École Polytechnique de l'Université de Tours
64, Avenue Jean Portalis
37200 TOURS, FRANCE
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Département Informatique
5e année
2014 - 2015

Rapport de Projet de Fin d'Études

Master Surgical Scheduling Problem

Encadrant
Yannick KERGOSIEN
yannick.kergosien@univ-tours.fr

Étudiant
Katy DUFEUTRELLE
katy.dufeutrelle@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2014 - 2015

Version du 8 mai 2015

Table des matières

1. Introduction

Les activités chirurgicales en milieu hospitalier sont très difficiles à gérer notamment à cause des ressources limitées. Les gestionnaires de blocs opératoires doivent décider des allocations des ressources et du temps opératoire aux différentes spécialités chirurgicales tout en respectant les nombreuses contraintes hospitalières existantes. Les listes d'attentes des hôpitaux ne cessant de s'allonger, le Master Surgical Scheduling Problem (MSSP) vise à s'occuper de la gestion des blocs opératoires afin de maximiser le nombre d'interventions possibles sur une période donnée et de réduire ces temps d'attente.

Ce document détaille le travail que j'ai effectué tout au long de l'année 2014-2015 dans le cadre de mon projet de fin d'études. Ayant déjà eu l'occasion de travailler sur ce sujet l'année précédente au travers de deux mini-projets de Gestion de Personnes et de Flux et de Science de la décision, je souhaitais pouvoir approfondir ce problème et ainsi continuer mon travail précédent.

Ce projet a pour objectif de travailler sur des méthodes de résolutions au niveau stratégique de ce problème et de développer un simulateur représentant une solution mise en place périodiquement sur un horizon de plusieurs mois.

Ce rapport contient dans un premier temps une présentation du problème accompagné d'un rappel du cahier des spécifications du système, puis dans un second temps le travail effectué sur les méthodes de résolution et le simulateur, et enfin dans un dernier temps un bilan du projet ainsi que les outils et technologies utilisés dans le développement et dans la gestion.

2. Présentation du projet

2.1 Le problème du MSSP

2.1.1 Contexte

Le MSSP est un problème NP-difficile portant sur l'ordonnancement des blocs opératoires et visant à réduire les listes d'attentes. En effet, le temps entre le diagnostic du médecin concernant une intervention chirurgicale sur un patient, et le moment où ce patient est opéré, est très long. L'objectif de ce problème est donc d'aider les gestionnaires de blocs opératoires en déterminant une planification des interventions chirurgicales dans les différentes salles opératoires en fonction du nombre d'arrivées de patients pour ce type d'opération et qui respectent les contraintes existantes du monde hospitalier. Cela permettra d'optimiser les ressources disponibles afin de maximiser le nombre d'interventions programmables possible. Cet ordonnancement devra être périodique car il sera répliqué dans le temps sur toute une saison.

Ce problème est récurrent dans les hôpitaux québécois, dont les affectations des spécialités n'est pas faîte en fonction des besoins mais en fonction de l'ancienneté du chirurgien. Des contraintes ministérielles ont été mises en place pour gérer ce problème, mais ne sont que rarement respectés. Ajoutant à ça une mauvaise gestion des ressources, les salles opératoires sont sous-utilisées entraînant une perte économique pour l'hôpital.

Tout d'abord, nous allons définir quelques mots du vocabulaire hospitalier utilisé dans ce rapport :
- Spécialité : Discipline médicale qui regroupe des interventions chirurgicales de même domaine. Exemple : pédiatrie, neurologie, cardiologie, etc...
- Intervention type : Intervention fictive créée à partir d'un regroupement d'interventions chirurgicales d'une même spécialité possédant les mêmes caractéristiques.
- Ressources communes : Ensemble des ressources communes non consommables à toutes les spécialités (matériel, infirmières, anesthésistes, ...).
- Ressources propres à une spécialité : Ensemble des ressources propres non consommables à chacune des spécialités (matériel, chirurgien,...).

2.1.2 Modélisation

Données

Données générales
- b_j : Nombre de lits disponibles le jour j
- J : Ensemble de jours dans la période

Salles
- h_{oj} : Durée d'ouverture de la salle o le jour j
- $a_{min_{sj}}$: Nombre minimum de salles à affecter pour la spécialité s le jour j
- O : Ensemble des salles opératoires

Ressources
— QRC_{ju} : Nombre de ressources communes de type u disponibles le jour j
— QRS^s_{ju} : Nombre de ressources propres de type u à la spécialité s disponibles le jour j
— BRC_{ku} : Besoin en ressources communes de type u pour l'intervention de type k
— BRS^s_{ku} : Besoin en ressources propres de type u à la spécialité s pour l'intervention de type k
— RC : Ensemble des ressources communes
— RS^s : Ensemble des ressources propres à la spécialité s
Spécialité
— $Fmax_s$: Nombre maximum de jours entre 2 programmations de la spécialité s
— S : Ensemble des spécialités
Interventions
— τ_k : Taux d'arrivée pour l'intervention k
— p_k : Durée de l'intervention k
— l_k : Durée d'hospitalisation de l'intervention k
— μ_k : Pondération de l'intervention k
— K : Ensemble des interventions
— K_s : Ensemble des interventions de la spécialité s

Variables

- $\mathbf{x_{ojk}} \in \{\mathbf{0}, \lfloor \frac{\mathbf{h_{oj}}}{\mathbf{p_k}} \rfloor\}$: entier égal au nombre d'interventions de type k programmées dans la salle o le jour j,$\forall o \in O; \forall j \in J; \forall k \in K$.
- $\mathbf{y_{ojk}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si au moins une intervention de type k est programmée dans la salle o le jour j, $\forall o \in O; \forall j \in J; \forall k \in K$.
- $\mathbf{z_{ojs}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si la spécialité s est programmée dans la salle o le jour j,$\forall o \in O; \forall j \in J; \forall s \in S$.

Contraintes

Respect de la durée d'ouverture de la salle opératoire

$$\sum_{k \in K} (p_k \times x_{ojk}) \leq h_{oj};\ \forall j \in J,\ \forall o \in O \tag{2.1}$$

Respect du minimum de salles à affecter par jour pour une spécialité

$$\sum_{o \in O} z_{ojs} \geq a_{min_{sj}};\ \forall s \in S,\ \forall j \in J \tag{2.2}$$

Respect du nombre maximum de journées sans programmation d'une spécialité

$$\sum_{i=j+1}^{Fmax_s+j} \sum_{o \in O} z_{o(i \bmod J)s} \geq 1;\ \forall s \in S,\ \forall j \in J \tag{2.3}$$

Affectation au maximum d'une seule spécialité par salle

$$\sum_{s \in S} z_{ojs} = 1;\ \forall j \in J,\ \forall o \in O \tag{2.4}$$

Relation entre les variables

$$si\ x_{ojk} \geq 1\ alors\ z_{ojs} = 1;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{2.5}$$

$$si\ z_{ojs} = 1\ alors\ x_{ojk} = 0;\ \forall k \notin K_s,\ \forall j \in J,\ \forall o \in O \tag{2.6}$$

$$si\ y_{ojk} = 1\ alors\ x_{ojk} \geq 1;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{2.7}$$

$$si\ y_{ojk} = 0\ alors\ x_{ojk} = 0;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{2.8}$$

$$si\ x_{ojk} = 0\ alors\ y_{ojk} = 0;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{2.9}$$

Respect de la disponibilité des ressources communes

$$\sum_{o \in O}(\max_{k \in K}(y_{ojk} \times BRC_{ku})) \leq QRC_{ju};\ \forall u \in RC,\ \forall j \in J \tag{2.10}$$

Respect de la disponibilité des ressources propres à une spécialité

$$\sum_{o \in O}(\max_{k \in K_s}(y_{ojk} \times BRS^s_{ku})) \leq QRS^s_{ju};\ \forall u \in RS^s,\ \forall j \in J,\ \forall s \in S \tag{2.11}$$

Respect du taux d'arrivée de patients (suppression de liste d'attente)

$$\sum_{o \in O}\sum_{j \in J} x_{ojk} \geq \tau_k;\ \forall k \in K \tag{2.12}$$

Respect du nombre de lits disponibles par jour

$$\sum_{o \in 0}\sum_{g \in J}\sum_{k \in K}(x_{ogt} \times (\lceil \frac{l_k + (g - j)}{|J|} \rceil - \beta)) \leq b_j \begin{cases} \beta = 1\ si\ g > j \\ \beta = 0\ sinon \end{cases} ;\forall j \in J \tag{2.13}$$

Fonction objectif

$$\max \sum_{k \in K} \mu_k \sum_{o \in 0}\sum_{j \in J} x_{ojk} \tag{2.14}$$

2.1.3 Existant

Ce sujet de PFE a déjà été traité l'an dernier par Antoine GIRET, qui avait mis en place un générateur de données pour le problème ainsi qu'une méthode de résolution avec le solveur Cplex. Les données générées et les résultats de sa méthode exacte me permettront de comparer mes méthodes de résolution aux solutions optimales.

De même, l'an dernier au cours de deux mini-projets j'ai travaillé sur une méthode de résolution basée sur une méta-heuristique. Ce travail sera repris cette année afin de l'améliorer et la rendre plus performante.

2.2 Présentation des acteurs

2.2.1 La maîtrise d'ouvrage

Dans ce projet, la maîtrise d'ouvrage est représentée par l'équipe Ordonnancement et Conduite de l'Ecole Polytechnique de l'Université de Tours ainsi que par M. Patrick Soriano, membre du CIRRELT (Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport) de Montréal.

2.2.2 La maîtrise d'oeuvre

Ici, la maîtrise d'oeuvre est représentée par moi-même, DUFEUTRELLE Katy, étudiante en 5ème année, encadré par M.Yannick Kergosien maître de conférence au Département Informatique de l'École Polytechnique de l'Université de Tours.

2.3 Objectifs du projet

La méthode de résolution faîte en PFE l'an dernier avec le solveur Cplex est performante mais seulement sur de petites instances. Or, dans de réelles conditions les données d'un hôpital québécois sont bien plus importantes, et leur résolution devient très gourmande en temps et en mémoire. C'est pour cette raison, qu'une méthode approchée doit être mise en place.

