus faire partie d'un ensemble chaîné d'actes médicaux qui ont des prédécesseurs et des successeurs. La

réaction en chaîne provoquée par un seul imprévu peut ainsi détruire partiellement ou complètement une solution qui paraissait pourtant adéquate.

Le planificateur et son équipe doivent alors fournir le plus rapidement possible un nouveau planning, pour que les rendez-vous et les actes médicaux les plus urgents ou importants puissent avoir lieu. Bien sûr, c'est une charge de travail supplémentaire plus que conséquente, mais c'est également la source d'un stress quasi permanent : étant donné la nature hautement sensible et les enjeux du domaine dans lequel ils exercent, les planificateurs savent qu'ils n'ont que peu le droit à l'erreur, ce qui peut peser lourd sur leur moral.

1.3 Le projet LORH

Pour faire face aux difficultés présentées jusqu'ici, les équipes des établissements de santé clients d'Evolucare ont exprimé le besoin d'un outil pour rechercher des solutions adaptées aux problèmes qu'ils rencontrent au quotidien. Il s'agit principalement pour ces professionnels d'améliorer la qualité de la prise en charge des patients, mais un tel logiciel améliorerait à la fois les conditions de travail du personnel planificateur et de toutes les équipes d'un établissement. Pour les planificateurs, ce serait une aide bienvenue dans leur travail difficile décrit précédemment. Pour tout le personnel d'un établissement, c'est une meilleure gestion des emplois du temps, où leurs préférences seront mieux prises en considération. Pour les établissements eux-même, une gestion plus efficace des ressources leur permet de faire des économies sur le fonctionnement des différents services.

Ces besoins ont été rapportés par les collaborateurs d'Evolucare en contact avec les établissements clients de l'entreprise. Ces derniers souhaitent des logiciels capables de fournir des planifications qui sont utilisables en pratique, et qui sont donc bien plus complexes que les outils existants. Pour répondre à cette demande, Evolucare a fait appel à l'équipe Evolucare Labs pour étudier le problème et proposer une solution logicelle permettant de répondre à la problèmatique posée. Ce besoin des professionnels de santé et l'expertise d'Evolucare Labs ont donné naissance au projet LORH, un logiciel proposant des planifications cohérentes et optimisées selon

les critéres définis par les utilisateurs. Néanmoins, plusieurs difficultés devront être surmontés pour parvenir à ce résultat.

1.3.1 Cadre scientifique du projet

Cette thèse a été réalisée dans le cadre d'un contrat CIFRE avec Evolucare Technologies et le laboratoire Modélisation, Information et Systèmes (MIS) de l'Université de Picardie Jules Verne (UPJV). Le but de cette collaboration est d'étudier le problème posé et d'améliorer les solutions qui y sont apportées, en nous appuyant à la fois sur l'expertise scientifique du laboratoire et des docteurs présents au sein d'Evolucare Labs, et sur l'expertise métier apportée par les collaborateurs d'Evolucare en contact avec le domaine de la santé.

Le concours de ces deux entités est indispensable pour tenter d'apporter des solutions au problème posé. La planification est un problème complexe, sujet de nombreuses recherches depuis des décennies dans de nombreux domaines, y compris le domaine médical. En raison de l'évolution de l'offre et de l'organisation des soins, cette problématique devient de plus en plus préoccupante pour les acteurs du monde médical. Ces derniers expriment ce besoin auprès des éditeurs de logiciels tels qu'Evolucare, ce qui nous permet d'avoir une description détaillée et concrète du problème, avec des cas réels à traiter et potentiellement des retours à moyen terme sur les solutions apportées par l'outil. Les techniques envisagées seront donc rapidement confrontées à des problèmes concrets, qui peuvent nous renseigner sur les avantages et les inconvénients de chaque méthode de résolution.

L'expertise du laboratoire et de ses chercheurs est essentielle pour mener l'étude de ce problème, qui requiert des compétences en informatique théorique et en mathématique, et des connaissances sur la littérature autour de ce sujet et sur les méthodes de résolution les plus efficaces. L'amélioration du moteur de planification sera nécessaire pour résoudre les problèmes de planification plus difficiles, dans des établissements plus variées qui impliquent plus de contraintes ou tout simplement plus de variables à gérer (rendez-vous et ressources).

1.3.2 Verrous technologiques

La planification dans le milieu médical amène plusieurs défis qui devront être surmontés au cours du projet LORH. Ils sont de natures différentes, et sont liés à la fois au problème de planification lui-même, à des facteurs exterieurs concernant les interactions entre la méthode de résolution et le système d'information de l'établissement ainsi que sur des facteurs humains quant à l'évaluation de la qualité d'une planification.

1.3.2.1 Complexité de la planification

La planification des rendez-vous en milieu hospitalier implique de nombreux éléments à considérer : les actes médicaux à réaliser, le personnel et le matériel, les locaux, les patients et les diverses contraintes à prendre en compte. Nous l'évoquions dans les sections précédentes, ce travail est humainement difficile à réaliser. De par sa complexité, c'est aussi un problème difficile à résoudre pour une machine.

Ces problèmes de planification sont en général définis par de nombreuses variables, des contraintes à respecter et une fonction permettant d'évaluer la qualité d'une solution. Construire une solution consiste à affecter des valeurs à ces variables, et une solution est dite réalisable si les valeurs attribuées respectent les contraintes posées. Pour notre problème, cela signifie généralement attribuer des dates de début et un ensemble de ressources aux actes médicaux qui sont à planifier.

Notre problème de planification en milieu hospitalier, et plus largement les problèmes de planification sous contraintes font partie des problèmes d'optimisation combinatoire. Le terme "combinatoire" évoque le très grand nombre de solutions possibles au problème posé, tandis que le terme "optimisation" signifie que l'objectif est de trouver la ou l'une des meilleures solutions parmi l'ensemble des solutions réalisables. Dans notre cas, il s'agit du meilleur planning possible, dont la qualité est évaluée par la fonction évoquée plus tôt. Même pour des problèmes de petites taille n'impliquant qu'une dizaine de rendez-vous et de ressources, le nombre de solutions (combinaisons d'affectations de variables) s'avère considérable et peut nécessiter des heures de calcul. Si on souhaite obtenir la solution optimale sur des problèmes plus complexes, ces quelques heures se transforment rapidement en jours, en mois, en années, en siècles.

La résolution de ces problèmes fait l'objet de nombreuses recherches depuis des décennies, mais il n'existe néanmoins toujours pas d'algorithme permettant de résoudre de manière optimale et en un temps raisonnable ce type de problématique. Bien sûr, des méthodes permettant d'obtenir des résultats satisfaisants en un temps acceptable existent. Ces méthodes, appelées heuristiques, sont construites sur des méthodes empiriques et permettent de réduire considérablement le nombre de solutions explorées. Ce faisant, elles n'offrent pas la garantie d'obtenir une solution optimale (ou même réalisable), mais réduisent drastiquement le temps de calcul nécessaire à l'obtention d'une solution acceptable. Pour qu'elles soient exploitables pour une utilisation en condition réelle, il faut donc établir des stratégies d'exclusion de sous-ensembles de solutions suffisamment intelligentes pour réduire considérablement le temps d'exécution tout en évitant d'éliminer les solutions intéressantes.

La création et l'implémentation d'une méthode de résolution nécessite un savoir-faire et un travail de recherche conséquent. Pour cette raison, Evolucare a confié cette tâche a sa division Evolucare Labs. L'équipe s'est ensuite associée au laboratoire MIS dans le cadre de cette thèse pour résoudre ce problème complexe. Les difficultés présentées et les solutions apportées seront donc les principaux thèmes abordés dans ce manuscrit.

1.3.2.2 Acquisition des données

La planification nécessite des données de sources différentes qu'il peut être difficile de réunir au sein d'un unique logiciel. Selon les établissements, ces informations se retrouvent souvent dispersées entre plusieurs logiciels de gestion ou, encore pire, n'existent tout simplement pas sous format numérique. Ces données concernent les différents acteurs du monde médical :

— Concernant les patients :

  1. les prescriptions de soins et d'actes médicaux ;
  2. les plans de soin associés aux pathologies ;
  3. les préférences émises par le patient et les souhaits de sa famille ;

— Concernant le personnel soignant et administratif :

  1. son domaine de compétence ;

2. ses disponibilités ;
3. ses préférences ;

— Concernant les locaux et le matériel :

1. ses fonctions ;
2. ses caractéristiques ;
3. ses capacités ;
4. ses disponibilités ;

La plupart de ces informations sont présentes dans les systèmes d'information des établissements, et sont potentiellement déjà prises en charge par des logiciels Evolucare, s'ils font partie des clients de l'entreprise. Certaines de ces données sont statiques : la liste des patients, intervenants, matériels et locaux, ainsi que la descriptions de leur caractéristiques ne changent pas, ou peu. D'autres informations sont dynamiques et peuvent évoluer en permanence : le planning des intervenants, leurs disponibilités et les séjours ou mouvements auxquels ils sont sujets. En plus des données concernant les protagonistes du milieu de la santé, le module d'optimisation de la planification a aussi besoin des données définissant le problème à résoudre :

— Concernant le planning :

1. l'ensemble des rendez-vous déjà fixés, avec les ressources utilisées ;
2. l'ensemble des rendez-vous à planifier, avec toutes les informations nécessaires à la planification : les ressources nécessaires, sa priorité et ses liens éventuels avec d'autres rendez-vous ;

— Concernant les contraintes :

1. les contraintes liées à certaines ressources ;
2. les souhaits du patient, concernant son médecin ou les horaires de l'acte ;
3. les préférences du personnel soignant ;
4. d'autres contraintes, définies par l'utilisateur

Les informations concernant le planning sont bien souvent déjà présentes dans les logiciels de gestion des établissements. En ce qui concerne les contraintes, l'absence de logiciel capable de planification complexe a rendu inutile la création et le stockage des données permettant de les décrire. Pour remédier à cela, le module d'optimisation

devra proposer une interface permettant de configurer ces contraintes, pour qu'elles puissent ensuite être transmises au moteur de planification.

La conception de cette interface est un travail délicat, car les planificateurs amenés à l'utiliser ne seront pas forcément familiers avec des formulations complexes de contraintes mathématiques pour la résolution d'un problème informatique tout aussi complexe. L'interface devra être suffisamment simple à appréhender mais devra aussi permettre de formuler des contraintes élaborées qui correspondront aux réalités de terrain auxquelles ils feront face.

La quantité de données et la diversité de leurs sources représentent un véritable défi en terme d'ergonomie, d'architecture logiciel et d'interopérabilité. Le logiciel LORH doit être capable de communiquer avec les systèmes d'information divers et variés présents dans les établissements de soin, car il n'est pas envisageable d'imposer comme pré-requis l'utilisation des logiciels de la gamme Evolucare. Ce travail portant sur l'interopérabilité est capital pour l'utilisation de notre moteur, mais également complexe, et nécessite donc un investissement important en temps et en ressource pour être mené à bien. L'élaboration de l'interface est tout aussi délicate, car elle se doit d'être suffisamment claire et intuitive pour permettre à des utilisateurs qui ne sont pas des spécialistes en informatique (théorique ou pratique) de créer nos problèmes de planification en définissant toutes ses caractéristiques. Cette interface doit également permettre de visualiser et d'agir sur le planning, pour placer manuellement un ou plusieurs rendez-vous et corriger les solutions renvoyées par le module de planification.

Pour répondre à ces défis, l'une des idées envisagées est de développer un module « agenda » qui se grefferait au logiciel LORH et qui permettrait d'une part de récupérer et centraliser toutes les données du problème (ressources, disponibilités, actes médicaux, etc.) et d'autre part de les afficher sur un calendrier qui donnerait la possibilité aux utilisateurs d'agir à leur guise sur le planning. Ils auraient ainsi la possibilité d'ajouter à la main certains actes médicaux et / ou de modifier des solutions renvoyées par le module de planification. Cette partie du logiciel proposerait aussi les interfaces permettant de configurer les contraintes et d'ajouter les demandes d'actes médicaux à traiter. Une fois les contraintes définies, le module de planification pourrait alors traiter les demandes de rendez-vous en attente de manière automatique.

Ces deux modules ; le module agenda et le module de planification ; pourront être installés et utilisés ensemble plus rapidement et plus simplement qu'en tentant de rendre le moteur de planification seul utilisable pour tant de systèmes d'information différents.

Si toutes ces considérations d'acquisition des données et d'ergonomie ne sont pas les problématiques principales abordées dans ce manuscrit, elles représentent néanmoins des obstacles majeurs pour l'installation et l'utilisation en situation réelle du logiciel LORH.

1.3.2.3 Qualité d'une planification

Le nombre de solutions possibles pour un problème de planification est généralement très important, comme nous l'évoquions précédemment. Pour discriminer les solutions obtenues pendant la recherche et orienter nos efforts vers les planifications les plus prometteuses, nous devons définir une fonction objective capable d'évaluer la qualité des plannings obtenus. Si cette fonction est mal définie, alors des solutions de bonne qualité ne seront pas reconnues et des plannings de mauvaise qualité seront éventuellement proposés aux planificateurs.

Cependant, définir une telle fonction est un travail délicat. Il est difficile pour les planificateurs eux-même de définir tous les critères qui leur permettent de juger de la qualité d'une planification. Ce jugement repose effectivement sur des éléments concrets : le nombre d'actes programmés, le respect de certaines contraintes, le respect des souhaits du patient. Mais une partie de la décision reste difficile à expliquer car elle repose sur l'expérience du planificateur, son instinct. Nous parlons d'expérience dans ce travail de planification, mais aussi d'expérience au sein d'un établissement donné, d'un service, avec ses propres règles et coutumes. Les façons de construire les plannings diffèrent selon les établissements et les services, et principalement selon le type de soin qu'on y prodigue : les blocs opératoires ne fonctionnent pas de la même manière qu'un centre de Soins de Suite et de Réadaptation. Ainsi, deux planificateurs ne vont pas forcément faire les mêmes choix lors de la construction de leur planning.

Pour notre travail de recherche, nous nous sommes appuyés sur des critères et des contraintes remontés par les collaborateurs d'Evolucare en contact avec des

planificateurs dans plusieurs établissements clients de l'entreprise en France. Ils nous serviront de bases solides pour définir la fonction objective adaptée à notre problème. Cependant, il n'est pas raisonnable de croire que ces retours sont exhaustifs et seront directement applicables à tous les problèmes de planification pouvant survenir dans le milieu médical, pour les raisons citées ci-dessus. Ainsi, au delà du travail effectué lors de cette thèse, il sera nécessaire à l'avenir de laisser la possibilité aux utilisateurs de définir leurs propres critères et leurs propres contraintes.

De plus, les planificateurs devront pouvoir vérifier et modifier simplement une solution qui leur paraît satisfaisante mais qu'ils pourraient améliorer. Cela reste une tâche pénible, car pour vérifier une solution ou y apporter des modifications, il faut parcourir les emplois du temps de toutes les ressources impliquées. Ce travail peut se montrer rébarbatif et être un frein dans l'adoption d'un tel outil, car les utilisateurs en attendront des solutions de bonne qualité. Encore une fois, il s'agit d'un travail conséquent sur l'interface du logiciel, pour permettre aux planificateurs de modifier directement un planning fourni par le moteur de planification.

1.3.3 Opportunités

Le projet LORH doit surmonter un certain nombre de défis, mais bénéficie également de facteurs favorisant sa mise sur le marché et son adoption par les clients. Nous avons déjà évoqué les besoins croissants en matière de soins et la nécessité d'en améliorer la qualité, et les investissements importants prévus pour atteindre ces objectifs. Ces deux opportunités favorisent l'émergence de nouvelles solutions et logiciels tels que le projet LORH pour permettre une gestion optimisée des établissements et des services qui les composent.

Les collaborateurs d'Evolucare en contact avec les clients (commerciaux, formateurs, installateurs, etc.) remontent régulièrement des demandes pour le développement d'un outil permettant la planification des différents rendez-vous dans les établissements. Les méthodes actuelles qui consistent à gérer les rendez-vous un par un restent difficiles à utiliser pour en planifier un grand nombre, surtout en composant avec des contraintes complexes. Par ailleurs, le recrutement du personnel planificateur est problématique, car il n'y a pas de formation ou de diplômes pour

cette tâche. Ils sont recrutés bien souvent en interne, en fonction de leur expérience, leur connaissance du fonctionnement de l'établissement et leurs facultés logistiques.

Nous faisions état de la complexité des problèmes de planification dans le domaine de la santé dans les sections précédentes. C'est en effet un problème impliquant de très nombreuses variables et contraintes qu'il est difficile de d'appréhender dans leur globalité pour un humain. Cette complexité risque de s'accroître avec le changement de l'organisation géographique des soins prévu par la loi de santé 2016 et la mise en place des Groupements Hospitalier de Territoires (GHT). Ce dispositif a pour but d'inciter les établissements de santé d'un même territoire à mutualiser les équipes médicales et les structures existantes. Plus concrètement, cela encourage à rassembler les petites unités médicales, qui ne sont pas rentables ou peu utilisées en services de plus grande taille. Ce changement d'organisation a un impact sur la planification et sa complexité : l'augmentation des tailles des services complique encore plus la tâche déjà difficile des planificateurs, avec encore plus d'actes médicaux à prévoir, avec plus de ressources potentielles. Par conséquent, le besoin d'un outil performant de planification se fera de plus en plus pressant au fil de l'application de ce dispositif, stimulant ainsi la demande et augmentant l'attractivité du marché.

1.3.4 Étude de la concurrence

Avant d'entreprendre la création d'un logiciel de planification, Evolucare a étudié la concurrence et les logiciels existants. Certains acteurs, qu'ils soient du domaine médical ou pas, ont déjà développé des outils de planification. Nous pouvons citer le logiciel Planning, spécialisé dans l'hospitalisation de jour et plus spécifiquement l'oncologie. Dans le domaine médical, l'entreprise Softway propose aussi un outil de planification. Dassault Systems et sa suite Quintic proposent des solutions de planification automatique de diverses opérations métier, plutôt orienté vers des problèmes logistiques. Également connus dans le domaine de la recherche opérationnelle, IBM et sa suite CPLEX et LocalSolver proposent des solutions « clés en main » pour résoudre ce genre de problèmes d'optimisation combinatoire.

Lorsqu'il est question de planification de rendez-vous, l'une des solutions les plus populaire est Doctolib. Cet outil permet en effet à un particulier de prendre un rendez-vous avec un médecin, ou même à un médecin d'organiser un autre rendez-

vous entre un praticien et son patient. C'est un point d'entrée dans les établissements, mais la planification des actes médicaux qui constituent le parcours de soins du patient reste du ressort des planificateurs. À notre connaissance, Doctolib ne propose pas aux établissements médiaux une planification optimale des rendez-vous qu'ils doivent organiser. Or, c'est cet objectif que nous poursuivons avec notre logiciel de planification.

Le projet LORH vise en effet à gérer toute la planification des actes au sein des établissements médicaux, afin de faciliter le travail des équipes soignantes et d'améliorer la qualité des soins apportés aux patients à travers des plannings plus efficaces, plus respectueux des souhaits de tous les acteurs impliqués. Étant intégré à la suite logicielle Evolucare (dossier patient, personnel et spécialités, structure de l'établissement, etc.), il permettra une gestion globale des rendez-vous. Pour les établissements n'utilisant pas toute la suite de logicielle, l'expérience du service interopérabilité permettra de récupérer les informations des logiciels des éditeurs tiers. L'ambition du projet est de gérer la planification des établissements médicaux dans leur ensemble, sans avoir à séparer les services.

Conclusion

Dans ce chapitre, nous avons présenté l'entreprise partenaire de cette thèse CIFRE et l'outil qu'elle souhaite développer pour résoudre des problèmes de planifications complexes. Evolucare Technologies profite d'une longue expérience dans le développement logiciel pour les établissements de santé et d'une équipe innovation offrant un cadre adapté à l'encadrement d'une thèse CIFRE. Le contexte économique est lui aussi favorable au développement d'un tel logiciel, avec une consommation des services de soins en hausse et des investissements importants à venir dans le système de santé pour en améliorer la qualité de service.

Les problèmes de planification des actes médicaux et autres rendez-vous sont des problème persistants et difficiles qui nécessitent des heures de travail pour les planificateurs. Nous avons présenté les nombreuses contraintes et variables à prendre en compte, le temps et les efforts fournis par les équipes pour construire ces emplois du temps et le peu d'outils dont ils disposent pour résoudre ces problèmes complexes.

Enfin, nous avons introduit le projet LORH, qui incarne la réponse d'Evolucare à ces problèmes de planification. Les difficultés à surmonter sont nombreuses et sont à la fois d'ordre technique et scientifique, nécessitant donc à la fois les connaissances métier de l'entreprise et l'expertise des chercheurs dans la recherche opérationnelle. Nous avons également présenté quelques cas concrets qui nous permettront de construire des instances, afin de tester les différentes méthodes que nous présenterons dans ce manuscrit.

Chapitre

2

Domaine de l'étude

Contents

= 2.1 Optimisation combinatoire

2.1.1 Définition

2.1.2 Complexité

2.2 Problèmes de planification dans le domaine de la santé

2.2.1 Généralités

2.2.2 Types de problèmes

2.3 Resource Constraint Project Scheduling Problem

2.3.1 Définition

2.3.2 Extensions du RCPSP

2.3.3 Méthodes de résolution

Dans ce chapitre nous décrivons le domaine de l'étude menée au cours de cette thèse. Nous commencerons par préciser la complexité de ce problème d'optimisation combinatoire. Puis nous situerons notre problématique parmi les nombreux problèmes de planification dans le domaine de la santé. Enfin, nous décrirons la formulation choisie et les méthodes de résolution existantes pour les problèmes de planification sous contraintes de ressources.

2.1 Optimisation combinatoire

Le problème de planification sous contraintes que nous étudions dans ce manuscrit fait partie des plus complexes existant dans le domaine de l'informatique. Ces

problèmes relèvent du domaine de l'optimisation combinatoire, qui réunit de nombreux chercheurs autour des mêmes objectifs : étudier les propriétés des problèmes, les formaliser et proposer des algorithmes efficaces pour les résoudre.

2.1.1 Définition

La planification sous contraintes fait partie de la famille des problèmes d'optimisation combinatoire. Ces problèmes sont présents dans de nombreux domaines, parmi lesquels se trouve logiquement l'informatique, mais que l'on retrouve également dans d'autres domaines où ce dernier s'est invité au fil du temps : la biologie, les mathématiques, l'électronique, la finance, etc. En informatique, de nombreux problèmes très connus de la littérature appartiennent à cette famille et font l'objet de recherches approfondies depuis des décennies (Papadimitriou and Steiglitz, 1998, Wolsey and Nemhauser, 1999, Korte et al., 2011) : le voyageur de commerce, le problème SAT, la planification, la coloration de graphe, et bien d'autres.

Ces problèmes d'optimisation combinatoire consistent à affecter des valeurs à des variables qui, ensemble, vont constituer une solution à la problématique posée. L'objectif est de trouver parmi l'ensemble des solutions réalisables, la meilleure solution. Ces solutions sont distinguées par une fonction permettant d'évaluer leur qualité. Le terme *optimisation* est utilisé car il s'agit de minimiser (ou de maximiser) cette fonction communément appelée fonction objectif. Le nombre de solutions, généralement très élevé et augmentant exponentiellement avec le nombre de variables et de données à considérer, rend ces problèmes très difficiles à concevoir dans leur globalité par un cerveau humain. Nous donnons une définition plus formelle d'un problème d'optimisation combinatoire dans la définition 1 :

Définition 1 *Un problème d'optimisation combinatoire* (Υ, f) *est défini par :*

- *Un ensemble de variables* $X = \{x_1,...,x_n\}$ *et leur domaine de définition respectif* D_i *;*
- Υ*, l'espace de recherche défini par l'ensemble des valeurs* $(v_1,...,v_n) \mid v_i \in D_i$ *;*
- *Un ensemble de contraintes entre les variables ;*
- *Une fonction objectif* f *à minimiser ou à maximiser.*

L'objectif est de trouver l'ensemble de solutions optimales au problème posé. La qualité d'une solution et son optimalité sont déterminées par la fonction $f : \Upsilon \rightarrow \mathbf{R}$. Il s'agit donc de trouver une solution $i* \in \Upsilon$ telle que $f(i*) \leq f(i)$ pour tout $i \in \Upsilon$. Le coût optimal est représenté par $f* = f(i*)$ et l'ensemble des solutions optimales par $\Upsilon* = \{i \in \Upsilon \mid f(i) = f*\}$. Le terme *optimisation* fait ainsi référence à la recherche de cette solution ou de cet ensemble de solutions optimales.

Les solutions sont constituées des valeurs $(v_1,...,v_n) \mid v_i \in D_i$ qui ont été assignées aux variables $\{x_1,...,x_n\} \in X$. Une solution est dite réalisable si elle respecte les contraintes existantes entre les variables. Durant le processus de recherche et au fil des affectations des valeurs, des solutions non réalisables sont rencontrées. L'ensemble des combinaisons de valeurs possibles pour l'ensemble des variables de décision X représente l'espace de recherche Υ. C'est de ce nombre de combinaisons de valeurs à affecter, généralement très élevé, que provient le terme *combinatoire*.

Parmi les problèmes d'optimisation combinatoire, nombreux sont ceux à être qualifiés de problème de décisions (Hoos and Stützle, 2004).

Définition 2 *Un problème de décision est un problème où la réponse est soit vrai, soit faux.*

Ces problèmes de décision peuvent souvent être généralisés à des problèmes d'optimisation, où les solutions sont évaluées par une fonction objectif f selon les valeurs des variables de décisions et du respect des contraintes. Comme nous le disions précédemment, le but est alors de trouver la ou les meilleures solutions.

Il y a beaucoup d'exemples de tels problèmes, et nous pouvons reprendre l'un de ceux cités plus tôt : le problème de coloration de graphe. Ce problème se présente de la manière suivante : soit un graphe non-orienté $G = (V,E)$ avec V un ensemble de sommets et E un ensemble d'arêtes. Trouver une coloration valide de ce graphe consiste à associer à tout sommet $v \in V$ une couleur $c(v)$, en veillant à ce que $c(v) \neq c(u)$ pour toutes arêtes $|u,v| \in E$. Ainsi pour chaque sommet et pour chaque couleur possible, une variable de décision prendra la valeur *vrai* ou *faux* selon la couleur assignée au sommet. Il existe une variante à ce problème de coloration qui vise à limiter le nombre de couleurs utilisées. Dans cette variante, une fonction

objectif f évalue chaque solution et l'objectif est de trouver la solution utilisant le moins de couleurs possibles, tout en respectant la contrainte initiale du problème.

Les problèmes de planification appartiennent également à ces catégories. Il s'agit de problèmes d'optimisation combinatoire, où l'espace de recherche Υ est composé de toutes les combinaisons possibles d'arrangement des activités et d'affectation de ressources. En prenant le cas plus simple où il ne s'agit que de planifier les activités sans gérer les ressources, prenons un problème avec A l'ensemble des activités à planifier et H l'horizon. Il y a pour chaque activité $\{j_1,...,j_n\} \in J$ autant de variables de décision que d'horaires $\{t_1,...,t_n\} \in H$ possibles pour les planifier. Nous avons donc $x_t^a = vrai$ si l'activité $a \in A$ débute à l'horaire $t \in H$. Pour certains problèmes de planification, l'objectif est de planifier les activités en réduisant au minimum leur temps total d'exécution, tandis que pour d'autres il s'agit de réduire l'utilisation de certaines ressources au minimum. Rien que sur ce problème assez simple, le nombre de combinaisons explose rapidement avec le nombre d'activités à traiter et d'horaires possibles.

2.1.2 Complexité

Les problèmes d'optimisation combinatoire figurent parmi les plus complexes à résoudre pour les informaticiens. Les plus difficiles ne sont toujours pas considérés comme résolus, et aucun algorithme s'exécutant en temps polynomial n'existe encore à l'heure actuelle pour ces problèmes difficiles. L'étude de la complexité théorique des algorithmes (Garey and Johnson, 1979) est un domaine de la recherche ancien mais toujours actif qui vise à approfondir les connaissances sur ce qui rend certains problèmes très difficiles, et qui permet donc de les classer selon leur complexité.

Il est effectivement nécessaire lorsque l'on commence l'étude d'un problème de savoir à quel point il est compliqué à résoudre, et quelle est l'efficacité des algorithmes élaborés jusqu'à maintenant. Les chercheurs et les scientifiques qui abordent un problème ont besoin de définitions plus précises et complètes de la complexité du problème et de l'efficience des algorithmes pour orienter leurs recherches vers les pistes les plus prometteuses. La distinction la plus répandue parmi les informaticiens pour distinguer les algorithmes est celle qui existe entre les algorithmes qui s'exécutent en un temps polynomial et ceux s'exécutant en un temps exponentiel.

Les premiers sont largement préférés aux seconds, en particulier sur les plus grosses instances, où les temps d'exécution des algorithmes en temps exponentiel deviennent colossaux et les rendent peu applicables.

La machine de Turing, définie par Alan Turing dans Turing et al. (1936) est une machine abstraite représentant le fonctionnement des appareils mécaniques de calcul. Cette machine est composée d'un ruban représentant une mémoire infinie, et d'une tête de lecture et d'écriture permettant de lire les symboles sur le ruban et de les changer. Les déplacements de cette tête de lecture sont définis par un ensemble fini de règles. La machine de Turing, capable de résoudre certains problèmes simples, est un composant fondamental des travaux sur la théorie de la complexité informatique. La définition formelle 3 d'une machine de Turing est donnée dans l'article de Lewis and Papadimitriou (1998).

Définition 3 *Une machine de Turing est un quintuplet* (K,Σ,δ,s,H)*, où :*

- K *est un ensemble fini d'états ;*
- Σ *est un alphabet de symboles comportant les symboles blanc* $\sqcup$ *et le symbole d'extrémité gauche* $\triangleright$ *;*
- $s \in K$ *l'état initial ;*
- $H \subseteq K$ *un ensemble ;*
- δ*, une fonction de transition de* $(K-H)\times\Sigma$ *vers* $K\times(\Sigma\cup\leftarrow,\rightarrow,-$*, avec* $\leftarrow$ *et* $\rightarrow$ *les instructions de déplacement de la tête de lecture/écriture vers la gauche et la droite et* $-$ *une instruction de non-déplacement ;*

La définition 3 décrit une machine de Turing déterministe, où l'application d'une instruction sur un état mène toujours au même résultat. Dans une machine de Turing non-déterministe, il est possible d'obtenir plusieurs états différents en appliquant une instruction sur un état donné. La machine de Turing nous permet de décrire la notion de complexité temporelle : elle est représentée par la fonction $t : \mathbb{N} \rightarrow \mathbb{N}$, où $t(n)$ représente le nombre d'étapes de calculs qu'il faut à une machine de Turing pour résoudre un problème. Avec cette notion, nous pouvons définir ce qu'est un temps polynomial :

Définition 4 *Un problème de taille* n *est décidable en un temps polynomial si sa complexité temporelle est* $O(t(n))$*.*

Les définitions 3 et 4 d'une machine de Turing et du temps de résolution polynomial permettent de définir les classes de complexité P et NP.

Définition 5 *Un problème de décision* Π *appartient à la classe* P *s'il peut être résolu par une machine de Turing déterministe en temps polynomial.*

Définition 6 *Un problème de décision* Π *appartient à la classe* NP *s'il est possible de vérifier une solution à ce problème avec une machine de Turing déterministe en temps polynomial.*

Cette notion de temps polynomial est utilisée pour définir deux classes de complexité : la classe de complexité polynomiale P et la classe non-déterministe polynomiale NP. Les problèmes appartenant à la classe P peuvent être décidés rapidement relativement à leur taille et sont considérés comme étant plus simples. Les problèmes appartenant à la classe P sont inclus dans la classe NP, car s'il est possible d'obtenir en temps polynomial une solution au problème, il est aussi possible de vérifier la solution en résolvant le problème. L'une des questions les plus fondamentale dans le domaine des mathématiques et de l'informatique est de savoir si $P = NP$. Si tel était le cas, cela signifierait que les problèmes les plus complexes peuvent être résolus en temps polynomial. Cette question est toujours source de nombreuses recherches et débats à l'heure actuelle.

Définition 7 *Un problème de décision* Π *est dit NP-complet si* $\Pi \in NP$ *et pour tout autre problème* $\Pi' \in NP$, $\Pi' \propto \Pi$.

La classe de complexité NP contient des problèmes réputés pour être les plus difficiles, nommés problèmes NP-complet. Ces problèmes appartiennent à NP mais pas à P, et il n'existe pas d'algorithme de complexité polynomial capable de les résoudre. Si ne serait-ce qu'un problème de décision NP-complet pouvait être résolu en temps polynomial, alors tous les problèmes appartenant à NP pourraient l'être également. Il y a un peu plus de cinquante ans, le premier problème a avoir été démontré NP-complet est le problème de satisfaisabilité (SAT), par Stephen Cook dans (Cook, 1971).

Définition 8 *Un problème* Π *est dit NP-difficile s'il existe un problème* Π' *NP-complet tel que* $\Pi' \propto \Pi$.

La dernière classe de complexité que nous introduirons dans ce chapitre est la classe des problèmes *NP*-difficiles. Ces problèmes, qui n'appartiennent pas systématiquement à *NP* sont au moins aussi difficiles que les problèmes les plus durs existants dans *NP*.

La définition de ces classes de complexité est importante dans le cadre de ce manuscrit car le problème de planification que nous traitons fait partie des problèmes *NP*-difficiles. Il n'existe aujourd'hui aucun algorithme permettant de les résoudre en un temps polynomial. Autrement dit, ce sont des problèmes parmi les plus compliqués qui existent dans le domaine de l'informatique, et sont le sujet de nombreuses recherches depuis des années.

2.2 Problèmes de planification dans le domaine de la santé

Les problèmes de planification, d'attribution de ressources et d'ordonnancement sont étudiés depuis de longues années et font partie des problématiques classiques du domaine informatique. Les problèmes de planification dans le domaine de la santé bénéficient quant à eux d'une attention particulière depuis une quinzaine d'années. Dans un récent passage en revue de la littérature scientifique sur ces thématiques (Abdalkareem et al., 2021), les auteurs dénombrent quelques 190 articles autour de ces thématiques entre 2010 et 2020. Ces derniers se concentrent principalement sur quelques sujets, comme les admissions de patients, la planification des blocs opératoires et les emplois du temps des infirmières et infirmiers. L'objectif de cette sous-section est de positionner la problématique étudiée dans ce manuscrit par rapport à la littérature.

2.2.1 Généralités

L'optimisation des processus de planification au sein des différents établissements de santé a pour objectif premier de faciliter l'accès des patients aux soins dont ils ont besoin, et d'aider le personnel dans son rôle de soignant. Dans un contexte économique et sanitaire difficile, la réduction des coûts et l'optimisation de l'utilisation des ressources est un enjeu essentiel pour les systèmes de soins de nombreux pays. Dans leur étude des nombreux problèmes de planification dans ce milieu (Hall

et al., 2012), les auteurs avancent d'autres arguments pour expliquer l'origine des différentes problématiques d'organisation qui existent dans ce milieu. L'un de ces arguments revient sur le manque de sensibilisation des équipes soignantes à l'optimisation et à la gestion des ressources. C'est évidemment compréhensible, nous l'évoquions dans le chapitre précédent, le ou les soignants qui s'occupent de ces tâches de planification n'y ont pas été formés au préalable. Ils ne disposent pas non plus d'outils qui leur permettraient justement de produire des solutions plus qualitatives à la fois pour eux et pour les patients.

Un autre élément qui participe à la difficulté de ces problèmes provient des activités à planifier, de nature très différentes de celles que nous pouvons retrouver dans les usines, par exemple. D'une part, les rendez-vous et les actes médicaux n'ont pas toujours la même durée. Le temps pris pour réaliser divers types d'actes (chirurgicaux, de rééducation ou une consultation classique, etc.) varie d'un patient à l'autre, en raison des différences qui existent entre les individus (différences génétiques, démographiques, symptomatiques) et l'aspect humain qui est omniprésent dans ce domaine d'activité. Des retards peuvent donc survenir même si le planning est parfait. Des recherches visent justement à minimiser ces retards et les temps d'attentes des patients pour accéder à leur rendez-vous (Zhu et al., 2020, Granja et al., 2014). Une autre particularité des rendez-vous est la présence obligatoire d'un patient dans la grande majorité des actes à planifier. Ces actes sont prévus pour le patient, et tout impondérable qui le touche se répercute directement sur son emploi du temps, et ceux de toutes les ressources impliquées.

2.2.1.1 Ressources

Pour tous les problèmes de planification dans le milieu médical, le but est de faire coïncider les besoins des patients avec les ressources disponibles. Il y a de nombreux types de ressources (Hall, 2012), et le choix des ressources à considérer et leur représentation (qui se veut le plus proche de la réalité possible) conditionne en partie la complexité du problème traité.

La gestion de l'emploi du temps des médecins et des différents spécialistes représente la majeure partie des efforts scientifiques menés dans la planification en milieu médical (Cardoen and Demeulemeester, 2008, Chern et al., 2008). Dans certains cas,

il s'agit d'organiser un ensemble de rendez-vous entre des praticiens et des patients. Parfois, il s'agit de planifier les vacations de ces spécialistes, qui détermineront les tâches qu'ils vont pouvoir accomplir pendant une période de temps donnée. Une partie importante de la littérature traite en particulier les emplois du temps des infirmiers (Liang et al., 2015, Burke et al., 2004).

Les locaux où prennent places les consultations ou les opérations font parfois partie des ressources à prendre en compte. C'est particulièrement vrai pour les problèmes de gestion de blocs opératoires, où les salles et les équipements ont un rôle important et sont soumis à des contraintes spécifiques. Les établissements hospitaliers de grande taille sont souvent organisés en services, qui contiennent des salles spécialisées pour certains types d'actes : ophtalmologie, traumatologie, pédiatrie, etc. Par conséquent, certains actes nécessitent des types de salle particuliers et ne peuvent avoir lieu que dans un service précis.

Les équipements peuvent être assignés aux salles de manière permanente ou temporaire. C'est le cas par exemple des machines à Imagerie par Résonance Magnétique (IRM), ou des équipements spécifiques des blocs opératoires. Certaines machines sont portables et peuvent être amenées jusqu'au patient quand il ne peut pas se déplacer. Ces ressources sont nécessaires pour certains actes médicaux mais ne sont pas toujours présentes dans le système d'information de l'établissement.

Les fournitures médicales, les médicaments, les instruments et les implants ou organes sont des ressources présentes dans le quotidien des établissements de santé. Elles peuvent être qualifiées de consommables et présentent une différence fondamentale les autres ressources présentées ci-dessus : elles ne sont généralement disponibles qu'en quantité limitée et sont consommées au cours des rendez-vous ou actes médicaux. Lorsqu'elles sont présentes dans le système d'information des établissements, seule la quantité disponible est indiquée pour chaque type de consommable. Ces ressources particulières ne sont pas présentes dans les problèmes que nous traitons dans ce manuscrit. Les planificateurs en contact avec Evolucare n'ont pas exprimé le besoin de gérer ces consommables qui ne sont pas leur priorité dans leurs problèmes de planification.

2.2.1.2 Niveaux de planification

La gestion des hôpitaux et autres établissements de santé peut être abordée selon des hauteurs et des angles différents. Dans notre problème de planification, nous traitons des demandes de rendez-vous sur des horizons de une ou deux semaines au maximum, qui sont donc considérés comme des décisions à court terme. Avant même de planifier les actes médicaux, des décisions sont prises quant à la stratégie d'un service ou d'un établissement : le nombre de soignants dans le service, les patients à prendre en charge, ou les moyens alloués pour tel ou tel service. Les différents niveaux de planification ont été proposés premièrement dans le domaine de la manufacture (Anthony, 1965) et ont été repris dans la littérature de la planification dans le milieu de la santé par la suite (Marynissen and Demeulemeester, 2016, Hans and Vanberkel, 2012).

Niveau stratégique Le niveau le plus haut de planification est le niveau stratégique, qui concerne les décisions structurelle sur le long terme. C'est à ce niveau qu'il est décidé combien de ressources sont allouées et où elles le sont, pour répondre aux besoins des patients. Il peut s'agir par exemple de définir les capacités d'accueil de patients d'un service en fonction des retours d'un système de prédiction (Bowers et al., 2005). D'autres exemples peuvent être trouvés dans la littérature (Cardoen and Demeulemeester, 2008, Froehle and Magazine, 2013).

Niveau tactique Le niveau tactique regroupe les décisions à moyen terme, qui s'étalent sur quelques semaines. En fonction des décisions prises au niveau stratégique, les demandes des patients sont prises en compte et leurs parcours de soins sont constitués. C'est aussi à ce niveau que les vacations du personnel sont établies, qui détermineront les types d'actes qu'ils pourront accomplir pendant des périodes de temps définies. C'est le cas par exemple dans cet article (Bikker et al., 2015), proposant d'optimiser la planification des vacations dans un service de radiothérapie.

Niveau opérationnel Ce dernier niveau de décision concerne toutes les décisions prises au fil des jours, et il peut être divisé en deux sous-catégorie. D'une part, la planification dite « online » est une planification au jour le jour, qui doit être réactive face à des évènements imprévisibles comme l'arrivée aux urgences d'un patient,

ou l'indisponibilité d'une ressource qui nécessite de reprogrammer un ou plusieurs rendez-vous. On peut prendre comme exemple le cas d'un service de radiologie aux États-Unis où les demandes sont gérées quand elles arrivent sans attendre d'avoir une série de rendez-vous pour les traiter (Azadeh et al., 2015), ou le cas de la gestion des rendez-vous de patients en soins ambulatoires au sein de plusieurs services différents (Vermeulen et al., 2008). D'autre part, la planification « offline » concerne la création de plannings pour un ensemble de rendez-vous avant le jour où ils ont lieu, sur un horizon d'une ou deux semaines. Les exemples que nous décrivions dans le chapitre précédent correspondent à cette définition, mais il existe d'autres exemples dans la littérature (Conforti et al., 2011, Du et al., 2013).

Le problème que nous traitons dans ce manuscrit se situe au niveau opérationnel, et les exemples que nous utilisons pour tester nos méthodes correspondent à des planifications « offline ». Cependant, la frontière entre les planifications « offline » et « online » reste fine, et le logiciel LORH devra être en mesure de fournir une réponse rapide et un nouvel emploi du temps en cas d'imprévus.

2.2.1.3 Types de patient

Les difficultés survenant lors de l'élaboration d'un emploi du temps dans ce domaine dépendent aussi du type des patients qui sont pris en charge. Dans les différents articles traitant de problème de planification dans le milieu hospitalier, trois catégories de patient reviennent fréquemment : les patients admis en soins ambulatoires, les patients admis en urgence et les patients hospitalisés.

Patients admis en urgence Les actes de soins pour les patients admis en urgence sont difficilement planifiables de par leur nature, et rentrent dans les aléas qu'il faut gérer par une planification dite « online » que nous décrivions plus tôt. La littérature sur les problèmes intégrant ce genre de patient se concentre sur la gestion des conséquences qu'ils ont sur tous les autres rendez-vous : il est généralement nécessaire de reporter certains actes pour traiter les patients urgents.

Patients hospitalisés Les patients hospitalisés sont des patients qui sont admis à temps plein au sein d'un établissement, pendant plusieurs jours. Par conséquent, ils ont besoin d'un lit (Conforti et al., 2011), ce qui ajoute une contrainte supplémen-

taire vis-à-vis du nombre de lits disponibles. Les recherches autour de ce type de patients se focalisent sur la durée de son parcours de soin, pour tenter soit de le réduire soit de mieux prendre en compte les souhaits du patient.

Patients en ambulatoire Les soins prodigués en ambulatoire sont de plus en plus fréquents avec les progrès de la médecine. Certaines opérations sont beaucoup moins lourdes qu'auparavant et permettent que les patients viennent à l'hôpital, subissent l'opération puis retournent chez eux. Ils ne passent donc pas la nuit au sein de l'établissement. Ces nouvelles pratiques entraînent des changements pour les hôpitaux, qui doivent gérer ces nouveaux types de patients avec potentiellement de nouveaux locaux et des créneaux spécifiques pour eux (Zonderland, 2014). Elles provoquent aussi des difficultés nouvelles. Les patients n'étant pas hospitalisés viennent pour un acte médical et repartent, rendant les retards plus gênants que pour des patients qui restent de toute façon au sein de l'établissement. Ils sont aussi plus susceptibles de ne pas venir pour leur rendez-vous, à cause d'un problème de transport ou simplement un oubli (Tsai and Teng, 2014).

2.2.2 Types de problèmes

La littérature sur les problèmes de planification dans le milieu médical est riche, et porte sur plusieurs types de problèmes différents. Les quelques uns que nous présentons dans cette sous-partie sont les plus étudiés de la communauté. Certains sont proches de notre problématique, tandis que d'autres ont des liens plus étroits. Nous souhaitons situer notre problème parmi ceux souvent repris dans la littérature.

2.2.2.1 Emploi du temps des infirmières

L'un des sujets les plus étudié de la littérature autour de la planification dans le milieu médical est celui des emplois du temps des infirmiers, ou « Nurse Rostering Problem » (NRP). Ce problème *NP*-difficile consiste à assigner des vacations aux infirmiers et infirmières ayant des compétences diverses. Les infirmiers effectuent généralement une garde par jour, et il s'agit donc de décider lesquels auront des gardes en journée, en soirée ou durant la nuit. En raison des pénuries d'infirmiers qui affectent de nombreux pays (Booth, 2002) et de l'impact que cela peut avoir sur les patients (Aiken et al., 2002), l'optimisation de leurs emplois du temps a pris

une importance cruciale pour les institutions et a attisé la curiosité des chercheurs au cours des dernières décennies (Rajeswari et al., 2017, Haspeslagh et al., 2014, Ceschia et al., 2020).

Pour résoudre ce problème, de nombreuses approches ont été utilisées. Parmi elles, beaucoup d'heuristiques et de métaheuristiques : des méthodes de recherches locales telles que le recuit simulé (Abobaker et al., 2011) et ou recherches locales à voisinage large (Ceschia et al., 2020), des algorithmes à colonie de fourmis (Wu et al., 2013) ou d'abeilles (Rajeswari et al., 2017), des algorithmes évolutionnaires hybrides (Zhuo et al., 2015) ou encore d'autres solutions hybrides mêlant méthodes exactes et métaheuristiques (Dang et al., 2016, Mischek and Musliu, 2019).

Le problème que nous étudions est différent de ce problème d'emploi du temps. Dans les exemples que nous utilisons pour éprouver nos approches, certaines ressources sont bien des infirmiers ou infirmières diplômés d'État (IDE), mais nous ne gérons pas les affectations de leurs vacations et de leurs gardes. C'est en revanche une donnée dont nous disposons pour les problèmes que nous tentons de résoudre, et ces vacations correspondent à des périodes de disponibilités durant lesquelles les IDE peuvent participer à certains types de rendez-vous.

2.2.2.2 Gestion des blocs opératoires

Un autre sujet concentrant une bonne partie de l'attention des chercheurs est la gestion des blocs opératoires. La planification des actes chirurgicaux est encore une fois une tâche ardue, qui implique de nombreuses ressources (le personnel soignant, les salles et les équipements nécessaires aux opérations) et contraintes (souhaits du personnel et des patients). Le bloc opératoire représente un poste de dépense important pour l'hôpital (Denton et al., 2007), et de nouvelles méthodes de gestions sont nécessaires pour réduire les coûts et améliorer à la fois la qualité des soins apportés aux patients et la qualité de vie au travail du personnel chirurgical.

La gestion des blocs opératoires représente en réalité plusieurs étapes dans le parcours de soins d'un patient : la partie pré-opératoire où le patient est préparé à l'opération, l'opération elle-même qui nécessite de nombreuses ressources et équipements et enfin les soins post-opératoires (Pariente, 2016, Guerriero and Guido, 2011). La gestion d'un bloc implique donc de prendre en compte de nombreux types

de ressources : chirurgiens, anesthésistes, IDE, machines, etc. Cette gestion peut être classée en différentes catégories : en fonction du niveau de planification que nous évoquions plus tôt, ou en fonction des politiques de gestion des rendez-vous (Van Huele and Vanhoucke, 2014).

La littérature sur ce sujet est riche et un nombre conséquent d'articles récents a été consacré à ce problème. Par conséquent, beaucoup de méthodes de résolution ont été étudiées. Parmi elles nous pouvons à nouveau citer les recherches locales (Ceschia and Schaerf, 2016, Mateus et al., 2018) et les colonies de fourmis (Xiang, 2017). De nombreux articles mettent en avant l'utilisation d'algorithmes génétiques classiques (Lee and Yih, 2012, Timuçin and Biroğul, 2019, Timucin and Birogul, 2018) ou hybridés avec d'autres méthodes comme l'optimisation par essaims particulaires (Particle Swarm Optimization, PSO) (Ansarifar et al., 2018) ou des recherches locales (Lin and Chou, 2020).

Ce problème de gestion des blocs opératoires est par bien des aspects similaires à notre problème. Le principe est identique : il s'agit de planifier un certain nombre de rendez-vous et d'actes médicaux tout en respectant les différentes contraintes du problème. Les principales différences se situent dans le nombre et le type de ressources à gérer, et dans les contraintes prises en compte. L'un des cas réels dont nous nous sommes inspirés pour créer un de nos scénarios était justement le cas d'un bloc opératoire. Néanmoins, les problèmes traités dans ce domaine sont soumis à de plus nombreuses contraintes, que nous avons relâché pour rester homogènes avec les autres scénarios. Le but de l'entreprise est de proposer un outil capable de résoudre un large panel de problèmes différents. L'intégration de contraintes spécifiques à ces problèmes se fera dans un second temps.

2.2.2.3 Admission de patients et gestion de lits

Les problèmes concernant l'admission des patients au sein des établissements et la gestion des lits qui en découle est un sujet important au sein de la littérature de la planification en milieu médical. L'objectif est d'assigner les patients aux chambres et plus particulièrement aux lits disponibles dans les services adéquats et pour tout le séjour du patient. Avec une meilleure gestion de ces lits, les établissements cherchent à optimiser l'utilisation de leurs lits, pour faire mieux avec les moyens dont ils

disposent. C'est également un moyen de fournir un service de meilleur qualité aux patients, en prenant en compte leurs désirs et en limitant des désagréments tels que des transferts d'un service à l'autre à cause d'un manque de lits. Une formulation de ce problème peut être trouvée dans un article de Demeester *et al.* (Demeester et al., 2010).

Cette gestion des lits et des admissions se fait à plusieurs niveaux de décisions. Sur le long terme, il s'agit de déterminer le nombre de lits attribués à l'établissement, puis aux différents services en fonction des besoins de certains soins. Une mauvaise estimation peut mener à des déséquilibres de nombre de lits d'un service à l'autre. Sur le moyen terme, il faut traiter les demandes d'admission des patients et les accepter ou non en fonction des capacités de l'établissement. Enfin, sur le court terme, les admissions et les sorties doivent être planifiées et les patients assignés aux lits en fonction de leur pathologie et de leurs souhaits. Différentes versions de ce problème ont été proposées au fil du temps (Ceschia and Schaerf, 2012), en intégrant des incertitudes liées aux patients admis en urgence ou aux durées des séjours.

De nombreuses approches ont été proposées pour résoudre ces problèmes. Dans l'article que nous citions précédemment (Demeester et al., 2010), les auteurs présentent une approche exacte par programmation linéaire en nombre entier et une recherche tabou hybride, efficaces sur des données générées. Une autre recherche locale, le recuit simulé, s'est également montrée efficace sur ce problème (Ceschia and Schaerf, 2011), notamment sur les instances crées par Demeester. D'autres métaheuristiques ont également été utilisées : un algorithme de biogéographie (Hammouri and Alrifai, 2014), reproduisant le comportement de migration des espèces et une recherche locale adaptative à voisinages larges (Lusby et al., 2016) qui se sont montrées efficaces sur des données générées ayant des horizons à tailles variables.

Sur le fond, les problèmes de gestion de lits ne sont pas fondamentalement différents des problèmes que nous traitons. Il s'agit d'attribuer des ressources pendant un temps déterminé en respectant les différentes contraintes liées au problème. Cependant, ils ne font pas partie des problèmes qui nous ont été soumis par les planificateurs en contact avec Evolucare, qui sont beaucoup plus tournés vers la planification des rendez-vous et actes médicaux.

2.2.2.4 Planification du parcours patient

La planification du parcours patient se concentre sur la planification des différents actes médicaux et rendez-vous qui sont nécessaires au traitement d'un patient. Cela implique souvent plusieurs disciplines de la médecine, avec des actes exécutés dans différents services. Cette multi-disciplinarité peut se heurter au cloisonnement des informations dans les différents services, et rendre difficile la planification des actes. En se concentrant sur le parcours patient, il est également question d'améliorer la qualité du service qui leur est prodigué, en prenant en compte leurs souhaits et en facilitant leur séjour au sein d'un établissement. La planification automatique des rendez-vous des patients est aussi une aide précieuse pour les équipes soignantes, qui peuvent consacrer plus de temps à leur cœur de métier.

La littérature autour de ce sujet remonte aux années 1950 (Bailey, 1952) et s'est développée au fil des années (Gupta and Denton, 2008, Berg and Denton, 2012), que ce soit pour des patients hospitalisés ou en soins ambulatoires. Dans une étude récente sur ce type de problèmes (Marynissen and Demeulemeester, 2016), les auteurs font état d'une grande ressemblance entre ce problème et les problèmes de séquençage de tâches avec une ou plusieurs machines (job-shop, open-shop, flow-shop). Le patient doit passer par une série d'actes avec différentes ressources (différents spécialistes) dans un ordre dépendant de sa pathologie (Azadeh et al., 2014).

Cette problématique est très proche de la notre. Dans les scénarios que nous décrivions dans le chapitre précédent, les patients sont également au centre des rendez-vous à planifier. Le plus souvent, il s'agit de planifier un parcours de soin pour plusieurs patients différents. Cependant, dans notre cas, nous voulons prendre en compte les disponibilités et les souhaits des patients, mais également de tous les autres intervenants. Les équipes soignantes ont également des préférences quant à leurs horaires ou d'autres critères. La similarité entre notre problème et celui du séquençage de tâches nous a poussé à le formuler comme un problème de gestion de projet à contraintes de ressources, plus communément appelé Resource Constraint Project Scheduling Problem (RCPSP).

2.3 Resource Constraint Project Scheduling Problem

Le problème que nous étudions dans ce manuscrit consiste à planifier un ensemble de demandes de rendez-vous nécessitant une quantité précise de ressources pendant un temps défini au préalable. Les problèmes d'ordonnancement de tâches et de planification concentrent l'attention des informaticiens depuis plus d'un demi-siècle et sont devenus des classiques de la littérature de la recherche opérationnelle. Ils sont très présents dans le domaine industriel : planification des travaux de maintenance pour divers types d'engins (trains, avions, réacteur nucléaire), création des plannings des satellites, ordonnancement des étapes de fabrication d'un produit, les exemples sont légion et touchent tous les domaines du secteur industriel.

Le problème de gestion de projet à contraintes de ressources, plus connu sous le nom de Resource Constraint Project Scheduling Problem (RCPSP), est issu d'une généralisation des problèmes de séquençage de tâches avec une ou plusieurs machines (job-shop, open-shop, flow-shop). Les origines de la définition du RCPSP remontent aux années 1960 (Pritsker et al., 1969). Ce problème d'optimisation combinatoire fait partie du groupe des problèmes NP-difficiles au sens fort, appartenance démontrée par (Blazewicz et al., 1983, Garey and Johnson, 1979).

2.3.1 Définition

Le problème RCPSP considère un projet composé d'un ensemble de ressources avec des disponibilités limitées et d'un ensemble d'activités liées par des relations de précédence dont la durée et la demande en ressource est connue. L'objectif est de planifier ces activités en assignant une date de début et des ressources à chaque activité de sorte que les contraintes de précédences et la disponibilité des ressources soient respectées.

De manière plus formelle, nous devons planifier un projet avec J un ensemble d'activités, nommées $j = 1,...,|J|$, et dont la durée est notée d_j. Les relations de précédences sont représentées par un ensemble de prédécesseurs P_j pour chaque activité j, qui ne peut démarrer avant que tous ses prédécesseurs $p \in P_j$ ne soient terminés. Nous disposons d'un ensemble K de ressources nommées $k = 1,...,|K|$ et de leur disponibilité R_k à chaque unité de temps. Chaque activité $j \in J$ requiert une quantité

r_{jk} de ressources de type k pour toute la durée d_j de l'activité. Traditionnellement, deux activités $j = 0$ et $j = J{+}1$ sans durée et sans ressources requises sont ajoutées, pour marquer le début et la fin du projet. Une planification est un ensemble d'attribution de date de début S_j à chaque activité $j = 0{,}1{,}...{,}J{+}1$. Dans sa définition classique, toutes les données du problème sont déterministes et connues.

L'objectif pour ce problème est d'attribuer une date de début S_j pour chaque activité $j \in J$ en veillant à ce que les ressources requises $k \in K$ soient disponibles en quantité suffisante pour satisfaire la demande r_{jk} pendant la durée d_j, en respectant les relations de précédences et en minimisant la durée totale du projet. La durée totale du projet correspond au nombre d'unité de temps s'écoulant entre le début de l'activité $j = 0$ et la fin de $j = J{+}1$, et est communément appelée ***makespan*** dans la littérature.

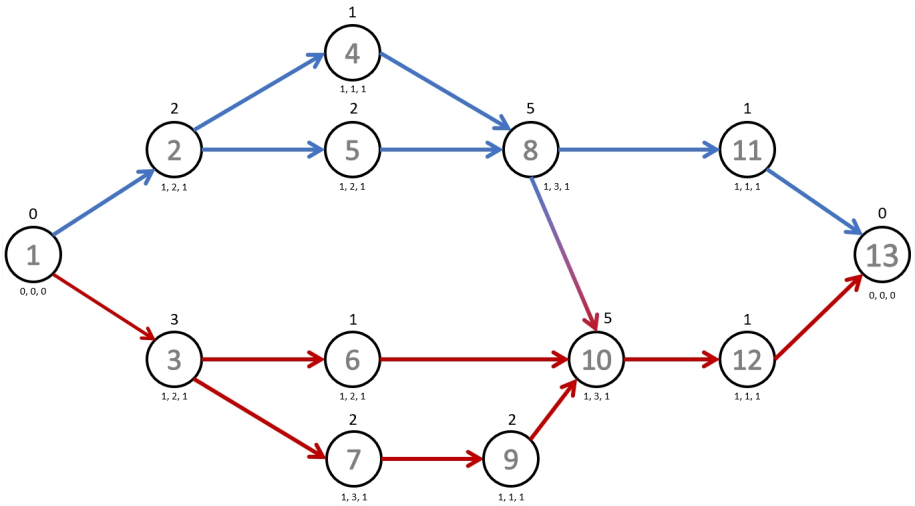


FIGURE 2.1: Ensemble d'activités et leur relation de précédence.

En guise d'exemple pour illustrer cette formulation du RCPSP, nous utilisons un problème inspiré du milieu médical : le parcours de soin de deux patients pour une transplantation de rein. La série de rendez-vous des deux patients est représentée dans la figure 2.1, avec 13 rendez-vous ou actes médicaux et 3 types de ressources. Les nœuds du graphe représentent des rendez-vous, caractérisés par la quantité de ressources requises r_{jk} pour chaque rendez-vous et par d_j, leur durée. Si nous prenons le nœud 8, il a une durée de 5 et nécessite 1 ressource de type k_1, 3 ressources de type k_2 et 1 ressource de type k_3. Les arcs entre les nœuds représentent les relations de précédence entre les rendez-vous, avec dans cet exemple le parcours de soin du

donneur avec les arcs bleus et le parcours du receveur avec les arcs rouges. Les rendez-vous 4 et 5 devront donc être terminés avant le rendez-vous 8. Les rendez-vous 0 et 13 marquent respectivement le début et la fin du parcours de soin des patients.

2.3.2 Extensions du RCPSP

Cette formulation classique du RCPSP permet de représenter les nombreux problèmes différents que nous évoquions plus tôt. Cependant, au fil des années, des variations de cette définition ont été proposées par les chercheurs pour représenter des contraintes plus spécifiques à certaines problématiques. Ces variantes du RCPCP concernent les principaux aspects du problème : les ressources, les activités, la fonction objective et la disponibilité des informations. Les principales extensions du RCPSP sont représentées dans la figure 2.2, et une revue récente de ces variantes peut être trouvée dans l'article de Habibi et al. (2018).

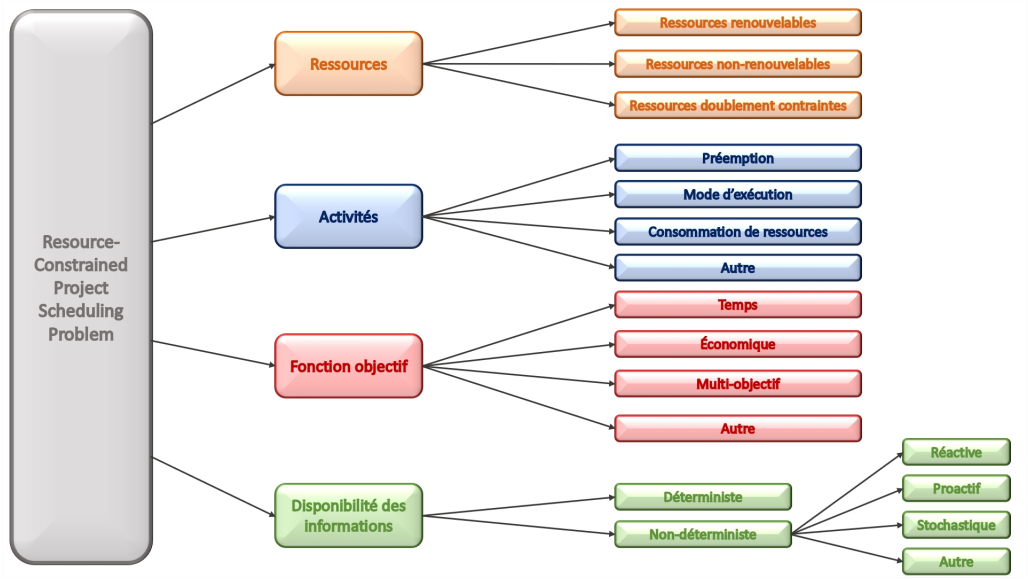


FIGURE 2.2: Classification des variantes du RCPSP.

