République algérienne démocratique et populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Akli Mohand Oulhadj de BOUIRA
Faculté des Sciences Exactes
Département de mathématiques

Mémoire de Master

En mathématiques

Spécialité : Recherche Opérationnelle

Thème

Optimisation de la planification du personnel infirmier : approches exactes et métaheuristiques

---

Réalisé par :
- MADANI MAROUA

Devant le jury composé de :

Nom	Établissement	Grade	Qualité
Hamid Karim	Université Akli Mohand Oulhadj	MCB	Président
Louadj Kahina	Université Akli Mohand Oulhadj	MCA	Examinatrice
Bekri Houria	Université Akli Mohand Oulhadj	MAA	Examinatrice
Imine Ouiza	Université Akli Mohand Oulhadj	MAA	Promotrice

Année académique 2025 / 2026

Résumé

Le problème de planification des horaires des infirmiers, connu sous le nom de *Nurse Rostering Problem* (NRP), constitue un cas particulier de problème d'optimisation combinatoire visant à construire des plannings efficaces tout en respectant un ensemble de contraintes organisationnelles, légales et individuelles. La diversité des exigences, telles que les besoins en personnel, les règles hospitalières et les préférences des infirmiers, rend ce problème particulièrement complexe et difficile à résoudre de manière exacte, notamment pour les grandes instances.

Dans ce travail, une modélisation mathématique du NRP basée sur la programmation linéaire en nombres entiers (MILP) est proposée afin de représenter fidèlement les contraintes du problème. En complément, deux approches métaheuristiques de trajectoire sont développées : la recherche locale (*Local Search*), permettant d'améliorer progressivement une solution initiale, et le recuit simulé (*Simulated Annealing*), offrant une meilleure exploration de l'espace des solutions.

Les approches proposées ont été évaluées sur plusieurs instances de tailles variées (petite, moyenne, grande et très grande). Les résultats expérimentaux montrent que le recuit simulé produit des solutions de meilleure qualité que la recherche locale, notamment sur les instances de grande taille, au prix d'un temps de calcul légèrement supérieur. L'analyse comparative met en évidence un compromis entre la qualité des solutions et le temps d'exécution.

Ces résultats confirment l'efficacité des approches proposées pour la résolution du NRP et soulignent l'intérêt des méthodes d'optimisation dans l'amélioration de la planification hospitalière.

Mots-clés : *Nurse Rostering Problem* (NRP), planification des infirmiers, MILP, recherche locale, recuit simulé, métaheuristiques, optimisation combinatoire.

Abstract

The *Nurse Rostering Problem* (NRP) is a specific type of combinatorial optimization problem that aims to construct efficient work schedules while satisfying a set of organizational, legal, and individual constraints. The diversity of requirements, including staffing needs, hospital regulations, and nurses' preferences, makes this problem particularly complex and difficult to solve exactly, especially for large-scale instances.

In this work, a mathematical model based on Mixed-Integer Linear Programming (MILP) is proposed to accurately represent the problem and its constraints. In addition, two trajectory-based metaheuristic approaches are developed : *Local Search*, which iteratively improves an initial solution, and *Simulated Annealing*, which allows a more effective exploration of the solution space.

The proposed approaches are evaluated on several instances of varying sizes (small, medium, large, and very large). Experimental results show that *Simulated Annealing* produces higher-quality solutions than *Local Search*, especially on large instances, at the cost of slightly higher computation times. The comparative analysis highlights the trade-off between solution quality and execution time.

These results confirm the effectiveness of the proposed approaches for solving the NRP and emphasize the importance of optimization techniques in improving hospital staff scheduling.

Keywords : *Nurse Rostering Problem* (NRP), nurse scheduling, Mixed-Integer Linear Programming (MILP), *Local Search*, *Simulated Annealing*, metaheuristics, combinatorial optimization.

Remerciements

Tout d'abord, je remercie Dieu Tout-Puissant de m'avoir accordé la force, la patience et la persévérance nécessaires pour mener à bien ce travail.

Je tiens à exprimer ma profonde gratitude à ma directrice de mémoire, Madame **Imine Ouiza** , pour son encadrement exemplaire, sa disponibilité constante, ses conseils précieux et son soutien tout au long de ce travail.

Monsieur le présedint **Hamid Karim** , pour avoir accepté de juger ce travail et pour l'attention qu'il y a portée.

Je souhaite adresser un remerciement tout particulier, sincère et chaleureux à Madame **Louadj Kahina**. Je lui exprime toute ma reconnaissance pour son aide précieuse, sa bienveillance, ses encouragements constants et l'intérêt qu'elle a toujours porté à mon travail.

Madame **Bekri Houria**, pour avoir accepté d'examiner ce travail et pour ses remarques pertinentes.

Je tiens à remercier profondément mes chers parents, pour leur amour inconditionnel, leur soutien moral et financier, leurs sacrifices et leurs prières.

Je remercie également ma chère sœur et mes deux chers frères pour leur soutien et leurs encouragements constants.

Je n'oublie pas de remercier tous les membres du comité d'encadrement pour leur disponibilité et leur contribution à l'évaluation de ce travail.

Enfin, je tiens à remercier toutes mes amies et mes collègues, pour leurs sympathies et leurs amitiés en leur souhaitant une bonne continuation et une bonne chance.

Dédicace

À mes très chers parents,

À ma mère, pour ses prières incessantes, sa patience infinie, son amour inconditionnel et son soutien moral qui m'ont accompagnée à chaque instant de mon parcours, et qui ont été ma plus grande source de force et de courage.

À mon père, pour ses sacrifices, son travail acharné, ses conseils précieux et son soutien permanent qui m'ont permis de poursuivre mes études dans les meilleures conditions.

À ma chère petite sœur ***Chaïma****, pour sa douceur, sa joie de vivre et sa présence qui illumine ma vie au quotidien.*

À mes deux grands frères ***Redouane*** *et* ***Sidi Ali****, pour leur protection, leur aide, leurs conseils et leur soutien constant dans les moments faciles comme difficiles.*

À toute ma famille, sans exception, , ainsi que tous mes proches, pour leur affection, leur soutien et leurs encouragements tout au long de mon parcours.

À mes très chères amies :

Ahlam, Naïma, Hassiba, Massilia, Chourouk, Zineb, Fatma, Rawia et Hnan*,*

pour leur sincère amitié, leur présence, leur soutien moral et tous les beaux moments partagés ensemble durant ces années d'études. Votre amitié restera à jamais gravée dans mon cœur.

Table des matières

Table des figures

Liste des tableaux

Liste des symboles

Symbole	**Signification**
I	Ensemble des infirmiers
N	Nombre total d'infirmiers
S	Ensemble des shifts
T	Ensemble des jours de planification
$x_{i,s,t}$	Variable binaire d'affectation
$y_{i,t}$	Variable de repos
$z_{i,t}$	Variable de séquence de repos
W_i	Charge de travail de l'infirmier i
$\overline{W}$	Charge moyenne de travail
V_h	Nombre de violations dures
V_s	Nombre de violations molles
σ	Écart-type
Z	Fonction objectif
T_0	Température initiale (SA)
α	Facteur de refroidissement
M	Constante Big-M
h_s	Durée en heures du shift $s \in S$
$\text{minStaff}_{s,t}$	Nombre minimum d'infirmiers requis pour le shift s le jour t
$\text{maxStaff}_{s,t}$	Nombre maximum d'infirmiers autorisés pour le shift s le jour t

maxH_i^w	Nombre maximum d'heures de travail par semaine pour l'infirmier i
maxH_i^m	Nombre maximum d'heures de travail par mois pour l'infirmier i
minRest	Nombre minimal de jours de repos consécutifs par semaine
$\text{pref}_{i,s,t}$	Vaut 1 si l'infirmier i souhaite travailler le shift s le jour t, 0 sinon
$p_{i,s,t}$	Pénalité associée à l'affectation de l'infirmier i au shift s le jour t contre sa préférence
$\alpha, \beta, \gamma \geq 0$	Coefficients de pondération de la fonction objectif, avec $\alpha + \beta + \gamma = 1$
u_i	Déviation de la charge de travail de l'infirmier i par rapport à la moyenne
$\varepsilon_{i,s,t}$	Écart entre la préférence souhaitée et l'affectation réelle

Notation et abréviations

Abréviation	**Signification**
NRP	**N**urse **R**ostering **P**roblem
MILP	**M**ixed **I**nteger **L**inear **P**rogramming
ILP	**I**nteger **L**inear **P**rogramming
LP	**L**inear **P**rogramming
B&B	**B**ranch and **B**ound
B&C	**B**ranch and **C**ut
SA	**S**imulated **A**nnealing (Recuit simulé)
LS	**L**ocal **S**earch (Recherche locale)
INRC	**I**nternational **N**urse **R**ostering **C**ompetition
NSPLib	**N**urse **S**cheduling **P**roblem **Lib**rary
FO	**F**onction **O**bjectif
PO	**P**roblème d'**O**ptimisation
PL	**P**rogrammation **L**inéaire
PB	**P**rogramme **B**inaire

Introduction générale

Dans le domaine hospitalier, la gestion efficace des ressources humaines constitue un enjeu majeur, tant sur le plan économique que sur le plan organisationnel. La planification des équipes d'infirmiers représente une tâche particulièrement complexe en raison de la diversité des contraintes à respecter, telles que la disponibilité du personnel, les besoins des services, ainsi que les règles légales et internes de travail.

La gestion du personnel infirmier représente également un enjeu économique de premier plan. En effet, le coût salarial du personnel soignant constitue la part la plus importante du budget de fonctionnement hospitalier. Par ailleurs, le ratio patient/infirmier est un indicateur essentiel de la qualité des soins et de la charge de travail : un déséquilibre dans ce ratio peut entraîner une surcharge de travail, une baisse de la performance et une dégradation de la qualité des soins.

Dans la pratique, la planification manuelle des plannings d'infirmiers reste encore largement utilisée dans plusieurs établissements. Cependant, cette approche est chronophage, sujette aux erreurs humaines et peut conduire à une répartition inéquitable de la charge de travail entre les infirmiers, ce qui impacte négativement leur motivation et l'efficacité du service.

Face à ces limites, il devient nécessaire de recourir à des approches plus avancées basées sur l'automatisation et les techniques d'optimisation mathématique. Ces méthodes permettent de modéliser le problème de manière rigoureuse et de générer des plannings de bonne qualité tout en respectant les différentes contraintes organisationnelles, légales et individuelles.

Le problème de planification du personnel infirmier suscite l'intérêt de la communauté de la recherche opérationnelle depuis plus de cinq décennies. Les premiers travaux, apparus dans les années 1970, reposaient essentiellement sur des modèles de programmation linéaire en nombres entiers et visaient à formaliser mathématiquement le problème afin d'obtenir des solutions optimales. Toutefois, ces approches exactes se sont rapidement heurtées à la complexité combinatoire du problème, particulièrement lorsque la taille des instances augmentait.

Dans les années 1990, face aux limites des méthodes exactes, sont apparues des méthodes stochastiques et heuristiques, telles que le recuit simulé, capables de produire des solutions de bonne qualité dans des temps de calcul raisonnables. Cette période

marque une transition importante vers des approches plus flexibles et adaptables à la diversité des contextes hospitaliers.

À partir de la fin des années 1990 et au début des années 2000, les métaheuristiques (recherche locale, algorithmes génétiques, recherche tabou) sont devenues dominantes dans la littérature, notamment grâce à l'apparition de bibliothèques de référence comme NSPLib permettant des comparaisons standardisées entre algorithmes. Plus récemment, les recherches se sont orientées vers des approches hybrides combinant méthodes exactes et heuristiques, ainsi que vers les hyperheuristiques et l'apprentissage automatique, témoignant de la richesse et de la vitalité du domaine.

Le problème de planification des horaires infirmiers, connu sous le nom de *Nurse Rostering Problem* (NRP), est un problème d'optimisation combinatoire NP-difficile. Il consiste à affecter des infirmiers à des shifts sur une période donnée tout en respectant un ensemble de contraintes dures et molles, et en optimisant plusieurs critères tels que l'équité, la satisfaction du personnel et la couverture des besoins des patients.

Ce problème est particulièrement critique dans les établissements de santé, où une mauvaise planification peut entraîner une surcharge de travail, une baisse de la qualité des soins et une augmentation des coûts opérationnels. De nombreuses approches de résolution ont été explorées dans la littérature pour répondre à cette complexité, parmi lesquelles les méthodes exactes (programmation linéaire en nombres entiers, programmation par contraintes, décomposition de Benders), les métaheuristiques (recuit simulé, recherche locale, algorithmes génétiques, colonies de fourmis), ainsi que des approches plus récentes telles que les hyperheuristiques et les méthodes hybrides.

Cependant, malgré les avancées dans les techniques d'optimisation, la transformation des contraintes réelles du terrain en un modèle exploitable et résoluble de manière efficace reste un défi important, en raison de la nature combinatoire et fortement contrainte du problème. C'est dans ce contexte que s'inscrit ce travail de mémoire, autour de la question centrale suivante :

> *Comment concevoir un modèle d'optimisation capable de représenter fidèlement le problème de planification des infirmiers et de produire des solutions efficaces, tout en respectant l'ensemble des contraintes et des objectifs de performance associés ?*

L'objectif principal de ce mémoire est de proposer une approche efficace pour la résolution du problème de planification des infirmiers (NRP). Dans ce cadre, les objectifs spécifiques de ce travail sont les suivants :

— proposer une modélisation mathématique complète et rigoureuse du NRP sous la forme d'un modèle de programmation linéaire en nombres entiers (MILP), permettant de représenter fidèlement les contraintes du problème ;
— étudier et implémenter différentes approches de résolution, incluant une méthode exacte ainsi que des métaheuristiques de trajectoire, afin d'évaluer leur efficacité

face à la complexité du problème ;

— comparer expérimentalement les performances des différentes méthodes proposées sur des instances de tailles variées, en analysant la qualité des solutions obtenues, le respect des contraintes ainsi que les temps de calcul.

L'ensemble de ces objectifs structure la démarche suivie dans ce mémoire et oriente les choix méthodologiques adoptés pour la modélisation et la résolution du problème étudié.

Les principales contributions de ce travail peuvent être résumées comme suit :

— **Modélisation mathématique :** développement d'un modèle de programmation linéaire en nombres entiers (MILP) pour le NRP. Ce modèle intègre un ensemble de contraintes dures (affectation unique, couverture des shifts, contraintes de repos, etc.) ainsi que des contraintes molles liées aux préférences des infirmiers et à l'équité de la charge de travail, afin de représenter fidèlement les exigences du contexte hospitalier.

— **Implémentation et comparaison d'approches de résolution :** mise en œuvre d'une approche exacte basée sur le solveur GLPK, ainsi que de métaheuristiques de trajectoire, à savoir la recherche locale (*Local Search*) et le recuit simulé (*Simulated Annealing*). Ces méthodes permettent d'explorer différentes stratégies de recherche dans un espace combinatoire complexe.

— **Étude expérimentale approfondie :** évaluation des méthodes proposées sur plusieurs instances de tailles variées (petite, moyenne, grande et très grande) issues de la bibliothèque NSPLib. Les performances sont analysées selon plusieurs critères : qualité des solutions, respect des contraintes, temps de calcul et écart par rapport à l'optimum.

Ce mémoire est organisé en cinq chapitres :

— Le **premier chapitre** présente une revue de la littérature sur le problème de planification des infirmiers (NRP), en mettant en évidence les principales approches de modélisation et de résolution proposées dans les travaux antérieurs.

— Le **deuxième chapitre** est consacré à la formulation mathématique du problème, où le modèle de programmation linéaire en nombres entiers (MILP) est présenté en détail, ainsi que l'ensemble des variables, contraintes et la fonction objectif.

— Le **troisième chapitre** traite des différentes méthodes de résolution adoptées dans ce travail, incluant les méthodes exactes ainsi que les métaheuristiques de trajectoire telles que la recherche locale et le recuit simulé.

— Le **quatrième chapitre** présente l'implémentation des méthodes proposées ainsi que le protocole expérimental utilisé, en décrivant l'environnement de développement, les instances de test et les métriques d'évaluation. Il inclut également le positionnement de notre contribution par rapport aux travaux existants dans

la littérature.

— Enfin, le **cinquième chapitre** est dédié à la présentation et à l'analyse des résultats expérimentaux, suivie d'une étude comparative des différentes approches.

Le mémoire se conclut par une synthèse générale rappelant les principaux résultats obtenus et présentant les perspectives de travaux futurs.

Chapitre 1

Le problème de planification des infirmiers : définition et approches de résolution

1.1 Introduction

Ce chapitre présente de manière approfondie le *Nurse Rostering Problem* (NRP), un problème d'optimisation combinatoire NP-difficile largement étudié dans la littérature de recherche opérationnelle [15]. Après une brève évolution historique présentée dans l'introduction générale, nous nous concentrons ici sur la **définition formelle** du problème, ses principales **caractéristiques** et les **approches de résolution** qui ont été proposées au fil des années.

L'objectif de ce chapitre est triple :

— donner une définition formelle du NRP et caractériser sa complexité ;
— décrire les principales catégories de contraintes et de variantes ;
— synthétiser les approches de résolution existantes et les ressources de référence utilisées dans la littérature.

Le positionnement de notre contribution par rapport à ces travaux sera développé au Chapitre 4, juste avant la présentation du protocole expérimental.

1.2 Définition et complexité du NRP

1.2.1 Définition générale

Dans les établissements hospitaliers, la génération régulière de plannings de travail pour le personnel infirmier est une tâche essentielle qui impacte directement la qualité des soins, l'organisation des services et la maîtrise des coûts. Selon Wren [24], un

problème de planification du personnel consiste à affecter des ressources humaines à des créneaux de travail selon un ensemble de règles et de contraintes.