Dans ce projet de fin d'études, nous avons choisi d'essayer deux autres méthodes de résolution : une méthode exacte utilisant la programmation par contraintes avec le solveur Choco et une méthode approchée utilisant une méta-heuristique avec un algorithme génétique. Comme vu dans la partie existant du projet, la méta-heuristique a déjà été réalisée. On commencera donc par analyser ses performances afin de détecter ses faiblesses pour trouver les améliorations dont elle a besoin pour s'approcher au plus des résultats optimaux.

Enfin dans un dernier temps, un simulateur de périodes sera développé. La solution donnée par l'une des méthodes de résolutions est représentée sous forme d'un planning contenant les affectations des spécialités et des interventions types sur une période. Une fois que l'on a obtenu cette solution, il faut la tester dans des conditions réelles. Pour cela, le simulateur dupliquera ce planning sur plusieurs périodes à la suite et devra gérer les différents évènements pouvant survenir (opération plus longue, patient devant être replanifié ultérieurement, manque de place, etc.) suivant un flux d'arrivées de patients différents à chaque période. Ce simulateur nous permettra d'obtenir des indicateurs quant aux ressources utilisées, au nombre de patients ne pouvant pas être opéré le jour prévu, au temps d'attente des patients, etc.

3. Spécifications du système

Ce chapitre rappelle certains points tirés du cahier des spécifications du système écrit au début du projet et validé par l'encadrant Y.Kergosien début janvier.

3.1 Présentation générale

3.1.1 Environnement du projet

Concernant l'existant, un projet de fin d'études a été réalisé l'année précédente par Antoine Giret sur la même problématique. Ainsi, les jeux de données de tests ont été repris de la génération faîte l'an dernier, ainsi que les deux modèles utilisés dans la résolution du problème. Ils ont simplement été adapté pour la programmation par contraintes. Pour finir, les résultats obtenus par cet étudiant (grâce à une résolution avec le solveur Cplex) permettront d'effectuer une comparaison avec ceux obtenus selon les deux méthodes de résolution prévues.

Les deux méthodes de résolution du problème reçoivent des jeux de données en paramètres donnant lieu à des solutions d'ordonnancement maximisant le nombre d'interventions à effectuer. Ces solutions seront ensuite reçues par le simulateur pour les tester et indiquer à l'utilisateur les résultats de cette solution (le nombre d'interventions reportées etc..).

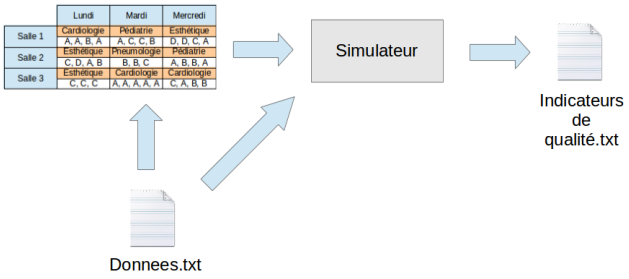
3.1.2 Caractéristiques des utilisateurs

Les programmes de résolution ainsi que le simulateur devront être utilisable par des utilisateurs non informaticiens. Leur utilisation devra donc être simple (indication simplement du chemin vers le fichier de données par exemple), ainsi que la lecture des résultats donnés par les résolutions ou le simulateur.

3.1.3 Fonctionnalités et structure générale du système

Pour les résolutions, après avoir indiqué le chemin vers le fichier de données, le programme renverra une solution, c'est à dire un planning permettant la planification des interventions dans les salles opératoires disponibles sous forme de fichier texte.

Pour le simulateur, en entrée du programme on donnera un fichier texte contenant le planning permettant d'éviter les temps d'attente pour les interventions et en sortie sera renvoyé dans un fichier texte des indicateurs montrant ce que donne réellement le planning au fil des périodes.



3.2 Description des fonctionnalités

3.2.1 Méthodes de résolution

Lecture et parsage des données

Identification : Les programmes de résolution doivent pouvoir lire une instance, qui doit se trouver sous un certain format d'écriture afin d'en récupérer les données.
Description : La fonction prendra en entrée une instance sous forme de fichier texte et instanciera les données du programme.

Trouver un macro planning

Identification : Cette fonctionnalité doit permettre de renvoyer une solution respectant toutes les contraintes imposées à partir des données se trouvant dans le fichier d'instance.

Afficher un résultat

Identification : Le programme doit pouvoir renvoyer la solution trouvée pour être utilisée ou pour vérifier que les contraintes soient correctement respectées.
Description : Cette fonction prendra en entrée le "planning" trouvé et renverra en sortie un affichage console ou un fichier contenant la solution en détails (quelle intervention quel jour dans quelle salle).

3.2.2 Simulateur

Simuler une solution

Identification : Cette fonctionnalité va permettre de simuler une solution en la répliquant de nombreuses fois.
Description : La fonction prend en entrée des données du type "arrivée d'un patient pour l'intervention A de la spécialité 2" ainsi qu'une solution à tester. A partir de ça, elle va ensuite remplir les listes d'évènements et de ressources, puis traiter chaque évènement selon des règles définies.

Générer des résultats

Identification : Le simulateur doit pouvoir générer des résultats au fur et à mesure de la simulation afin de déterminer la qualité de la solution testée.
Description : La fonction renverra en sortie des indicateurs comme le taux de lits occupés, le nombre d'interventions reportées etc...

3.3 Conditions de fonctionnement

3.3.1 Performances

Les deux programmes de résolution ainsi que le simulateur devront s'exécuter dans des temps raisonnables, tout en essayant de trouver une solution optimale. Pour tout lancement de scripts de tests, un ordinateur est attribué à ce projet de fin d'études afin que les calculs puissent se faire avec les meilleurs performances.

3.3.2 Capacités

Le risque des problèmes d'ordonnancement est la taille des jeux de données qui influence sur le nombre de variables créées, le nombre de combinaisons différentes à essayer et donc peut causer un manque de mémoire. La capacité des programmes à traiter des données importantes pourra être limitée.

3.3.3 Contrôlabilité

Le simulateur au cours de ses tests pourra renseigner l'utilisateur de l'avancée des indicateurs en temps réel grâce à un affichage.

4. Méthode de résolution par la programmation par contraintes

4.1 Présentation de la programmation par contraintes

La programmation par contraintes (PPC) est apparu dans les années 1980 et permet de résoudre des problèmes combinatoires. La partie modélisation est représentée à l'aide de problèmes de satisfaction de contraintes. La partie résolution utilise les contraintes du problème pour réduire la taille de l'espace des solutions à parcourir, c'est ce qu'on appelle la propagation de contraintes. La PPC est utilisée dans les problèmes de décision, de planifications, d'ordonnancement, etc.

Pour résoudre un problème avec la PPC, on utilise des variables de décision et des contraintes pour le modéliser. Les contraintes sont des relations entre une ou plusieurs variables permettant de limiter les valeurs que peuvent prendre ces variables liées par ces contraintes. Les algorithmes de recherche de solution s'appuient sur la propagation de contraintes pour réduire le nombre de solutions possibles à explorer, ainsi que sur une recherche systématique parmi les différentes affectations possibles de chacune des variables. De tels algorithmes garantissent de trouver une solution, quand elle existe, et permettent de prouver qu'il n'existe pas de solution à un problème s'ils n'ont pas trouvé de solution à la fin de la recherche exhaustive.

4.2 Le solveur Choco

Choco est une librairie Java dédiée à la programmation par contraintes et aux problèmes de satisfactions de contraintes. Choco est un logiciel libre principalement développer par des membres de l'Ecole des Mines de Nantes et financé par Bouygues SA et Amadeus SA.

FIGURE 4.1 – Logo de Choco Solver

J'ai tout d'abord commencé à développer sous Choco 2, puis j'ai tenté de transposer mon code sous la version Choco 3 mais les opérateurs ayant changés, il n'était plus possible d'exprimer certaines contraintes. Je suis donc revenu à la version 2, plus adaptée à mes modèles.

4.3 Écriture des deux modèles pour la PPC

Pour cette méthode de résolution, deux modèles vont être implémentés. Ces deux modèles ont été fournis par l'encadrant. L'un est un modèle normal, qui comprend les contraintes du problème tandis que l'autre met en place des journées types : ce sont toutes les combinaisons possibles d'interventions types d'une spécialité dans une salle, un jour donné.

Ces modèles étant linéaires, il a fallut dans un premier temps les réécrire pour qu'ils correspondent à la programmation par contraintes. En effet, les contraintes et les relations entre les variables peuvent être modélisées de manière non linéaires (par exemple $Si...Alors$ ou encore $AllDifferent$ qui impose que les variables aient des valeurs distincts). Ces deux modèles se trouvent en annexes.

4.4 Implémentation des modèles

4.4.1 Structure du solveur

Tout d'abord une classe est créé pour parser le fichier d'entrée. Ensuite, on crée un "Model" qui contiendra les variables et les contraintes. Pour déclarer une variable et l'ajouter au Model, il faut définir le domaine dans lequel elle peut prendre des valeurs. Ensuite, on ajoute les contraintes contenant les variables précédemment déclaré, et on termine en donnant ce Model à lire au solveur. Pour lancer la résolution, on donne la fonction objectif au solveur, dans notre cas on demande la solution où la variable donnant le nombre d'interventions types programmées est maximale.

4.4.2 Stratégies d'optimisation

Afin d'optimiser la recherche de la solution et réduire le nombre de noeuds dans l'arbre de recherche et donc le temps d'exécution, il est possible de mettre en place des stratégies à l'aide de branchements. Cela permet de décider dans la création de l'arbre de recherche, l'ordre des variables à instancier ou quelle valeur du domaine doit être choisie en premier par exemple.

Dans notre cas, la première variable à instancier est le nombre d'interventions types : le reste s'instanciera tout seul grâce à la propagation de contraintes. La stratégie mise en place est de commencer par les valeurs maximales du domaine de la variable et de continuer dans un ordre décroissant jusqu'à la plus petite valeur du domaine.

4.5 Tests et résultats

Un premier test a été effectué sur les 160 instances fournies par le PFE d'Antoine Giret. Un temps de résolution maximum a été fixé à 30 minutes. Sur le modèle 1, sur seulement 6 instances le solveur a pu trouver une solution (non optimale) dans les temps. Le modèle 2 (journées types), seul 24 instances ont pu être traitées dont 8 avec une solution non optimale dans le temps imparti, pour les autres on se retrouve avec une explosion mémoire lors de la création de l'arbre, dû aux trop grand nombre de variables.