Ressources Dans la définition que nous donnions précédemment, les ressources $k \in K$ sont considérées comme étant des ressources renouvelables. Au cours d'une activité j, un nombre r_{jk} de ressources de type k sont consommées et soustraites au total de ressources disponibles pendant la durée de l'activité d_j. Ces ressources étant renouvelables, elles seront restituées à la fin de l'activité et pourront être

utilisées pour une autre activité. Par exemple, les ressources humaines sont considérées comme des ressources renouvelables (Carlier and Moukrim, 2015), car elles sont à nouveau disponibles après l'activité. Pour revenir au domaine de la santé, les patients, les médecins, les spécialistes, les machines et les salles sont également considérées comme des ressources renouvelables.

Les ressources peuvent aussi être partiellement renouvelables, avec des capacités qui fluctuent au cours du temps, ce qui n'est pas représenté dans la définition classique du RCPSP. Ce concept, introduit par (Böttcher et al., 1999), ajoute la notion d'un sous-ensemble $D(k)$ d'unités de temps pour lesquels chaque type de ressource $k \in K$ est disponible avec une capacité notée C_k. De cette manière, il est possible d'ajouter la notion de disponibilité des ressources au fil du temps. Elles ne sont alors plus disponibles avec une certaine quantité sur toute la durée du projet, mais plutôt disponibles (toujours en quantité limitée) sur certaines périodes de temps. Cela nous permet de représenter de manière plus réaliste les disponibilités des ressources humaines, qui ne peuvent être disponibles en permanence. Dans le cadre des établissements de soins, c'est bien sûr le cas des patients, qui ne peuvent être sollicités n'importe quand, et c'est également le cas des praticiens qui ne travaillent pas en permanence.

Certaines ressources ne sont disponibles qu'en quantité limitée tout au long du projet. Ces ressources non-renouvelables sont consommés lors des activités, et il n'est pas possible d'en consommer plus que la quantité disponible pour l'ensemble du projet (Słowiński, 1980). C'est souvent le cas par exemple de matériau requis pour des processus de fabrication. Les ressources non-renouvelables peuvent être doublement contraintes et être à la fois limitées en quantité totale sur toute la durée du projet et limitée également à certaines périodes du projet. Ce genre de problématique se posent avec le financement, qui est fixé au lancement d'un projet, mais dont les fonds sont débloqués au fil des étapes du projet.

Dans la définition du RCPSP que nous avons donné précédemment, les ressources n'avaient qu'une seule compétence. Dans l'extension ***Multi-Skill*** du RCPSP (MS-RCPSP) (Néron and Baptista, 2002), un ensemble de compétences U est donné, et chaque ressource $k \in K$ maîtrisera un ensemble U_k de compétences. Les activités ne nécessitent pas un certain nombre de ressources d'un type $k \in K$, mais

elles demandent un nombre défini de compétences $u \in U$ pour être menées à bien. Cette extension du RCPSP permet une représentation plus proche de la réalité dans les nombreux problèmes impliquant des ressources humaines, qui généralement maîtrisent plusieurs compétences et sont polyvalents dans l'accomplissement de certaines tâches. C'est notamment le cas dans le milieu médical, où le personnel est capable de réaliser des actes médicaux différents nécessitant des compétences différentes.

Activités Tout comme les ressources, les activités qui composent le projet à planifier ont également été le sujet de nombreuses extensions. Dans le RCPSP standard, les activités doivent être réalisées en une seule fois. Autrement dit, lorsqu'une activité démarre, elle ne peut être interrompue et doit être menée à son terme : les activités ne sont pas préemptives. La notion de préemptivité des activités a été introduite par (Kaplan, 1988) et reprise par de nombreux auteurs, comme (Demeulemeester and Herroelen, 1996) et (Moukrim et al., 2015). Le Preemption Resource-Constrained Project Scheduling Problem (PRCPSP) permet les interruptions, avec ou sans restrictions, des activités au cours de leur déroulement. Sans restrictions, la durée d_j d'une activité $a \in A$ peut être divisée en d_j unités de durée. Des contraintes peuvent être ajoutées sur la durée minimale d'une unité de durée ou sur des intervalles de temps minimum à respecter entre l'arrêt et la reprise d'une activité.

Chaque activité, dans la définition classique du RCPSP, a une durée d_j et une quantité de ressources requises r_{jk} de type k qui est unique. Pour de nombreux problèmes, il est cependant possible d'exécuter une tâche de diverses manières : en allouant plus de ressources pour réduire la durée, en utilisant d'autres types de ressources, etc. Pour pouvoir représenter ces situations, le concept de modes d'exécution des activités a été introduit par Elmaghraby (1977) et demeure encore aujourd'hui l'une des variations du RCPSP la plus étudiée par la communauté. Dans ce Multi-Mode Resource-Constrained Project Scheduling Problem (MM-RCPSP), chaque activité $j \in J$ peut être exécutée dans un mode $m = 1,...|M|$ avec M l'ensemble des modes possibles. Chaque mode m définit une durée d_{jm} et une demande en ressource unique. Les activités débutées dans un mode m doivent être complétées dans ce même mode. Encore une fois, cette variation du RCPSP est particulièrement adaptée aux problèmes impliquant des ressources humaines, avec des tâches

pouvant être menées par des types de ressources différentes. Cela peut être appliqué aux problèmes dans le milieu médical, avec des actes médicaux ou des rendez-vous qui nécessitent des compétences possédées par plusieurs métiers.

Une autre variation du problème RCPSP permet de faire varier la demande de ressources d'une activité au fil du temps. La quantité de ressources de type $k \in K$ pour l'activité j n'a plus une valeur fixe pour toute la durée d_j de l'activité, mais une quantité r_{jkt} de ressource k pour chaque période t de l'activité. Cette formulation qui permet de représenter des cas où la demande en ressources n'est pas constante pour toute la durée d'une activité a été introduite dans plusieurs articles, dont (Bartusch et al., 1988) et (Reyck et al., 1999). La quantité de ressources pour chaque période est cependant connue et ne change pas. Par exemple, dans le cas d'une opération chirurgicale, de nombreux spécialistes interviennent mais ne doivent pas être présents durant toute l'opération. L'anesthésiste intervient avant l'opération pour endormir le patient et revient à intervalles réguliers pour s'assurer qu'il ne se réveille pas. Les chirurgiens opèrent mais ne sont pas nécessaires pour toute la durée de l'acte. Cette extension permet donc de formaliser ces situations de manière plus précise.

Fonction objectif Dans la définition classique du RCPSP, l'objectif est de réduire la durée totale du projet, communément appelée ***makespan***. De nombreux auteurs ont élaboré au fil du temps d'autres types de fonctions objectif pour pouvoir traiter des problèmes différents. Dans certains de ces problèmes, les activités $j \in J$ sont accompagnées d'une date d'échéance h_j (plus souvent appelée ***due date***). Le retard d'une activité par rapport à sa date d'échéance peut être notée $L_j = C_j - h_j$, avec C_j la date de complétion de j. Dans leurs articles, (Singh and Weiskircher, 2011) et (Wang et al., 2019) cherchent à réduire le retard total des activités par rapport à leur date d'échéance. Le retard noté $T_j = max(0, C_j - h_j)$ est similaire à L_j mais ne peut prendre des valeurs négatives. Dans le même ordre d'idée, certaines fonctions objectives prennent en compte l'avance E_j des activités sur leur date d'échéance h_j. Dans leurs travaux, (Viana and de Sousa, 2000, Bagherinejad and Majd, 2014) définissent une fonction objective prenant en compte ces deux dernières notions pour résoudre leur problème.

Nous évoquions précédemment le problème MM-RCSPSP, qui permettait à chaque activité d'être réalisée de plusieurs manières différentes, appelées modes. Certains

modes permettent de réaliser les activités plus rapidement ou avec d'autres ressources, mais cela peut entraîner un coût supplémentaire. Certains travaux (Berthaut et al., 2014, Hariga et al., 2019) visent justement à réduire ces coûts, qui sont considérés dans le calcul de la qualité d'une solution.

De nombreux autres objectifs existent, par exemple basés sur la robustesse d'une planification (Palacio and Larrea, 2017), sur la valeur apportée par un projet planifié (Vanhoucke, 2010), ou encore sur la qualité de la reprise d'une planification précédente (Deblaere et al., 2011, Chakrabortty et al., 2016). Pour notre problème de planification en milieu médical, nous ne souhaitons pas réduire la durée totale du projet, mais nous voulons avoir un maximum de rendez-vous et d'actes médicaux planifiés. De plus, certains actes étant plus prioritaires que d'autres entraîneront des pénalités de plus en plus importantes en fonction de leur retard par rapport au début de l'intervalle de faisabilité du rendez-vous. À ces deux objectifs principaux pourront s'ajouter d'autres contraintes souples qui seront spécifiques à certains problèmes, comme la réduction du nombre de salle utilisées pour la gestion d'un bloc opératoire, par exemple. Cette flexibilité du RCPSP sur la fonction objectif offre des perspectives à l'entreprise qui pourra par la suite proposer aux utilisateurs de définir leurs propres contraintes. Si dans un premier temps, il est raisonnable d'imaginer qu'ils utiliseront le moteur de planification sur des problèmes relativement simple, sur le long terme, il sera nécessaire que le logiciel soit en mesure d'intégrer ces contraintes supplémentaires. Dans le cadre de ce manuscrit, nous conserverons la fonction objectif visant à réduire les retards et le nombre de rendez-vous non planifiés.

Disponibilité des informations Dans la définition dite classique du RCPSP, les données du problème sont considérées comme étant connues et fiables. L'objectif est alors de prévoir à l'avance une planification des évènements dont la pertinence reposera sur ces informations. Pour certains domaines cependant, ces informations ne sont pas toujours fiables, ou ne sont pas toujours complètes. Pour d'autres, la planification ne peut se faire intégralement avant le début du projet, et doit donc intégrer les informations durant la résolution. Ces deux niveaux de disponibilité des informations, déterministe et notamment non-déterministe, impliquent des évolutions de représentation du problème.

Pour le cas déterministe, les informations sont considérées comme étant parfaitement justes et connues lors de la résolution. C'est le cas pour notre problème de planification. Pour la résolution du problème, nous considérons que toutes les informations concernant les disponibilités des patients, du personnel médical et des locaux sont exactes. Il en est de même pour toutes les demandes de rendez-vous à traiter. Évidemment, confrontées à la réalité du terrain, il est possible qu'une planification se révèle imparfaite, ou perturbée par des absences de certaines ressources, ou l'indisponibilité d'un patient. Pour ces situations, il sera parfois nécessaire de lancer un nouveau calcul avec les informations mises à jour pour trouver une nouvelle planification.

Il existe néanmoins de nombreux problèmes qu'il n'est pas envisageable de résoudre en considérant les données comme étant fiables et totalement connues. Ces incertitudes peuvent être causées par plusieurs phénomènes : l'absence d'une ressource, des activités plus longues ou plus courtes que prévu, etc. Ces aléas peuvent provoquer des coûts supplémentaires ou rendre un planning infaisable en raison d'annulation en cascade des activités. Pour adresser les problèmes avec ces données non-déterministes, de nombreuses variations du RCPSP ont été proposées. La première, que nous mentionnions dans le paragraphe précédent, consiste à relancer le calcul d'une planification en cas d'imprévus. De nombreux auteurs ont concentré leurs recherches sur le moment (Wu et al., 1993, Vieira et al., 2000) et la manière (Smith, 1995, Chakrabortty et al., 2016) de relancer cette planification. D'autres approches basées sur la planification stochastique ont été formulées. Elles reposent principalement sur des durées variables pour les activités (Valls et al., 1999, Ke and Liu, 2005, Chen et al., 2010), et non pas fixes comme dans la définition standard du RCPSP.

De nombreuses autres variations et extensions pour le RCPSP existent et ne sont pas recensées dans ce manuscrit. Elles peuvent être combinées et permettre de formuler des problèmes de plus en plus complexes, et cela d'une manière plus proche de la réalité. Des études (Habibi et al., 2018, Hartmann and Briskorn, 2022) sur ces variations sont régulièrement menées sur les nouveautés et les améliorations trouvées par les chercheurs.

2.3.3 Méthodes de résolution

Le problème de gestion de projet à contraintes de ressources est une problématique classique de la littérature concernant la recherche opérationnelle, et de nombreuses méthodes de résolution ont été proposées depuis des décennies pour résoudre ce problème d'optimisation combinatoire. Ces méthodes peuvent être rangées dans deux catégories : les méthodes exactes, qui fournissent une solution optimale en énumérant une grande partie des solutions au problème, et les méthodes approchées, qui ne donnent pas systématiquement la solution optimale mais ont des capacités d'exploration suffisantes pour fournir une solution proche de l'optimalité.

2.3.3.1 Méthodes exactes

Les méthodes exactes ont été les premières a être utilisées sur le RCPSP (Pritsker et al., 1969), afin de résoudre de manière optimale de petites et moyennes instances. Elles sont aujourd'hui encore utilisées soit seules, soit hybridées avec d'autres méthodes pour parvenir à résoudre les instances de plus grandes taille, qui restent problématiques pour ces méthodes exactes.

Branch-and-Bound Les algorithmes par séparation et évaluation, plus couramment appelés ***Branch-and-Bound*** (B&B), explorent un arbre représentant les énumérations des solutions au problème trouver la meilleure. Chaque branche représente un sous-ensemble de l'ensemble des solutions, et n'est explorée que si elle est intéressante en regard des bornes inférieures et supérieures estimées de la solution optimale. Cette approche proposée en 1960 (Land and Doig, 2010) s'est montrée particulièrement efficace sur de nombreux problèmes d'optimisation combinatoire.

La méthode du ***Branch-and-Bound*** est toujours le sujet de nombreuses articles et est notamment utilisée pour résoudre des problèmes liés aux variations les plus complexes du RCPSP. Elle a récemment été appliquée (Watermeyer and Zimmermann, 2020) à un problème RCPSP avec ressources partiellement renouvelables (RCPSP/π) en améliorant l'algorithme avec des calculs de bornes inférieures plus efficaces et des méthodes de branchement plus pertinentes. Dans un autre article (Alipouri, 2021), c'est un problème de gestion de projet avec incertitudes sur la durée des activités qui est adressé avec un algorithme ***Brand-and-Bound*** construi-

sant l'arbre de recherche par un parcours en profondeur, et exploitant le concept de démarrage au plus tôt des activités pour le parcourir. Dans un dernier exemple (Davari and Demeulemeester, 2019), un algorithme ***Brand-and-Bound*** est utilisé pour résoudre une variante du RCPSP également portée sur les incertitudes des durées des activités.

Programmation linéaire Un problème d'optimisation linéaire consiste à minimiser une fonction et des contraintes décrites par des fonctions linéaires. Si toutes les variables ne peuvent prendre que des valeurs entières, c'est alors un problème d'optimisation linéaire en nombre entiers (**M**ixed **I**nteger **L**inear **P**rogramming). Nombre d'algorithmes ont été élaborés pour résoudre ces problèmes, parmi lesquels figurent l'algorithme du simplex (Dantzig et al., 1955), la résolution par génération de colonnes (Dantzig and Wolfe, 1960). Ces méthodes, toujours intégrées aujourd'hui dans certains solveurs (CPLEX [1], Gurobi [2]) sont par conséquent utilisées pour résoudre de nombreux problèmes d'optimisation combinatoire, comme le RCPSP.

Récemment, des formulations **MILP** ont été proposées pour résoudre la version classique du RCPSP. Dans leur article (Gnägi et al., 2018), les auteurs proposent une formulation en temps continu (Continuous-time, CT), qui permet de commencer les activités à n'importe quel point dans le temps. Dans un autre article récent (Villafáñez et al., 2018), les auteurs décrivent deux formulations MILP, l'un privilégiant la réduction du ***makespan*** et l'autre la résistance de l'emploi du temps fourni à certains aléas (durée des activités, retards). Dans ces deux articles, les approches MILP fournissent de bons résultats sur un ensemble d'instances reconnus dans la littérature RCPSP : les instances PSPLIB (Kolisch and Sprecher, 1997), réparties en différentes tailles et pour différentes variations du RCPSP. D'autres formulations peuvent également être trouvées dans certains articles (Koné et al., 2011, Artigues et al., 2015) les répertoriant et les comparant entre elles.

2.3.3.2 Méthodes approchées

Pour résoudre le problème RCPSP et ses nombreuses variations, d'innombrables méthodes approchées ont été proposées au cours de ces dernières années. Ces heu-

1. https ://www.ibm.com/fr-fr/products/ilog-cplex-optimization-studio
2. https ://www.gurobi.com/products/gurobi-optimizer/

ristiques et métaheuristiques permettent d'explorer efficacement l'espace des solutions sans en énumérer toutes les possibilités. Elles sont donc capables de traiter efficacement des problèmes complexes ou de grande taille, sans toutefois garantir l'optimalité des réponses fournies. De plus en plus de méthodes mêlant les principes de différentes approches (exactes ou non), communément appelées hybrides, sont utilisées pour résoudre ce problème de planification.

Algorithmes glouton Les algorithmes gloutons sont régulièrement utilisés sur de nombreux problèmes d'optimisation. Un algorithme glouton repose sur des sélections localement optimales à chaque itération pour trouver une solution d'une bonne qualité globale. Néanmoins cette stratégie qui ne prend pas en compte le problème dans sa globalité mène souvent l'algorithme dans des optimums locaux dont il ne parviendra pas à se sortir par la suite. Les choix faits lors de la construction d'une solution sont définitifs, et par conséquent, la qualité des solutions obtenues est souvent éloignée de l'optimalité. Cependant, les algorithmes gloutons s'exécutent rapidement et sont souvent faciles à implémenter. Pour ces raisons, ils sont régulièrement utilisés pour générer des solutions initiales pour des métaheuristiques.

Recherches locales Les méthodes de recherche locale consistent à explorer l'espace de recherche en se déplaçant de solution en solution. Ces déplacements se font en appliquant un léger changement à une solution pour obtenir une solution dite "voisine". Le choix d'un voisin se fait en fonction de divers critères. Souvent, il s'agit d'améliorer la solution et donc de se diriger vers le meilleur voisin. Mais parfois, le choix se porte sur des solutions qui présentent des caractéristiques inédites par rapport aux solutions précédentes, ce qui met alors l'accent sur l'exploration de l'espace de solution.

À partir de ce principe, d'autre métaheuristiques ont été conçues. Le recuit simulé (Kirkpatrick et al., 1983) s'inspire de la stabilisation d'un matériau en fusion pour faire converger la recherche vers une solution optimale. Au fil de la recherche, un paramètre représentant la température du matériau diminue, et permet des mouvements de moins en moins importants sur la solution. Ainsi, c'est l'exploration qui est privilégiée au lancement de l'algorithme, puis l'exploitation des bonnes solutions, avec des mouvements de moindre ampleur. Cette approche a été utilisée avec suc-

cès sur de nombreux problèmes d'optimisation combinatoire (Abbasi et al., 2006, Laurent et al., 2017).

Les recherches locales à voisinage large (**L**arge **N**eighborhood **S**earch, (Shaw, 1998)) reprennent le même principe que les recherches locales, mais avec un voisinage potentiel plus élargi. L'algorithme applique des mouvements différents à la solution, permettant de la modifier plus profondément (mouvements de destruction et de construction) que la recherche locale classique, en explorant plusieurs voisinages différents. Basée sur la même idée, la recherche locale adaptative à voisinage large (**A**daptive **L**arge **N**eighborhood **S**earch, (Ropke and Pisinger, 2006)) ajoute une couche d'apprentissage à l'algorithme, qui sélectionne le mouvement à appliquer à la solution en fonction de ses performances passées. Les méthodes de recherches à voisinage variable se sont montrées efficaces sur des problèmes RCPSP (Palpant et al., 2004, Muller, 2009, 2011).

Algorithmes à essaims L'étude des comportements sociaux de nombreux animaux et insectes a mené à l'élaboration d'un grand nombre d'algorithmes reproduisant en partie le comportement de ces espèces (Engelbrecht, 2007, Blum and Merkle, 2008). Dans ces algorithmes, les individus d'un groupe, leurs déplacements et leurs communications sont représentés. Des actions et des interactions des individus entre eux va émerger un comportement global. C'est cette intelligence de groupe qui va permettre de résoudre divers problèmes complexes. Les essaims et troupeaux qui ont inspiré des modèles informatiques sont divers et variés : colonies de fourmis, essaims d'abeilles et nuées d'oiseaux pour les plus connus, meutes de loups, troupes de lions et groupes de baleines à bosse pour les plus exotiques. Les algorithmes à essaim de particules (**P**articule **S**warm **O**ptimization) font également partie des approches exploitant l'intelligence collective pour résoudre des problèmes complexes.

Les algorithmes par colonies de fourmis (**A**nt **C**olony **O**ptimization, (Colorni et al., 1991)) font partie des algorithmes à essaim les plus connus et les plus utilisés. Ils sont inspirés du comportement collectif des colonies de fourmis pour trouver de la nourriture. Des fourmis éclaireuses explorent les alentours du nid pour chercher des sources de nourritures, par des chemins plus ou moins directs. Pour rentrer au nid, elles vont également emprunter des chemins plus ou moins directs. Sur tous les chemins empruntés, elles déposent des pistes de phéromones, qui inciteront les

autres fourmis à suivre ce même chemin. Les chemins les plus courts, qui seront parcourus par plus de fourmis, auront des pistes de phéromones plus fortes, qui inciteront de plus en plus de fourmis à les suivre. De cette manière, le plus court chemin vers la nourriture est trouvé grâce au collectif de fourmis, en se basant sur les actions de chaque individu. Cette méthode est régulièrement utilisée pour résoudre des problèmes RCPSP. Dans leur article, (Merkle et al., 2002) obtiennent de nouvelles meilleures solutions sur des instances dérivées des plus larges de la bibliothèque PSPLIB.

Algorithmes génétiques Tout comme les algorithmes à essaim, les algorithmes génétiques font partie des algorithmes évolutionnaires, qui ont la particularité de reproduire des phénomènes et comportements observés dans la nature pour résoudre des problèmes informatiques complexes. Les algorithmes génétiques s'inspirent de la théorie de l'évolution des espèces décrite par Charles Darwin pour faire évoluer une population de solutions à un problème posé. Cette évolution artificielle est reproduite en faisant passer ces solutions par un cycle d'opérations ayant pour but d'explorer l'espace de recherche tout en exploitant les solutions trouvées les plus prometteuses. Ce cycle de reproduction, de mutation et de sélection des solutions permet de faire émerger au fil des générations (itérations de l'algorithme) les solutions les plus adaptées au problème traité. Cette approche a été popularisée dans les années 1980 par Holland et Goldberg (Booker et al., 1989, Holland, 1992).

Les algorithmes génétiques ont été utilisés pour résoudre de nombreux problèmes au cours de ces dernières décennies (Chambers, 2000). Ils se sont aussi montrés particulièrement efficaces sur les problèmes de planification et le problème RCPSP, qui nous intéressent dans ce manuscrit. À la fin des années 1990 et au début des années 2000 (Hartmann, 1998, 2002), Hartmann propose un algorithme génétique pour résoudre le RCPSP classique. Dans ces articles, plusieurs représentations d'individus et opérateurs sont comparés sur des instances issues de PSPLIB. La représentation par liste d'activités, qui a la particularité d'assurer le respect des précédences pour chaque individu, est encore aujourd'hui très utilisée et performante sur ce type de problème. Une amélioration de cette représentation sera proposée par Alcaraz et Maroto (Alcaraz and Maroto, 2001). Quelques années plus tard, Debels et Vanhoucke proposent un algorithme génétique faisant évoluer deux populations en parallèle

(Debels and Vanhoucke, 2005), l'une avec des plannings justifiés à gauche (classés par ordre décroissant des dates de fin des activités), et l'autre à droite (classés par ordre croissant des dates de début des activités).

Plus récemment, les algorithmes génétiques seuls font toujours l'objet d'améliorations pour traiter des variantes plus complexes du RCPSP. Kadri et Boctor proposent un algorithme génétique avec une méthode de crossover à deux points (Kadri and Boctor, 2018) pour résoudre le RCPSPTT (Resource-Constrained Project Scheduling Problem with Transfer Times), pour lequel le transfert des ressources d'une activité à l'autre est pris en compte. Dans un autre article récent (Ma et al., 2018), un algorithme génétique utilisant des recherches locales a obtenu de bons résultats sur des instances générées avec l'outil ProGen et sur des instances de PSPLIB. À l'image de ce dernier exemple, un nombre important de recherches portent sur des algorithmes génétiques tirant parti d'autres méthodes de résolutions.

Approches hybrides Les approches hybrides consistent à combiner les principes de plusieurs méthodes pour profiter des forces des unes et combler les faiblesses des autres. Il est par exemple possible d'utiliser une méthode exacte au sein d'une méthode approchée, pour profiter des capacités d'exploration de l'approchée et des capacités d'exploitation des méthodes exactes. Un autre exemple cité précédemment : l'ajout d'un ou plusieurs opérateurs « recherche locale » à un algorithme génétique (Ma et al., 2018). Les stratégies d'hybridation sont nombreuses et permettent de combiner différentes métaheuristiques en les intégrant l'une à l'autre, en utilisant des informations récupérées par la première pour améliorer la seconde ou encore en décomposant le problème pour traiter les sous-problèmes avec plusieurs métaheuristiques différentes.

Ces méthodes hybrides ont été utilisées pour résoudre diverses variations du RCPSP au cours des dernières années. L'une des stratégies les plus couramment utilisée est la combinaison des algorithmes à population avec des recherches locales. Dans un article de 2010, les auteurs remplacent l'opération de crossover par une recherche locale, méthode qui s'est montrée efficace sur une version classique du RCPSP (Zamani, 2010). Plus récemment, une approche similaire intégrant une recherche locale à un algorithme génétique (Afshar et al., 2019) a été proposée et obtient des solutions satisfaisantes sur des instances générées avec ProGen.

D'autres hybridations sont basées sur la collaboration entre les approches utilisées. Dans un article récent (Myszkowski et al., 2018), les auteurs utilisent un algorithme d'évolution différentielle et transmettent ensuite la meilleure solution de la génération à un algorithme glouton, pour générer un planning de bonne qualité. Les deux algorithmes ne sont pas intégrés l'un à l'autre, mais partagent leurs informations pour obtenir de meilleurs résultats. Cette collaboration entre approches est également utilisée dans un autre article sur la résolution d'un problème RCPSP (Zamani, 2017) avec un algorithme génétique et un algorithme d'énumération implicite, qui obtiennent également des résultats de bonne qualité sur les instances de la bibliothèque PSPLIB.

Le nombre de combinaisons d'algorithmes et d'approches est trop important pour en faire une liste exhaustive dans ce manuscrit, mais ce domaine étant le sujet de nombreuses recherches ces dernières années, il est possible de trouver des études (Ting et al., 2015, Pellerin et al., 2020) répertoriant ces algorithmes et leurs performances sur différents problèmes.

Conclusion

Dans ce chapitre, nous nous sommes intéressés à la complexité du problème de planification que nous traitons, et aux différentes méthodes utilisées dans la littérature pour résoudre des problèmes similaires. C'est un problème NP-difficile, largement étudié au cours des dernières années et présents dans divers domaine.

Dans le milieu médical, c'est également une problématique récurrente, que ce soit pour prévoir les vacations des Infirmiers Diplômés d'État ou pour planifier les opérations au sein d'un bloc opératoire. Les problèmes de planifications que nous souhaitons traiter sont proches des problèmes de planification du parcours patient, qui consistent à prévoir tous les rendez-vous d'un patient au sein d'un établissement. Le patient y est central, mais les établissements consultés par l'entreprise souhaitent également optimiser l'emploi de temps de leurs équipes, au delà du temps gagné avec un outils de planification automatique.

Nous avons rapproché ce problème d'un autre bien connu de la recherche opérationnelle : le Resource Constraint Project Scheduling Problem. Cette formulation est par de nombreux aspects proche du problème que nous traitons, et nous permettra

de le formaliser pour pouvoir ensuite élaborer nos méthodes de résolution. Nous utiliserons la version standard du RCPSP, qui suffit à représenter les problèmes traités durant cette thèse. À l'avenir, les nombreuses variations et extensions du RCPCP permettront d'étoffer le modèle afin de répondre aux demandes d'améliorations des premiers clients du logiciel.

Au cours des dernières décennies, de nombreuses approches ont été proposées pour résoudre le RCPSP. Que ce soit des méthodes exactes, approchées ou une hybridation des deux, certaines se montrent particulièrement efficaces sur les instances RCPSP. Parmi ces méthodes, nous nous intéresserons en premier lieu à la programmation linéaire en nombre entier, qui nous permettra d'obtenir des solutions optimales sur certaines instances, et des bornes inférieures et supérieures pour les instances plus complexes. Ensuite, nous élaborerons des recherches locales pour traiter des instances plus grandes et posant des difficultés à notre méthode exactes. Enfin, nous porterons notre attention vers les algorithmes génétiques, particulièrement efficaces sur ces problèmes de planification.

Chapitre

3

Description du problème et formalisation

Contents

= 3.1 Formulation du problème

3.1.1 Données du problème

3.1.2 Contraintes

3.1.3 Solution & Fonction objectif

3.2 Modèle mathématique

3.3 Instances & Résultats

3.3.1 Cas d'utilisation concrets

3.3.2 Instances de l'entreprise

3.3.3 Instances PSPLIB

Dans cette partie nous présentons notre problème de planification et proposons une formulation pour représenter tous les éléments de la problématique : données, contraintes et objectif. Nous décrirons ensuite notre modèle de programmation linéaire en 0-1. Nous présenterons les scénarios de l'entreprise et les instances que nous avons générées ainsi que les instances PSPLIB, qui font office de référence dans la littérature concernant le RCPSP. Enfin, nous utiliserons le solveur CPLEX sur nos instances et nous présenterons les résultats obtenus.

3.1 Formulation du problème

L'entreprise Evolucare propose à ses clients de nombreuses solutions qui gèrent chaque année plus d'un million de patients, et environ 70 millions de rendez-vous. Néanmoins, les solutions actuellement proposées par l'entreprise ne fournissent pas de services de planification automatique de ces rendez-vous, une fonctionnalité pourtant demandée régulièrement par les clients. Cette planification permettrait aux équipes soignantes de gagner un temps précieux qu'elles pourraient consacrer aux soins des patients. Nous nous intéressons dans ce manuscrit à l'étude et à l'élaboration des méthodes permettant de résoudre ce problème.

Cette planification consiste à attribuer des ressources et des heures de début à des ensembles de rendez-vous, généralement les parcours de soins des patients, sur des durées de quelques jours ou quelques semaines. Ces rendez-vous et actes médicaux nécessitent plusieurs ressources : le patient, les soignants, les salles, le matériel, etc. Ces ressources possèdent une ou plusieurs propriétés qui vont être requises pour les rendez-vous. Les affectations de ressources et le choix des heures de début des rendez-vous doivent respecter certaines contraintes, appelées contraintes « dures ». Elles assurent la cohérence de la solution en interdisant par exemple l'ubiquité des ressources ou en vérifiant que les affectations respectent bien les propriétés requises par un rendez-vous ou son intervalle de faisabilité. D'autres contraintes (appelées contraintes « souples ») ne sont pas obligatoires mais auront un impact sur la qualité d'une solution si elles ne sont pas respectées. Trouver une solution au problème nécessite donc d'affecter des ressources et des dates aux rendez-vous en respectant les contraintes dures et en minimisant les violations des contraintes souples.

Pour la formulation de notre problème, nous nous sommes inspirés du RCPSP en raison des nombreux points communs existant entre ce problème et le nôtre. Dans les sous-parties suivantes, nous présentons les données et les contraintes que nous devons respecter pour produire une solution valide.

3.1.1 Données du problème

Dans cette partie nous formalisons les données de notre problème de planification. Elles sont nombreuses, de natures diverses et difficilement intégrables dans leur

globalité pour un cerveau humain. Pour traiter ce problème, nous proposons une formalisation inspirée du RCPSP pour les représenter. Nous pouvons les ranger dans trois catégories : les données des ressources, des rendez-vous et des créneaux horaires sur lesquels ces rendez-vous seront planifiés. Par ailleurs, pour les problèmes que nous traitons, nous considérons que les informations sont fiables et complètes, et que nous ne devons faire face à aucune incertitude.

Créneaux de temps La période de temps donnée durant laquelle nous devons planifier les rendez-vous est communément appelée l'horizon. Pour nos problèmes, l'horizon H est décomposé en un ensemble discret de créneaux de temps t de même durée, appelés aussi **timeslots**. Un rendez-vous ou un acte médical ne peut commencer entre deux créneaux de temps. Par exemple, sur un problème où l'horizon commence à huit heures et avec une granularité des créneaux de quinze minutes, un rendez-vous ne peut commencer à huit heures dix ou huit heures vingt.

Ressources Pour représenter les données de nos ressources, nous définissons un ensemble R de ressources et un ensemble Π de propriétés. À chaque ressource $r \in R$ est associé un ensemble de propriétés Π_r qu'elle possède et qui pourront être requises par les rendez-vous. En ce qui concerne les ressources humaines il est en effet courant qu'une personne soit polyvalente et puisse effectuer plusieurs tâches différentes (mais pas simultanément). Par exemple, un chirurgien orthopédique possédant les propriété $\pi_1 = chirurgien$ et $\pi_2 = medecin$ peut effectuer des opérations dans sa spécialité (qui demanderont la propriété π_1) mais il peut aussi assurer des consultations médicales (qui demanderont la propriété π_2). Nous disposons également de toutes les disponibilités de ces ressources. Pour tous les créneaux $t \in H$, et pour toutes les ressources $r \in R$, $dispo_t^r = 1$ si la ressource r est disponible au créneau t, 0 sinon.

Rendez-vous L'ensemble des demandes de rendez-vous ou d'actes médicaux A contient toutes les demandes à satisfaire en leur assignant des ressources et une date de début. Chaque demande $a \in A$ est caractérisée par :

— sa durée en nombre de créneaux $duration_a$;

— son intervalle de faisabilité ES_a et LS_a, avec ES_a la date de début au plus tôt et LS_a la date de début au plus tard ;

— $qtreq_a^\pi$ la quantité requise de ressources possédant la propriété π ;

— Π_a l'ensemble des propriétés requises par a ;

— $pred_a$ et $succ_a$ qui sont respectivement les ensembles de rendez-vous devant précéder et suivre a ;

— $PreAssigned_a$ un ensemble de couple (r,π) de ressource r avec la propriété π pré-affectées à a ;

— $Essential_a \geq 1$ le coefficient d'importance du rendez-vous a ;

— $Emergency_a \geq 0$ le coefficient de priorité du rendez-vous a.

Nous notons également Re_a l'ensemble des ressources possédant une des propriétés requises par a. Les coefficients $Essential_a$ et $Emergency_a$ permettent respectivement de quantifier l'importance et la priorité d'un rendez-vous, et seront pris en compte dans le calcul de la qualité de la solution.

3.1.2 Contraintes

Les contraintes dures présentées ci-dessous permettent d'assurer la cohérence d'une solution et sont inspirées des contraintes exprimées pour le problème RCPSP. Les contraintes souples répondent aux désirs des planificateurs qui ont été consultés dans le cadre du projet LORH. Les contraintes dures sont les suivantes :

CD 1 Une ressource ne peut être affectée à plusieurs rôles dans un même rendez-vous. Si certaines peuvent avoir plusieurs propriétés et être capables de répondre à plusieurs demandes différentes, elles ne peuvent néanmoins pas remplir plusieurs rôles simultanément.

CD 2 Une ressource ne peut être affectée à une activité avec une propriété qu'elle ne possède pas. De même, une ressource qui ne possède aucune des propriétés requises par un rendez-vous ne peut y être affectée.

CD 3 Chaque rendez-vous, s'il est planifié, débute à une date comprise entre ES_a et LS_a, son intervalle de faisabilité. Cet intervalle définit la période de temps durant laquelle un rendez-vous peut être planifié.

CD 4 Si un rendez-vous est planifié, alors toutes les demandes de propriétés sont satisfaites durant toute la durée du rendez-vous, et les ressources qui y sont affectées doivent être disponibles pendant tout le rendez-vous. Ils sont « non-préemptifs » et ne peuvent être interrompus une fois qu'ils ont débuté.

CD 5 Si un rendez-vous est planifié, ses relations de précédence doivent être respectées. Il est possible que certains rendez-vous d'une chaîne de précédence ne soient pas planifiés, mais l'ordre entre les rendez-vous déjà placés doit être respecté.

CD 6 Les rendez-vous ne peuvent être programmés qu'une seule fois. Même si certains actes peuvent être récurrents et revenir à intervalles réguliers, ils seront considérés comme plusieurs demandes de rendez-vous différentes.

CD 7 Une ressource ne peut être affectée à plusieurs rendez-vous simultanément. Les ressources ne possèdent pas le don d'ubiquité, une solution les affectant à plusieurs rendez-vous sur les mêmes créneaux de temps n'est pas cohérente.

CD 8 Les ressources pré-affectées aux rendez-vous sont effectivement affectées aux rendez-vous. Dans le milieu médical, les patients sont souvent assignés directement aux rendez-vous, qui sont créés spécifiquement pour eux. Ces pré-affectations doivent être respectées, un rendez-vous prévu pour un patient sans la présence de ce dernier n'ayant pas de sens.

Les contraintes dures présentées ci-dessus doivent toutes être respectées par une solution pour qu'elle soit considérée comme étant valide. Ces différents critères, qu'ils découlent du bon sens, de la définition du RCPSP ou des spécifications données par les planificateurs, assurent la cohérence de la solution. Nous détaillerons la formalisation mathématique de ces contraintes dans la section suivante.

Ces contraintes dures s'accompagnent de contraintes dites « souples » qui détermineront la qualité d'une solution. Elles entraîneront des pénalités si elles ne

sont pas respectées, nous permettant ainsi de distinguer les solutions respectant les contraintes dures ci-dessus.

CS 1 L'objectif premier pour les planificateurs est d'attribuer des ressources et une date de début à un maximum de rendez-vous. Chaque rendez-vous du problème qui n'est pas planifié entraînera une pénalité proportionnelle à son importance, définie dans les données du problème.

CS 2 Certains rendez-vous seront plus prioritaires que d'autres, et les planificateurs souhaitent qu'ils soient planifiés le plus tôt possible. Les rendez-vous prioritaires planifiés entraîneront une pénalité proportionnelle à leur retard par rapport au début de leur intervalle de faisabilité et à à leur coefficient de priorité.

3.1.3 Solution & Fonction objectif

3.1.3.1 Structure des solutions

Notre objectif pour ces problèmes de planification est de construire des emplois du temps avec le plus de rendez-vous planifiés possible, tout en réduisant les retards de rendez-vous prioritaires. Planifier un rendez-vous ou un acte médical consiste à lui assigner des ressources et une date de début, en s'assurant que les contraintes dures énoncées précédemment soient respectées. Pour chaque demande de rendez-vous $a \in A$ nous définissons le triplet (a,t_a,R_a), avec R_a l'ensemble des ressources assignées à a et t_a la date de début assignée à a. R_a est un ensemble de ressources disponibles pendant toute la durée $t_a+duration_a$ et répondant exactement aux propriétés requises pour a. Par ailleurs, t_a doit respecter l'intervalle de faisabilité de a et être compris entre ES_a et LS_a. Quand un rendez-vous n'a pas reçu d'affectation, t_a est nul et R_a est un ensemble vide.

Une solution valide Sol est un ensemble de triplets (a,t_a,R_a) respectant toutes les contraintes dures. Nous notons A_{Sol} l'ensemble des rendez-vous planifiés et $A_{\overline{Sol}}$ l'ensemble des rendez-vous qui ne sont pas planifiés.

3.1.3.2 Fonction objectif

L'objectif de l'outil souhaité par les planificateurs des établissements contactés par l'entreprise est de planifier un maximum de rendez-vous parmi les demandes à traiter sur l'horizon donné. Ils souhaitent également pouvoir différencier certaines demandes des autres en leur accordant plus d'importance ou une priorité plus élevée. Notre fonction objectif prendra donc en compte les coefficients $Essential_a$ et $Emergency_a$ de toutes les demandes $a \in A$ pour calculer la qualité globale de la solution. Le premier terme de la fonction objectif 3.1 fait la somme pour tous les rendez-vous non-planifiés $a \in A_{\overline{Sol}}$ de leur coefficient $Essential_a$. Le second terme calcule la somme des retards des rendez-vous planifiés $a \in A_{Sol}$, pondérés par leur coefficient de priorité $Emergency_a$. Ce retard est calculé comme suit : le nombre de créneaux entre le début d'un rendez-vous et le début de son intervalle de faisabilité $t_a - ES_a$ est divisé par le nombre total de créneau dans l'intervalle de faisabilité $LS_a - ES_a$. Ce ratio est ensuite multiplié par $Emergency_a$. Ainsi, plus le retard est important, plus la pénalité sera conséquente.

Le but de nos méthodes de résolution est de construire une solution respectant toutes les contraintes présentées ci-dessus, minimisant le nombre de rendez-vous non planifiés et minimisant également les retards des rendez-vous prioritaires.

$$f(Sol) = \sum_{a \in A_{\overline{Sol}}} Essential_a + \sum_{a \in A_{Sol}} \frac{t_a - ES_a}{LS_a - ES_a} \times Emergency_a \tag{3.1}$$

Nous illustrons les problèmes soumis à l'entreprise avec un exemple simple mais représentatif des situations rencontrées par les planificateurs. Les données de cet exemple simple sont présentées dans le tableau 3.1. Sur un horizon $H = \{0,1,...,12\}$, nous voulons planifier les rendez-vous de l'ensemble $A = \{a_1,a_2,a_3,a_4,a_5,a_6\}$. Nous disposons d'un ensemble $R = \{r_1,r_2,r_3,r_4\}$ de ressources et d'un ensemble de propriétés $\Pi = \{\pi_1,\pi_2,\pi_3\}$. Les ressources r_1 et r_2 possèdent la propriété $\pi_1 = patient$, et les ressources r_3 et r_3 possèdent respectivement les propriétés $\pi_2 = kinsithrapeute$ et $\pi_3 = ergothrapeute$. Les quantités de ressources requises r avec les propriétés π sont également décrites dans le tableau 3.1. Dans ce problème, chaque rendez-vous nécessite un patient précis et l'un des deux praticiens. Certains sont liés par des re-

lations de précédence : a_2 doit être précédé par a_1, et a_3 et a_5 doivent être précédés par a_4.

TABLE 3.1: Caractéristiques des demandes $a \in A$ de l'exemple.

a	$duration_a$	$qtreq$	$PreAssigned_a$	$Essential_a$	$Emergency_a$	$pred_a$	ES_a	LS_a
a_1	3	$qtreq_{a_1}^{\pi_2} = 1$	$\{ r_1 \}$	1	0	$\emptyset$	t_0	t_{12}
a_2	2	$qtreq_{a_2}^{\pi_3} = 1$	$\{ r_1 \}$	1	0	$\{a_1\}$	t_0	t_{12}
a_3	2	$qtreq_{a_3}^{\pi_2} = 1$	$\{ r_1 \}$	1	0	$\{a_4\}$	t_0	t_{12}
a_4	4	$qtreq_{a_4}^{\pi_2} = 1$	$\{ r_2 \}$	5	3.75	$\emptyset$	t_0	t_{12}
a_5	3	$qtreq_{a_5}^{\pi_3} = 1$	$\{ r_2 \}$	1	0	$\{a_4\}$	t_0	t_{12}
a_6	4	$qtreq_{a_6}^{\pi_3} = 1$	$\{ r_2 \}$	10	0	$\emptyset$	t_0	t_{12}

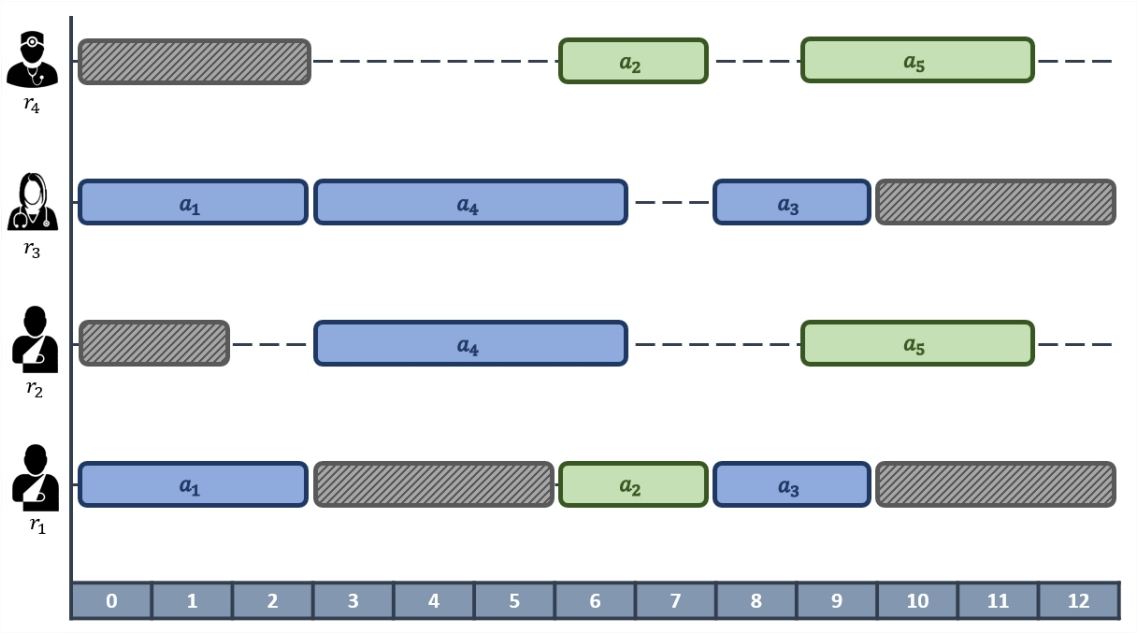


FIGURE 3.1: Solution *Sol* possible à l'exemple donné.

La figure 3.1 représente une solution valide *Sol* au problème posé. Nous y trouvons les emplois du temps des quatre ressources en fonction de l'horizon H composé de treize créneaux horaires. Tous les rendez-vous sont planifiés (sauf a_6) et sont indiqués dans l'emploi du temps de toutes les ressources qui y sont affectées. Le rendez-vous a_1 par exemple apparaît bien dans les emplois du temps des ressources r_1 et r_2. Les rendez-vous impliquant le premier praticien, la ressource r_3, sont indiqués en bleu tandis que les rendez-vous impliquant le second, la ressource r_4 sont

indiqués en vert. Les périodes d'indisponibilité des ressources sont illustrées par des zones grises hachurées. Toutes les contraintes présentées dans la section 3.1 sont respectées. Cependant, la solution n'est pas optimale : le rendez-vous a_6 n'est pas planifié et le rendez-vous a_4, qui était prioritaire, entraînera également une pénalité sur la qualité de la solution. L'évaluation de la qualité pour cette solution est égale à $f(Sol) = 10+0.9375$, 10 par la pénalité due au rendez-vous a_6 qui n'est pas posé et 1.25 de pénalité de retard pour le rendez-vous a_4, qui était prioritaire.

Nous avons présenté notre formulation pour les données et la représentation choisie pour nos solutions. Nous avons également détaillé la fonction objectif nous permettant d'évaluer les solutions obtenues. Dans la partie suivante, nous proposons un modèle mathématique représentant les problèmes de planification qui nous intéressent dans ce manuscrit en nous appuyant sur les données et la fonction objectif présentées précédemment.

3.2 Modèle mathématique

Nous présentons dans cette section le modèle de programmation linéaire en nombre entier (ILP) appliqué à notre problème de planification. Ces modèles expriment les contraintes et la fonction objectif en fonctions linéaires. Dans notre modèle, les variables de décisions ne prennent que les valeurs 0 et 1. Ces variables de décisions sont les suivantes :

— $y_a^t = 1$ si le rendez-vous a débute au créneau t, 0 sinon, $\forall a \in A$, $\forall t \in H$.

— $x_a^{r,\pi} = 1$ si la ressource r avec la propriété π est affectée au rendez-vous a, 0 sinon, $\forall a \in A$, $\forall r \in R$, $\forall \pi \in \Pi$.

Les deux variables y_a^t et $x_a^{r,\pi}$ représentent respectivement les affectations de date de début et de ressources aux rendez-vous. L'équation 3.2 est l'expression de notre fonction objectif 3.1 relativement à ces contraintes. Le premier terme calcule pour chaque rendez-vous $a \in A$ si un rendez-vous est planifié ou non, le second terme calcule quant à lui les retards des rendez-vous prioritaires.

$$minimise \sum_{a \in A} (1 - \sum_{t \in [ES_a; LS_a]} y_a^t) \times Essential_a + \sum_{a \in A} \sum_{t \in [ES_a; LS_a]} y_a^t \times \frac{t - ES_a}{LS_a - ES_a} \times Emergency_a \tag{3.2}$$

Cette fonction doit être minimisée en respectant les contraintes dures décrites précédemment. La transcription de ces contraintes est donnée ci-dessous :

$$\forall a \in A, \forall r \in R_a, \sum_{\pi \in \Pi_r} x_a^{r,\pi} \leq 1 \tag{3.3}$$

L'équation 3.3 représente la contrainte **CD 1** et assure qu'une ressource r n'est affectée qu'à un rôle dans le rendez-vous, et qu'elle n'est utilisée qu'avec une seule de ses propriétés $\pi \in \pi_r$.

$$\forall r \in R, \sum_{\pi \in \Pi \setminus \Pi_r} \sum_{a} x_a^{r,\pi} = 0 \tag{3.4}$$

$$\forall a \in A, \sum_{r \in R \setminus R_a} \sum_{\pi \in \Pi} x_a^{r,\pi} = 0 \tag{3.5}$$

Les équations 3.4 et 3.5 vérifient que les ressources ne sont pas affectées avec des propriétés qu'elles ne possèdent pas. L'équation 3.4 vérifie qu'une ressource r ne peut être affectée avec une propriété π qu'elle n'a pas tandis que l'équation 3.5 s'assure qu'aucune ressource r ne peut être affectée à un rendez-vous a si elle ne possède aucune des propriétés requises par a. Ces deux équations correspondent à la contrainte **CD 2** décrite précédemment.

$$\forall a \in A, \forall r \in R_a, \forall t \in [ES_a; LS_a],$$
$$duration_a \times \sum_{\pi \in \Pi_a} x_a^{r,\pi} - y_a^t \times \sum_{t'=t}^{t+duration_a-1} dispo_t^r \leq (1 - y_a^t) \times H \tag{3.6}$$

$$\forall r \in R, \forall a \in A_r, \forall b \in A_r - \{a\}, \forall t \in [\max(ES_a; ES_b); \min(LS_a; LS_b)],$$
$$\sum_{\pi \in \Pi_r} x_a^{r,\pi} + \sum_{t'=t-duration_b+1}^{t} y_a^{t'} + \sum_{\pi \in \Pi_r} x_b^{r,\pi} + \sum_{t'=t-duration_b+1}^{t} y_b^{t'} \leq 3 \tag{3.7}$$

Toujours concernant les ressources et en référence à la contrainte **CD 4**, les équations 3.6 et 3.7 s'assurent que les ressources ne sont affectées que si elles sont disponibles. L'équation 3.6 contrôle qu'une ressource r ne peut être affectée à un rendez-vous a planifié en t que si $dispo_{t'}^r = 1$ pour tout t' compris entre t et $t+duration_a-1$. L'équation 3.7 vérifie qu'une ressource r n'est pas affectée à plusieurs rendez-vous a en même temps.

$$\forall a \in A, |H| \times |\Pi| \times \sum_{t \in [ES_a; LS_a]} y_a^t \geq \sum_{r \in R_a} \sum_{\pi \in \Pi_a} x_a^{r,\pi} \tag{3.8}$$

L'équation 3.8 garantit que les ressources $r \in R$ restent disponibles si elles ne sont pas affectées à un rendez-vous a.

$$\forall a \in A, \sum_{t \in H} y_a^t \times ES_a \leq \sum_{t \in H} y_a^t \times t \leq LS_a \tag{3.9}$$

$$\forall a \in A \sum_{t \in [ES_a; LS_a]} y_a^t \leq 1 \tag{3.10}$$

Les équations 3.9 et 3.10, correspondant respectivement aux contraintes **CD 3** et **CD 6** assurent que les rendez-vous $a \in A$ sont programmés dans leur intervalles de faisabilité et qu'ils ne sont planifiés qu'une seule fois.

$$\forall a \in A, \forall \pi \in \Pi_a, \sum_{r \in R_a} x_a^{r,\pi} = qtreq_a^{\pi} \times \sum_{t \in [ES_a; LS_a]} y_a^t \tag{3.11}$$

$$\forall a \in A, \forall \pi \in \Pi_a, \sum_{r | (r,\pi) \in PreAssigned_a} x_a^{r,\pi} = \sum_{t \in [ES_a; LS_a]} y_a^t \tag{3.12}$$

L'équation 3.11 s'assure que les quantités de ressources requises pour chaque propriété $qtreq_a^{\pi}$ sont respectées pour tous les rendez-vous planifiés. Dans le même ordre d'idée, la contrainte 3.12 garantit que les ressources pré-affectées à r sont bien assignées à a. Les deux équations 3.11 et 3.12 correspondent respectivement aux contraintes **CD 4** et **CD 8**.

$$\forall b \in A, \forall a \in pred_b,$$
$$\sum_{t \in [ES_b; LS_b]} y_b^t \times t - (\sum_{t \in [ES_a; LS_a]} y_a^t \times (t + duration_a) \geq (2 - \sum_{t \in [ES_b; LS_b]} y_b^t - \sum_{t \in [ES_a; LS_a]} y_a^t) \times (-3 * H) \tag{3.13}$$

$$\forall b \in A, \forall a \in pred_b,$$
$$\sum_{t \in [ES_b; LS_b]} y_b^t \times t - (\sum_{t \in [ES_a; LS_a]} y_a^t \times (t + duration_a) \leq (2 - \sum_{t \in [ES_b; LS_b]} y_b^t - \sum_{t \in [ES_a; LS_a]} y_a^t) \times H \tag{3.14}$$

Enfin, les équations 3.13 et 3.14 représentent la contrainte **CD 5** et concernent les relations de précédences. Elles contrôlent que pour tout rendez-vous $b \in A$ planifiés, les rendez-vous $a \in pred_b$ sont également planifiés.

3.3 Instances & Résultats

Le modèle présenté ci-dessus nous permet de représenter les données et les contraintes qui s'appliquent aux problèmes de planification que nous traitons. Après de nombreuses consultations auprès des établissements clients de l'entreprise, nous disposons de plusieurs scénarios décrivant des problèmes souvent rencontrés par les équipes de planificateurs. À partir de ces cas d'usage typiques du logiciel LORH, nous avons construit des instances que nous utiliserons pour évaluer l'efficacité de nos méthodes. En parallèle, nous confronterons aussi ces approches à des instances reconnues de la littérature traitant des problèmes RCPSP.

3.3.1 Cas d'utilisation concrets

Pour déterminer les contours d'une application, il est nécessaire de recueillir les besoins des futurs utilisateurs. Dans le cadre de ce travail de construction de cas d'utilisation typiques du logiciel, les collaborateurs d'Evolucare en contact avec les planificateurs ont pu notamment nous faire remonter les problèmes qu'ils devaient résoudre dans leur quotidien. Plus précisément, les planificateurs de plusieurs établissements en France nous ont décrit les problèmes typiques auxquels ils devaient faire face. Autrement dit, les problèmes sur lesquels ils seraient les plus enclins à utiliser un logiciel de planification. Ces informations sont évidemment utiles pour le développement de l'application et des différentes interfaces avec l'utilisateur, mais elles le sont également pour notre travail de recherche.

Ces descriptions de problèmes réels nous permettront de tester sur des cas d'utilisation concrets les différentes approches envisagées pour tenter de résoudre ce problème de planification. Ces scénarios sont intéressants pour plusieurs raisons. Premièrement, ils nous ont permis de mieux comprendre la problématique de planification, dont la difficulté de résolution apparaît clairement en observant ces cas d'usage. Ceci est d'autant plus frappant lorsque que l'on considère qu'ils sont, le

plus souvent, résolus à la main ou en tout cas, rendez-vous par rendez-vous. Ensuite, il est important pour le travail de recherche que nous menons de confronter les différentes méthodes à des problèmes de la vie réelle, qui peuvent être fondamentalement différents des instances théoriques. Certaines approches sont très efficaces sur les problèmes théoriques mais s'avèrent être moins performantes sur des problèmes concrets, ou inversement. Ces scénarios décrivent des problèmes différents, provenant de types d'établissements variés (hôpitaux, centre de rééducation, etc.), qui nous permettront d'étudier le comportement de nos méthodes sur des problèmes aux structures différentes.

Dans les sections suivantes, nous présentons les scénarios qui servirons à nos expérimentations. Ils ont tout d'abord quelques points communs. Pour chaque situation, les planificateurs ont décrit l'ensemble des ressources : leurs propriétés, leurs compétences et leurs disponibilités. Nous avons également l'ensemble des actes médicaux et rendez-vous à planifier. Pour ces quelques exemples et de manière générale, les actes médicaux font partie d'un parcours de soins et sont prévus pour des patients spécifiquement. Les patients sont donc pré-affectés aux rendez-vous qui les concernent.

3.3.1.1 Scénario *CardioRehab* : séjour en centre de réadaptation cardio-vasculaire

Ce problème représente le parcours de soins de patients au sein d'un centre de réadaptation cardio-vasculaire. Les séjours en centre de réadaptation cardio-vasculaire sont prescrits après un infarctus du myocarde ou une opération lourde comme l'angioplastie coronaire pour réduire les facteurs de risque, retrouver une plus grande autonomie et réapprendre à vivre après un accident cardiaque ou avec une maladie. Les équipes dans ces centres sont pluridisciplinaires et sont composée de cardiologues, médecins généralistes, kinésithérapeutes, ergothérapeutes, Infirmiers Diplômés d'État (IDE), diététiciens, etc. La durée du séjour change selon les patients, mais reste de l'ordre de quelques semaines.

Le scénario qui nous a été décrit est la planification des 10 actes médicaux et rendez-vous de 16 patients sur un horizon d'une semaine. L'équipe soignante est composée de 5 spécialistes : un médecin généraliste, un cardiologue, un kinésithérapeute, un ergothérapeute et un IDE. Chaque patient a dans son parcours de soins

deux rendez-vous de deux heures avec chacun de ces praticiens. Ces derniers sont disponibles pendant 8 demi-journées dans la semaine, tandis que les patients sont disponibles toute la semaine. L'objectif dans cet exemple est de planifier les 160 rendez-vous dans la semaine.

3.3.1.2 Scénario *SurgSep* : gestion d'un bloc opératoire

Le bloc opératoire comprend l'ensemble des locaux et des équipements nécessaires aux opérations chirurgicales. Il comprend une ou plusieurs salles d'opération qui peuvent être modulables, les locaux de stockages et les couloirs pour la circulation des patients. C'est dans le bloc opératoire que sont pratiqués les interventions chirurgicales et plus généralement tout acte invasif pour le diagnostic ou le soin de diverses pathologies.

Cet exemple traite de la planification des opérations de 16 patients dans un bloc opératoire. Chaque patient doit subir une intervention orthopédique ou gastrique. Les opérations sont de différents types : opération du genou, opération de l'épaule, opération de gastrectomie et opération réparatrice de hernie hiatale. Ces interventions ont des durées différentes, nécessitent la présence d'un chirurgien (orthopédique ou gastrique) et la disponibilité d'une salle. Par simplification, nous considérons dans cet exemple uniquement les chirurgiens et faisons abstraction des autres personnels soignants nécessaires à l'accomplissement de l'opération. Ces 16 opérations sont à planifier sur une seule journée, de huit heures à dix-huit heures. Dans ce scénario, nous avons 4 chirurgiens (2 chirurgiens orthopédiques et 2 chirurgiens gastriques) et 4 salles d'opération. Le but est de prévoir les 16 opérations en tenant compte du nombre limité de salles disponibles.

3.3.1.3 Scénario *Admission* : admission à l'hôpital

Un hôpital est un établissement de soins qui prend en charge les patients dont les pathologies ou traumatismes ne peuvent être traités dans le cabinet d'un médecin ou à domicile. Ce sont généralement des structures de grandes tailles, qui regroupent plusieurs services de différentes spécialités : chirurgie, maternité, pédiatrie, etc. Les patients peuvent être admis et passer plusieurs jours, semaines ou mois dans l'hôpital pour suivre leur parcours de soin. Les hospitalisations peuvent être programmées

par un médecin ou décidées suite à une admission aux urgences. La planification dans ce type d'établissement est complexe, car le nombre de variables à prendre en compte peut devenir très important. Il y a beaucoup de patients, de spécialistes, de personnel soignants, de matériel, et le nombre de combinaisons possibles, de solutions potentielles au problème de planification, explose rapidement.