Le *Nurse Rostering Problem* (NRP), également appelé *Nurse Scheduling Problem* (NSP), apparaît dans ce cadre comme un problème d'optimisation combinatoire de référence. Il consiste à construire un planning périodique (hebdomadaire, bihebdomadaire ou mensuel) pour les infirmiers d'un service, en respectant des contraintes organisationnelles, légales et individuelles, et en optimisant des critères tels que l'équité de la charge de travail et la satisfaction des préférences [12]. La couverture continue 24h/24 et 7j/7, la diversité des règles selon les établissements et le grand nombre de demandes individuelles font du NRP une tâche particulièrement complexe.

1.2.2 Formulation formelle

Pour modéliser rigoureusement le NRP, on définit [12, 7] :

— I : ensemble des infirmiers, indexé par i ;

— T : ensemble des jours de planification, indexé par t ;

— S : ensemble des shifts (matin, après-midi, nuit), indexé par s.

Les variables de décision binaires principales sont :

$$x_{i,s,t} = \begin{cases} 1 & \text{si l'infirmier } i \text{ est affecté au shift } s \text{ le jour } t, \\ 0 & \text{sinon,} \end{cases}$$

$$y_{i,t} = \begin{cases} 1 & \text{si l'infirmier } i \text{ est en repos le jour } t, \\ 0 & \text{sinon.} \end{cases}$$

L'objectif est généralement de minimiser les violations des contraintes molles tout en satisfaisant l'ensemble des contraintes dures [7]. Cette formulation servira de base à la modélisation MILP détaillée au Chapitre 2.

1.2.3 Complexité NP-difficile

Le NRP appartient à la classe des problèmes **NP-difficiles** [15, 6]. Le problème de *timetabling*, NP-complet, peut en effet se réduire à une version décisionnelle du NRP. Cette classification implique qu'aucun algorithme polynomial connu ne permet de résoudre toutes les instances de manière optimale, et que le temps de calcul croît exponentiellement avec la taille du problème.

L'introduction de contraintes réalistes (séquences interdites, repos consécutifs, équité de charge) accentue encore cette complexité [19]. Lorsque le nombre d'infirmiers, de jours et de shifts augmente, l'espace de recherche atteint des tailles prohibitives pour toute énumération exhaustive, ce qui justifie le recours aux approches heuristiques et

métaheuristiques pour les instances de grande taille.

1.3 Contraintes et variantes du NRP

1.3.1 Catégories de contraintes

Les contraintes du NRP traduisent les exigences légales, organisationnelles et opérationnelles de chaque établissement. Elles sont classiquement regroupées en trois catégories [18] :

- **Contraintes d'affectation :** règles régissant l'attribution des infirmiers aux shifts, prenant en compte disponibilités, compétences et préférences ;
- **Contraintes de couverture :** besoins en effectifs pour chaque période de travail, exprimés par des bornes minimales et maximales ;
- **Contraintes d'horaires :** organisation temporelle (repos quotidien et hebdomadaire, séquences de travail, enchaînements interdits comme la succession nuit→matin).

1.3.2 Contraintes dures et contraintes molles

Une distinction essentielle est faite entre deux types de contraintes :

Les contraintes dures (*hard constraints*) doivent être impérativement satisfaites. Leur violation rend le planning invalide. Elles correspondent aux exigences légales et de sécurité (affectation unique par jour, couverture minimale, limites horaires, repos obligatoire, séquences autorisées).

Les contraintes molles (*soft constraints*) sont des préférences ou objectifs souhaitables mais non obligatoires. Leur non-respect entraîne des pénalités dans la fonction objectif (préférences individuelles, équité de charge, confort de travail).

Cette distinction transforme le NRP d'un problème de satisfaction de contraintes en un problème d'**optimisation** : trouver un planning respectant toutes les contraintes dures tout en minimisant les violations des contraintes molles.

Contraintes dures	**Contraintes molles**
Doivent être obligatoirement respectées	Peuvent être violées avec pénalités
Leur violation rend la solution invalide	Leur violation dégrade la qualité
Exemples : couverture minimale, repos obligatoire, limites horaires	Exemples : préférences, équité, confort
Imposées par la réglementation	Liées aux préférences et à l'optimisation

TABLE 1.1 – Comparaison entre contraintes dures et molles dans le NRP

1.3.3 Variantes du problème

Le NRP admet plusieurs variantes selon trois dimensions principales [7, 14] : la **structure temporelle** (cyclique vs non-cyclique), l'**organisation des services** (mono-service vs multi-services) et les **compétences du personnel** (mono-compétence vs multi-compétences). L'ajout de chaque dimension réaliste augmente significativement la taille de l'espace de solutions et la complexité combinatoire du problème, ce qui justifie le recours à des méthodes d'optimisation avancées telles que la programmation linéaire en nombres entiers mixtes (MILP).

1.4 Approches de résolution dans la littérature

1.4.1 Méthodes exactes

La **programmation linéaire en nombres entiers (ILP)**, et sa version mixte (MILP), constitue l'approche exacte la plus utilisée pour le NRP. Elle permet une représentation rigoureuse des contraintes et fournit des bornes optimales servant de référence. La résolution s'appuie sur des algorithmes tels que *Branch-and-Bound* et *Branch-and-Cut*, ainsi que sur des techniques de plans de coupe [20].

Cependant, ces méthodes présentent des limites importantes pour les instances de grande taille : les temps de calcul peuvent atteindre plusieurs heures, et la résolution complète du problème original devient impraticable au-delà d'une certaine taille [22]. D'autres approches exactes, comme la programmation par contraintes ou la décomposition de Benders [21], ont été proposées pour traiter des structures spécifiques, mais souffrent également de problèmes de passage à l'échelle.

Cette limitation a motivé le développement d'approches heuristiques et métaheuristiques, capables de produire des solutions de bonne qualité dans des délais raisonnables.

1.4.2 Heuristiques et métaheuristiques

La **recherche locale** (*Local Search*, LS) est l'une des approches fondamentales [11]. Elle améliore progressivement une solution initiale en explorant son voisinage. Rapide et simple à implémenter, elle reste néanmoins sensible au piégeage dans les optima locaux.

Le **recuit simulé** (*Simulated Annealing*, SA), inspiré des processus de refroidissement physique, accepte parfois des solutions de qualité inférieure selon une probabilité décroissante avec la température, ce qui lui permet d'échapper aux optima locaux [2, 5].

D'autres familles ont également été appliquées avec succès : algorithmes génétiques [1], recherche tabou [10], et plus récemment hyperheuristiques [3]. Le Tableau 1.2 synthétise les principales caractéristiques de ces approches.

Approche	**Caractéristique principale**	**Limite principale**
Méthodes exactes (MILP)	Optimalité garantie	Scalabilité limitée
Recherche locale (LS)	Amélioration rapide	Optima locaux
Recuit simulé (SA)	Exploration probabiliste	Sensibilité aux paramètres
Hyperheuristiques	Adaptabilité élevée	Complexité de conception

TABLE 1.2 – Synthèse des approches de résolution du NRP

1.5 Benchmarks standardisés : NSPLib

Pour évaluer rigoureusement les algorithmes de résolution du NRP, des bibliothèques d'instances standardisées ont été développées. La plus utilisée est la **NSPLib** (*Nurse Scheduling Problem Library*) [8, 23], qui propose des instances de tailles variées (généralement 25 à 100 infirmiers) avec des horizons de 4 à 8 semaines, et intègre des contraintes réalistes : couverture des besoins, périodes de repos, séquences de travail et préférences individuelles.

L'utilisation de NSPLib présente plusieurs avantages méthodologiques : standardisation des comparaisons, reproductibilité des expériences et évaluation multicritères des algorithmes (qualité des solutions, respect des contraintes, temps de calcul) [9]. C'est cette bibliothèque que nous utilisons dans nos expérimentations (Chapitre 4).

D'autres initiatives, comme les compétitions internationales *International Nurse Rostering Competition* (INRC-I en 2010 et INRC-II en 2015) [17], ont également contribué à la structuration du domaine en introduisant des cadres expérimentaux contrôlés et des instances multi-périodes plus proches des conditions réelles.

1.6 Conclusion

Ce chapitre a présenté le NRP comme un problème combinatoire NP-difficile, ses contraintes typiques (dures et molles) et ses principales variantes. Les approches de résolution proposées dans la littérature se répartissent essentiellement entre méthodes exactes (MILP, programmation par contraintes, décomposition de Benders) et méta-heuristiques (recherche locale, recuit simulé, hyperheuristiques), motivées par les limites de passage à l'échelle des premières. La bibliothèque NSPLib s'est imposée comme *benchmark* de référence pour la comparaison équitable des algorithmes.

Notre travail s'inscrit dans cette continuité en proposant une comparaison entre une approche exacte (MILP résolu par GLPK) et deux métaheuristiques de trajectoire (LS et SA) sur des instances NSPLib. Le positionnement détaillé de notre contribution par rapport à la littérature sera développé au Chapitre 4. Le chapitre suivant détaille la formulation mathématique adoptée.

Chapitre 2

Modélisation en programmation linéaire mixte du Nurse Rostering Problem

2.1 Introduction

Le problème de l'élaboration d'un planning pour la rotation des infirmiers (NRP) est un problème particulier d'emploi du temps dont l'objectif est la production d'un planning qui satisfait les exigences de l'établissement ainsi que les demandes du personnel requis tout en respectant un certain nombre de contraintes.

Les NRP varient considérablement d'un établissement à l'autre, en raison du large éventail de contraintes spécifiques (les demandes des infirmiers, la politique hospitalière, la réglementation, etc.). Il est bien connu que le NRP est un problème d'optimisation combinatoire très difficile : sa complexité est théoriquement NP-difficile [12, 7]. En pratique, il est très difficile à résoudre en raison de la grande masse de données et de l'explosion combinatoire. De tels problèmes sont compliqués et difficiles, en particulier pour les résoudre à l'optimalité [4]. C'est pourquoi les approches mathématiques exactes ou heuristiques sont souvent utilisées pour générer efficacement un grand nombre de solutions faisables.

2.2 Paramètres du modèle

Avant de formuler les variables et les contraintes, il est indispensable de définir l'ensemble des *paramètres d'entrée* du modèle, c'est-à-dire les données connues avant la résolution.

Paramètre	**Définition**
N	Nombre total d'infirmiers (N = |I|)
$\|S\|$	Nombre de shifts par jour (matin, après-midi, nuit)
$\|T\|$	Horizon de planification en jours
h_s	Durée en heures du shift $s \in S$
$\text{minStaff}_{s,t}$	Nombre minimum d'infirmiers requis pour le shift s le jour t
$\text{maxStaff}_{s,t}$	Nombre maximum d'infirmiers autorisés pour le shift s le jour t
maxH_i^w	Nombre maximum d'heures de travail par semaine pour l'infirmier i
maxH_i^m	Nombre maximum d'heures de travail par mois pour l'infirmier i
minRest	Nombre minimal de jours de repos consécutifs par semaine
$\text{pref}_{i,s,t}$	Vaut 1 si l'infirmier i souhaite travailler le shift s le jour t, 0 sinon
$p_{i,s,t}$	Pénalité associée à l'affectation de l'infirmier i au shift s le jour t contre sa préférence
$\alpha, \beta, \gamma \geq 0$	Coefficients de pondération de la fonction objectif, avec $\alpha + \beta + \gamma = 1$

TABLE 2.1 – Paramètres d'entrée du modèle NRP-MILP

2.3 Variables de décision

Soit un ensemble I de N infirmiers à planifier. Chaque infirmier ne peut effectuer qu'un seul shift par jour, sur une période de planification de $|T|$ jours, chaque jour étant composé de $|S|$ shifts.

Les ensembles utilisés sont les suivants :

— I : ensemble des infirmiers, indexé par i (avec $i = 1, \ldots, N$) ;

— S : ensemble des shifts (matin, après-midi, nuit), indexé par s ;

— T : ensemble des jours de la période de planification, indexé par t.

Les variables de décision du modèle sont définies comme suit :

$$x_{i,s,t} = \begin{cases} 1, & \text{si l'infirmier } i \text{ travaille sur le shift } s \text{ le jour } t, \\ 0, & \text{sinon.} \end{cases}$$

$$y_{i,t} = \begin{cases} 1, & \text{si l'infirmier } i \text{ est en repos le jour } t, \\ 0, & \text{sinon.} \end{cases}$$

$$z_{i,t} = \begin{cases} 1, & \text{si l'infirmier } i \text{ commence une séquence de repos le jour } t, \\ 0, & \text{sinon.} \end{cases}$$

$u_i \geq 0$ déviation de la charge de travail de l'infirmier i par rapport à la moyenne ;

$\varepsilon_{i,s,t} \geq 0$ écart entre la préférence souhaitée et l'affectation réelle.

2.4 Charge de travail

La charge de travail totale de l'infirmier i sur l'horizon de planification est définie par :

$$W_i = \sum_{s \in S} \sum_{t \in T} h_s \cdot x_{i,s,t} \qquad \forall\, i \in I \tag{2.1}$$

où h_s est la durée en heures du shift s. La charge de travail moyenne sur l'ensemble des infirmiers est :

$$\overline{W} = \frac{1}{|I|} \sum_{i \in I} W_i. \tag{2.2}$$

2.5 Fonction objectif

La fonction objectif est un critère d'efficacité ou de préférence du décideur, formalisé mathématiquement, dont la valeur est optimisée lors de la résolution du problème. Le NRP est souvent modélisé en tant que problème multi-objectif. L'objectif général est de minimiser le coût global, composé de trois sous-objectifs :

— **Minimiser les violations de contraintes :** assurer le respect des règles opérationnelles et légales en pénalisant leurs transgressions.

— **Minimiser le non-respect des préférences :** intégrer les souhaits individuels des infirmiers.

— **Équilibrer la charge :** promouvoir l'équité en minimisant les écarts de charge de travail entre infirmiers.

Mathématiquement, la fonction objectif est définie comme suit :

$$\min Z = \underbrace{\alpha \sum_{i \in I} \sum_{t \in T} \sum_{s \in S} p_{i,s,t} \cdot x_{i,s,t}}_{\text{pénalités de préférence}} + \underbrace{\beta \sum_{i \in I} \sum_{s \in S} \sum_{t \in T} \varepsilon_{i,s,t}}_{\text{violations des contraintes souples}} + \underbrace{\gamma \sum_{i \in I} u_i}_{\text{équité de charge}} \tag{2.3}$$

où $\alpha + \beta + \gamma = 1$ et $\alpha, \beta, \gamma \geq 0$.

Définition des termes :

$x_{i,s,t}$ variable de décision égale à 1 si l'infirmier i est affecté au shift s le jour t.

$p_{i,s,t}$ coefficient de pénalité représentant le non-respect de la préférence de l'infirmier i.

$\varepsilon_{i,s,t}$ variable d'écart (slack) mesurant la différence entre la préférence souhaitée et l'affectation réelle.

u_i déviation de la charge de travail de l'infirmier i par rapport à la moyenne $\overline{W}$.

W_i charge de travail totale de l'infirmier i (voir équation 2.1).

$\overline{W}$ charge de travail moyenne (voir équation 2.2).

α, β, γ paramètres de pondération ajustant l'importance relative de chaque critère.

2.6 Contraintes du modèle

Les contraintes varient d'un établissement de santé à un autre, car chaque institution possède sa propre politique administrative, ses besoins et ses caractéristiques. Ces contraintes sont traditionnellement classées dans la littérature de planification en trois catégories principales : les contraintes d'affectation, les contraintes de couverture et les contraintes d'horaires.

Contraintes d'affectation : règles qui régissent l'affectation des infirmiers à des shifts ou à des tâches données. Elles assurent une répartition homogène du personnel en tenant compte des disponibilités et des compétences professionnelles.

Contraintes de couverture : obligations relatives aux effectifs pour garantir la présence d'un nombre d'infirmiers suffisant pour chaque période de travail, définies par une borne inférieure *et* une borne supérieure.

Contraintes d'horaires : organisation temporelle du travail (repos quotidien, hebdomadaire, successions de shifts).

Le NRP distingue deux types de contraintes :

Contraintes dures (Hard) : doivent être satisfaites dans tout planning réaliste. Leur violation rend le planning inacceptable.

Contraintes molles (Soft) : souhaitables mais non obligatoires. Leur violation est tolérée et pénalisée dans la fonction objectif.

2.6.1 Formalisation mathématique des contraintes

Contraintes dures (Hard)

$$y_{i,t} = 1 - \sum_{s \in S} x_{i,s,t} \qquad \forall\, i \in I,\ t \in T \tag{H1}$$

$$\text{minStaff}_{s,t} \ \leq \ \sum_{i \in I} x_{i,s,t} \ \leq \ \text{maxStaff}_{s,t} \qquad \forall\, s \in S,\ t \in T \tag{H2}$$

$$\sum_{s \in S} \sum_{t \in \text{sem. } w} h_s \cdot x_{i,s,t} \leq \text{maxH}_i^w \qquad \forall\, i \in I,\ w \tag{H3}$$

$$\sum_{s \in S} \sum_{t \in \text{mois } m} h_s \cdot x_{i,s,t} \leq \text{maxH}_i^m \qquad \forall\, i \in I,\ m \tag{H4}$$

$$x_{i,s,t} + x_{i,s',t+1} \leq 1 \qquad \forall\, i \in I,\ (s, s') \text{ incompatibles} \tag{H5}$$

$$x_{i,\text{nuit},t} + x_{i,\text{matin},t+1} \leq 1 \qquad \forall\, i \in I,\ t = 1, \ldots, |T| - 1 \tag{H6}$$

$$y_{i,t} + y_{i,t+1} \geq 2\, z_{i,t} \qquad \forall\, i \in I,\ t = 1, \ldots, |T| - 1 \tag{H7a}$$

$$\sum_{t \in \text{sem. } w} z_{i,t} \geq 1 \qquad \forall\, i \in I,\ w \tag{H7b}$$

Notes sur la formulation :

- **(H1)** Cette contrainte définit explicitement la variable $y_{i,t}$ comme étant le complément de la somme des affectations aux shifts ; elle évite toute ambiguïté dans le statut « travail / repos ».
- **(H2)** Cette contrainte impose à la fois une borne inférieure (couverture minimale, pour garantir la qualité des soins) et une borne supérieure (couverture maximale, pour éviter le sur-effectif et maîtriser les coûts).
- **(H5)** et **(H6)** se complètent : (H5) interdit toute paire de shifts incompatibles entre deux jours consécutifs, tandis que (H6) explicite le cas particulier — fréquent et critique — de la succession nuit → matin.
- **(H7a)**–**(H7b)** La contrainte (H7a) garantit que si $z_{i,t} = 1$ (début de séquence de repos), alors les jours t et $t + 1$ sont effectivement des jours de repos. La contrainte (H7b) impose qu'au moins une séquence de repos soit débutée par semaine, garantissant ainsi le repos hebdomadaire obligatoire.