Un deuxième test a donc été fait avec 50 petits jeux de données. Pour le modèle 1, le solveur a trouvé 15 solutions optimales (temps moyen de 54s), 24 solutions non optimales et 3 solutions non réalisables en 4 mS. Le modèle 2 a permis au solveur de trouvé 8 solutions optimales (temps moyen de 4 minutes et 42s), 30 solutions non optimales et les 3 solutions non réalisables en moins d'1s.

4.6 Conclusion sur cette méthode

Cette méthode de résolution n'a pas été concluante pour ce projet. Le sujet étant trop complexe, le nombre de variables est trop grand pour être supporté par le solveur. Le modèle 2 avec les journées types n'est pratiquement pas exploitable, le solveur ne pouvant pas créer l'arbre dû au nombre trop grand de branches possibles. Des solutions optimales sont trouvées avec des tout petits jeux de données mais les temps d'exécutions sont trop longs.

5. Méthode de résolution par une méta-heuristique

5.1 Présentation de l'existant

Dans cette partie, nous avons décidé de repartir sur le travail que j'avais précédemment effectué sur le sujet l'an passé au travers de deux mini-projets. Il avait été mis en place une heuristique permettant de créer une solution respectant certaines contraintes (spécialités, durée opératoire, ressources propres et communes) et une méta-heuristique avec un algorithme génétique.

5.1.1 Heuristique

Pour affecter les spécialités, l'heuristique commence par les contraintes ministérielles concernant le nombre minimum de salles à avoir par spécialité suivant le jour de la période. Ensuite on vérifie, et ajoute aux besoins, si la contrainte du nombre maximum entre deux affectations d'une spécialité est respectée sur la période. Le reste des affectations se fait suivant les taux d'arrivée les plus importants entre les spécialités.

Pour les interventions types, on prend une salle au hasard, on établit une liste des interventions types qui peuvent être programmées, c'est à dire qui respectent les contraintes sur la durée d'ouverture de la salle et sur les ressources propres et communes, on choisit l'une d'elles au hasard et on l'affecte à la salle. Cette opération est répétée jusqu'à ce qu'aucune intervention ne puisse être ajoutée dans aucune salle.

La fonction objectif est la maximisation du nombre d'interventions. Le nombre de contraintes violées est pénalisé (multiplié par 1000) auquel on retire le nombre d'interventions programmées, le but étant donc de minimiser la valeur de cette fonction.

La solution est représentée par un tableau en deux dimensions (les jours en largeur, les salles en profondeur) et à chaque case est affecté une spécialité ainsi qu'une liste d'interventions possibles.

	Jour 1	Jour 2	...	Jour 11	Jour 12
Salle 1	Spécialité : 2 Interventions : 1, 1, 2	Spécialité : 3 Interventions : 3, 3, 2		Spécialité : 1 Interventions : 2 ,3	Spécialité : 3 Interventions : 1, 2
Salle 2	Spécialité : 1 Interventions : 2	Spécialité : 2 Interventions : 2, 1		Spécialité : 2 Interventions :	Spécialité : 3 Interventions : 2
Salle 3	Spécialité : 1 Interventions : 1, 3, 2	Spécialité : 2 Interventions : 3		Spécialité : 3 Interventions : 1, 2	Spécialité : 1 Interventions : 3, 2, 2

FIGURE 5.1 – Représentation de la solution

5.1.2 Méta-heuristique

Pour trouver une solution optimale, une méta-heuristique se basant sur un algorithme génétique a été créée. Elle se compose de la façon suivante après l'initialisation d'une population par l'heuristique :

- Sélection par tournois de deux individus dans la population parents.
- Croisement de ces deux individus afin de donner deux enfants : choix du croisement à un ou deux points, et de manière vertical(largeur) ou horizontal(profondeur).
- Changement de population parent : choix entre un échange des populations parents et enfants, une sélection de la moitié des meilleurs individus de chaque population (parents et enfants) ou une sélection des meilleurs individus parmi les populations parents et enfants.
- Opérateur de voisinage appliqué sur chaque individu de la population. Composé de deux parties, il permet de modifier et d'améliorer les solutions : la première partie effectue des échanges entre les cases du tableau de solutions et "répare" les cases au besoin, c'est à dire s'assure que la liste d'intervention respecte les contraintes de durée opératoire et des ressources (propres et communes), seulement cet échange n'est gardé que si la solution est améliorée. La deuxième partie prend les cases du tableau une par une et cherche si une modification améliore la solution et laquelle, parmi l'insertion, la suppression ou le changement d'une intervention.

5.2 Analyses sur la méta-heuristique existante

Cette méthode de résolution est très longue et les solutions données sont relativement éloignées des solutions optimales. Afin d'améliorer cette méta-heuristique il convient de mettre en place des analyses pour pouvoir étudier le comportement d'évolution de la population.

Pour ces tests, nous avons choisi de garder le croisement à deux points en largeur (vertical) qui présentait de meilleurs résultats que les autres, mais nous avons gardé les trois méthodes de choix de population car les différences de résultats ne sont pas assez grandes. Par des résultats aberrants, les analyses ont aussi permis de trouver deux erreurs dans le code impactant la valeur de la solution et donc le choix du croisement. Après corrections, aucune méthode de croisement ne se distinguait des autres, cependant nous avons tout de même choisi de garder celui à deux points en largeur.

Voici les analyses qui ont été appliquées :

- Observation sur le comportement des solutions : indication à chaque itération de la meilleure solution trouvée, de la meilleure solution de la population, de la pire solution, de la moyenne des valeurs des solutions et du pourcentage de solutions non réalisables dans la population.
- Variation du nombre d'échanges dans la première partie de l'opérateur de voisinage. - Observation du pourcentage d'amélioration de chaque partie de l'opérateur de voisinage à chaque itération.
- Observation du comportement avec une grande population.
- Observation du comportement avec une petite population mais un grand nombre d'itérations.
- Application uniquement de l'opérateur de voisinage (sans croisement) sur une seule itération pour étudier son amélioration sur les solutions indépendantes des unes des autres.

Les résultats ont principalement montré que les individus convergeaient très vite vers une solution. Les individus se ressemblent et se retrouvent "bloquer" dans une configuration. Les deux opérateurs de voisinage (les échanges et l'amélioration de cases) ne permettent pas assez la diversification ce qui empêche les individus de se remettre en question afin de se rapprocher de la solution optimale.

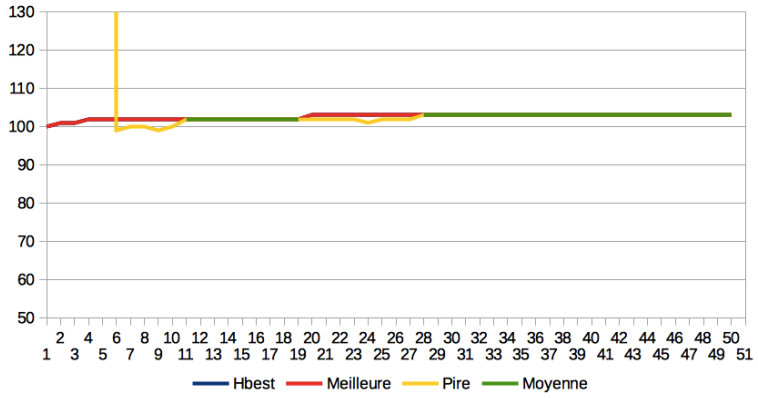


FIGURE 5.2 – La population convergence très rapidement

On peut aussi conclure que l'opérateur d'échanges a une utilité limitée. En effet, il peut permettre aux solutions non réalisables de devenir réalisables, mais dans le cas d'une solution déjà réalisable (qui l'était de base ou qui l'est devenue grâce aux échanges), il n'y aura aucune amélioration de sa fonction objective car les échanges ne changeront pas le nombre d'interventions programmées. De plus, si un échange donne une solution réalisable (de fonction objective égale) alors la configuration de l'opérateur implémenté fait que cet échange ne sera pas gardé, ainsi au fil des itérations les individus ne sont pas changés. Cet opérateur ne permet donc pas de diversifier les individus et donc entraîne la rapide convergence des solutions.

5.3 Choix et implémentation des modifications à apporter

Grâce à ses analyses, nous avons pu prendre des décisions pour améliorer la méthode de résolution. En effet, nous avons choisi de ne modifier que les opérateurs de voisinage. Nous les conservons en les améliorant pour combler leur faiblesse et en ajoutons deux autres permettant de diversifier de manière plus conséquentes la population : changement de la spécialité d'une case et croisement d'interventions dans deux cases d'une même spécialité.

La création et la modification de ces opérateurs a soulevé une importante réflexion. De nombreuses questions se sont posées comme le choix des cases à modifier ou encore combien de fois un opérateur va-t-il effectuer des changements. Le but étant de donner un "sens" à ces opérateurs, des critères doivent être pris en compte : est-il plus intéressant de changer la spécialité d'une case où le taux d'occupation de la salle est bas ou bien d'une case dont le nombre d'interventions est inférieur à un certain paramètre ? Il sera plus d'intéressants d'établir ainsi des critères de sélections intelligemment plutôt que de choisir aléatoirement. Les différents opérateurs doivent travailler ensemble, sans détruire le travail du précédent, tout en diversifiant et améliorant la population.

Pour les trois premiers opérateurs, un paramètre booléen sera mis en place pour permettre le choix entre garder les solutions qui se sont strictement améliorées ou celles qui ont gardé la même valeur. De plus, tous ces opérateurs de voisinage auront leurs opérations faîtes un certain nombre maximum de fois, à définir en paramètre de la méta-heuristique pour chacun d'eux.

5.3.1 Échanges de deux cases

Comme dit précédemment, nous avons conservé la première partie de l'opérateur de voisinage, nous choisissons aléatoirement deux cases pour les échanger. Quelques changements ont cependant été apportés : ces deux cases peuvent s'échanger si elles respectent les deux contraintes sur les spécialités (minimum de salles à affecter à une spécialité et nombre maximum de jours séparant deux spécialités) et qu'elles n'appartiennent pas à la même journée. De plus, les listes d'interventions sont vidées et remplies de la même manière que le fait l'heuristique (respect des contraintes de durées et ressources). Cet opérateur échange donc les spécialités des deux cases et leur réattribue des interventions.