Dans ce scénario, il s'agit de planifier l'admission et le parcours de soins de 8 patients au sein d'un hôpital sur un horizon d'une semaine. Les actes médicaux de ce suivi nécessitent différents spécialistes et sont de durées variables. Chaque patient doit avoir 17 rendez-vous, pour un total de 136 rendez-vous à planifier. Le personnel médical est composé de 5 spécialistes : un médecin généraliste, un kinésithérapeute, un Infirmier Diplômé d'État (IDE), un diététicien et un psychologue. Tous n'ont pas une charge égale de travail, certains sont plus sollicités que d'autres. De fait, il n'est pas forcément possible de planifier l'ensemble des rendez-vous du problème. L'objectif dans ce problème est de placer le maximum de rendez-vous au regard des disponibilités limitées des praticiens.

3.3.1.4 Scénario *RehabCenter* : journée de soins en centre de rééducation

Les centres de rééducation, tout comme les centres de réadaptation cardio-vasculaire, permettent aux patients de récupérer après une maladie ou un traumatisme et de recouvrer leurs conditions de vie d'avant leur séjour hospitalier. Ces Soins de Suite et de Réadaptation (SSR) se trouvent généralement en fin de parcours de soins, et sont prodigués dans des établissements disposant d'équipements spéciaux dédiés aux soins de rééducation physique.

Ce dernier cas d'étude représente une journée dans un centre de rééducation, avec tous les actes médicaux à planifier pour 24 patients. Chaque patient doit participer à 4 rendez-vous, deux rendez-vous avec un médecin généraliste, un rendez-vous avec un kinésithérapeute et un rendez-vous avec un ergothérapeute. Cela représente au total 96 rendez-vous, dont 48 rendez-vous avec un médecin, 24 rendez-vous avec un kinésithérapeute et 24 rendez-vous avec un ergothérapeute. Tous les patients et tous les praticiens sont disponibles de huit heures à midi et de quatorze heures à dix-huit heures. Encore une fois, l'objectif dans ce scénario est de construire un planning avec un maximum de rendez-vous posés, tout en respectant les contraintes.

TABLE 3.2: Description des scénarios.

Scénario	Nombre de ressources		Disponibilité des ressources	Durée des rendez-vous	Nombre de rendez-vous		Horizon
					Par patient	Au total	
SurgDep	patient	16	100%	entre 3 et 5 heures	1	16	un jour
	chirurgien	4	83%				
	salle	4	83%				
Admission	patient	8	75%	entre 30 minutes et 1 heure 30	17	136	une semaine
	spécialiste	4	75%				
RehabCenter	patient	24	100%	2 heures	4	96	un jour
	médecin	12	100%				
	kinésithérapeute	6	100%				
	ergothérapeute	6	100%				
CardioRehab	patient	16	77%	2 heures	10	160	une semaine
	spécialiste	10	77%				

3.3.2 Instances de l'entreprise

Les scénarios décrits dans la section précédente sont quelques exemples inspirés des situations auxquelles les planificateurs font face au quotidien. À partir de ces scénarios, nous avons généré de nombreuses instances que nous tenterons de résoudre avec nos différentes méthodes. Par soucis d'homogénéité entre les instances, nous avons relâché les contraintes spécifiques de certains scénarios. Leurs caractéristiques et les paramètres utilisés pour générer les instances sont décrits dans les sections suivantes.

3.3.2.1 Résumé des caractéristiques et génération d'instances

Les quelques exemples présentés ci-dessus représentent différentes situations concrètes dans différents établissements. Il existe beaucoup d'autres situations dans les types d'établissements cités, et les scénarios décrits ne sont qu'une fraction des problèmes que les planificateurs traitent au quotidien. Les caractéristiques des scénarios sont résumées dans le tableau 3.2. Nous y détaillons la taille du problème, le nombre de ressources de chaque type, leur disponibilité sur l'horizon, la durée et le nombre de rendez-vous et l'horizon du problème.

À partir de la structure de base de chaque scénario, nous pouvons agir sur certains paramètres pour générer des instances différentes, en nombre suffisant pour pouvoir mener des tests pertinents. Pour conserver la structure et donc la particularité de chaque problème, les demandes de ressources, la durée des rendez-vous

et l'horizon ne changeront pas d'une instance à l'autre. La structure générale des disponibilités, souvent découpée en demi-journées correspondants aux vacations du personnel, sera également conservée. Pour générer des instances différentes et plus ou moins difficiles, nous agissons sur les paramètres suivants : **(1)** la taille de l'instance (nombre de rendez-vous et de ressources), **(2)** le nombre de relations de précédence entre les rendez-vous, **(3)** les coefficients d'importance et d'urgence et enfin **(4)** la disponibilité des ressources.

(1) Taille de l'instance Pour modifier la taille de l'instance, nous multiplions le nombre de demandes de rendez-vous $a \in A$ et le nombre de ressources $r \in R$ par un facteur allant de 1 à 3. L'horizon H, lui, n'est pas modifié. C'est l'augmentation du nombre de ressources qui permet de répondre au nombre accru de rendez-vous. Nous avons choisi de conserver les horizons pour que les instances restent cohérentes et proches du problème de base. Les scénarios que nous traitons se déroulent sur une journée ou une semaine, et les prolonger de plusieurs jours ou semaines les rendrait trop différents des problèmes soumis par les planificateurs. C'est également vrai pour la taille des instances : une augmentation trop importante les rendrait peu vraisemblables.

(2) Précédences Dans la définition initiale des scénarios, les rendez-vous ne possèdent pas de relations de précédence entre eux et peuvent être effectués dans n'importe quel ordre. Néanmoins, les parcours de soins imposant un ordre spécifique entre les actes médicaux ne sont pas rares et notre logiciel sera confronté à des rendez-vous liés par des précédences. Nous pouvons simuler des parcours de soins similaires en ajoutant des précédences aux rendez-vous de certains patients du scénario. En fonction d'une probabilité p, des patients sont sélectionnés et des précédences sont ajoutées entre les rendez-vous qui les concernent. Avec un degré maximum k, les précédences sont ajoutées aléatoirement selon la procédure décrite dans l'algorithme 1.

À partir d'un ensemble de sommets V représentant l'ensemble des rendez-vous d'un patient et d'un entier k représentant le degré maximum souhaité du graphe de précédence, l'algorithme 1 construit un graphe de précédence GP de degré k. Tant que V n'est pas vide, un sommet s est choisi aléatoirement avec la fonction $randomSelectSommet$ et retiré de V. Un ensemble constitué de x sommets est

Algorithme 1: AjoutPrécédence()

Données: $V = \{s_1,s_2,...,s_n\}$ un ensemble de sommets, k un entier

Résultat: un graphe de précédence GP de degré k

1 **Début**
2 $GP \leftarrow \emptyset$
3 **Tant que** $V \neq \emptyset$ **faire**
4 $s \leftarrow randomSelectSommet(V)$
5 $V \leftarrow V \setminus \{s\}$
6 $x \leftarrow min(|GP|,random[0,k-1])$
7 $pred(s) \leftarrow randomSelectPred(GP,x)$
8 $GP \leftarrow GP \cup \{s+$les arcs (t,s), $t \in pred(s)\}$
9 **Fin**
10 Retourne GP
11 **Fin**

ensuite sélectionné par la fonction $randomSelectPred$, et des arcs sont ajoutés entre s et les sommets $t \in pred(s)$. Cet ensemble est ensuite ajouté au graphe GP.

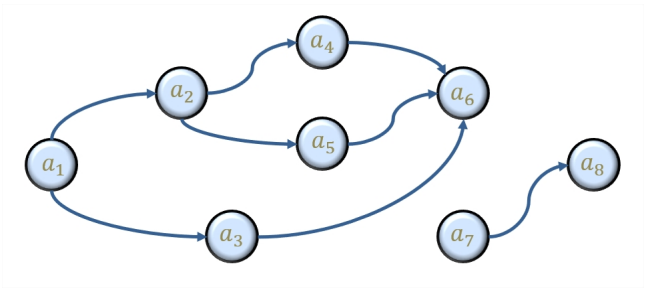


FIGURE 3.2: Exemple d'ajout de précédence sur un ensemble de rendez-vous.

La figure 3.2 illustre un graphe de précedences $G = (V,E)$ obtenu avec cette méthode et $k = 3$, où l'ensemble des sommets V est composé des rendez-vous A_{r_1} d'un patient r_1. L'ensemble des arêtes E, initialement vide, représente les relations de précédences entre les rendez-vous. Après ajout des précédences, nous obtenons un graphe orienté sans cycle, qui n'est pas nécessairement connexe.

(3) Importance et urgence Les notions d'importance et d'urgence des rendez-vous permettent aux planificateurs de distinguer certains actes qui sont plus prioritaires

que les autres. Initialement, pour tous les rendez-vous $a \in A$, $Essential_a = 1$ et $Emergency_a = 0$. Selon la formule 3.1, cela signifie que chaque rendez-vous qui n'est pas planifié entraîne une pénalité de 1. Aucun des rendez-vous n'étant prioritaire, il n'y a aucune pénalité relative aux retards.

Nous avons généré certaines de nos instances en faisant varier ces coefficients, afin d'obtenir des problèmes différents et d'observer le comportement de nos approches sur ces rendez-vous importants et prioritaires. Pour l'importance, un nombre i de rendez-vous $a \in A$ sont sélectionnés aléatoirement et voient leur coefficient $Essential_a$ multiplié par un facteur 5, 25 ou 125. De cette manière, certains rendez-vous non-planifiés auront un impact conséquent sur la qualité de la solution.

En ce qui concerne les priorités, le procédé est similaire : un nombre p de rendez-vous $a \in A$ sont choisis aléatoirement parmi les rendez-vous avec $Essential_a = 1$ et $Emergency_a = 0$. Leur coefficient $Emergency_a$ est multiplié par un facteur 3,75, 18,75 ou 93,75. Les coefficients $Essential$ sont également multipliés par 5, 25 et 125 pour qu'il soit toujours préférable selon la formule 3.1 de planifier un rendez-vous prioritaire tardivement que de ne pas le planifier du tout.

(4) Disponibilités Le dernier point sur lequel nous agissons pour la génération des instances est la disponibilité des ressources. Plus précisément, c'est sur la disponibilité des patients que se portent les modifications. Le personnel médical dispose déjà de demi-journées de repos dans la définition initiale des scénarios. Nous retirons des disponibilités aux patients pour illustrer les situations où ils ont des périodes réservées aux visites ou tout simplement lorsqu'ils ont exprimé le besoin d'avoir des périodes de repos. Le retrait des disponibilités des patients se fait de la manière suivante. Un nombre d de patients sont sélectionnés aléatoirement. Pour les scénarios *SurgSep* et *RehabCenter* ayant un horizon d'une journée, nous retirons une demi-journée de disponibilité, aussi choisie aléatoirement, à ces patients. Pour les scénarios *Admission* et *CardioRehab*, nous retirons deux demi-journées.

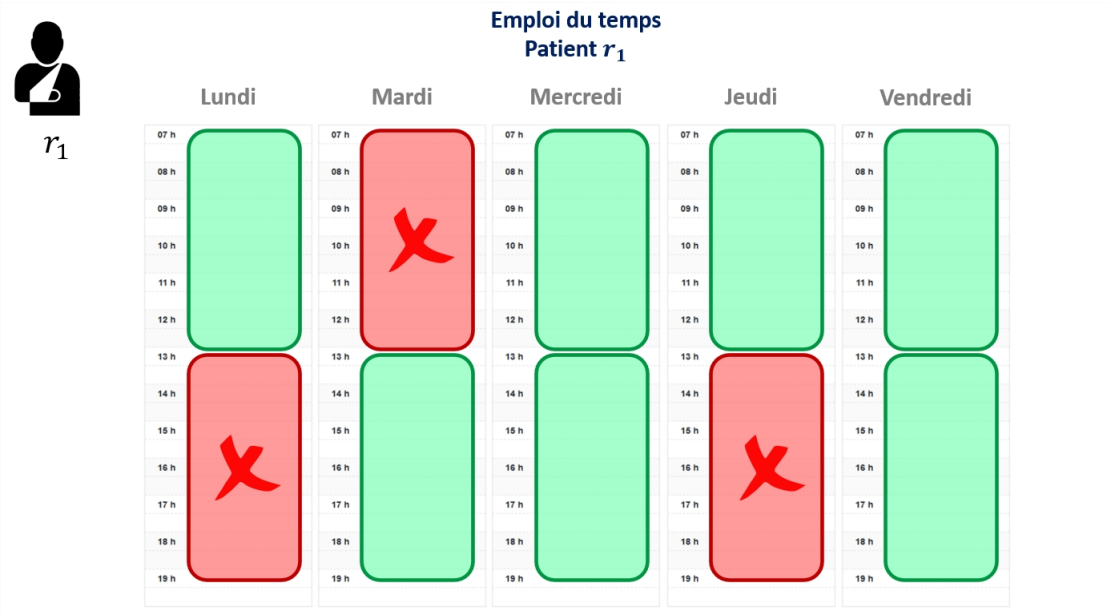


FIGURE 3.3: Retrait de trois demi-journées de disponibilité à un patient r_1.

Une illustration du retrait de ces disponibilités peut être trouvée dans la figure 3.3. Les disponibilités d'un patient r_1 sont représentées en vert, tandis que les disponibilités retirées sont indiquées en rouge. Le retrait de ces disponibilités peut rendre difficile, voir impossible, la planification de tous les rendez-vous du patient. Cependant, ce sont des situations qui peuvent se produire, et l'objectif pour les planificateurs est alors d'en planifier le plus possible.

Les combinaisons de ces quatre critères nous permettent de générer de nombreuses instances, respectant toujours les structures des scénarios définis par les planificateurs et représentant aussi des situations pouvant survenir dans une utilisation classique de l'outil de planification que l'entreprise souhaite commercialiser. Pour notre travail de recherche, cela permet d'obtenir des instances variées et d'observer les résultats obtenus par nos approches sur des problèmes différents.

3.3.2.2 Expérimentations & Résultats

Pour obtenir les résultats optimaux sur ces instances, nous utiliserons le modèle mathématique décrit en section 3.2. Ce modèle de programmation linéaire en 0-1 a été implémenté avec le solveur CPLEX en version 12.8.0.0. L'objectif pour chaque instance, guidé par la fonction objectif de l'équation 3.2, est de planifier le maximum de rendez-vous du problème et de limiter les retards pour les rendez-vous urgents.

Les expérimentations ont été menées de la manière suivante : les tests ont été lancés avec une limite de temps de 12 heures par instance, avec CPLEX en configuration par défaut. Ces calculs ont été effectués sur la plateforme MatriCS de l'Université de Picardie Jules Verne.

TABLE 3.3: Résultat sur les instances *SurgDep* : de 16 à 48 rendez-vous.

SurgDep.					**Lower bound**	**Upper bound**	**Rendez-vous**
Taille	**Importance**	**Urgence**	**Précédence**	**Disponibilités**			
16	0	0	0	0	0	0	16/16
16	4	0	0	0	0	0	16/16
16	8	0	0	0	0	0	16/16
16	4	4	0	0	1.479	1.479	16/16
16	8	8	0	0	9.538	9.538	16/16
16	0	0	0	4	0	0	16/16
16	8	4	0	4	1.479	1.479	16/16
48	0	0	0	0	0	9	39/48
48	12	0	0	0	0	7	41/48
48	24	0	0	0	0	8	40/48
48	12	12	0	0	57.875	68.161	42/48
48	24	24	0	0	85.653	137.598	39/48
48	0	0	0	4	0	6	42/48
48	0	0	0	12	0	5	43/48
48	24	12	0	12	114.773	116.271	47/48

Nous reportons les résultats obtenus sur les instances ***SurgDep*** dans le tableau 3.3. Nous indiquons pour chaque instance : le nombre de rendez-vous au total, le nombre rendez-vous importants, le nombre de rendez-vous urgents, le nombre de relations de précédence et le nombre de demi-journées retirées aux patients. Pour cette famille d'instance, aucune précédence n'est ajoutée. Pour chaque instance, nous donnons les bornes inférieures et supérieures fournies par CPLEX et le nombre de rendez-vous planifiés par rapport au nombre total de rendez-vous pour les bornes supérieures fournies par CPLEX. En cas d'égalité entre la borne inférieure et supérieure, la solution optimale est atteinte, ce qui est le cas sur toutes les instances à 16 rendez-vous, quelque soit les paramètres utilisés. Tous les rendez-vous sont placés, et les résultats supérieurs à 0 sont dus aux rendez-vous urgents, qui ne peuvent être tous programmés au plus tôt. L'optimalité n'est jamais atteinte sur les instances de 48 rendez-vous. Le nombre de rendez-vous important seul n'a pas d'effet significatif sur la difficulté de l'instance, les résultats étant soit identiques soit meilleurs que

sur les instances sans cette notion d'importance. Le nombre de rendez-vous urgent a un effet plus prononcé. Nous constatons une différence moyenne entre les bornes inférieures et supérieures données par CPLEX plus importante sur les instances avec rendez-vous prioritaires que sans (11.503 contre 8). À l'inverse, la différence moyenne entre les bornes sur les instances où les patients sont moins disponibles est plus basse (5.5 contre 7.136). Ces meilleures performances s'expliquent par le nombre réduit de possibilité à traiter pour CPLEX, conséquence des disponibilités réduites des patients. Enfin, le taux de rendez-vous planifiés accuse une chute de 13,28% entre les instances à 16 rendez-vous et les instances à 48 rendez-vous, témoignant de la difficulté de CPLEX à résoudre les instances de plus grande taille.

TABLE 3.4: Résultat sur les instances *RehabCenter* : de 96 à 288 rendez-vous.

RehabCenter					**Lower bound**	**Upper bound**	**Rendez-vous**
Taille	**Importance**	**Urgence**	**Précédence**	**Disponibilités**			
96	0	0	0	0	0	0	96/96
96	24	0	0	0	0	0	96/96
96	48	0	0	0	0	0	96/96
96	0	24	0	0	265.953	317.625	94/96
96	0	48	0	0	557.812	603.25	94/96
96	0	0	15	0	0	0	96/96
96	48	24	15	0	308.569	315	96/96
96	96	48	15	0	532.875	1426.5	67/96
288	0	0	0	0	0	196	92/288
288	72	0	0	0	0	1914	90/288
288	144	0	0	0	0	4960	88/288
288	0	72	0	0	689.062	2609.75	70/288
288	0	144	0	0	1958.352	6064.5	99/288
288	0	0	25	0	0	196	92/288
288	144	72	24	0	689.062	5729.875	71/288
288	288	144	25	0	1958.352	8740.5	98/288

Les résultats obtenus sur les instances de la famille ***RehabCenter*** sont présentés dans le tableau 3.4. Sur les instances de 96 rendez-vous, CPLEX est en mesure de fournir des solutions optimales sur les instances ne comportant pas d'urgence. Sur les autres, des écarts apparaissent entre les bornes inférieures et supérieures, avec une différence moyenne de 249.291. Ces écarts sont causés par les retards sur les rendez-vous prioritaires. Le taux d'actes planifiés reste élevé, à 95.7%. Sur l'instance la plus difficile à 96 rendez-vous, tous importants et avec 48 rendez-vous prioritaires, l'écart est beaucoup plus grand et le taux de rendez-vous placés chute à 69.79%. Sur

les instances à 288 rendez-vous, les différences entre les bornes supérieures et inférieures sont plus conséquentes (3139,474 d'écart moyen) et le taux d'actes planifiés chute à 30.38%. Cela s'explique par le nombre de rendez-vous et de ressources plus important, et donc du nombre de combinaisons beaucoup plus grand, qui devient alors difficile à traiter pour CPLEX. Sur la plupart des instances de cette taille, il reste au minimum 196 actes à planifier, et donc une grande marge d'amélioration sur ce scénario en particulier. Ce nombre important de rendez-vous à planifier explique les écarts importants constatés entre les bornes données par CPLEX.

TABLE 3.5: Résultat sur les instances *Admission* : de 136 à 408 rendez-vous.

Admission					**Lower bound**	**Upper bound**	**Rendez-vous**
Taille	**Importance**	**Urgence**	**Précédence**	**Disponibilités**			
136	0	0	0	0	0	4	132/136
136	34	0	0	0	0	4	132/136
136	68	0	0	0	0	4	132/136
136	0	34	0	0	126.316	204.848	132/136
136	0	68	0	0	149.84	298.757	130/136
136	0	0	48	0	0	4	132/136
136	0	0	0	35	0	4	132/136
136	68	34	48	35	127.082	202.633	132/136
136	136	68	48	35	154.664	306.781	131/136
408	0	0	0	0	0	24	384/408
408	102	0	0	0	0	31	377/408
408	204	0	0	0	0	65	383/408
408	0	102	0	0	165.64	2841.446	251/408
408	0	204	0	0	345.113	5333.518	203/408
408	0	0	64	0	0	24	384/408
408	0	0	85	0	0	24	384/408
408	0	0	0	91	0	24	384/408
408	204	102	64	91	165.64	3136.232	245/408
408	408	204	85	91	345.82	6904.066	206/408

Le tableau 3.5 présente les résultats sur le scénario ***Admission***. Nous remarquons qu'aucune solution optimale n'a été trouvée. Cependant, sur ce scénario, les ressources médicales ont plusieurs demi-journées de repos, et ne sont pas disponibles en permanence. De fait, certains rendez-vous ne peuvent pas être planifiés, par manque de ressources. Le taux de rendez-vous planifiés reste élevé : 96.81%. Nous constatons une nouvelle fois des écarts importants entre les bornes sur les instances ayant des rendez-vous prioritaires, avec une différence moyenne de 113.779 (contre 4 pour les autres). Sur les instances à 408 rendez-vous, des solutions satisfaisantes

sont obtenues sur les instances sans rendez-vous prioritaires, avec un écart moyen de 32 entre les bornes inférieures et supérieures. L'écart moyen entre les bornes sur les instances comportant des actes urgents est bien plus important (4298.262), avec des bornes supérieures 10 à 20 fois plus grandes que les bornes inférieures. Sur cette taille d'instance, le taux de rendez-vous placés est de 78.456%, en légère baisse par rapport à la taille initiale. Ce taux chute encore plus sur les instances ayant des rendez-vous prioritaires (55.453%).

TABLE 3.6: Résultat sur les instances *CardioRehab* : de 160 à 480 rendez-vous.

CardioRehab.					**Lower bound**	**Upper bound**	**Rendez-vous**
Taille	**Importance**	**Urgence**	**Précédence**	**Disponibilités**			
160	0	0	0	0	0	0	160/160
160	40	0	0	0	0	0	160/160
160	80	0	0	0	0	0	160/160
160	0	40	0	0	12.537	32.321	147/160
160	0	80	0	0	57.927	141.624	149/160
160	0	0	27	0	0	0	160/160
160	0	0	36	0	0	0	160/160
160	0	0	0	19	0	0	160/160
160	0	0	0	38	0	0	160/160
160	80	40	27	19	12.924	43.566	141/160
160	160	80	36	38	58.797	185.415	149/160
480	0	0	0	0	0	154	326/480
480	120	0	0	0	0	498	314/480
480	240	0	0	0	0	656	304/480
480	0	120	0	0	168.726	3622.926	322/480
480	0	240	0	0	223.125	11145.423	16/480
480	0	0	38	0	0	154	326/480
480	0	0	48	0	0	154	326/480
480	0	0	0	60	0	163	317/480
480	0	0	0	122	0	122	358/480
480	240	120	38	60	182.738	6471.851	271/480
480	480	240	48	122	235.408	21690.747	5/480

Enfin, les résultats obtenus sur les instances ***CardioRehab*** sont présentés dans le tableau 3.6. Nos observations sur ces résultats sont très similaires aux résultats du scénario précédent. Sur les instances à 160 rendez-vous, la solution optimale est obtenue sur toutes celles ne comportant pas de rendez-vous urgents. Sur les autres, l'écart entre les bornes est faible (65,185), et le taux de rendez-vous planifiés reste satisfaisant (96.932%). Sur les instances de plus grande taille, les écarts se creusent entre les bornes se creusent. L'écart moyen entre les bornes sur les instances sans

rendez-vous prioritaire est de 271.571, contre 0 sur la taille initiale du scénario. Sur les autres, l'écart moyen est très important, particulièrement sur les instances comportant le plus de rendez-vous prioritaires, très difficiles à résoudre pour ce solveur. Le taux de rendez-vous planifiés subit une lourde chute, passant de 96.932 % sur les instances de 160 rendez-vous à 54.64% sur celles à 480 rendez-vous.

En observant l'ensemble des résultats du modèle de programmation linéaire implémenté sous CPLEX, nous constatons qu'il est performant sur les instances de petite taille ou ayant un nombre de combinaisons de solutions faible. Nous obtenons les solutions optimales sur de nombreuses instances ayant la taille initiale des scénarios. Néanmoins, des écarts parfois importants sont constatés entre les bornes inférieures et supérieures sur les instances comportant des actes prioritaires. Sans surprise, nous ne parvenons pas à obtenir de solution optimale sur les instances les plus grandes ou les plus difficiles. Pour obtenir de meilleurs résultats sur ces instances, nous nous intéresserons à des approches qui se sont montrées efficaces sur des problèmes similaires.

3.3.3 Instances PSPLIB

Le problème de planification que nous traitons est très similaire aux problèmes RCPSP, qui font l'objet de recherches très actives au sein de la communauté de la recherche opérationnelle. La bibliothèque Project Scheduling Problem Library (PSPLIB) fournit de nombreuses instances pour les différentes variations du RCPSP et sert de référence pour de nombreux articles. Elle fut constituée en 1996 par Kolisch et Sprecher à l'aide du générateur d'instance ProGen. Cet ensemble d'instances est divisé en quatre sous-ensembles nommés J30, J60, J90, J120 demandant respectivement de programmer 30, 60, 90 et 120 activités. Les ensembles J30, J60, J90 sont composés de 480 instances chacun. L'ensemble J120 est composé quant à lui de 600 instances. Les caractéristiques et les paramètres utilisés pour créer ces ensembles d'instances sont décrits dans Kolisch and Sprecher (1997).

Les résultats obtenus par notre modèle de programmation linéaire 0-1 et CPLEX nous permettront de mesurer les performances de nos approches sur ces instances. Néanmoins, certaines des plus grosses instances ne peuvent être traitées par CPLEX et les bornes fournies ne nous permettront pas d'évaluer l'efficacité de nos méthodes.

Nous avons donc choisi d'expérimenter également sur des instances de la littérature RCPSP, car ce problème est proche de ceux que nous traitons.

Nous n'avons fait aucune modification sur les instances PSPLIB. Notre problème et les problèmes RCPSP sont théoriquement proches, et seul l'objectif diffère de notre côté : la fonction objectif pour ces ensembles d'instances de PSPLIB minimise la durée totale du projet. Par conséquent nous utiliserons la fonction de l'équation 3.15 pour la résolution de ces instances. Dans cette fonction, la date de début de la première activité planifiée $b \in A$ est soustraite à la date de fin de la dernière activité planifiée $a \in A$, ceci afin d'obtenir la durée totale du projet, que nous cherchons à minimiser.

$$minimise\ max(t_a + duration_a) - min(t_b) \tag{3.15}$$

3.3.3.1 Différences avec les instances de l'entreprise

Des différences structurelles sont tout de même présentes entre les instances générées à partir des scénarios de l'entreprise et les instances PSPLIB. La principale différence provient des relations de précédences. Initialement absentes des scénarios rapportés par les planificateurs à l'entreprise, nous les avons ensuite ajouté aux instances. Ces précédences sont générées au sein des parcours de soins des patients et sont compartimentées par patients : il n'existe pas de liens de précédence entre les rendez-vous d'un patient $r_1 \in R$ et ceux d'un patient $r_2 \in R$. Prenons l'exemple d'un problème avec 15 rendez-vous $A = a_1, a_2, ..., a_15$ concernant 3 patients r_1, r_2 et r_3. La figure 3.4 illustre le graphe de précédence de ces rendez-vous obtenu avec la méthode décrite dans l'algorithme 1. C'est un graphe non connexe et relativement peu dense, avec peu d'arêtes par rapport au nombre de sommets. Cet exemple est représentatif des précédences de nos instances, peu nombreuses et ayant un impact moindre sur la structure du problème.

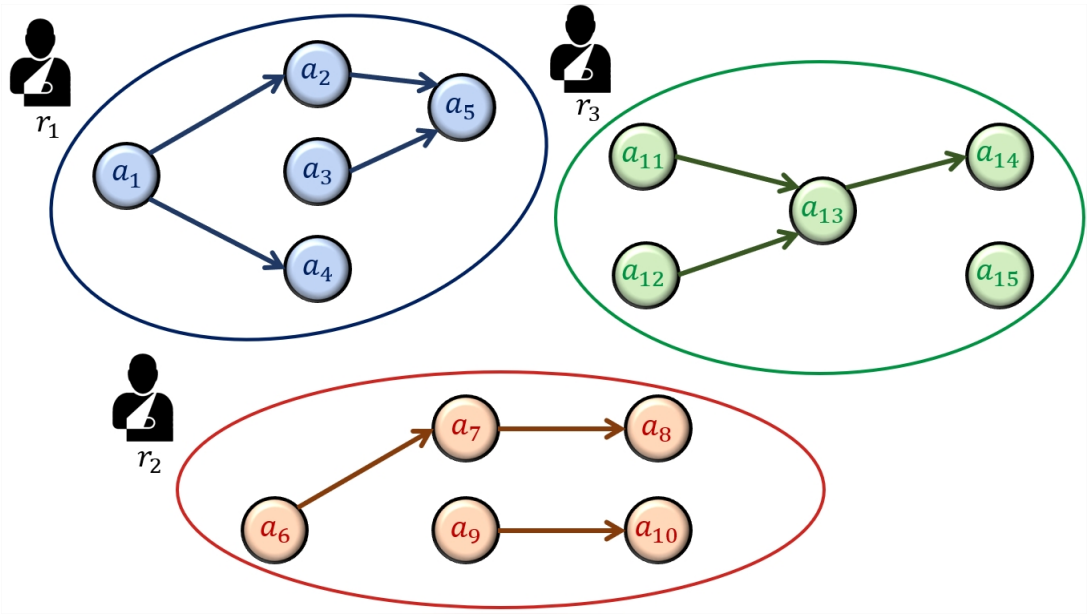


FIGURE 3.4: Graphe de précédence d'un exemple similaire aux instances de l'entreprise.

Les instances PSPLIB ont des structures différentes des nôtres, avec des relations de précédence beaucoup plus nombreuses et englobant l'ensemble des activités à planifier, car elles ne sont pas compartimentées en fonction des ressources, comme les nôtres. En reprenant l'exemple précédent sans limiter les relations à une ressource et à un degré maximum, nous obtenons le graphe de précédence illustré dans la figure 3.5, très semblable à ce que nous pouvons obtenir en réalisant le graphe de précédence d'une instance de l'ensemble PSPLIB. Nous voyons que ce graphe est connexe et bien plus dense que celui de l'illustration précédente. Les chaînes de précédence dans les instances PSPLIB sont plus grandes et plus denses que celles générés sur nos instances et ont donc un impact plus conséquent sur la difficulté des instances.

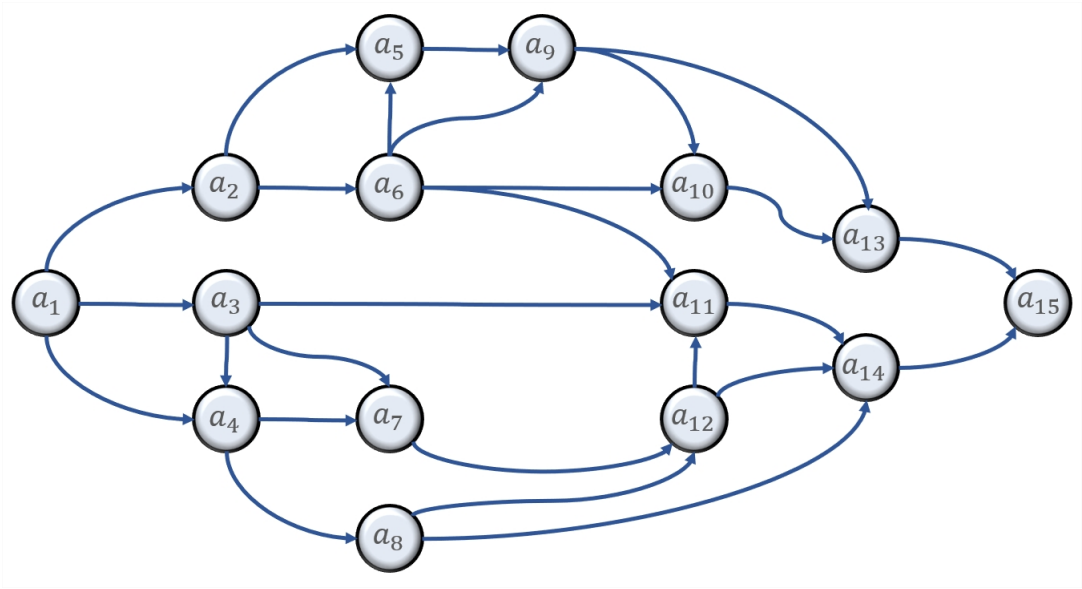


FIGURE 3.5: Graphe de précédence d'un exemple type PSPLIB.

3.3.3.2 Solutions

Les instances PSPLIB ont été utilisées dans de nombreux articles depuis leur création, et le générateur ProGen est lui aussi fréquemment cité par des auteurs créant leurs propres instances. Les chercheurs Kolisch et Hartmann ont publié plusieurs articles (Kolisch and Hartmann, 1999, Hartmann and Kolisch, 2000, Kolisch and Hartmann, 2006) comparant les performances obtenues par les méthodes exactes et heuristiques de la littérature. Plus récemment, Pellerin et al. (2020) proposent une étude des derniers résultats obtenus sur les instances PSPLIB. Nous comparerons nos résultats avec ceux fournis et disponibles sur le site de la librairie PSPLIB (Kolisch and Sprecher, 2020), regroupant les solutions optimales et les meilleures solutions trouvées jusqu'à maintenant.

Conclusion

Nous avons présenté dans ce chapitre la formulation de nos problèmes et de toutes ses composantes : les données, regroupant les différentes ressources et leurs disponibilités, la période de temps sur laquelle s'étend la planification et les rendez-vous et actes médicaux à planifier. Nous avons aussi formulé les contraintes permettant d'assurer la cohérence d'une solution.

Nous avons ensuite proposé le modèle mathématique représentant notre problématique, en définissant les variables de décision, les contraintes à respecter et l'objectif à atteindre. Ce modèle a ensuite été implémenté dans le solveur CPLEX pour obtenir des solutions optimales sur certaines de nos instances. Nous avons également obtenu des bornes inférieures et supérieures de la qualité des solutions, qui nous permettront d'évaluer les résultats que nous obtiendrons avec nos heuristiques.

Enfin, nous avons décrit formellement les scénarios de l'entreprise et nos méthodes de génération pour en obtenir des instances suffisamment variées et nombreuses pour nous permettre d'évaluer et de comparer nos différentes approches. Nous avons également introduit les instances PSPLIB, que nous utiliserons également pour mesurer les performances de nos méthodes.

Chapitre

4

Recherche locale

Contents

4.1 Méthode de construction aléatoire

4.2 Mouvements

4.2.1 Destruction aléatoire et reconstruction simple

4.2.2 Destruction aléatoire et reconstruction optimale

4.2.3 Destruction basée sur la difficulté et reconstruction optimale

4.2.4 Destruction ciblée

4.2.5 Échange de date de début

4.2.6 Décalage à gauche

4.3 Adaptive Large Neighborhood Search

4.3.1 Principe général

4.3.2 Formule d'ajustement des poids

4.4 Expérimentations & Résultats

4.4.1 Paramétrage

4.4.2 Instances de l'entreprise

4.4.3 Instances PSPLIB

Ce chapitre sera consacré à la description des mouvements et d'une méthode de recherche locale nommée **A**daptive **L**arge **n**eighborhood **S**earch (ALNS) que nous avons développée pour résoudre nos problèmes de planification. Dans un premier temps, nous décrirons un algorithme de construction qui nous permettra d'obtenir des solutions initiales satisfaisantes pour l'ALNS. Ensuite nous décrirons

l'ensemble des mouvements utilisés pour passer d'une solution à l'autre, et l'algorithme *LORH_ALNS*, qui les utilisera. Enfin, nous mènerons différents tests pour mesurer les performances de l'ALNS sur les instances de l'entreprise et de la littérature.

4.1 Méthode de construction aléatoire

Avant de nous intéresser aux méthodes de recherche locale, nous devons générer une solution initiale qui sera ensuite exploitée pour explorer l'espace de recherche, de voisin en voisin jusqu'à trouver des solutions de bonne qualité. Des méthodes telles que les algorithmes gloutons figurent parmi les plus populaires pour la génération de solutions initiales pour d'autres heuristiques. Ils sont rapides, relativement simples à concevoir et sont implémentés facilement. Le principe de ces algorithmes est relativement simple : à chacune de ses itérations, le glouton sélectionne le meilleur choix qui lui est disponible. Ce faisant, en sélectionnant les optimums locaux au fil de sa recherche, l'objectif de l'algorithme est de se rapprocher le plus possible de la ou des solutions optimales. Nous utiliserons une méthode moins déterministe dans ses choix que les algorithmes gloutons. Pour les problèmes de planification que nous traitons, notre approche construit la solution en planifiant les rendez-vous les uns après les autres, sans revenir sur les affectations précédentes au cours de la recherche. Toutefois, ces affectations ne sont pas nécessairement optimales, à la différence d'un glouton, et seront basées sur des choix aléatoires pour les ressources et les dates de début.

Avant d'aller plus loin dans la description de l'algorithme de construction, nous introduisons deux scores associés aux ressources et aux demandes de rendez-vous : α_r le score de stress d'une ressource r et δ_a le score de difficulté d'une demande a. α_r augmente à chaque fois qu'une ressource r ne peut être attribuée à un rendez-vous a, par manque de disponibilité. À l'inverse, ce score, initialement nul, décroît quand une ressource r est affectée à un rendez-vous a : les ressources r les plus sollicitées ou ayant moins de disponibilités connaîtront plus d'échecs d'attribution et auront un score α_r plus élevé. Ces variations sont calculées pour chaque tentative d'attribution des ressources, que ce soit pour la méthode de construction des solutions décrite ci-

dessous ou pour les différents mouvements de la recherche locale. Ainsi, nous gardons une trace des ressources les plus stressées au fil de la recherche.

Le score de difficulté δ_a attribué à chaque rendez-vous $a \in A$ est très similaire dans son fonctionnement au score de stress α_r décrit précédemment. Ce score, initialement nul, augmente lorsqu'un rendez-vous a n'a pas pu être planifié et décroît lorsqu'il est planifié avec succès. Les rendez-vous a qui ne parviennent pas ou peu à obtenir des affectations auront des scores δ_a plus élevés. Là aussi, ces scores sont calculés tout au long de la recherche et nous permettent de savoir quels rendez-vous sont les plus difficiles à placer. Ces deux scores seront utilisés dans certains des mouvements décrits dans la section suivante, mais mis à jour dès la construction de la solution initiale.

L'algorithme 2 a pour objectif de trouver des affectations aux demandes de rendez-vous $a \in A$. Pour cela, nous sélectionnons aléatoirement un rendez-vous a dans cet ensemble. Un ordre est ensuite calculé pour le traitement de a et de ses éventuels prédécesseurs et successeurs soit en planifiant la chaîne de précédence à laquelle a appartient à partir de la source (la première demande de la chaîne), soit en planifiant la chaîne à partir du rendez-vous a choisi.

À chaque demande a sont associés deux ensembles $pred_a$ et $succ_a$ éventuellement vides de prédécesseurs et de successeurs directs, respectivement. À chaque demande $b \in pred_a \cup succ_b$ sont aussi associés deux ensembles $pred_b$ et $succ_b$ de prédecesseurs et successeurs directs de b, et qui sont donc les prédecesseurs et successeurs indirects de a. La méthode $sortFromSource()$ construit l'ensemble des rendez-vous de la chaîne de précédence auquel appartient a et récursivement l'ensemble des demandes de rendez-vous identifiées dans les ensembles $pred_a$ et $succ_a$. La fonction $sortFromGivenAppointment()$ ordonne l'ensemble en partant de a, ajoute ses prédécesseurs directs et indirects, puis ses successeurs, directs ou indirects et progresse de proche en proche jusqu'à reconstituer l'ensemble de la chaîne.

Si, pour une raison quelconque, l'un des rendez-vous de la chaîne ne peut être planifié, nous planifions tout de même le reste de la chaîne de précédence si possible. La contrainte 3.1.2 décrite dans le chapitre précédent permet permet que certains rendez-vous de la chaîne ne soient pas planifiés, tant que l'ordre des relations de précédence est respecté par les rendez-vous planifiés. Pour notre algorithme de

Algorithme 2: algorithmConstruction()

Données: un ensemble de demandes de rendez-vous A

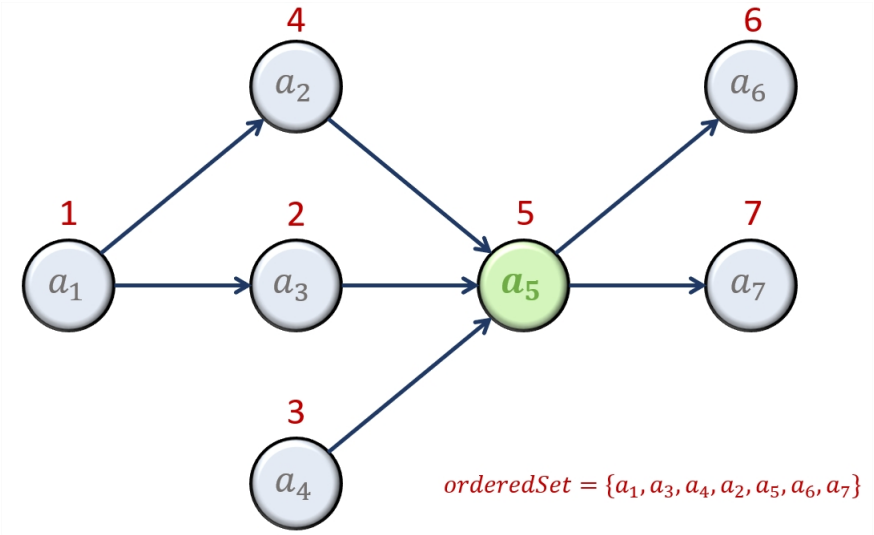
Résultat: une solution initiale Sol

```
1  Début
2      Sol ← ∅ ; Temp ← A
3      Tant que Temp ≠ ∅ faire
4          a ← selectRandom(Temp) ; Temp ← Temp \ {a}
5          Si pred_a ≠ ∅ alors
6              Si random() < 0.5 alors
7                  orderedSet ← sortFromSource(a,Sol)
8              Sinon
9                  orderedSet ← sortFromGivenAppointment(a,Sol)
10             Fin
11             Pour chaque rendez-vous b ∈ orderedSet faire
12                 Sol ← trySetSingleAppointment(a,Sol)
13             Fin
14         Sinon
15             Sol ← trySetSingleAppointment(a,Sol)
16         Fin
17     Fin
18     Retourne Sol
19 Fin
```

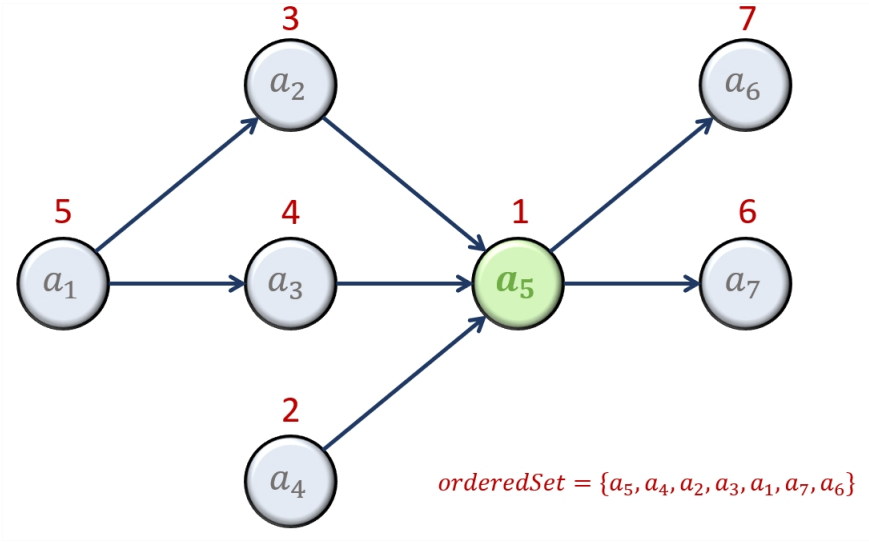
construction, nous posons la condition suivante : pour chaque demande a non planifiée et pour chaque demande $b \in pred_a$ et $c \in succ_a$, un nombre de créneaux égal à $duration_a$ devra séparer les dates de début de b et de c : $t_c \geq t_b + duration_a$. De cette manière il reste possible de planifier a en respectant ses relation de précédence si des ressources requises pour a se libèrent pendant la recherche.

Pour les deux méthodes, si a n'a aucun prédécesseur ou successeur, il sera l'unique élément présent dans $orderedSet$. Les figures 4.1a et 4.1b illustrent les ordres obtenus avec les deux méthodes. Un ensemble $A = \{a_1,...,a_7\}$ de rendez-vous et leurs liens de précédences sont représentés, et dans les deux cas, le rendez-vous choisi au

préalable est le rendez-vous a_5. La figure 4.1a montre l'ordre obtenu avec la méthode $sortFromSource()$, tandis que la figure 4.1b illustre l'ordre obtenu avec la méthode $sortFromGivenAppointment()$.



(a) Ordre calculé en partant de la source de la chaîne de précédence $pred_a$.



(b) Ordre calculé en partant de a.

FIGURE 4.1: Exemples d'ordres calculés en partant de la source et de a.

À partir de cet ensemble ordonné de rendez-vous $orderedSet$, nous tentons d'attribuer des ressources et une date de début à tous les rendez-vous $b \in orderedSet$ avec la fonction $trySetSingleAppointment()$.

La méthode $trySetSingleAppointment()$ est décrite dans l'algorithme 3. Elle attribue, si possible, une date de début et des ressources à un rendez-vous a. La recherche commence à partir d'un créneau t choisi soit aléatoirement dans l'horizon

Algorithme 3: trySetSingleAppointment()

Données: une demande de rendez-vous a et une solution Sol

Résultat: une solution Sol

```
1  Début
2      Si Emergency_a > 0 ou random() < probEarliestTimeslot alors
3          t_start ← ES_a
4      Sinon
5          t_start ← selectRandom([ES_a,LS_a])
6      Fin
7      t ← t_start
8      Tant que R_a ≠ ∅ et t ≠ t_start faire
9          R_a ← availableResources(a,t)
10         Si t = LS_a alors
11             t ← ES_a
12         Sinon
13             t ← t+1
14         Fin
15     Fin
16     Si R_a ≠ ∅ alors
17         Sol ← Sol ∪ { (a,t,R_a) }
18         δ_a ← δ_a−1
19     Sinon
20         δ_a ← δ_a+1
21     Fin
22     Retourne Sol
23 Fin
```

H, soit égal au début de l'intervalle de faisabilité du rendez-vous ES_a (notamment dans les cas où le rendez-vous a est un rendez-vous prioritaire). À chaque créneau $t \in H$, la méthode $availableResources()$ vérifie qu'il y a assez de ressources disponibles pour répondre exactement à la demande a. Si c'est le cas, la recherche s'arrête et le triplet (a,t_a,R_a) est ajouté à la solution Sol, avec t_a la date de début trouvée

et R_a l'ensemble des ressources affectées à a. Si la fonction $availableResources()$ ne parvient pas à trouver de ressources pour satisfaire la demande de a, la recherche reprend sur les créneaux suivants. Si aucun créneau ne permet de trouver les ressources nécessaire, le rendez-vous est laissé sans affectation et l'algorithme passe aux rendez-vous suivants.

L'algorithme 4 décrit la recherche de ressources pour répondre aux quantités de ressources requises $qtreq_a^\pi$ à partir du créneau t. Pour chaque propriété $\pi \in \Pi_a$, nous recherchons dans l'ensemble des ressources possédant cette propriété, $Possible\text{-}Resources$, des ressources disponibles pour toute la durée du rendez-vous. Si des ressources disponibles sont trouvées, elles sont ajoutées à l'ensemble R_a^π de ressources avec la propriété π temporairement affectées à a. Si le nombre de ressources dans cet ensemble correspond à la quantité requise $qtreq_a^\pi$, il est ajouté à R_a. Sinon, la demande ne pourra pas être satisfaite pour ce rendez-vous sur ce créneau t.

Pour illustrer la construction initiale de la solution et les divers mouvements que nous allons décrire ensuite, nous proposons un exemple simple d'un problème de planification. Les caractéristiques des différentes demandes $a \in A$ de l'exemple sont présentées dans le tableau 4.1. Ce problème est constitué de 7 demandes de rendez-vous, et nous indiquons pour chaque demande, dans l'ordre : la durée du rendez-vous, les quantités de ressources requises par propriété, l'ensemble des ressources pré-assignées, les coefficients d'importance et de priorité, l'ensemble des relations de précédence et l'intervalle de faisabilité. Les ressources $r \in R = \{ \ r_1,r_2,r_3,r_4,r_5 \ \}$ possèdent respectivement les propriétés $\pi \in \Pi = \{ \ \pi_1,\pi_2,\pi_3,\pi_4,\pi_5 \ \}$. L'horizon H s'étend sur une seule journée, de 7h à 20h.

Toutes les ressources sont disponibles sur tout l'horizon H, comme nous pouvons le voir sur la figure 4.2. Pour construire la solution initiale que nous nommons Sol, nous utilisons l'algorithme de construction que nous venons de présenter. À chaque itération, une demande $a \in A$ est sélectionnée aléatoirement. Prenons le cas où a_1 est choisie en premier. a_1 nécessite les ressources r_1 et r_2, la première car elle possède la propriété π_1 requises par a et la seconde car elle est pré-affectée. C'est un rendez-vous prioritaire ($Emergency_{a_1} = 18.75$), la recherche de créneau de début commence donc au début de l'intervalle de faisabilité. Pour cet exemple, la recherche

Algorithme 4: availableResources()

Données: un rendez-vous a, un créneau t

Résultat: un ensemble de ressources R_a

```
1  Début
2      R_a ← ∅
3      Si AreAvailable(PreAssigned_a,t) alors
4          Pour chaque propriété π ∈ Π_a faire
5              R_a^π ← ∅
6              PossibleResources ← Re_a^π
7              Tant que |R_a^π| ≠ qtreq_a^π et PossibleResources ≠ ∅ faire
8                  r ← selectRandom(PossibleResources)
9                  PossibleResources ← PossibleResources \ {r}
10                 Si IsAvailable(r,t) alors
11                     R_a^π ← R_a^π ∪ {r}
12                 Sinon
13                     α_r ← α_r+1
14                 Fin
15             Fin
16             Si |R_a^π| = qtreq_a^π alors
17                 R_a ← R_a ∪ {R_a^π}
18             Sinon
19                 Retourne ∅
20             Fin
21         Fin
22     Fin
23     Pour chaque ressource r ∈ R_a faire
24         α_r ← α_r−1
25     Fin
26     Retourne R_a
27 Fin
```

de ressources disponibles s'achève rapidement car r_1 et r_2 peuvent être affectées à a_1 dès le premier créneau t_0. Le triplet $(a_1,t_0,\{r_1,r_2\})$ est ajouté à la solution Sol.

TABLE 4.1: Caractéristiques des demandes $a \in A$ de l'exemple.

a	$duration_a$	$qtreq$	$PreAssigned_a$	$Essential_a$	$Emergency_a$	$pred_a$	ES_a	LS_a
a_1	2	$\pi_1 = 1$	$\{ r_2 \}$	25	18.75	$\emptyset$	t_0	t_{12}
a_2	3	$\pi_4 = 1$	$\{ r_3 \}$	1	0	$\emptyset$	t_0	t_{12}
a_3	2	$\pi_1 = 1$ $\pi_2 = 1$ $\pi_4 = 1$ $\pi_5 = 1$	$\{ r_3 \}$	25	0	$\emptyset$	t_0	t_{12}
a_4	3	$\pi_2 = 1$	$\{ r_4 \}$	5	3.75	$\emptyset$	t_0	t_{12}
a_5	3	$\pi_2 = 1$	$\{ r_1 \}$	1	0	$\emptyset$	t_0	t_{12}
a_6	2	$\pi_1 = 1$	$\{ r_2 \}$	5	0	$\emptyset$	t_0	t_{12}
a_7	3	$\pi_4 = 1$	$\{ r_5 \}$	125	93.75	$\emptyset$	t_0	t_{12}

FIGURE 4.2: Disponibilités des ressources $r \in R$.

Ce processus se répète pour chaque demande de rendez-vous $a \in A$. Dans la figure 4.3, nous représentons les affectations de ressources pour tous les rendez-vous planifiés, et l'ordre dans lequel les demandes ont été traitées. La demande a_5 a par exemple été traitée après a_1. Par exemple, le rendez-vous a_7 qui est prioritaire n'a pu avoir de date de début proche de ES_a, les ressources r_4 et r_5 étant déjà occupées par des rendez-vous gérés avant lui. Avec cet ordre et ces affectations, il n'a pas été possible d'affecter des ressources à la demande a_4 (la ressource r_2 n'ayant plus

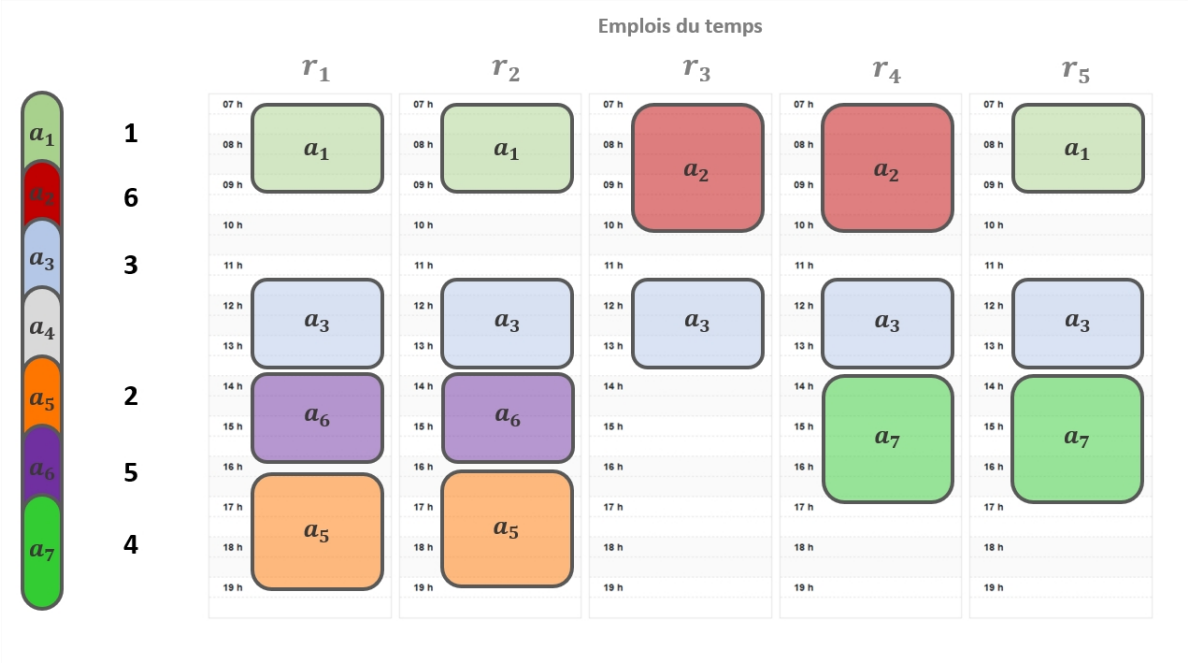


FIGURE 4.3: Solution *Sol* obtenue après la construction initiale, avec l'ordre de traitement des rendez-vous et les emplois du temps des ressources.

assez de disponibilité), cette demande reste donc sans affectation à la sortie de la construction initiale.

La complexité de l'algorithme de construction est en $O(|A|\times|H|\times|R|)$ dans le pire des cas, qui nécessiterait de rechercher pour toutes les demandes $a \in A$ des ressources disponibles sur l'ensemble des créneaux de l'horizon. Dans le reste de ce mémoire, nous nommerons cet algorithme $LORH_ARC$.

4.2 Mouvements

Les méthodes de recherches locales sont depuis longtemps utilisées pour résoudre des problèmes d'optimisation combinatoire, parmi lesquels figurent de nombreux problèmes de planification. Le recuit simulé par exemple, dont les principes ont été proposés dans l'article de Kirkpatrick et al. (1983), est toujours utilisé pour résoudre des problèmes RCPSP (Laurent et al., 2017) ou plus généralement des problèmes de planification (Gallo and Capozzi, 2019, Pang et al., 2018). Les recherches locales à voisinages larges (Shaw, 1998) sont un autre exemple de recherches locales, elles explorent l'espace de recherche en détruisant et en reconstruisant une solution selon différentes méthodes. Elles sont utilisées depuis des décennies pour résoudre des pro-

blèmes complexes similaires à nos problèmes de planification (Chakrabortty et al., 2016).

Ces différentes approches construisent leurs solutions en explorant leur voisinage. Le voisinage d'une solution Sol est un ensemble de solutions Sol' qui sont obtenues par l'application d'un mouvement $m \in M$ sur une solution Sol, $Sol' = m(Sol)$. Ce voisinage est par conséquent défini par les mouvements utilisés pour explorer l'espace de recherche. Ils vont guider la recherche et ont donc un impact majeur sur les performances des recherches locales. Certains effectuent de légères modifications sur les solutions : ajouter et retirer un élément à la solution, échanger deux éléments, etc. D'autres sont plus importants et modifient des pans entiers de la solution. Pour notre méthode de recherche locale, nous avons élaboré différents mouvements spécifiques au problème que nous traitons. Dans les sections qui suivent, nous détaillerons les mouvements que nous avons créés pour explorer les voisinages de nos solutions.

Les mouvements que nous avons élaborés ont pour objectif d'améliorer la solution courante. Pour cela, le principe général est de détruire une partie de la solution puis de tenter de la reconstruire en espérant que la solution résultante soit meilleure que la précédente. Ces mouvements sont décrits dans les sous-sections suivantes.

4.2.1 Destruction aléatoire et reconstruction simple

Algorithme 5: randomDestruction()

Données: une solution Sol et un entier k

Résultat: une solution Sol avec k rendez-vous planifiés de moins

```
1 Début
2   Pour i allant de 0 à k faire
3     a ← selectRandom(A_Sol)
4     Sol ← Sol \ { (a,t_a,R_a) }
5   Fin
6   Retourne Sol
7 Fin
```

Le premier mouvement que nous décrivons consiste à retirer certains rendez-vous sélectionnés aléatoirement de la solution courante avant de tenter de planifier le plus

de rendez-vous possibles. La procédure de destruction est décrite dans l'algorithme 5. L'ampleur du mouvement est contrôlée par k, indiquant le nombre de rendez-vous planifiés maximum à retirer de la solution actuelle. Logiquement, plus ce nombre est grand, plus la solution voisine sera éloignée de la solution courante.

Algorithme 6: randomDestructionGreedyReconstruction()

Données: une solution Sol et un entier k_{max}

Résultat: une solution Sol

1 **Début**
2 $randomDestruction(Sol,random(1,k_{max}))$
3 **Pour chaque demande de rendez-vous** $a \in A_{\overline{Sol}}$ **faire**
4 $trySetSingleAppointment(a,Sol)$
5 **Fin**
6 **Retourne** Sol
7 **Fin**

Nous décrivons le mouvement dans l'algorithme 6. Un nombre k de rendez-vous $a \in A_{Sol}$ sont choisis aléatoirement et retirés de la solution Sol avec la méthode de destruction décrite précédemment. Le mouvement tente ensuite de planifier l'ensemble des rendez-vous non-planifiés $A_{\overline{Sol}}$, qui sont sans affectation soit du fait de la destruction précédente ou parce qu'ils n'étaient initialement pas planifiés dans Sol.

Cette reconstruction partage la même complexité temporelle que l'algorithme glouton : $O(|A_{\overline{Sol}}|\times|H|\times|R|)$ dans le pire des cas, le traitement pour retirer les rendez-vous étant négligeable. Dans le pire des cas, tous les créneaux $t \in H$ sont explorés pour trouver des ressources. Cependant si des ressources répondant au besoin du rendez-vous sont trouvées, la recherche s'arrête. L'ampleur du mouvement est également contrôlée avec le paramètre k et nous permet de détruire de petites parties de la solution ou de faire des coupes beaucoup plus grandes qui vont la remodeler complètement.

4.2.2 Destruction aléatoire et reconstruction optimale

Le mouvement de destruction aléatoire et de reconstruction optimale, à l'instar du mouvement précédent, retire aléatoirement des rendez-vous à la solution courante

avant de tenter d'en replacer un maximum. Cependant, lors de la reconstruction, le mouvement tente de planifier les rendez-vous de la meilleure manière possible.