Contraintes souples (Soft)

$$\varepsilon_{i,s,t} \geq \text{pref}_{i,s,t} - x_{i,s,t} \qquad \forall\, i \in I,\ s \in S,\ t \in T \qquad \text{(S1)}$$

$$\varepsilon_{i,s,t} \geq 0 \qquad \forall\, i \in I,\ s \in S,\ t \in T \qquad \text{(S2)}$$

Note : la variable $\varepsilon_{i,s,t}$ mesure l'écart entre la préférence souhaitée par l'infirmier ($\text{pref}_{i,s,t} = 1$) et son affectation réelle ($x_{i,s,t}$). Elle est minimisée dans la fonction objectif via le terme pondéré par β.

Linéarisation de l'équité de charge

$$W_i - \overline{W} \leq u_i \qquad \forall\, i \in I \qquad \text{(E1)}$$

$$\overline{W} - W_i \leq u_i \qquad \forall\, i \in I \qquad \text{(E2)}$$

Ces deux contraintes linéarisent la valeur absolue $|W_i - \overline{W}|$: en minimisant u_i dans la fonction objectif, on force u_i à égaler l'écart réel entre la charge de l'infirmier i et la charge moyenne.

Domaine des variables

$$x_{i,s,t} \in \{0,1\} \qquad \forall\, i \in I,\ s \in S,\ t \in T$$

$$y_{i,t} \in \{0,1\} \qquad \forall\, i \in I,\ t \in T$$

$$z_{i,t} \in \{0,1\} \qquad \forall\, i \in I,\ t \in T$$

$$u_i \geq 0 \qquad \forall\, i \in I$$

$$\varepsilon_{i,s,t} \geq 0 \qquad \forall\, i \in I,\ s \in S,\ t \in T$$

2.7 Analyse de la complexité du MILP

Le *Mixed-Integer Linear Programming* (MILP) est une technique d'optimisation qui associe la programmation linéaire (LP) à des variables entières, rendant possible la modélisation de problèmes complexes impliquant des variables tant continues que discrètes.

2.7.1 Taille du modèle

Dans le cas de la planification des plannings infirmiers (NRP), la taille du modèle dépend typiquement du nombre d'infirmiers, du nombre de shifts par jour et de la durée de l'horizon de planification. Si l'on considère N infirmiers, $|S|$ shifts et $|T|$ jours, le nombre de variables binaires d'affectation peut être approché par $N \times |S| \times |T|$. À ces variables s'ajoutent des contraintes portant sur la couverture minimale des services, les temps de repos, les règles d'organisation et les préférences individuelles.

Exemple illustratif : Prenons un service hospitalier avec 27 infirmiers et 3 shifts (matin, soir, nuit) répartis sur 26 jours. Le nombre total de variables peut être approximé par :

$$27 \times 3 \times 26 = 2\,106 \text{ variables binaires.}$$

À ces variables s'ajoutent plusieurs contraintes :

— contraintes de couverture des shifts : environ $3 \times 26 = 78$ contraintes ;

— contraintes d'affectation unique par jour : environ $27 \times 26 = 702$ contraintes ;

— contraintes de repos, de limite horaire, de préférences individuelles, etc.

Ainsi, le modèle peut comporter plus de 2 000 variables et un grand nombre de contraintes, ce qui augmente fortement la complexité et le temps de résolution.

2.7.2 Différentes classes de complexité

Afin de constituer un outil de classement, les problèmes pouvant être résolus dans des temps similaires sont regroupés en classes de complexité : P, NP, NP-complet et NP-difficile.

La classe P

La classe P (temps déterministe polynomial) contient tous les problèmes de décision pour lesquels on connaît des algorithmes efficaces, pouvant être résolus en temps polynomial par une machine déterministe.

La classe NP

La classe NP (temps non-déterministe polynomial) est définie comme la classe des problèmes de décision pouvant être résolus de manière non-déterministe en temps polynomial. Contrairement aux machines déterministes, les machines non-déterministes choisissent toujours la meilleure séquence d'instructions menant à la bonne réponse.

La classe NP-complet

La classe NP-complet est un sous-ensemble de NP. Un problème est NP-complet si tout problème dans NP peut lui être réduit en temps polynomial. Trouver un algorithme polynomial pour un problème NP-complet impliquerait l'existence d'algorithmes rapides pour tous les problèmes NP.

La classe NP-difficile (NP-dur)

Un problème est NP-difficile s'il existe un problème NP-complet qui lui est réductible par la réduction de Turing. De manière générale, le MILP est classé parmi les problèmes NP-difficiles.

2.7.3 Difficulté du problème et méthodes de résolution

Le problème de planification des infirmiers (NRP) est un problème MILP de complexité NP-difficile. En pratique, il est très difficile à résoudre en raison de la grande masse de données et de l'explosion combinatoire. Il existe trois grandes catégories de méthodes de résolution :

- **Méthodes exactes :** garantissent la solution optimale (Branch and Bound, Branch and Cut), mais nécessitent un temps de calcul prohibitif pour les grandes instances.
- **Méthodes heuristiques :** produisent une solution réalisable rapidement, sans garantie d'optimalité.
- **Méthodes métaheuristiques :** plus complètes que les simples heuristiques, elles permettent généralement d'obtenir des solutions de très bonne qualité en des temps raisonnables (algorithmes génétiques, recuit simulé, recherche tabou, recherche locale).

2.8 Conclusion

Dans ce chapitre, nous avons abordé la difficulté du problème de planification optimale des plannings infirmiers, aussi bien du point de vue de la complexité théorique que de la mise en œuvre pratique. La mise en place d'un planning optimal est considérée comme une tâche très lourde, nécessitant un temps de calcul excessivement long. C'est pourquoi l'on opte généralement pour des algorithmes suffisamment rapides produisant des solutions approchées dans des délais raisonnables — les métaheuristiques — que nous aborderons dans le chapitre suivant.

Chapitre 3

Méthodes de résolution pour le problème de planification du personnel infirmier

3.1 Introduction

La résolution du problème de planification des infirmiers (NRP) peut s'envisager selon deux grandes familles d'approches. D'une part, les **méthodes exactes** permettent d'obtenir des solutions dont l'optimalité est garantie ; elles reposent sur l'exploration systématique de l'espace de recherche et conviennent particulièrement aux instances de petite taille. D'autre part, lorsque la taille du problème devient prohibitive, on peut chercher des solutions de bonne qualité, sans garantie d'optimalité, au profit d'un temps de calcul plus réduit ; on applique alors des méthodes appelées **métaheuristiques**.

Les heuristiques et les métaheuristiques exploitent généralement des processus aléatoires dans l'exploration de l'espace de recherche pour faire face à l'explosion combinatoire. Les métaheuristiques sont en outre le plus souvent itératives ; leur principal intérêt provient de leur capacité à éviter les minima locaux en admettant une dégradation de la fonction objectif au cours de leur progression.

Ce chapitre présente d'abord les méthodes exactes (Branch and Bound, Branch and Cut), puis les principales métaheuristiques retenues dans ce travail (recherche locale et recuit simulé), et enfin une comparaison de ces approches accompagnée d'une illustration concrète.

3.2 Méthodes exactes

Les méthodes exactes permettent de garantir la solution optimale pour une instance de taille finie dans un temps limité. Cependant, elles nécessitent souvent un coût de

recherche prohibitif en termes de ressources. Parmi les méthodes exactes, on trouve le Branch and Bound et le Branch and Cut.

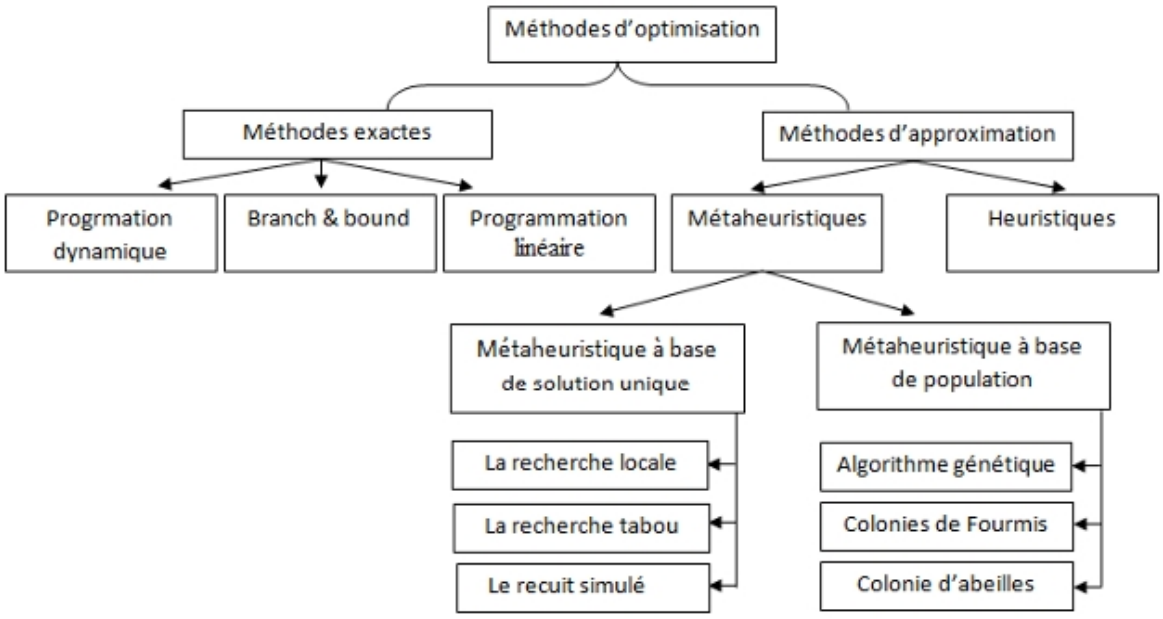

FIGURE 3.1 – Classification des méthodes d'optimisation combinatoire.

3.2.1 Branch and Bound (B&B)

Le *Branch-and-Bound* est un paradigme général de programmation utilisé pour résoudre des problèmes d'optimisation combinatoire difficiles comme le NRP. Le *branching* est le processus de création de sous-problèmes, et le *bounding* consiste à ignorer les solutions partielles qui ne peuvent pas être meilleures que la meilleure solution actuelle.

Soit le problème suivant :

$$(P) \quad \begin{cases} z = \min H(x) \\ x \in S \subset \mathbb{N}^n \end{cases} \tag{3.1}$$

L'algorithme Branch and Bound repose sur une méthode arborescente de recherche d'une solution optimale par séparations et évaluations, en représentant les états-solutions par un arbre d'états. Il est basé sur trois axes principaux :

1. la procédure de séparation ;
2. la procédure d'évaluation ;
3. la stratégie de parcours.

La séparation

La séparation consiste à diviser le problème en sous-problèmes. L'ensemble des nœuds restant à parcourir, susceptibles de contenir une solution optimale, est appelé ensemble des nœuds actifs.

La stratégie de parcours

La largeur d'abord : favorise les sommets les plus proches de la racine. Cette stratégie est moins efficace que les deux autres.

La profondeur d'abord : avantage les sommets les plus éloignés de la racine. Elle mène rapidement à une solution optimale en économisant la mémoire.

Le meilleur d'abord : explore les sous-problèmes possédant la meilleure borne. Cette stratégie évite l'exploration de sous-problèmes à mauvaise évaluation.[13]

Applications

Le Branch and Bound est largement utilisé pour : l'optimisation combinatoire (TSP, sac à dos, NRP), la satisfaction de contraintes, et l'allocation des ressources.

Avantages et inconvénients

Avantages : optimalité garantie ; efficacité mémoire relative ; flexibilité ; parallélisable.

Inconvénients : complexité de mise en œuvre ; dépendance heuristique ; inadapté aux environnements dynamiques[16]

3.2.2 Branch and Cut

La méthode *Branch-and-Cut* combine l'algorithme Branch and Bound et la méthode des coupes planes. Comme le B&B, elle explore un arbre de recherche où chaque nœud correspond à un sous-problème avec des contraintes supplémentaires. La différence essentielle réside dans l'utilisation des *plans de coupe* : après la résolution de la relaxation linéaire, si la solution viole certaines inégalités valides, des contraintes supplémentaires (coupes) sont ajoutées pour éliminer les solutions fractionnaires sans exclure de solutions entières réalisables Le Branch-and-Cut constitue l'approche principale implémentée dans la plupart des solveurs MILP modernes (GLPK, CPLEX, OR-Tools). Sa structure modulaire permet une intégration transparente avec des heuristiques, le prétraitement et des stratégies de branchement avancées.

3.3 Métaheuristiques

Les métaheuristiques sont des algorithmes d'optimisation approchée, généraux et itératifs, conçus pour explorer efficacement de grands espaces de recherche. Elles sont particulièrement adaptées au NRP lorsque les méthodes exactes deviennent trop coûteuses.

3.3.1 Recherche locale (LS)

La recherche locale est une méthode générale utilisée pour résoudre des problèmes d'optimisation, c'est-à-dire des problèmes où l'on cherche la meilleure solution dans un ensemble de solutions candidates. La recherche locale consiste à passer d'une solution à une autre solution proche dans l'espace des solutions candidates, jusqu'à trouver une solution optimale ou que le temps imparti soit dépassé.

Principe de la recherche locale appliqué au NRP

Dans le cadre du *Nurse Rostering Problem* (NRP), la recherche locale repose sur l'amélioration progressive d'un planning des infirmiers. Son principe peut être résumé comme suit :

Initialisation : génération d'un planning initial des infirmiers (solution réalisable).

Définition du voisinage : création de solutions voisines à partir du planning courant, par exemple :

— échange de shifts entre deux infirmiers ;
— déplacement d'un shift d'un jour à un autre ;
— modification d'un jour de repos.

Évaluation (fonction objectif) : chaque planning est évalué selon :

— le respect des contraintes dures (jours de repos obligatoires, couverture) ;
— la minimisation des contraintes souples (préférences, équilibre du travail).

Sélection du voisin : choix d'une meilleure solution voisine selon une stratégie d'acceptation.

Mise à jour : remplacement du planning courant par le nouveau planning s'il est meilleur.

Itération : répétition du processus pour améliorer progressivement le planning.

Critère d'arrêt : l'algorithme s'arrête lorsque :

— aucune amélioration n'est possible (optimum local) ;
— ou bien après un nombre d'itérations / temps limite atteint.

3.3.2 Paramètres et stratégies de la recherche locale

La recherche locale dépend de plusieurs paramètres et choix de conception qu'il convient de fixer avant son exécution :

- **Solution initiale** (s_0) : solution de départ de l'algorithme, généralement obtenue par une heuristique constructive.
- **Type de voisinage** : méthode utilisée pour générer les solutions voisines (échange de shifts, déplacement, modification d'affectation).
- **Stratégie de sélection du voisin** : deux stratégies classiques sont utilisées pour explorer le voisinage :
  - *First Improvement* (première amélioration) : l'algorithme parcourt le voisinage et accepte la première solution qui améliore la fonction objectif. Cette stratégie est rapide, mais peut ne pas donner la meilleure solution possible.
  - *Best Improvement* (meilleure amélioration) : l'algorithme explore tout le voisinage et sélectionne la meilleure solution parmi toutes les solutions voisines. Cette approche permet d'obtenir une meilleure qualité de solution, mais elle est plus coûteuse en temps de calcul.
- **Nombre maximal d'itérations** (nb_iter) : critère d'arrêt principal limitant la durée de la recherche.

3.3.3 Pseudo-code de la recherche locale (LS) pour le NRP

Algorithm 1 Recherche locale appliquée au NRP (LS-NRP)

Require: I infirmiers, T jours, S shifts, contraintes, nb_iter

Ensure: Meilleur planning trouvé s_{best}

Générer une solution initiale s (matrice $I \times T$: shift ou repos)
Réparer les violations de contraintes dures dans s
$s_{best} \leftarrow s$
Calculer $Z(s)$ ▷ Évaluation de la fonction objectif
for $k = 1$ **à** nb_iter **do**
    Choisir un infirmier i et un jour t aléatoires
    Générer un voisin s' en modifiant $x_{i,t}$ par un shift compatible
    Réparer les contraintes dures violées dans s'
    Calculer $Z(s')$
    if $Z(s') < Z(s)$ **then**
        $s \leftarrow s'$
    end if
    if $Z(s) < Z(s_{best})$ **then**
        $s_{best} \leftarrow s$
    end if
end for
return s_{best}

Application de la recherche locale au NRP

Dans le cadre du NRP, la recherche locale est utilisée pour améliorer un planning initial des infirmiers. La solution est représentée sous forme d'une matrice $I \times T$, conformément au modèle défini précédemment.