5.3.2 Modification de la spécialité d'une case

Ce nouvel opérateur va se concentrer sur la modification de la spécialité des cases. La case sur laquelle on effectue cette opération est choisie parmi une liste de A cases modifiables. Pour qu'une case soit dans cette liste, sa spécialité doit pouvoir toujours respecter les contraintes sur les spécialités et sur le taux d'arrivée des patients. Dans le cas où il y ait un nombre de cases sélectionnables plus grand que A, on retirera de la liste celles dont le taux d'occupation des salles est le plus élevé jusqu'à atteindre le nombre A. Une fois la case sélectionnée, on testera toutes les autres spécialités, et bien sûr la liste d'interventions sera vidée et remplie de la même façon qu'avec l'heuristique.

5.3.3 Croisement des interventions de deux listes

Ce troisième opérateur va choisir aléatoirement une spécialité et deux de ses cases de jours différents. Sur la première case, on va chercher à échanger chaque intervention avec une intervention de la deuxième case choisi aléatoirement parmi celles dont l'échange respectera les contraintes de durée et de ressources. L'échange est conservé si la solution est améliorée. Un prétraitement visant à limiter le temps d'exécution permettra de définir rapidement si l'intervention peut ou non être échangée, en respectant les contraintes voulues.

5.3.4 Amélioration d'une case

On ne change pas cet ancien opérateur non plus qui consiste à trouver une amélioration parmi l'insertion, la suppression et la modification d'une intervention dans une case, seulement on introduit un nouveau paramètre pour qu'à la place de ne faire qu'une seule amélioration par case, on puisse en faire N fois.

5.4 Analyses des modifications

A la suite de l'implémentation, des analyses ont permis d'étudier le comportement des solutions. Deux études ont été faites, une première sur le comportement de la solution suivant les paramètres donnés et la deuxième plus approfondie sur ce qui pouvait empêcher à une solution d'atteindre la solution optimale.

La première étude a permis de donner une idée sur les valeurs des paramètres à explorer ainsi que l'utilité des opérateurs de voisinages, ce qui sera utile pour déterminer les tests à lancer sur un groupe d'instances.

La deuxième étude a permis de revoir la structure de l'opérateur d'améliorations qui avait une action trop ciblée ainsi que de mettre un place un opérateur de mutation permettant d'inverser les salles opératoires d'une même journée.

L'opérateur d'améliorations, au lieu de tenter d'améliorer un certain nombre de fois chaque case, va tenter d'améliorer une seule case choisie aléatoirement un certain nombre de fois.

Les heures d'ouvertures étant différentes pour chaque salle, échanger deux affectations de salles permet à des spécialités qui en ont besoin de profiter de plus de temps opératoires et ainsi remettre en cause l'affectation des spécialités faîte par l'heuristique. Cette mutation se fera sur 10% de la population après les opérateurs de sélection et de croisement.

5.5 Tests sur les paramètres

Une fois que les différentes modifications ont pu être apportées sur la méta-heuristique, il reste à déterminer le meilleur jeu de paramètres permettant de s'approcher au maximum de la solution optimale tout en gardant un temps de résolution convenable.

Rappel des paramètres à tester :

- L'opérateur de croisement : vertical en 1 point, vertical en 2 points, horizontal en 1 point ou horizontale en 2 points.
- L'opérateur de sélection : sélection de la population enfant, sélection à 50% des meilleurs de la population parent et 50% des meilleurs de la population enfant ou sélection des meilleurs de la population parent et enfant mélangés.
- Nombre d'individus dans la population.
- Nombre d'itérations de l'algorithme génétique.
- Valeurs des paramètres des 4 opérateurs de voisinages.
- Booléen permettant d'autoriser ou non la garde d'une solution égale à la meilleure solution trouvée actuelle dans les opérateurs de voisinage.

De nombreux tests ont été faits dont les résultats se trouvent en annexes :

Sur 103 instances réalisables, tests sur les 4 paramètres possibles de l'opérateur de croisement avec une population de 100 individus, 100 itérations, un opérateur de sélection fixé à 50% des meilleurs de la population parent et 50% des meilleurs de la population enfant, les opérateurs de voisinages à 50 et le booléen à vrai. Mêmes tests avec les opérateurs de voisinage à 200.

Le meilleur résultat trouvé est avec le croisement vertical en 1 point.

Sur 103 instances réalisables, tests sur les 3 paramètres possibles de l'opérateur de sélection avec une population de 80 individus, 70 itérations, un opérateur de croisement vertical en 1 point, les opérateurs de voisinages à 50 et le booléen à vrai. Mêmes tests avec les opérateurs de voisinage à 200.

Le meilleur résultat trouvé est avec l'opérateur de sélection fixé à 50% des meilleurs de la population parent et 50% des meilleurs de la population enfant.

Sur 50 instances réalisables, tests sur 9 combinaisons de paramètres pour le nombre d'individus dans la population et le nombre d'itérations de la méta-heuristique parmi les valeurs 100, 150 et 200. L'opérateur de croisement est vertical en 1 point, l'opérateur de sélection fixé à 50% des meilleurs de la population parent et 50% des meilleurs de la population enfant, les opérateurs de voisinages à 50 et le booléen à vrai.

Les deux meilleurs résultats trouvés au niveau qualité et temps d'exécution sont une population de 100 et 150 individus pour un nombre d'itérations à 100.

Sur 50 instances réalisables, tests pour connaître l'efficacité et l'utilité des opérateurs de voisinages. Valeur du paramètre à 50 d'abord pour un paramètre seul, puis deux par deux et enfin trois par trois, sur les deux meilleurs combinaisons du nombre d'individus et d'itérations trouvés précédemment. Les autres paramètres sont fixés comme précédemment.

Les résultats ont montré que l'opérateur de croisement de listes est peu efficace voir pénalisant. L'opérateur d'échanges de deux cases est très utile et peu coûteux en temps. On remarque aussi qu'il y a peu d'impact sur une population à 150 individus plutôt que 100.

Sur 50 instances réalisables, test sur les deux valeurs possibles du paramètre booléen permettant la diversité chez les individus de la population. Le nombre d'individus est fixé à 100, le nombre d'itérations à 100, les opérateurs de voisinages à 50 excepté pour le croisement de listes qui est à 0 et les autres paramètres sont fixés comme précédemment.

Les résultats ne montrent pas l'utilité de ce paramètre, il y a peu de différence, ne pas autoriser la garde d'une solution de même valeur que celle trouvée dans les opérateurs de voisinage est très légèrement meilleure que de l'autoriser.

Maintenant que les valeurs des paramètres ont été choisi, nous sommes revenus sur le nombre d'individus et d'itérations ainsi que sur les valeurs des paramètres des opérateurs de voisinages, afin de vérifier l'impact sur la qualité et le temps d'exécution si leur valeur était réduite.

Le temps d'exécution est pratiquement divisé par deux mais la qualité en est un peu affectée.

De plus, deux autres tests ont été faits pour regarder la convergence et la stabilité de la méta-heuristique. Les graphes ci-dessous montrent une bonne stabilité de la méthode et une bonne convergence.

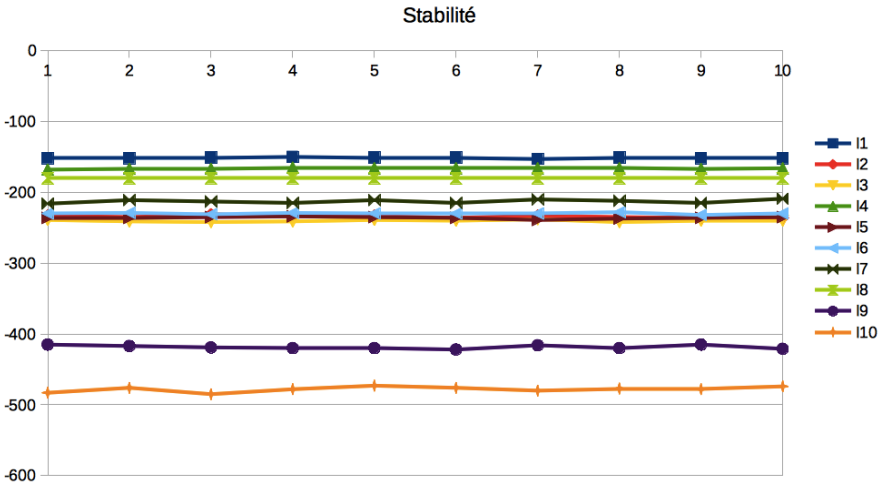


FIGURE 5.3 – Résultats sur la stabilité

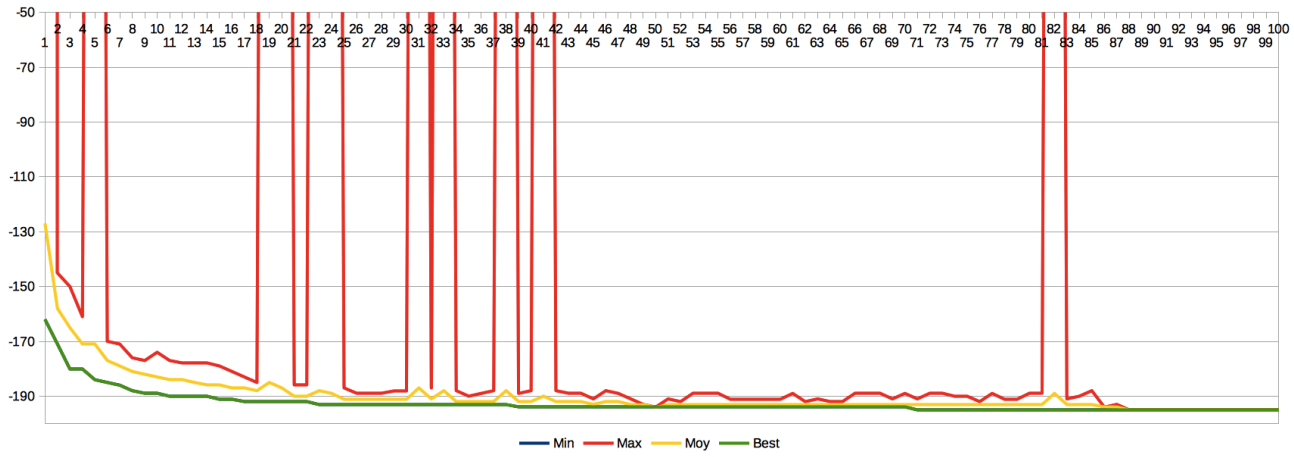


FIGURE 5.4 – Résultats sur la convergence

6. Mise en place d'un simulateur de planning

Dans cette dernière partie, un outil de simulation est mis en place afin de vérifier la viabilité de la solution trouvée par une méthode de résolution, dans des conditions "réelles". Ce simulateur tournera sur plusieurs périodes et renverra des indicateurs permettant de connaître l'état des ressources et des patients au fil du temps. Il pourra permettre au gestionnaire de blocs opératoires de situer les problèmes important, comme un nombre de ressources trop limité, des salles sous-utilisées, etc.