Algorithme 7: randomDestructionOptimalReconstruction()

Données: une solution Sol et un entier k

Résultat: une solution Sol

1 **Début**
2 $randomDestruction(Sol,random(1,k_{max}))$
3 $\lambda \leftarrow A_{\overline{Sol}}$
4 **Pour** i **allant de** 0 **à** $|\lambda|$ **faire**
5 $a \leftarrow rouletteWheel(\lambda,\frac{P_a}{\sum_{j=0}^{N} P_a})$
6 $\lambda \leftarrow \lambda \setminus \{ a \}$
7 $\zeta \leftarrow findPossibleSlots(a,Sol)$
8 $\xi \leftarrow R_i^t \mid t \in \zeta \mid R_i^t$ satisfait a
9 $(t_a,R_i^t) \leftarrow \underset{t \in \zeta}{\text{argmin}}[stress(R_i^t),R_i^t \in \xi]$
10 **Si** $R_a \neq \emptyset$ **alors**
11 $Sol \leftarrow Sol \cup (a,t_a,R_a)$
12 **Fin**
13 **Fin**
14 **Retourne** Sol
15 **Fin**

L'algorithme 7 décrit ce mouvement. La méthode de destruction est identique : un nombre k de rendez-vous $a \in A_{Sol}$ sont retirés de la solution Sol. Ensuite, la reconstruction tente de planifier le plus de rendez-vous $a \in A_{\overline{Sol}}$ possible. L'ordre dans lequel ils seront traités est obtenu avec une sélection par roulette, dont les probabilités sont calculées selon la formule suivante :

$$p_a \frac{C_a}{\sum_{j=0}^{N} C_a} \tag{4.1}$$

Avec C_a la somme des coefficients d'importance et de priorité d'un rendez-vous a. De cette manière, les rendez-vous prioritaires et importants ont plus de chance d'apparaître en premier dans l'ordre de reconstruction, et d'être planifiés plus tôt.

Pour tous les rendez-vous $a \in A_{\overline{Sol}}$, la fonction $findPossibleSlots()$ calcule pour chaque créneau $t \in [ES_a, LS_a]$ les ensembles possibles de ressource R_a. Parmi ces ensembles $R_i^t \in \xi$, celui dont la somme du score de stress α_r de toutes ses ressources $r \in R_i^t$ est le plus bas est choisi et attribué à la demande a, avec le créneau t associé.

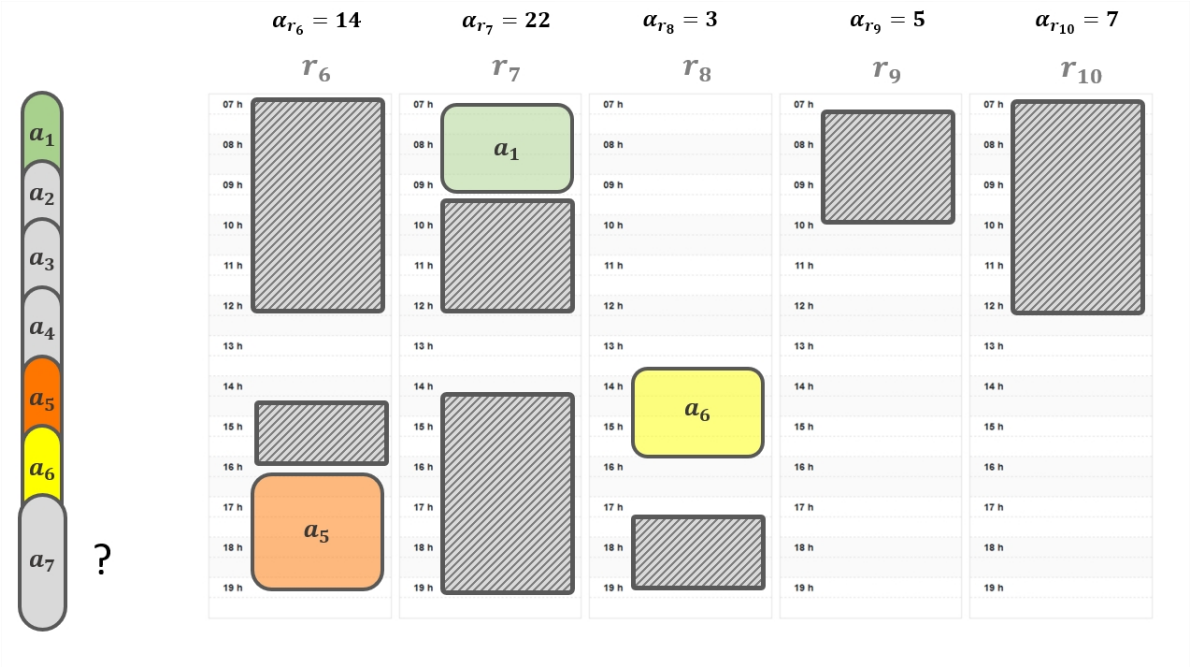


FIGURE 4.4: Emplois du temps des ressources r_6 à r_{10} et état de la solution Sol.

L'heuristique favorise pour un rendez-vous a l'ensemble de ressource R_a avec le score de stress cumulé le plus bas. Les rendez-vous qui ne peuvent être planifiés restent dans $A_{\overline{Sol}}$. Pour illustrer cette notion de stress, nous utilisons à nouveau l'exemple décrit précédemment, avec une légère modification : tous les rendez-vous $a \in A$ nécessitent en plus des autres propriétés une ressource possédant la propriété π_6. Cette propriété est possédée par 5 nouvelles ressources $r_6, r_7, ..., r_{10}$. Les emploi du temps de ces ressources sont représentés sur la figure 4.4. Nous utilisons à nouveau la solution Sol construite par l'algorithme 2 de construction dans l'exemple précédent. Le mouvement de destruction aléatoire lui a été appliqué et 5 des demandes nécessitent encore une affection. Sur les emplois du temps des ressources, nous avons représenté en cadres gris hachurés des périodes d'indisponibilité, et au dessus, les scores α de chacune des ressources.

Certaines ressources, comme r_6 et r_7 ont de grosses périodes d'indisponibilité, et sont par ailleurs occupées par d'autres rendez-vous. Elles ont donc de grandes

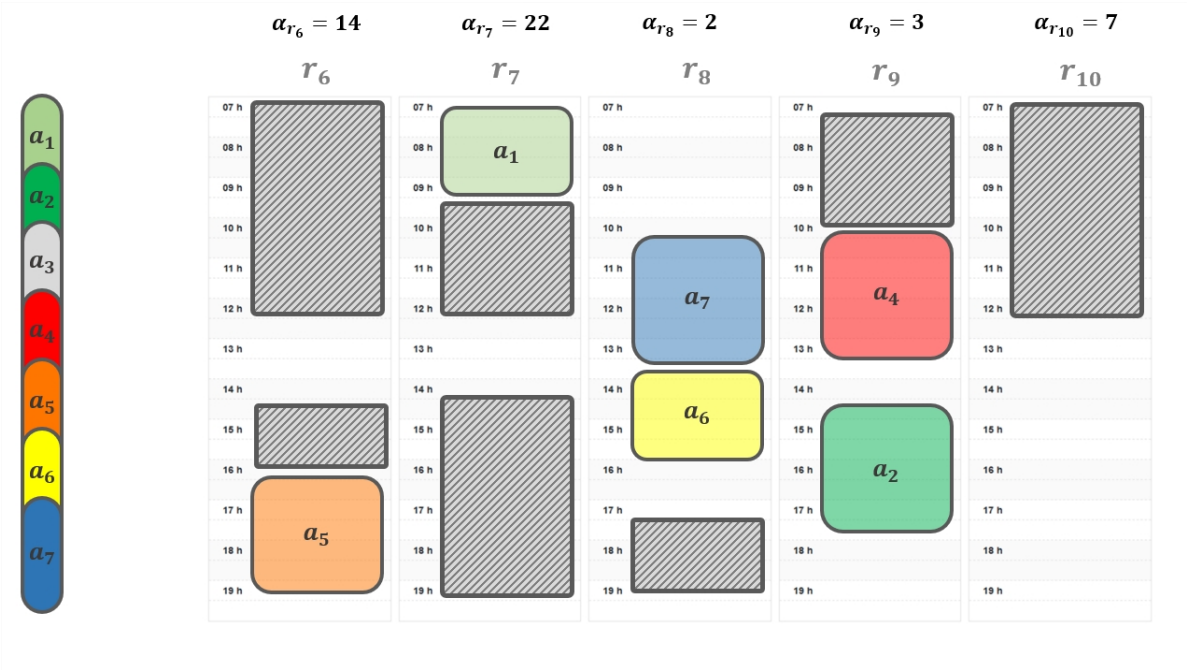


FIGURE 4.5: Emplois du temps des ressources après l'affectation des rendez-vous a_2, a_4 et a_7.

chances de ne pas pouvoir être affectées à une nouvelle demande et ont donc des scores α_{r_6} et α_{r_7} élevés. À l'inverse, les ressources r_8, r_9 et r_{10} sont beaucoup plus disponibles et ont des scores α plus faibles. Elles seront donc privilégiées dans les choix faits par la construction optimale, pour préserver le peu de disponibilité restant aux autres ressources, pour qu'elles puissent être affectées aux rendez-vous où elles sont indispensables. Dans notre exemple, le rendez-vous a_7 a été sélectionné en premier pour recevoir une affection. Ce dernier nécessite les propriétés des ressources r_4 et r_5, qui possèdent suffisamment de disponibilités et ne sont pas représentées dans cet exemple. Dans cette situation, la fonction $findPossibleSlots()$ renvoie $\zeta = \{ t_3,t_4,...,t_12\}$, avec tous les créneaux où suffisamment de ressources sont disponibles. Comme illustré sur la figure 4.5, pour la dernière propriété requise π_6, le mouvement privilégie la ressource ayant le score α le plus bas, dans notre cas, r_8. Pour les demandes restantes, le mouvement utilise les ressources les moins stressées. Dans notre exemple, les ressources r_8 et r_9 sont donc privilégiées et nous obtenons au final la solution Sol illustrée en figure 4.5. Les disponibilités des ressources r_1 à r_5, aussi impactées par ces affectations, seront représentées dans l'exemple suivant.

La complexité de ce mouvement est en $O(|A_{\overline{Sol}}|\times|H|\times|R|)$. L'ensemble des créneaux $t \in H$ seront explorés quoi qu'il arrive : toutes les associations de créneau t_a

et de ressources R_a sont envisagés. L'idée derrière ce mouvement est de préserver les ressources les plus demandées pour les rendez-vous où elles sont indispensables, lorsqu'elles sont pré-affectées par exemple.

4.2.3 Destruction basée sur la difficulté et reconstruction optimale

Ce mouvement utilise une mécanique de reconstruction similaire au précédent, mais le choix des rendez-vous à détruire est différent : il se base sur la difficulté intrinsèque des rendez-vous pour détruire les plus faciles à replacer pour éventuellement libérer des ressources afin de planifier les plus difficiles.

Algorithme 8: difficultyBasedDestruction()

Données: une solution Sol et un entier k

Résultat: une solution Sol

1 **Début**

2 $a \leftarrow selectRouletteWheelDifficulty(A_{Sol}, \frac{\delta_a}{\sum_{j=0}^{N} \delta_a})$

3 $Sol \leftarrow Sol \setminus \{ (a,t_a,R_a) \}$

4 **Pour** i **allant de** 0 **à** $k-1$ **faire**

5 $a \leftarrow selectRandom(A_{Sol})$

6 $Sol \leftarrow Sol \setminus \{ (a,t_a,R_a) \}$

7 **Fin**

8 **Fin**

Pour ce mouvement, deux sélections décrites dans l'algorithme 8 sont dédiées aux rendez-vous à retirer : un rendez-vous est choisi en fonction de son score de difficulté δ_a selon la fonction suivante :

$$p_a = \frac{\delta_a}{\sum_{j=0}^{N} \delta_a} \tag{4.2}$$

où δ_a est le score de difficulté d'un rendez-vous a. Les autres rendez-vous sont sélectionnés aléatoirement. L'objectif est de retirer des rendez-vous pour permettre de planifier un rendez-vous identifié comme difficile à un autre moment ou avec

d'autres ressources que celles qui y sont affectées actuellement. Une fois ces rendez-vous retirés, ce mouvement tente de placer tous les rendez-vous $a \in A_{\overline{Sol}}$ de manière optimale. Ce mouvement a également une complexité temporelle de $O(|A_{\overline{Sol}}| \times |H| \times |R|)$. Nous souhaitons ainsi pouvoir faire des modifications sur les rendez-vous les plus difficiles, en maximisant les chances qu'ils puissent être planifiés à nouveau en retirant d'autres rendez-vous plus faciles à placer.

4.2.4 Destruction ciblée

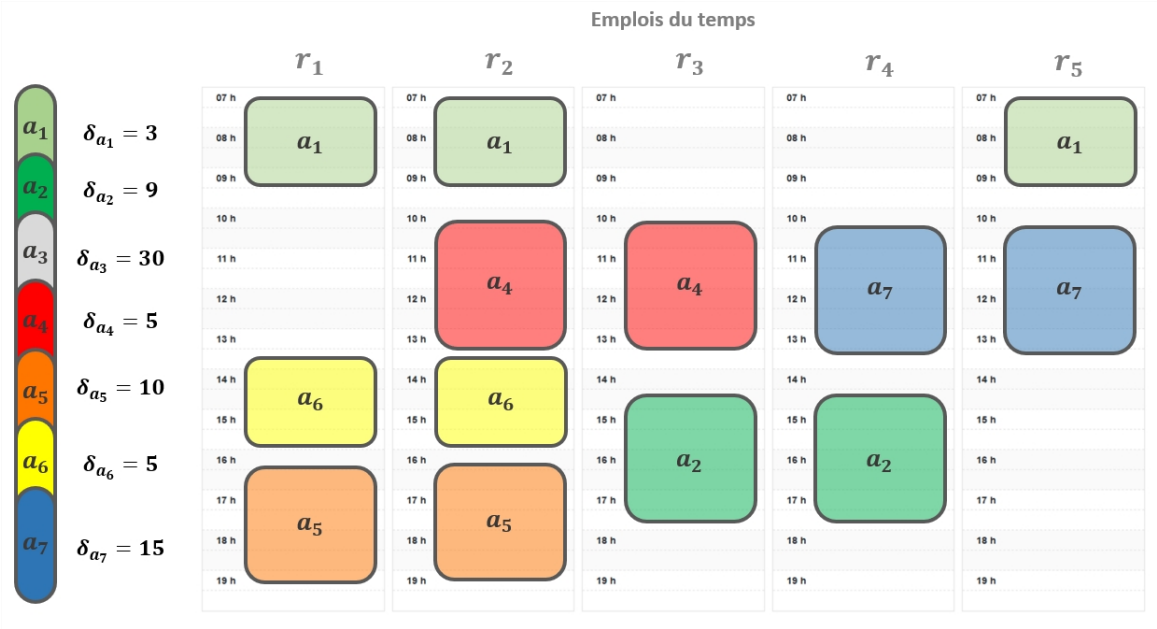


FIGURE 4.6: Ensemble de rendez-vous planifiés au sein d'une solution Sol, avec les scores de difficultés δ_a de chaque demande.

Avec ce dernier mouvement, nous souhaitons placer les rendez-vous ayant des scores δ_a les plus élevés (et donc les rendez-vous les plus difficiles) qui n'ont pas encore d'affectations de ressources et de date. Pour cela, certains rendez-vous avec des scores δ_a faibles (donc plus faciles à placer) sont déprogrammés afin de libérer des ressources. Nous illustrons cette situation dans la figure 4.6, reprenant une nouvelle fois le problème décrit en début de chapitre : nous avons 7 demandes de rendez-vous à placer, chacune nécessitant diverses propriétés. La figure 4.6 représente une solution avec le score de difficulté associé à chaque demande a. La demande a_3 n'a toujours pas reçu d'affectation, et figure parmi les demandes ayant les scores δ les plus élevés. Pour être réalisable, ce rendez-vous nécessite les propriétés π_1, π_2, π_4,

et π_5 possédées respectivement par les ressources r_1, r_2, r_4 et r_5. La ressource r_3 est pré-affectée à a_3 et doit également être présente. Le rendez-vous a_3 à une durée de 2 créneaux. Nous donnons les emplois du temps de ces ressources. Ceux-ci sont très largement occupés par les autres rendez-vous, et il n'est plus possible de planifier a_3.

Notre objectif avec ce mouvement est retirer un ensemble $\Upsilon_i = \{ (b_{i1},t_{ib_1},R_{ib_1}),$-$...,(b_{in},t_{ib_n},R_{ib_n}) \}$, $n \in \{ 1,...|Sol| \}$ de rendez-vous pour permettre la planification d'un autre, a_3 dans ce cas précis. L'ensemble $\Upsilon = \{\Upsilon_0,...,\Upsilon_i\}$ est composé de couples (Υ_i,t), avec Υ_i un ensemble de rendez-vous dont le retrait permettrait de planifier le rendez-vous a sélectionné (dans notre exemple, a_3) et t le créneau sur lequel planifier ce rendez-vous. Avant tout, le choix de la demande traitée par ce mouvement se fait grâce à une sélection par roulette avec la fonction de probabilité décrite dans l'équation 8 déjà utilisée précédemment. La priorité sera donc donnée aux rendez-vous ayant les scores δ_a les plus élevés.

La recherche de ces ensembles Υ_i de rendez-vous à retirer se déroule sur un nombre de créneaux $t \in T$ déterminé par un taux d'échantillonnage β. Ce taux nous permet de contrôler l'ampleur de la recherche et de restreindre le nombre de créneaux sur lesquels nous cherchons à planifier a, qui seront sélectionnés dans $[ES_a,LS_a]$. Pour décrire ce mouvement, nous introduisons deux éléments : le couple $(RP_a^t,\ qtd_t^{PA})$ et l'ensemble PI_a^t.

Le couple $(RP_a^t,\ qtd_t^{PA})$ est composé de RP_a^t, l'ensemble des ressources pré-affectées à a étant occupées par d'autres rendez-vous sur l'intervalle $[t,\ t{+}duration_a]$ et qtd_t^{PA} la quantité de ressources pré-affectées à a ayant suffisamment de disponibilités sur ce même intervalle $[t,\ t{+}duration_a]$. Il faut ici bien faire la différence entre les ressources qui ne sont pas utilisables car déjà occupées ailleurs et celles qui ne sont pas disponibles du tout sur cet intervalle de temps, pour a ou pour tout autre rendez-vous. Dans le premier cas, il est possible de libérer ces ressources pour qu'elles soient attribuées à a. Dans le second cas, où $qtd_t^{PA} \neq |PreAssigned_a|$, nous ne pouvons rien faire et a ne peut pas débuter au créneau t.

$PI_a^t = \cup_{\pi_i \in \Pi_a}(\pi_i,\ qtm_a^{\pi_i},\ R_{\pi_i})$ est un ensemble de triplets de propriétés π_i requises par a et manquantes à l'instant t, du nombre $qtm_a^{\pi_i}$ de ressources possédant la propriété π_i disponibles au créneau t et de l'ensemble R_{π_i} de ressources possédant la

propriété π_i mais déjà occupées sur le créneau t. Ces deux éléments nous permettent de savoir quelles ressources peuvent être libérées pour pouvoir planifier a. L'ensemble PI_a^t peut être vide pour deux raisons : soit il y a assez de ressources libres au créneau t pour planifier a, soit trop peu ont des disponibilités suffisantes pour que le rendez-vous ait lieu. Dans le premier cas, il n'est pas nécessaire de retirer des rendez-vous déjà planifiés pour trouver les ressources qui ne sont pas pré-assignées. Dans le second cas, encore une fois, nous ne pouvons rien faire et a ne pourra être planifié au créneau t.

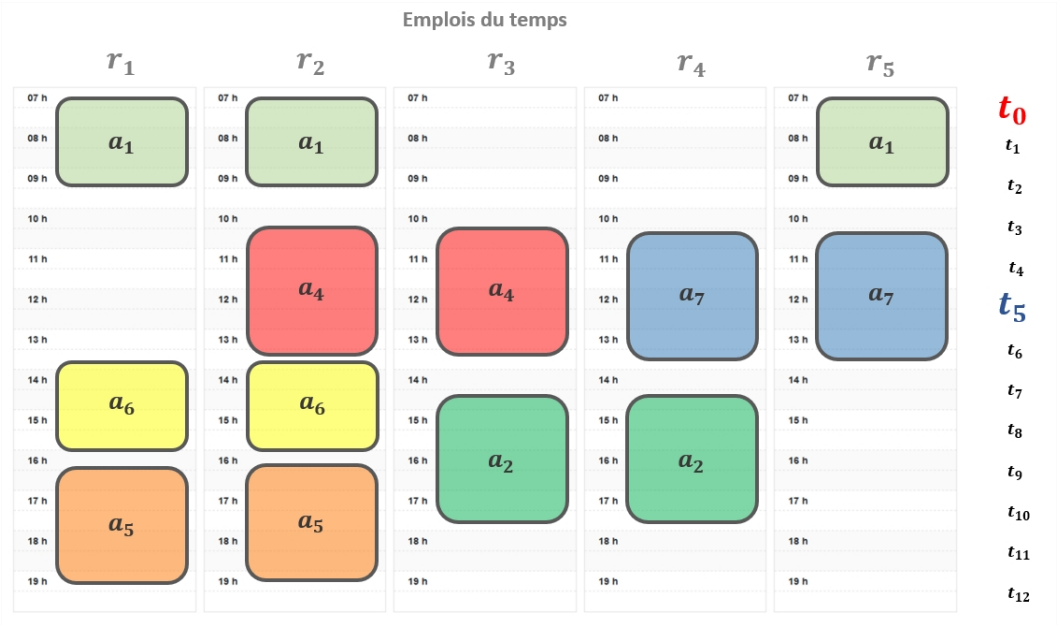


FIGURE 4.7: Illustration des créneaux correspondant à chaque horaire de l'emploi du temps.

Nous illustrons ces éléments dans la figure 4.7 avec quelques exemples tirés de la situation décrite dans la figure précédente. Chaque horaire visible sur ces illustrations correspond à une heure de début et $t_0 = 7$h, $t_1 = 8$ h, etc.. Pour rappel, dans cet exemple, les seules ressources possédant les propriété π_1,π_2,π_4,π_5 requises pour a sont représentées, ainsi que la ressource r_3, qui est pre-affectée à a. Si nous prenons le créneau t_0 et pour la demande a_3 nous avons $RP_{a_3}^{t_0} = \emptyset$ et $qtd_{t_0}^{PA} = 1$, car la ressource r_3 est disponible. L'ensemble $PI_{a_3}^{t_0}$ est composé des triplets (π_1,0,{r_1}), (π_2,0,{r_2}) et (π_5,0,{r_5}) car les ressources r_1, r_2 et r_5 possédant respectivement les propriétés π_1, π_2 et π_5 sont déjà occupées par le rendez-vous a_1. Dans cette configuration, le retrait du rendez-vous a_1 permettrait de planifier a_3. Nous pouvons donc ajouter

le couple ($\Upsilon_0 = \{(a_1,t_0,R_{a_1})\},t_0$) à Υ. Prenons un autre créneau : pour t_5, nous avons $RP_{a_3}^{t_5} = \{r_3\}$ et $qtd_{t_0}^{PA} = 1$, car la ressource r_3 est occupée par a_4. L'ensemble $PI_{a_3}^{t_5}$ est composé des triplets (π_1,0,$\{r_1\}$), (π_2,0,$\{r_2\}$), (π_4,0,$\{r_4\}$) et (π_5,0,$\{r_5\}$) car toutes les ressources qui ne sont pas pré-affectées mais possèdent les propriétés requises par a_3 sont occupées par les rendez-vous a_4, a_6 et a_7. Nous ajoutons donc le couple ($\Upsilon_1 = \{(a_4,t_3,R_{a_4}),(a_6,t_6,R_{a_6}),(a_7,t_3,R_{a_7}\},t_5$) à Υ. Nous pourrions répéter ce processus pour chaque créneau et obtenir l'ensemble des combinaison possibles de retrait pour pouvoir planifier a_3.

Il est aisé de voir sur la figure 4.7 quels rendez-vous nous devons retirer à chaque créneau afin de pouvoir y planifier a_3. L'algorithme 9 présente la fonction récursive permettant de calculer toutes les combinaisons possibles de rendez-vous à retirer pour planifier un rendez-vous a au créneau t. Pour cela nous parcourons dans un premier temps l'ensemble RP_a^t pour libérer les ressources pré-affectées à a. La fonction $UpdateRP$ permet de mettre à jour RP_a^t avant chaque appel récursif pour répercuter le retrait des rendez-vous pour libérer les ressources pré-affectées. Lorsque RP_a^t est vide, nous cherchons les combinaisons de rendez-vous permettant de libérer le nombre exact de ressources nécessaires à la demande a. Le principe est très similaire à la méthode utilisée pour les ressources pré-affectées : nous libérons et calculons les rendez-vous à retirer pour libérer les ressources une par une. La fonction $UpdatePI$ met à jour PI_a^t au fil des rendez-vous retirés avant l'appel récursif à la fonction. Les appels récursifs se poursuivent jusqu'à libérer à la fois les ressources pré-affectées ($RP_a^t = \emptyset$) et suffisamment de ressources pour a ($PI_a^t = \emptyset$).

Algorithme 9: minimumFreeResource

Données: une demande de rendez-vous a, les ensembles PI_a^t et RP_a^t, un créneau t, les ensembles Υ et Υ_i et une solution Sol

Résultat: un ensemble Υ mis à jour

1 **Début**
2 **Si** $PI_a^t = \emptyset$ *and* $RP_a^t = \emptyset$ **alors**
3 $\Upsilon \leftarrow \Upsilon \cup \{\Upsilon_i\}$
4 **Sinon**
5 **Pour chaque triplet** $(t_{a'},R_{a'},a') \in Sol \mid r \in R_{a'} \mid r \in RP_a^t$ **faire**
6 **Si** a' **débute ou se termine entre** t **et** $t{+}duration_a$ **alors**
7 $TempPI_a^t \leftarrow PI_a^t$; $TempRP_a^t \leftarrow RP_a^t$
8 $UpdateRP(RP_a^t, Sol\backslash\{ (t_{a'},R_{a'},a') \},(t_{a'},R_{a'},a'),r)$
9 $UpdatePI(PI_a^t, Sol\backslash\{ (t_{a'},R_{a'},a') \},(t_{a'},R_{a'},a'))$
10 $minimumFreeResource(a,PI_a^t,RP_a^t,\Upsilon,\Upsilon_i \cup \{ (t_{a'},R_{a'},a') \},Sol \setminus \{ (t_{a'},R_{a'},a') \})$
11 $PI_a^t \leftarrow TempPI_a^t$; $RP_a^t \leftarrow TempRP_a^t$
12 **Fin**
13 **Fin**
14 **Pour chaque triplet** $(\pi_i,qtm_a^{\pi_i},R_{\pi_i}) \in PI_a^t$ **et pour chaque ressource** $r \in R_{\pi_i}$ **faire**
15 $\Upsilon_j \leftarrow \emptyset$
16 **Pour chaque triplet** $(t_{a'},R_{a'},a') \in Sol \mid r \in R_{a'}$ **faire**
17 **Si** a' **débute ou se termine entre** t **et** $t{+}duration_a$ **alors**
18 $\Upsilon_j \leftarrow \Upsilon_j \cup \{ (t_{a'},R_{a'},a') \}$
19 **Fin**
20 **Fin**
21 $TempPI_a^t \leftarrow PI_a^t$; $UpdatePI(PI_a^t,Sol \backslash\{ \Upsilon_j \},\Upsilon_j)$
22 $minimumFreeResource(a,PI_a^t,RP_a^t,\Upsilon,\Upsilon_i \cup\{ \Upsilon_j \},Sol \setminus \{ \Upsilon_j \})$
23 $PI_a^t \leftarrow TempPI_a^t$
24 **Fin**
25 **Fin**
26 **Fin**

Algorithme 10: targetedDestruction

Données: une solution Sol et un réel β

Résultat: une solution Sol

1 **Début**

2 $a \leftarrow selectRouletteWheel(A_{\overline{Sol}}, \frac{\delta_a}{\sum_{j=0}^{N} \delta_a})$

3 $T \leftarrow sample([ES_a, LS_a], \beta)$; $\Upsilon \leftarrow \emptyset$; $\Upsilon_i \leftarrow \emptyset$

4 **Pour chaque créneau** $t \in T$ **faire**

5 $PI_a^t \leftarrow NAvailable(a, t, Sol)$

6 $(RP_a^t, qtd_{t_0}^{PA}) \leftarrow NAvailablePA(a, t, Sol)$

7 **Si** $qtd_{t_0}^{PA} = |PreAssigned_a|$ **et** $PI_a^t \neq \emptyset$ **et** $RP_a^t \neq \emptyset$ **alors**

8 $minimumFreeResource(a, PI_a^t, RP_a^t, \Upsilon, \Upsilon_i, Sol)$

9 **Fin**

10 **Fin**

11 **Si** $\Upsilon \neq \emptyset$ **alors**

12 $\Upsilon^* \leftarrow rouletteWheel(\Upsilon, Sol,)$

13 **Fin**

14 **Pour chaque triplet** $(a', t_{a'}, R_{a'}) \in \Upsilon^*$ **faire**

15 $Sol \leftarrow Sol \setminus \{ (a', t_{a'}, R_{a'}) \}$

16 **Fin**

17 $trySetSingleAppointment(a, Sol)$

18 **Pour chaque demande de rendez-vous** $a' \in A_{\overline{Sol}}$ **faire**

19 $trySetSingleAppointment(a', Sol)$

20 **Fin**

21 **Retourne** Sol

22 **Fin**

Le mouvement de destruction ciblée est décrit dans l'algorithme 10. Dans un premier temps, un rendez-vous $a \in A_{\overline{Sol}}$ est choisi en fonction de son score de difficulté δ_a, en privilégiant les rendez-vous les plus difficiles à planifier. Un ensemble de créneaux T est ensuite sélectionné aléatoirement dans l'intervalle de faisabilité $[ES_a, LS_a]$. Le nombre de créneaux de cet ensemble dépend d'un paramètre β, qui correspond à un taux d'échantillonnage. Pour chaque créneau $t \in T$, les fonctions

$NAvailable()$ et $NAvailablePA()$ construisent respectivement les ensembles PI_a^t et RP_a^t décrits précédemment.

Si au moins l'un de ces ensembles n'est pas vide, toutes les combinaisons $\Upsilon_i = \{(b_{i1},t_{ib1},R_{ib1}),...,(b_{in},t_{ibn},R_{ibn})\}$, $n \in \{1,...,|Sol|\}$ de rendez-vous dont le retrait de la solution libérerait des ressources requises par a sont calculées par la fonction $minimumFreeResource()$. L'ensemble Υ est ainsi constitué. Le choix final de l'ensemble de rendez-vous à retirer se fait en fonction des formules suivante :

$$p_{retrait} = 1 - \frac{\sum_{\forall a \in \Upsilon_i} Essential_a + Emergency_a}{\sum_{\forall \Upsilon_i \in \Upsilon} \sum_{\forall a \in \Upsilon_i} Essential_a + Emergency_a} \tag{4.3}$$

$$p_{retard} = \frac{t - ES_a}{LS_a - ES_a} \times Emergency_a \tag{4.4}$$

$$p_i = \frac{p_{retrait} + p_{retard}}{2} \tag{4.5}$$

$p_{retrait}$ prend en compte les facteurs d'importance $Essential_b$ et de priorité $Emergency_b$ des rendez-vous $b \in \Upsilon_i$, afin de diminuer les conséquences de leur retrait sur la qualité de la solution Sol. p_{retard} prend en compte le créneau t si a est un rendez-vous prioritaire. Le calcul de la probabilité p_i est une moyenne de ces deux probabilités. Enfin, le rendez-vous a initialement choisi est planifié et nous tentons de réparer la solution en planifiant à nouveau le plus de rendez-vous $a \in A_{\overline{Sol}}$ possible.

Ce mouvement de destruction ciblée a une complexité temporelle égale à $O(max(|A|,|H|) \times |R| \times |T|)$, avec T l'ensemble des créneaux choisis dans H. La complexité dépend ici de la taille de l'échantillon de créneaux choisis pour tenter de planifier a.

4.2.5 Échange de date de début

Le mouvement d'échange de date de début consiste à échanger la date de début de deux rendez-vous choisis aléatoirement dans la solution. Cette opération, plus

communément appelée ***swap***, permet de réaliser de petits changements sur une solution pour explorer son voisinage proche.

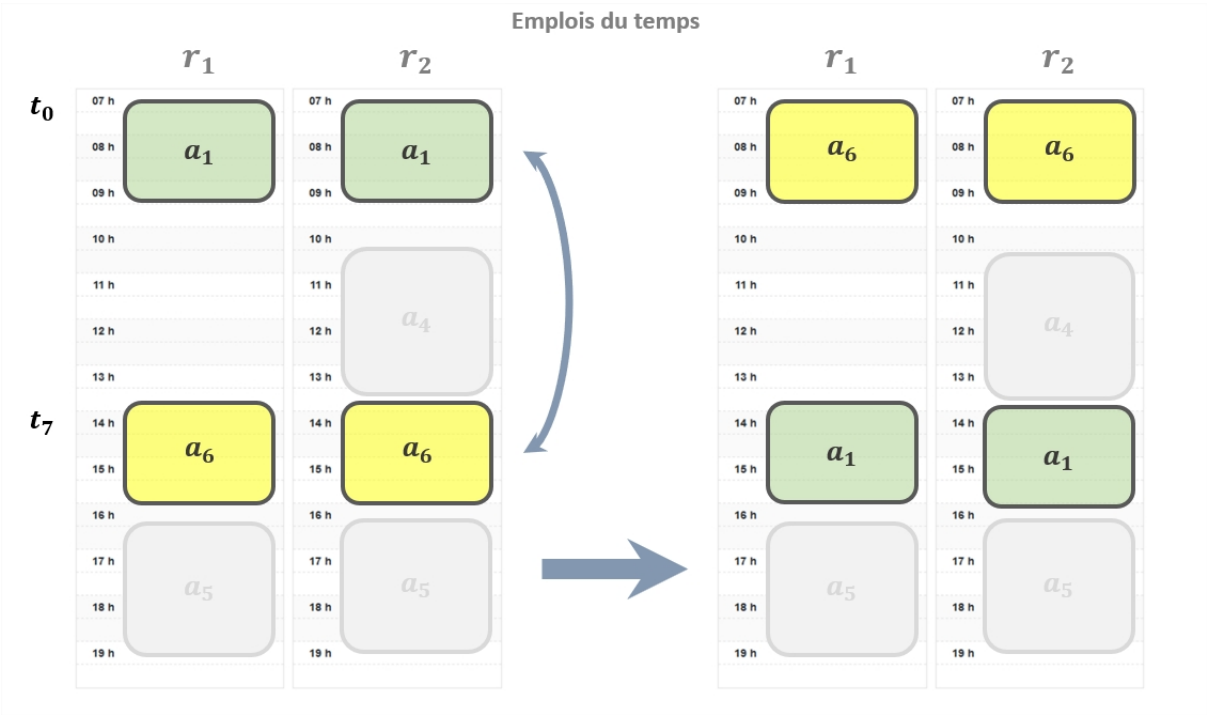


FIGURE 4.8: Exemple d'un échange de date de début entre deux rendez-vous, a_1 et a_6.

Ce mouvement est représenté dans la figure 4.8. Nous utilisons à nouveau l'exemple vu tout au long de ce chapitre. Pour effectuer cet échange, deux triplets sont sélectionnés dans la solution Sol : dans notre exemple, il s'agit de (a_1,t_{a_1},R_{a_1}) et (a_6,t_{a_6},R_{a_6}). Leur date de début, t_{a_1} et t_{a_6} sont échangés si la disponibilités des ressources affectées R_{a_1} et R_{a_6} le permet. Dans cet exemple, nous voyons que les ressources r_1 et r_2 affectées à ces rendez-vous sont disponibles. La ressource r_5, également affectée à a_1 mais non représentée dans cette illustration, permet elle aussi l'échange des rendez-vous. Dans les cas où l'échange n'est possible que dans un sens, le rendez-vous qui ne peut être planifié restera sans affectation et rejoindra $A_{\overline{Sol}}$. Si nous revenons à la figure 4.6, nous voyons que l'échange de a_4 et a_6 provoquerait le retrait de a_6, qui ne peut prendre la place de a_4.

La complexité de ce mouvement est en $O((duration_{a_1}+duration_{a_2})\times|R|)$, de par la recherche de ressources pour les deux rendez-vous à partir de leur date de début échangée.

4.2.6 Décalage à gauche

Le décalage à gauche consiste à déplacer les rendez-vous d'un ensemble de ressources de manière à ce que leur emploi du temps soit le plus compact possible. Au fil de la recherche et de l'application des différents mouvements sur la solution, des périodes de disponibilité trop courtes pour être utilisée apparaissent dans l'emploi du temps des ressources entre les divers rendez-vous auxquels elles participent. Ces temps d'attente, en plus de nuire à la planification, sont peu appréciés des planificateurs et du personnel en général.

Algorithme 11: leftShift()

Données: un entier k et une solution Sol

Résultat: une solution Sol

1 **Début**
2 $R' \leftarrow select(R,k)$
3 $\Gamma \leftarrow \Gamma \cup (a,t_a,R_a) \in Sol \mid R_a \cap R' \neq \emptyset$
4 **Pour chaque triplet** $(a,t_a,R_a) \in \Gamma$ **ordonnés selon** t_a **faire**
5 $t' \leftarrow ES_a$
6 **Tant que** $t' \leq LS_a$ **et** $R'_a = \emptyset$ **faire**
7 $R'_a \leftarrow availableResources(a,t')$
8 **Fin**
9 **Si** $R'_a \neq \emptyset$ **alors**
10 $Sol \leftarrow Sol \setminus (a,t_a,R_a)$; $Sol \leftarrow Sol \cup (a,t'_a,R'_a)$
11 **Fin**
12 **Sinon**
13 $t' \leftarrow t'+1$
14 **Fin**
15 **Fin**
16 **Retourne** Sol
17 **Fin**

Dans un premier temps, nous sélectionnons aléatoirement un sous-ensemble $R' \subset R$ de k ressources. Pour toutes les ressources $r \in R'$, nous récupérons tous les triplets (a,t_a,R_a) où la ressource r apparaît dans R_a. Ensuite, pour tous ces triplets, nous

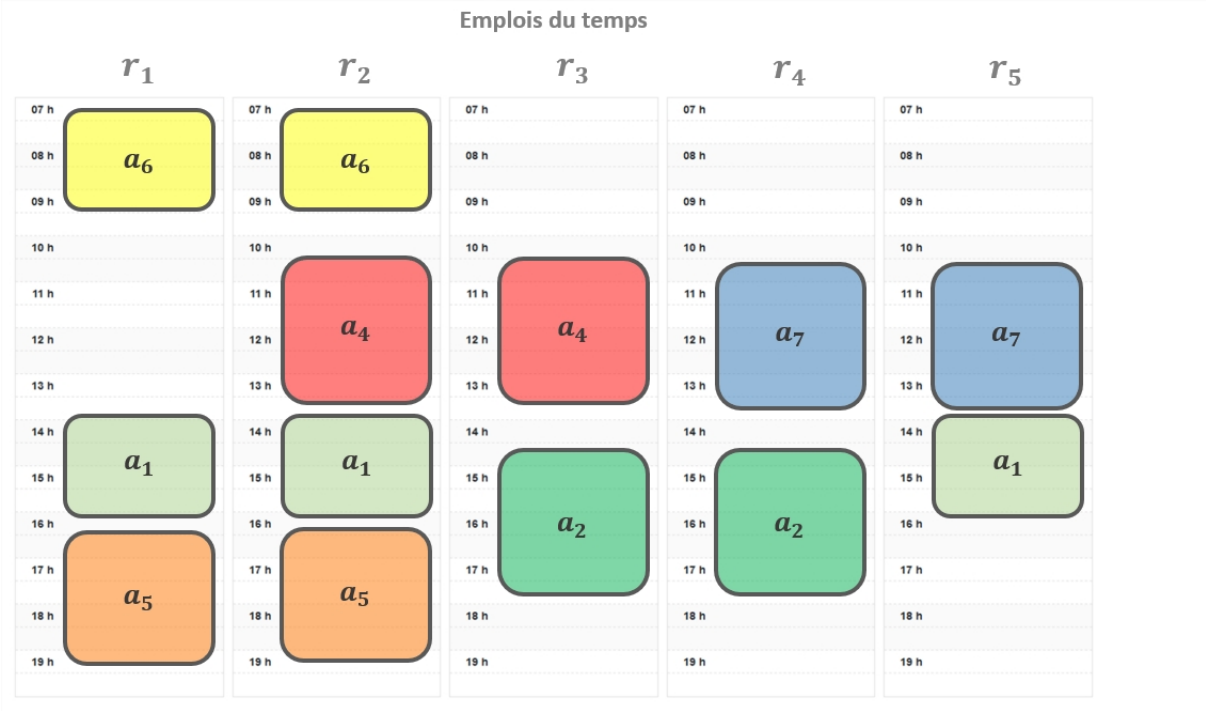
cherchons une nouvelle date de début t_a, la plus proche possible de ES_a et pour laquelle l'ensemble des ressources R_a est disponible. Le but avec ce mouvement n'est pas de changer les ressources affectées aux rendez-vous, par conséquent si aucune date t_a ne peut être trouvée plus proche de ES_a, aucune modification n'est faite.

La figure 4.9 représente les emplois du temps des ressources r_1, r_2, r_3, r_4 et r_5 avant et après l'application du décalage à gauche. Nous voyons des périodes de disponibilités inutilisables : pour la ressource r_2 par exemple, la période entre a_6 et a_4 ne pas contenir un autre rendez-vous. Idem pour la ressource r_5 et le temps entre a_7 et a_1. La complexité du décalage à gauche est en $O(|A_{Sol}|\times|H|\times|R|)$ dans le pire des cas. Néanmoins ces cas, pour lesquels la recherche de créneau concernerait l'ensemble des ressources et où les dates t_a serait toutes très proches de LS_a, se présentent rarement.

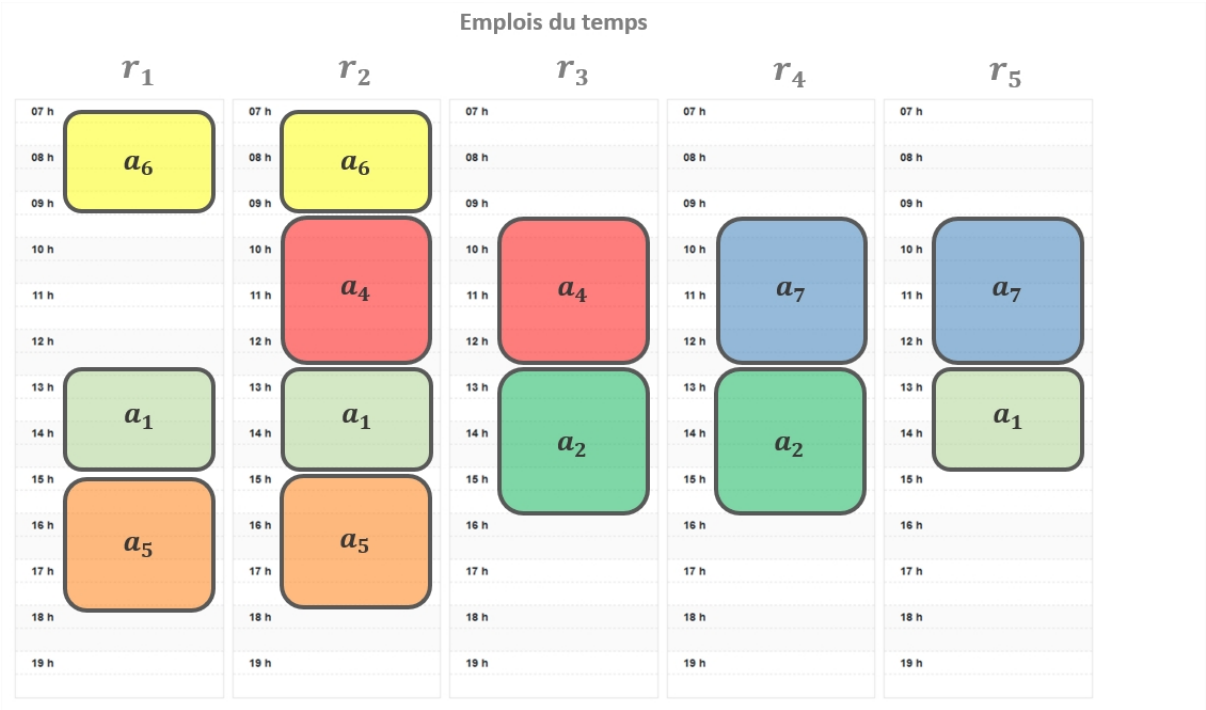
L'algorithme $LORH_ARC$ décrit précédemment nous permet d'obtenir des solutions initiales et les différents mouvements proposés ci-dessus permettent d'explorer l'espace de recherche en visitant le voisinage plus ou moins proche de nos solutions. Pour utiliser ces méthodes, nous avons choisi une approche capable d'adapter l'utilisation des différents mouvements au déroulement de la recherche : la recherche à voisinage large adaptative, plus communément appelée **Adaptive Large Neighborhood Search**.

4.3 Adaptive Large Neighborhood Search

L'Adaptive Large Neighborhood Search (ALNS) est une approche basée sur la recherche à voisinage large (Large Neighborhood Search) définie dans l'article de Shaw (1998). L'ALNS a quant a elle été introduite dans l'article de Ropke and Pisinger (2006) et appliquée sur de nombreux problèmes, y compris des problèmes de planification similaires au nôtre (Muller, 2011, He et al., 2018). L'ALNS applique à chacune de ses itérations un mouvement sélectionné parmi l'ensemble des mouvements disponibles sur la solution courante. Ces mouvements ont pour objectif d'explorer le voisinage de la solution de manière différente afin d'en trouver une meilleure et consistent souvent à détruire une partie de la solution avant de la reconstruire à l'aide d'heuristiques de destruction et de construction. Pour notre problème de



(a) Emplois du temps des ressources r_1, r_2, r_3, r_4 et r_5 avant le décalage à gauche.



(b) Emplois du temps des ressources r_1, r_2, r_3, r_4 et r_5 après le décalage à gauche.

FIGURE 4.9: État de la solution *Sol* avant et après le décalage à gauche.

planification, il s'agit de retirer certains rendez-vous de la solution courante avant de tenter de les planifier à nouveau.

4.3.1 Principe général

L'ALNS diffère principalement de la LNS par la couche adaptative permettant de faire varier les **poids** associés aux mouvements. Le principe général de l'algorithme est donné en figure 4.10. Les étapes affichées en bleu constituent l'algorithme de la LNS tandis que les étapes en rouge sont ajoutées par l'ALNS. Les poids influencent la probabilité de choisir un mouvement m dans l'ensemble des mouvements M à chaque itération de l'algorithme. Ils évoluent au fil de l'exécution en fonction des performances de chaque mouvement. Dans son article, Ropke suggère de diviser la recherche en blocs d'itérations successives, appelés **segments**. À chaque itération d'un segment s, un mouvement m est choisi par une sélection par roulette en fonction de son poids w_m^s pour être ensuite appliqué à la solution courante Sol. Dans la LNS, cette sélection se fait de manière aléatoire. Pour toute la durée du segment, un score π_m est associé à chaque mouvement m. Ce score représente la récompense totale du mouvement durant un segment s, relativement aux résultats obtenus suite à ses applications sur la solution. À la fin de chaque segment s, les poids w_m^s sont mis à jour en fonction des scores π_m obtenus par les mouvements durant le segment.

L'ensemble M est constitué de tous les mouvements décrits dans les sections précédentes, eux-mêmes composés d'une heuristique de destruction et de construction. Nous ajoutons à cela un mouvement de remise à zéro de la solution qui retire tous les rendez-vous $a \in A_{Sol}$ et relance la méthode de construction aléatoire $algorithmConstruction()$ décrite en début de chapitre. Cette remise à zéro est déclenchée après un nombre ι_{max} d'itérations sans amélioration de la meilleure solution.

L'algorithme général de notre ALNS, basé sur la définition originale de Ropke and Pisinger (2006), est donné dans l'algorithme 12. La solution initiale est obtenue avec la méthode de construction décrite dans l'algorithme 2. À tous les mouvements m de l'ensemble M sont attribués des poids w_m^0, identiques au lancement de l'algorithme et ayant pour valeur $1 \ / \ |M|$. L'ensemble M est constitué des mouvements décrits précédemment, qui permettront d'explorer les solutions voisines de la solution courante pour l'améliorer. Si la recherche stagne pendant trop longtemps, nous générons une nouvelle solution, et la recherche reprend à partir de cette solution. Ce processus se répète jusqu'à ce que les conditions d'arrêt soient atteintes, que ce

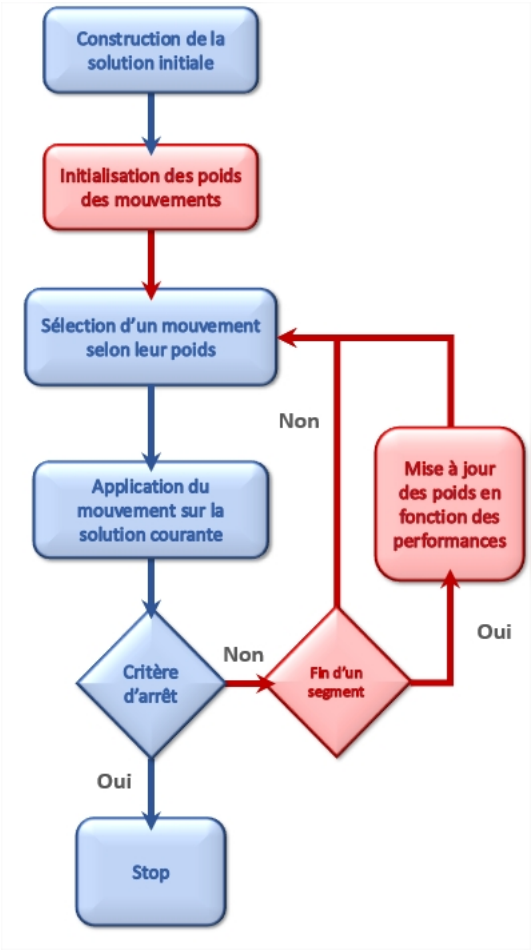


FIGURE 4.10: Étapes de l'ALNS et différences avec la LNS.

soit après un certain nombre d'itération, une durée ou une qualité suffisante de la solution. Nous nommerons cette méthode *LORH_ALNS* par la suite.

4.3.2 Formule d'ajustement des poids

Nous l'évoquions plus tôt, la recherche est divisée en segments. Le score π_m^0 de tous les mouvements m est initialisé à zéro au début de chaque segment. À chaque itération au sein d'un segment s, le score π_m^s augmente en fonction d'une récompense σ selon les conditions suivantes :

$$\sigma = \begin{cases} \sigma_1, & \text{si la solution produite est meilleure que celles obtenues jusqu'ici} \\ \sigma_2, & \text{si la solution produite est meilleure que la solution courante} \\ \sigma_3, & \text{si la solution produite est moins bonne que la solution courante} \end{cases} \tag{4.6}$$

Algorithme 12: ALNS()

Données: un ensemble de mouvement M et un entier s_{it}

Résultat: une solution Sol

```
1  Début
2      Sol ← algorithmConstruction()
3      Sol_best ← Sol ; ι ← 0
4      ∀ m ∈ M, w_m^0 ← 1 / |M|
5      Tant que les critères d'arrêt ne sont pas atteints faire
6          Pour chaque segment s de s_it itérations faire
7              m ← rouletteWheel(M,w_m^s)
8              Sol ← m(Sol)
9              π_m est mis à jour en fonction de f(Sol)
10             Si f(Sol) < f(Sol_best) alors
11                 Sol_best ← Sol ; ι ← 0
12             Sinon
13                 ι++
14             Fin
15             Si ι ≥ ι_max alors
16                 Sol ← restart(Sol)
17             Fin
18         Fin
19         mise à jour des poids w_m^{s+1} pour chaque mouvement m
20     Fin
21     Retourne Sol_best
22 Fin
```

À la fin de chaque segment s, le poids w_{s+1}^m du mouvement m pour le prochain segment s+1 est calculé selon l'équation 4.7 :

$$w_m^{s+1} = (1-r)\ w_m^s + r\ (\frac{\pi_m^s}{\theta_m^s}) \tag{4.7}$$

où θ_m^s est le nombre d'utilisations du mouvement m durant un segment s. Le facteur de réaction r contrôle la force du changement du poids d'un segment s au

segment suivant $s+1$. Si $r = 0$, il n'y a aucun changement dans les poids, et si $r = 1$, les poids pour le segment $s+1$ dépendent seulement des performances du mouvement durant le segment s. Le nombre d'itérations d'un segment, les différents scores σ_1, σ_2 et σ_3 et le facteur de réaction r sont des paramètres de l'ALNS que nous chercherons à régler dans la section suivante.

4.4 Expérimentations & Résultats

Dans cette section nous souhaitons évaluer les performances de l'ALNS sur nos instances et sur les instances PSPLIB. Pour cela nous comparerons dans un premier temps les résultats obtenus avec notre algorithme *LORH_ALNS* aux solutions optimales obtenues avec CPLEX quand cela fut possible et présentés dans le chapitre 3. Ensuite, nous comparerons l'ALNS à sa version sans couche adaptative, pour étudier l'intérêt de ce mécanisme d'adaptation au vu des calculs supplémentaires qu'il entraîne et des paramètres supplémentaires qui l'accompagnent. Enfin, nous testerons *LORH_ALNS* sur les instances PSPLIB et comparerons les résultats à ceux obtenus dans la littérature. En premier lieu, nous nous intéressons au paramétrage de l'ALNS.

4.4.1 Paramétrage

Le paramétrage des algorithmes d'optimisation est souvent un problème supplémentaire complexe qui s'ajoute à la problématique initiale que ces algorithmes tentent de résoudre. Leurs performances peuvent changer drastiquement d'une instance à l'autre selon le paramétrage, et le nombre de paramètres à ajuster peut rapidement atteindre plusieurs dizaines pour certaines métaheuristiques. Si nous prenons l'exemple d'un algorithme génétique, ses performances varient fortement selon le taux de mutation choisi : un taux trop bas ne permet pas l'apport de diversité nécessaire à son bon fonctionnement tandis qu'un taux trop élevé le transforme en recherche aléatoire. Pendant des années, les chercheurs ont mené des tests « à la main » pour déterminer le meilleur ensemble de paramètres pour leurs algorithmes. Ces méthodes manuelles sont cependant chronophages (devenant parfois plus longues à appliquer que le développement de l'algorithme à l'origine) et peuvent être biaisés

par l'expérience et l'intuition du chercheur qui dirige les tests. Pour ces raisons, des méthodes automatiques ont été développées pour cette recherche des meilleurs paramètres :

— des méthodes basées sur des « courses » entre les configurations possibles pour les algorithmes (Birattari et al., 2002) ;

— des métaheuristiques utilisées pour paramétrer d'autres méthodes (Ansótegui et al., 2009, Nannen and Eiben, 2006) ;

— des approches de modélisation statistiques (Hutter et al., 2010).

Pour les algorithmes que nous avons présentés, nous identifions plusieurs paramètres conditionnant leur efficacité. Les mouvements basés sur la destruction d'un ensemble d'affectations nécessitent un nombre de rendez-vous à détruire. Nous avons k_{max} un taux maximum de rendez-vous à détruire par rapport au nombre total de rendez-vous de l'instance $|A|$. Le nombre k de rendez-vous à détruire est ensuite déterminé aléatoirement pour chaque application du mouvement entre 1 et k_{max}. Le temps de calcul du mouvement de destruction ciblée dépend fortement du taux d'échantillonnage β de créneaux à prendre en compte dans sa recherche. Pour l'ALNS lui-même, plusieurs paramètres sont à considérer : le nombre d'itérations d'un segment s_{it}, les récompenses σ_1, σ_2 et σ_3 et le facteur de réaction r.

Pour déterminer les meilleures valeurs à attribuer à ces paramètres, nous nous sommes appuyés d'une part sur les paramètres utilisés dans la littérature concernant l'ALNS, et d'autre part sur une méthode automatique d'optimisation des paramètres, reconnue dans la littérature et utilisées sur de nombreuses approches. Le logiciel « iRace » (López-Ibáñez et al., 2016) repose sur un principe de course itérative entre différentes configurations de l'algorithme pour sélectionner au fil des étapes les configurations les plus performantes. Pour fonctionner, iRace requiert l'ensemble des paramètres à optimiser et leur domaine de définition ainsi qu'un ensemble d'instances I sur lesquelles seront confrontées les différentes configurations. Initialement, des configurations sont générées aléatoirement et mises en compétition sur I. Au fil des étapes, les moins performantes sont éliminées et d'autres configurations dérivées des meilleures sont ajoutées. Ce cycle se répète jusqu'à obtenir un nombre minimum de configurations ou après avoir testé sur un nombre prédéfini d'instances.

L'ensemble d'instances d'entraînement a été généré à partir des scénarios décrits dans le chapitre 3 en faisant varier les différents paramètres (taille, importances, urgences, etc.). Cet ensemble d'entraînement ne sera utilisé que pour nos tests sur le paramétrage. Pour définir le domaine de définition de certains paramètres, nous nous sommes appuyés sur les valeurs utilisées dans la littérature, notamment sur les configurations étudiées dans l'article de Turkeš et al. (2021). Chaque paramètre a été individuellement soumis au processus de course d'iRace, avec chaque autre paramètre fixé pour obtenir les meilleures valeurs sur chacun des paramètres. Nous avons ensuite relancé les courses en incluant de plus en plus de paramètres pour vérifier la cohérence des résultats individuels lorsqu'ils sont combinés les uns avec les autres.

TABLE 4.2: Domaines de définition des paramètres et valeurs retenues par iRace.

Paramètre	**Domaine de définition** D	**Valeur retenue**
σ_1	$[0,100] \in \mathbb{N}$	73
σ_2	$[0,100] \in \mathbb{N}$	21
σ_3	$[0,100] \in \mathbb{N}$	1
s_{it}	$[0,1000] \in \mathbb{N}$	103
k_{max}	$[0,1] \in \mathbb{R}$	15
r	$[0,1] \in \mathbb{R}$	0.08
β	$[0,1] \in \mathbb{R}$	0.4

Le tableau 4.2 présente les différents paramètres, leur domaine de définition et les valeurs obtenues avec iRace. Pour rappel, les différents σ représentent les récompenses accordées aux mouvements selon les trois cas de figure présentés précédemment. s_{it} est la taille d'un segment s en nombre d'itérations. k_{max} est un taux maximum de rendez-vous à détruire par rapport au nombre total de rendez-vous de l'instance $|A|$. Le nombre k de rendez-vous à détruire est ensuite déterminé aléatoirement pour chaque mouvement entre 1 et $k_{max} \times |A|$. r est le facteur de réaction contrôlant la force des changements de probabilité d'un segment à l'autre. Enfin, β est le taux d'échantillonnage des créneaux de l'algorithme 10. Ces valeurs obtenues avec les différentes exécutions d'iRace (nous avons retenu la moyenne des meilleurs

configurations) sont cohérentes avec les paramètres utilisés dans la littérature, et nous ont fournis les meilleurs résultats. Nous utiliserons ce paramétrage pour les instances issues de nos scénarios et pour les instances PSPLIB.

4.4.1.1 Apport de la couche adaptative de l'ALNS

L'apport de l'ALNS, par rapport à la Large Neighborhood Search, est l'adaptation des poids des différents mouvements en fonction de leurs performances. De cette manière, des mouvements efficaces sur certaines instances seront privilégiés et utilisés plus souvent. Les poids s'adaptent également à l'évolution de la recherche, où des mouvements peuvent devenir plus pertinents à mesure que la recherche progresse. Dans leur article, Turkeš et al. (2021) réalisent une étude comparative sur de nombreux ALNS présentés dans la littérature afin de quantifier l'apport réel de la couche adaptative. Nous souhaitons faire de même sur notre algorithme, afin de démontrer que la couche adaptative apporte un gain de performance suffisant pour justifier le surcoût de calculs qu'elle entraîne, et sa plus grande complexité par rapport à la LNS.

Pour cela, nous simulerons la LNS avec l'ALNS que nous avons décrit plus tôt. L'ALNS adapte les poids des mouvements en fonction d'un paramètre, le facteur de réaction r. Plus ce facteur est grand, plus l'adaptation sera forte à chaque segment $s+1$ et plus les poids dépendront de leurs performance au segment précédent s. À l'inverse, si $r = 0$, les poids des mouvements ne changent jamais, quelles que soient leurs performances passées. Notre version de l'ALNS sans adaptation, notée $(\neg A)$LNS, reprendra tous les paramètres optimaux que nous avons défini précédemment à l'exception du paramètre r, qui sera égal à 0. Tous les calculs en rapport avec l'adaptation sont aussi retirés de l'algorithme. Chaque mouvement aura une probabilité égale d'être choisi pendant l'intégralité du déroulement de l'algorithme.

4.4.2 Instances de l'entreprise

Pour nos expérimentations avec l'ALNS, nous avons utilisé le même ensemble d'instances que dans le chapitre précédent. Pour rappel, nous avons un ensemble de 80 instances de quatre familles (ou scénarios) qui représentent des problèmes de planification de quatre établissements de santé différents. Nous utilisons toujours la

formule 3.1 pour évaluer la qualité de la solution obtenue. Nous limitons le temps de calcul à 10 minutes.

Sur ces instances, nous souhaitons dans un premier temps comparer les résultats de nos méthodes aux solutions optimales obtenues par CPLEX. Les résultats de CPLEX, obtenus avec une limite de temps de calcul de 12 heures ont été présentés dans la sous-section 3.3.2.2 et seront reportés dans les tableaux suivants. Nous voulons ensuite étudier l'apport de la couche adaptative de l'ALNS, pour justifier d'une part les efforts pour l'implémenter et optimiser ses paramètres et d'autre part le temps de calcul supplémentaire qu'il entraîne. L'ALNS et sa version sans couche adaptative ont été implémentés en C#, et les expérimentations ont été menées sur une machine dotée d'un processeur Intel i7-7700HQ.