La fonction objectif utilisée correspond à celle définie dans le chapitre de modélisation (équation 1.1), intégrant les pénalités liées aux préférences, aux contraintes souples et à l'équilibre de la charge de travail. Les contraintes dures et souples considérées sont celles présentées dans la section 1.4.

Le voisinage est généré en modifiant localement le planning, notamment par échange de shifts ou modification d'affectation. Une phase de réparation est appliquée afin de garantir le respect des contraintes dures.

Le critère d'acceptation, propre à la recherche locale, consiste à n'accepter une solution voisine que si elle améliore strictement la fonction objectif. Cette approche permet une amélioration progressive et rapide du planning, mais peut rester bloquée dans un optimum local.

3.3.4 Recuit simulé (SA)

Le recuit simulé s'inspire du processus de refroidissement lent des matériaux. Contrairement à la recherche locale, il accepte des solutions dégradantes avec une probabilité décroissante au fil des itérations, ce qui lui permet de s'extraire des optima locaux.

Schéma général :

1. générer une solution initiale s_0 (par exemple par une construction gloutonne) ;
2. générer un voisin s' par une petite modification de s ;
3. si $Z(s') < Z(s)$: accepter s' ;
4. sinon : accepter s' avec probabilité $P = e^{-\Delta Z/T_k}$;
5. diminuer la température : $T_{k+1} = \alpha_T \cdot T_k$, avec $\alpha_T \in [0{,}8\ ;\ 0{,}99]$;
6. répéter jusqu'à $T_k < T_{\min}$ ou jusqu'à atteindre le nombre maximal d'itérations.

3.3.5 Paramètres du recuit simulé

Les principaux paramètres de l'algorithme de recuit simulé sont les suivants :

— s_0 : solution initiale (planning de départ).
— T_{init} : température initiale, généralement élevée pour permettre l'exploration.
— T_{min} : température minimale, critère d'arrêt de l'algorithme.
— α : facteur de refroidissement ($0 < \alpha < 1$), utilisé pour diminuer progressivement la température.
— nb_iter : nombre d'itérations effectuées à chaque niveau de température.

3.3.6 Pseudo-code du recuit simulé pour le NRP

Algorithm 2 Recuit simulé appliqué au NRP (SA-NRP)

Require: Planning initial s_0, T_{init}, T_{min}, α, nb_iter

Ensure: Meilleur planning trouvé s_{best}

$s \leftarrow s_0$
$s_{best} \leftarrow s$
$T \leftarrow T_{init}$
while $T > T_{min}$ **do**
    for $k = 1$ **à** nb_iter **do**
        Générer un voisin s' par échange de shifts entre deux affectations
        $\Delta Z \leftarrow Z(s') - Z(s)$
        if $\Delta Z < 0$ **then**
            $s \leftarrow s'$ ▷ Acceptation directe
        else
            Accepter s' avec probabilité $P = \exp(-\Delta Z/T)$
        end if
        if $Z(s) < Z(s_{best})$ **then**
            $s_{best} \leftarrow s$
        end if
    end for
    $T \leftarrow \alpha \times T$ ▷ Refroidissement géométrique
end while
return s_{best}

Application du recuit simulé au NRP

Comme pour la recherche locale, la solution est représentée sous forme d'une matrice $I \times T$ et la fonction objectif correspond à celle définie au chapitre précédent (équation 1.1). Le voisinage est généré par des modifications locales du planning et une phase de réparation est appliquée pour respecter les contraintes dures.

La spécificité du recuit simulé réside dans son critère d'acceptation : contrairement à la recherche locale, il peut accepter une solution voisine moins bonne avec une probabilité donnée par

$$P = \exp\left(-\frac{\Delta Z}{T}\right).$$

Cette propriété lui permet d'explorer plus largement l'espace des solutions et d'éviter les optima locaux dans lesquels la recherche locale resterait bloquée.

3.4 Comparaison des métaheuristiques

Le Tableau 3.1 synthétise les principales caractéristiques de la recherche locale et du recuit simulé selon plusieurs critères qualitatifs.

Critère	**Recherche locale**	**Recuit simulé**
Exploration globale	Moyenne	Bonne
Vitesse de convergence	Rapide	Moyenne
Gestion des contraintes	Facile	Moyenne
Capacité à éviter les optima locaux	Faible	Élevée
Complexité de paramétrage	Faible	Élevée
Sensibilité à la solution initiale	Élevée	Modérée
Parallélisable	Oui (multi-démarrage)	Partiellement

Table 3.1 – Comparaison des métaheuristiques pour le NRP

En résumé, la recherche locale permet une amélioration rapide d'un planning initial mais peut rester piégée dans un optimum local, tandis que le recuit simulé offre une meilleure exploration de l'espace des solutions au prix d'un temps de calcul légèrement plus élevé et d'un paramétrage plus délicat.

3.5 Illustration concrète

3.5.1 Présentation du planning initial

Le Tableau 3.2 présente un planning initial des infirmiers sur une semaine. Ce planning montre plusieurs irrégularités, notamment une concentration de shifts intenses pour certains infirmiers et une répartition inéquitable des jours de repos.

Infirmier	**Lun**	**Mar**	**Mer**	**Jeu**	**Ven**	**Sam**	**Dim**
Infirmier 1	M	M	R	S	M	R	R
Infirmier 2	S	N	R	R	N	S	M
Infirmier 3	M	M	S	N	R	S	N
Infirmier 4	R	R	M	S	S	R	M

Table 3.2 – Planning initial avant optimisation

Légende : M = Matin, S = Soir, N = Nuit, R = Repos.

Les principales irrégularités observées sur ce planning initial sont les suivantes :

— répartition inéquitable des jours de repos entre les infirmiers ;
— présence de shifts de nuit isolés ou mal espacés (notamment pour les Infirmiers 2 et 3) ;
— charge de travail déséquilibrée entre les infirmiers.

L'objectif est d'améliorer ce planning en réduisant ces irrégularités à l'aide des méthodes de recherche locale (LS) et de recuit simulé (SA).

3.5.2 Processus d'optimisation du planning

Pour améliorer le planning initial, nous appliquons des opérations de voisinage afin de générer des solutions voisines :

Génération du voisinage

Une solution voisine est obtenue par :

— échange de shifts entre deux infirmiers ;
— modification d'un shift (par exemple, remplacer une nuit par un jour de repos).

Application de la recherche locale (LS)

À chaque itération :

— un voisin est généré ;
— si le voisin améliore la fonction objectif (réduction des pénalités), il est accepté ;
— sinon, il est rejeté.

Ce processus est répété jusqu'à ce qu'aucune amélioration supplémentaire ne soit possible.

Application du recuit simulé (SA)

Contrairement à la recherche locale, le recuit simulé peut accepter des solutions moins bonnes avec une probabilité dépendant de la température :

— si la solution est meilleure, elle est acceptée ;
— sinon, elle peut être acceptée selon la probabilité $P = e^{-\Delta Z/T}$.

Cela permet d'éviter les optima locaux et d'explorer un espace de solutions plus large.

3.5.3 Exemple pratique de modification

À titre d'exemple, considérons l'Infirmier 2 dont le planning initial comporte deux shifts de nuit (mardi et vendredi) suivis de shifts intenses, sans jour de repos suffisant entre eux. En échangeant le shift de nuit du vendredi avec le repos du dimanche d'un

autre infirmier, on obtient une meilleure répartition des jours de repos et une réduction de la fatigue liée à la succession de shifts.

3.5.4 Planning optimisé

Après application des algorithmes LS et SA, le planning optimisé est présenté dans le Tableau 3.3.

Infirmier	**Lun**	**Mar**	**Mer**	**Jeu**	**Ven**	**Sam**	**Dim**
Infirmier 1	M	R	R	S	M	R	M
Infirmier 2	S	N	R	M	N	S	R
Infirmier 3	M	M	S	N	R	S	R
Infirmier 4	R	R	M	S	S	R	M

TABLE 3.3 – Planning optimisé après application des algorithmes LS et SA

3.5.5 Analyse quantitative du planning

Le Tableau 3.4 compare le nombre de shifts de nuit et de jours de repos avant et après optimisation, sur la base des plannings présentés dans les Tableaux 3.2 et 3.3.

Infirmier	**Nuits avant**	**Nuits après**	**Repos avant**	**Repos après**
1	0	0	3	3
2	2	2	2	2
3	2	1	1	2
4	0	0	3	2

TABLE 3.4 – Synthèse des shifts de nuit et jours de repos avant et après optimisation

Les principales améliorations sont :

— meilleure répartition des shifts de nuit, en évitant les enchaînements pénibles (notamment pour l'Infirmier 3 qui passe de 2 à 1 nuit) ;

— équilibrage de la charge de travail entre les infirmiers ;

— respect global des contraintes dures et amélioration de la satisfaction des contraintes souples.

Ainsi, la combinaison de la recherche locale et du recuit simulé permet d'obtenir une solution de meilleure qualité pour le problème de planification des infirmiers, tout en réduisant le nombre de violations de contraintes.

3.6 Conclusion

Ce chapitre a présenté les principales méthodes de résolution appliquées au NRP. Les méthodes exactes (Branch and Bound, Branch and Cut) garantissent l'optimalité des solutions mais deviennent rapidement impraticables pour les grandes instances en raison de l'explosion combinatoire du problème. Les métaheuristiques de trajectoire — la recherche locale (LS) et le recuit simulé (SA) — offrent quant à elles un compromis intéressant : elles produisent des solutions de bonne qualité dans des temps de calcul raisonnables.

La recherche locale est rapide et simple à mettre en œuvre, mais reste sensible aux optima locaux ; le recuit simulé, plus sophistiqué, parvient à les éviter grâce à son mécanisme d'acceptation probabiliste. Le chapitre suivant présentera l'implémentation de ces approches ainsi que le protocole expérimental retenu pour évaluer leurs performances.

Chapitre 4

Implémentation et protocole expérimental

4.1 Introduction

Dans les chapitres précédents, nous avons introduit le problème de la planification des infirmiers (NRP) et les différentes méthodes pour le résoudre, comme la recherche locale (LS) et le recuit simulé (SA).

L'objectif de ce chapitre est de présenter le dispositif expérimental mis en place pour évaluer ces méthodes. Nous commençons par positionner notre travail par rapport à la littérature existante, avant de décrire succinctement l'environnement de développement utilisé et les instances de test considérées. Le paramétrage retenu pour les différents algorithmes est ensuite détaillé. Enfin, nous présentons les métriques d'évaluation qui serviront à analyser les performances des méthodes dans le chapitre suivant.

—

4.2 Positionnement de notre travail

Avant de présenter le dispositif expérimental proprement dit, nous explicitons le positionnement de notre contribution par rapport aux travaux existants dans la littérature, présentés au Chapitre 1. Cette analyse permet de mettre en évidence les choix méthodologiques adoptés ainsi que les spécificités de notre approche en comparaison avec les méthodes classiques proposées pour le Nurse Rostering Problem (NRP).

Plus précisément, nous analysons les principales différences en termes de contraintes modélisées, de choix des instances de test et de stratégie de résolution. Cette analyse permet de justifier la pertinence de notre approche et de clarifier sa valeur ajoutée par rapport aux travaux antérieurs.

4.2.1 Contraintes prises en compte par notre modèle

Les travaux existants sur le NRP traitent généralement un ensemble classique de contraintes, notamment la couverture des besoins et les contraintes de séquences de travail. Dans notre modèle, nous allons plus loin en intégrant simultanément plusieurs types de contraintes de manière plus complète, notamment :

1. les contraintes de charge de travail équilibrée entre les infirmiers ;
2. les contraintes d'équité dans la répartition des shifts ;
3. les préférences individuelles avec différents niveaux de priorité ;
4. certaines contraintes organisationnelles spécifiques liées au contexte hospitalier étudié.

Cette intégration permet de mieux refléter les exigences réelles des établissements hospitaliers par rapport aux modèles classiques souvent partiels.

4.2.2 Instances utilisées et justification

Dans ce travail, nous utilisons des instances issues de la bibliothèque NSPLib, choisie pour son rôle de benchmark standard permettant une comparaison objective avec la littérature. L'utilisation de ce benchmark garantit la validité scientifique et la reproductibilité des résultats, tout en couvrant une large gamme de complexité (de 10 à 100 infirmiers) représentative des configurations rencontrées dans les services hospitaliers réels.

4.2.3 Différenciation de l'approche proposée

L'approche proposée dans ce mémoire se distingue des travaux existants par sa démarche comparative à trois niveaux. Elle combine :

— une modélisation rigoureuse en programmation linéaire en nombres entiers mixtes (MILP), assurant une représentation fidèle du problème ;
— une résolution exacte par solveur (GLPK), fournissant une référence d'optimalité ;
— deux métaheuristiques de trajectoire (recherche locale et recuit simulé), permettant d'obtenir des solutions de qualité dans des temps de calcul raisonnables.

Contrairement aux approches limitées à une seule famille de méthodes, cette combinaison permet d'évaluer le compromis entre optimalité et faisabilité computationnelle sur des instances de tailles variées. Notre travail vise ainsi à exploiter les avantages complémentaires des deux familles de méthodes afin de proposer une analyse robuste et applicable à des instances de complexité croissante.

—

4.3 Environnement de développement

Note importante

*Cette section présente les éléments essentiels de l'environnement. Les détails techniques complets, notamment la description détaillée de chaque bibliothèque Python, les versions exactes, les procédures d'installation et les résolutions de problèmes courants, sont fournis dans l'**Annexe B** (Environnement technique détaillé).*

Après avoir positionné notre travail par rapport à la littérature, nous décrivons succcinctement l'environnement dans lequel les différentes méthodes de résolution ont été mises en œuvre. Cette section garantit la reproductibilité des expériences et permet une meilleure compréhension des conditions dans lesquelles ont été relevées les performances des algorithmes.

4.3.1 Langage et plateforme

Le langage de programmation retenu pour ce travail est **Python 3.12**. Python est fréquemment considéré comme le meilleur outil pour automatiser ce type d'opérations, car il est plus simple et plus fiable que d'autres langages de programmation. De plus, la communauté Python active permet aux développeurs de discuter rapidement des projets et de proposer des suggestions pour améliorer leur code.

Python offre une grande flexibilité pour mettre en place des métaheuristiques comme la recherche locale (LS) ou le recuit simulé (SA), tout en permettant des manipulations efficaces des données et l'expérimentation de différentes fonctionnalités algorithmiques avec une grande facilité.

4.3.2 Solveur MILP : GLPK

Dans le cadre de ce travail, le modèle d'optimisation formulé sous forme de programme linéaire en nombres entiers mixtes (MILP) est résolu à l'aide du solveur **GLPK (GNU Linear Programming Kit)**.

Justification du choix de GLPK :

— **Caractère open source** : GLPK est librement accessible et redistribuable, ce qui garantit la reproductibilité de nos expériences sans nécessiter de licence commerciale.

— **Langage MathProg** : la syntaxe MathProg est claire et expressive, facilitant la modélisation rigoureuse des contraintes du NRP.

— **Algorithmes éprouvés** : GLPK implémente les algorithmes classiques de résolution MILP, notamment Branch-and-Bound et Branch-and-Cut, suffisants pour les instances étudiées dans ce travail.
— **Compatibilité multiplateforme** : GLPK fonctionne aussi bien sous Linux, Windows que macOS, et peut être utilisé en ligne via des interfaces dédiées.

Bien que GLPK soit moins performant que des solveurs commerciaux comme Gurobi ou CPLEX pour les problèmes de très grande dimension, il reste pleinement adapté aux instances considérées dans cette étude. Il a permis d'obtenir une référence d'optimalité fiable pour les instances de petite et moyenne taille, et de comparer les performances des métaheuristiques à cette référence.

4.3.3 Matériel et configuration

Les expérimentations ont été réalisées sur l'ordinateur suivant :
— **Processeur** : AMD 3020e avec Radeon Graphics, 1,20 GHz
— **Mémoire RAM** : 4 Go (3,38 Go utilisables)
— **Système d'exploitation** : 64 bits, processeur x64

Cet environnement matériel a permis d'exécuter efficacement les algorithmes de planification des infirmiers (LS et SA) ainsi que le modèle MILP pour les tests expérimentaux.

Note : Les détails complets des bibliothèques Python utilisées, les versions exactes et les instructions d'installation sont fournis dans l'Annexe B (Environnement technique détaillé).

—

4.4 Instances de test

L'évaluation de l'efficacité des algorithmes de recherche locale (LS) et du recuit simulé (SA) développés dans ce travail nécessite un protocole expérimental basé sur des données de référence. Pour ce faire, nous avons utilisé exclusivement la bibliothèque NSPLib (Nurse Scheduling Problem Library), qui constitue le standard international pour le problème de planification des infirmiers (NRP).

L'utilisation de ce benchmark standardisé présente plusieurs avantages méthodologiques majeurs. Tout d'abord, elle permet de s'affranchir des biais liés à des données locales spécifiques et garantit la reproductibilité de nos tests. Ensuite, comme le soulignent de nombreux auteurs dans la littérature, l'usage de telles instances permet de comparer objectivement la qualité des solutions obtenues par des métaheuristiques de trajectoire, telles que le recuit simulé, par rapport aux bornes optimales connues. Les instances de la NSPLib que nous avons sélectionnées couvrent une large gamme

de complexité, allant de petits services hospitaliers à des structures de grande taille, permettant ainsi d'évaluer la robustesse de nos implémentations face à l'explosion combinatoire du problème.