6.1 Réflexion sur la construction

La première phase de cette partie commence par la réflexion sur la construction du simulateur. Celle-ci s'est faîte avec l'encadrant du projet M.Kergosien qui m'a expliqué les bases de fonctionnement d'un simulateur.

En premier lieu, nous avons commencé à déterminer les entrées du simulateur : des données fixes et des données aléatoires. Les données fixes sont le nombre de lits, le nombre de salles, la durée d'ouverture des salles, le nombre de ressources communes et propres disponibles chaque jour, les durées opératoires moyennes des interventions, le nombre de jours d'une période, le planning des affectations des spécialités sur la période et le nombre de jours total sur lequel la simulation doit se faire. Les données aléatoires sont la durée de l'opération d'un patient, la durée du séjour du patient et le taux d'arrivée des patients pour une spécialité. Ces données aléatoires, à défaut de recevoir des données réelles d'archives des CHUQ, seront déterminées à l'avance selon des lois mathématiques et en lien avec nos données fixes, elles ne seront donc pas dynamiques. Avec celles-ci, il pourra par exemple être généré une dizaine de fichier d'instances de données aléatoires par instance de données fixes afin de pouvoir observer différents comportements lors la simulation. Ils devront être constitué d'un numéro unique pour chaque patient/intervention, du nom et type de l'intervention, de la durée opératoire et de la durée d'hospitalisation qui suit.

En second lieu, nous avons organisé le simulateur en deux "modules" cycliques : l'Operating Rooms Scheduling et la partie de simulation. Ainsi, après la résolution du MSSP, le planning d'affectations retourné sera donné au premier module (ORS) qui déterminera un planning "réel" d'une période se basant sur celui du MSSP et sur le taux d'arrivée des patients. Ce nouveau planning sera envoyé au deuxième module qui effectuera donc une simulation grâce aux données aléatoires et ressortira des indicateurs. A chaque fin de période, le cycle recommence.

Le module d'ORS ne sera pas détaillé dans ce rapport car il s'insère dans le projet de l'option de 5ème année Logistique et Optimisation dont je fais partie. Son fonctionnement et sa méthode de planification sont à retrouver dans le rapport de projet concerné. En voici tout de même les grandes lignes pour la compréhension, qui s'organise en deux temps, le premier au début d'une période i où

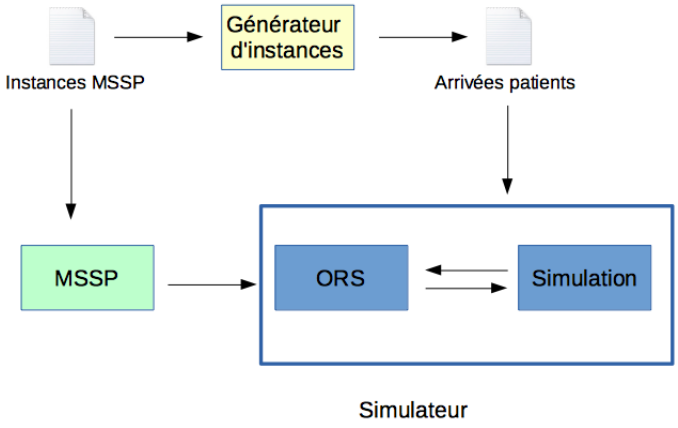


FIGURE 6.1 – Schéma explicatif du fonctionnement du simulateur

l'on planifie la période i+1 (ORS1) et le second à la fin de période i où l'on revoit le planning construit pour la période i+1 (ORS2).

Au début d'une période i, la phase ORS1 intervient pour planifier, à l'aide de la liste des patients prévus et du planning donné par la résolution, la période i+1 en prenant en priorité des patients qui ont été déplacés suite à la période i-1 appelés Di. Si par manque de temps ou de ressources, certains patients ne peuvent être planifiés sur cette période i+1, ils seront reportés à la période i+2 et seront appelés Di2.

Ensuite, on simule la période i dans le module de simulation, qui quand il se termine peut renvoyer des patients « reliquats » , ce sont les patients qui sont arrivés dans l'hôpital mais qui finalement n'ont pas pu être opéré le jour prévu. Ils sont toujours dans l'hôpital, prennent un lit et doivent être opérés le plus tôt possible lors de la période suivante.

C'est à ça que sert la deuxième phase, ORS2. Elle reprend la planification de la phase ORS1 en la réorganisant de manière à ce que les reliquats Ri soient en priorité le plus tôt possible (tout en gardant aussi la priorité des Di). Ce remaniement risque de devoir reporter certains patients par manque de temps et/ou de ressources, ils seront ajoutés aux Di2.

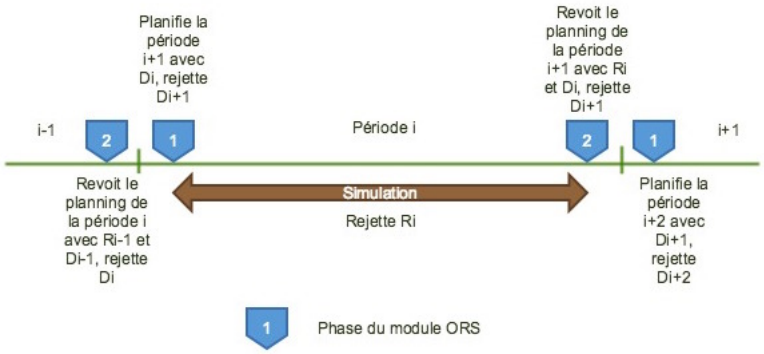


FIGURE 6.2 – Schéma explicatif du module ORS

Le module de simulation aura donc en entrée le planning concret des affectations fourni par le module d'ORS et en sortie, les reliquats à destination de ce même module ainsi qu'un historique

qui permettra d'établir des indicateurs (consommation de lits, de ressources, occupation des salles, nombre d'interventions reportées, temps supplémentaire de séjour pour les patients dont on a reporté l'opération, etc...). Il se compose d'entités, d'événements et de règles à suivre que l'échéancier fera tourner.

Entités :

- Patient
  - Numéro id
  - Date d'arrivée
  - Date de départ
  - Durée opératoire
  - Durée d'hospitalisation
  - États : arrivé, attente, convalescence ou
- Ressources disponibles
  - Lits
  - Ressources propres
  - Ressources communes
- Salle
  - Numéro id
  - Temps ouverture
  - Liste de patients

6.2 Générateur d'instances d'arrivées de patients

Un générateur d'instances a du être mis en place, à défaut de pouvoir obtenir des données des archives des hôpitaux du Québec. L'instance généré doit comprendre le numéro du patient, l'intervention type dont il a besoin, la spécialité associée et la durée opératoire et d'hospitalisation que cette opération nécessitera.

6.2.1 Paramètres

Afin de générer une instance correspondante aux données qu'utilise la méthode de résolution, on reprend le même fichier dans lequel doit être récupéré :

- Le nombre de spécialités
- Le nombre d'interventions types par spécialités
- Le taux d'arrivées moyen pour chaque intervention type.
- La durée opératoire moyenne pour chaque intervention type
- La durée d'hospitalisation moyenne pour chaque intervention type

De plus, il faut choisir le nombre de périodes à générer.

6.2.2 Lois mathématiques

Pour la génération de l'instance sur une période, on utilisera des lois mathématiques adéquates. Pour générer le nombre de patients par interventions types, la loi Normale sera utilisée et pour la durée opératoire ce sera la loi Log Normale qui sera bornée entre la moitié et 1,5 fois la durée moyenne. Pour la durée d'hospitalisation, un pourcentage de chances d'ajouter un jour supplémentaire à celui moyen est déterminé. Il est basé sur le ratio nombre de jours moyen divisé par 24.

6.3 Règles de fonctionnement du simulateur

— Les événements sont regroupés dans une liste triée par ordre croissant de leur attribut date.
— Les salles opératoires ouvrent toutes à 6h.
— A la fin de chaque journée, on essaie de replanifier les patients qui sont dans l'hôpital et non pour l'instant pas pu être opérés. La replanification se fait à 23h59.
— Un patient sort de l'hôpital à 5h avant les arrivées journalières à 5h30.
— Quand une salle a terminé de traiter tous ces patients elle regarde les interventions qui n'ont pas commencé dans l'ordre décroissant de leur durée opératoire moyenne et cherche à en effectuer une.
— Quand un patient arrive, s'il n'y a plus de lits disponibles, on passe en négatif, cela entraînera une surconsommation de lits.

6.4 Modélisation du simulateur

6.4.1 Enchaînement des périodes

On commence par lancer la phase ORS1 du module ORS afin que la première période soit planifié. Puis, on lance le cycle qui se fait pour chaque période : la phase ORS1, la simulation, la phase ORS2. A la fin, il ne reste qu'à simuler la dernière période et récupérer les reliquats et les déplacés restants.

6.4.2 Historique et affichages

A chaque période est conservé de nombreux indicateurs : le nombre de lits disponibles selon le temps, le taux d'utilisation des salles, si il y a eu un dépassement d'horaires dans une salle, le nombre de ressources propres et communes disponibles selon le temps, le nombre de patients déplacés et le nombre de patients reliquats. Ces indicateurs vont être affichés dans un fichier CSV permettant de les manipuler ou faire des tests, et serviront aussi à être affiché dans une interface graphique.

De plus, tout au longe de la simulation, des informations sont écrites dans un fichier de log pour garder une trace de ce qui s'est déroulé : quel patient a été opéré à quelle heure, qui a du être reporté, quel patient est sortie quand, etc.