Nous indiquons dans chaque tableau et pour chaque instance son nombre de demandes de rendez-vous **T**, le nombre de rendez-vous importants **I**, le nombre de rendez-vous prioritaires **U**, le nombre de relations de précédence **P** et le nombre de demi-journées de disponibilité retirées aux patients **D**. Nous indiquons ensuite les bornes inférieures et supérieures **LB** et **UB** obtenues par CPLEX. Nous indiquons les meilleurs résultats $f(Sol_{best})$ obtenus par l'algorithme de construction $LORH_ARC$, la (¬A)LNS et $LORH_ALNS$. Pour la (¬A)LNS et l'ALNS, nous donnons les écarts avec l'algorithme de construction ($Gap_f = ((\neg A)LNS - LORH_ARC/LORH_ARC)$ pour le premier et l'écart avec la (¬A)LNS ($Gap_f = (LORH_ALNS - (\neg A)LNS/(\neg A)LNS)$ pour le second.

Les résultats sur le scénario ***SurgDep*** sont reportés dans le tableau 4.3. Sur cette première famille d'instances, nous avions réussi à obtenir l'optimalité avec CPLEX sur l'ensemble des instances de plus petites taille (de 16 rendez-vous, **T**=16). Les résultats obtenus par $LORH_ARC$ ne sont jamais optimaux sur ces petites instances, et présentent des écarts importants avec la solution optimale. La (¬A)LNS parvient à améliorer ces solutions et obtient l'optimalité sur les trois premières instances (avec **I**=0, **I**=4 et **I**=8). Sur celles comportant des rendez-vous prioritaires (**U**≠0), la (¬A)LNS améliore les résultats de l'algorithme de construction (entre 47.1 et 67.6% d'amélioration) mais n'obtient pas les solutions optimales. $LORH_ALNS$ améliore encore ces résultats et trouve les solutions optimales sur deux instances supplémentaires.

TABLE 4.3: Résultats sur les instances de la famille ***SurgDep***

SurgDep					**CPLEX**		***LORH_ARC***	***(¬A)LNS***		***LORH_ALNS***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ARC***	$f(Sol_{best})$	Gap_f ***(¬A)LNS***
16	0	0	0	0	0	0	2	**0**	-100%	**0**	0%
16	4	0	0	0	0	0	1	**0**	-100%	**0**	0%
16	8	0	0	0	0	0	1	**0**	-100%	**0**	0%
16	4	4	0	0	1.48	1.48	6.04	2.6	-56.9%	**1.48**	-43.2%
16	8	8	0	0	9.54	9.54	42.50	13.77	-67.6%	**9.54**	-30.72%
16	0	0	0	4	0	0	3	**2**	0%	**2**	0%
16	8	4	0	4	19.02	19.02	47.42	25.08	-47.1%	**20.6**	-17.88%
48	0	0	0	0	0	9	3	**0**	-100%	**0**	0%
48	12	0	0	0	0	7	3	1	-66.67%	**0**	-100%
48	24	0	0	0	0	8	12	1	-91.67%	**0**	-100%
48	12	12	0	0	57.88	68.16	232.10	97.36	-58.05%	**60.77**	-37.58%
48	24	24	0	0	85.65	137.60	531.14	154.97	-70.82%	**107.63**	-30.55%
48	0	0	0	4	0	6	4	**0**	-100%	**0**	0%
48	0	0	0	12	0	5	4	3	-25%	**2**	-33.33%
48	24	12	0	12	114.77	116.27	304.29	207.1	-31\.94%	**185.39**	-10.48%

Sur les instances à 48 rendez-vous, nous n'avions pas obtenu de solutions optimales avec CPLEX. Pour les instances ne comportant pas de rendez-vous urgents (**U**≠0), les bornes inférieures **UB** correspondent bien aux solutions optimales sur ces instances. Là encore, l'algorithme de construction n'obtient aucune des solutions optimales. Avec la (¬A)LNS, nous trouvons la solution optimale sur deux instances et améliorons nettement les solutions de l'algorithme de construction. *LORH_ALNS* nous permet d'obtenir deux solutions optimales supplémentaires, et égale et surpasse la (¬A)LNS sur toutes ces instances. En dehors des améliorations de 100%, nous constatons des écarts allant de 10.48 à 37.58%.

Les résultats pour le scénario ***RehabCenter*** sont reportés dans le tableau 4.4. Sur les instances de 96 rendez-vous, nous avions obtenu avec CPLEX les solutions optimales sur toutes les instances sans rendez-vous prioritaires. La (¬A)LNS améliore assez peu les résultats de *LORH_ARC*, à l'exception d'une instance avec précédences (**P**=15), et nous notons des écarts entre 1.93 et 6.18%. L'ALNS n'apporte aucune amélioration sur la majorité des instances de cette famille.

Sur les instances de taille **T**=288, la (¬A)LNS obtient des solutions optimales sur les deux premières instances, et améliore assez largement (de 5.06 à 100%) les

TABLE 4.4: Résultats sur les instances de la famille ***RehabCenter***.

RehabCenter					**CPLEX**		***LORH_ARC***	***(¬A)LNS***		***LORH_ALNS***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ARC***	$f(Sol_{best})$	Gap_f ***(¬A)LNS***
96	0	0	0	0	0	0	**1**	**1**	0%	**1**	0%
96	24	0	0	0	0	0	**1**	**1**	0%	**1**	0%
96	48	0	0	0	0	0	**1**	**1**	0%	**1**	0%
96	24	24	0	0	265.95	317.63	317	**315**	-0.63%	**315**	0%
96	48	48	0	0	557.81	603.25	602	**600**	-0.33%	**600**	0%
96	0	0	15	0	0	0	3	**0**	-100%	**0**	0%
96	48	24	15	0	308.57	315	335.75	**315**	-6.18%	**315**	0%
96	96	48	15	0	532.88	1426.5	616.88	605	-1.93%	**600**	-0.83%
288	0	0	0	0	0	196	4	**0**	-100%	**0**	0%
288	72	0	0	0	0	1914	5	**0**	-100%	**0**	0%
288	144	0	0	0	0	4960	5	**1**	-100%	**1**	0%
288	72	72	0	0	689.06	2609.75	**1125**	**1125**	0%	**1125**	0%
288	144	144	0	0	1958.35	6064.5	2948.5	**2790**	-5.38%	**2790**	0%
288	0	0	25	0	0	196	5	3	-40%	**0**	-100%
288	144	72	24	0	689.06	5729.88	1254	1173.19	-6.44%	**1125**	-4.11%
288	288	144	25	0	1958.35	8740.5	2991.25	**2840**	-5.06%	**2840**	0%

solutions initiales. À nouveau sur cette famille, l'ALNS n'apporte que peu d'amélioration en comparaison de sa version sans couche adaptative. Une solution optimale est obtenue, mais nous constatons en dehors de cela qu'une seule amélioration assez faible de 4.11%.

Dans le tableau 4.5 nous présentons les résultats obtenus sur la famille d'instance ***Admission***. Pour les instances de taille **T**=136, l'algorithme de construction ne trouve aucune solution optimale, et le nombre de rendez-vous qui ne sont pas planifiés (visible sur les instances sans rendez-vous urgents) est important. La (¬A)LNS améliore assez nettement ces résultats sur l'ensemble des instances, avec un écart moyen à 46.07%. Les solutions obtenues avec *LORH_ALNS* sont systématiquement meilleures que les deux autres méthodes sur l'ensemble des instances de ce scénario. Les gains sont conséquents et atteignent 40.48%, avec un écart moyen à 24.45%.

Sur les instances de plus grande taille (**T**=408), nous n'avions obtenu aucune solution optimale avec CPLEX. L'algorithme de construction semble aussi avoir des difficultés à traiter ces grandes instances, avec un nombre élevé de rendez-vous qui ne sont pas posés. À l'instar des instances de 136 rendez-vous, la (¬A)LNS parvient

TABLE 4.5: Résultats sur les instances de la famille ***Admission***.

Admission					**CPLEX**		***LORH_ARC***	***(¬A)LNS***		***LORH_ALNS***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ARC***	$f(Sol_{best})$	Gap_f ***(¬A)LNS***
136	0	0	0	0	0	4	12	5	-58.33%	**4**	-20%
136	34	0	0	0	0	4	36	6	-83.33%	**5**	-16.67%
136	68	0	0	0	0	4	91	9	-15.69%	**5**	-44.44%
136	34	34	0	0	126.32	204.85	360.16	260.86	-27.57%	**216.17**	-17.13%
136	68	68	0	0	149.84	298.76	512.91	426.16	-16.91%	**363.82**	-14.63%
136	0	0	48	0	0	4	17	9	-47.06%	**6**	-33.33%
136	0	0	0	35	0	4	10	5	-50%	**4**	-20%
136	68	34	48	35	127.08	202.63	441.3	399.55	-9.46%	**237.8**	-40.48%
136	136	68	48	35	154.66	306.78	713.66	601.68	-15.69%	**548.18**	-8.89%
408	0	0	0	0	0	24	14	13	-7.14%	**12**	-7.69%
408	102	0	0	0	0	31	104	15	-85.58%	**12**	-20%
408	204	0	0	0	0	65	122	43	-64.75%	**12**	-72.09%
408	102	102	0	0	165.64	2841.45	994.88	899.94	-9.54%	**722.06**	-19.77%
408	102	204	0	0	345.11	5333.52	2240.42	2052.43	-8.39%	**1919.11**	-6.5%
408	0	0	64	0	0	24	24	15	-37.5%	**13**	-13.33%
408	0	0	85	0	0	24	26	17	-34.62%	**13**	-23.53%
408	0	0	0	91	0	24	14	14	0%	**13**	-14.29%
408	204	102	64	91	165.64	3136.23	1380.07	1345.01	-2.54%	**1253.2**	-6.83%
408	408	204	85	91	345.82	6904.07	3436.33	3141.23	-8.59%	**2885.59**	-8.14%

à trouver de meilleures solutions, avec un écart moyen de 24.16%. L'ALNS améliore encore ces résultats sur toutes les instances.

Les résultats sur le scénario ***CardioRehab*** sont présentés dans le tableau 4.6. Dans le chapitre précédent, nous constations que de nombreuses solutions optimales étaient obtenues par CPLEX sur les instances de 160 rendez-vous (**T**=160) de ce scénario. L'algorithme de construction *LORH_ARC* parvient également à trouver la solution optimale sur les deux premières instances, et en est proche sur la troisième. Sur les instances plus difficiles (avec des précédences et des disponibilités retirées), quelques rendez-vous ne sont pas planifiés. La (¬A)LNS améliore encore largement ces solutions initiales sur toutes les instances de cette taille, et parvient à trouver les solutions optimales sur quatre instances supplémentaires. Si l'ALNS ne parvient pas à trouver l'optimalité sur l'instance ayant 38 demi-journées de disponibilité retirées aux patients (**D**=38), il trouve de meilleures solutions sur toutes les autres instances.

Sur les instances de 480 rendez-vous, les constatations sont similaires. Les solutions initiales de l'algorithme de construction sont améliorées par la (¬A)LNS (25 à

TABLE 4.6: Résultats sur les instances de la famille ***CardioRehab***.

CardioRehab					**CPLEX**		***LORH_ARC***	***(¬A)LNS***		***LORH_ALNS***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ARC***	$f(Sol_{best})$	Gap_f ***(¬A)LNS***
160	0	0	0	0	0	0	**0**	**0**	0%	**0**	0%
160	40	0	0	0	0	0	**0**	**0**	0%	**0**	0%
160	80	0	0	0	0	0	1	1	-100%	**0**	-100%
160	40	40	0	0	12.537	32.321	79.16	59.07	-25.38%	**19.22**	-67.46%
160	80	80	0	0	57.927	141.624	395.28	307.3	-22.26%	**128.91**	-58.05%
160	0	0	27	0	0	0	3	**0**	-100%	**0**	0%
160	0	0	36	0	0	0	6	**0**	-100%	**0**	0%
160	0	0	0	19	0	0	4	**0**	-100%	**0**	0%
160	0	0	0	38	0	0	5	**0**	-100%	1	0%
160	80	40	27	19	12.924	43.566	131.98	63.74	-51.71	**50.39**	-20.93%
160	160	80	36	38	58.797	185.415	388.44	361.93	-6.82%	**310.8**	-14.13%
480	0	0	0	0	0	154	1	**0**	-100%	**0**	0%
480	120	0	0	0	0	498	1	**0**	-100%	**0**	0%
480	240	0	0	0	0	656	1	**0**	-100%	**0**	0%
480	120	120	0	0	168.726	3622.926	2680.14	748.3	-72.08%	**635.98**	-15.01%
480	240	240	0	0	223.125	11145.423	3545.31	1667.83	-52.96%	**1472.84**	-11.69%
480	0	0	38	0	0	154	7	1	-85.71%	**0**	-100%
480	0	0	48	0	0	154	4	3	-25%	**0**	-100%
480	0	0	0	60	0	163	7	2	-71.43%	**0**	-100%
480	0	0	0	122	0	122	11	3	-72.73%	**1**	-66.67%
480	240	120	38	60	182.738	6471.851	2668.62	969.59	-63.67%	**840.11**	-13.35%
480	480	240	48	122	235.408	21690.747	5030.68	3584.6	-28.75%	**3215.39**	-10.3%

100% d'amélioration), avec notamment trois instances résolues de manière optimale. L'ALNS trouve la solution optimale sur trois instances supplémentaires, et surpasse aussi la (¬A)LNS sur les instances composées de rendez-vous prioritaires.

TABLE 4.7: Résumés des écarts moyens entre les différentes approches, et taux de rendez-vous planifiés pour les recherches locales.

Scénarios	$\|A\|$	Gap_f moyens		Average $\|A_{Sol_{best}}\|/\|A\|$	
		LORH_ARC / (¬A)LNS	(¬A)LNS / LORH_ALNS	(¬A)LNS	LORH_ALNS
SurgDep	16	-71.5%	-9.92%	95.83%	98.61%
	48	-67.83%	-36.36%	91.44%	98.38%
RehabCenter	96	-23.23%	-0.09%	99.54%	99.65%
	288	-44.84%	-22.68%	99%	98.03%
Admission	136	-46.07%	-24.45%	92.25%	95.59%
	408	-24.16%	-18.17%	95.52%	96.77%
CardioRehab	160	-55.11%	-5.51%	98.35%	99.83%
	480	-67.23%	-41.7%	96.33%	99.69%

Les résultats résumés de ces expérimentations sont reportés dans le tableau 4.7. Nous donnons pour chaque scénario et pour chaque taille **T** le Gap_f moyen entre les résultats de la $(\neg A)$LNS et ceux de $LORH_ARC$, puis entre la $(\neg A)$LNS et $LORH_ALNS$. Nous reportons aussi le pourcentage de rendez-vous planifiés pour les deux recherches locales. Comme nous l'avions constaté sur les tableaux précédents, la $(\neg A)$LNS améliore nettement les solutions obtenues avec l'algorithme de construction initiale, avec des écarts importants allant jusqu'à 71.5%, pour un gain moyen sur l'intégralité des scénarios de 50%.

Avec ces expérimentations, nous voulions aussi étudier l'apport réel de la couche adaptative de l'ALNS, comparée à la $(\neg A)$LNS dans laquelle les probabilités associées aux mouvements ne changent pas. Nous obtenons de meilleurs résultats avec l'ALNS que sa version sans adaptation sur tous les scénarios, avec des gains souvent significatifs, particulièrement sur les grandes instances. Les gains moyens sur les quatre scénarios sont de 19.86% et sont suffisants pour justifier l'utilisation de la couche adaptative au regard des résultats obtenus dans l'article de Turkeš et al. (2021).

4.4.3 Instances PSPLIB

Dans cette section nous présentons les résultats obtenus par $LORH_ALNS$ sur les instances de l'ensemble PSPLIB, présentées dans le chapitre précédent. Pour rappel, il est composé de quatre groupes d'instances de tailles différentes : J30, J60,

J90 et J120, composés de plusieurs centaines d'instances. Les solutions sont évaluées selon la formule 3.15 visant à réduire la durée totale du projet. Nous avons sélectionné aléatoirement un sous-ensemble de 80 instances dans chacune de ces familles pour nos tests. Les résultats obtenus par $LORH_ALNS$ seront ensuite comparés soit aux solutions optimales obtenues dans la littérature, soit aux meilleures solutions trouvées jusqu'à maintenant (Kolisch and Sprecher, 2020). Nous utiliserons les paramètres déterminés par iRace, entraîné sur les instances de l'entreprise. Les tests sont réalisés avec une machine dotée d'un processeur Intel I7-7700HQ. Les résultats sont reportés dans les tableaux ci-dessous.

Dans les tableaux suivants, nous donnons pour chaque instance de chaque famille l'identifiant **ID paramètres** de l'ensemble de paramètres utilisés pour générer l'instance, **ID** l'identifiant de l'instance, le meilleur ***makespan*** trouvé dans la littérature et reporté sur ces instances PSPLIB, le ***makespan*** trouvé par $LORH_ALNS$ et l'écart par rapport au résultat de la littérature ($Gap_{makespan} = LORH_ALNS - PSPLIB/PSPLIB$).

TABLE 4.8: Résultats de $LORH_ALNS$ sur la famille J30 des instances PSPLIB.

J30		***PSPLIB***	***LORH_ALNS***	
ID paramètres	**ID instance**	***Makespan***	***Makespan***	$Gap_{makespan}$
2	7	47	50	6.38%
3	6	54	60	11.11%
7	1	55	59	7.27%
9	9	63	75	19.05%
11	10	38	45	18.42%
13	4	72	89	23.61%
17	6	63	74	17.46%
30	2	68	81	19.12%
36	4	59	59	0%
41	6	103	110	6.8%
47	8	48	53	10.42%

Nous avons reporté les résultats obtenus sur quelques instances de l'ensemble J30 dans le tableau 4.8. Sur ces instances de 30 activités, nous constatons des écarts importants entre la meilleure solution et celles renvoyées par l'algorithme $LORH_ALNS$. Dans cet échantillon d'instance, nous trouvons la solution optimale

sur une seule instance, et les écarts varient de 6.8 à 23.61%. Sur l'ensemble des 80 instances de cette famille sur lesquelles nous avons mené nos tests, l'écart moyen entre les résultats de *LORH_ALNS* et les résultats de la littérature est de 10.59%.

TABLE 4.9: Résultats de *LORH_ALNS* sur la famille J60 des instances PSPLIB.

J60		***PSPLIB***	***LORH_ALNS***	
ID paramètres	**ID instance**	***Makespan***	***Makespan***	$Gap_{makespan}$
1	7	72	76	5.56%
1	8	75	87	16%
2	4	78	82	5.13%
5	6	74	94	27.03%
7	7	89	98	10.11%
11	10	58	58	0%
14	2	65	77	18.46%
17	4	71	80	12.68%
24	10	66	68	3.03%
32	9	74	76	2.7%
41	3	98	132	34.69%

Dans le tableau 4.9 nous présentons les résultats obtenus sur une dizaine d'instances de la famille J60 des instances PSPLIB. Nos constatations sont similaires sur cette famille, avec des écarts conséquents allant jusqu'à 34.69%. L'écart moyen entre le ***makespan*** obtenu avec *LORH_ALNS* reste du même ordre que pour la famille précédente : 11.16%.

Les résultats sur l'ensemble J90 sont donnés dans le tableau 4.10. Sur ces instances de plus grande taille, les écarts constatés sont encore plus importants. Trois solutions optimales sont trouvées. Sur tout l'ensemble, nous trouvons la solution optimale sur 24 instances. Sur les autres instances de ce tableau, les écarts commencent à 1.39% et montent jusqu'à 32.91%.

Nous reportons les résultats de *LORH_ALNS* sur les instances de l'ensemble J120 dans le tableau 4.11. Encore une fois, nous observons des écarts importants entre les résultats de notre algorithme et ceux de la littérature. Sur ces quelques instances, nous ne trouvons aucune solution optimale et sur l'ensemble des 80 instances, nous n'obtenons que deux solutions optimales. Les écarts entre la solution

TABLE 4.10: Résultats de *LORH_ALNS* sur la famille J90 des instances PSPLIB.

J90		***PSPLIB***	***LORH_ALNS***	
ID paramètres	**ID instance**	***Makespan***	***Makespan***	$Gap_{makespan}$
1	9	72	90	25%
4	9	79	79	0%
6	10	94	110	17.02%
8	3	70	70	0%
9	10	111	144	29.73%
14	2	79	105	32.91%
22	5	96	110	14.58%
29	1	135	178	31.85%
35	6	72	73	1.39%
40	7	87	87	0%
45	5	173	226	30.64%

TABLE 4.11: Résultats de *LORH_ALNS* sur la famille J120 des instances PSPLIB.

J120		***PSPLIB***	***LORH_ALNS***	
ID paramètres	**ID instance**	***Makespan***	***Makespan***	$Gap_{makespan}$
2	5	103	133	29.13%
3	9	86	101	17.44%
4	9	79	90	13.92%
5	2	80	88	10%
6	2	134	192	43.28%
7	5	131	178	35.88%
8	7	87	136	56.32%
9	5	114	175	53.51%
17	10	134	175	30.6%
23	7	104	127	22.12%
34	8	89	115	29.21%

optimale et le résultat de *LORH_ALNS* sont plus importants que dans les familles précédentes, et montent jusqu'à 56.32%.

Sur les instances de la littérature RCPSP, nous obtenons avec notre algorithme *LORH_ALNS* des résultats peu concluants, avec des écarts de plus en plus importants en fonction de la taille des instances. Sur les familles J30, J60, J90 et J120,

nous obtenons des écarts moyens de 10.59%, 11.16%, 17.1% et 26.52%, respectivement. Ces résultats peuvent être expliqués par les différences structurelles entre nos instances et les instances de la littérature. Tous nos mouvements ont été spécialement conçus pour les particularités de instances d'Evolucare. Les scores de difficulté associée aux rendez-vous et de stress associé aux ressources sont beaucoup moins pertinents sur les instances PSPLIB. La difficulté de ces instances réside dans les nombreuses relations de précédence, qui sont peu prises en compte dans nos mouvements.

Conclusion

Ce chapitre a été consacré à la description d'une méthode de recherche locale appelée Adaptive Large Neighborhood Search (ALNS) et des différents mouvements qu'elle utilise pour explorer l'espace de recherche. Nous avons dans un premier temps décrit notre méthode de construction de solutions initiales, qui place les rendez-vous soit au plus tôt soit à des créneaux aléatoires avec les premières ressources disponibles.

Les différents mouvements de destruction et de construction ont ensuite été présentés et expliqués. Au nombre de six, ils explorent des voisinages différents et effectuent des choix en fonction de critères tels que la difficulté à placer certains rendez-vous, leur importance ou leur priorité ou encore l'encombrement du plannings des diverses ressources du problème.

Ces mouvements ont ensuite été intégrés dans l'ALNS. Cette méthode de recherche locale choisit et utilise à chacune de ses itérations l'un des mouvements pour l'appliquer à la solution. Selon les améliorations apportées à la solution, les probabilités associées aux mouvements évoluent au fil de l'exécution de l'algorithme. Ainsi, certains mouvements sont privilégiés s'ils se montrent efficaces sur un problème, ce qui permet en théorie à l'ALNS de rester efficace sur de nombreux problèmes différents.

Nous avons confronté trois approches sur les instances de l'entreprise Evolucare : l'algorithme de construction aléatoire, l'ALNS et une version de cette méthode privée de sa couche adaptative, nommée (¬A)LNS. Nous avons également comparé les

résultats de ces méthodes aux solutions optimales obtenues avec CPLEX dans le chapitre précédent.

Ces résultats nous fournissent des nouvelles solutions optimales et de nouvelles bornes supérieures obtenues par nos deux recherches locales sur de nombreuses instances, particulièrement celles composées d'un grand nombre de rendez-vous à planifier. Nous constatons aussi que l'algorithme de construction est efficace dans son rôle qui consiste à fournir des solutions de qualité suffisante rapidement, afin d'être les solutions initiales de nos autres approches. Nous constatons que les taux de rendez-vous planifiés obtenus par les méthodes de recherche locale sont élevés pour toutes les instances de tous les scénarios, avec un taux moyen de 98.32% pour l'ALNS. Ces scores nous font supposer que la difficulté sur la plupart de nos instances réside dans la planification des quelques derniers rendez-vous. Des progrès peuvent aussi être faits sur les instances avec des rendez-vous prioritaires (combinés ou pas à d'autres facteurs de difficulté), sur lesquels il reste une marge d'amélioration vis-à-vis des bornes inférieures LB de CPLEX.

Sur les instances de la bibliothèque PSPLIB, nous obtenons des résultats plus mitigés. Sur toutes les familles d'instances, nous constatons des différences importantes entre le *makespan* trouvé par $LORH_ALNS$ et les résultats de la littérature. Ces mauvaises performances s'expliquent par les différences entre les instances d'Evolucare, pour lesquelles les mouvement utilisés au sein de l'ALNS ont été créés, et les instances de la littérature. Contrairement aux nôtres, ces dernières contiennent des chaînes de précédence bien plus nombreuses et denses, et nos mouvements sont très peu adaptés à ce type de problématique.

Chapitre

5

Algorithme génétique

Contents

5.1 Principe de l'algorithme génétique

5.1.1 Principe général

5.1.2 Codage des solutions

5.1.3 Reproduction

5.1.4 Mutation

5.1.5 Sélection

5.1.6 Algorithme général

5.2 Métriques et convergence de l'algorithme

5.2.1 Métriques

5.2.2 Calcul de diversité

5.2.3 Algorithme génétique avec calcul de distance

5.3 Expérimentations & Résultats

5.3.1 Paramétrage

5.3.2 Résultats sur les instances de l'entreprise

5.3.3 Résultats sur les instances PSPLIB

Dans ce chapitre nous présentons notre adaptation de l'algorithme génétique à notre problème. En premier lieu nous rappellerons les principes de cette métaheuristique avant de décrire le codage des solutions que nous utilisons. Tous nos opérateurs génétiques (de reproduction, de mutation et de sélection) seront également décrits. La section suivante sera consacrée au phénomène de convergence et son impact sur le déroulement d'un algorithme génétique. Nous présenterons

nos métriques utilisées pour tenter de limiter ce phénomène et ainsi améliorer les résultats de l'algorithme génétique. Nous terminerons en présentant les résultats obtenus avec ces approches.

5.1 Principe de l'algorithme génétique

Les algorithmes génétiques font partie des algorithmes évolutionnaires, une famille de méthodes du domaine de l'intelligence artificielle reproduisant des comportements ou des phénomènes naturels. À titre d'exemple, l'algorithme de recuit simulé, faisant lui aussi partie de cette famille d'algorithme s'appuie sur le phénomène de refroidissement d'un matériau préalablement chauffé : les solutions sont au lancement de l'algorithme très « malléables » et peuvent subir d'importants changements, avant de progressivement refroidir, limitant ainsi l'ampleur des mouvements qu'elles subissent. Les algorithmes d'optimisation par essaim figurent également parmi les algorithmes évolutionnaires et s'inspirent des comportements de certains animaux : le comportement de chasse des loups ou des baleines à bosse, la recherche de nourriture des fourmis, etc. Depuis de nombreuses années les chercheurs trouvent l'inspiration dans l'observation des animaux ou des phénomènes liés à la physique. De nos jours, de nouvelles approches évolutionnaires sont toujours proposées pour résoudre divers problèmes d'optimisation combinatoire.

5.1.1 Principe général

Ces algorithmes sont inspirés des théories de l'évolution d'abord suggérées par Jean-Baptiste de Lamarck (de Lamarck, 1809) puis énoncées par Charles Darwin (Darwin, 1859). Dans son ouvrage, Darwin théorise l'origine des espèces, qui ont évoluées à partir d'un ou plusieurs ancêtres communs. Cette évolution s'est faite par le phénomène de sélection naturelle : les individus qui survivent assez longtemps pour procréer sont ceux qui sont le plus adaptés à leur environnement. Des ancêtres communs sont nées d'autres espèces par des phénomènes de mutations, qui ont elles-mêmes ensuite disparues ou donné naissance à d'autres espèces. Ce long cycle de reproduction, de mutation et de sélection a abouti à l'apparition des animaux que

nous connaissons de nos jours, et c'est de ce cycle que les algorithmes génétiques s'inspirent pour faire évoluer un ensemble de solutions à un problème donné.

Le concept d'une machine reproduisant ce phénomène d'évolution est ancien et était déjà suggéré par Alan Turing dans les années 1950 (Turing, 1950). Dans les années qui suivirent, des travaux furent menés par de nombreux chercheurs (informaticiens et biologistes) tels que Alex Fraser (Fraser, 1957) et Jack Crosby (Crosby et al., 1973) pour développer des simulations capables de reproduire le phénomène d'évolution sur des machines. Les algorithmes génétiques que nous connaissons aujourd'hui ont été démocratisés par John H. Holland en 1975 (Holland John, 1975) et David E. Goldberg en 1989 (Golberg, 1989) et sont toujours utilisés de nos jours pour résoudre des problèmes d'optimisation complexes.

Le principe de l'algorithme génétique se calque sur les théories de l'évolution : une population de solutions (ou individus) est créée et passe par les cycles de reproduction, de mutation et de sélection pour progressivement faire émerger la solution la plus adaptée au problème posé. Lors de la reproduction, les informations génétiques de plusieurs individus sont recombinées pour créer de nouveaux individus. Ces individus nouvellement créés vont ensuite subir aléatoirement des mutations, qui vont modifier légèrement les gènes dont ils sont constitués. Enfin, une nouvelle population est construite en sélectionnant des survivants parmi les individus enfants et parents : les meilleures solutions sont plus susceptibles d'être choisies et de poursuivre leur évolution dans la génération suivante.

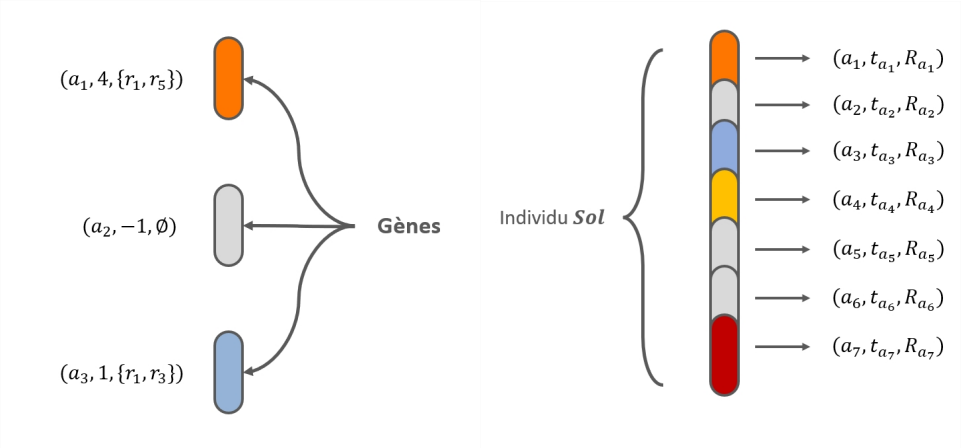
Dans les sous-sections suivantes, nous détaillons notre implémentation de l'algorithme génétique, notamment les mécaniques de reproduction et de mutation adaptées au problème de planification que nous traitons, et les méthodes de sélection utilisées au sein de l'algorithme. En premier lieu nous décrirons le codage de nos solutions pour ce problème.

5.1.2 Codage des solutions

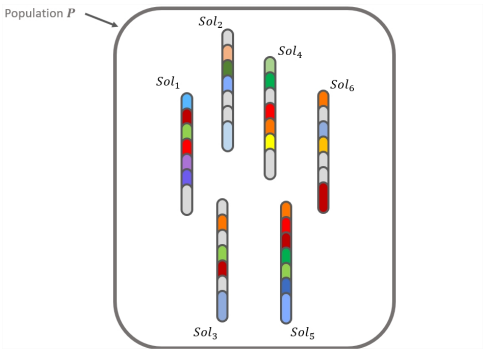
Le codage des solutions au sein d'un algorithme génétique correspond à la manière de représenter les solutions au problème. Il est par conséquent spécifique à chaque problème et varie parfois également selon les chercheurs et les approches

qu'ils utilisent. Les termes employés pour décrire ces éléments sont empruntés à la biologie et à la génétique.

L'algorithme génétique fait évoluer une population d'individus (aussi appelés phénotypes, ou chromosomes), qui sont dans notre cas un ensemble de solutions $P = \{Sol_1, Sol_2, ..., Sol_n\}$, avec n la taille de la population. Chaque individu Sol est constitué de gènes définissant leurs caractéristiques qui seront évaluées pour déterminer la qualité de la solution. Un gène correspond à un triplet (a, t_a, R_a) que nous avions défini pour nos méthodes précédentes pour représenter les solutions. Ainsi, un individu Sol est composé d'un ensemble de gènes (a, t_a, R_a), avec a une demande de rendez-vous, t_a la date de début de a et R_a l'ensemble des ressources affectées à a, pour chaque rendez-vous $a \in A$. Les solutions Sol peuvent contenir des demandes de rendez-vous qui n'ont pas encore reçu d'affectation de date ou de ressources. Pour ces gènes, t_a sera égal à -1 et l'ensemble R_a sera vide.



(a) Exemple de gènes. (b) Exemple d'individu.



(c) Exemple de population.

FIGURE 5.1: Illustrations du codage employé.

Nous illustrons le codage proposé dans les figures 5.1. Pour cela nous utilisons un exemple simple avec 7 demandes de rendez-vous ($A = \{a_1,a_2,a_3,a_4,a_5,a_6,a_7\}$) à planifier sur un horizon d'une semaine. Sur la figure **(a)**, nous schématisons 3 gènes, 3 triplets pour les trois premiers rendez-vous de A. Sur cet exemple, le premier rendez-vous a_1 est planifié au créneau 4 avec les ressources $\{r_1,r_2\}$. Le second rendez-vous n'a pas reçu d'affectation, donc $t_{a_2} = 0$ et $R_{a_2} = \emptyset$. Pour chaque affectation (créneau, ressources), nous colorons le gène d'une couleur différente, pour nous permettre de les différencier plus facilement pour nos futures explications. Les demandes n'ayant pas reçu d'affectation seront colorées en gris. Un individu complet est illustré en figure **(b)**. Pour chaque rendez-vous $a \in A$, un gène est représenté avec son affectation. L'ensemble de ces gènes forme un individu, une solution Sol. Certains gènes (pour les demandes a_2, a_5 et a_6) n'ont pas d'affectation de date de début ou de ressources au sein de cet individu et sont donc représentés en gris. Pour tous les individus, l'ordre des demandes de rendez-vous sera toujours identique. Les différences entre les individus se feront sur les dates et les ressources affectées aux rendez-vous. La figure **(c)** illustre un ensemble de solutions, chacune constituées de 7 gènes (pour chaque rendez-vous $a \in A$), qui forment une population de 6 individus. L'algorithme génétique agira sur cette population pour la faire évoluer au fil des générations et produire une solution de bonne qualité.

5.1.3 Reproduction

L'opération de reproduction, plus souvent appelée ***crossover***, consiste à combiner les informations de plusieurs individus pour en former un ou plusieurs individus enfants. Elle est inspirée de la recombinaison génétique qui se produit lors de la formation des cellules reproductrices chez les eucaryotes. Lors de la méiose, les chromosomes présents dans les cellules se séparent en chromatides qui s'assemblent ensuite par paires homologues : l'un des chromatides du père avec l'un des chromatides de la mère. Certains segments du génome seront échangés durant cet assemblage, échanges causés par des cassures et des réparations dans la cellule d'ADN chez les deux chromatides parents. Le chromosome produit portera des informations génétiques (des gènes) provenant du père et de la mère.

Le mécanisme de reproduction au sein d'un algorithme génétique est similaire : deux solutions de la population sont sélectionnées pour produire deux individus. Les solutions parents sont découpées en plusieurs endroits puis recombinés dans la solution enfant. L'individu ainsi formé sera constitué de parties de solutions provenant de ses deux parents. Les solutions résultant d'un tel mélange peuvent être incohérentes et ne plus respecter les diverses contraintes du problème. Dans ces cas là, les chromosomes enfants passent par un processus de réparation pour corriger les éventuels problèmes provoqués par la recombinaison des informations. Pour le bon fonctionnement de l'algorithme génétique, la phase de reproduction revêt une importance particulière : elle permet d'une part de produire des solutions potentiellement meilleures que leurs parents, en profitant des bonnes parties de solution du père et de la mère. D'autre part, elle entretient la diversité génétique de la population en produisant des solutions a priori différentes de la population de parents.

5.1.3.1 Méthodes de crossover

De nombreuses méthodes existent pour réaliser le crossover. L'approche la plus commune, qui peut être utilisée sur la majorité des problèmes, est le crossover à un point. En considérant une représentation des individus par un ensemble de gènes contigus, semblable à notre représentation en figure 5.1, le crossover à un point consiste à marquer une position, puis à couper les deux parents en suivant ce marqueur. Les deux parties de chaque parent sont ensuite combinés pour former deux nouveaux individus.

Nous pouvons illustrer ce type de crossover avec notre représentation décrite plus tôt. Pour cela nous posons à nouveau un problème simple avec 5 demandes de rendez-vous $\{a_1,a_2,a_3,a_4,a_5\}$, ne nécessitant toutes qu'une ressource r_1, qui leur est pre-assignée, et ayant une durée d'une heure. Sur la figure 5.2 nous représentons deux solutions Sol_1 et Sol_2 et l'emplacement des rendez-vous posés pour chaque individu sur l'emploi du temps de l'unique ressource r_1.

Un entier p est choisi aléatoirement tel que $1 \leq p \leq |A|$. La solution fille Sol_F est formée avec les triplets $(a,t_a,R_a) \in Sol_M,Sol_P$ de la mère M et du père P en positions $i = 1,...,|A|$, tel que :

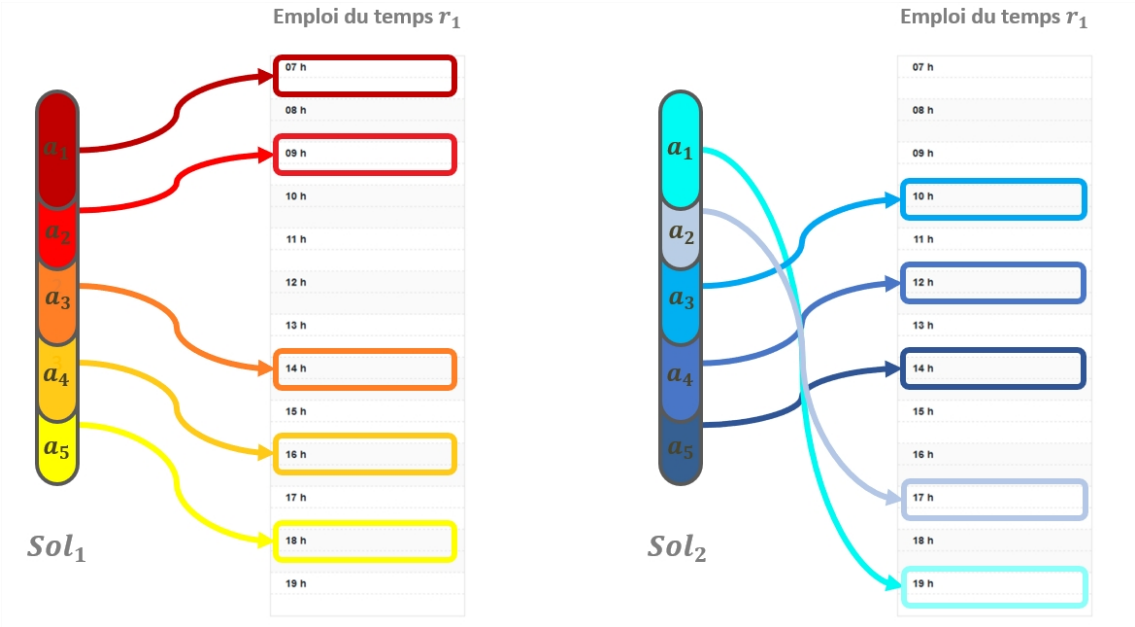


FIGURE 5.2: Deux solutions Sol_1 et Sol_2 choisies pour le crossover.

$$(a_i^F, t_{a_i}^F, R_{a_i}^F) = \begin{cases} (a_i^M, t_{a_i}^M, R_{a_i}^M) & \text{pour } 1 \leq i \leq p. \\ (a_i^P, t_{a_i}^P, R_{a_i}^P) & \text{pour } p \leq i \leq |A|. \end{cases} \tag{5.1}$$

avec a_i^F, a_i^M et a_i^P les gènes en position i de la solution fille Sol_F, de la solution mère Sol_M et de la solution père Sol_P, respectivement. La solution fils est formée de la même façon, en prenant les triplets du père puis de la mère. Sur la figure 5.3 nous appliquons ce crossover sur notre exemple précédent avec $p = 2$. Les solutions Sol_3 et Sol_4 seront créées en fonction des solutions Sol_1 et Sol_2 et de ce point de crossover p. La solution Sol_3 hérite ainsi des triplets en position $i = 1{,}2$ de Sol_2 et des triplets en position $i = 3{,}4{,}5$ de Sol_1. Inversement, l'individu Sol_4 est formé en prenant les premiers triplets dans Sol_1 puis de Sol_2.

Les emplois du temps obtenus par cette méthode sont illustrés sur la figure 5.4. Ce crossover à un point peut être généralisé en crossover à n point, avec des points de crossover $p_1, p_2, \ldots, p_n$ tels que $1 \leq p_1 < p_2 < \ldots < p_n \leq |A|$. Les triplets en position a_1 à a_{p_1} sont alors transmis par la mère, puis les triplets en position a_{p_1} à a_{p_2} par l'individu père, et ainsi de suite pour tous les points de crossover. Bien qu'il soit possible de faire des croisements avec de nombreuses coupures, les crossovers les plus utilisés dans la littérature restent les crossovers à un et deux points.

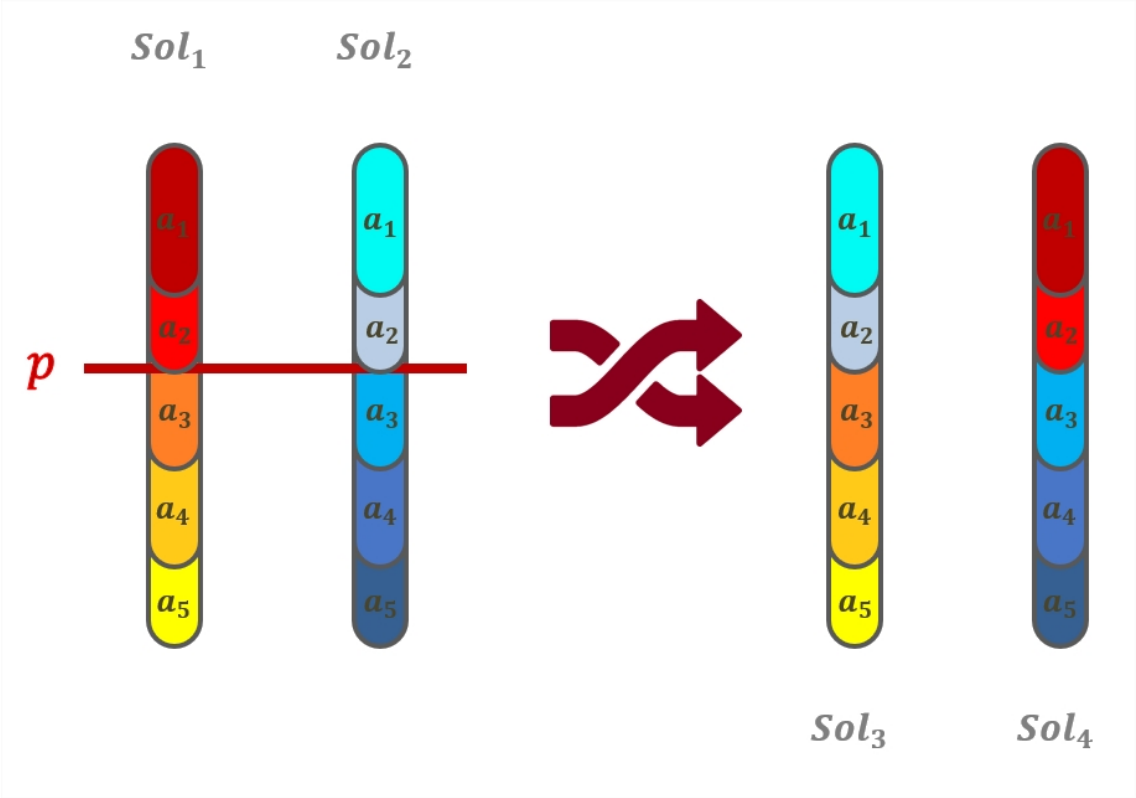


FIGURE 5.3: Séquences génétiques des solutions Sol_3 et Sol_4 obtenues après le crossover.

Le crossover uniforme est aussi une méthode très utilisée, et consiste à choisir aléatoirement pour chaque gène de l'enfant s'il provient de la solution « mère » ou « père ». Pour notre problème, avec un individu père P et un individu mère M et une probabilité $p = [0,1] \in \mathbb{R}$ nous avons :

$$(a_i^F, t_{a_i}^F, R_{a_i}^F) = \begin{cases} (a_i^M, t_{a_i}^M, R_{a_i}^M), & \text{si } p < 0.5. \\ (a_i^P, t_{a_i}^P, R_{a_i}^P) & \text{sinon.} \end{cases} \tag{5.2}$$

Le deuxième enfant est formé avec les gènes de l'autre parent à chaque position i. Ces quelques méthodes de reproduction de la population sont les plus connues et sont génériques : elles peuvent être appliquées à n'importe quel problème représenté sous forme de gènes contigus. D'autres approches existent, plus spécifiques et réservées à certains problèmes. C'est le cas des problèmes RCPSP, pour lesquels le nombre important de relations de précédence impose une représentation spécifique et des méthodes de crossover particulières pour préserver la cohérence des solutions au fil des opérations.

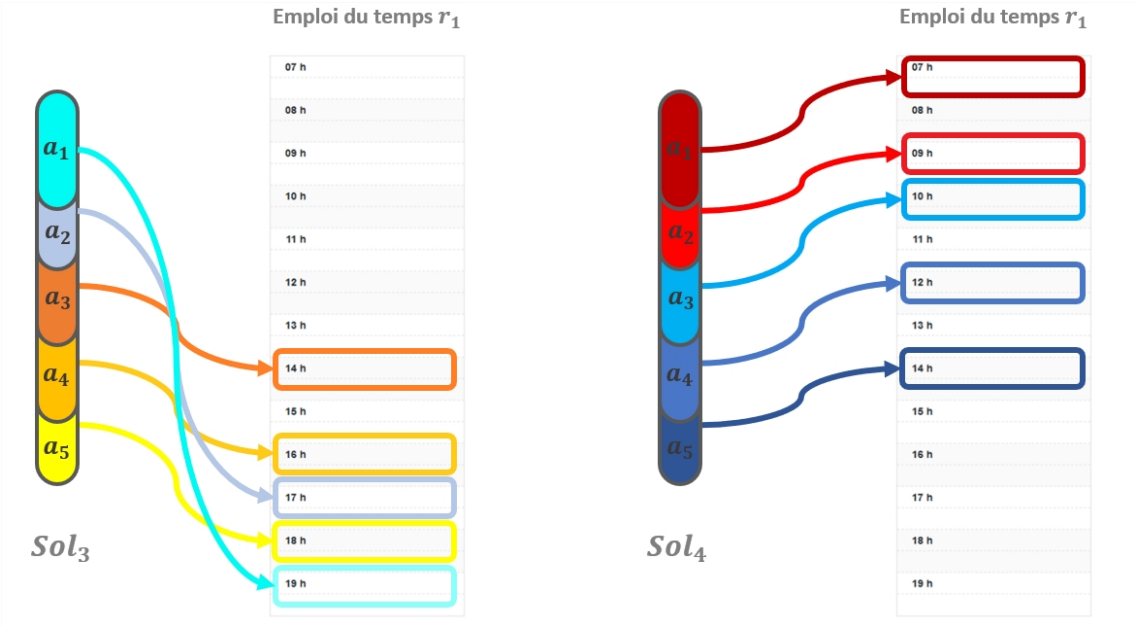


FIGURE 5.4: Emploi du temps de la ressource r_1 obtenus après le crossover.

5.1.3.2 Réparations

Nous l'évoquions plus tôt, la reproduction et les dégâts occasionnés par les recombinaisons sur les gènes peuvent nécessiter des réparations, que ce soit dans le phénomène biologique ou sa contrepartie informatique. Pour notre problème, cela se produit lorsqu'après avoir combiné les rendez-vous des parents, certains rendez-vous utilisent les mêmes ressources au même moment. Autrement dit, si pour deux rendez-vous planifiés (a_x,t_{a_x},R_{a_x}) et (a_y,t_{a_y},R_{a_y}), nous avons $\{t_1;t_{a_x}+ duration_{a_x}\} \cap \{t_2;t_{a_y}+ duration_{a_y}\} \neq \emptyset$ et $R_{a_x} \cap R_{a_y} \neq \emptyset$.

Une telle situation peut se produire après un crossover, et est illustrée dans la figure 5.5. Nous reprenons le problème posé pour illustrer le principe du crossover, mais cette fois, nous appliquons le crossover sur deux points $p_1 = 2$ et $p_2 = 4$. Avec ce découpage nous obtenons les individus Sol_3 et Sol_4. Nous remarquons que l'individu Sol_3 contient une anomalie : la même date de début (14 heures) a été affectée aux demandes a_3 et a_5, avec la même ressource r_1. Cette solution ne peut rester en l'état car elle enfreint la contrainte **CD 7** (3.1.2) stipulant qu'une ressource ne peut être affectée simultanément à deux rendez-vous au même moment. La solution Sol_3 nécessite donc une réparation.

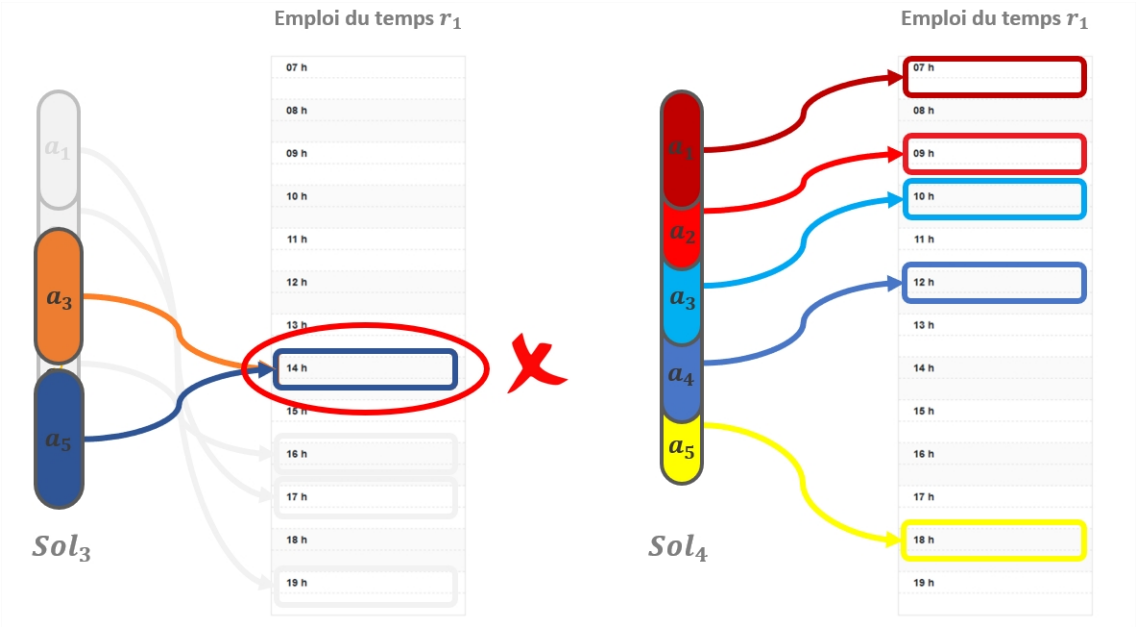


FIGURE 5.5: Séquences génétiques des solutions Sol_3 et Sol_4 après un crossover à deux points.

Pour réparer un individu, certaines affectations problématiques sont retirées pour permettre à la solution de retrouver sa cohérence. Pour tous les rendez-vous planifiés (a_x,t_{a_x},R_{a_x}) et (a_y,t_{a_y},R_{a_y}) d'une solution Sol tels que $\{t_1;t_{a_x}+ duration_{a_x}\} \cap \{t_2;t_{a_y}+ duration_{a_y}\} \neq \emptyset$, l'un ou l'autre est retiré (probabilité 50/50) de la solution Sol. Dans l'exemple présenté ci-dessus, ce sont les affectations pour les demandes a_3 et a_5 qui sont en conflit. Un jet aléatoire est effectué pour les départager, et l'affectation perdante est retirée de la solution. Plusieurs affectations peuvent être supprimées de cette manière. Chaque solution enfant passera par une phase de saturation après la mutation qui tentera de planifier le plus de demandes sans affectation possible.

5.1.3.3 Crossover sur l'ordre des rendez-vous

Dans le chapitre 3, nous avions constaté des différences non négligeables entre les instances de l'entreprise et les instances PSPLIB, qui sont des instances de référence de la littérature RCPSP. La plus notable concernait les relations de précédences, qui sont très nombreuses dans les instances PSPLIB en comparaison avec les instances Evolucare. L'ALNS décrit dans le chapitre 4 a d'ailleurs obtenu des performances moyennes sur ces instances, car aucun des mouvements que nous avons développés

ne traitait spécifiquement de ces relations de précédence. Avec l'algorithme génétique, nous avons fait le même constat : sans gestion particulière des chaînes de précédences dans les phases de reproduction et de mutation, les résultats sont également décevants. Pour remédier à cela, nous avons donc intégré dans nos opérations de crossover un traitement spécifique pour les rendez-vous chaînés entre eux.

Pour améliorer les performances de notre algorithme génétique sur les instances PSPLIB, nous nous sommes inspirés des algorithmes génétiques de la littérature traitant des problèmes RCPSP et des instances PSPLIB en particulier. Nous avons notamment repris le principe des crossover et des mutations de l'article d'Alcaraz et Maroto (Alcaraz and Maroto, 2001). Dans cet article, et pour de nombreux algorithmes génétiques traitant de problèmes similaires, les solutions sont représentées sous formes de listes d'activités.

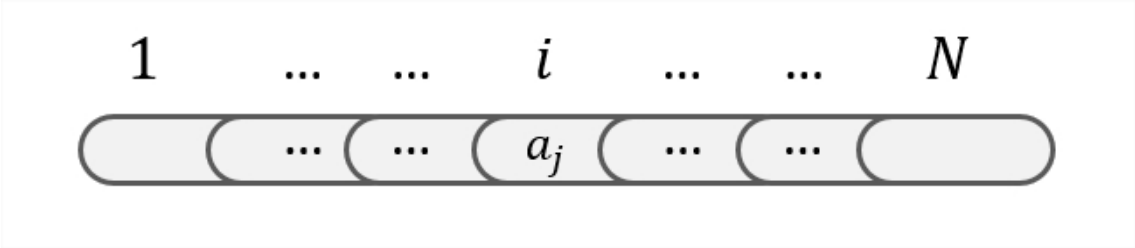


FIGURE 5.6: Exemple d'une solution représentée avec le codage RCPSP.

La figure 5.6 reprend l'illustration donnée dans l'article de Alcaraz and Maroto (2001), pour un problème avec N activités à planifier. À l'instar de notre représentation, chaque individu comporte autant de positions que d'activités à planifier, mais à la différence de notre codage, les individus ne représentent pas directement un planning. Ici, c'est l'ordre de planification qui est représenté. Dans cet exemple, le calcul de la planification de l'activité a_j en position i se fera après le calcul de la planification de toutes les activités qui la précèdent et apparaissent dans les positions 1,...,$i-1$. De cette manière, les emplois du temps générés respectent toujours les relations de précédence. Pour générer un planning, plusieurs méthodes existent et sont décrites dans l'article. La première, le « forward scheduling », consiste à planifier les activités dans l'ordre en partant du début de la liste, où chaque activité est planifiée après ses prédécesseurs. Dans la seconde, le « backward scheduling », les activités sont planifiées en partant de la fin de la liste. Dans ce mode, chaque activité est

planifiée avant ses successeurs à sa date au plus tard. Un gène est ajouté à chaque individu pour définir la méthode de génération de l'emploi du temps, forward ou backward. Cette représentation de méta-informations au sein des individus étant incompatibles avec notre représentation, nous ignorons ces gènes supplémentaires, qui ne seront pas ajoutés à nos individus.

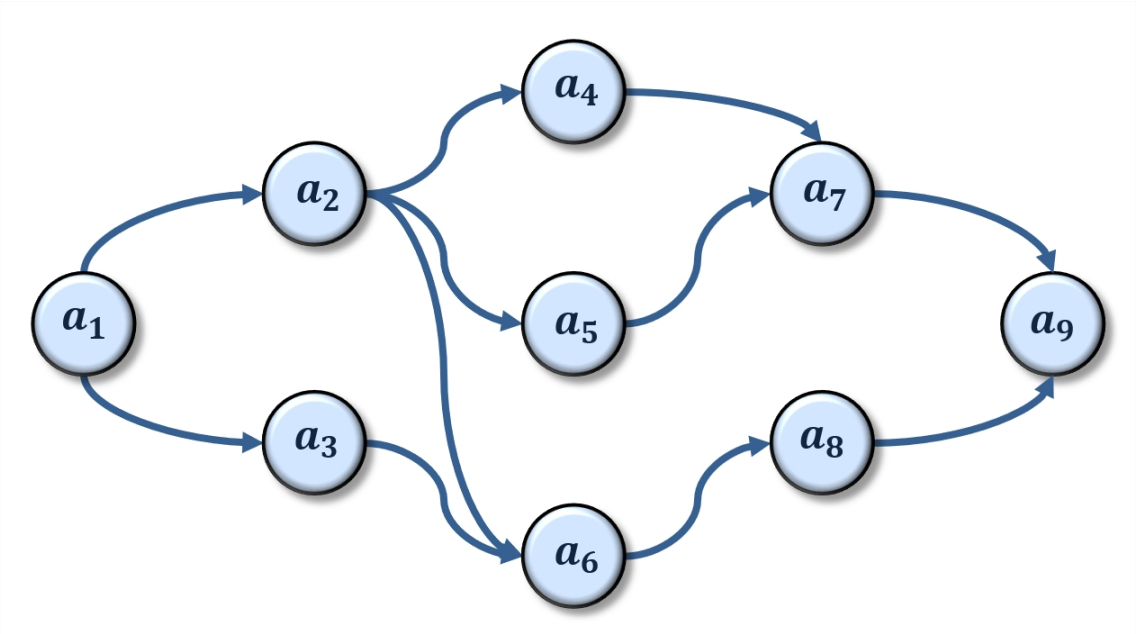


FIGURE 5.7: Relations de précédence entre les rendez-vous a_1, ..., a_9 de l'exemple.

Le crossover décrit par Alcaraz et Maroto agit donc sur l'ordre de planification et mêle les ordres de la solution mère avec ceux de la solution père. Pour illustrer ce crossover, nous reprenons l'exemple utilisé dans leur article. Nous avons donc un problème avec un ensemble $A = \{a_1,...,a_9\}$ de rendez-vous à planifier. Le graphe des relations de précédence est donné dans la figure 5.7.

Nous notons Ω_M et Ω_F deux ensembles des demandes de rendez-vous $a \in A$ ordonnés selon leur date de début t_a. Le crossover à deux points employé par Alcaraz and Maroto (2001) produit deux nouveaux ordres Ω_D et Ω_S à partir de l'ordre Ω_M de la mère et de l'ordre Ω_F du père. Deux points de crossover p_1 et p_2 sont choisis tels que $1 \leq p_1 < p_2 < |A|$. Pour former un ordre fille Ω_D, les rendez-vous en position $1,...,p_1$ de la mère Ω_M sont ajoutés à la fille. Les rendez-vous en position $p_1{+}1,...,p_2$ chez le père Ω_F sont ajoutés dans leur ordre relatif à Ω_F. Enfin, les rendez-vous en position $p_2{+}1,...,|A|$ de la mère sont finalement ajoutés à la fille Ω_F, dans leur ordre relatif.

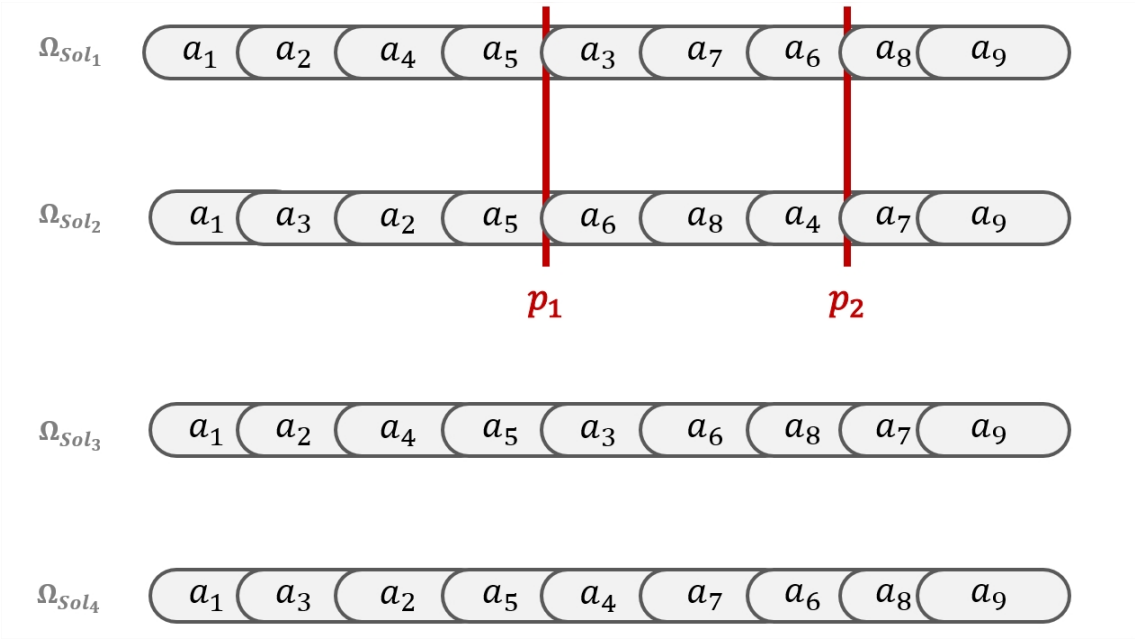


FIGURE 5.8: Génération de deux enfants Sol_3 et Sol_4 à partir des solutions Sol_1 et Sol_2, avec un crossover à deux points.