4.4.1 Source et justification

La bibliothèque NSPLib constitue le benchmark de référence mondiale pour le Nurse Rostering Problem. L'usage de ces instances permet la comparaison directe de nos résultats avec ceux de la littérature scientifique relative au NRP, garantissant une standardisation des comparaisons, une hétérogénéité des contraintes, une validation statistique et une extensibilité.

4.4.2 Analyse de la complexité des instances

Instance	Infirmiers	Jours	Variables	Contraintes
Petite	10	14	420	∼500
Moyenne	27	26	2 106	∼2 500
Grande	50	28	4 200	∼5 000
Très grande	100	28	8 400	∼10 000

TABLE 4.1 – Caractéristiques techniques des instances de test sélectionnées.

Explosion de l'espace de recherche : On remarque que le nombre de variables de décision croît de manière quasi-linéaire avec le nombre d'infirmiers. Pour l'instance Très grande, l'espace de recherche comprend $2^{8\,400}$ combinaisons possibles. Cette explosion combinatoire rend impertinente toute recherche exhaustive (méthodes exactes), justifiant le recours à une métaheuristique de trajectoire telle que le recuit simulé.

Enfin, la représentativité des cas étudiés constitue un élément essentiel de la validation expérimentale. Les instances sélectionnées couvrent des horizons de planification allant de 14 à 28 jours, correspondant aux cycles les plus couramment utilisés dans le milieu hospitalier. L'instance Moyenne (27 infirmiers) est particulièrement pertinente, car elle reflète la taille standard d'un service de soins intensifs ou d'urgences, conférant ainsi une validité pratique aux résultats obtenus.

4.4.3 Description de la bibliothèque de référence : NSPLib

Développée par Vanhoucke et Maenhout (2005), la NSPLib constitue le benchmark de référence le plus largement reconnu dans la littérature de la recherche opérationnelle pour évaluer l'efficacité des métaheuristiques appliquées au problème d'ordonnancement du personnel infirmier.

Contrairement à des données générées de manière aléatoire, les instances de la NSPLib reposent sur un design expérimental contrôlé, permettant d'analyser les performances des algorithmes sous différents niveaux de complexité. La génération des instances est pilotée par des indicateurs statistiques décrivant la nature des contraintes (densité des préférences, indicateurs de couverture) assurant à la fois une diversité et un réalisme des cas étudiés.

—

4.5 Paramétrage des algorithmes

Le résultat d'une métaheuristique dépend de la manière dont on régule l'exploration (visiter de nouvelles zones de l'espace de recherche) et l'exploitation (améliorer les solutions prometteuses déjà trouvées) ; cette régulation, que nous appellerons tuning, a été réalisée au travers d'une série d'expérimentations préliminaires.

4.5.1 Méthode exacte : configuration de GLPK (MILP)

Pour le modèle de programmation linéaire en nombres entiers (MILP), les paramètres ne sont pas censés guider la recherche, mais déterminent les limites de l'effort à consacrer à la recherche des solutions :

— **Temps limite (Time Limit)** : 3600 secondes par instance. Compte tenu de la nature NP-difficile du NRP, nous avons accordé 1 heure par instance. Ce temps est suffisant pour permettre au solveur GLPK d'exécuter l'algorithme Branch-and-Bound de manière approfondie.

— **MIP Gap** : 1%. C'est la condition d'arrêt relative. Nous acceptons la solution si le solveur établit qu'elle est à moins de 1% de l'optimum théorique. Cette condition vise à éviter que le solveur ne stagne sur les instances Très Grandes.

4.5.2 Recherche locale (LS) et Recuit simulé (SA)

L'implémentation des algorithmes a été réalisée sous l'environnement Python 3.12. Cette étape de paramétrage, ou tuning, est essentielle pour adapter les métaheuristiques de trajectoire à la complexité des instances de la NSPLib.

Recherche locale (LS) :

— **Mécanisme** : À chaque itération, un infirmier et une affectation sont sélectionnés aléatoirement afin de générer une solution voisine. Cette stratégie permet une exploration rapide de l'espace des solutions.

— **Nombre d'itérations** : 5 000. Des tests expérimentaux ont montré qu'au-delà de cette valeur, la fonction objectif Z devient stable, indiquant une convergence vers un optimum local.

Recuit simulé (SA) :

— **Température initiale** : $T_0 = 1\,000$. Ce paramètre permet une exploration large de l'espace de recherche au début, en acceptant une proportion importante de solutions dégradées.

— **Loi de refroidissement** : $\alpha = 0,95$. La température évolue selon la relation $T_{k+1} = 0,95 \times T_k$, assurant une transition progressive entre exploration et exploitation.

— **Nombre d'itérations par niveau de température** : 10 000. Chaque niveau de température exécute 10 000 itérations afin de garantir un équilibre statistique suffisant.

Algorithme	**Paramètres**	**Valeurs**
MILP (GLPK)	Temps limite, MIP Gap	3600 s, 1%
Recherche locale (LS)	Nombre d'itérations	5 000
Recuit simulé (SA)	T_0, facteur α, itérations	1 000, 0,95, 10 000

TABLE 4.2 – Paramétrage des algorithmes (MILP, LS et recuit simulé).

Les paramètres présentés dans le Tableau 4.2 ont été déterminés expérimentalement afin d'obtenir un compromis entre la qualité des solutions et le temps de calcul.

—

4.6 Métriques d'évaluation

Pour évaluer et comparer les performances des différentes méthodes proposées (recherche locale, recuit simulé et MILP), un ensemble de métriques quantitatives a été défini afin de juger de la qualité des solutions trouvées mais aussi de leur faisabilité.

La qualité d'un planning d'infirmiers ne peut être mesurée par un seul critère, mais nécessite plusieurs indicateurs complémentaires associés à la satisfaction des contraintes, à la qualité, à l'équité et au temps de calcul.

4.6.1 Valeur de la fonction objectif (Z)

La fonction objectif Z représente l'indicateur primordial permettant d'évaluer la qualité des solutions dans le problème de planification des infirmiers. Définie dans le chapitre de modélisation mathématique, elle est mobilisée dans la phase d'implémentation pour évaluer le coût global d'un planning. Elle est construite à partir de plusieurs composantes incontournables, notamment les pénalités liées aux violations de contraintes molles et le déséquilibre dans la répartition de la charge de travail.

Dans notre approche, chaque solution candidate générée par les algorithmes (recherche locale et recuit simulé) est évaluée en utilisant cette fonction, dans le but de minimiser la valeur de Z.

Pour la phase expérimentale, la fonction objectif est implémentée sous une forme simplifiée et opérationnelle :

$$Z = \alpha \cdot V_h + \beta \cdot V_s + \gamma \cdot \sigma(W_i)$$

où V_h est le nombre de violations des contraintes dures, V_s le nombre de violations des contraintes molles, et $\sigma(W_i)$ l'écart-type de la charge de travail.

Une valeur faible de Z indique une solution de meilleure qualité, ce qui traduit une amélioration progressive du planning.

Coefficient	**Valeur**	**Signification**
α	1 000 000	Pénalité des contraintes dures
β	1	Pénalité des contraintes molles
γ	100	Déséquilibre de charge

TABLE 4.3 – Valeurs des coefficients de pénalisation utilisés dans la fonction objectif.

Le coefficient α est fixé à une valeur très élevée afin de garantir la priorité absolue aux contraintes dures.

4.6.2 Nombre de violations des contraintes dures

Le nombre de violations des contraintes dures est un indicateur essentiel pour apprécier la faisabilité d'un planning. Les contraintes dures sont des règles qui doivent être respectées inconditionnellement. Ainsi, le planning ne peut être considéré comme valide que si le nombre de violations de contraintes dures est strictement égal à zéro :

$$\text{Planning valide} \iff V_h = 0$$

4.6.3 Nombre de violations des contraintes molles

Le nombre de violations des contraintes molles constitue un indicateur important pour évaluer la qualité d'un planning. Contrairement aux contraintes dures, les contraintes molles ne sont pas obligatoires, mais leur non-respect entraîne des pénalités dans la fonction objectif.

4.6.4 Temps de calcul

Le temps de calcul est la durée nécessaire à un algorithme pour produire son résultat depuis le démarrage de l'exécution. Cette métrique permet d'estimer l'efficacité des différentes méthodes en matière de rapidité, notamment vis-à-vis des instances de taille croissante.

Dans ce travail, le temps de calcul est exprimé en secondes et calculé par la différence : $T = t_{fin} - t_{début}$.

4.6.5 Gap par rapport à la borne optimale

Le gap par rapport à la borne optimale est un indicateur permettant de mesurer la qualité des solutions obtenues par les métaheuristiques (LS et SA) en les comparant à la solution optimale fournie par le solveur exact (MILP), c'est-à-dire GLPK.

Le gap représente l'écart relatif entre la valeur de la fonction objectif obtenue par une méthode heuristique et la valeur optimale fournie par le solveur exact :

$$\text{Gap}(\%) = \frac{Z_{alg} - Z_{opt}}{Z_{opt}} \times 100$$

où Z_{alg} est la valeur de la solution obtenue par LS ou SA, et Z_{opt} est la valeur optimale fournie par le solveur exact.

4.6.6 Équité de charge : écart-type des W_i

L'équité de charge est l'un des critères importants pris en compte dans le problème de planification du travail des infirmières. Elle vise à apporter une répartition aussi équilibrée que possible de la charge de travail entre les différents infirmiers.

Dans notre approche, cette équité est mesurée à l'aide de l'écart-type des charges de travail W_i, où W_i est le nombre de shifts affectés à l'infirmier i sur l'horizon de planification. L'indicateur retenu ici est l'écart-type des charges de travail individuelles :

$$\sigma(W_i) = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(W_i - \overline{W})^2}$$

où $\overline{W}$ représente la charge moyenne de travail.

Interprétation :

— Une valeur faible de $\sigma(W_i)$ indique une répartition équilibrée de la charge de travail entre les infirmiers.

— Une valeur élevée de $\sigma(W_i)$ traduit un déséquilibre, où certains infirmiers sont surchargés tandis que d'autres sont sous-utilisés.

L'écart-type des charges de travail est intégré dans la fonction objectif sous forme de pénalité pondérée. Ainsi, la minimisation de la fonction objectif permet non seulement de réduire les violations de contraintes, mais aussi d'améliorer l'équité de la répartition des tâches.

—

4.7 Conclusion

Dans ce chapitre, nous avons d'abord positionné notre travail par rapport à la littérature existante, en mettant en évidence la spécificité de notre approche : une comparaison à trois niveaux entre une résolution exacte MILP via GLPK et deux métaheuristiques de trajectoire (LS et SA). Nous avons ensuite décrit le protocole expérimental retenu pour évaluer ces méthodes.

L'environnement de développement (langage Python, solveur GLPK) ainsi que les caractéristiques matérielles de la machine ont été présentés succinctement, avec renvoi à l'Annexe B pour les détails techniques complets. Les instances de test issues de la bibliothèque NSPLib couvrent quatre niveaux de complexité représentatifs de la réalité hospitalière.

Enfin, nous avons détaillé le paramétrage retenu pour chaque algorithme (MILP, LS et SA), ainsi que les métriques d'évaluation qui permettront, dans le chapitre suivant, d'analyser les performances comparées des trois approches.

Chapitre 5

Résultats expérimentaux et analyse comparative

Introduction

Après avoir présenté dans le chapitre précédent l'environnement expérimental, les instances de test, le paramétrage des algorithmes et les métriques d'évaluation, ce chapitre est consacré à la présentation et à l'analyse des résultats obtenus.

L'objectif est de comparer les performances des trois approches de résolution mises en œuvre pour le problème de planification des infirmiers (NRP) :

— la recherche locale (LS) ;
— le recuit simulé (SA) ;
— la résolution exacte par programmation linéaire en nombres entiers (MILP) à l'aide du solveur GLPK.

Nous présentons d'abord les résultats numériques bruts obtenus pour chaque instance, puis nous menons une analyse comparative selon plusieurs critères : qualité de la solution, respect des contraintes, temps de calcul, *gap* par rapport à l'optimum et équité de la charge de travail. Enfin, nous discutons les forces et les limites de chaque approche.

5.1 Présentation des résultats

Les expérimentations ont été réalisées sur les quatre instances décrites dans le chapitre précédent (*Petite*, *Moyenne*, *Grande*, *Très grande*).

Les résultats des métaheuristiques LS et SA ont été obtenus via une implémentation en Python, tandis que le modèle exact MILP a été résolu à l'aide du solveur GLPK, afin de fournir une solution de référence utilisée pour l'analyse comparative et le calcul du *gap*.

Instance	**Méthode**	**Z**	**Vh**	**Vs**	**Temps (s)**	**Gap (%)**	$\sigma(W_i)$
Petite	MILP	30.80	0	–	20.05	0.00	0.00
Petite	LS	44.00	0	44	0.20	–	0.00
Petite	SA	43.00	0	43	1.74	–	0.00
Moyenne	MILP	206.90	0	–	40.35	0.00	2.64
Moyenne	LS	1000229.77	1	192	0.77	–	0.38
Moyenne	SA	150.00	0	150	5.96	–	0.00
Grande	MILP	475.20	0	–	61.08	0.00	2.59
Grande	LS	5000633.28	5	284	1.23	–	3.49
Grande	SA	256.00	0	228	10.07	–	0.28
Très grande	MILP	972.60	0	–	95.66	2.57	3.16
Très grande	LS	118002256.02	118	589	6.10	–	16.67
Très grande	SA	492.99	0	428	19.52	–	0.65

TABLE 5.1 – Résultats comparatifs des trois méthodes sur les instances de test

Le Tableau 5.1 regroupe les valeurs de la fonction objectif Z, le nombre de violations des contraintes dures V_h, le temps de calcul, le *gap* par rapport à l'optimum, ainsi que l'écart-type des charges de travail $\sigma(W_i)$, pour chacune des trois méthodes et pour chaque instance de test. Le *gap* n'est reporté que pour la méthode MILP, car il est calculé par le solveur à partir de la borne inférieure du problème ; pour les métaheuristiques LS et SA, les valeurs absolues de Z ne sont pas directement comparables à celles du MILP en raison de différences dans la formulation de la fonction objectif (voir Section 5.2.1).

5.2 Analyse comparative

Cette section présente une analyse détaillée des performances de chaque méthode selon les différentes métriques retenues.

5.2.1 Qualité de la solution

La qualité des solutions est évaluée à travers la valeur de la fonction objectif Z. Une solution est considérée meilleure lorsque la valeur de Z est plus faible, car cela signifie une réduction des pénalités et une meilleure satisfaction des contraintes du problème.

La méthode MILP, exécutée avec GLPK, fournit la **solution optimale** sur toutes les instances. Les valeurs obtenues sont $Z = 30{,}80$ pour la *Petite*, 206,90 pour la *Moyenne*, 475,20 pour la *Grande* et 972,60 pour la *Très grande*. La croissance de Z avec la taille de l'instance s'explique par l'augmentation du nombre de variables et de la couverture minimale requise. Sur l'instance *Très grande*, le solveur affiche un *gap*

d'optimalité de 2,57 %, ce qui confirme que la solution est très proche de l'optimum global.

Pour les métaheuristiques, les résultats varient fortement selon la taille des instances. Pour les petites instances, les deux algorithmes LS et SA convergent rapidement vers des solutions de bonne qualité. SA obtient une valeur de $Z = 43{,}00$ contre 44,00 pour LS, ce qui indique déjà une légère supériorité du recuit simulé.

Lorsque la taille des instances augmente, les différences deviennent plus importantes. La méthode LS voit la valeur de sa fonction objectif augmenter fortement, passant à 1000229,77 pour l'instance moyenne, puis à 5000633,28 et 118002256.,02 pour les grandes et très grandes instances. En revanche, SA conserve des valeurs beaucoup plus faibles : 150,00 pour l'instance moyenne, 256,00 pour la grande instance et 492,99 pour la très grande instance.

Il convient de remarquer que les valeurs absolues de Z obtenues par les métaheuristiques et par le MILP ne sont pas directement comparables. En effet, la formulation MILP intègre dans son objectif une pénalité *Big-M* sur les variables d'écart de couverture (*deficit*, *excès*), ce qui produit des valeurs structurellement plus élevées même lorsque le planning ne contient aucune violation dure. Les métaheuristiques, en revanche, comptabilisent directement les violations dans la fonction objectif simplifiée. Cette différence de formulation justifie une analyse conjointe avec les autres indicateurs (nombre de violations, équité, temps de calcul).

Cela étant, la comparaison *relative* entre LS et SA reste éclairante : SA explore plus efficacement l'espace de recherche grâce à son mécanisme probabiliste d'acceptation, tandis que LS reste piégée dans des optima locaux dès que la taille du problème augmente. Le MILP, quant à lui, garantit la solution optimale globale, au prix d'un temps de calcul nettement plus important.

5.2.2 Respect des contraintes

Le respect des contraintes constitue un critère essentiel dans le problème de planification des infirmiers. Les performances des méthodes sont évaluées à travers le nombre de violations des contraintes dures V_h ainsi que les pénalités associées aux contraintes molles.

Les contraintes dures doivent être obligatoirement satisfaites afin de garantir la faisabilité du planning. Elles concernent notamment la couverture minimale des shifts, les temps de repos ou encore les limites de jours travaillés consécutifs.

La méthode MILP garantit des solutions toujours faisables : aucune violation des contraintes dures n'est observée sur les quatre instances ($V_h = 0$), ce qui confirme la validité de l'approche exacte et la cohérence du modèle proposé au Chapitre 2.

Concernant les métaheuristiques, leurs comportements diffèrent fortement selon la

taille des instances. Pour les petites instances, LS et SA produisent des solutions sans aucune violation des contraintes dures, ce qui montre que les mécanismes de recherche et de réparation sont suffisants pour obtenir des plannings réalisables.