6.5 Module ORS

6.5.1 Première phase : ORS1

La première phase du module a été développé de plusieurs manières : trois algorithmes peuvent être utilisés. Le premier est un algorithme glouton qui affecte les interventions dans la première salle venue tant que les contraintes d'ouverture de la salle et des ressources propres et communes sont respectées. Le deuxième est une méta-heuristique mise en place par moi-même et Alain Krok lors d'un projet de l'option Logistique et Optimisation, elle ne sera pas décrite ici. Le dernier algorithme est une affectation simple qui ne s'utilise qu'avec des données d'arrivées de patients correspondants exactement aux données moyennes, ceci ayant pour but de vérifier le fonctionnement du simulateur.

6.5.2 Deuxième phase : ORS2

La deuxième phase commence par essayer de planifier en priorité les reliquats dans le planning de la période suivante afin de les faire attendre le moins possible.

On commence par ajouter les reliquats appartenant à une même spécialité au plus tôt dans la première journée où apparaît cette spécialité de manière équilibrée au sein des différentes salles affectées à cette spécialité.

Ensuite on vérifie pour chaque case si les contraintes sont respectées, si ce n'est pas le cas plusieurs cas peuvent apparaître :

- Si la dernière intervention est un Ri ou un Di, elle est replacée le plus tôt possible au début de la première journée suivante où apparaît la même spécialité (si c'est un Di, à la suite des Ri déjà prévus).
- Si l'avant dernière intervention est un Ri et que la dernière n'est ni un Ri ni un Di, c'est la première intervention que l'on va replacer le plus tôt possible au début de la première journée suivante où apparaît la même spécialité. En effet, ne pas laisser un Ri comme dernière intervention permet de ne pas avoir à repousser ce Ri une nouvelle fois si durant la simulation les interventions précédentes prennent trop de temps et que son intervention ne peut avoir lieu.
- Si l'avant dernière intervention n'est pas un Ri et la dernière n'est ni un Ri, ni un Di, alors la dernière intervention va essayer d'être replacée un jour suivant à la fin d'une programmation d'une salle de même spécialité. Si aucune salle ne peut l'accueillir, elle fera parti des déplacés, c'est à dire des nouveaux Di (les Di2).

6.6 Module Simulation

6.6.1 Fonctionnement

A chaque début de journée, le module lit le planning d'affectations et crée les évènements d'arrivées des patients dans l'hôpital. Ensuite, l'évènement de replanification est crée, sauf si c'est le dernier jour de la période. Puis les évènements sont triés par ordre d'apparition au plus tôt.

L'échéancier est lancé, les événements s'enchaînent jusqu'à la fin de la journée en commençant par les éventuels sortie de patients.

6.6.2 Les évènements

Arrivée d'un patient

Prend un lit, change l'état du patient de "arrivé" à "attente" et si ce patient est le premier de la journée a être opéré dans cette salle, génère un événement "Préparation de l'opération" à la date d'ouverture de la salle.

Préparation de l'opération

Vérifie si l'opération peut se produire, c'est à dire si il reste du temps disponible dans la salle pour que la durée moyenne de l'opération ne dépasse pas l'heure de fermeture de la salle moyennant une marge de 30 minutes. Si c'est le cas, alors on génère l'événement "Fin de l'opération" à la date courante + durée opératoire réelle. Sinon on essaie avec l'opération suivante dans la liste.

Fin de l'opération

Génère les événements "Préparation de l'opération" si il y a encore des patients prévus pour cette salle à la date courant, et "Sortie du patient" à la date courante + durée d'hospitalisation. Change l'état du patient de "attente" à "convalescence".

Sortie du patient

Lorsque le patient sort de l'hôpital, il libère un lit et son état passe de "convalescence" à "rentré".

Replanification

A chaque fin de journée, on trie les interventions qui n'ont pas pu être effectué dans la journée par ordre décroissant des durées opératoire moyennes et on essaie de les replanifier dans les jours suivants de la période. Si elles ne peuvent être déplacées alors elles sont mises dans la liste des reliquats de la période Ri.

6.7 Fonctionnalités

L'outil de simulation permet de choisir les fichiers contenant les données souhaitées. Ils sont au nombre de trois : les données générales du problème, le planning trouvé par une méthode de résolution à partir de ces données générales et la liste des patients arrivant pour chaque période.

Ensuite, il faut choisir les paramètres de la simulation : le nombre de périodes à simuler à chaque fois, la marge en minutes à imposer et l'algorithme à mettre en place parmi l'algorithme glouton, la méta-heuristique et un algorithme d'affectations respectant parfaitement le planning de résolution fourni.

Pour finir, le bouton "Lancer" permet de lancer la simulation, qui lorsqu'elle se termine, engendre l'apparition d'un bouton "Afficher les résultats" et permettra donc d'afficher les résultats de la simulation.

6.8 Interface graphique

Une petite interface graphique a été créé pour faciliter l'utilisation et donner de l'esthétisme à cet outil. Après avoir choisi les paramètres et les fichiers des différentes données nécessaires, la simulation est lancée et les résultats apparaissent dans une nouvelle fenêtre. On y retrouve un récapitulatif des données de l'hôpital et du simulateur, puis, différentes indicateurs sous formes de tableaux et de graphes.