La figure 5.8 représente deux ensembles Ω_{Sol_1} et Ω_{Sol_2} que nous utilisons pour générer deux ensembles enfants Ω_{Sol_3} et Ω_{Sol_4}. Les ensembles parents respectent les relations de précédence de la figure 5.7. Dans notre exemple, nous avons $p_1 = 4$ et $p_2 = 7$. Pour former le premier ensemble Ω_{Sol_3}, les rendez-vous en position 1,...,3 de Ω_{Sol_1} sont ajoutés. Ensuite, les rendez-vous en position p_1+1,...,p_2 sont pris dans leur ordre relatif dans Ω_{Sol_2}. En position 5 dans Ω_{Sol_2}, nous trouvons le rendez-vous a_6. Celui-ci doit être précédé par a_3, or celui-ci n'est pas encore présent dans l'ensemble enfant Ω_{Sol_3}, il est donc ajouté juste avant a_6. Toujours suivant l'ordre dans Ω_{Sol_2}, le rendez-vous a_8 est ajouté à Ω_{Sol_3}. Le rendez-vous a_4 est déjà présent dans l'ensemble enfant, il n'est pas nécessaire de l'ajouter. Enfin, les rendez-vous en position p_2+1,...,$|A|$ sont repris dans leur ordre relatif dans Ω_{Sol_1}. Dans ce cas, a_8 est déjà présent dans Ω_{Sol_3}, à l'inverse de a_9. Ce dernier est précédé par a_7, qui n'est pas encore dans Ω_{Sol_3}, a_7 et a_9 sont donc ajoutés en dernier dans Ω_{Sol_3}. Ω_{Sol_4} est formé de manière similaire, en commençant par ajouter les rendez-vous de Ω_{Sol_2}, puis de Ω_{Sol_1} avant de terminer avec Ω_{Sol_2}, toujours dans leur ordre relatif.

Pour recourir à cette forme de crossover, et pouvoir l'utiliser au sein de notre algorithme avec notre représentation des solutions, nous devons opérer quelques changements. Dans le chapitre précédent, nous expliquions que pour la construction

des solutions, nous relâchions la contrainte **CD 5** (3.1.2) concernant les précédences, et nous permettions la planification de rendez-vous même si leur prédécesseurs ne sont pas planifiés. Par conséquent certaines solutions peuvent contenir des chaînes de rendez-vous incomplètes. La première étape pour que nous puissions appliquer le crossover décrit ci-dessus est donc de calculer l'ordre des rendez-vous des parents, puis de le réparer en y intégrant tous les rendez-vous qui ne sont pas planifiés, en respectant les précédences. Pour cela, nous calculons l'ensemble Ω des rendez-vous triés selon leur date de début t_a dans les solutions parents. Les rendez-vous manquants $a \in A_{\overline{Sol}}$ sont ajoutés dans une position i choisie aléatoirement entre la position $i_{maxPred}$ du dernier prédécesseur de a et la position $i_{minSucc}$ du premier successeur de a.

Algorithme 13: crossoverOrder()

Données: deux solutions Sol_1 et Sol_2

Résultat: deux solutions enfants Sol_3 et Sol_4

1 **Début**

2 $\Omega_{Sol_1} \leftarrow computeOrder(Sol_1)$; $\Omega_{Sol_2} \leftarrow computeOrder(Sol_2)$

3 $\Omega_{Sol_1} \leftarrow repairOrder(\Omega_{Sol_1}, Sol_1)$

4 $\Omega_{Sol_2} \leftarrow repairOrder(\Omega_{Sol_2}, Sol_2)$

5 $\Omega_{Sol_3} \leftarrow recombinateOrder(\Omega_{Sol_1}, \Omega_{Sol_2})$

6 $\Omega_{Sol_4} \leftarrow recombinateOrder(\Omega_{Sol_2}, \Omega_{Sol_1})$

7 $Sol_3 \leftarrow \emptyset$; $Sol_4 \leftarrow \emptyset$

8 **Pour chaque demande** $a_1 \in \Omega_{Sol_3}$ **et** $a_2 \in \Omega_{Sol_4}$ **faire**

9 $Sol_3 \leftarrow trySetSingleAppointment(a_1, Sol_3)$

10 $Sol_4 \leftarrow trySetSingleAppointment(a_2, Sol_4)$

11 **Fin**

12 Retourne Sol_3 et Sol_4

13 **Fin**

La procédure pour générer des enfants est décrite dans l'algorithme 13. Nous utilisons le croisement des ordres des rendez-vous décrit dans l'article d'Alcaraz et Maroto, et pour cela nous utilisons également de manière transitoire la représentation des solutions sous forme de liste de rendez-vous. Les fonctions $computeOrder()$ et $repairOrder()$ permettent de passer de notre représentation à cette liste de rendez-

vous, triés selon leur date de début, puis de réparer l'ordre si celui-ci ne comporte pas tous les rendez-vous $a \in A$. Les ordres permettant de générer les enfants sont ensuite créés avec la fonction $recombinateOrder()$ décrite précédemment. Enfin, ces ordres sont utilisés pour créer les solutions enfants Sol_3 et Sol_4 avec la fonction $trySetSingleAppointment()$ décrite dans l'algorithme 3 du chapitre 4.

Cette méthode a pour objectif de réduire les réparations nécessaires sur les enfants après la phase de reproduction, qui sont très nombreuses sur les instances riches en relations de précédence avec les méthodes de crossover décrites précédemment. Les performances de chaque méthode seront comparées dans la section 5.3 afin de choisir le type de crossover utilisé sur les instances de l'entreprise et sur les instances PSPLIB.

5.1.4 Mutation

La mutation est une opération appliquée sur tous les individus issus de la phase de reproduction et consiste à altérer légèrement une solution en faisant « muter » quelques uns de ses gènes. Cette opération est elle aussi inspirée d'un phénomène naturel (parfois artificiel) se produisant sur un sous-ensemble restreint de gènes et modifiant l'information génétique présente dans le génome par l'insertion, la suppression ou la substitution des nucléotides qui le composent. La mutation a joué un rôle important dans l'évolution des espèces, en dotant parfois des individus d'un avantage évolutif leur permettant de mieux s'adapter à leur environnement.

Elle a également un rôle important dans un algorithme génétique : elle permet d'introduire des nouvelles informations, de nouveaux gènes, qui peuvent rapprocher l'individu d'une solution optimale. Si ces bons gènes ne sont pas présents lors de l'initialisation de la population, ils ne pourront pas apparaître par les croisements d'individus, par définition. La fréquence des mutations dépend généralement d'un paramètre appelé taux de mutation, qui a un impact prépondérant sur les performances de l'algorithme génétique. Avec un taux trop bas, les nouveaux éléments (parfois bénéfiques) apparaîtront trop peu souvent et la recherche sera fortement ralentie. Avec un taux trop élevé, les individus subiront des modifications trop importantes et perdront l'héritage de leurs parents. Les solutions ne pourront plus se construire efficacement au fil du temps via les crossover et les mutations, trans-

formant l'algorithme génétique en recherche aléatoire. L'ampleur du changement apporté à un individu dépend de l'incidence de la mutation, en plus de sa fréquence.

L'opération de mutation est spécifique au problème et varie selon les approches choisie. Dans les sous-parties suivantes, nous présentons deux opérations de mutation : la première change la date de début t_a et tente de trouver un autre créneau pour planifier le rendez-vous a, potentiellement avec un ensemble de ressources R_a différent. La seconde, inspirée des méthodes utilisées sur les problèmes RCPSP, reprend la notion d'ordre des rendez-vous Ω décrite pour les crossover et insère à une position aléatoire un rendez-vous entre son premier prédécesseur et son premier successeur.

5.1.4.1 Mutation de la date de début

Cette mutation consiste à changer la date de début de rendez-vous sélectionnés aléatoirement, dont le nombre dépend du taux de mutation. Pour une solution Sol et pour chaque demande $a \in A_{Sol}$, un tirage aléatoire est effectué et comparé au taux de mutation τ. S'il est supérieur au taux de mutation, le gène (a,t_a,R_a) subit une modification de t_a.

Algorithme 14: mutate()

Données: une population $P_{children}$, une probabilité de mutation tau

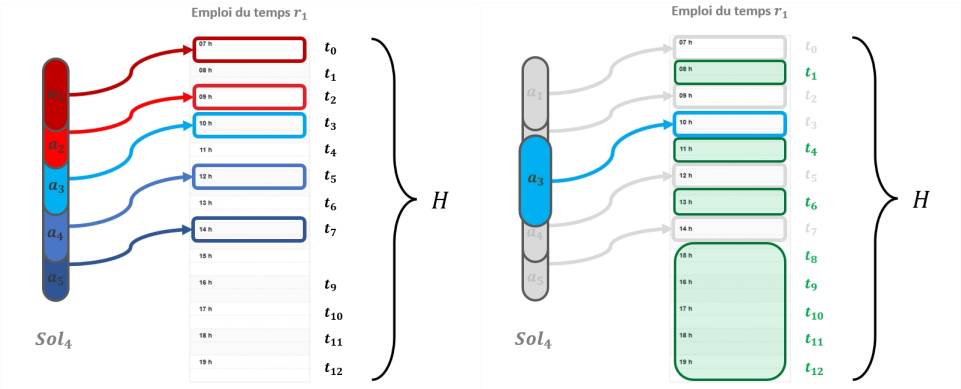
Résultat: une solution Sol

```
1  Début
2      Pour chaque solution Sol ∈ P_children faire
3          Pour chaque triplet (a,t_a,R_a) ∈ A_Sol faire
4              Si random() < τ alors
5                  Sol ← Sol \ {(a,t_a,R_a)}
6                  trySetSingleAppointment(a,H \ {t_a},Sol);
7              Fin
8          Fin
9      Fin
10 Fin
```

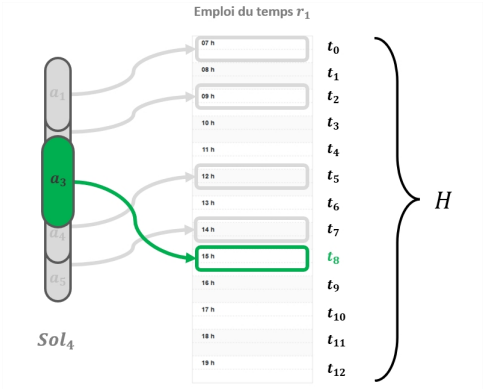
Un nouveau créneau t est recherché dans l'intervalle de faisabilité $[ES_a,LS_a]$, en excluant la date de début actuelle t_a. La recherche d'une nouvelle date de début utilise les fonctions $trySetSingleAppointment()$ et $findResources()$ décrites dans les algorithmes 3 et 4 du chapitre précédent, mais ces fonctions sont privées du créneau t_a sur lequel nous ne voulons plus planifier a. En partant soit du début de l'intervalle de faisabilité ES_a soit d'un créneau aléatoire au sein de ce même intervalle (probabilité équivalentes), un créneau sur lequel suffisamment de ressources sont disponibles pour répondre exactement à la demande a est recherché. Si un tel créneau existe, la recherche s'arrête et la nouvelle date et les ressources trouvées remplace t_a et R_a. Dans le cas d'un rendez-vous faisant partie d'une chaîne de précédence, d'autres rendez-vous de la chaîne peuvent être remis en cause. Pour chaque rendez-vous de la chaîne, un tirage aléatoire est effectué pour l'inclure ou non dans le processus de mutation. Si aucun créneau permettant de réunir les ressources nécessaires ne peut être trouvé, le triplet (a,t_a,R_a) est retiré de la solution Sol. C'est également le cas des rendez-vous $b \in preda$, qui restent sans affectation s'il n'a pas été possible de les replacer. S'il n'est pas possible de placer ces rendez-vous à un autre moment, retirer a et ses prédécesseurs ou successeurs est un moyen d'apporter malgré tout une modification à ce gène. Cela permet aussi de libérer des ressources qui permettront par la suite de placer d'autres rendez-vous. Le processus de mutation est décrit dans l'algorithme 14.

Pour illustrer cette mutation, nous utilisons l'un des enfants obtenus en figure 5.4 pour notre exemple de crossover, la solution Sol_4. Pour rappel, il s'agissait d'un problème avec 5 rendez-vous à planifier mettant en jeu une seule et unique ressource. Sur la figure 5.9a est représentée cette solution, avec l'emploi du temps de la ressource r_1 et les créneaux $\{t_0,t_1,...t_{12}\} \in H$. Dans notre exemple, le gène $(a_3,t_3,\{r_1\})$ est appelé à subir une mutation, que nous décrivons ici. Sur la figure 5.9b, nous voyons les créneaux potentiels pour planifier la demande a_3. Sur les créneaux t_0, t_2, t_5 et t_7, la ressource r_1 est déjà occupée par les autres rendez-vous du problème. Le créneau t_3 est la date de début actuelle attribuée à a_3 et ne peut donc plus être choisie à nouveau dans le cadre de la mutation.

La recherche d'un nouveau créneau débute à partir d'un créneau aléatoire, et sélectionne le premier créneau permettant de réunir les ressources nécessaires au



(a) État initial de l'individu Sol_4. (b) Créneaux possibles pour la demande a_3.



(c) Individu Sol_4 après mutation.

FIGURE 5.9: Exemple d'une mutation sur un individu.

rendez-vous. Pour cet exemple, nous prenons le créneau t_7 comme point de départ pour la recherche. Ce dernier étant déjà pris par la demande a_5, la recherche se poursuit au créneau suivant. La ressource r_1 est bien disponible sur le créneau t_8, nous pouvons donc placer la demande a_3 sur ce créneau. Le gène muté, à l'origine (a_3,t_3,{r_1}), sera remplacé par le triplet (a_3,t_8,{r_1}), et l'individu obtenu est illustré dans la figure 5.9c.

5.1.4.2 Mutation sur l'ordre des rendez-vous

Tout comme l'opération de reproduction, nous proposons aussi une opération de mutation agissant sur l'ordre de planification des rendez-vous. Sur les instances générées à partir de nos scénarios, la mutation consiste à reprogrammer un rendez-vous, et éventuellement ses prédécesseurs et successeurs, à un autre créneau de l'horizon. Les instances PSPLIB sont composées de nombreuses relations de précédence entre

les activités des projets à planifier, et utiliser la même mutation que sur les instances de l'entreprise pose plusieurs problèmes. Comme nous l'expliquions dans le chapitre 3, les graphes de précédences des instances PSPLIB ont une ou des composantes connexes très grandes et très denses, et le retrait en cascade (en fonction d'une probabilité donnée) utilisé lors de la mutation précédente revient généralement à détruire une grande partie de la solution. Même en agissant sur un seul rendez-vous, le placement de celui-ci sans toucher aux autres est très contraint par les dates de début de ces prédécesseurs et ses successeurs, ce qui mène souvent au retrait du rendez-vous de la solution.

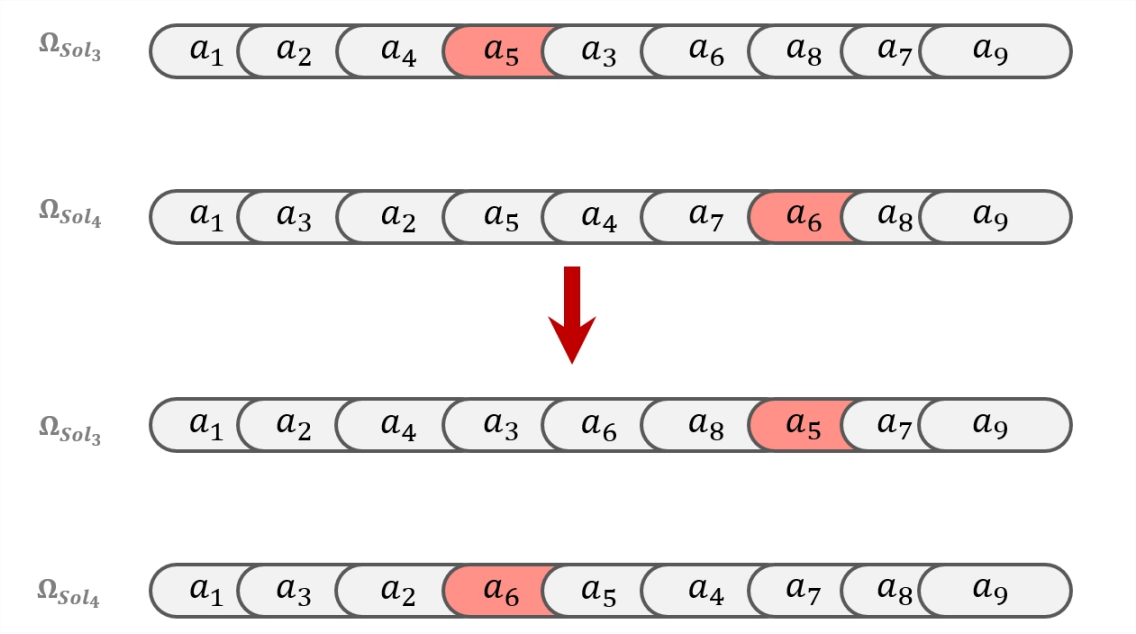


FIGURE 5.10: Exemple d'une mutation sur les gènes a_5 et a_6 des ensembles Ω_{Sol_3} et Ω_{Sol_4}.

Nous allons à nouveau nous inspirer de l'article d'Alcaraz et Maroto (Alcaraz and Maroto, 2001). La mutation décrite dans cet article agit toujours sur l'ordre des rendez-vous avant la génération de l'emploi du temps. Elle consiste à insérer le gène muté à une autre position i telle que $i_{min} < i < i_{max}$ dans la liste de rendez-vous Ω, avec $i_{min} = max(pred_a)$ la position de son premier prédécesseur et $i_{max} = min(succ_a)$ la position de son premier successeur, pour respecter les relations de précédence.

Un exemple est présenté dans la figure 5.10. Nous prenons à nouveau les ensembles Ω_{Sol_3} et Ω_{Sol_4} obtenus après l'opération de crossover décrite en figure 5.8.

Dans l'ensemble Ω_{Sol_3}, le gène a_5 (indiqué en rouge) est muté. En suivant les relations de précédence présentées en figure 5.7, nous avons $pred_{a_5} = \{a_2\}$ et $succ_{a_5} = \{a_7\}$, et les positions $i_{a_2} = 2$ et $i_{a_7} = 8$. a_5 peut donc être déplacé entre la position 2 et 8. Dans cet exemple, il est déplacé juste avant a_7, en position 7. Pour Ω_{Sol_4}, c'est le gène a_6 qui subit une mutation. Nous avons les relations de chaînage $pred_{a_6} = \{a_2,a_3\}$ et $succ_{a_6} = \{a_8\}$, et les positions $i_min = max(pred_{a_6}) = 3$ et $i_min(succ_a) = 8$. Dans notre exemple, il est déplacé en position 4 juste après a_2 son premier prédécesseur. Les nouvelles positions sont tirées aléatoirement entre les positions du premiers prédécesseur et successeur du gène muté.

5.1.5 Sélection

L'étape de sélection écarte les individus qui ne se sont pas suffisamment adaptés et permet aux autres de survivre plus longtemps, pour qu'ils évoluent et puisse produire une nouvelle génération. La sélection naturelle fut un phénomène long, complexe et global faisant émerger au fil des générations des individus ou des espèces plus adaptées à leur environnement et ayant ainsi plus de chance de survivre et de se reproduire. L'un des exemples le plus représentatif de cette sélection naturelle est celui de la phalène du bouleau : des scientifiques ont observé une prolifération des individus sombres, couleurs résultant d'une mutation, qui étaient de fait beaucoup moins visibles sur les bouleaux noircis par la pollution industrielle. La sélection naturelle était exercée dans ce cas par les prédateurs, qui chassaient les individus de couleurs claires, beaucoup plus visibles sur les bouleaux sombres.

Le principe de la sélection au sein de l'algorithme génétique est similaire : les solutions sont sélectionnées en fonction de leur adaptation au problème, elle-même mesurée par la fonction 3.1 quantifiant la qualité d'une solution. Cependant, la sélection doit respecter un équilibre délicat pour le bon fonctionnement de l'algorithme. D'une part, elle doit permettre aux meilleurs individus de survivre jusqu'à la prochaine génération pour qu'ils puissent transmettre leurs bonnes caractéristiques à leurs enfants, faisant ainsi progresser la recherche. D'autre part, elle ne doit pas écarter systématiquement les mauvais individus, sous peine d'avoir des populations d'individus excessivement proches les uns des autres, et freinant drastiquement la progression de la recherche. De plus, certaines de ces solutions de qualité moindre

peuvent néanmoins contenir des bons gènes, qui pourront faire progresser la recherche des meilleures solutions.

5.1.5.1 Politiques de sélection

Dans la littérature des algorithmes génétiques, nous distinguons deux politiques de sélection. Dans la première, une seule sélection est effectuée pour choisir les individus qui engendreront la population de la génération suivante. Avec une probabilité λ, les deux parents génèrent deux enfants, et avec une probabilité $1-\lambda$, les deux parents sont transmis directement à la génération suivante.

Dans la seconde politique de sélection, deux sélections sont effectuées. La première permet toujours de choisir les paires de parents qui produiront une population de solutions enfants, cette fois sans passer par le taux de crossover mentionné plus tôt. Une deuxième sélection est effectuée après la phase de mutation pour choisir parmi la population courante et celle des enfants ceux qui survivront et participeront à la génération suivante.

La première politique de sélection permet d'économiser quelques opérations de crossover lors de la phase de reproduction, qui peuvent être coûteuses sur certains problèmes, tout en permettant un brassage suffisant de la population. Les parents transmis directement à la génération suivante subiront également la phase de mutation. Une solution peut ainsi survivre et être modifiée pendant plusieurs générations. La deuxième politique de sélection permet aux bons individus de passer d'une génération à l'autre sans subir de modification (par le crossover ou la mutation), au prix d'une sélection supplémentaire.

Pour notre algorithme génétique, nous avons adopté la seconde politique de sélection, qui a obtenu des meilleurs résultats que la première. Dans les sous-parties suivantes nous décrivons les méthodes de sélection utilisés pour choisir les individus, indépendamment de la politique.

5.1.5.2 Sélection par tournoi

La sélection par tournoi est une des premières méthodes utilisées au sein d'un algorithme génétique pour choisir les individus pour la reproduction (Goldberg et al., 1989). Elle est simple à implémenter, demande peu de calculs et permet l'ajustement

de la pression de sélection. Plus la pression est forte, plus la probabilité pour les individus de moindre qualité d'être choisis est faible. Au contraire, une pression plus faible leur donne plus de probabilité de survivre et de se reproduire.

Le principe de la sélection par tournoi est le suivant : un nombre s d'individus sont sélectionnés aléatoirement au sein d'une population pour participer à un « tournoi ». Les qualités de chaque solution Sol_1,...,Sol_s sont comparées et la meilleure solution gagne le tournoi et participera à la phase de reproduction ou à la génération suivante. La taille s de chaque tournoi détermine la pression de sélection : plus elle est élevée, plus les individus faibles ont de chances de tomber contre des individus forts, et donc de ne pas être sélectionnés. Une version non-déterministe du tournoi consiste à ne pas systématiquement accepter le meilleur individu, mais de le choisir selon une probabilité p, le second avec une probabilité $p\times(1-p)$, et ainsi de suite. Dans cette version, la pression de sélection est contrôlée par la taille du tournoi s et la probabilité p de choisir le meilleur individu. Les tournois se poursuivent jusqu'à ce que le nombre souhaité de solutions soient atteint.

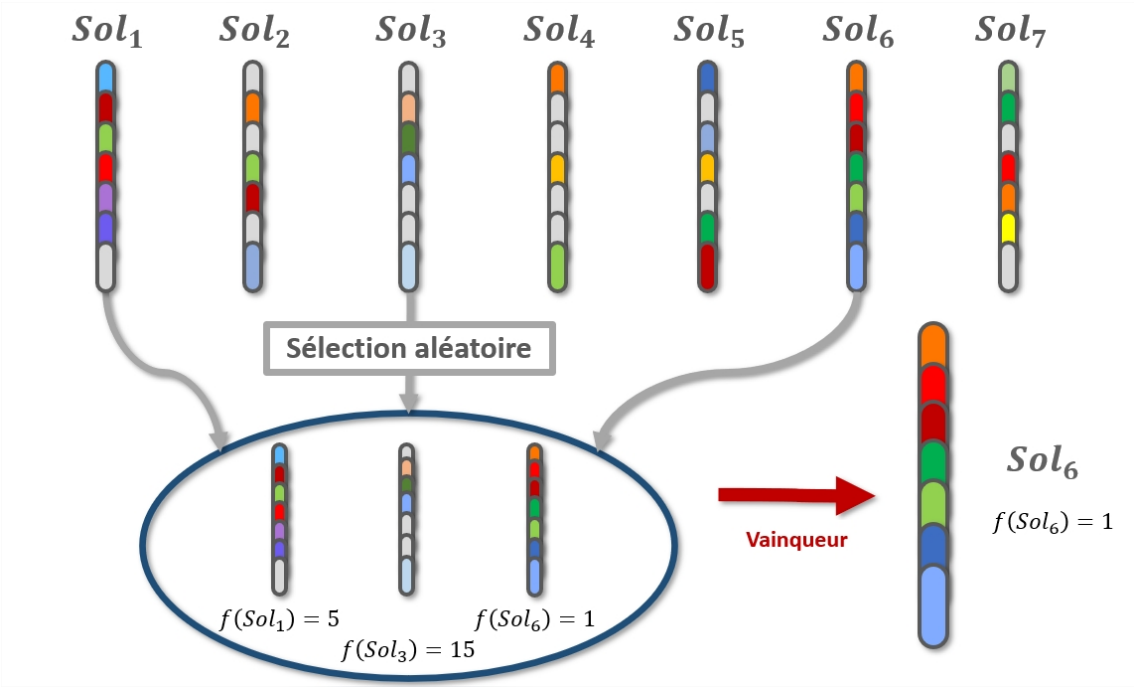


FIGURE 5.11: Exemple d'un tournoi déterministe sur une population de 7 solutions.

Le fonctionnement de la sélection par tournoi déterministe est illustré dans la figure 5.11. Une population $P = \{ Sol_1,...,Sol_7 \}$ passe par le processus de tournoi, avec une taille $s = 3$. Dans un premier temps, 3 solutions sont sélectionnées aléatoi-

rement, puis confrontées lors du tournoi. Dans le cas d'un problème de minisation, la meilleure solution ici est Sol_6, avec une qualité $f(Sol_6) = 1$ et remporte le tournoi. Elle sera intégrée soit à la population de parents soit à la génération suivante.

Des travaux, anciens (Miller et al., 1995) et plus récents (Lavinas et al., 2018), ont été menés pour analyser l'influence de la taille du tournoi sur la convergence de la population et les performances en général de l'algorithme génétique. Dans le cadre de ce manuscrit, nous confronterons divers paramétrages afin de sélectionner le plus efficace sur nos instances.

5.1.5.3 Sélection par roulette

La méthode de sélection par roulette, et plus largement les méthodes de sélection proportionnelles à la qualité des solutions, font également partie des méthodes utilisées très tôt dans la littérature des algorithmes génétiques. À l'instar de la sélection par tournoi, elles sont relativement simples à implémenter et permettent une sélection des individus par rapport à leur qualité.

La sélection par roulette, ou **Roulette Wheel Selection** (RWS), attribue sur une roue fictive une portion d'espace proportionnelle à sa qualité à chaque individu. Un tirage aléatoire simule son lancer et son arrêt sur une zone correspondant à un individu, qui sera sélectionné. Nous reprenons la définition de Sastry et al. (2005) : pour chaque individu i de la population, une probabilité p_i est calculée en fonction de sa qualité f_i :

$$p_i = 1-(f_i \,/\, \sum_{j=0}^{n} f_j) \tag{5.3}$$

où n est la taille de la population. La probabilité cumulative q_i est calculée selon la formule suivante :

$$q_i = \sum_{j=0}^{i} p_j \tag{5.4}$$

Un nombre aléatoire $r \in [0,1]$ est généré. La solution sélectionnée sera la solution i telle que $q_{i-1} < r < q_i$. Ce processus est répété jusqu'à avoir une population de parents ou de survivants complète. Cette méthode privilégie les individus d'une

qualité prometteuse, et peut provoquer des convergences rapides de la population vers des optimums locaux.

Une méthode dérivée de la sélection par roulette laissant plus de chances aux mauvais individus a été proposée par Goldberg and Deb (1991). Les chances d'être choisie pour chaque solution sont calculées en fonction de leur rang :

$$p_i = \frac{rank(i)}{n \times (n-1)} \tag{5.5}$$

La fonction $rank()$ renvoie le classement de l'individu i au sein de la population, calculé selon leur qualité. Pour n valeurs de qualité différentes au sein d'une population de solution, la ou les pires auront le rang 1 tandis la ou les meilleures obtiennent le rang n. Les solutions ayant la même évaluation de leur qualité partagent le même rang. Cette sélection par rang est plus permissive pour les solutions de qualité moindre, et leur offre plus de chances d'être sélectionnés que la sélection par roulette classique. Avec cette méthode, nous espérons réduire la vitesse de la convergence des solutions. Elle nécessite néanmoins un calcul supplémentaire pour définir les rangs des solutions.

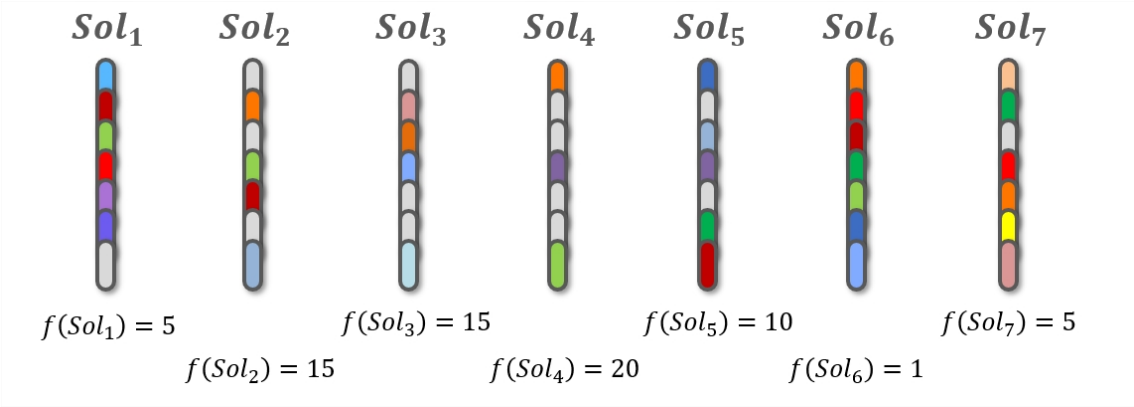
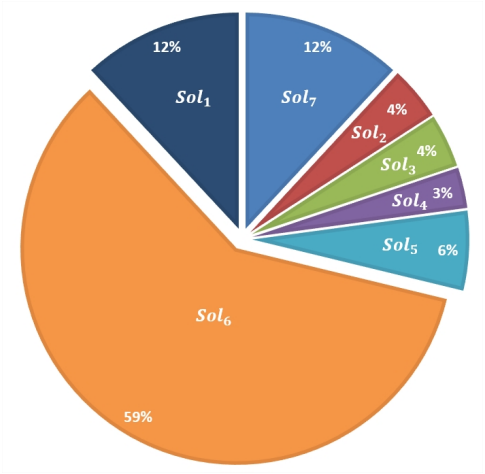


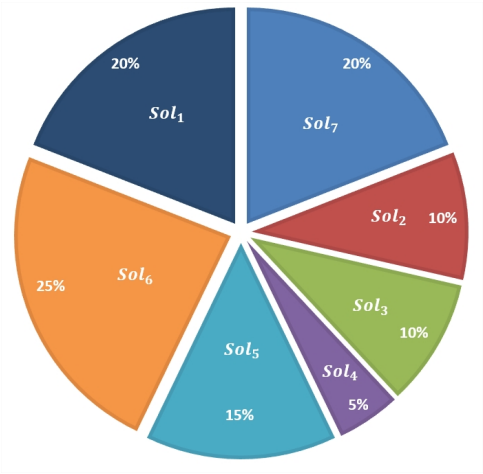
FIGURE 5.12: Une population de 6 solutions Sol_1, Sol_2, ..., Sol_7 et l'évaluation de leur qualité.

Nous prenons à nouveau l'exemple utilisé avec la sélection par tournoi pour illustrer le fonctionnement de la sélection par roulette et la différence entre le calcul par qualité de solution et par rang. Sur la figure 5.12 nous avons la population de solution $P = \{ Sol_1,...,Sol_7 \}$ et la qualité $f(Sol)$ de chacune d'entre elle. Les rangs sont déduits des valeurs de f, et nous avons dans cet exemple 5 valeurs différentes, et donc 5 rangs. Les rangs des solutions sont les suivants : $rank(Sol_1) = 4$,

$rank(Sol_2) = 2$, $rank(Sol_3) = 2$, $rank(Sol_4) = 1$, $rank(Sol_5) = 3$, $rank(Sol_6) = 5$, et $rank(Sol_7) = 4$.



(a) Roulette et probabilités obtenues avec le calcul par rapport à f.



(b) Roulette et probabilités obtenues avec le calcul par rapport au rang des solutions.

FIGURE 5.13: Roulettes obtenues selon les deux méthodes de calcul.

Dans la figure 5.13a, nous représentons les probabilités obtenues pour l'ensemble des solutions de P. En utilisant la formule 5.3, la probabilité de choisir la solution Sol_6, qui est la meilleure, domine largement toutes les autres, et laisse peu de place aux solutions de moins bonne qualité. La figure 5.13b illustre les probabilités obtenues en utilisant l'équation 5.5. Nous av. Ici les probabilités sont mieux réparties entre les différentes solutions et laissent plus de probabilités aux solutions de moins bonne qualité d'être sélectionnées.

Nous comparerons les performances de l'algorithme génétique en utilisant ces différentes méthodes de sélection : la sélection par tournoi, déterministe et non-déterministe, et les sélections par roulette avec le calcul en fonction de la qualité des solutions et en fonction du rang des solutions.

5.1.6 Algorithme général

Dans les sous-sections précédentes, nous avons présenté les différents composants bioinspirés dont nous nous servirons au sein de notre algorithme génétique et le codage des solutions utilisé pour adresser le problème de planification que nous

traitons. Nous suivrons les principes énoncés par Holland et Goldberg dans l'algorithme 15 :

Algorithme 15: geneticAlgorithm()

Données: un ensemble de demandes de rendez-vous A

Résultat: une solution Sol

1 **Début**

2 $P \leftarrow algorithmConstruction(A)$

3 $computeFitness(P)$

4 $Sol_{best} \leftarrow Sol_{best} \cup best(P)$

5 **Tant que** les critères d'arrêt ne sont pas atteints **faire**

6 $P_{parents} \leftarrow selectParents(P)$

7 $P_{children} \leftarrow crossover(P_{parents})$

8 $P_{children} \leftarrow mutation(P_{children})$

9 $computeFitness(P_{children})$

10 $Sol_{best} \leftarrow manageBestSolutions(P_{children})$

11 $P \leftarrow selectSurvivors(P \cup P_{children})$

12 **Fin**

13 Return $bestSol$

14 **Fin**

Initialement, la population P est générée avec l'algorithme de construction 2 décrit dans la section 4.1 du chapitre précédent. Nous calculons ensuite la qualité de toutes ces solutions initiales, et gardons les k meilleures (k étant un paramètre donné par l'utilisateur) dans l'ensemble Sol_{best} renvoyé au terme de la recherche. Le cycle des générations se met ensuite en place, jusqu'à ce que les critères d'arrêt soient atteints, avec la structure suivante : au début de chaque génération, une population d'individus parents $P_{parents}$ est constituée à partir de la population P par une méthode de sélection. Des croisements sont effectués entre les individus de $P_{parents}$ pour produire la nouvelle population $P_{children}$. Elle passera par le processus de mutation, qui vise à modifier certains individus pour obtenir de nouvelles informations utiles à la résolution du problème. Nous calculons alors la qualité des solutions obtenues

dans $P_{children}$. La fonction $manageBestSolution()$ met à jour l'ensemble Sol_{best}, ses plus mauvaises solutions étant remplacées par les meilleures obtenues au fil des générations. Une génération se conclut par la sélection des individus pour la génération suivante, selon la politique de sélection adoptée.

Nous nommerons cette approche $LORH_GA$. Le nombre d'individus dans la population est un des paramètres de l'algorithme, tout comme le type de crossover et de sélection utilisés. Les taux de mutation et de crossover sont également des paramètres à fixer. Le paramétrage et son impact sur les performances de l'algorithme seront abordés dans la section 5.3.1 de ce chapitre.

5.2 Métriques et convergence de l'algorithme

Les algorithmes génétiques sont des méthodes efficaces pour résoudre divers problèmes d'optimisation complexes, dont la recherche peut néanmoins être freinée par un phénomène de convergence prématurée des solutions vers un optimum local. Ce phénomène fut identifié très tôt dans l'exploitation des algorithmes génétiques (Holland, 1992, Squillero and Tonda, 2016) et d'autres méthodes évolutionnaires.

Au fil des générations, et sans contre-mesure pour éviter ce phénomène, les populations de solutions tendent rapidement vers certains individus, jusqu'à devenir presque homogènes. Malheureusement, ces individus sont rarement les solutions optimales au problème, mais plutôt des optimums locaux attrayants dont il sera difficile pour l'algorithme de s'éloigner. Avec une population constituée d'individus très semblables voir identiques, l'opération de croisement n'apporte plus rien à la recherche, tous les individus produits étant eux aussi identiques. L'algorithme ne peut compter dans cette situation que sur la mutation pour apporter les blocs manquants nécessaires pour trouver la solution optimale, un processus potentiellement très long.

Pour mesurer ce phénomène, nous avons besoin de moyens pour évaluer la distance entre deux solutions. Nous pourrons ensuite constater l'ampleur et la rapidité de la convergence des solutions sur notre problème, et développer des contre-mesures le cas échéant.

5.2.1 Métriques

Les quelques métriques que nous proposons sont très largement inspirées de métriques conçues pour mesurer la distance entre deux chaînes de caractères. Elles ont depuis été adaptées à d'autres problèmes et se sont montrées efficace pour estimer la distance entre des solutions aux structures bien plus complexes. Nous parlons ici des distances de Hamming, de Kendall-Tau et de Levenshtein que nous adaptons à notre problème.

5.2.1.1 Distance de Hamming

La distance de Hamming a été introduite par Richard Hamming dans son article sur les codes correcteurs linéaires (Hamming, 1950). D'abord utilisée sur lesdits codes, elle fut beaucoup utilisée pour comparer des chaînes de caractères, puis adaptée à des problèmes tels que le nôtre. Elle consiste à compter pour deux séquences S_1 et S_2 de longueur identique le nombre δ de différences à chaque position i telles que $S_1^i \neq S_2^i$. δ est ensuite divisé par $|S_1|$. Cela peut être également décrit comme le nombre de substitutions pour passer d'une séquence à l'autre.

Dans notre cas, ce sont les plannings obtenus avec l'algorithme génétique que nous souhaitons comparer. Ces planifications sont représentées sous forme de séquences de gènes (a,t_a,R_a) représentant les affectations de dates et de ressources à l'ensemble des demandes de rendez-vous $a \in A$, avec un ordre identique d'une solution à l'autre. Nous pouvons donc aisément adapter la distance de Hamming à cette situation, en comptant le nombre d'affectations (de date ou de ressources) différentes pour une même demande a dans les deux solutions comparées. Nous voulions cependant faire une mesure plus fine des différences au niveau des affectations, et pouvoir mesurer un taux de différence entre deux gènes.

Le calcul de la distance entre deux individus est détaillé dans l'algorithme 16. Pour deux solutions respectant les contraintes du problème Sol^1 et Sol^2, nous comparons un à un les gènes pour tout rendez-vous $a \in A$. La distance entre deux individus que nous voulons obtenir est un réel $\Delta \in [0,1]$ représentant la moyenne des distances entre chaque gène dans les deux solutions. En premier lieu (lignes 6 à 9) nous vérifions les affectations (a,t_a^1,R_a^1) et (a,t_a^2,R_a^2) dans les deux solutions : si un rendez-vous a est planifié dans une solution et pas dans l'autre, nous considérons

Algorithme 16: HammingDistance()

Données: une solution $Sol_1 = \bigcup (a,t_a,R_a)$, une solution $Sol_2 = \bigcup (a,t_a,R_a)$

Résultat: un réel Δ représentant la distance entre Sol_1 et Sol_2

```
1  Début
2      Δ ← 0
3      cst ← ln(100 / (e−1))
4      Pour chaque a ∈ A faire
5          (a,t_a^1,R_a^1) ∈ Sol_1 ; (a,t_a^2,R_a^2) ∈ Sol_2
6          Si R_a^1 = ∅ or R_a^2 = ∅ alors
7              Si R_a^1 ≠ R_a^2 alors
8                  Δ ← Δ+1
9              Fin
10         Sinon
11             δ_ressource ← 0 ; δ_temps ← 0
12             Si | R_a^1 | > | PreAssigned_a | alors
13                 Pour i ∈ | R_a^1 | faire
14                     Si R_a^1[i] ∉ R_a^2 alors
15                         δ_ressource ← δ_ressource+1
16                     Fin
17                 Fin
18                 δ_ressource ← δ_ressource / (| R_a^1 |−|PreAssigned_a|)
19             Fin
20             d ← |t_a^1−t_a^2| / (LS_a−ES_a)
21             δ_temps ← (e^(d+cst)−e^cst)/100
22             Δ ← Δ+(δ_ressource+δ_temps) / 2
23         Fin
24     Fin
25     Retourne Δ / |A|
26 Fin
```

que la distance entre les solutions sur ce gène est maximale, donc égale à 1. S'il n'y a aucune affectation dans les deux solutions, les gènes sont identiques, la distance pour ce gène vaut 0. Lorsque le rendez-vous a est planifié dans les deux solutions, nous calculons deux scores de différence : $\delta_{ressource}$ et δ_{temps} représentant respectivement la distance entre les affectations de ressources et la distance entre les affectations de date des individus.

La gestion des ressources se fait des lignes 12 à 19. Nous vérifions qu'il y a plus de ressources dans R_a^1 que dans $PreAssigned_a$. Si $|R_a^1| < |PreAssigned_a|$, alors toutes les ressources pré-affectées ne sont pas présentes dans R_a^1, et la contrainte **CD 8** (3.1.2) exigeant que toutes les pré-affectations soient respectées serait enfreinte. Une telle situation ne peut se produire dans une solution valide. Si $|PreAssigned_a| = |R_a|$, toutes les ressources $r \in R_a$ affectées à a sont des ressources pré-assignées. Toujours selon la contrainte **CD 8**, il ne peut donc y avoir de différences entre les affectations R_a^1 et R_a^2, par conséquent $\delta_{ressource} = 0$. S'il y a plus de ressources dans R_a^1 que dans $PreAssigned_a$, nous vérifions pour chaque ressource $r \in R_a^1$ si elle est présente dans R_a^2, et incrémentons le score de différence $\delta_{ressource}$ si ce n'est pas le cas. Ce score est ensuite divisé par le nombre de ressources affectées au rendez-vous qui ne font pas partie des ressources pré-affectées ($|R_a^1| - |PreAssigned_a|$).

Le score de différence des dates δ_{temps} est calculé en fonction des dates de début attribuées t_a^1 et t_a^2 selon les formules suivantes :

$$d = \frac{|t_a^1 - t_a^2|}{LS_a - ES_a} \tag{5.6}$$

Dans la formule 5.6 nous calculons d, le ratio entre la différence des dates de début t_a^1 et t_a^2 des deux solutions et l'amplitude de l'intervalle de faisabilité du rendez-vous.

$$\delta_{temps} = (e^{d+cst} - e^{cst})/100 \tag{5.7}$$

L'équation 5.7 renvoie un réel compris entre 0 et 1 augmentant de manière exponentielle en fonction de d. La différence des affectations pour un rendez-vous a correspondra à la moyenne des deux scores $\delta_{ressource}$ et δ_{temps}. Ces calculs sont réalisés pour chaque demande de rendez-vous $a \in A$, et la distance entre les individus Sol_1 et Sol_2 sera égale à la moyenne des distances calculées pour chaque rendez-vous.

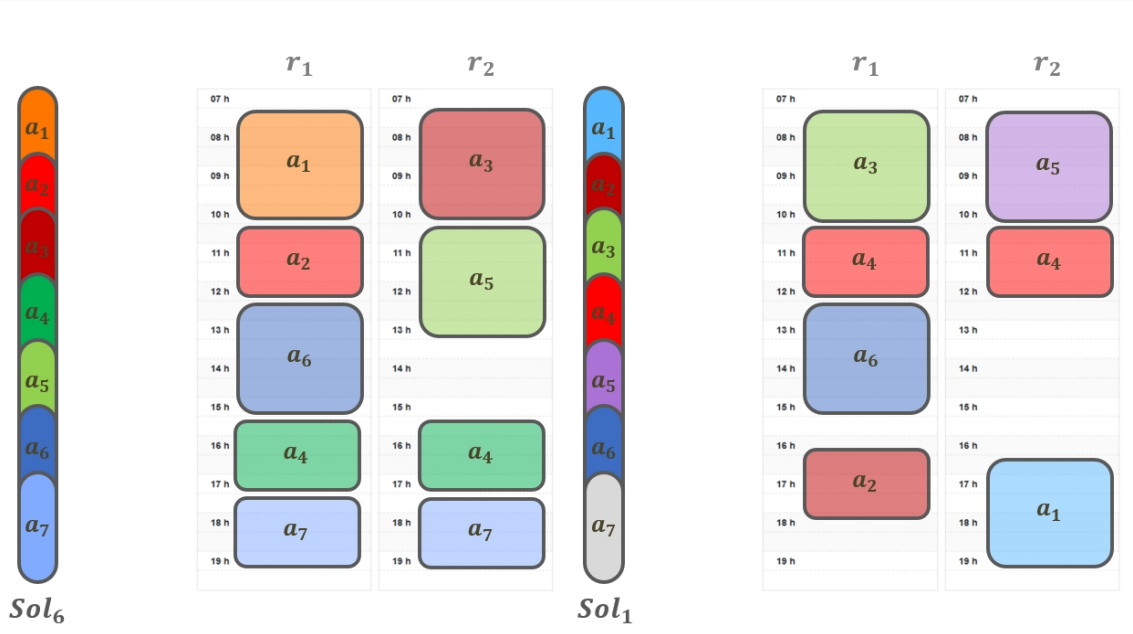


FIGURE 5.14: Deux solutions Sol_1 et Sol_6 et les plannings des ressources r_1 et r_2.

Afin d'illustrer cette métrique avec un exemple, nous utilisons deux solutions déjà aperçues dans nos explications sur les méthodes de sélection. Dans la figure 5.14, les solutions Sol_1 et Sol_6 sont représentées ainsi que les emplois du temps des ressources r_1 et r_2 impliquées. Les rendez-vous a_1,...,a_7 de cet exemple nécessitent une propriété π_1 possédée par r_1 et r_2. Si nous suivons l'algorithme 16, nous voyons qu'une distance est calculée entre les affectations (a,t_a^1,R_a^1) et (a,t_a^6,R_a^6) de chaque rendez-vous $a \in A$, dans les deux solutions Sol_1 et Sol_6. Dans cet exemple, nous nous concentrons sur les affectations de a_1 dans les deux solutions.

Sur la figure 5.15 nous avons mis en valeur les affectations dans les deux solutions pour la demande de rendez-vous a_1 dans les deux solutions Sol_1 et Sol_6. Notre distance calcule deux scores de différence : $\delta_{ressource}$ pour les ressources, et δ_{temps} pour la date de début. Dans cet exemple, nous voyons que r_1 est affectée à a_1 dans la solution Sol_6 tandis que c'est la ressource r_2 qui y est affectée dans la solution Sol_1. Il n'y a pas ici de pré-affectation, une seule ressource est requise pour ce rendez-vous, nous avons donc une différence $\delta_{ressource} = 1$, au maximum de ce qu'elle peut être. Pour la date de début, nous avons $t_{a_1}^6 = 0$ et $t_{a_1}^1 = 9$. Toujours en suivant l'algorithme, nous avons $d = \frac{|t_{a_1}^6 - t_{a_1}^1|}{LS_a - ES_a}$, donc ici $d = \frac{|0-9|}{12-0} = 0.75$. Selon la formule 5.7, δ_{temps} est ici égal à 0.65. Pour le rendez-vous a_1, la distance entre les deux

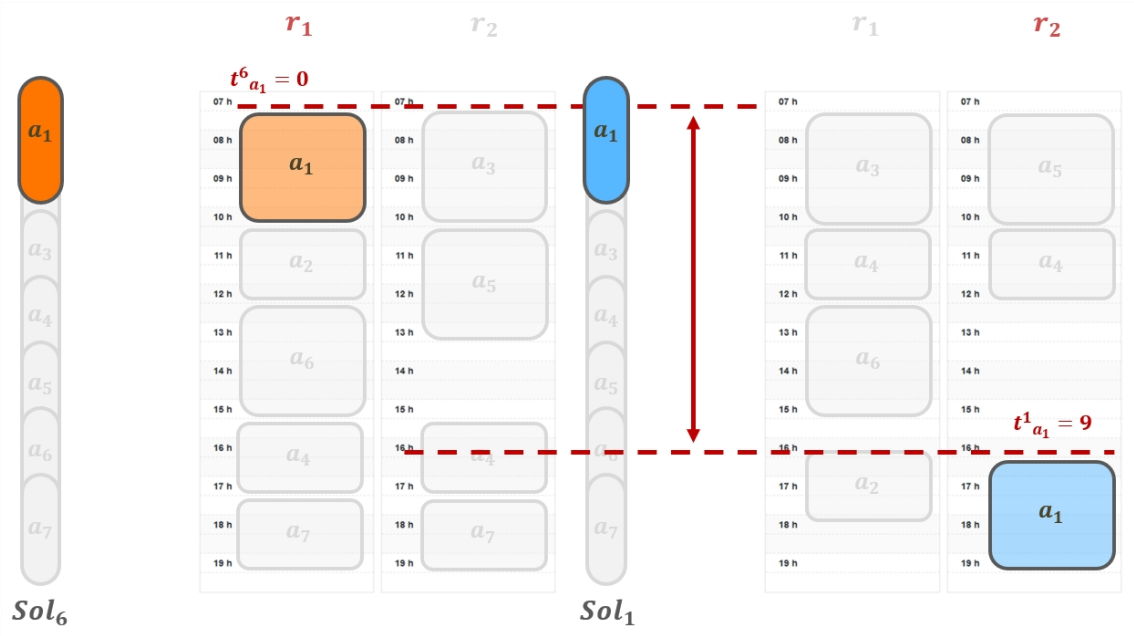


FIGURE 5.15: Comparaison des affectations de a_1 dans les deux solutions Sol_6 et Sol_1.

solutions est égale à $\frac{1+0.65}{2} = 0.825$. La distance totale entre les deux individus correspond à la moyenne de ces distances pour tous les rendez-vous $a \in A$.

5.2.1.2 Distance de Kendall-Tau

La distance de Kendall-Tau est une mesure de la corrélation des rangs au sein de deux séquences d'éléments décrite par Maurice G. Kendall (Kendall, 1938). Elle consiste à comparer dans deux séquences S_1 et S_2 l'ordre relatif des éléments i et j, et peut être définie par la formule suivante :

$$K_d(S_1,S_2) = \sum_{\{i,j\} \in Pa, i<j} \overline{K}_i j(S_1,S_2) \tag{5.8}$$

où Pa est l'ensemble non-ordonné des paires d'éléments dans S_1 et S_2. $\overline{K}_i j(S_1,S_2) = 0$ si i et j sont dans le même ordre dans les deux séquences, et $\overline{K}_i j(S_1,S_2) = 1$ sinon. Cette distance est adaptable à notre problème et au codage de nos solutions afin d'avoir de nouvelles informations, notamment sur l'ordre relatif des rendez-vous dans les deux solutions comparées.

Pour pouvoir utiliser cette métrique, il nous faut attribuer à chaque rendez-vous un rang, que nous pourrons comparer d'une solution à l'autre pour établir un score

symbolisant la distance les séparant. Pour ordonner les rendez-vous, nous utilisons leur date de début t_a, et nous les classons par ordre chronologique. Les rendez-vous débutant à la même date t_a auront le même rang. Les rendez-vous qui ne sont pas planifiés se verront attribuer le rang -1 ($\forall a \in A_{\overline{Sol}}$, $rank(a) = -1$), et ils pourront également être comparés aux rendez-vous de l'autre solution.

Algorithme 17: KendallTauDistance()

Données: une solution $Sol_1 = \bigcup (a,t_a,R_a)$, une solution $Sol_2 = \bigcup (a,t_a,R_a)$

Résultat: un réel Δ estimant la distance entre Sol_1 et Sol_2

```
1  Début
2      Δ ← 0
3      Pour i allant de 1 à |A| faire
4          Pour j allant de i+1 à |A| faire
5              Si rank(a_i ∈ sol_1) ≥ rank(a_j ∈ sol_1) alors
6                  Si rank(a_i ∈ sol_2) < rank(a_j ∈ sol_2) alors
7                      Δ ← Δ+1
8                  Fin
9              Sinon
10                 Si rank(a_i ∈ sol_2) ≥ rank(a_j ∈ sol_2) alors
11                     Δ ← Δ+1
12                 Fin
13             Fin
14         Fin
15     Fin
16     Retourne Δ / (|A|×(|A|−1))/2
17 Fin
```

L'adaptation de la distance de Kendall-Tau à nos solutions est décrite dans l'algorithme 17. Les rangs sont attribués et maintenus au fil des opérations sur les solutions, la fonction $rank(a)$ est appelée pour obtenir le rang d'un rendez-vous a. L'ordre des rendez-vous est comparé pour toutes les paires distinctes de rendez-vous (a_i,a_y) dans les deux solutions Sol_1 et Sol_2. Nous regardons l'ordre de a_i par rapport à a_j dans Sol_1. Si cet ordre n'est pas préservé dans Sol_2, nous incrémentons Δ de

1. Chaque paire de rendez-vous est ainsi vérifiée et la distance Δ est divisée par le nombre de couples de rendez-vous $\frac{|A|\times(|A|-1)}{2}$. La distance renvoyée est, à l'instar de la métrique précédente, un réel $\Delta \in [0,1]$.

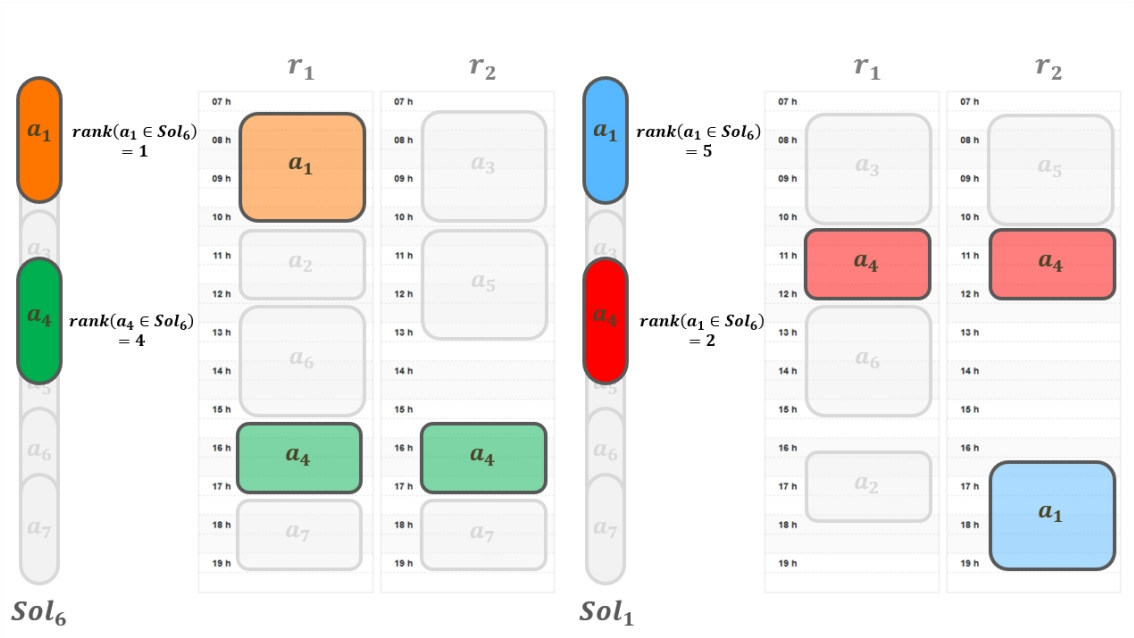


FIGURE 5.16: Comparaison de l'ordre relatif des rendez-vous a_1 et a_4 dans les solutions Sol_1 et Sol_6.

Pour illustrer ce calcul avec la distance de Kendall-Tau, nous utilisons à nouveau l'exemple présenté pour la distance de Hamming. Dans cet exemple, nous comparons deux solutions Sol_1 et Sol_6 afin d'estimer la distance qui les sépare. La métrique de Kendall-Tau se concentre sur la notion d'ordre relatif des rendez-vous 2 à 2 dans les deux solutions. La figure 5.16 met en avant les rendez-vous a_1 et a_4, et plus précisément leur rang. Nous voyons que dans la solution Sol_6, a_1 débute avant a_4, $rank(a_1 \in Sol_6) = 1$ et $rank(a_4 \in Sol_6) = 4$. Dans la solution Sol_1, c'est l'inverse, et le rendez-vous a_1 début après a_4, $rank(a_4 \in Sol_1) = 2$ et $rank(a_4 \in Sol_1) = 5$. L'ordre n'est pas le même dans les deux solutions, et la distance Δ est incrémentée de 1.

5.2.2 Calcul de diversité

Avec les métriques décrites précédemment, nous sommes en mesure d'estimer la distance entre deux individus au sein de la population de l'algorithme génétique.

Pour une population P, nous notons $\Gamma_P = \frac{|P|\times(|P|-1)}{2}$ le nombre de couples distincts d'individus au sein de la population. Nous définissons la diversité d'une population selon la formule suivante :

$$Div(P) = \frac{\sum_{\forall(Sol_i,Sol_j)\in Pa} D(Sol_i,Sol_j)}{|Pa|} \tag{5.9}$$

avec $D(Sol_i,Sol_j)$ la distance entre deux solutions Sol_i et Sol_j et Pa l'ensemble des couples distincts de solutions de P. Pour calculer D, nous utilisons l'une des métriques présentées dans la sous-section 5.2.1. Nous calculons pour tous les couples distincts d'individus d'une population la distance D qui les sépare, qui est ensuite divisée par le nombre total de couple Γ_P. Un score faible indique que beaucoup de solutions au sein de la population sont similaires et par conséquent les échanges d'informations sont moins fructueux lors des phases de reproduction. Une diversité trop élevée est également nocive au fonctionnement de l'algorithme, qui ne parvient plus à converger vers de bonnes solutions. L'indicateur de diversité nous permettrait de mieux contrôler ce phénomène de convergence afin d'améliorer les performances de l'algorithme génétique.

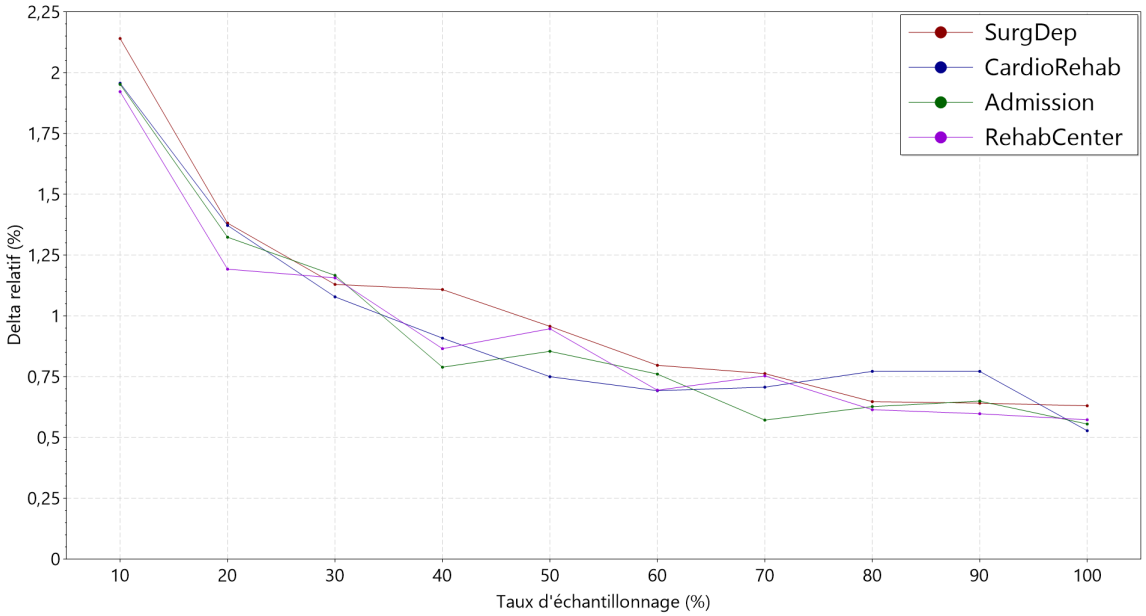
Le calcul de diversité peut s'avérer coûteux, surtout s'il est récurent dans l'algorithme génétique. En calculant l'ensemble des distances entre tous les couples d'individus d'une population (soit $\Gamma_P = (|P|\times|P|-1)/2$ distances pour une population P), nous obtenons la diversité la plus précise possible. Afin de diminuer le temps de calcul, nous voulons savoir s'il est possible de réaliser un échantillonnage des individus pour le calcul de métrique tout en conservant un résultat satisfaisant par rapport au calcul complet. Ce calcul échantillonné, que nous nommerons $Div_s(P)$, ne réalise pas toutes les comparaisons entre les paires distinctes de l'ensemble Pa, mais un nombre $\Gamma_P\times x$ de comparaisons effectuées entre deux individus Sol_i et Sol_j sélectionnés aléatoirement dans les deux populations. x est ici le taux d'échantillonnage, et nous voulons déterminer celui offrant le meilleur rapport coût de calcul / précision.