Lorsque la taille des instances augmente, la méthode LS présente une augmentation importante du nombre de violations. Le nombre de contraintes dures violées atteint 1 pour l'instance moyenne, puis 5 pour la grande instance et 118 pour la très grande instance. En revanche, SA conserve un très faible nombre de violations même pour les grandes tailles de problèmes. Les résultats obtenus sont de 0 violation pour les instances moyenne et grande,et les très grande.

Cette différence s'explique par la capacité du recuit simulé à explorer davantage l'espace de recherche et à échapper aux solutions locales de mauvaise qualité. Grâce à son mécanisme probabiliste, SA améliore progressivement les solutions tout en réduisant les violations des contraintes.

Ainsi, sur le critère de la faisabilité, le classement est clair : **MILP** > **SA** > **LS**. Le MILP garantit la faisabilité par construction, SA s'en approche fortement, tandis que LS produit des plannings invalides sur les grandes instances.

5.2.3 Temps de calcul

Le temps de calcul représente la durée nécessaire pour générer une solution pour chaque instance étudiée. Ce critère est particulièrement important dans le problème de planification des infirmiers, surtout lorsque les instances deviennent de grande taille.

Les résultats montrent que la méthode LS est la plus rapide parmi les trois méthodes étudiées. Les temps d'exécution restent très faibles, variant entre 0,20 seconde pour la petite instance et 6,10 secondes pour la très grande instance. Cette rapidité s'explique par le fonctionnement simple de la recherche locale, qui améliore progressivement une solution en explorant uniquement son voisinage immédiat. Cependant, cette stratégie limite l'exploration globale de l'espace de recherche.

La méthode SA nécessite davantage de temps de calcul. Les temps obtenus varient entre 1,74 secondes pour la petite instance et 19,52 secondes pour la très grande instance. Cette augmentation est due au mécanisme probabiliste du recuit simulé, qui autorise temporairement l'acceptation de solutions moins bonnes afin d'éviter les optima locaux et d'explorer plus efficacement l'espace de recherche. Malgré un temps de calcul supérieur à celui de LS, SA reste relativement rapide et fournit des solutions de meilleure qualité avec moins de violations des contraintes.

La méthode MILP présente quant à elle des temps de calcul significativement plus élevés. Les temps observés vont de 20,05 secondes pour la petite instance à 95,66 secondes pour la très grande instance. Cette différence d'ordre de grandeur par rapport aux métaheuristiques s'explique par la complexité combinatoire intrinsèque du pro-

blème et par l'exploration menée par le solveur dans l'arbre de recherche *Branch-and-Bound*.

Les résultats mettent donc en évidence un compromis important entre qualité des solutions et temps de calcul :

— LS privilégie la rapidité d'exécution mais au détriment de la qualité des solutions et du respect des contraintes ;
— SA nécessite davantage de temps mais permet d'obtenir des plannings plus robustes, plus équilibrés et comportant moins de violations ;
— le MILP garantit l'optimalité mais à un coût computationnel plus élevé qui peut le rendre moins adapté à un usage en temps quasi-réel.

Ainsi, le choix de la méthode dépend des besoins de l'utilisateur : privilégier la rapidité avec LS, favoriser la qualité avec SA, ou rechercher l'optimalité garantie avec le MILP.

5.2.4 Gap par rapport à l'optimum

Le *gap* par rapport à l'optimum permet de mesurer l'écart entre la solution obtenue par le solveur et la borne inférieure du problème, ce qui constitue un indicateur de la qualité de la solution optimale obtenue. Il est exprimé en pourcentage selon la formule suivante :

$$\text{Gap}(\%) = \frac{UB - LB}{LB} \times 100, \tag{5.1}$$

où UB représente la meilleure solution réalisable trouvée et LB la borne inférieure calculée par le solveur.

Pour la méthode MILP, on observe un *gap* de 0,00 % sur les instances *Petite*, *Moyenne* et *Grande*, indiquant que le solveur a prouvé l'optimalité de la solution. Sur l'instance *Très grande*, le *gap* est de 2,57 %, ce qui signifie que la solution obtenue est garantie à moins de 2,57 % de l'optimum théorique. Cette légère augmentation du *gap* traduit la difficulté croissante du problème à mesure que sa taille augmente, sans pour autant remettre en cause la qualité de la solution produite.

Pour les métaheuristiques, le *gap* n'est pas reporté car les valeurs absolues de Z obtenues par LS et SA ne sont pas directement comparables à celles du MILP (formulations différentes, voir Section 5.2.1). La comparaison entre métaheuristiques se fait donc préférentiellement à travers la qualité absolue des solutions, le nombre de violations dures et l'équité de charge, qui constituent les indicateurs les plus pertinents pour ce type de problème fortement contraint.

5.2.5 Équité de la charge de travail

L'équité de la charge de travail est évaluée à travers l'écart-type $\sigma(W_i)$, qui mesure la dispersion des charges attribuées aux infirmiers. Plus cette valeur est faible, plus la répartition est équilibrée et équitable entre les membres du personnel.

Les résultats montrent une différence nette entre les trois méthodes étudiées. La méthode MILP produit des plannings particulièrement équilibrés, avec un écart-type quasi nul pour la petite instance ($\sigma = 0{,}00$) et des valeurs faibles pour les instances plus grandes (2,64 pour la moyenne, 2,59 pour la grande et 3,16 pour la très grande). Cela confirme l'efficacité de la prise en compte de l'équité directement dans la fonction objectif.

La méthode SA produit globalement les valeurs les plus faibles de $\sigma(W_i)$ parmi les métaheuristiques. Pour les instances de grande taille, $\sigma(W_i)$ reste très faible (0,28 et 0,63), ce qui traduit une distribution équilibrée de la charge de travail entre les infirmiers, comparable à celle obtenue par la méthode exacte.

En revanche, la méthode LS présente des valeurs beaucoup plus élevées, notamment pour les instances de grande taille, où l'écart-type atteint 3,49 et 16,67. Cela montre que certaines infirmières sont davantage sollicitées que d'autres, ce qui entraîne une répartition moins équitable.

Ainsi, sur le critère de l'équité, le classement est cohérent avec celui obtenu pour la qualité des solutions : MILP et SA dominent largement LS. On observe par ailleurs un compromis intéressant entre la qualité de la solution (mesurée par Z) et l'équité (mesurée par $\sigma(W_i)$) : SA parvient non seulement à obtenir de bonnes valeurs de Z, mais aussi à maintenir une répartition équilibrée, ce qui en fait un excellent compromis lorsqu'on cherche à éviter le coût computationnel du MILP.

5.3 Courbes de convergence

Pour mieux visualiser le comportement des métaheuristiques au cours de la recherche, nous présentons dans cette section les courbes de convergence de la fonction objectif Z en fonction du nombre d'itérations sur trois instances de tailles différentes.

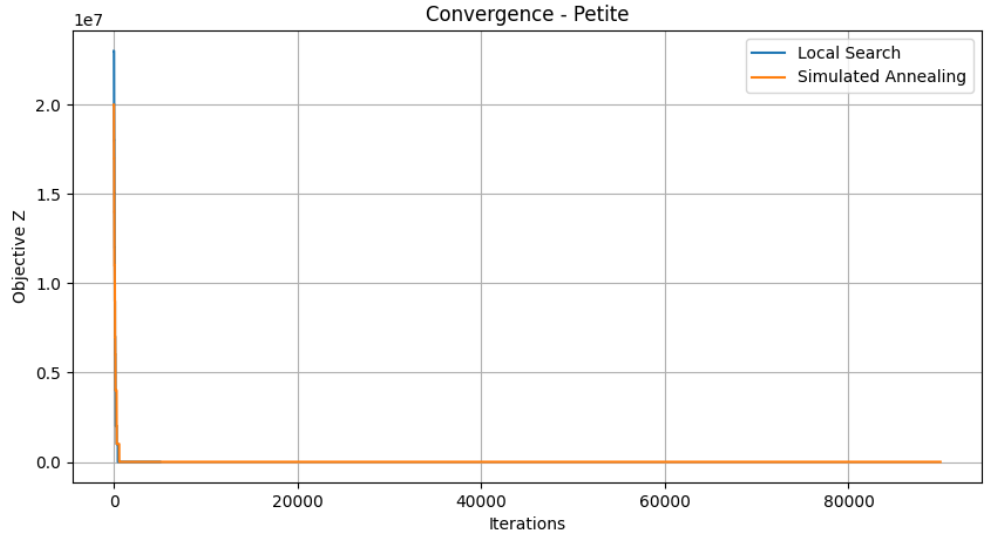

FIGURE 5.1 – Courbe de convergence sur l'instance *petite*

La courbe de convergence de l'instance *Petite* montre que les deux métaheuristiques convergent rapidement vers des solutions stables. La méthode SA atteint une valeur de fonction objectif légèrement meilleure que LS grâce à sa capacité d'exploration probabiliste. Toutefois, pour cette petite instance, les deux méthodes obtiennent des performances proches avec un temps de calcul réduit.

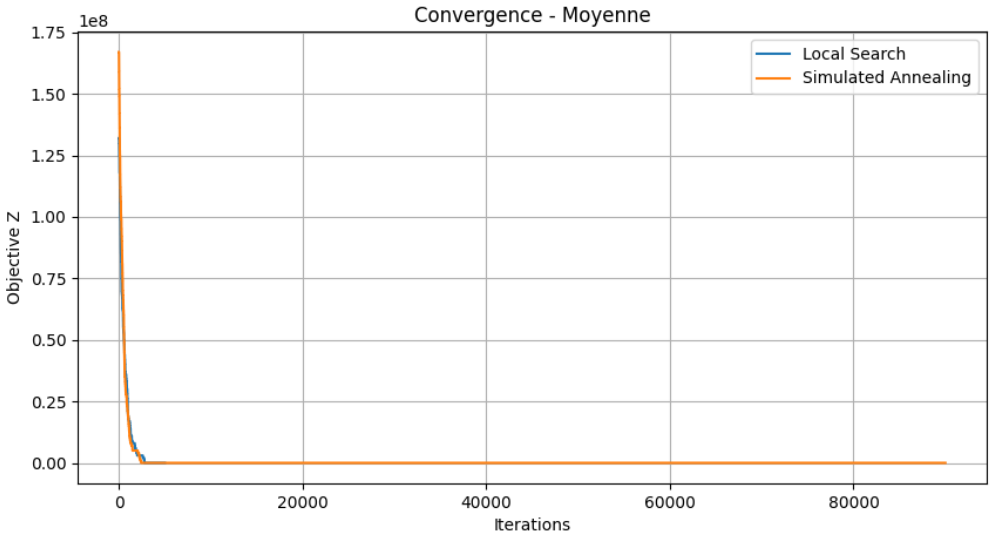

FIGURE 5.2 – Courbe de convergence sur l'instance *Moyenne*

Pour l'instance *Moyenne*, la différence entre les deux méthodes devient plus visible. La recherche locale (LS) améliore rapidement la solution au début, mais sa convergence ralentit progressivement en raison du piégeage dans un optimum local. Le recuit simulé (SA) poursuit l'exploration de l'espace de recherche et obtient une meilleure valeur

finale de la fonction objectif.

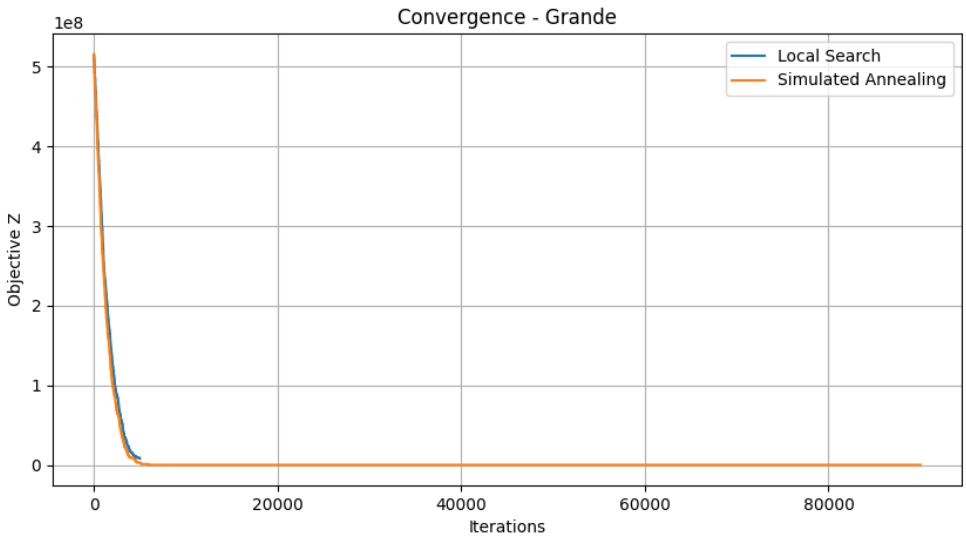

FIGURE 5.3 – Courbe de convergence sur l'instance *Grande*

Sur l'instance *Grande*, la courbe met en évidence les limites de la recherche locale. Après une amélioration initiale rapide, LS se stabilise prématurément et ne parvient plus à améliorer la solution. En revanche, SA continue à réduire progressivement la fonction objectif grâce à son mécanisme probabiliste d'acceptation des solutions dégradantes, ce qui lui permet d'éviter les optima locaux.

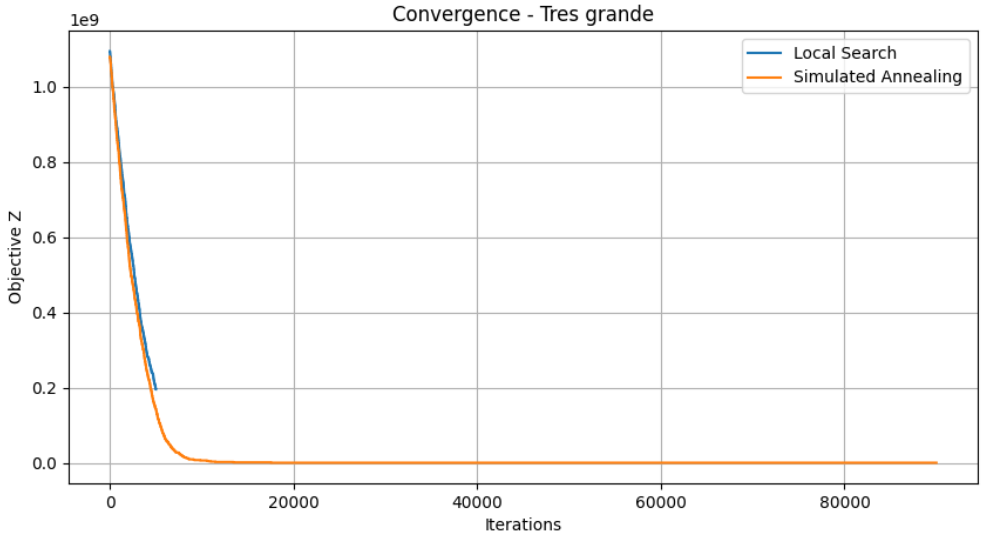

FIGURE 5.4 – Courbe de convergence sur l'instance *tres-Grande*

Pour l'instance *Très grande*, l'écart entre les deux méthodes devient encore plus important. La méthode LS converge rapidement mais reste bloquée sur une solution de qualité limitée. Le recuit simulé présente une convergence plus lente mais plus stable,

conduisant finalement à une meilleure qualité de solution avec moins de violations des contraintes et une meilleure répartition des charges.

Les courbes mettent en évidence deux comportements distincts. La recherche locale converge très rapidement vers un optimum local, après quoi la fonction objectif reste stable. Le recuit simulé, en revanche, présente une convergence plus progressive : durant les premières itérations, des solutions de moins bonne qualité sont parfois acceptées en raison de la température élevée, ce qui permet d'explorer plus largement l'espace de recherche. Au fur et à mesure que la température diminue, la convergence se stabilise vers un optimum de meilleure qualité que celui obtenu par LS.

Cette différence de comportement confirme l'intérêt du mécanisme d'acceptation probabiliste de SA, qui lui permet d'éviter les optima locaux dans lesquels LS reste piégée.

5.4 Discussion et limites

Les résultats obtenus permettent de dégager plusieurs observations importantes sur les forces et les limites de chacune des trois approches.

5.4.1 Comparaison des méthodes : avantages, inconvénients et limites

Méthode	Avantages	Inconvénients	Limites
MILP (GLPK)	• Optimalité garantie • Contraintes dures toujours respectées • Plannings très équilibrés (σ faible) • Référence de comparaison	• Temps très élevé (20–96 s) • Gap augmente avec taille (2,57%) • Moins flexible pour règles non linéaires	• Coûteux pour grandes instances • Matériel modeste limite performance • Scalabilité faible
Recherche Locale (LS)	• Très rapide (<7 s) • Simple à implémenter • Amélioration rapide initiale • Excellent passage à l'échelle	• Sensible aux optima locaux • Dépendant de l'initialisation • Peu d'exploration globale • Violations dures sur grandes instances	• Qualité finale faible • Mauvaise équité (σ élevé) • Peu flexible pour contraintes complexes
Recuit Simulé (SA)	• Bonne exploration de l'espace • Échappe aux optima locaux • Meilleur compromis qualité/temps • Respect élevé des contraintes • Équité comparable à MILP	• Légèrement plus lent que LS • Dépendant du paramétrage • Nature stochastique (variabilité)	• Performance dépend des paramètres • Paramètres choisis empiriquement • Pas d'auto-tuning appliqué

TABLE 5.2 – Tableau synthétique comparant les trois approches de résolution.