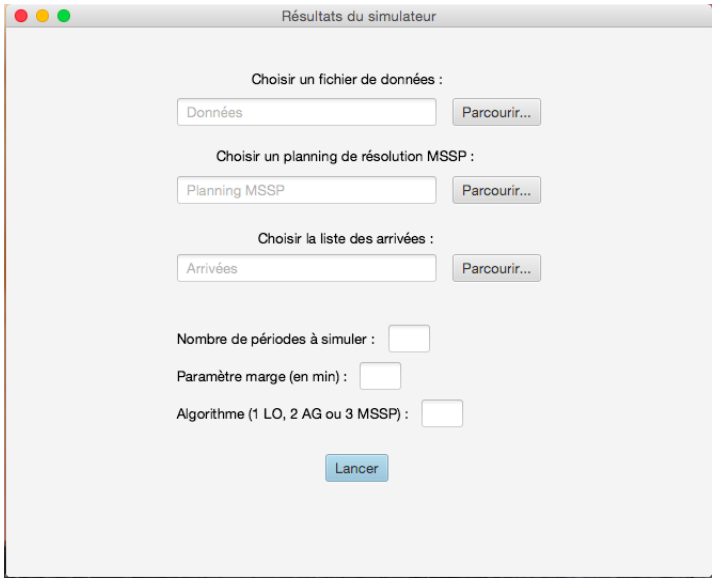


FIGURE 6.3 – Fichiers et paramètres à choisir

FIGURE 6.4 – Fin de la simulation

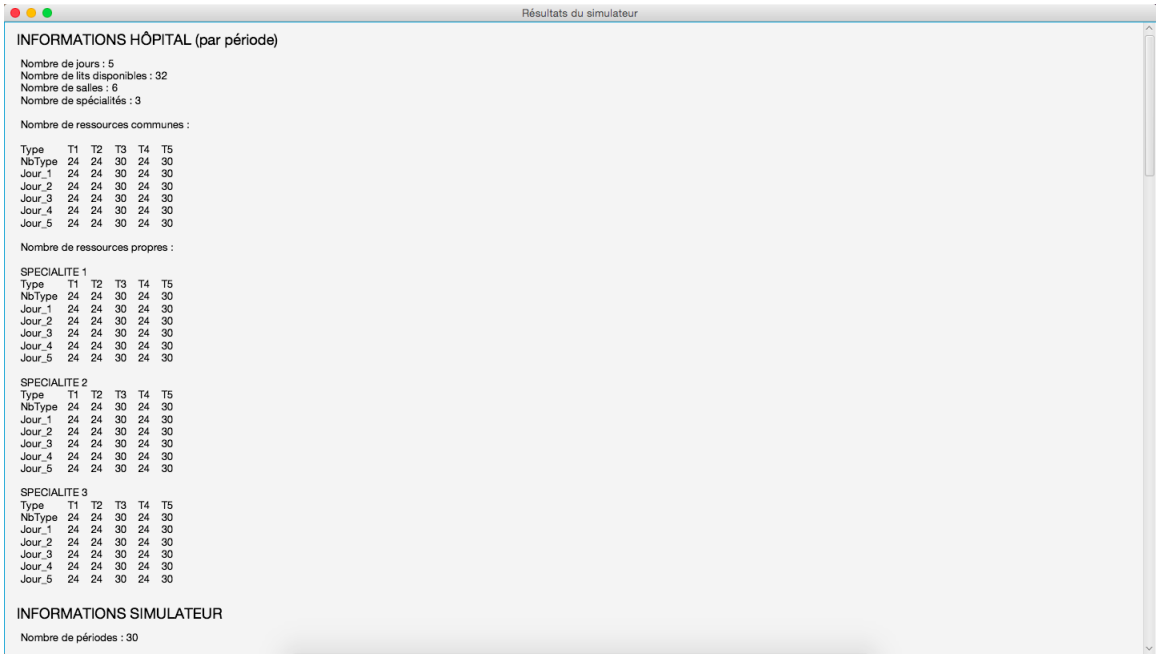


FIGURE 6.5 – Affichage des résultats : informations générales

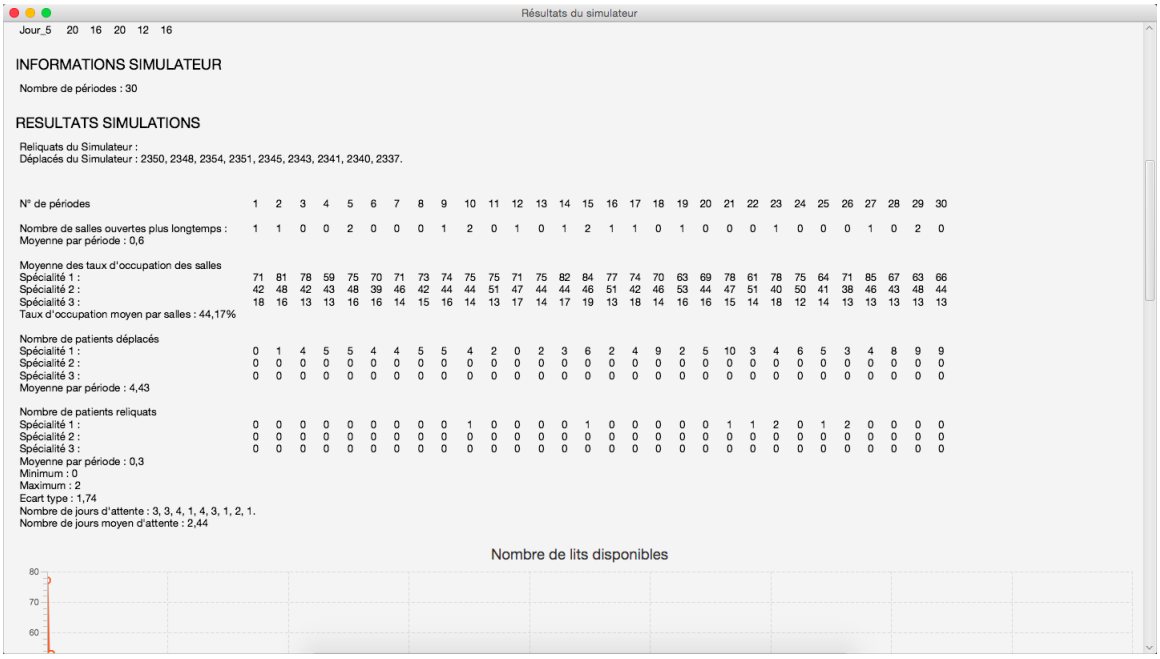


FIGURE 6.6 – Affichage des résultats : indicateurs sur les périodes

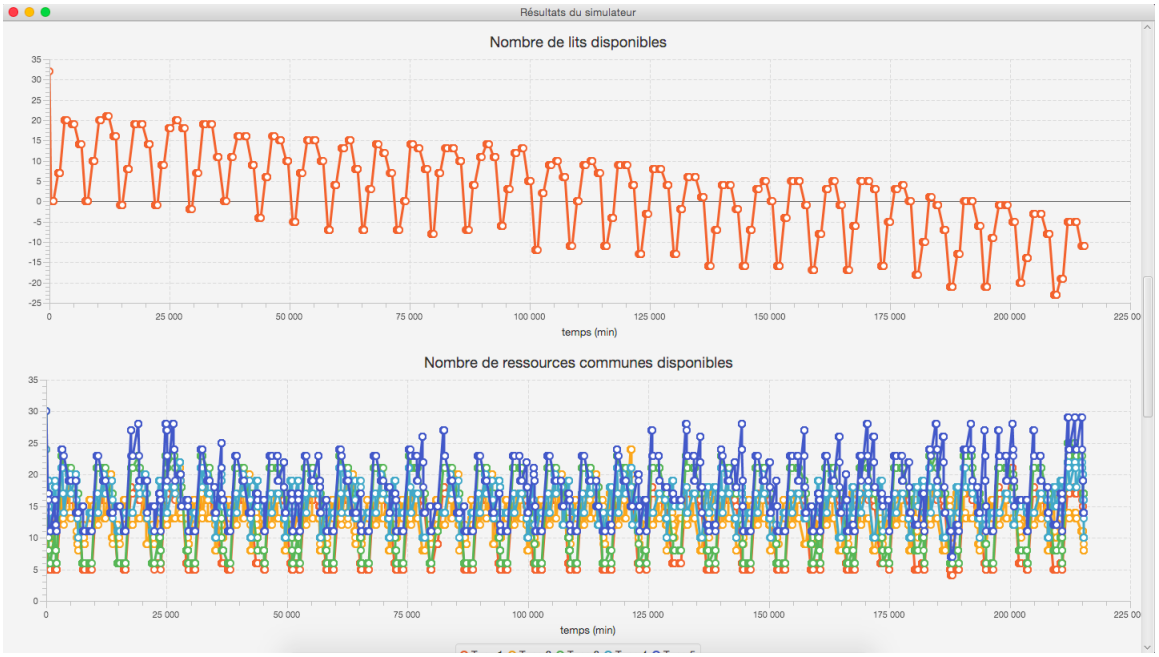


FIGURE 6.7 – Affichage des résultats : indicateurs en graphiques

7. Bilan du projet

7.1 Gestion et suivi de projet

Pour la gestion de ce projet, deux outils principaux ont été utilisés : Trello, un outil de gestion en ligne et GIT un gestionnaire de versions.

7.1.1 Trello

Trello est un outil de gestion de projet en ligne, lancé en septembre 2011, et inspiré par la méthode Kanban de Toyota. Il est basé sur une organisation des projets en planches listant des cartes (tâches). Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.

FIGURE 7.1 – Logo Trello

7.1.2 Git

Git est un système libre et Open Source de gestion de versions qui a été créé par Linus Torvalds (le créateur du système d'exploitation open-source Linux) pour aider la mise à jour du code. Git permet de définir un répertoire de fichiers comme un «référentiel» que Git contrôlera afin de suivre les changements des lignes individuelles du code dans chacun de vos fichiers de projet. Git permet d'avoir de nombreuses versions différentes du référentiel existant dans différents endroits qui ensuite seront synchronisé ensemble ligne par ligne en fusionnant tous les changements de manière transparente.

FIGURE 7.2 – Logo Git

7.1.3 Bitbucket

Bitbucket est un service qui fournit un endroit pour stocker une copie de votre référentiel de projet en toute sécurité. Il est possbile de «cloner» le répertoire de Bitbucket à un ordinateur local, apporter des modifications aux fichiers, 'commit' des changements dans Git, puis synchroniser les modifications en 'poussant' et 'tirant depuis le répertoire principal Bitbucket.

FIGURE 7.3 – Logo Bitbucket

7.2 Planification des tâches

Le projet a été découpé en tâches, dont on a estimé le temps de réalisation, afin de planifier le projet dans la durée impartie. Le planning initial se trouve en annexes.

- Du 18 au 24 sept : Découverte du sujet et définitions des étapes du projet.
- Du 25 sept au 11 dec : Développement d'une méthode de résolution en programmation par contraintes.
- Du 13 dec au 8 janv : Rédaction du cahier de spécifications et du plan de développement.
- Du 8 janv au 25 fev : Amélioration d'une méthode de résolution par une méta-heuristique.
- Du 17 fev au 16 avril : Développement d'un simulateur de périodes.

Cette planification a été à peu près respectée, la tâche d'implémentation des changements de la méta-heuristique a pris moins de temps que prévu mais la dernière phase sur le développement du simulateur, étant complexe, a Ce projet étant découpé en trois parties et la planification ayant été réalisé en début d'année, les deuxième et troisième parties concernant respectivement la méta-heuristique et le simulateur, ont été difficiles à planifier correctement car leurs différentes tâches ont été étudiées plus tard, à leur commencement.

7.3 Outils utilisés

7.3.1 Environnement de développement

Eclipse

Eclispe est un projet Open Source basé sur Java qui permet à un développeur de créer un environnement de développement personnalisé grâce à des plugins construits par des membres de la communauté Eclipse. Principalement écrit en Java, Eclipse peut être utilisé pour développer des applications.

FIGURE 7.4 – Logo Eclipse

Code Blocks

Code Blocks est un logiciel Open Source disposant d'un environnement de développement intégré en C et C++. Il est conçu pour être extensible et complètement configurable grâce à une multitude d'outils pour un travail de développement sur n'importe quelle plateforme.

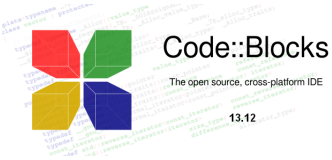


FIGURE 7.5 – Logo Code Blocks en version 13.12

7.3.2 JavaFX

JavaFX est une technologie créée par Sun Microsystems appartenant à Oracle depuis 2009, qui est devenu l'outil de création d'interface graphique officiel du langage Java pour toutes les sortes d'application (applications mobiles, applications sur poste de travail, applications Web...) JavaFX contient des outils très divers, notamment pour les médias audio et vidéo, le graphisme 2D et 3D, la programmation Web, la programmation multi-fils etc.

FIGURE 7.6 – Logo de JavaFX

8. Conclusion

Travailler sur le sujet du Master Surgical Scheduling Problem m'a permis d'aller plus loin sur ce problème que j'avais déjà abordé l'an dernier. Aimant particulièrement l'optimisation, j'ai beaucoup apprécié ce sujet car il concerne un domaine concret, la médecine, et il permet de faciliter le travail des hôpitaux et les conditions d'hospitalisations des patients.

Les objectifs fixés au début du projet ont été atteints, les deux méthodes de résolutions ont été développés ainsi qu'un outil de simulation. Étant donné la complexité du problème, la méthode exacte n'a pas été concluante car demandant trop de temps d'exécution et de mémoire pour toutes les variables nécessaires. La méthode approchée a été améliorée et s'approche plus de la solution optimale qu'auparavant, mais les temps d'exécutions restent encore important. Le simulateur est fonctionnel, il permet de mettre en évidence les difficultés rencontrées sur la disponibilité des ressources sur les périodes ainsi que les temps d'attentes des patients.

Afin d'aller plus loin sur ce sujet, il serait intéressant de comparer la méta-heuristique en place avec des instances de grandes tailles avec la méthode de résolution utilisant le solveur Cplex créée par Antoine GIRET dans son projet de fin d'études de l'an dernier.

Le simulateur pourrait être amélioré, notamment dans la planification des interventions du module ORS, et d'autres fonctionnalités pourraient être ajoutés dans l'interface graphique, afin de visualiser tous les indicateurs nécessaires.

Annexes

A. Modèles mathématiques

A.1 Modèle mathématique 1

Données

Données générales

— b_j : Nombre de lits disponibles le jour j
— J : Ensemble de jours dans la période

Salles

— h_{oj} : Durée d'ouverture de la salle o le jour j
— $a_{min_{sj}}$: Nombre minimum de salles à affecter pour la spécialité s le jour j
— O : Ensemble des salles opératoires

Ressources

— QRC_{ju} : Nombre de ressources communes de type u disponibles le jour j
— QRS^s_{ju} : Nombre de ressources propres de type u à la spécialité s disponibles le jour j
— BRC_{ku} : Besoin en ressources communes de type u pour l'intervention de type k
— BRS^s_{ku} : Besoin en ressources propres de type u à la spécialité s pour l'intervention de type k
— RC : Ensemble des ressources communes
— RS^s : Ensemble des ressources propres à la spécialité s

Spécialité

— $Fmax_s$: Nombre maximum de jours entre 2 programmations de la spécialité s
— S : Ensemble des spécialités

Interventions

— τ_k : Taux d'arrivée pour l'intervention k
— p_k : Durée de l'intervention k
— l_k : Durée d'hospitalisation de l'intervention k
— μ_k : Pondération de l'intervention k
— K : Ensemble des interventions
— K_s : Ensemble des interventions de la spécialité s

Variables

- $\mathbf{x_{ojk}} \in \{\mathbf{0}, \lfloor \frac{\mathbf{h_{oj}}}{\mathbf{p_k}} \rfloor\}$: entier égal au nombre d'interventions de type k programmées dans la salle o le jour j,$\forall o \in O; \forall j \in J; \forall k \in K$.