5.2.2.1 Échantillonnage

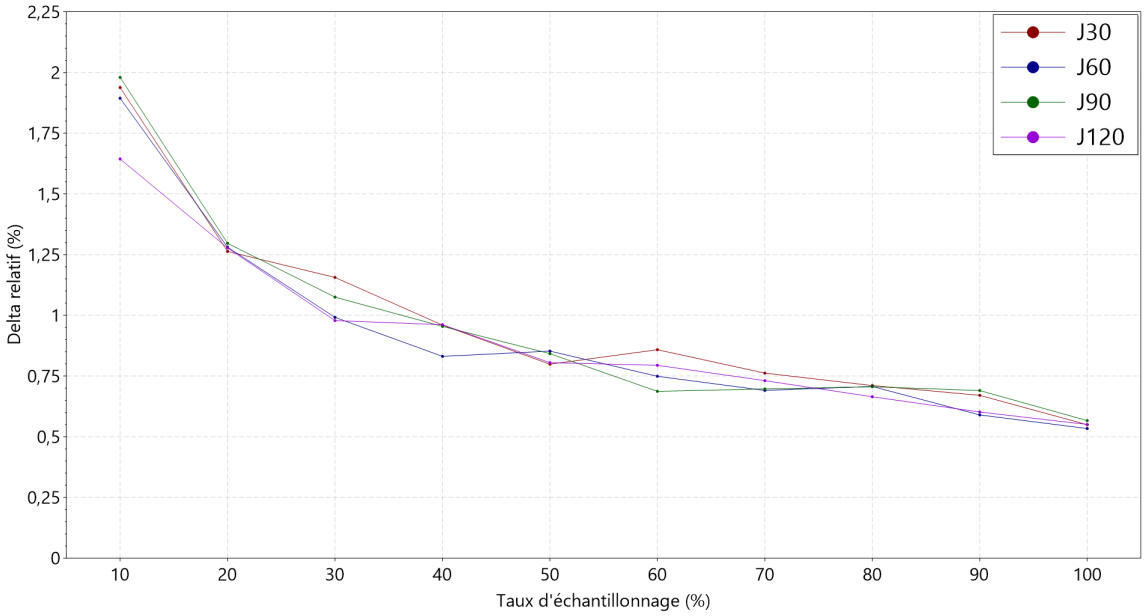
Le protocole de test pour déterminer le taux d'échantillonnage est le suivant : un individu est généré avec l'algorithme de construction, puis dupliqué afin d'ob-

tenir une population de 100 clones. Une portion de cette population subit ensuite une phase de mutation avec un taux plus ou moins important pour obtenir une population plus ou moins hétérogène : plus le taux de mutation est élevée, plus les individus seront distants les uns des autres. Pour chaque population obtenue, nous réalisons plusieurs calculs : le calcul complet défini par l'équation 5.9 avec tous les couples d'individus, et plusieurs calculs avec des taux d'échantillonnage différents. Avec une population P de 100 individus, nous avons un nombre total de couples $\Gamma_P = 4950$. Nous faisons varier le taux de mutation τ entre 0.1 et 1, avec un pas de 0.1. Le taux d'individus à subir ces mutations, que nous notons θ, varie également entre 0.1 et 1 avec un pas de 0.1. Nous faisons varier dans un premier temps le taux d'échantillonnage entre 10% et 100% de Γ_P et reportons la moyenne des résultats obtenus sur plusieurs instances de chaque famille.

Dans les graphes 5.17a et 5.17b nous reportons la moyenne des écarts relatifs ($|(Div(P) - Div_s(P)/Div(P)|$) entre la valeur absolue obtenue par calcul complet et par échantillonnage, pour tous les taux d'échantillonnage entre 10% et 100% pour toutes nos familles d'instances. Dans le graphe 5.17a nous reportons les moyennes obtenues pour les instances ***SurgDep***, ***RehabCenter***, ***Admission*** et ***CardioRehab***. Dans le graphe 5.17b nous donnons les moyennes obtenues sur les quatre familles d'instances PSPLIB ***J30***, ***J60***, ***90*** et ***J120***. Dans les deux graphes nous constatons une tendance similaire avec un écart relatif aux environs de 2% avec un taux d'échantillonnage de 10% qui baisse progressivement à mesure que le taux d'échantillonnage augmente. Les valeurs des écarts sont très basses, même au taux le plus bas de ces graphes, et se traduisent par une différence faible de 0.4 à 0.5 pour les instances de l'entreprise et aux alentours de 1 pour les instances PSPLIB. Le calcul échantillonné $Div_s(P)$ n'est cependant jamais égal à $Div(P)$ (même avec un taux d'échantillonnage à 100%), car le tirage des solutions Sol_1 et Sol_2 comparées est aléatoire. L'ensemble des couples testés par les deux calcul n'est donc pas identique. Nous avons mené d'autres expérimentations en suivant le même protocole avec des taux d'échantillonnage plus bas, entre 0.1 et 10%. Nous reportons dans les graphes 5.18a et 5.18b les moyennes des écarts relatifs obtenues entre ces taux pour nos instances et les instances PSPLIB. Sur ces résultats, nous observons que les courbes chutent rapidement de 0.1 à 3%, avant de se stabiliser après les 3% d'échantillonnage. Les valeurs sont plus hautes que dans les graphes précédents, avec un écart



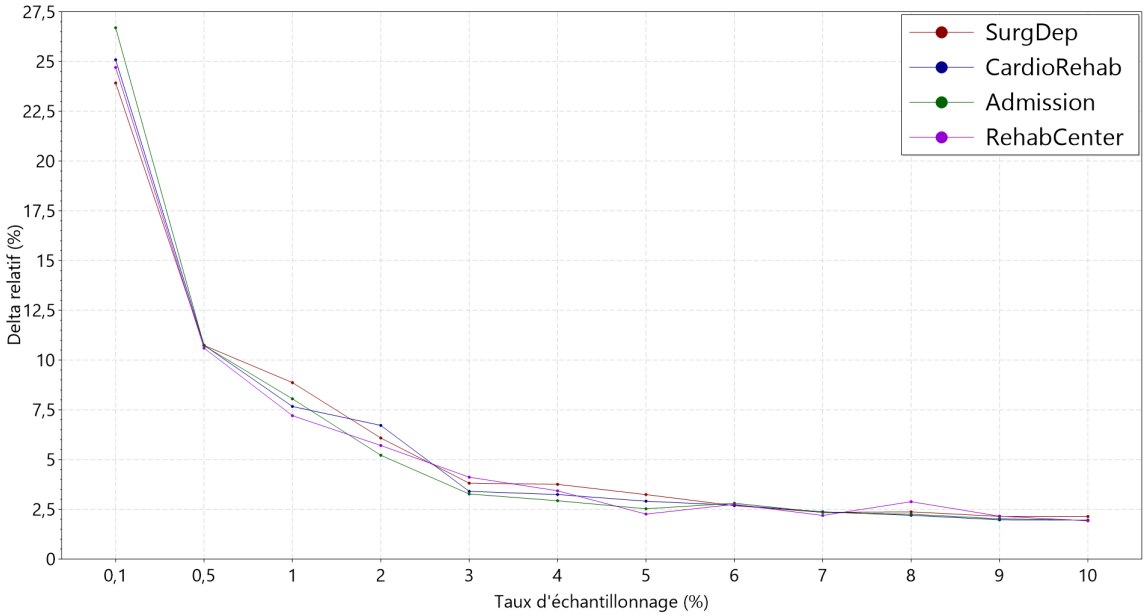
(a) Écarts relatifs moyens sur les instances de l'entreprise entre $Div(P)$ et $Div_s(P)$.



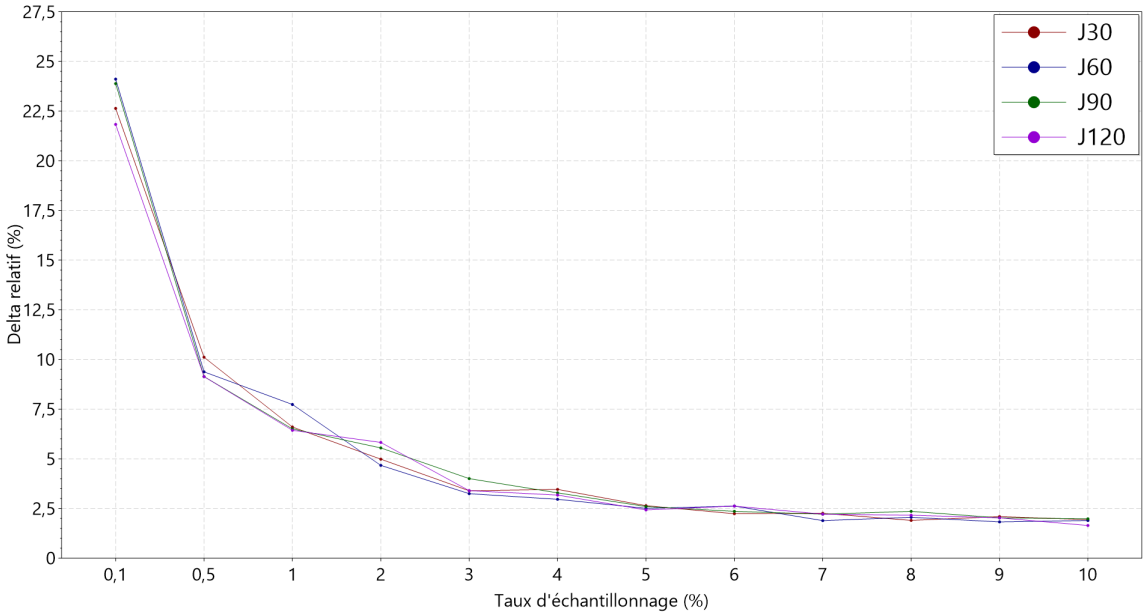
(b) Écarts relatifs moyens sur les instances PSPLIB entre $Div(P)$ et $Div_s(P)$.

FIGURE 5.17: Écarts relatifs moyens entre $Div(P)$ et $Div_s(P)$, avec un taux d'échantillonnage allant de 10 à 100%.

relatif montant jusqu'à 27% pour les instances PSPLIB, et proche de 22.5% sur nos instances. Ces écarts représentent des différences importantes de quelques points, entre 5 et 6 sur nos instances et jusqu'à 12 pour les instances PSPLIB, entre la valeur retournée par le calcul complet et l'échantillonnage.



(a) Écarts relatifs moyens sur les instances de l'entreprise entre $Div(P)$ et $Div_s(P)$.



(b) Écarts relatifs moyens sur les instances PSPLIB entre $Div(P)$ et $Div_s(P)$.

FIGURE 5.18: Écarts relatifs moyens entre $Div(P)$ et $Div_s(P)$, avec un taux d'échantillonnage allant de 0.1 à 10%.

Ces tests sur les différents taux d'échantillonnage nous permettent de constater qu'un taux d'échantillonnage très bas, inférieur à 10% permet toujours d'obtenir des valeurs de diversité proches du calcul complet entre tous les couples d'individus d'une population et donc pertinentes. En dessous de 10%, le taux d'échantillonnage de 3% semble donc être un bon compromis entre le coût de calcul et la précision

obtenue avec ce taux. L'écart relatif à ce taux varie entre 3 et 4% selon les familles d'instances, et représente une différence de 0.01 ou 0.02 entre le résultat du calcul complet et échantillonné. Pour nos futures utilisations des métriques pour calculer la diversité d'une population, nous utiliserons donc ce taux d'échantillonnage de 3%.

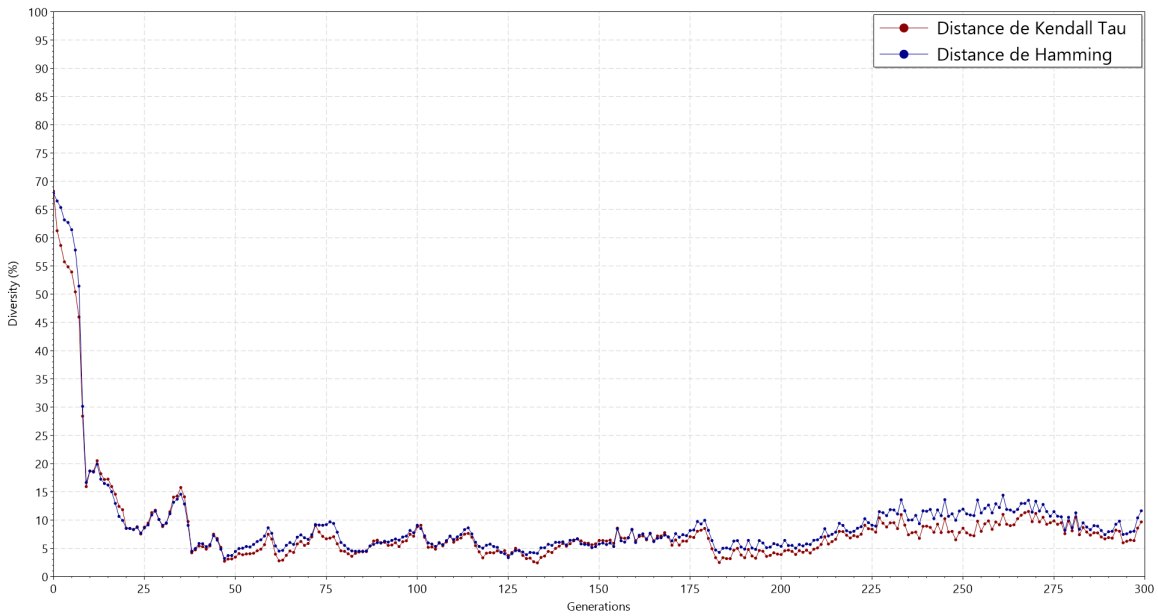
5.2.2.2 Comparaison des métriques

Nous avons déterminé le taux d'échantillonnage garantissant le meilleur rapport entre temps investi dans le calcul et justesse de la diversité obtenue. Nous pouvons désormais les utiliser toutes les deux au sein de l'algorithme génétique pour connaître l'évolution de la diversité des populations au fil de son exécution.

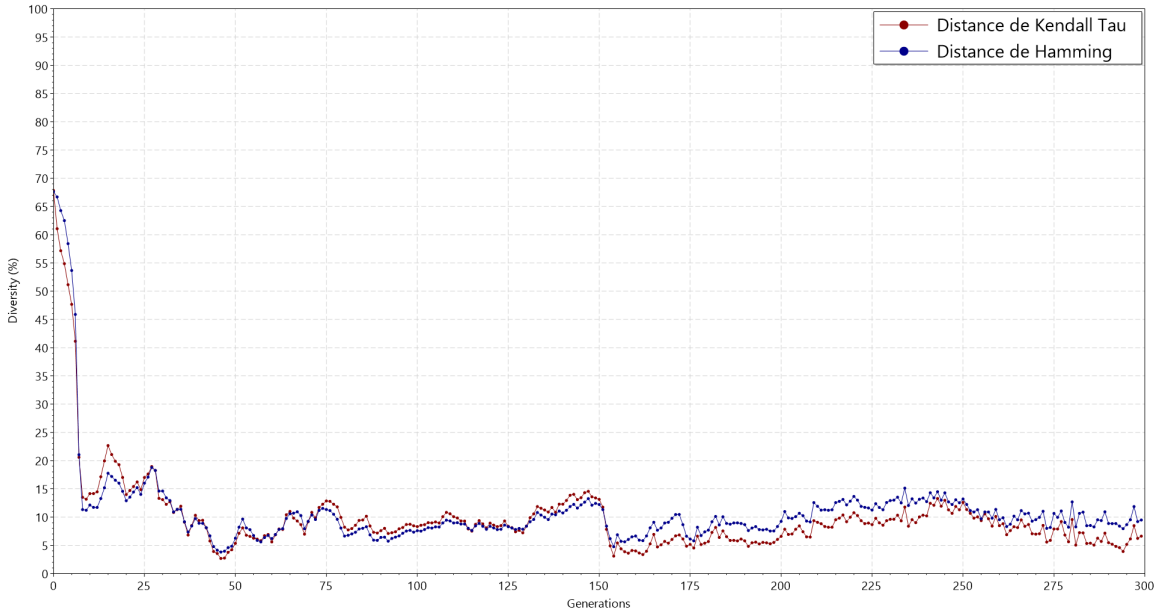
Pour cela, nous réaliserons des mesures à la fin de chaque cycle évolutif, après la phase de mutation de la population d'individus enfants. Ces mesures seront faites pendant 300 générations sur quelques instances de chaque scénario dont nous disposons pour que nous puissions observer le phénomène de convergence des solutions et comparer les deux métriques que nous avons. Ces mesures seront faites sur population de 100 individus.

Les graphes 5.19 représentent l'évolution de la diversité au fil des générations pour le scénario ***SurgDep***. Sur ces deux exemples, la diversité chute rapidement après une dizaine de générations, puis diminue progressivement jusqu'à la génération 50 avant de fluctuer autour d'une diversité de 10%. Nous observons des résultats similaires pour les deux métriques, qui suivent les mêmes tendances aux mêmes moments tout au long de l'exécution de l'algorithme. Nous constatons l'homogénéisation rapide de la population, et une diversité qui reste faible (de l'ordre de 10%) jusqu'à l'arrêt de l'algorithme.

Le graphique 5.20 représente l'évolution de la diversité pour une instance de la famille ***CardioRehab***. À nouveau, la diversité chute brutalement en 25 générations et reste très faible (5%) pendant les 300 générations. Les métriques de Kendall Tau et de Hamming donnent sur cet exemple aussi des courbes très proches. C'est également vrai sur la plupart des scénarios et la plupart des instances. Mêmes si les valeurs ne sont pas systématiquement identiques, les deux métriques indiquent se comportent de manière identique aux mêmes générations.



(a) Calcul des distances de Hamming et de Kendall Tau sur l'instance ***SurgDep*** **T=16 I=0 U=0 D=0**



(b) Calcul des distances de Hamming et de Kendall Tau sur l'instance ***SurgDep*** **T=48 I=24 U=12 D=12**.

FIGURE 5.19: Scores de diversité renvoyés par les métriques de Hamming et de Kendall Tau sur deux instances de la famille ***SurgDep***.

Une seule famille fait exception : ***RehabCenter***. Nous donnons un exemple dans le graphique 5.21. Nous pouvons voir que les scores de diversité prodigués par la métrique de Hamming et Kendall Tau divergent plus que sur les autres scénarios après la chute initiale, avec des valeurs plus élevées renvoyées par la distance de Hamming.

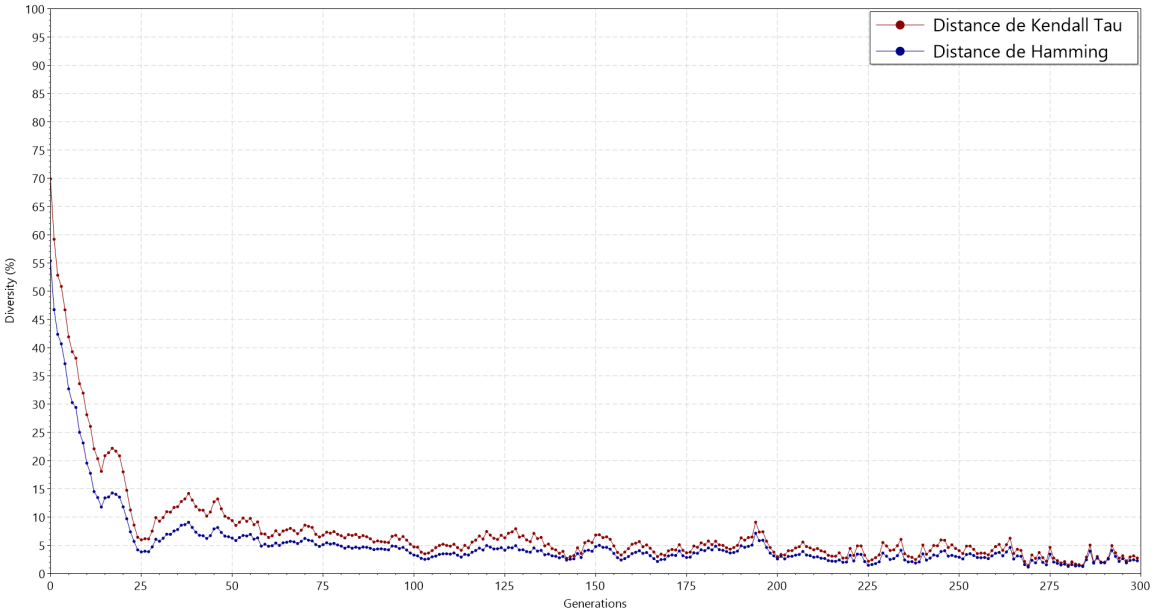


FIGURE 5.20: ***CardioRehab*** **T=160 I=160 U=80 P=36 D=38**.

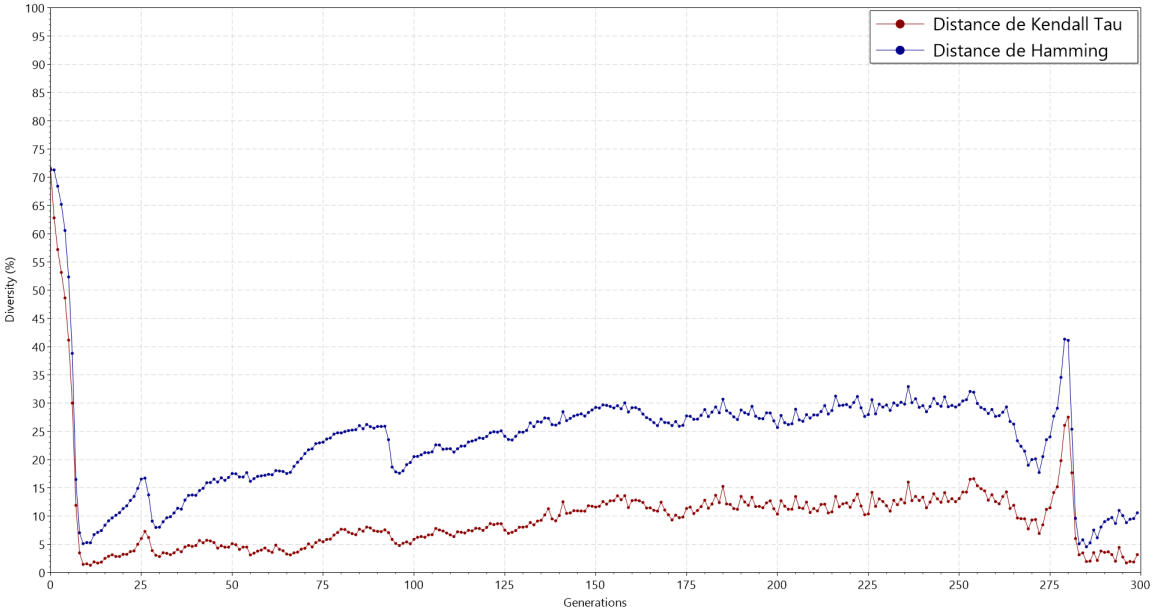


FIGURE 5.21: ***RehabCenter*** **T=96 I=96 U=48 P=15**.

Les courbes ont toujours des tracés semblables : aux générations 25, 90 et 280, les deux métriques indiquent une baisse rapide de la diversité. Cette particularité se retrouve sur toutes les instances de la famille ***RehabCenter***, avec des retours plus élevés pour la métrique de Hamming que pour la métrique de Kendall Tau. Les deux métriques mesurent des différences au niveau des dates de début des rendez-vous : la distance de Hamming en comparant directement les créneaux et la distance de Kendall Tau en comparant les ordres des rendez-vous deux à deux dans chaque solu-

tion. La distance de Hamming, à l'inverse de Kendall Tau, prend aussi en compte les différences d'affectation de ressources dans les deux solutions. En observant de plus près cette différence $\delta_{ressource}$, nous avons remarqué qu'elle était assez élevée pour influer sur la distance renvoyée par la métrique de Hamming. Les affectations de ressources semblent être, sur cette famille d'instance, plus diversifiées d'une solution à l'autre, et les scores de la métrique de Hamming sont par conséquent plus élevés que la distance de Kendall Tau, qui ne prend pas en compte les ressources dans son calcul. Néanmoins on peut constater que $\delta_{ressource}$ n'a pas beaucoup d'influence sur la diversité puisque la distance de Kendall Tau et de Hamming varient à l'identique.

5.2.3 Algorithme génétique avec calcul de distance

Les métriques présentées dans les parties précédentes nous permettent de mesurer efficacement l'évolution de nos populations au fil des générations et nous avons pu constater une baisse de la diversité rapide après le lancement de la recherche. Nous voulons développer des méthodes nous permettant de mieux contrôler ce phénomène, et pouvant réinjecter de potentiels blocs d'informations intéressants lorsque la recherche ne progresse plus. Pour parvenir à ce résultat, nous modifions l'algorithme génétique présenté précédemment : une étape de gestion de la population est ajoutée à la fin d'une génération. Elle ajustera le taux de mutation selon la diversité et la stagnation de la recherche. Nous proposons aussi une nouvelle méthode de sélection privilégiant des individus hétérogènes et retardant ainsi une convergence des solutions trop prononcée.

Nous reprenons l'algorithme 15 en intégrant la gestion de la diversité de la population à travers les fonctions $selectSurvivorsDiversity()$ et $managePopulationDiversity()$ pour obtenir l'algorithme 18. Cet algorithme génétique sera nommé $LORH_GADM$ dans le reste de ce mémoire.

5.2.3.1 Méthode de gestion de la population : *managePopulationDiversity()*

Notre objectif avec cette nouvelle étape dans l'algorithme génétique est de corriger une convergence trop prononcée de la population vers un optimum local. Si celle-ci peut permettre d'explorer plus intensément une région de l'espace de recherche prometteuse et ainsi trouver une solution de bonne qualité, elle est néanmoins nocive

Algorithme 18: geneticAlgorithmDiversity()

Données: un ensemble de demandes de rendez-vous A

Résultat: une solution Sol

1 **Début**

2 $P \leftarrow algorithmConstruction(A)$

3 $computeFitness(P)$

4 $Sol_{best} \leftarrow Sol_{best} \cup best(P)$

5 $\lambda \leftarrow 0$; $nbGeneration \leftarrow 0$

6 $lowDiversity \leftarrow faux$

7 **Tant que** les critères d'arrêt ne sont pas atteints **faire**

8 $P_{parents} \leftarrow selectParents(P)$

9 $P_{children} \leftarrow crossover(P_{parents})$

10 $P_{children} \leftarrow mutation(P_{children})$

11 $computeFitness(P_{children})$

12 $Sol_{best} \leftarrow manageBestSolution(P_{children},\lambda)$

13 $P \leftarrow selectSurvivorsDiversity(P_{parents},P_{children})$

14 $managePopulationDiversity(P,\delta_{low},\lambda,nbGeneration)$

15 **Fin**

16 Retourne Sol_{best}

17 **Fin**

si elle se prolonge trop longtemps. La difficulté pour cette gestion de la population est donc de trouver un équilibre entre diversité trop élevée et trop basse, les deux situations étant problématiques pour l'efficacité de l'algorithme.

Plusieurs méthodes ont été envisagées pour mieux contrôler cette diversité. Les premières tentatives consistaient à adapter le taux de mutation selon les variations de diversité : si la diversité variait fortement d'une génération à l'autre, nous faisions varier le taux de mutation, pour diminuer ou augmenter la diversité de la population. Dans une autre tentative de maîtrise de la diversité, nous augmentions le taux de mutation lorsque la diversité de la population chutait sous un certain seuil. Ces deux approches n'ont pas donné de résultats satisfaisant, et nous avons identifié

deux raisons à cela : les variations trop progressives ne permettaient pas de sortir des optimums locaux, et des périodes de convergence sont nécessaires pour obtenir de bonnes solutions.

Algorithme 19: managePopulationDiversity()

Données: une population d'individus P, un booléen δ_{low}, un booléen τ_{div}, un entier λ et un entier $nbGenerations$

Résultat: un ajustement du taux de mutation en fonction de la diversité

```
1  Début
2      Si τ_div = vrai alors
3          Si nbGeneration ≥ κ_end alors
4              τ_div ← faux ; δ_low ← faux
5              decreaseMutationRate(τ)
6          Fin
7      Fin
8      Si Div_s(P) < Div_min alors
9          δ_low ← vrai
10     Sinon
11         δ_low ← faux
12     Fin
13     Si δ_low = vrai alors
14         Si λ ≥ ν alors
15             τ_div ← vrai
16             κ_end ← nbGenerations+κ
17             increaseMutationRate(τ_high)
18         Fin
19     Fin
20 Fin
```

À partir de ces constats, nous avons élaboré une autre méthode pour gérer la convergence, prenant en compte à la fois la diversité de la population et la progression de la recherche, décrite dans l'algorithme 19. Notre idée est de déclencher de courtes

périodes de très fortes mutations lorsque la recherche stagne et que la diversité n'est pas bonne. Nous définissons plusieurs paramètres :

— Div_{min} un réel [0;1] définissant un seuil minimum de diversité ;

— ν un entier représentant le délai entre le passage sous le seuil Div_{min} et le passage en phase de diversification ;

— κ un entier définissant la durée de la phase de diversification. La variable κ_{end} est la génération d'arrêt de cette phase ;

— τ_{high} un réel [0;1] fixant le taux de mutation pendant la phase de diversification.

Si la diversité $Div_s(P)$ calculée avec l'une des métriques chute sous le seuil Div_{min}, et qu'aucune amélioration de la meilleure solution Sol_{best} n'a eu lieu depuis ν générations, l'algorithme passe en phase de diversification. Durant cette phase d'une durée de κ générations, le taux de mutation est fixé à τ_{high} afin de faire remonter la diversité de la population. Le délai ν nous permet de retarder le déclenchement de ces mesures tant que Sol_{best} est régulièrement améliorée, car nous nous voulons pas perturber l'exploitation d'une région prometteuse de l'espace de recherche si que l'algorithme génétique parvient à en tirer de bonnes solutions. C'est la principale différence avec nos tentatives précédentes, qui déclenchaient les phases de diversification sans prendre en compte les améliorations de Sol_{best}. Les phases d'exploitation étaient perturbées et les résultats obtenus étaient inférieurs aux résultats de l'algorithme génétique classique.

Dans l'algorithme 19, l'entier λ représente le nombre de génération depuis la dernière amélioration du meilleur individu est noté. Le booléen δ_{low} nous permet de savoir si l'algorithme traverse une période de diversité faible. Le booléen τ_{div} est fixé à *vrai* si l'algorithme est en phase de diversification. L'entier $nbGeneration$ renseigne le nombre de générations faites par l'algorithme génétique.

5.2.3.2 Sélection par diversité : *selectionDiversity()*

Cette méthode de sélection a pour but de constituer une population P_{next} de survivants jusqu'à la prochaine génération la plus hétérogène possible avec les individus présents dans P et $P_{children}$, tout en privilégiant les individus les plus adaptés au problème. Pour cela nous utilisons à nouveaux les métriques de Hamming et de Kendall Tau.

Algorithme 20: selectionDiversity()

Données: la population courante P et la population d'enfant $P_{children}$

Résultat: une population P_{next}

1 **Début**
2 $P_{total} \leftarrow shuffle(P \cup P_{children})$
3 $P_{next} \leftarrow \emptyset$; $P_{temp} \leftarrow \emptyset$
4 **Tant que** $|P_{next}| \neq |P|$ et $P_{total} \neq \emptyset$ **faire**
5 $Sol_i \leftarrow rouletteWheel(P_{total})$; $P_{total} \leftarrow P_{total} \setminus \{Sol_i\}$
6 $maxComparison \leftarrow \lceil \sqrt{|P_{next}|} \rceil$; $y \leftarrow 0$
7 $toBeAdded \leftarrow vrai$
8 **Tant que** $y < maxComparison$ **et** $toBeAdded = vrai$ **faire**
9 $Sol_y \leftarrow P_{next}[y]$
10 **Si** $D(Sol_i, Sol_y) < D_{min}$ **alors**
11 **Si** $f(Sol_i) < f(Sol_y)$ **alors**
12 $P_{next} \leftarrow P_{next} \setminus \{Sol_y\}$; $P_{temp} \leftarrow P_{temp} \cup \{Sol_y\}$
13 **Sinon**
14 $toBeAdded \leftarrow faux$
15 $P_{temp} \leftarrow P_{temp} \cup \{Sol_i\}$
16 **Fin**
17 $y{+}{+}$
18 **Fin**
19 **Si** $toBeAdded = vrai$ **alors**
20 $P_{next} \leftarrow P_{next} \cup \{Sol_i\}$
21 **Fin**
22 **Fin**
23 **Tant que** $|P_{next}| < |P|$ **faire**
24 $P_{next} \leftarrow P_{next} \cup rand(P_{temp})$
25 **Fin**
26 **Retourne** P_{next}
27 **Fin**

Cette sélection est présentée dans l'algorithme 20. Elle est partiellement basée sur la sélection par roulette : les probabilités sont calculées en fonction de la qualité

des solutions selon la formule 5.3. Nous ne souhaitons intégrer dans P_{next} que des individus distants de ceux déjà sélectionnés. Pour cela, chaque vainqueur Sol_i de la sélection par roulette est comparé (avec les métriques) avec des individus $Sol_y \in P_{next}$ avant d'y être inséré. Si la distance $D(Sol_i,Sol_y)$ est inférieure à un seuil D_{min} fixé au préalable, nous comparons la qualité des deux solutions. La meilleure des deux solutions intègre P_{next} tandis que l'autre est ajoutée à une population P_{temp}. Le nombre maximum de comparaison est donné dans la formule suivante :

$$maxComparison = \lceil \sqrt{|P_{next}|} \rceil \tag{5.10}$$

Nous avons choisi cette valeur pour ne pas comparer systématiquement la solution Sol_i avec toutes les solutions déjà présentes dans P_{next}, cette opération étant répétée $|P|$ fois. Avec ces critères de sélection supplémentaires, il est possible que la population P_{next} ne contienne pas assez d'individus ($|P_{next}| \neq |P|$). Dans ce cas, nous complétons la population avec des individus choisis aléatoirement dans la population P_{temp} de individus sélectionnés par la roulette mais qui n'ont pas intégré P_{next}.

Avec cette méthode de sélection, nous voulons limiter l'apparition de doublons ou d'individus très proches dans la génération suivante. Elle sera utilisée avec la méthode de gestion de population présentée précédemment et nous permettra de travailler sur des populations plus saines, contenant moins d'individus semblables.

Les métriques et les différentes méthodes les utilisant nous permettent d'avoir plus de contrôle sur le phénomène de convergence des solutions afin d'en profiter tout en limitant ses effets néfastes sur le long terme pour l'exécution de l'algorithme génétique. Nous évaluerons dans la section suivante l'apport de cette gestion de diversité dans l'algorithme génétique.

5.3 Expérimentations & Résultats

Dans cette section nous présentons les résultats obtenus avec les algorithmes génétiques *LORH_GA* et *LORH_GADM*. Nous reviendrons en premier lieu sur le paramétrage des algorithmes, avant de présenter les résultats obtenus sur les instances d'Evolucare. Nous les comparerons à nos méthodes précédentes : les solutions

optimales obtenues avec le modèle de programmation linéaire en 0-1 et les résultats obtenus avec l'algorithme de construction ***LORH_ARC*** et notre algorithme de recherche locale *LORH_ALNS*. Enfin, nous comparerons les résultats obtenus par *LORH_GA* et *LORH_GADM* aux résultats de la littérature sur les instances PSPLIB.

5.3.1 Paramétrage

Les algorithmes génétiques sont soumis à de nombreux paramètres, qui peuvent avoir un impact conséquent sur leurs performances. Pour déterminer les meilleurs paramètres pour notre problème, nous avons mis en place un protocole de test afin de choisir pour chaque opérateur de l'algorithme les paramètres permettant d'obtenir les meilleurs solutions. Ces tests seront menés sur un ensemble d'instances d'entraînement généré à partir des scénarios décrits dans le chapitre 3 en faisant varier les différents paramètres (taille, importances, urgences, etc.) et sur instances PSPLIB de la littérature.

5.3.1.1 Méthodes de sélection

Les algorithmes génétiques présentés dans ce chapitre comportent deux étapes de sélection pour choisir d'une part les individus utilisés lors de la phase de reproduction et d'autre part ceux qui survivront jusqu'à la génération suivante.

5.3.1.2 Sélection des parents

Pour déterminer la meilleure méthode de sélection des parents, nous avons mis en compétition les quatre méthodes de sélection présentées dans ce chapitre : les deux sélections par roulette, avec calcul des probabilités selon la qualité des solutions (5.3), et avec calcul selon le classement des solutions (5.5), et les deux sélections par tournoi, déterministe et non-déterministe. Chaque sélection a été testée sur l'algorithme génétique privé de la phase de mutation sur 100 générations, avec une taille de la population fixée à 100. La sélection des individus pour la génération suivante sera pour ces tests une simple sélection des meilleurs individus. Pour les sélections par tournoi, nous avons testé plusieurs tailles de tournoi (entre 2 et 5) et plusieurs

probabilités p d'accepter le meilleur individu pour la version non déterministe du tournoi (entre 50 et 90%, par pas de 10%).

Pour la sélection des parents, nous avons obtenu les meilleurs résultats avec la sélection par roulette basée sur le classement des solutions. Cette sélection sera utilisée pour cette étape dans tous nos expérimentations suivantes sur le paramétrage.

5.3.1.3 Sélection des survivants

Pour déterminer la meilleure méthode pour la sélection des survivants, nous utilisons un protocole similaire : les quatre méthodes de sélection sont testés, avec la sélection par roulette basée sur le classement des solutions pour sélectionner les parents. L'algorithme génétique est toujours privé de la phase de mutation, et les tests se déroulent toujours sur 100 générations et avec une population de 100 individus.

Pour la sélection des survivants, c'est la sélection par tournoi non-déterministe avec une taille de tournoi de 3 et une probabilité $p = 80\%$ pour le meilleur individu de gagner le tournoi qui nous donne les meilleurs résultats. Ces méthodes seront utilisées pour la suite des expérimentations sur le paramétrage et pour nos tests sur les différentes instances, à l'exception de l'algorithme génétique $LORH_GADM$ qui dispose d'une sélection des survivants qui lui est spécifique.

5.3.1.4 Type de crossover

La phase de reproduction peut être faite de nombreuses manières, et nous avons présenté précédemment plusieurs types de crossover : des crossovers à un et deux points, et un crossover sur l'ordre des rendez-vous. Pour trouver celui permettant d'obtenir les meilleurs résultats, nous utilisons un protocole similaire à celui des tests pour les méthodes de sélection : pas de phase de mutation, des tests sur 100 générations avec une population de 100 individus. Nous utilisons toujours l'ensemble d'entraînement décrit précédemment.

Sur les instances de l'entreprise, les meilleurs résultats ont été obtenus avec le crossover à deux points, tandis que le crossover sur l'ordre a obtenu de meilleurs résultats sur les instances PSPLIB. Ce dernier résultat n'est pas surprenant et nous avions déjà constaté de mauvaises performances des crossovers classiques sur ces instances. Sur les instances de l'entreprise, le crossover sur l'ordre donne des résultats

en deça des crossover classique, même sur les instances contenant des relations de précédence. Ces dernières, peu nombreuses et peu denses, ne profitent pas de l'apport de ce crossover et le coût supplémentaire engendré par la reconstruction des solutions selon l'ordre n'est pas rentabilisé. L'algorithme génétique $LORH_GA$ avec et sans le crossover et la mutation sur l'ordre des rendez-vous seront à nouveau comparés lors de nos tests sur les instances PSPLIB.

5.3.1.5 Taux de mutation

Le taux de mutation est un autre facteur affectant grandement l'efficacité des algorithmes génétiques. Nous avons testé deux approches pour la phase de mutation : muter la totalité de la population selon un taux τ de mutation appliqué sur chaque gène, ou muter seulement un pourcentage de la population, avec un taux de mutation plus élevé. Pour la première approche, nous faisons varier le taux de mutation par pas de 0.1 entre 0.1 et 1%, et par pas de 1% entre 1 et 10%. Nous testons en plus de cela deux valeurs de taux fréquemment utilisées au cours de nos travaux et dépendantes de la taille de l'instance traitée mais limitées à un taux maximum de 5% : $max(0.05, \frac{1}{|A|})$ et $max(0.05, \frac{2}{|A|})$. Pour la seconde approche, nous faisons varier le pourcentage de la population mutée entre 10 et 30% de la population (par pas de 5%), avec un taux de mutation entre 1 et 20% (par pas de 2%).

C'est la première approche, avec le taux de mutation adaptatif $max(0.05, \frac{2}{|A|}$ qui a fourni les meilleurs résultats à la fois sur les instances de l'entreprise et sur les instances PSPLIB.

5.3.1.6 Métriques et gestion de la population

L'algorithme avec gestion de la diversité de la population utilise des paramètres supplémentaires : Div_{min} le seuil minimum de diversité pour les fonctions $manage\text{-}PopulationDiversity()$ et $selectionDiversity()$, ν le nombre de générations sans amélioration à attendre avant de passer en phase de diversification, κ le nombre de génération d'une phase de diversification et enfin τ_{high} le taux de mutation appliqué durant cette phase. La métrique utilisée est également un paramètre de $LORH_GADM$.

Lors de notre comparaison des scores de diversité renvoyés par les distances de Hamming et de Kendall Tau, nous avions observé qu'ils étaient similaires sur la plupart de nos instances. Nous voulons tester différents paramétrages pour la méthode de gestion de la population et, dans un second temps, comparer les résultats obtenus avec les deux métriques. Pour la configuration de la gestion de la diversité, nous avons fait varier Div_{min} entre 5 et 40% par pas de 5%, ν entre 5 et 50 par pas de 5, κ entre 1 et 10 et τ_{high} entre 30 et 100%, par pas de 10%. Nous avons déterminé ces limites de manière empirique lors de l'élaboration de cette approche.

La configuration ayant obtenu les meilleurs résultats est la suivante : $Div_{min} = 10\%$, $\nu = 10$, $\kappa = 5$ et $\tau_{high} = 0.5$. Concernant les métriques, la métrique de Hamming s'est révélée être légèrement plus performante avec un écart de 1.03% (Hamming - Kendall / Kendall Tau) sur nos instances. Sur les instances PSPLIB, c'est au contraire la distance de Kendall Tau qui obtient de meilleurs résultats, avec une différence de 2.29%.

5.3.1.7 Taille de la population

Pour obtenir la taille de la population, nous avons utilisé le logiciel iRace, que nous avons déjà utilisé pour les paramètres de *LORH_ALNS*. Tous les paramètres évoqués précédemment sont fixés avec les meilleures valeurs que nous avons trouvées et seule la taille de la population varie (entre 2 et 500 individus) dans les itérations d'iRace.

iRace a déterminé une taille de population égale à 70, que nous utiliserons sur les deux algorithmes *LORH_GA* et *LORH_GADM*.

5.3.2 Résultats sur les instances de l'entreprise

Nous avons mené nos expérimentations sur le même ensemble de 80 instances générées à partir de quatre scénarios construits par l'entreprise, qui représentent des problèmes de planification de quatre établissements de santé différents. Nous utilisons la formule 3.1 pour évaluer la qualité de la solution obtenue. Les deux algorithmes génétiques ont été implémentés en C#. Comme pour nos expérimentations avec les recherches locales, nous limitons le temps de calcul à 10 minutes et utilisons la même machine, dotée d'un processeur Intel i7-7700HQ.

Pour tous les tableaux, nous reportons pour chaque instance son nombre de demandes de rendez-vous **T**, le nombre de rendez-vous importants **I**, le nombre de rendez-vous prioritaires **U**, le nombre de relations de précédence **P** et le nombre de demi-journées de disponibilité retirées aux patients **D**. Nous reportons à nouveau les bornes inférieures **LB** et supérieures **UB** obtenues par CPLEX. Nous indiquons aussi les meilleurs résultats $f(Sol_{best})$ obtenus par l'algorithme de construction aléatoire ***LORH_ ARC*** décrit dans la section 4.1 et par l'ALNS *LORH_ ALNS* décrit dans la section 4.3. Nous donnons aussi les meilleurs résultats de nos deux algorithmes génétiques *LORH_ GA* et *LORH_ GADM*, décrits respectivement dans les sous-sections 5.1.6 et 5.2.3. Pour *LORH_ GA* nous donnons l'écart $Gap_f = (LORH_GA - LORH_ALNS/LORH_ALNS)$. Pour *LORH_ GADM* nous donnons l'écart avec *LORH_ GA*, $Gap_f = (LORH_GADM - LORH_GA/LORH_GA)$. Les meilleures solutions obtenues pour chaque instance sont indiquées en gras.

TABLE 5.1: Résultats sur les instances de la famille ***SurgDep***.

SurgDep					***CPLEX***		***LORH_ ARC***	***LORH_ ALNS***	***LORH_ GA***		***LORH_ GADM***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ ALNS***	$f(Sol_{best})$	Gap_f ***LORH_ GA***
16	0	0	0	0	0	0	2	**0**	**0**	0%	**0**	0%
16	4	0	0	0	0	0	1	**0**	**0**	0%	**0**	0%
16	8	0	0	0	0	0	1	**0**	**0**	0%	**0**	0%
16	4	4	0	0	1.48	1.48	6.042	**1.48**	**1.48**	0%	**1.48**	0%
16	8	8	0	0	9.54	9.54	42.50	**9.54**	**9.54**	0%	**9.54**	0%
16	0	0	0	4	0	0	3	**2**	**2**	0%	**2**	0%
16	8	4	0	4	19.02	19.02	47.42	20.6	19.02	-7.69%	**19.02**	0%
48	0	0	0	0	0	9	3	**0**	**0**	0%	**0**	0%
48	12	0	0	0	0	7	3	**0**	**0**	0%	**0**	0%
48	24	0	0	0	0	8	12	**0**	**0**	0%	**0**	0%
48	12	12	0	0	57.88	68.16	232.10	60.77	**59.02**	-2.88%	60.85	3.11%
48	24	24	0	0	85.65	137.60	531.14	107.63	105.32	-2.14%	**101.30**	-3.82%
48	0	0	0	4	0	6	4	**0**	**0**	0%	**0**	0%
48	0	0	0	12	0	5	4	2	2	0%	**1**	-50%
48	24	12	0	12	114.77	116.27	304.29	185.39	117.27	-36.75%	**115.68**	-1.36%

Les résultats sur le scénario ***SurgDep*** sont reportés dans le tableau 5.1. Sur les instances de 16 rendez-vous de ce scénario, les deux algorithmes génétiques *LORH_ GA* et *LORH_ GADM* trouvent toutes les solutions optimales que nous avions obtenus avec CPLEX (**LB** = **UB**) et qui avaient été également trouvées par l'ALNS (à l'exception de la dernière, **T**=16, **I**=8, **U**=4, **D**=4). Sur les instances de taille 48, l'algorithme génétique *LORH_ GA* parvient à améliorer les résultats

obtenus par l'ALNS sur trois instances, dont la plus difficile de l'ensemble (**T**=48, **I**=28, **U**=12, **D**=12) avec un gain de 36.75%. Sur ce scénario, les différences entre *LORH_GA* et *LORH_GADM* sont minimes, à l'exception de l'instance **T=48** et **D=12** qui est améliorée de 50% par *LORH_GADM*.

TABLE 5.2: Résultats sur les instances du scénario ***RehabCenter***.

RehabCenter					***CPLEX***		***LORH_ARC***	***LORH_ALNS***	***LORH_GA***		***LORH_GADM***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ALNS***	$f(Sol_{best})$	Gap_f ***LORH_GA***
96	0	0	0	0	0	0	1	1	**0**	-100%	**0**	0%
96	24	0	0	0	0	0	1	1	**0**	-100%	**0**	0%
96	48	0	0	0	0	0	1	1	**0**	-100%	**0**	0%
96	24	24	0	0	265.95	317.63	317	**315**	**315**	0%	**315**	0%
96	48	48	0	0	557.81	603.25	602	**600**	**600**	0%	**600**	0%
96	0	0	15	0	0	0	3	**0**	**0**	0%	**0**	0%
96	48	24	15	0	308.57	315	335.75	**315**	**315**	0%	**315**	0%
96	96	48	15	0	532.88	1426.5	616.88	**600**	**600**	0%	**600**	0%
288	0	0	0	0	0	196	4	**0**	**0**	0%	**0**	0%
288	72	0	0	0	0	1914	5	**0**	**0**	0%	**0**	0%
288	144	0	0	0	0	4960	5	1	**0**	-100%	**0**	0%
288	72	72	0	0	689.06	2609.75	1125	**1125**	**1125**	0%	**1125**	0%
288	144	144	0	0	1958.35	6064.5	2948.5	**2790**	**2790**	0%	**2790**	0%
288	0	0	25	0	0	196	5	**0**	**0**	0%	**0**	0%
288	144	72	24	0	689.06	5729.88	1254	**1125**	**1125**	0%	**1125**	0%
288	288	144	25	0	1958.35	8740.5	2991.25	2840	**2790**	-1.76%	**2790**	0%

Nous reportons les résultats sur le scénario ***RehabCenter*** dans le tableau 5.2. Ici nous remarquons peu d'améliorations dans les résultats des algorithmes génétiques par rapport à l'ALNS. L'algorithme génétique classique *LORH_GA* améliore les résultats de *LORH_ALNS* sur cinq instances. Sur quatre d'entre elle, un écart de 100% est constaté et s'explique par la planification d'un rendez-vous supplémentaire. Ainsi pour les deux tailles d'instance, nous avons des écarts moyens importants de 33.33 et 11.31% qui sont dus à ce rendez-vous supplémentaire planifié sur ces quelques instances. L'algorithme *LORH_GADM* n'apporte aucune amélioration par rapport à *LORH_GA*, et obtient des résultats identiques sur toutes les instances.

Nous retrouvons dans le tableau 5.3 les résultats de nos méthodes sur le scénario ***Admission***. Les résultats de *LORH_GA* sur les instances de taille **T=136** sont mitigés lorsque que nous les comparons avec *LORH_ALNS* : nous notons une amélioration entre 3.87 et 25.63% sur les instances comportant des rendez-vous prioritaires (**U**$\neq$ 0), mais de meilleurs résultats de l'ALNS sur certaines instances sans rendez-vous prioritaire. Sur ces instances, l'écart est en moyenne de 3.7% en faveur

TABLE 5.3: Résultats sur les instances du scénario ***Admission***.

Admission					*CPLEX*		*LORH_ARC*	*LORH_ALNS*	*LORH_GA*		*LORH_GADM*	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ALNS***	$f(Sol_{best})$	Gap_f ***LORH_GA***
136	0	0	0	0	0	4	12	**4**	5	25%	**4**	-20%
136	34	0	0	0	0	4	36	**5**	**5**	0%	**5**	0%
136	68	0	0	0	0	4	91	5	5	0%	**4**	-20%
136	34	34	0	0	126.31	204.85	360.16	216.17	**207.59**	-3.97%	218.79	5.4%
136	68	68	0	0	149.84	298.76	512.91	363.82	**300.26**	-17.47%	315.03	4.92%
136	0	0	48	0	0	4	17	6	6	0%	**4**	-16.67%
136	0	0	0	35	0	4	10	**4**	5	25%	**4**	-20%
136	68	34	48	35	127.08	202.63	441.30	237.8	**228.59**	-3.87%	236.61	3.51%
136	136	68	48	35	154.66	306.78	713.66	548.18	407.69	-25.63%	**379.28**	-6.97%
408	0	0	0	0	0	24	14	**12**	**12**	0%	**12**	0%
408	102	0	0	0	0	31	104	**12**	13	8.33%	13	0%
408	204	0	0	0	0	65	122	**12**	13	8.33%	**12**	-7.69%
408	102	102	0	0	165.64	2841.45	994.88	722.06	605.30	-16.17%	**579.17**	-4.32%
408	204	204	0	0	345.11	5333.52	2240.42	1919.11	1386.96	-27.73%	**1164.65**	-16.03%
408	0	0	64	0	0	24	24	13	**12**	-7.69%	**12**	0%
408	0	0	85	0	0	24	26	13	**12**	-7.69%	**12**	0%
408	0	0	0	91	0	24	14	13	**12**	-7.69%	**12**	0%
408	204	102	64	91	165.64	3136.23	1380.07	1253.2	604.75	-51.74%	**565**	-6.57%
408	408	204	85	91	345.82	6904.07	3436.33	2885.59	2632.57	-8.77%	**2366.80**	-10.1%

de l'ALNS. L'algorithme avec gestion de la diversité *LORH_GADM* améliore les résultats par rapport aux deux autres méthodes sur trois instances qui ne comportent pas de rendez-vous prioritaires. Sur trois instances ayant des rendez-vous urgents, *LORH_GADM* obtient de moins bonnes solutions que *LORH_GA*, avec des écarts entre 3.51 et 5.4%. Pour les instances de 408 rendez-vous, *LORH_GA* surpasse *LORH_ALNS* sur la majorité des instances, et présente un écart moyen de 9.38% avec l'ALNS. L'algorithme *LORH_GADM* améliore ou égale les résultats de l'algorithme génétique classique sur toutes les instances de cette taille, avec un écart moyen de 4.06%.

Enfin, les résultats sur les instances du scénario ***CardioRehab*** sont présentés dans le tableau 5.4. Sur ce scénario, la plupart des solutions optimales sur les instances de 160 et 480 sans rendez-vous prioritaires (**U**=0) avaient déjà été atteintes par *LORH_ALNS*. Sur ces instances, les algorithmes *LORH_GA* et *LORH_GADM* obtiennent la solution optimale sur deux instances supplémentaires comportant des précédences (**D=38** et **D=122**), et des résultats identiques sur toutes les autres. Sur les instances comportant des rendez-vous prioritaires (**U**≠0), les résultats pour l'algorithme génétique classique par rapport à l'ALNS sont plus mitigés, avec des amé-

TABLE 5.4: Résultats sur les instances du scénario ***CardioRehab***.

CardioRehab					***CPLEX***		***LORH_ARC***	***LORH_ALNS***	***LORH_GA***		***LORH_GADM***	
T	**I**	**U**	**P**	**D**	**LB**	**UB**	$f(Sol_{best})$	$f(Sol_{best})$	$f(Sol_{best})$	Gap_f ***LORH_ALNS***	$f(Sol_{best})$	Gap_f ***LORH_GA***
160	0	0	0	0	0	0	**0**	**0**	**0**	0%	**0**	0%
160	40	0	0	0	0	0	**0**	**0**	**0**	0%	**0**	0%
160	80	0	0	0	0	0	1	**0**	**0**	0%	**0**	0%
160	40	40	0	0	12.54	32.32	79.16	19.22	19.22	0%	**16.27**	-15.33%
160	80	80	0	0	57.93	141.62	395.28	128.91	131.43	1.96%	**127.27**	-3.17%
160	0	0	27	0	0	0	3	**0**	**0**	0%	**0**	0%
160	0	0	36	0	0	0	6	**0**	**0**	0%	**0**	0%
160	0	0	0	19	0	0	4	**0**	**0**	0%	**0**	0%
160	0	0	0	38	0	0	5	1	**0**	100%	**0**	0%
160	80	40	27	19	12.92	43.57	131.98	50.39	**19.22**	-61.86%	22.32	16.16%
160	160	80	36	38	58.80	185.42	388.44	310.8	181.42	-41.63%	**151.70**	-16.38%
480	0	0	0	0	0	154	1	**0**	**0**	0%	**0**	0%
480	120	0	0	0	0	498	1	**0**	**0**	0%	**0**	0%
480	240	0	0	0	0	656	1	**0**	**0**	0%	**0**	0%
480	120	120	0	0	168.73	3622.93	2680.14	635.98	842.67	32.5%	**517.85**	-38.55%
480	240	240	0	0	223.13	11145.42	3545.31	1472.84	1898.24	28.88%	**1281.39**	-32.5%
480	0	0	38	0	0	154	7	**0**	**0**	0%	**0**	0%
480	0	0	48	0	0	154	4	**0**	**0**	0%	**0**	0%
480	0	0	0	60	0	163	7	**0**	**0**	0%	**0**	0%
480	0	0	0	122	0	122	11	1	**0**	-100%	**0**	0%
480	240	120	38	60	182.74	6471.85	2668.62	840.11	922.39	9.79%	**792.83**	-14.05%
480	480	240	48	122	235.41	21690.75	5030.68	3215.39	2746.57	-14.58%	**2029.77**	-26.1%

liorations allant jusqu'à 61.86% mais aussi des solutions pouvant être jusqu'à 32.5% moins bonnes que celles de l'ALNS, notamment sur les instances plus grandes. L'algorithme *LORH_GADM* surpasse les deux autres méthodes sur toutes les instances sauf une, et présente un écart moyen avec *LORH_GA* de 1.7% sur les instances de 160 rendez-vous et de 11.12% sur les instances de 480 rendez-vous.

TABLE 5.5: Résumé des écarts moyen entre les différentes méthodes et du taux de rendez-vous posés sur chaque famille d'instance.

Scénarios	$\|A\|$	Gap_f **moyens**		**Taux de rendez-vous planifiés** $\|A_{Sol_{best}}\|/\|A\|$		
		LORH_ALNS/LORH_GA	***LORH_GA/LORH_GADM***	***LORH_ALNS***	***LORH_GA***	***LORH_GADM***
SurgDep	16	-5.5%	0%	98.61%	98.61%	100%
	48	-4.86%	-5.87%	98.38%	99.07%	99.31%
RehabCenter	96	-22.22%	0%	100%	99.65%	100%
	288	-11.31%	0%	98.03%	100%	100%
Admission	136	3.7%	-12.06%	95.59%	96.26%	96.32%
	408	-9.38%	-4.06%	96.77%	96.52%	96.68%
CardioRehab	160	-27.41%	-1.7%	99.83%	100%	99.9%
	480	-4.34%	-11.12%	99.69%	99.38%	99.4%

Dans le tableau 5.5 nous résumons les gains de performance entre $LORH_ALNS$ et l'algorithme génétique $LORH_GA$, puis entre $LORH_GA$ et l'algorithme avec gestion de diversité $LORH_GADM$. Nous reportons aussi pour ces trois méthodes le taux moyen de rendez-vous planifiés ($|ASol_{best}|/|A|$) pour chaque famille d'instance. Sur la majorité des différentes familles, $LORH_GA$ surpasse l'ALNS, avec des améliorations allant jusqu'à 33.3%. Cependant, ces écarts parfois importants sont à nuancer avec les différences très faibles entre la qualité des solutions. Les taux de rendez-vous planifiés sont très légèrement meilleurs pour $LORH_GA$, avec des différences faibles de moins de 1%. Concernant les différences entre $LORH_GA$ et $LORH_GADM$, ce dernier obtient en moyenne de meilleurs résultats sur les scénarios ***SurgDep***, ***Admission*** et ***CardioRehab***. Sur les scénarios ***Admission*** et ***CardioRehab***, nous observons des améliorations allant jusqu'à 11.12% et 12.06%. Concernant le taux de rendez-vous posés, les chiffres des deux algorithmes génétiques restent proches, avec moins de 1% de différence sur toutes les familles d'instances. Les améliorations apportées par $LORH_GADM$ par rapport à l'algorithme génétique classique sont concentrées sur les instances comportant des rendez-vous prioritaires. En observant la diversité des populations, nous avons remarqué qu'elles étaient particulièrement basses sur ce type d'instances, avec des solutions très semblables. Cela peut s'expliquer par les rendez-vous prioritaires qui sont toujours placés le plus proche possible du début de l'intervalle de faisabilité, limitant ainsi les possibilités de placement. Les phases de diversification et la sélection par diversité permettent de faire émerger des solutions hétérogènes, ce qui semble améliorer les performances de l'algorithme sur ce type d'instance.

5.3.3 Résultats sur les instances PSPLIB

Pour comparer les performances de nos algorithmes génétiques sur des instances de la littérature, nous utilisons à nouveau le même ensemble d'instances PSPLIB utilisé dans la sous-section 4.4.3. Cet ensemble est composé de 80 instances de chaque famille J30, J60, J90 et J120 (de 30, 60, 90 et 120 activités, respectivement) de la bibliothèque PSPLIB. Les solutions sont évaluées selon la formule 3.15, qui vise à réduire le *makespan*, c'est-à-dire la durée entre le début de la première activité planifiée et la fin de la dernière activité. Nous comparerons les résultats obtenus

avec ceux de la littérature. Le meilleur ***makespan*** trouvé pour chaque instance est répertorié sur le site de la bibliothèque PSPLIB (Kolisch and Sprecher, 2020).