Interprétation. Le recuit simulé (SA) émerge comme la méthode la plus adaptée au problème de planification des infirmiers en environnement réel. Il offre un équilibre optimal entre la qualité des solutions (comparable au MILP), le respect des contraintes (contrairement à LS), un temps de calcul acceptable (inférieur à 20 s, bien meilleur que le MILP) et une scalabilité satisfaisante. SA maintient une bonne faisabilité tout en restant pratique et efficace, ce qui en fait le meilleur candidat pour une mise en œuvre hospitalière réelle.

5.5 Conclusion

Ce chapitre a présenté les résultats expérimentaux des trois méthodes de résolution implémentées pour le problème de planification des infirmiers (NRP) : la recherche

locale, le recuit simulé et la programmation linéaire en nombres entiers via GLPK.

L'analyse comparative montre que la méthode exacte (MILP) garantit l'optimalité et la faisabilité des solutions sur l'ensemble des instances étudiées, avec un *gap* nul jusqu'à l'instance *Grande* et inférieur à 3 % sur l'instance *Très grande*. Toutefois, son temps de calcul reste plus élevé que celui des métaheuristiques, ce qui peut limiter son usage en contexte opérationnel temps quasi-réel.

À l'inverse, les métaheuristiques offrent un compromis intéressant entre qualité de la solution et temps de calcul. Le recuit simulé se distingue particulièrement par sa capacité à éviter les optima locaux, fournissant des solutions de meilleure qualité que la recherche locale, au prix d'un temps de calcul légèrement plus élevé. Sur le critère de l'équité, SA se rapproche du comportement de la méthode exacte, ce qui en fait une alternative pertinente lorsqu'on cherche à éviter le coût computationnel du MILP. Ces résultats confirment l'intérêt complémentaire des approches exactes et métaheuristiques pour des problèmes combinatoires complexes comme le NRP, en particulier dans les contextes hospitaliers où la rapidité de génération d'un planning de qualité est essentielle.

Conclusion générale

Ce mémoire s'est intéressé à la résolution du Nurse Rostering Problem (NRP), un problème d'optimisation combinatoire NP-difficile crucial dans le contexte hospitalier. À travers une démarche associant modélisation mathématique et expérimentation algorithmique, nous avons proposé et comparé trois approches de résolution.

Modélisation MILP. La première contribution consiste en la formulation d'un modèle de programmation linéaire en nombres entiers mixtes (MILP) rigoureuse du NRP, intégrant des contraintes dures (faisabilité) et molles (préférences, équité).

Approches de résolution. Deux métaheuristiques de trajectoire ont été développées et implémentées : la recherche locale (LS), qui améliore progressivement une solution par exploration de voisinage, et le recuit simulé (SA), qui échappe aux optima locaux via un mécanisme d'acceptation probabiliste. Ces approches ont été comparées à la résolution exacte (GLPK/MILP).

Étude expérimentale. Évaluation sur 4 instances NSPLib (10 à 100 infirmiers) selon plusieurs critères : qualité de solution (Z), violations dures (Vh), temps de calcul et équité ((Wi)).

Les résultats confirment l'efficacité des approches proposées :

- **MILP (GLPK)** : Optimalité garantie mais coûteux en temps (>90s sur très grande instance).
- **Recherche locale (LS)** : Très rapide (<7s) mais accumule violations dures et mauvaise équité sur grandes instances.
- **Recuit simulé (SA)** : **Meilleur compromis** — qualité supérieure à LS, temps acceptable (<20s), bon respect équité et contraintes.

Le recuit simulé émerge comme la méthode la plus adaptée pour le NRP en environnement réel, offrant un équilibre optimal entre qualité des solutions et faisabilité computationnelle.

Limites identifiées : (1) instances limitées à 100 infirmiers ; (2) multi-compétences et absences imprévues non intégrées ; (3) paramétrage empirique des métaheuristiques ; (4) matériel modeste (4 Go RAM).

Perspectives futures :

- Apprentissage par renforcement pour planification dynamique
- Optimisation multi-objectif (fronts de Pareto)

— Extension aux systèmes multi-services et multi-départements
— Gestion en temps réel des absences imprévues
— Validation en environnement hospitalier réel

Ce travail illustre le potentiel des méthodes d'optimisation pour résoudre des enjeux concrets de santé publique. Les résultats obtenus constituent une base solide pour la communauté scientifique et les praticiens hospitaliers, en démontrant que le recuit simulé offre une solution pratique, efficace et scalable au problème de planification des infirmiers.

Bibliographie

[1] U. Aickelin et K. A. Dowsland. "An indirect genetic algorithm for a nurse scheduling problem". In : *Computers & Operations Research* 31.5 (2004), p. 761-778.

[2] I. Berrada, J. A. Ferland et P. Michelon. "Scheduling of hospital staff using simulated annealing". In : *European Journal of Operational Research* 94.1 (1996), p. 22-34.

[3] Burak Bilgin, Ender Ozcan et Esra E. Korkmaz. "An experimental study on hyper-heuristics for nurse rostering". In : (2012).

[4] Peter Brucker, Rong Qu et Edmund Burke. "Personnel scheduling : Models and complexity". In : *European Journal of Operational Research* 210.3 (2011), p. 467-473.

[5] Michael J. Brusco et Lawrence W. Jacobs. "A simulated annealing approach to the nurse scheduling problem". In : *European Journal of Operational Research* (1993).

[6] E. K. Burke et al. *The nurse rostering problem : A critical appraisal*. 2001.

[7] E. K. Burke et al. "The state of the art of nurse rostering". In : *Journal of Scheduling* 7 (2004), p. 441-499.

[8] Edmund K. Burke, Patrick De Causmaecker et Greet Vanden Berghe. *NSPLib : A Nurse Scheduling Problem Library*. `http://www.cs.nott.ac.uk/~nza/NSPLib/`. 2008.

[9] Edmund K. Burke, Patrick De Causmaecker et Greet Vanden Berghe. "The state of the art of nurse rostering". In : *Journal of Scheduling* 7.6 (2004), p. 441-499.

[10] Edmund K. Burke, Graham Kendall et Eric Soubeiga. "A tabu-search hyperheuristic for timetabling and rostering". In : *Proceedings of the International Conference on the Practice and Theory of Automated Timetabling (PATAT)*. 2003.

[11] Edmund K. Burke, Jingpeng Li et Rong Qu. “A hybrid simulated annealing method for the nurse rostering problem”. In : *IEEE International Conference on Systems, Man and Cybernetics*. T. 3. 2004, p. 2603-2608.

[12] B. Cheang et al. “Nurse Rostering Problems : A Bibliographic Survey”. In : *European Journal of Operational Research* 151.3 (2003), p. 447-460.

[13] *Cours C2SI*. Support de cours au format PDF. 2024.

[14] P. De Causmaecker et G. Vanden Berghe. “A categorisation of nurse rostering problems”. In : *Journal of Scheduling* 14.1 (2011), p. 3-16.

[15] A. T. Ernst et al. “Staff scheduling and rostering : A review of applications, methods and models”. In : *European Journal of Operational Research* 153.1 (2004), p. 3-27.

[16] GeeksforGeeks. *Branch and Bound Algorithm : Applications, Advantages and Disadvantages*. `https://www.geeksforgeeks.org/dsa/applications-advantages-and-disadvantages-of-branch-and-bound-algorithm/`. Consulté le 28 mars 2026. 2026.

[17] S. Haspeslagh et al. “The second international nurse rostering competition (INRC-II)”. In : *PATAT Proceedings*. 2014.

[18] J. Menana, S. Demassey et N. Jussien. *Modélisation et optimisation des préférences en planification de personnel*. Rapp. tech. École des Mines de Nantes, LINA CNRS UMR 6241, 2010.

[19] T. Messelis et al. *Investigating the complexity of nurse rostering problems*. 2009.

[20] A. Mystakidis et al. “Optimizing nurse rostering : a case study using integer programming to enhance operational efficiency and care quality”. In : *Healthcare* 12.24 (2024), p. 2545.

[21] Ragheb Rahmaniani et al. “The Benders decomposition algorithm : A literature review”. In : *European Journal of Operational Research* 259.3 (2017), p. 801-817.

[22] H. G. Santos et al. “Integer Programming Techniques for the Nurse Rostering Problem”. In : *Annals of Operations Research* (2014).

[23] Mario Vanhoucke et Broos Maenhout. “NSPLib – A Nurse Scheduling Problem Library : A tool to evaluate (meta-)heuristic procedures”. In : *ORAHS 2005 Proceedings*. 2005.

[24] A. Wren. “Scheduling, timetabling and rostering — a special relationship ?” In : *Lecture Notes in Computer Science* (1996).

Annexe A : Codes sources des implémentations

Cette annexe regroupe les principaux codes sources utilisés pour les expérimentations présentées dans ce mémoire. Deux implémentations du modèle MILP du *Nurse Rostering Problem* y sont fournies :

— une implémentation en **Python** à l'aide de la bibliothèque PuLP, résolue par le solveur CBC ;

— une implémentation en **MathProg** (langage de modélisation de GLPK), résolue par le solveur glpsol.

Les deux codes correspondent à l'instance Petite (10 infirmiers, 14 jours, 3 shifts) décrite au Chapitre 4. Ils sont entièrement commentés afin de faciliter la compréhension de chaque section du modèle.

A.1 : Implémentation Python (PuLP + CBC)

Le code suivant implémente le modèle MILP complet en Python à l'aide de la bibliothèque PuLP. Les contraintes dures et molles, ainsi que la fonction objectif, sont définies conformément à la formulation présentée au Chapitre 2.

```
1
2  from pulp import *
3  import random
4  import statistics
5  import time
6
7  # ============================================================
8  # DEFINITION DES 4 INSTANCES
9  # ============================================================
10
11 INSTANCES = {
12     "Petite": {"N": 10, "DAYS": 14, "minS": 2, "maxS": 4, "timeLimit": 60},
13     "Moyenne": {"N": 27, "DAYS": 26, "minS": 2, "maxS": 6, "timeLimit": 120},
14     "Grande": {"N": 50, "DAYS": 28, "minS": 3, "maxS": 8, "timeLimit": 180},
15     "Tres grande": {"N": 100, "DAYS": 28, "minS": 5, "maxS": 15, "timeLimit": 300},
16 }
17
18 SHIFTS = ["M", "A", "N"]
19
20 # ============================================================
21 # FONCTION DE RESOLUTION MILP
22 # ============================================================
23
24 def solve_nrp(name, params):
```

```
    N = params["N"]
    DAYS = params["DAYS"]
    minS = params["minS"]
    maxS = params["maxS"]
    timeLimit = params["timeLimit"]

    I = range(N)
    T = range(DAYS)
    S = range(len(SHIFTS))

    h = {0: 8, 1: 8, 2: 10}
    minStaff = {(s, t): minS for s in S for t in T}
    maxStaff = {(s, t): maxS for s in S for t in T}

    max_week_hours = 48
    max_month_hours = 160

    random.seed(42)
    pref = {}
    for i in I:
        for s in S:
            for t in T:
                pref[(i, s, t)] = random.randint(0, 1)

    penalty = {(i, s, t): 1 - pref[(i, s, t)] for i in I for s in S for t in T}

    alpha = 0.4
    beta = 0.3
    gamma = 0.3
    BIG_M = 1000000

    model = LpProblem(f"NRP_{name}", LpMinimize)

    x = LpVariable.dicts("x", ((i, s, t) for i in I for s in S for t in T), cat="Binary")
    y = LpVariable.dicts("y", ((i, t) for i in I for t in T), cat="Binary")
    deficitStaff = LpVariable.dicts("deficit", ((s, t) for s in S for t in T), lowBound=0)
    excesStaff = LpVariable.dicts("exces", ((s, t) for s in S for t in T), lowBound=0)
    u = LpVariable.dicts("u", (i for i in I), lowBound=0)
    eps = LpVariable.dicts("eps", ((i, s, t) for i in I for s in S for t in T), lowBound=0)

    W_i = {i: lpSum(h[s] * x[i, s, t] for s in S for t in T) for i in I}
    W_avg = (1.0 / N) * lpSum(W_i[i] for i in I)

    obj_pref = lpSum(penalty[(i, s, t)] * x[i, s, t] for i in I for s in S for t in T)
    obj_soft = lpSum(eps[i, s, t] for i in I for s in S for t in T)
    obj_balance = lpSum(u[i] for i in I)
    obj_violations = lpSum(deficitStaff[s, t] + excesStaff[s, t] for s in S for t in T)

    model += (alpha * obj_pref + beta * obj_soft + gamma * obj_balance + BIG_M * obj_violations)

    # CONTRAINTES H1-H6
    for i in I:
        for t in T:
            model += y[i, t] == 1 - lpSum(x[i, s, t] for s in S)

    for s in S:
        for t in T:
            model += lpSum(x[i, s, t] for i in I) + deficitStaff[s, t] >= minStaff[(s, t)]
            model += lpSum(x[i, s, t] for i in I) - excesStaff[s, t] <= maxStaff[(s, t)]

    for i in I:
        for w_start in range(0, DAYS, 7):
            week_days = range(w_start, min(w_start + 7, DAYS))
            model += lpSum(h[s] * x[i, s, t] for s in S for t in week_days) <= max_week_hours

    for i in I:
        model += lpSum(h[s] * x[i, s, t] for s in S for t in T) <= max_month_hours

    for i in I:
        for t in T:
            model += lpSum(x[i, s, t] for s in S) <= 1

    for i in I:
        for t in range(DAYS - 1):
            model += x[i, 2, t] + x[i, 0, t + 1] <= 1

    # CONTRAINTES MOLLES ET EQUITE
    for i in I:
        for s in S:
            for t in T:
```

```
                model += eps[i, s, t] >= pref[(i, s, t)] - x[i, s, t]

    for i in I:
        model += W_i[i] - W_avg <= u[i]
        model += W_avg - W_i[i] <= u[i]

    start = time.time()
    solver = PULP_CBC_CMD(msg=False, timeLimit=timeLimit)
    model.solve(solver)
    end = time.time()

    status = LpStatus[model.status]
    Z = value(model.objective)
    Vh = sum(int(round(value(deficitStaff[s, t]) or 0)) + int(round(value(excesStaff[s, t]) or 0)) for s in S for t
      in T)
    loads = [value(W_i[i]) or 0 for i in I]
    sigma_W = statistics.pstdev(loads) if loads else 0

    return {"name": name, "N": N, "DAYS": DAYS, "status": status, "Z": Z, "Vh": Vh, "time": end - start, "sigma":
     sigma_W}

if __name__ == "__main__":
    results = []
    for name, params in INSTANCES.items():
        res = solve_nrp(name, params)
        results.append(res)

    for r in results:
        print(f"{r['name']}: Z={r['Z']:.2f}, Vh={r['Vh']}, Temps={r['time']:.2f}s, sigma={r['sigma']:.2f}")
```

Listing 1 – Code MILP en Python/PuLP pour le NRP

A.2 : Implémentation MathProg (GLPK)

À titre de validation croisée, le modèle MILP a également été implémenté en MathProg, le langage de modélisation du solveur open-source GLPK.

```

# Nurse Rostering Problem (NRP) - Modele MathProg / GLPK
# Instance : Petite (10 infirmieres, 14 jours, 3 shifts)

set I;  # ensemble des infirmieres
set J;  # ensemble des jours
set S;  # ensemble des shifts

param minStaff {S, J}, integer, >= 0;
param maxStaff {S, J}, integer, >= 0;
param h {S}, >= 0;
param maxHWeek, >= 0;
param maxHMonth, >= 0;
param alpha, >= 0;
param beta, >= 0;
param gamma, >= 0;

var x {I, J, S}, binary;
var y {I, J}, binary;
var totalHours {I} >= 0;
var deficitStaff {J, S} >= 0;
var excesStaff {J, S} >= 0;
var u {I} >= 0;

s.t. H1 {i in I, j in J}:
    y[i, j] = 1 - sum {s in S} x[i, j, s];

s.t. H2_min {j in J, s in S}:
    sum {i in I} x[i, j, s] + deficitStaff[j, s] >= minStaff[s, j];

s.t. H2_max {j in J, s in S}:
    sum {i in I} x[i, j, s] - excesStaff[j, s] <= maxStaff[s, j];

s.t. totalH {i in I}:
    totalHours[i] = sum {j in J, s in S} h[s] * x[i, j, s];

```

```
s.t. H4 {i in I}:
    totalHours[i] <= maxHMonth;

s.t. H5 {i in I, j in J}:
    sum {s in S} x[i, j, s] <= 1;

s.t. H6 {i in I, j in J : j < card(J)}:
    x[i, j, "N"] + x[i, j + 1, "M"] <= 1;

minimize Z :
    alpha * sum {i in I, j in J, s in S} x[i, j, s]
    + beta * sum {j in J, s in S} (deficitStaff[j, s] + excesStaff[j, s])
    + gamma * sum {i in I} u[i];

data;
set I := 1 2 3 4 5 6 7 8 9 10;
set J := 1 2 3 4 5 6 7 8 9 10 11 12 13 14;
set S := M A N;

param h := M 8 A 8 N 10;
param maxHWeek := 48;
param maxHMonth := 160;
param alpha := 0.4;
param beta := 0.3;
param gamma := 0.3;

param minStaff : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 :=
    M 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    A 2 2 2 2 2 2 2 2 2 2 2 2 2 2
    N 2 2 2 2 2 2 2 2 2 2 2 2 2 2;

param maxStaff : 1 2 3 4 5 6 7 8 9 10 11 12 13 14 :=
    M 4 4 4 4 4 4 4 4 4 4 4 4 4 4
    A 4 4 4 4 4 4 4 4 4 4 4 4 4 4
    N 3 3 3 3 3 3 3 3 3 3 3 3 3 3;

end;
```

Listing 2 – Implémentation MILP en MathProg/GLPK

—

Annexe B : Environnement technique détaillé

Cette annexe fournit les détails complets de l'environnement de développement utilisé pour les implémentations et expérimentations du Nurse Rostering Problem. Elle est destinée à faciliter la reproductibilité des travaux et l'installation de l'environnement sur d'autres machines.

B.1 : Bibliothèques Python utilisées

L'implémentation des méthodes de résolution a mobilisé plusieurs bibliothèques Python spécialisées. Cette section en détaille l'utilisation, les versions et les raisons du choix.

B.1.1 NumPy

Version recommandée : $\geq 1.21.0$

Installation :

```
pip install numpy>=1.21.0
```

Utilisation dans le projet :

NumPy est utilisée pour la gestion efficace des tableaux multidimensionnels et la réalisation des calculs numériques nécessaires à l'implémentation des métaheuristiques, notamment la recherche locale (LS) et le recuit simulé (SA).

Spécifiquement :

— Représentation des matrices de plannings infirmiers (dimension $N \times |T| \times |S|$)

— Calculs vectorisés pour l'évaluation rapide de la fonction objectif

— Génération de nombres aléatoires (sélection d'infirmiers, jours, shifts)

— Opérations statistiques (moyenne, écart-type pour l'équité)

B.1.2 Pandas

Version recommandée : $\geq$ 1.3.0

Installation :

```
pip install pandas>=1.3.0
```

Utilisation dans le projet :

Pandas est employée pour la lecture et l'écriture des fichiers CSV contenant les instances NSPLib, ainsi que pour la manipulation et l'analyse des plannings générés.

Spécifiquement :

— Chargement des matrices de couverture minimale/maximale depuis les fichiers NSPLib

— Chargement des matrices de préférences des infirmiers

— Exécution et post-traitement des résultats dans des DataFrames

— Export des résultats pour analyse ultérieure

B.1.3 Matplotlib

Version recommandée : $\geq$ 3.4.0

Installation :

```
pip install matplotlib>=3.4.0
```

Utilisation dans le projet :

Matplotlib est utilisée pour la visualisation des résultats expérimentaux et le tracé des courbes de convergence des algorithmes. Elle permet une analyse visuelle de la progression des métaheuristiques.

Spécifiquement :

— Génération des courbes de convergence : Z en fonction des itérations

— Comparaison graphique LS vs SA sur les 4 instances

— Graphiques de performance (temps vs qualité)

— Visualisation des distributions de charge de travail

B.1.4 PuLP

Version recommandée : $\geq$ 2.6

Installation :

```
pip install pulp>=2.6
```

Utilisation dans le projet :

PuLP est utilisée pour la modélisation MILP en Python, avec une interface vers le solveur CBC (Coin-or branch-and-cut solver). Elle permet de formuler le problème NRP sous forme de programme linéaire en nombres entiers et de le résoudre.

Spécifiquement :

— Définition des variables binaires $x_{i,s,t}$ et $y_{i,t}$

— Formulation de la fonction objectif

— Ajout de toutes les contraintes dures et molles

— Interface avec le solveur CBC pour la résolution

— Extraction des valeurs de solution et statistiques

Note : PuLP utilise par défaut le solveur CBC. Pour utiliser GLPK, une configuration supplémentaire est nécessaire (voir section B.2).

—

B.2 : Solveurs

B.2.1 GLPK (GNU Linear Programming Kit)

Version utilisée : GLPK 4.65 ou plus récente

Installation sous Linux/macOS :

```
# macOS avec Homebrew
brew install glpk

# Debian/Ubuntu
sudo apt-get install glpk-utils libglpk-dev

# Fedora/RedHat
sudo dnf install glpk glpk-devel
```

Installation sous Windows :

Télécharger depuis `https://www.gnu.org/software/glpk/`

Vérification de l'installation :

```
glpsol --version
```

Utilisation :

GLPK peut être utilisé de deux manières dans ce projet :

1. **Directement via ligne de commande** : Pour résoudre un fichier MathProg :

```
glpsol --math nrp.mod --output nrp_solution.txt
```

2. **Via PuLP** : Configuration avancée pour utiliser GLPK comme solveur backend :

```
from pulp import *

problem = LpProblem("NRP", LpMinimize)
# ... définir le modèle ...

# Utiliser GLPK
glpk_solver = GLPK(msg=0, timeLimit=3600)
problem.solve(glpk_solver)
```

B.2.2 CBC (Coin-or Branch-and-Cut)

Version : Incluse avec PuLP

Installation :

CBC est automatiquement installé avec PuLP. Aucune installation supplémentaire nécessaire.

Utilisation :

```
from pulp import PULP_CBC_CMD

solver = PULP_CBC_CMD(timeLimit=3600, msg=0)
problem.solve(solver)
```

—

B.3 : Instructions d'installation complète

B.3.1 Installation de l'environnement Python

Étape 1 : Vérifier Python

```
python --version  # Doit être >= 3.12
```

Étape 2 : Créer un environnement virtuel (recommandé)

```
python -m venv env_nrp
source env_nrp/bin/activate    # Linux/macOS
# ou
env_nrp\Scripts\activate        # Windows
```

Étape 3 : Installer les dépendances

```
pip install --upgrade pip
pip install numpy>=1.21.0 pandas>=1.3.0 matplotlib>=3.4.0 pulp>=2.6
```

Étape 4 : Installer GLPK (optionnel, pour comparaison)

```
# Linux/macOS
brew install glpk  # ou apt-get install...

# Vérifier l'installation
glpsol --version
```

B.3.2 Structure du projet recommandée

```
NRP_Project/
 nrp_milp_all.py          # Code Python MILP (PuLP + CBC)
 nrp.mod                   # Code MathProg (GLPK)
 instances/                # Fichiers NSPLib
    petite.nsp
    moyenne.nsp
    grande.nsp
    tres_grande.nsp
 results/                  # Résultats expérimentaux
    results_table.csv
    convergence_plots/
 README.md
```

B.3.3 Vérification de l'installation

Créer un fichier `test_installation.py` :

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pulp import *

print("NumPy version:", np.__version__)
print("Pandas version:", pd.__version__)
print("Matplotlib version:", plt.matplotlib.__version__)

# Test PuLP et CBC
```

```
problem = LpProblem("Test", LpMinimize)
x = LpVariable("x", 0, 10)
problem += x
problem.solve(PULP_CBC_CMD(msg=0))
print("PuLP/CBC: OK" if problem.status == 1 else "PuLP/CBC: ERREUR")
```

Exécuter :

```
python test_installation.py
```

—

B.4 : Configuration de l'environnement de développement

B.4.1 IDE recommandés

— **VS Code** : Gratuit, léger, extensible
— **PyCharm Community Edition** : Complet, gratuit
— **Jupyter Notebook** : Pour l'exploration et la visualisation

B.4.2 Extensions VS Code recommandées

— Python (Microsoft)
— Pylance
— Jupyter

B.4.3 Fichier requirements.txt

Pour faciliter l'installation sur d'autres machines, créer un fichier `requirements.txt` :

```
numpy>=1.21.0
pandas>=1.3.0
matplotlib>=3.4.0
pulp>=2.6
```

Installation via :

```
pip install -r requirements.txt
```

—

B.5 : Résolution de problèmes courants

B.5.1 Erreur : "ModuleNotFoundError : No module named 'pulp'"

Solution : Réinstaller PuLP dans l'environnement virtuel actif

```
pip install --force-reinstall pulp
```

B.5.2 Erreur : "glpsol : command not found"

Solution : GLPK n'est pas installé. Utiliser :

```
# macOS
brew install glpk

# Ubuntu/Debian
sudo apt-get install glpk-utils libglpk-dev
```

B.5.3 Erreur : "CBC solver not found"

Solution : CBC est inclus avec PuLP. Réinstaller :

```
pip install --force-reinstall pulp
```

B.5.4 Performance lente

Vérifications :

— Vérifier la version de NumPy (doit être compilée avec BLAS/LAPACK)
— Désactiver le mode debug si activé
— Augmenter les ressources CPU/mémoire allouées
—

B.6 : Reproduire les expériences

B.6.1 Avec Python (PuLP + CBC)

```
python nrp_milp_all.py
```

Cela résout automatiquement les 4 instances et affiche un tableau comparatif.

B.6.2 Avec GLPK (MathProg)

```
glpsol --math nrp.mod
```

B.6.3 Générer les graphiques

Un script Python supplémentaire (non inclus dans le mémoire) peut générer les courbes de convergence. Adaptation facile à partir du code principal.

—

B.7 : Notes sur la compatibilité et les versions

Composant	Version testée	Version min. requise
Python	3.12.10	3.8
NumPy	1.24.3	1.21.0
Pandas	2.0.2	1.3.0
Matplotlib	3.7.1	3.4.0
PuLP	2.7.0	2.6
GLPK	4.65	4.60

TABLE 3 – Versions des composants utilisés et compatibilité.

Les versions listées ont été testées sur Ubuntu 24.04 et macOS 13.x. D'autres versions compatibles peuvent également fonctionner, mais ces versions minimales garantissent une reproductibilité fiable.

—

B.8 : Support et dépannage

Pour tout problème :

1. Vérifier d'abord la section B.5 (Résolution de problèmes courants)
2. Consulter la documentation officielle :
   — PuLP : `https://coin-or.github.io/pulp/`
   — GLPK : `https://www.gnu.org/software/glpk/`
   — NumPy : `https://numpy.org/doc/`
3. Tester l'installation avec le script `test_installation.py`

—

Annexe C : Implémentation des métaheuristiques

Cette annexe fournit les codes Python complets pour l'implémentation des deux métaheuristiques de trajectoire : la recherche locale (Local Search - LS) et le recuit simulé (Simulated Annealing - SA). Ces codes sont entièrement commentés en français pour faciliter la compréhension et la reproduction des expériences.

C.1 : Code Python complet - Recherche Locale (LS) et Recuit Simulé (SA)

```
1
2  import random
3  import math
4  import statistics
5  import time
6  import matplotlib.pyplot as plt
7
8  # ============================================================
9  # DEFINITION DES INSTANCES (NSPLib)
10 # ============================================================
11
12 INSTANCES = {
13     "Petite": {"N": 10, "DAYS": 14, "minS": 2, "maxS": 4},
14     "Moyenne": {"N": 27, "DAYS": 26, "minS": 2, "maxS": 6},
15     "Grande": {"N": 50, "DAYS": 28, "minS": 3, "maxS": 8},
16     "Tres grande": {"N": 100, "DAYS": 28, "minS": 5, "maxS": 15},
17 }
18
19 SHIFTS = ["M", "A", "N", "R"]
20
21 # ============================================================
22 # PARAMETRES DU MODELE
23 # ============================================================
24
25 ALPHA = 1000000
26 BETA = 1
27 GAMMA = 100
28
29 MAX_WEEK_HOURS = 48
30 MAX_MONTH_HOURS = 160
31
32 SHIFT_HOURS = {0: 8, 1: 8, 2: 10, 3: 0}
33
34 # ============================================================
35 # GENERATION SOLUTION INITIALE
36 # ============================================================
37
38 def generate_initial_solution(N, DAYS):
39     """Genere une solution initiale aleatoire."""
```

```
    solution = []
    for i in range(N):
        nurse = [random.randint(0, 3) for t in range(DAYS)]
        solution.append(nurse)
    return solution

# ============================================================
# GENERATION DES PREFERENCES
# ============================================================

def generate_preferences(N, DAYS):
    """Genere les preferences aleatoires des infirmiers."""
    pref = {}
    for i in range(N):
        for s in range(3):
            for t in range(DAYS):
                pref[(i, s, t)] = random.randint(0, 1)
    return pref

# ============================================================
# FONCTION OBJECTIF
# ============================================================

def evaluate(solution, params, pref):
    """Evalue la qualite d'une solution."""
    N = params["N"]
    DAYS = params["DAYS"]
    minS = params["minS"]
    maxS = params["maxS"]

    hard = 0
    soft = 0
    loads = []

    # Calcul des charges de travail
    for i in range(N):
        load = sum(SHIFT_HOURS[solution[i][t]] for t in range(DAYS))
        loads.append(load)

    sigma = statistics.pstdev(loads)

    # Contrainte H2 : Couverture
    for t in range(DAYS):
        for s in range(3):
            count = sum(1 for i in range(N) if solution[i][t] == s)
            if count < minS:
                hard += (minS - count)
            if count > maxS:
                hard += (count - maxS)

    # Contrainte H6 : Interdiction Nuit -> Matin
    for i in range(N):
        for t in range(DAYS - 1):
            if solution[i][t] == 2 and solution[i][t+1] == 0:
                hard += 1

    # Contrainte H4 : Limite mensuelle
    for i in range(N):
        if loads[i] > MAX_MONTH_HOURS:
            hard += 1

    # Contrainte molle : Preferences
    for i in range(N):
        for t in range(DAYS):
            if solution[i][t] != 3:
                if pref[(i, solution[i][t], t)] == 0:
                    soft += 1

    Z = ALPHA * hard + BETA * soft + GAMMA * sigma
    return Z, hard, soft, sigma

# ============================================================
# GENERATION DU VOISINAGE
# ============================================================

def generate_neighbor(solution):
    """Genere une solution voisine par modification aleatoire."""
    new_sol = [row[:] for row in solution]
    i = random.randint(0, len(solution)-1)
    t = random.randint(0, len(solution[0])-1)
```

```
    new_sol[i][t] = random.randint(0, 3)
    return new_sol

# ============================================================
# RECHERCHE LOCALE (LS)
# ============================================================

def local_search(name, params):
    """Algorithme de recherche locale - 5000 iterations."""
    N = params["N"]
    DAYS = params["DAYS"]
    pref = generate_preferences(N, DAYS)
    current = generate_initial_solution(N, DAYS)
    best = current
    best_Z, _, _, _ = evaluate(best, params, pref)
    history = [best_Z]
    start = time.time()

    for k in range(5000):
        neighbor = generate_neighbor(current)
        Z_current, _, _, _ = evaluate(current, params, pref)
        Z_neighbor, _, _, _ = evaluate(neighbor, params, pref)

        if Z_neighbor < Z_current:
            current = neighbor
        if Z_neighbor < best_Z:
            best = neighbor
            best_Z = Z_neighbor
        history.append(best_Z)

    end = time.time()
    Z, Vh, Vs, sigma = evaluate(best, params, pref)

    return {"method": "LS", "Z": Z, "Vh": Vh, "Vs": Vs, "sigma": sigma,
            "time": end - start, "history": history}

# ============================================================
# RECUIT SIMULE (SA)
# ============================================================

def simulated_annealing(name, params):
    """Algorithme de recuit simule - Temperature initiale 1000."""
    N = params["N"]
    DAYS = params["DAYS"]
    pref = generate_preferences(N, DAYS)
    current = generate_initial_solution(N, DAYS)
    best = current
    best_Z, _, _, _ = evaluate(best, params, pref)
    history = [best_Z]
    T = 1000
    Tmin = 0.1
    alpha = 0.95
    start = time.time()

    while T > Tmin:
        for k in range(1000):
            neighbor = generate_neighbor(current)
            Z_current, _, _, _ = evaluate(current, params, pref)
            Z_neighbor, _, _, _ = evaluate(neighbor, params, pref)
            delta = Z_neighbor - Z_current

            if delta < 0:
                current = neighbor
            else:
                if random.random() < math.exp(-delta / T):
                    current = neighbor

            if Z_neighbor < best_Z:
                best = neighbor
                best_Z = Z_neighbor
            history.append(best_Z)

        T *= alpha

    end = time.time()
    Z, Vh, Vs, sigma = evaluate(best, params, pref)

    return {"method": "SA", "Z": Z, "Vh": Vh, "Vs": Vs, "sigma": sigma,
            "time": end - start, "history": history}

```

```python
# ============================================================
# EXECUTION ET RESULTATS
# ============================================================

if __name__ == "__main__":
    results = []

    for name, params in INSTANCES.items():
        print("\n" + "="*50)
        print(f"INSTANCE : {name}")
        print("="*50)

        res_ls = local_search(name, params)
        res_sa = simulated_annealing(name, params)

        print(f"\nLS: Z={res_ls['Z']:.2f}, Vh={res_ls['Vh']}, sigma={res_ls['sigma']:.2f}")
        print(f"SA: Z={res_sa['Z']:.2f}, Vh={res_sa['Vh']}, sigma={res_sa['sigma']:.2f}")

        # Courbes de convergence
        plt.figure(figsize=(10, 5))
        plt.plot(res_ls["history"], label="LS", color='green', linewidth=2)
        plt.plot(res_sa["history"], label="SA", color='orange', linewidth=2)
        plt.title(f"Convergence LS vs SA - Instance {name}")
        plt.xlabel("Iterations")
        plt.ylabel("Fonction objectif Z")
        plt.grid(True, alpha=0.3)
        plt.legend(fontsize=12)
        plt.savefig(f"convergence_LS_SA_{name}.png", dpi=300, bbox_inches='tight')
        plt.show()

        results.append((name, res_ls, res_sa))

    # Tableau final
    print("\n" + "="*70)
    print("TABLEAU RECAPITULATIF FINAL")
    print("="*70)
    for name, ls, sa in results:
        print(f"\n{name}:")
        print(f"  LS - Z={ls['Z']:.2f}, Vh={ls['Vh']}, Temps={ls['time']:.2f}s")
        print(f"  SA - Z={sa['Z']:.2f}, Vh={sa['Vh']}, Temps={sa['time']:.2f}s")
```

Listing 3 – Métaheuristiques LS et SA - Code complet commenté

C.2 : Exécution du code

Prérequis : `pip install matplotlib`

Lancer : `python` $ls_anrp.py$

Le script exécutera automatiquement LS et SA sur les 4 instances, générera les courbes de convergence (PNG) et affichera le tableau récapitulatif.