Ces expérimentations ont été réalisées dans les mêmes conditions qu'avec l'ALNS : nous limitons le temps de calcul à deux minutes, et les tests sont menés sur une machine équipée d'un processeur Intel I7-7700HQ. Nous comparons trois méthodes : deux versions de l'algorithme génétique classique LORH_GA, avec et sans les opérateurs de crossover et de mutation sur l'ordre des activités, et l'algorithme génétique avec gestion de diversité LORH_GADM, avec les opérateurs sur l'ordre.

Dans les tableaux suivants, nous donnons pour chaque famille et pour chaque instance : dans la première colonne **ID paramètres**, l'identifiant de l'ensemble de paramètres utilisés pour générer l'instance, et **ID** l'identifiant de l'instance, le meilleur ***makespan*** trouvé dans la littérature et reporté sur ces instances PSPLIB, puis pour chaque méthode, le ***makespan*** trouvé et l'écart par rapport au résultat de la littérature.

TABLE 5.6: Résultats sur la famille J30 des instances PSPLIB.

J30		*PSPLIB Makespan*	*LORH_GA Opérateurs classiques*		*LORH_GA Opérateurs ordre*		*LORH_GADM Opérateurs ordre*	
ID paramètres	**ID instance**		***Makespan***	$Gap_{makespan}$	***Makespan***	$Gap_{makespan}$	***Makespan***	$Gap_{makespan}$
2	7	47	47	0%	47	0%	47	0%
3	6	54	54	0%	54	0%	54	0%
7	1	55	55	0%	55	0%	55	0%
9	9	63	71	12.7%	66	4.76%	66	4.76%
7	1	55	55	0%	55	0%	55	0%
13	4	72	86	19.44%	74	2.78%	76	5.56%
17	6	63	72	14.29%	63	0%	63	0%
30	2	68	80	17.65%	71	4.41%	70	2.94%
36	4	59	59	0%	59	0%	59	0%
41	6	103	110	6.8%	103	0%	104	0.97%
47	8	48	51	6.25%	48	0%	48	0%

Les résultats obtenus sur certaines instances de la famille J30 sont donnés dans le tableau 5.6. Sur cette première famille, nous remarquons des écarts importants entre la meilleure solution et celles renvoyées par l'algorithme *LORH_GA* avec les opérateurs classiques. Si celui-ci parvient à trouver la solution optimale sur certaines instances, des écarts entre 6.25 et 19.44% apparaissent avec la solution optimale. Les solutions obtenues avec les opérateurs sur l'ordre sont systématiquement meilleures, avec des écarts toujours présents mais plus faibles, entre 2.78 et 4.76%. Nous no-

tons peu de différences entre l'algorithme *LORH_ GADM* et *LORH_ GA* avec les opérateurs sur l'ordre des rendez-vous.

TABLE 5.7: Résultats sur la famille J60 des instances PSPLIB.

J60		***PSPLIB***	***LORH_ GA***		***LORH_ GA***		***LORH_ GADM***	
		Makespan	***Opérateurs classiques***		***Opérateurs ordre***		***Opérateurs ordre***	
ID paramètres	**ID instance**		***Makespan***	$Gap_{makespan}$	***Makespan***	$Gap_{makespan}$	***Makespan***	$Gap_{makespan}$
1	7	72	78	8.33%	75	4.17%	75	4.17%
1	8	75	83	10.67%	76	1.33%	76	1.33%
2	4	78	82	5.13%	78	0%	78	0%
5	6	74	94	27.03%	84	13.51%	81	9.46%
7	7	89	94	5.62%	89	0%	89	0%
11	10	58	58	0%	58	0%	58	0%
14	2	65	75	15.38%	68	4.62%	68	4.62%
17	4	71	80	12.68%	71	0%	71	0%
24	10	66	66	0%	66	0%	66	0%
32	9	74	75	1.35%	74	0%	74	0%
41	3	98	125	27.55%	110	12.24%	111	13.27%

Nous reportons les résultats obtenus sur une dizaine d'instances de la famille J60 dans le tableau 5.7. Sur cette famille d'instances comportant 60 activités à planifier, nous notons que l'algorithme *LORH_ GA* sans les opérateurs parvient difficilement à trouver une solution proche ou égale à la meilleure solution de la littérature sur la plupart des instances de cet échantillon. Les écarts sont aussi plus importants. Ici aussi, les résultats de *LORH_ GA* avec les opérateurs spécifiques aux instances riches en relations de précédence sont meilleurs, et nous obtenons quelques solutions de qualité égale aux meilleures solutions de la littérature. Des écarts entre 1.33 et 13.51% sont présents sur ces quelques instances. L'approche *LORH_ GADM* obtient des résultats légèrement meilleurs sur deux instances, avec des différences de l'ordre de 1 à 4%.

Les résultats sur quelques instances de la famille J90 sont reportés dans le tableau 5.8. Nous faisons ici les mêmes constatations que sur le tableau précédent : l'algorithme génétique classique sans les méthodes permettant un meilleur traitement des précédences trouve peu de résultats égaux aux résultats de la littérature. Sur ces quelques instances, l'écart est supérieur à 15% sur la moitié d'entre elles. Les algoritmes *LORH_ GA* et *LORH_ GADM* avec le crossover et la mutation sur l'ordre trouvent ici encore des résultats identiques à ceux de la littérature. Cepen-

TABLE 5.8: Résultats sur la famille J90 des instances PSPLIB.

J90		*PSPLIB Makespan*	*LORH_GA Opérateurs classiques*		*LORH_GA Opérateurs ordre*		*LORH_GADM Opérateurs ordre*	
ID paramètres	ID instance		*Makespan*	$Gap_{makespan}$	*Makespan*	$Gap_{makespan}$	*Makespan*	$Gap_{makespan}$
1	9	72	85	18.06%	76	5.56%	76	5.56%
4	9	79	79	0%	79	0%	79	0%
6	10	94	106	12.77%	94	0%	94	0%
8	3	70	70	0%	70	0%	70	0%
9	10	111	136	22.52%	135	21.62%	131	18.02%
14	2	79	103	30.38%	85	7.59%	85	7.59%
22	5	96	104	8.33%	96	0%	96	0%
29	1	135	173	28.15%	162	20%	166	22.96%
35	6	72	73	1.39%	72	0%	72	0%
40	7	87	87	0%	87	0%	87	0%
45	5	173	217	25.43%	210	21.39%	209	20.81%

dant des écarts importants supérieurs ou égaux à 20% sont présents pour ces deux méthodes, ce qui n'était pas le cas dans les ensembles d'instances précédents.

TABLE 5.9: Résultats sur la famille J120 des instances PSPLIB.

J120		*PSPLIB Makespan*	*LORH_GA Opérateurs classiques*		*LORH_GA Opérateurs ordre*		*LORH_GADM Opérateurs ordre*	
ID paramètres	ID instance		*Makespan*	$Gap_{makespan}$	*Makespan*	$Gap_{makespan}$	*Makespan*	$Gap_{makespan}$
2	5	103	129	25.24%	114	10.68%	117	13.59%
3	9	86	101	17.44%	86	0%	86	0%
4	9	79	87	10.13%	79	0%	79	0%
5	2	80	85	6.25%	80	0%	80	0%
6	2	134	182	35.82%	165	23.13%	164	22.39%
7	5	131	172	31.3%	156	19.08%	159	21.37%
8	7	87	118	35.63%	106	21.84%	104	19.54%
9	5	114	139	21.93%	114	0%	114	0%
17	10	134	168	25.37%	163	21.64%	159	18.66%
23	7	104	121	16.35%	106	1.92%	106	1.92%
34	8	89	113	26.97%	109	22.47%	109	22.47%

Les résultats sur un ensemble d'instances de la famille J120 sont donnés dans le tableau 5.9. Ces instances sont les plus grandes des quatre familles, avec 120 activités à planifier. Sur ces grandes instances, *LORH_GA* avec les opérateurs classiques ne peut trouver aucune solution de qualité égale à la littérature. Sur l'ensemble des 80 instances de cette famille, cette méthode ne trouve la meilleure solution que sur deux instances. De plus, les écarts sont très importants et varient de 6.25 à 35.82% sur ces quelques instances. Avec les opérateurs spécifiques, les deux méthodes *LORH_GA*

et $LORH_GADM$ trouvent la meilleure solution sur quatre instances. Sur les autres, nous constatons des écarts jusqu'à 23.13%.

TABLE 5.10: Écarts moyens pour chaque méthode et pour chaque famille d'instances PSPLIB.

Familles d'instances	$\|A\|$	**Moyenne des** $Gap_{makespan}$		
		LORH_GA ***Opérateurs classiques***	***LORH_GA*** ***Opérateurs ordre***	***LORH_GADM*** ***Opérateurs ordre***
J30	30	5.33%	0.77%	0.72%
J60	60	8.17%	3.16%	3.3%
J90	90	9.49%	4.19%	4.41%
J120	120	22.19%	13.46%	13.38%

Les résultats obtenus sur les instances PSPLIB sont résumés dans le tableau 5.10. Nous reportons pour chaque famille d'instances le nombre d'activités des instances et les écarts moyens relevés entre les résultats de nos trois méthodes et les meilleurs résultats obtenus dans la littérature. Nous pouvons confirmer avec ces résultats l'efficacité des opérateurs spécifiques à ce type d'instances, très riches en relation de précédence : l'algorithme $LORH_GA$ obtient, sans ces opérateurs, un écart moyen de 11.3%, qui se réduit à 5.4% avec ces opérateurs. Nous constatons peu de différences entre l'algorithme génétique classique et avec gestion de diversité, quelle que soit la famille d'instance.

Conclusion

Dans ce chapitre, nous avons présenté un algorithme génétique adapté à notre problème de planification. En premier lieu nous avons rappelé les principes de cette algorithme évolutionnaire, avant de décrire le codage de nos solutions pour cette méthode, et les différents opérateurs dont nous nous servons : les différentes opé-

rations de reproduction, de sélection et de mutation destinées à faire évoluer nos populations de solutions.

Ensuite, nous avons introduit les métriques de Hamming et de Kendall Tau, que nous avons adaptées aux spécificités de notre problème. Ces métriques ont été utilisées au sein d'un second algorithme génétique ayant pour objectif de limiter le phénomène de convergence trop rapide des solutions vers des optimums locaux. Pour cela, nous avons décrit nos méthodes pour mieux gérer la diversité des populations et des phases d'exploration et d'intensification de l'algorithme.

Nous avons confronté les deux algorithmes génétiques $LORH_GA$ et $LORH_GADM$ sur les instances d'Evolucare et sur des instances de référence du problème RCPSP, issues de la bibliothèque PSPLIB. Avant ces expérimentations nous avons mené des tests pour tenter d'optimiser les nombreux paramètres de ces deux algorithmes. Sur les instances PSPLIB, nous avons comparé les performances des nos approches. Les deux algorithmes génétiques utilisant des opérateurs créés spécialement pour traiter les composantes fortement liées par des relations de précédence sont plus efficaces que l'ALNS, et nous obtenons un écart moyen de 5.4% avec les meilleurs résultats de la littérature. Sur les instances de l'entreprise, $LORH_GA$ et $LORH_GADM$ ont été comparés à l'ALNS décrit dans le chapitre 4 et nous avons constaté une amélioration globale de 10.2% des résultats de $LORH_GA$ par rapport à l'ALNS. L'algorithme avec gestion de diversité $LORH_GADM$ améliore encore ces résultats de 4.35% en moyenne sur l'ensemble des scénarios. Les deux algorithmes génétiques nous permettent aussi d'obtenir un taux de planification des rendez-vous de 98.95% en moyenne sur toutes nos instances.

Conclusion générale

Les problématiques de planification sous contraintes sont des problèmes difficiles à résoudre, présents dans tous les domaines, et représentant des enjeux économiques importants dans le secteur industriel. La communauté scientifique s'est emparée de ce sujet et propose depuis des décennies des formulations et des méthodes de résolution pour traiter de nombreux problèmes de planification. Dans ce manuscrit, nous nous sommes intéressés à un problème de planification dans le milieu de la santé en collaboration avec Evolucare, une entreprise éditrice de logiciels destinés aux établissements de santé. Dans ce mémoire de thèse, nous avons présenté nos différentes contributions pour résoudre les problèmes rencontrés par l'entreprise.

Dans le premier chapitre nous avons présenté le contexte industriel de ce problème. Nous avons présenté l'entreprise, le contexte et les facteurs qui ont favorisé le montage de ce projet. Nous avons décrit la problématique rencontrée par les équipes médicales, et les difficultés qu'ils éprouvent à planifier, généralement à la main, les rendez-vous des patients avec leurs médecins. Nous avons introduit le projet LORH, dans lequel s'inscrit cette thèse CIFRE, et les difficultés qui devront être surmontées avant sa mise sur le marché et son utilisation par les équipes soignantes.

Parmi ces difficultés se trouve la complexité même du problème d'optimisation qui est traité. Les problèmes de planification sous contraintes font partie des problèmes NP-difficiles et figurent parmi les plus complexes à traiter pour les informaticiens. La littérature des problèmes de planification du domaine médical s'est largement enrichie au cours des dernières années, et nous avons situé notre propre problématique parmi les divers problèmes qu'elle traite. Les cas d'utilisation du logiciel présentés à l'entreprise par les planificateurs présentent de nombreuses similitudes dans leur définition avec le problème de gestion de projet à contraintes de ressources, plus souvent appelé Resource Constraint Project Scheduling Problem (RCPSP).

Nous avons ensuite décrit formellement le problème dans une modélisation mathématique intégrant l'ensemble des données et des contraintes du problème que nous traitons. La structure des solutions a été expliquée et nous avons proposé une fonction d'évaluation permettant de définir la qualité d'une solution, en accord avec les critères donnés par l'entreprise et les futurs utilisateurs du logiciel. Nous avons décrit plusieurs scénarios et nous avons construit un ensemble d'instances nous permettant de tester nos différentes approches pour résoudre ce problème. Le modèle mathématique a été implémenté avec le solveur CPLEX et nous a fourni des solutions optimales sur un petit nombre d'instances. En raison de la grande taille de certaines instances et du nombre de possibilités de solutions, nous n'avons pu obtenir avec ce modèle que des bornes inférieures et supérieures de la fonction d'évaluation sur les autres. Les temps de calculs n'étaient de plus pas compatibles avec une utilisation de cette méthode au sein du logiciel.

Pour obtenir des solutions satisfaisantes en un temps raisonnable, nous nous sommes tournés vers des méthodes approchées. Pour fournir des solutions initiales à ces méthodes, nous avons avant tout développé un algorithme de construction, nommé *LORH_ARC*. Ce dernier génère une solution initiale en tentant de planifier les demandes de rendez-vous du problème selon un ordre aléatoire. Bien que cette méthode soit rapide, elle ne permet pas d'obtenir des solutions d'une qualité satisfaisante : de nombreux rendez-vous ne sont pas planifiés, et les placements des rendez-vous prioritaires peuvent être grandement améliorés. Néanmoins, ces solutions représentent de bonnes bases pour les méta-heuristiques que nous avons conçues.

En premier lieu nous avons opté pour une méthode de recherche locale. Les mouvements que nous avons développés consistent à détruire une partie de la solution pour libérer des ressources selon des critères tels que l'importance des rendez-vous ou leur facilité à être planifiés à nouveau, puis à les reconstruire en planifiant le plus de demandes possibles. Cette reconstruction se fait également en suivant certains objectifs, comme planifier les rendez-vous prioritaires plus tôt, ou soulager l'emploi du temps des ressources les plus demandées. Nous avons intégré ces mouvements dans une approche nommée Adaptive Large Neighborhood Search (ALNS). L'ALNS est une méthode de recherche locale qui sélectionne à chaque itération l'un des mouve-

ments et l'applique sur la solution, pour se diriger vers un voisin de cette solution. Les probabilités associées aux mouvements évoluent tout au long du déroulement de l'algorithme et favorisent ceux fournissant les meilleurs résultats. Nous avons comparé les résultats de cette méthode, que nous avons appelé *LORH_ALNS*, aux résultats obtenus avec notre algorithme de construction et les résultats de CPLEX sur les instances d'Evolucare. *LORH_ALNS* a amélioré les résultats obtenus avec *LORH_ARC* de 50% en moyenne sur l'ensemble des familles. Nous obtenons la majorité des solutions optimales de CPLEX avec *LORH_ALNS* sur nos plus petites instances. Pour constater l'apport de la couche adaptative de l'ALNS, nous avons comparé les résultats obtenus par l'algorithme avec et sans l'adaptation des poids des mouvements : *LORH_ALNS* surpasse l'(¬A)LNS, avec un gain de 19.86% grâce à la couche adaptative de l'ALNS.

Les performances de *LORH_ALNS* ont aussi été évaluées sur les instances PSPLIB, une bibliothèque reconnue et régulièrement utilisée dans la littérature RCPSP. Nous avons constaté des écarts importants entre les résultats de *LORH_ALNS* et les meilleures solutions de la littérature : entre 10.59 et 26.52% selon les familles d'instances. Nous expliquons ces mauvais résultats par les particularités des instances PSPLIB, qui ne sont pas prises en compte par les mouvements utilisés dans l'ALNS. En effet, les activités de ces instances sont toutes reliées par de très nombreuses relations de précédence, et nos mouvements n'ont pas été conçus pour traiter de tels problèmes. Les instances RCPSP, bien que proches en théorie des nôtres, ont des structures particulières sur lesquelles nos mouvements sont peu efficaces.

Pour améliorer nos résultats à la fois sur les instances de l'entreprise et les instances de la littérature, nous nous sommes tournés vers les algorithmes génétiques, des méthodes particulièrement performantes sur les problèmes de planification. Nous avons décrit cette méthode, et notre manière de l'adapter à notre problème, ce qui comprend le codage de nos solutions et les opérateurs génétiques spécifiques aux différentes particularités des problèmes que nous traitons. En plus des techniques de croisement classiques, nous avons développé une méthode de croisement et une mutation adaptées aux composantes fortement liées par des relations de précédence. Notre algorithme génétique *LORH_GA* surpasse nos méthodes précédentes et nous permet d'obtenir de nombreuses solutions optimales supplémentaires, en particu-

lier sur les instances sans rendez-vous prioritaires. Nous notons un gain de 10.16% par rapport aux résultats obtenus avec *LORH_ALNS*, et nous avons aussi obtenu de meilleurs résultats sur les instances PSPLIB. En utilisant des opérateurs agissant sur l'ordre des rendez-vous, *LORH_GA* réduit significativement les écarts avec les meilleures solutions de la littérature, avec des différences moyennes allant de 0.77% sur les plus petites instances, à 13.46% sur les plus grandes, pour un écart moyen de 5.4%. À l'inverse, le même algorithme *LORH_GA* sans les opérateurs sur l'ordre des rendez-vous obtenait toujours des résultats décevants : s'ils étaient légèrement meilleures que *LORH_ALNS*, les écarts constatés étaient toujours importants (11.3% en moyenne). Ces résultats et les écarts constatés avec et sans les opérateurs spécifiques soulignent l'importance des différences existant entre nos instances issues de scénarios inspirés des problèmes réels et celles de la littérature, qui étaient pourtant proches dans leur définition.

Nous avons développé un autre algorithme génétique, nommé *LORH_GADM*, pour tenter de maîtriser un phénomène constaté sur l'algorithme génétique précédent : la convergence rapide des solutions vers des optimums locaux. Pour étudier et mieux comprendre ce phénomène, nous avons adapté les métriques de Hamming et de Kendall Tau à notre problème afin de mesurer la distance séparant deux solutions. Avec ces distances, nous étions capables de mesurer la diversité d'une population et donc de concevoir des méthodes pour mieux gérer les baisses prolongées qui nuisent aux performances de l'algorithme. L'algorithme génétique *LORH_GADM* augmente fortement le taux de mutation lorsque qu'une période de diversité faible et ne menant à aucune amélioration de la solution est détectée. Nous avons aussi ajouté une méthode de sélection qui privilégie les individus étant à la fois de bonne qualité et distants des autres individus sélectionnés. Sur les instances PSPLIB, nous avons noté peu de différences entre les résultats de *LORH_GADM* et *LORH_GA*. Sur les instances de l'entreprise, *LORH_GADM* améliore les résultats de *LORH_GA*, avec un gain allant jusqu'à 12.06% sur certaines de nos familles d'instances et de 4.35% en moyenne sur toutes les familles. Nous obtenons aussi avec cette méthode le taux de planification des rendez-vous le plus élevé, avec une moyenne de 98.95% de rendez-vous planifiés.

Tout au long de cette thèse réalisée en entreprise, nos différentes méthodes ont été intégrées au logiciel LORH et nous ont permis de démontrer la pertinence du prototype au cours de nombreuses démonstrations. À l'heure où nous écrivons ce mémoire, le projet LORH est en attente d'intégration au sein d'un module d'agenda, indispensable pour pouvoir proposer aux clients les fonctionnalités de planification automatique. Les travaux de cette thèse nous permettent de proposer un outil capable de résoudre les premiers problèmes de planification soumis par les clients de l'entreprise, et même d'anticiper sereinement les problèmes plus complexes qui nous seront proposés.

La fin de ces travaux nous ouvre de nombreuses perspectives, que ce soit au niveau académique ou industriel. Notre objectif premier est de permettre l'intégration rapide de LORH afin de pouvoir proposer cet outil rapidement aux équipes soignantes. Nous continuerons aussi d'améliorer les performances de nos méthodes, afin d'être capables de répondre à d'autres problèmes de planification, plus variés et plus difficiles. Pour ce qui est de nos méthodes de résolution, nous avons laissé de nombreuses idées en suspens au cours de ces dernières années. En ce qui concerne *LORH_ALNS*, nous aurions voulu intégrer d'autres mouvements, notamment des méthodes permettant d'en améliorer les résultats sur des instances riches en relation de précédence. Pour ce qui est des algorithmes génétiques, de nombreuses améliorations sont envisagées : créer un hybride avec nos méthodes de recherche locale, implémenter un système d'îles pour y résoudre des sous-parties du problème ou encore une autre hybridation avec des méthodes d'apprentissage nous permettant d'influencer l'algorithme dans ses choix lors de la planification.

Remerciements

Ces quatre années et même les deux qui les ont précédées ont été riches en surprises, en émotions et en rencontres. Beaucoup de joies, quelques déceptions, des espoirs et des craintes qui ont rythmé cette expérience unique et exigeante de la thèse. Ces moments, je les ai traversés aux côtés de ma famille et des nombreuses personnes dont j'ai fait la connaissance durant cette thèse. Elles m'ont soutenu au fil de ces années, et je tiens à leur rendre hommage à travers ces quelques lignes. J'espère n'oublier personne, mais j'espère que vous m'excuserez si votre nom n'est pas mentionné dans ce qui suit !

Dans un premier temps, je souhaite remercier mes directrices de thèse, Corinne et Laure, ainsi que Sylvain, qui m'ont encadré au cours de ces travaux. Corinne, tu m'as fait découvrir ce monde de la recherche et avec Laure, vous avez su me me guider et me soutenir : vous vous êtes montrées à l'écoute et patientes tout au long de ces travaux. Je n'aurai pu espérer de meilleures encadrantes. Nos excursions en conférences me manqueront ! Sylvain, je n'aurai jamais réussi sans ton soutien, tes précieux conseils et tes blagues (parfois) tordantes. À l'avenir, je ne peux qu'espérer inspirer un jour autant de respect que j'en ai pour toi aujourd'hui. Ce serait un plaisir de travailler à nouveau avec toi un jour.

Ensuite, j'adresse mes plus sincères remerciements à Sandra Bringay et Antoine Jouglet d'avoir accepté et pris le temps de rapporter mes travaux dans un délai particulièrement court : vous vous êtes montrés patients et compréhensif et vous m'avez permis de préparer la soutenance plus sereinement. Je remercie également Gilles Dequen et François d'avoir accepté d'être membre du jury de cette thèse, et pour les échanges intéressants que nous avons eu lors de la soutenance.

Cette thèse CIFRE m'a amené à faire la connaissance de personnes exceptionnelles, à la fois au laboratoire MIS et dans l'entreprise Evolucare. Du côté du MIS, je pense à tous les permanents, et plus particulièrement à ceux de l'équipe GOC : Céline, Yu, Chu Min, Laure et Corinne, pour nos ballades dans Amiens le vendredi

midi et nos discussions passionnantes au cours de ces années. Je remercie aussi mes collègues doctorants, notamment ceux avec qui j'ai partagé le mythique bureau 418 : dans un premier temps, Clément, Romuald, Simon et Richardson qui m'ont accueilli dans le laboratoire lors de mes débuts de stage, il y a six ans déjà. Ensuite, la génération suivante de doctorants avec qui j'ai partagé le bureau et des bons moments avant la pandémie : Monica, Clémence et Fabien, nos discussions et rigolades me manqueront ! Et pour finir avec le bureau, je remercie la génération d'après, à savoir Sébastien, Pierre et Kostas, pour ces quelques moments agréables et bienvenus après le Covid. Je n'oublie pas nos collègues des bureaux voisins, toujours présents pour boire un thé pendant des (longues) pauses bien méritées : Jordan, Anass, Audrey, Nathan, Thibault, Doha et Thawsif. J'ai une pensée particulière pour nos extraordinaires secrétaires : Valérie qui m'avait accompagné lors de mes premiers pas au labo, puis Juliette, pour nos papotages et pour ton aide précieuse !

À Evolucare, j'ai été intégré dans l'équipe Elabs, constituée d'une dizaine de personnes lors de mon arrivée dans le service et qui s'est ensuite encore agrandie. Les mots me manquent pour décrire cette ambiance incroyable, hors-norme dont j'ai pu profiter pendant une grande partie de ma thèse, aussi je vais simplement remercier toutes celles et ceux qui m'ont accompagné pendant ces années : Alice, Clément, Fabien, Florian, Ismaël, Justine, Marc-Antoine, Pierre, Philippe, Quentin, Romuald, Sébastien, Sylvain et enfin Vincent. Un grand merci à vous pour tous ces bons moments, vous êtes tous formidables et je sais que je ne pourrai probablement jamais retrouver une cohésion d'équipe semblable à la nôtre. Une petite pensée aussi pour Fabienne / Jérémie, sa vivacité légendaire et son soutien moral indispensable.

Je tiens aussi à remercier plus spécifiquement Clément et Romuald. Au delà de votre accueil lorsque je suis arrivé au MIS, vous avez toujours été disponibles pour me conseiller, m'aider, répondre à mes questions ou sortir boire un verre ! Sans vous, je ne me serais probablement jamais lancé dans cette aventure, donc encore une fois : merci. Mention spécial à Clément pour cette découverte des remparts de Montreuil-sur-Mer, tout simplement inoubliable. Merci aussi à Jordan, Fabien, Philippe et Sylvain pour toutes nos soirées, absolument indispensables pour garder le moral tout au long de ces années ! J'ai une pensée particulière également pour Charlotte et

pour nos bons moments passés ensemble, que ce soit aux Pint of Science ou autour d'un bon sandwich le vendredi midi !

Évidemment, je veux adresser quelques mots à ma famille, qui m'a soutenu au cours de ces longues années de thèse. Mes parents tout d'abord : vous avez fait de moi ce que je suis aujourd'hui et j'espère que vous êtes aussi fiers de moi que je suis fier de vous. Vous étiez là, toujours, dans les bons moments comme dans les mauvais, et c'est grâce à vous que j'ai pu réussir. Ensuite ma soeur, mon beau-frère et mon adorable nièce ensuite, eux aussi toujours là pour me remonter le moral et m'aider quand j'en avais besoin. Je remercie aussi mes grands-mères pour leur gentillesse et leur patience pendant tout ce temps. Une pensée spéciale pour mon parrain, qui m'a fait découvrir l'informatique quand j'avais trois ans et qui est finalement un peu à l'origine de tout ça ! Merci à vous tous, je ne serai pas là où je suis aujourd'hui sans vous, et même si je ne vous le dis pas assez : je vous aime.

Enfin, j'aimerais adresser les derniers mots de ces remerciements à mes grands-pères, Daniel et Roland, partis avant le début et après la fin de cette thèse. C'est aussi grâce à vous que j'en suis là, et vous me manquez. Encore merci à vous deux.

Bibliographie

Abbasi, B., Shadrokh, S., and Arkat, J. (2006). Bi-objective Resource-Constrained Project Scheduling with Robustness and Makespan Criteria. *Applied Mathematics and Computation*, 180(1) :146–152.

Abdalkareem, Z. A., Amir, A., Al-Betar, M. A., Ekhan, P., and Hammouri, A. I. (2021). Healthcare Scheduling in Optimization Context : a review. *Health and Technology*, pages 1–25.

Abobaker, R. A., Ayob, M., and Hadwan, M. (2011). Greedy Constructive Heuristic and Local Search Algorithm for Solving Nurse Rostering Problems. In *2011 3rd Conference on Data Mining and Optimization (DMO)*, pages 194–198. IEEE.

Afshar, M. R., Shahhosseini, V., and Sebt, M. H. (2019). A Genetic Algorithm with a New Local Search Method for Solving the Multimode Resource-Constrained Project Scheduling Problem. *International Journal of Construction Management*, pages 1–9.

Aiken, L. H., Clarke, S. P., Sloane, D. M., Sochalski, J., and Silber, J. H. (2002). Hospital Nurse Staffing and Patient Mortality, Nurse Burnout, and Job Dissatisfaction. *Jama*, 288(16) :1987–1993.

Alcaraz, J. and Maroto, C. (2001). A Robust Genetic Algorithm for Resource Allocation in Project Scheduling. *Annals of Operations Research*, 102(1) :83–109.

Alipouri, Y. (2021). A Resource Flow-based Branch-and-Bound Algorithm to Solve Fuzzy Stochastic Resource-Constrained Project Scheduling Problem. *Soft Computing*, 25(22) :14315–14331.

Ansarifar, J., Tavakkoli-Moghaddam, R., Akhavizadegan, F., and Hassanzadeh Amin, S. (2018). Multi-objective Integrated Planning and Scheduling Model for Operating Rooms under Uncertainty. *Proceedings of the Institution of Mechanical Engineers, Part H : Journal of Engineering in Medicine*, 232(9) :930–948.

Ansótegui, C., Sellmann, M., and Tierney, K. (2009). A Gender-based Genetic Algorithm for the Automatic Configuration of Algorithms. In *International Conference on Principles and Practice of Constraint Programming*, pages 142–157. Springer.

Anthony, R. N. (1965). *Planning and Control Systems : a Framework for Analysis*. Division of Research, Graduate School of Business Administration, Harvard.

Artigues, C., Koné, O., Lopez, P., and Mongeau, M. (2015). Mixed-Integer Linear Programming Formulations. In *Handbook on Project Management and Scheduling Vol. 1*, pages 17–41. Springer.

Azadeh, A., Baghersad, M., Farahani, M. H., and Zarrin, M. (2015). Semi-online Patient Scheduling in Pathology Laboratories. *Artificial Intelligence in Medicine*, 64(3) :217–226.

Azadeh, A., Farahani, M. H., Torabzadeh, S., and Baghersad, M. (2014). Scheduling Prioritized Patients in Emergency Department Laboratories. *Computer Methods and Programs in Biomedicine*, 117(2) :61–70.

Bagherinejad, J. and Majd, Z. R. (2014). Solving the MRCPSP/max with the Objective of Minimizing Tardiness/Earliness Cost of Activities with Double Genetic Algorithms. *The International Journal of Advanced Manufacturing Technology*, 70(1) :573–582.

Bailey, N. T. (1952). A Study of Queues and Appointment Systems in Hospital Out-patient Departments, with Special reference to waiting-times. *Journal of the Royal Statistical Society : Series B (Methodological)*, 14(2) :185–199.

Bartusch, M., Möhring, R. H., and Radermacher, F. J. (1988). Scheduling Project Networks with Resource Constraints and time Windows. *Annals of operations Research*, 16(1) :199–240.

Berg, B. and Denton, B. T. (2012). Appointment Planning and Scheduling in Outpatient Procedure Centers. In *Handbook of Healthcare System Scheduling*, pages 131–154. Springer.

Berthaut, F., Pellerin, R., Perrier, N., and Hajji, A. (2014). Time-cost Trade-offs in Resource-Constraint Project Scheduling Problems with Overlapping Modes. *International Journal of Project Organisation and Management*, 6(3) :215–236.

Bikker, I. A., Kortbeek, N., van Os, R. M., and Boucherie, R. J. (2015). Reducing Access Times for Radiation Treatment by Aligning the Doctor's Schemes. *Operations research for health care*, 7 :111–121.

Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., et al. (2002). A Racing Algorithm for Configuring Metaheuristics. In *Gecco*, volume 2.

Blazewicz, J., Lenstra, J. K., and Kan, A. R. (1983). Scheduling Subject to Resource Constraints : Classification and Complexity. *Discrete Applied Mathematics*, 5(1) :11–24.

Blum, C. and Merkle, D. (2008). *Swarm Intelligence : Introduction and Applications*. Springer Science & Business Media.

Booker, L. B., Goldberg, D. E., and Holland, J. H. (1989). Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, 40(1-3) :235–282.

Booth, R. Z. (2002). The Nursing Shortage : A Worldwide Problem. *Revista Latinoamericana de Enfermagem*, 10 :392–400.

Böttcher, J., Drexl, A., Kolisch, R., and Salewski, F. (1999). Project Scheduling Under Partially Renewable Resource Constraints. *Management Science*, 45(4) :543–559.

Bowers, J., Lyons, B., Mould, G., and Symonds, T. (2005). Modelling Outpatient Capacity for a Diagnosis and Treatment Centre. *Health Care Management Science*, 8(3) :205–211.

Burke, E. K., De Causmaecker, P., Berghe, G. V., and Van Landeghem, H. (2004). The State of the Art of Nurse Rostering. *Journal of Scheduling*, 7(6) :441–499.

Cardoen, B. and Demeulemeester, E. (2008). Capacity of Clinical Pathways — A Strategic Multi-level Evaluation Tool. *Journal of Medical Systems*, 32(6) :443–452.

Carlier, J. and Moukrim, A. (2015). Storage Resources. In *Handbook on Project Management and Scheduling Vol. 1*, pages 177–189. Springer.

Ceschia, S., Guido, R., and Schaerf, A. (2020). Solving the Static INRC-II Nurse Rostering Problem by Simulated Annealing Based on Large Neighborhoods. *Annals of Operations Research*, 288(1) :95–113.

Ceschia, S. and Schaerf, A. (2011). Local Search and Lower Bounds for the Patient Admission Scheduling Problem. *Computers & Operations Research*, 38(10) :1452–1463.

Ceschia, S. and Schaerf, A. (2012). Modeling and Solving the Dynamic Patient Admission Scheduling Problem under Uncertainty. *Artificial Intelligence in Medicine*, 56(3) :199–205.

Ceschia, S. and Schaerf, A. (2016). Dynamic Patient Admission Scheduling with Operating Room Constraints, Flexible Horizons, and Patient Delays. *Journal of Scheduling*, 19(4) :377–389.

Chakrabortty, R. K., Sarker, R. A., and Essam, D. L. (2016). Multi-Mode Resource Constrained Project Scheduling under Resource Disruptions. *Computers & Chemical Engineering*, 88 :13–29.

Chambers, L. D. (2000). *The Practical Handbook of Genetic Algorithms : Applications*. Chapman and Hall/CRC.

Chen, W.-N., Zhang, J., Liu, O., and Liu, H.-l. (2010). A Monte-Carlo Ant Colony System for Scheduling Multi-mode Projects with Uncertainties to Optimize Cash Flows. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE.

Chern, C.-C., Chien, P.-S., and Chen, S.-Y. (2008). A Heuristic Algorithm for the Hospital Health Examination Scheduling Problem. *European Journal of Operational Research*, 186(3) :1137–1157.

Colorni, A., Dorigo, M., Maniezzo, V., et al. (1991). Distributed Optimization by Ant Colonies. In *Proceedings of the First European Conference on Artificial Life*, volume 142, pages 134–142. Paris, France.

Conforti, D., Guerriero, F., Guido, R., Cerinic, M. M., and Conforti, M. L. (2011). An Optimal Decision Making Model for Supporting Week Hospital Management. *Health Care Management Science*, 14(1) :74–88.

Cook, S. A. (1971). The Complexity of Theorem-proving Procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158.

Crosby, J. L. et al. (1973). *Computer simulation in genetics.* John Wiley & Sons.

Dang, N. T. T., Ceschia, S., Schaerf, A., De Causmaecker, P., and Haspeslagh, S. (2016). Solving the Multi-stage Nurse Rostering Problem. In *Proceedings of*

the 11th International Conference of the Practice and Theory of Automated Timetabling, pages 473–475.

Dantzig, G. B., Orden, A., Wolfe, P., et al. (1955). The Generalized Simplex Method for Minimizing a Linear Form under Linear inequality Restraints. *Pacific Journal of Mathematics*, 5(2) :183–195.

Dantzig, G. B. and Wolfe, P. (1960). Decomposition Principle for Linear Programs. *Operations Research*, 8(1) :101–111.

Darwin, C. (1859). *On the Origin of Species.* John Murray.

Davari, M. and Demeulemeester, E. (2019). A Novel Branch-and-Bound Algorithm for the Chance-constrained Resource-Constrained Project Scheduling Problem. *International Journal of Production Research*, 57(4) :1265–1282.

de Lamarck, J.-B. (1809). *Philosophie Zoologique.* Musée d'Histoire Naturelle.

Debels, D. and Vanhoucke, M. (2005). A Bi-population Based Genetic Algorithm for the Resource-Constrained Project Scheduling Problem. In *International Conference on Computational Science and its Applications*, pages 378–387. Springer.

Deblaere, F., Demeulemeester, E., and Herroelen, W. (2011). Reactive Scheduling in the Multi-mode RCPSP. *Computers & Operations Research*, 38(1) :63–74.

Demeester, P., Souffriau, W., De Causmaecker, P., and Berghe, G. V. (2010). A Hybrid Tabu Search Algorithm for Automatically Assigning Patients to Beds. *Artificial Intelligence in Medicine*, 48(1) :61–70.

Demeulemeester, E. L. and Herroelen, W. S. (1996). An Efficient Optimal Solution Procedure for the Preemptive Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 90(2) :334–348.

Denton, B., Viapiano, J., and Vogl, A. (2007). Optimization of Surgery Sequencing and Scheduling Decisions under Uncertainty. *Health Care Management Science*, 10(1) :13–24.

Du, G., Jiang, Z., Yao, Y., and Diao, X. (2013). Clinical Pathways Scheduling Using Hybrid Genetic Algorithm. *Journal of Medical Systems*, 37(3) :1–17.

Elmaghraby, S. E. (1977). *Activity Networks : Project Planning and Control by Network Models.* John Wiley & Sons.

Engelbrecht, A. P. (2007). *Computational Intelligence : an Introduction.* John Wiley & Sons.

Fraser, A. S. (1957). Simulation of Genetic Systems by Automatic Digital Computers. *Australian journal of biological sciences*, 10(4) :484–491.

Froehle, C. M. and Magazine, M. J. (2013). Improving Scheduling and Flow in Complex Outpatient Clinics. In *Handbook of Healthcare Operations Management*, pages 229–250. Springer.

Gallo, C. and Capozzi, V. (2019). A Simulated Annealing Algorithm for Scheduling Problems. *Journal of Applied Mathematics and Physics*, 7(11) :2579–2594.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability*, volume 174. Freeman San Francisco.

Geay, C. and de Lagasnerie, G. (2013). *Projection des Dépenses de Santé à l'Horizon 2060, le Modèle PROMEDE.* Direction Générale du Trésor.

Gnägi, M., Zimmermann, A., and Trautmann, N. (2018). A Continuous-time Unit-based MILP Formulation for the Resource-Constrained Project Scheduling Problem. In *2018 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 432–436. IEEE.

Golberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. *Addion wesley*, 1989(102) :36.

Goldberg, D. E. and Deb, K. (1991). A Comparative Analysis of Selection Schemes used in Genetic Algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy Genetic Algorithms : Motivation, Analysis, and First Results. *Complex systems*, 3(5) :493–530.

Gonzalez, L., Levebvre, G., Mikou, M., and Portela, M. (2021). *Les Dépenses de Santé en 2020 - Résultats des Comptes de la Santé - Édition 2021.* DREES.

Granja, C., Almada-Lobo, B., Janela, F., Seabra, J., and Mendes, A. (2014). An Optimization Based on Simulation Approach to the Patient Admission Scheduling Problem Using a Linear Programing Algorithm. *Journal of Biomedical Informatics*, 52 :427–437.

Guerriero, F. and Guido, R. (2011). Operational Research in the Management of the Operating Theatre : a Survey. *Health care management science*, 14(1) :89–114.

Gupta, D. and Denton, B. (2008). Appointment Scheduling in Health Care : Challenges and Opportunities. *IIE Transactions*, 40(9) :800–819.

Gérard, O., Brisoux Devendeville, L., and Lucet, C. (2020). Problème de Planification dans le Domaine de la Santé. In *21ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'20)*, Montpellier, France.

Gérard, O., Brisoux Devendeville, L., and Lucet, C. (2021a). An Adaptive Large Neighborhood Search Method to Plan Patient's Journey in Healthcare. In *17th Advances in Production Management Systems(APMS'21)*, Nantes, France.

Gérard, O., Brisoux Devendeville, L., and Lucet, C. (2021b). Planning Problem in Healthcare Domain. In *17th International Workshop on Project Management and Scheduling (PMS'20)*, Toulouse, France.

Gérard, O., Brisoux Devendeville, L., and Lucet, C. (2021c). Une Méthode ALNS Appliquée à la Planification dans le Domaine de la Santé. In *22ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'21)*, Mulhouse, France.

Gérard, O., Brisoux Devendeville, L., and Lucet, C. (2022). Différentes Mesures de la Diversité de la Population d'un Algorithme Génétique pour l'Optimisation du Parcours Patient. In *23ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF'23)*, Lyon, France.

Habibi, F., Barzinpour, F., and Sadjadi, S. (2018). Resource-Constrained Project Scheduling Problem : Review of Past and Recent Developments. *Journal of Project Management*, 3(2) :55–88.

Hall, R. (2012). Matching Healthcare Resources to Patient Needs. In *Handbook of Healthcare System Scheduling*, pages 1–9. Springer.

Hall, R. W. et al. (2012). *Handbook of Healthcare System Scheduling*. Springer.

Hamming, R. W. (1950). Error Detecting and Error Correcting Codes. *The Bell system technical journal*, 29(2) :147–160.

Hammouri, A. I. and Alrifai, B. (2014). Investigating Biogeography-based Optimisation for Patient Admission Scheduling Problems. *Journal of Theoretical & Applied Information Technology*, 70(3).

Hans, E. W. and Vanberkel, P. T. (2012). Operating Theatre Planning and Scheduling. In *Handbook of Healthcare System Scheduling*, pages 105–130. Springer.

Hariga, M., Shamayleh, A., and El-Wehedi, F. (2019). Integrated Time–cost Tradeoff and Resources Leveling Problems with Allowed Activity Splitting. *International Transactions in Operational Research*, 26(1) :80–99.

Hartmann, S. (1998). A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics (NRL)*, 45(7) :733–750.

Hartmann, S. (2002). A Self-adapting Genetic Algorithm for Project Scheduling under Resource Constraints. *Naval Research Logistics (NRL)*, 49(5) :433–448.

Hartmann, S. and Briskorn, D. (2022). An Updated Survey of Variants and Extensions of the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 297(1) :1–14.

Hartmann, S. and Kolisch, R. (2000). Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 127(2) :394–407.

Haspeslagh, S., De Causmaecker, P., Schaerf, A., and Stølevik, M. (2014). The First International Nurse Rostering Competition 2010. *Annals of Operations Research*, 218(1) :221–236.

He, L., Liu, X., Laporte, G., Chen, Y., and Chen, Y. (2018). An Improved Adaptive Large Neighborhood Search Algorithm for Multiple Agile Satellites Scheduling. *Computers & Operations Research*, 100 :12–25.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems : an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. MIT Press.

Holland John, H. (1975). Adaptation in natural and artificial systems. *Ann Arbor : University of Michigan Press*.

Hoos, H. H. and Stützle, T. (2004). *Stochastic Local Search : Foundations and Applications*. Elsevier.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2010). Automated Configuration of Mixed Integer Programming Solvers. In *International Conference on Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*, pages 186–202. Springer.

Kadri, R. L. and Boctor, F. F. (2018). An Efficient Genetic Algorithm to Solve the Resource-Constrained Project Scheduling Problem with Transfer Times : The Single Mode Case. *European Journal of Operational Research*, 265(2) :454–462.

Kaplan, L. A. (1988). *Resource-Constrained Project Scheduling with Preemption of Jobs*. PhD thesis, University of Michigan.

Ke, H. and Liu, B. (2005). Project Scheduling Problem with Stochastic Activity Duration Times. *Applied Mathematics and Computation*, 168(1) :342–353.

Kendall, M. G. (1938). A New Measure of Rank Correlation. *Biometrika*, 30(1/2) :81–93.

Kirkpatrick, S., Gelatt Jr, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *science*, 220(4598) :671–680.

Kolisch, R. and Hartmann, S. (1999). Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem : Classification and Computational Analysis. In *Project scheduling*, pages 147–178. Springer.

Kolisch, R. and Hartmann, S. (2006). Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling : An Update. *European journal of operational research*, 174(1) :23–37.

Kolisch, R. and Sprecher, A. (1997). Psplib-a project scheduling problem library : Or software-orsep operations research software exchange program. *European journal of operational research*, 96(1) :205–216.

Kolisch, R. and Sprecher, A. (2020). Project scheduling problem library - psplib.

Koné, O., Artigues, C., Lopez, P., and Mongeau, M. (2011). Event-based MILP Models for Resource-Constrained Project Scheduling Problems. *Computers & Operations Research*, 38(1) :3–13.

Korte, B. H., Vygen, J., Korte, B., and Vygen, J. (2011). *Combinatorial Optimization*, volume 1. Springer.

Land, A. H. and Doig, A. G. (2010). An Automatic Method for Solving Discrete Programming Problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. Springer.

Laurent, A., Deroussi, L., Grangeon, N., and Norre, S. (2017). A New Extension of the RCPSP in a Multi-site Context : Mathematical Model and Metaheuristics. *Computers & Industrial Engineering*, 112 :634–644.

Lavinas, Y., Aranha, C., Sakurai, T., and Ladeira, M. (2018). Experimental Analysis of the Tournament Size on Genetic Algorithms. In *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3647–3653. IEEE.

Lee, S. and Yih, Y. (2012). Surgery Scheduling of Multiple Operating Rooms under Uncertainty and Resource Constraints of Post-anesthesia Care Units. In *IIE Annual Conference*, page 1. Institute of Industrial and Systems Engineers (IISE).

Lewis, H. R. and Papadimitriou, C. H. (1998). Elements of the Theory of Computation. *ACM SIGACT News*, 29(3) :62–78.

Liang, B., Turkcan, A., Ceyhan, M. E., and Stuart, K. (2015). Improvement of Chemotherapy Patient Flow and Scheduling in an Outpatient Oncology Clinic. *International Journal of Production Research*, 53(24) :7177–7190.

Lin, Y.-K. and Chou, Y.-Y. (2020). A Hybrid Genetic Algorithm for Operating Room Scheduling. *Health Care Management Science*, 23(2) :249–263.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The iRace Package : Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives*, 3 :43–58.

Lusby, R. M., Schwierz, M., Range, T. M., and Larsen, J. (2016). An Adaptive Large Neighborhood Search Procedure Applied to the Dynamic Patient Admission Scheduling Problem. *Artificial Intelligence in Medicine*, 74 :21–31.

Ma, Z., He, Z., Wang, N., Yang, Z., and Demeulemeester, E. (2018). A Genetic Algorithm for the Proactive Resource-Constrained Project Scheduling Problem with Activity Splitting. *IEEE Transactions on Engineering Management*, 66(3) :459–474.

Marynissen, J. and Demeulemeester, E. (2016). Literature Review on Integrated Hospital Scheduling Problems. *KU Leuven, Faculty of Economics and Business, KBI_1627.*

Mateus, C., Marques, I., and Captivo, M. E. (2018). Local Search Heuristics for a Surgical Case Assignment Problem. *Operations Research for Health Care*, 17 :71–81.

Merkle, D., Middendorf, M., and Schmeck, H. (2002). Ant Colony Optimization for Resource-Constrained Project Scheduling. *IEEE Transactions on Evolutionary Computation*, 6(4) :333–346.

Miller, B. L., Goldberg, D. E., et al. (1995). Genetic Algorithms, Tournament Selection, and the Effects of Noise. *Complex systems*, 9(3) :193–212.

Mischek, F. and Musliu, N. (2019). Integer Programming Model Extensions for a Multi-stage Nurse Rostering Problem. *Annals of Operations Research*, 275(1) :123–143.

Moukrim, A., Quilliot, A., and Toussaint, H. (2015). An Effective Branch-and-price Algorithm for the Preemptive Resource Constrained Project Scheduling Problem based on Minimal Interval Order Enumeration. *European Journal of Operational Research*, 244(2) :360–368.

Muller, L. F. (2009). An Adaptive Large Neighborhood Search Algorithm for the Resource-Constrained Project Scheduling Problem. In *Proceedings of the VIII Metaheuristics International Conference (MIC)*.

Muller, L. F. (2011). An Adaptive Large Neighborhood Search Algorithm for the Multi-Mode Resource-Constrained Project Scheduling Problem.

Myszkowski, P. B., Olech, Ł. P., Laszczyk, M., and Skowroński, M. E. (2018). Hybrid differential evolution and greedy algorithm (degr) for solving multi-skill resource-constrained project scheduling problem. *Applied Soft Computing*, 62 :1–14.

Nannen, V. and Eiben, A. E. (2006). A Method for Parameter Calibration and Relevance Estimation in Evolutionary Algorithms. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 183–190.

Néron, E. and Baptista, D. (2002). Heuristics for the Multi-Skill Project Scheduling Problem. In *International Symposium on Combinatorial Optimization (CO'2002)*.

Palacio, J. D. and Larrea, O. L. (2017). A Lexicographic Approach to the Robust Resource-Constrained Project Scheduling Problem. *International Transactions in Operational Research*, 24(1-2) :143–157.

Palpant, M., Artigues, C., and Michelon, P. (2004). LSSPER : Solving the Resource-Constrained Project Scheduling Problem with Large Neighbourhood Search. *Annals of Operations Research*, 131(1) :237–257.

Pang, J., Zhou, H., Tsai, Y.-C., and Chou, F.-D. (2018). A Scatter Simulated Annealing Algorithm for the Bi-objective Scheduling Problem for the Wet Station of Semiconductor Manufacturing. *Computers & Industrial Engineering*, 123 :54–66.

Papadimitriou, C. H. and Steiglitz, K. (1998). *Combinatorial Optimization : Algorithms and Complexity*. Courier Corporation.

Pariente, J. M. M. (2016). *Operating Theatre Planning and Scheduling in Real-life Settings : Problem Analysis, Models, and Solution Procedures*. PhD thesis, Universidad de Sevilla.

Pellerin, R., Perrier, N., and Berthaut, F. (2020). A Survey of Hybrid Metaheuristics for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 280(2) :395–416.

Pritsker, A. A. B., Waiters, L. J., and Wolfe, P. M. (1969). Multiproject Scheduling with Limited Resources : A Zero-one Programming Approach. *Management Science*, 16(1) :93–108.

Rajeswari, M., Amudhavel, J., Pothula, S., and Dhavachelvan, P. (2017). Directed Bee Colony Optimization Algorithm to Solve the Nurse Rostering Problem. *Computational Intelligence and Neuroscience*, 2017.

Reyck, B. D., Demeulemeester, E., and Herroelen, W. (1999). Algorithms for Scheduling Projects with Generalized Precedence Relations. In *Project Scheduling*, pages 77–105. Springer.

Ropke, S. and Pisinger, D. (2006). An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4) :455–472.

Sastry, K., Goldberg, D., and Kendall, G. (2005). Genetic Algorithms. In *Search methodologies*, pages 97–125. Springer.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to solve Vehicle Routing Problems. In *International Conference on Principles and Practice of Constraint Programming*, pages 417–431. Springer.

Singh, G. and Weiskircher, R. (2011). A Multi-agent System for Decentralised Fractional Shared Resource Constraint Scheduling. *Web Intelligence and Agent Systems : An International Journal*, 9(2) :99–108.

Słowiński, R. (1980). Two Approaches to Problems of Resource Allocation among Project Activities — A Comparative Study. *Journal of the Operational Research Society*, 31(8) :711–723.

Smith, S. F. (1995). Reactive Scheduling Systems. In *Intelligent scheduling systems*, pages 155–192. Springer.

Squillero, G. and Tonda, A. (2016). Divergence of Character and Premature Convergence : A Survey of Methodologies for Promoting Diversity in Evolutionary Optimization. *Information Sciences*, 329 :782–799.

Timucin, T. and Birogul, S. (2018). Implementation of Operating Room Scheduling with Genetic Algorithm and the Importance of Repair Operator. In *2018 2nd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, pages 1–6. IEEE.

Timuçin, T. and Biroğul, S. (2019). Effect the Number of Reservations on Implementation of Operating Room Scheduling with Genetic Algorithm. In *The International Conference on Artificial Intelligence and Applied Mathematics in Engineering*, pages 252–265. Springer.

Ting, T., Yang, X.-S., Cheng, S., and Huang, K. (2015). Hybrid Metaheuristic Algorithms : Past, Present, and Future. *Recent Advances in Swarm Intelligence and Evolutionary Computation*, pages 71–83.

Tsai, P.-F. J. and Teng, G.-Y. (2014). A Stochastic Appointment Scheduling System on Multiple Resources with Dynamic Call-in Sequence and Patient No-shows for an Outpatient Clinic. *European Journal of Operational Research*, 239(2) :427–436.

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, LIX(236) :433–460.

Turing, A. M. et al. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *J. of Math*, 58(345-363) :5.

Turkeš, R., Sörensen, K., and Hvattum, L. M. (2021). Meta-analysis of Metaheuristics : Quantifying the Effect of Adaptiveness in Adaptive Large Neighborhood Search. *European Journal of Operational Research*, 292(2) :423–442.

Valls, V., Laguna, M., Lino, P., Pérez, A., and Quintanilla, S. (1999). Project Scheduling with Stochastic Activity Interruptions. In *Project Scheduling*, pages 333–353. Springer.

Van Huele, C. and Vanhoucke, M. (2014). Analysis of the Integration of the Physician Rostering Problem and the Surgery Scheduling Problem. *Journal of Medical Systems*, 38(6) :1–16.

Vanhoucke, M. (2010). A Scatter Search Heuristic for Maximising the Net Present Value of a Resource-Constrained Project with Fixed Activity Cash Flows. *International Journal of Production Research*, 48(7) :1983–2001.

Vermeulen, I. B., Bohte, S. M., Elkhuizen, S. G., Bakker, P. J., and La Poutré, H. (2008). Decentralized Online Scheduling of Combination-appointments in Hospitals. In *ICAPS*, pages 372–379.

Viana, A. and de Sousa, J. P. (2000). Using Metaheuristics in Multiobjective Resource Constrained Project Scheduling. *European Journal of Operational Research*, 120(2) :359–374.

Vieira, G. E., Herrmann, J. W., and Lin, E. (2000). Analytical Models to Predict the Performance of a Single-machine System under Periodic and Event-driven Rescheduling Strategies. *International Journal of Production Research*, 38(8) :1899–1915.

Villafáñez, F. A., Poza, D., López-Paredes, A., and Pajares, J. (2018). A Unified Nomenclature for Project Scheduling Problems (RCPSP and RCMPSP). *Dirección y Organización*, 64 :56–60.

Wang, Q., Liu, C., and Zheng, L. (2019). A Column-generation-based Algorithm for a Resource-Constrained Project Scheduling Problem with a Fractional Shared Resource. *Engineering Optimization*.

Watermeyer, K. and Zimmermann, J. (2020). A Branch-and-Bound Procedure for the Resource-Constrained Project Scheduling Problem with Partially Renewable Resources and General Temporal Constraints. *OR Spectrum*, 42(2) :427–460.

Wolsey, L. A. and Nemhauser, G. L. (1999). *Integer and Combinatorial Optimization*, volume 55. John Wiley & Sons.

Wu, J.-j., Lin, Y., Zhan, Z.-h., Chen, W.-n., Lin, Y.-b., and Chen, J.-y. (2013). An Ant Colony Optimization Approach for Nurse Rostering Problem. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1672–1676. IEEE.

Wu, S. D., Storer, R. H., and Pei-Chann, C. (1993). One-machine Rescheduling Heuristics with Efficiency and Stability as Criteria. *Computers & Operations Research*, 20(1) :1–14.

Xiang, W. (2017). A Multi-objective ACO for Operating Room Scheduling Optimization. *Natural Computing*, 16(4) :607–617.

Zamani, R. (2010). An Accelerating Two-Layer Anchor Search with Application to the Resource-Constrained Project Scheduling Problem. *IEEE Transactions on Evolutionary Computation*, 14(6) :975–984.

Zamani, R. (2017). An Evolutionary Implicit Enumeration Procedure for Solving the Resource-Constrained Project Scheduling Problem. *International Transactions in Operational Research*, 24(6) :1525–1547.

Zhu, S., Fan, W., Liu, T., Yang, S., and Pardalos, P. M. (2020). Dynamic Three-stage Operating Room Scheduling Considering Patient Waiting Time and Surgical Overtime Costs. *Journal of Combinatorial Cptimization*, 39(1) :185–215.

Zhuo, X., Huang, H., Cai, Z., and Hu, H. (2015). An Hybrid Evolutionary Algorithm with Scout Bee Global Search Strategy for Chinese Nurse Rostering Problems. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 769–775. IEEE.

Zonderland, M. E. (2014). *Appointment Planning in Outpatient Clinics and Diagnostic Facilities*. Springer.

LORH : un outil pour la planification du parcours patient dans le milieu hospitalier

Résumé :

Dans cette thèse, nous nous sommes intéressés à un problème de planification du parcours de soin du patient, proposé par l'entreprise Evolucare Technologies. L'objectif de l'entreprise est de fournir un logiciel capable de produire des plannings respectant les diverses contraintes et répondant au mieux aux souhaits des patients et des équipes soignantes. Le projet LORH incarne la réponse d'Evolucare à ce problème, que nous avons étudié et pour lequel nous avons proposé plusieurs méthodes de résolution. Ce problème NP-difficile est proche du Resource Constraint Project Scheduling Problem (RCPSP), une problématique reconnue de la littérature.

Nous avons en premier lieu décrit formellement la problématique et élaboré la fonction objectif. Notre première approche est un modèle de programmation linéaire en 0-1 intégrant l'ensemble des variables et des contraintes du problème. Les solutions initiales requises pour nos méthodes de résolution ont été obtenues avec un algorithme de construction aléatoire nommé LORH_RCA. Notre seconde approche, LORH_ALNS, est une recherche locale basée sur l'Adaptive Large Neighborhood Search (ALNS) utilisant un ensemble de mouvements développés spécifiquement pour ce problème. Nous avons ensuite proposé un algorithme génétique, LORH_GA, et des opérations de croisement et de mutation adaptés à notre problématique. Enfin, la dernière méthode, nommée LORH_GADM, est également un algorithme génétique, avec une gestion de la diversité des solutions pour limiter le phénomène de convergence précoce vers des optimums locaux.

Nous avons évalué ces méthodes sur un ensemble d'instances générées à partir de problèmes rapportés par Evolucare. Le modèle de programmation linéaire, implémenté sous CPLEX, nous a permis d'obtenir la solution optimale sur certaines de nos instances. Nous avons ensuite obtenu avec LORH_ALNS des solu-

tions optimales supplémentaires et de meilleures bornes supérieures sur toutes nos familles d'instances. Ces résultats ont été améliorés successivement par LORH_GA et LORH_GADM avec un gain de 16.42% entre LORH_ALNS et LORH_GADM. Nous avons également évalué LORH_ALNS, LORH_GA et LORH_GADM sur les instances de la littérature RCPSP et obtenu un écart moyen de 5.4% avec les solutions optimales avec nos algorithmes génétiques LORH_GA et LORH_GADM.

LORH : a tool for planning patient journey in hospital environment

Abstract :

In this thesis, we studied a problem of patient care planning, proposed by the Evolucare Technologies company. The company aims to provide a software capable of producing schedules that respect the various constraints and meet the needs of patients and care teams. The LORH project is Evolucare's answer to this issue. We studied the problem and proposed several resolution methods. This NP-hard problem is close to the Resource Constraint Project Scheduling Problem (RCPSP), a well-known problem in the literature.

We first formally described the problem and developed the objective function. Our first approach is a 0-1 linear programming model incorporating all variables and constraints of our problem. The initial solutions required for other resolution methods were obtained with a randomized construction algorithm named LORH_RCA. Our second approach LORH_ALNS is a local search based on Adaptive Large Neighborhood Search (ALNS) using a set of moves specifically designed for this problem. We then proposed a genetic algorithm LORH_GA with crossover and mutation operators designed for our problem. The last method named LORH_GADM is also a genetic algorithm with solution diversity management in order to reduce the early convergence to local optimums.

We evaluated these approaches on a set of instances generated from problems reported by Evolucare. The linear programming model implemented under CPLEX allowed us to obtain optimal solutions on some instances. We then obtained with LORH_ALNS additional optimal solutions and better upper bounds for all our instance families. These results were successively improved by LORH_GA and LORH_GADM with a gain of 16.42% between LORH_ALNS and LORH_GADM. We also evaluated LORH_ALNS, LORH_GA and LORH_GADM on the RCPSP

literature instances and obtained an average difference of 5.4% with the optimal solutions with LORH_GA and LORH_GADM.