- $\mathbf{y_{ojk}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si au moins une intervention de type k est programmée dans la salle o le jour j, $\forall o \in O; \forall j \in J; \forall k \in K$.
- $\mathbf{z_{ojs}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si la spécialité s est programmée dans la salle o le jour j,$\forall o \in O; \forall j \in J; \forall s \in S$.

Contraintes

Respect de la durée d'ouverture de la salle opératoire

$$\sum_{k \in K} (p_k \times x_{ojk}) \leq h_{oj};\ \forall j \in J,\ \forall o \in O \tag{A.1}$$

Respect du minimum de salles à affecter par jour pour une spécialité

$$\sum_{o \in O} z_{ojs} \geq a_{min_{sj}};\ \forall s \in S,\ \forall j \in J \tag{A.2}$$

Respect du nombre maximum de journées sans programmation d'une spécialité

$$\sum_{i=j+1}^{Fmax_s+j} \sum_{o \in O} z_{o(i \bmod J)s} \geq 1;\ \forall s \in S,\ \forall j \in J \tag{A.3}$$

Affectation au maximum d'une seule spécialité par salle

$$\sum_{s \in S} z_{ojs} = 1;\ \forall j \in J,\ \forall o \in O \tag{A.4}$$

Relation entre les variables

$$si\ x_{ojk} \geq 1\ alors\ z_{ojs} = 1;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{A.5}$$

$$si\ z_{ojs} = 1\ alors\ x_{ojk} = 0;\ \forall k \notin K_s,\ \forall j \in J,\ \forall o \in O \tag{A.6}$$

$$si\ y_{ojk} = 1\ alors\ x_{ojk} \geq 1;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{A.7}$$

$$si\ y_{ojk} = 0\ alors\ x_{ojk} = 0;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{A.8}$$

$$si\ x_{ojk} = 0\ alors\ y_{ojk} = 0;\ \forall k \in K_s,\ \forall j \in J,\ \forall o \in O \tag{A.9}$$

Respect de la disponibilité des ressources communes

$$\sum_{o \in O} (\max_{k \in K}(y_{ojk} \times BRC_{ku})) \leq QRC_{ju};\ \forall u \in RC,\ \forall j \in J \tag{A.10}$$

Respect de la disponibilité des ressources propres à une specialité

$$\sum_{o \in O} (\max_{k \in K_s}(y_{ojk} \times BRS^s_{ku})) \leq QRS^s_{ju};\ \forall u \in RS^s,\ \forall j \in J,\ \forall s \in S \tag{A.11}$$

Respect du taux d'arrivée de patients (suppression de liste d'attente)

$$\sum_{o \in O} \sum_{j \in J} x_{ojk} \geq \tau_k;\ \forall k \in K \tag{A.12}$$

Respect du nombre de lits disponibles par jour

$$\sum_{o\in 0}\sum_{g\in J}\sum_{k\in K}(x_{ogt}\times(\lceil\frac{l_k+(g-j)}{|J|}\rceil-\beta))\leq b_j \begin{cases}\beta=1\ si\ g>j\\ \beta=0\ sinon\end{cases};\forall j\in J \tag{A.13}$$

Fonction objectif

$$\max\sum_{k\in K}\mu_k\sum_{o\in 0}\sum_{j\in J}x_{ojk} \tag{A.14}$$

A.2 Modèle mathématique 2

Données

Données générales

- b_j : Nombre de lits disponibles le jour j
- J : Ensemble de jours dans la période

Salles

- h_{oj} : Durée d'ouverture de la salle o le jour j
- $a_{min_{sj}}$: Nombre minimum de salles à affecter pour la spécialité s le jour j
- O : Ensemble des salles opératoires

Journées types

- Ω : Ensemble des journées types
- Ω_s : Ensemble des journées types de la spécialité s

Ressources

- QRC_{ju} : Nombre de ressources communes de type u disponibles le jour j
- QRS^s_{ju} : Nombre de ressources propres de type u à la spécialité s disponibles le jour j
- BRC_{ku} : Besoin en ressources communes de type u pour l'intervention de type k
- BRS^s_{ku} : Besoin en ressources propres de type u à la spécialité s pour l'intervention de type k
- γC_{tu} : Besoin en ressources communes de type u pour la journée type t
- γS^s_{tu} : Besoin en ressources propres de type u à la spécialité s pour la journée type t
- RC : Ensemble des ressources communes
- RS^s : Ensemble des ressources propres à la spécialité s

Spécialités

- $Fmax_s$: Nombre maximum de jours entre 2 programmations de la spécialité s
- S : Ensemble des spécialités

Interventions

— τ_k : Taux d'arrivée pour l'intervention k
— α_k^t : Nombre d'interventions k programmées dans la journée type t
— p_k : Durée de l'intervention k
— l_k : Durée d'hospitalisation de l'intervention k
— μ_k : Pondération de l'intervention k
— K : Ensemble des interventions
— K_s : Ensemble des interventions de la spécialité s

Variables

- $\mathbf{x_{ojt}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si la journée type t est programmée dans la salle o le jour j, $\forall o \in O; \forall j \in J; \forall t \in \Omega$.
- $\mathbf{w_{js}} \in \{\mathbf{0}, \mathbf{1}\}$: binaire égale à 1 si la spécialité s est programmée le jour j, $\forall j \in J; \forall s \in S$.

Contraintes

Respect de la durée d'ouverture de la salle opératoire

$$si \sum_{k \in K_s} (\alpha_k^t \times p_k) > h_{oj} \; alors \; x_{ojt} = 0; \; \forall j \in J, \; \forall o \in O, \; \forall t \in \Omega_s \tag{A.15}$$

Respect du minimum de salles à affecter par jour pour une spécialité

$$\sum_{o \in 0} \sum_{t \in \Omega_s} x_{ojt} \geq a_{min_{sj}}; \; \forall j \in J, \; \forall s \in S \tag{A.16}$$

Respect du nombre maximum de journées sans programmation d'une spécialité

$$\sum_{i=j}^{j+F_{max_s}} w_{(i \bmod |J|)s} \geq 1; \; \forall j \in J, \; \forall s \in S \tag{A.17}$$

Affectation d'une seule journée type par salle

$$\sum_{t \in \Omega} x_{ojt} = 1; \; \forall j \in J, \; \forall o \in O \tag{A.18}$$

Relation entre les variables

$$si \sum_{t \in \Omega_s} x_{ojt} = 0 \; alors \; w_{js} = 0; \; \forall j \in J, \; \forall o \in O, \; \forall s \in S \tag{A.19}$$

$$si \; x_{ojt} = 1 \; alors \; w_{js} = 1; \; \forall t \in \Omega_s, \; \forall j \in J, \; \forall o \in O \tag{A.20}$$

Respect de la disponibilité des ressources communes

$$\sum_{o \in 0} \sum_{t \in \Omega} (x_{ojt} \times \gamma C_{tu}) \leq QRC_{ju}; \; \forall j \in J, \; \forall u \in RC \tag{A.21}$$

Respect de la disponibilité des ressources propres à une spécialité

$$\sum_{o \in 0} \sum_{t \in \Omega_s} (x_{ojt} \times \gamma S_{tu}^s) \leq QR_{ju}^s; \; \forall j \in J, \; \forall u \in RS_s, \; \forall s \in S \tag{A.22}$$

Respect du taux d'arrivée de patients (suppression de liste d'attente)

$$\sum_{o\in 0}\sum_{j\in J}\sum_{t\in\Omega}(x_{ojt}\times\alpha_k^t)\geq\tau_k;\ \forall k\in K_s \tag{A.23}$$

Respect du nombre de lits disponibles par jour

$$\sum_{o\in 0}\sum_{g\in J}\sum_{t\in\Omega}x_{ogt}\sum_{k\in K_s}(\alpha_k^t\times(\lceil\frac{l_k+(g-j)}{|J|}\rceil-\beta))\leq b_j\begin{cases}\beta=1\ si\ g>j\\ \beta=0\ sinon\end{cases};\forall j\in J \tag{A.24}$$

Fonction objectif

$$\max\sum_{k\in K}\mu_k\sum_{o\in 0}\sum_{j\in J}\sum_{t\in\Omega}(x_{ojt}\times\alpha_k^t) \tag{A.25}$$

B. Résultats des tests de la méta-heuristique

Se trouve ci-dessous les résultats des tests effectués sur la méta-heuristique, permettant d'évaluer les valeurs des différents paramètres. On y retrouve le Gap pour atteindre la solution optimale, le temps d'éxécution en secondes et le nombre d'instances trouvées non réalisables (NR).

B.1 Opérateur de croisement

Test	Croist vertical 1 pt Op 50	Croist 2 pt vertical Op 50	Croist 1 pt horiz Op 50
Gap	3,09	3,26	3,53
Temps	538	537	556

Test	Croist 2 pt horiz Op 50	Croist 1 pt vertical Op 200	Croist 2 pt vertical Op 200
Gap	3,72	2,66	2,82
Temps	535	2106	2091

Test	Croist 1 pt horiz Op 200	Croist 2 pt horiz Op 200
Gap	3,02	3,19
Temps	2114	2100

B.2 Opérateur de sélection

Test	Sélection Enfants Op 50	Sélection 50% Op 50	Sélection élite Op 50
Gap	95,30	96,82	95,70
Temps	526	536	526

Test	Sélection Enfants Op 200	Sélection 50% Op 200	Sélection élite Op 200
Gap	96,14	97,31	96,13
Temps	2089	2093	2048

B.3 Nombre d'individus et d'itérations

Test	Pop 100 Ite 100 Op 50	Pop 100 Ite 150 Op 50	Pop 100 Ite 200 Op 50
Gap	96,87	97,05	95,09
Temps	926	1370	1821
NR	0	1	1

Test	Pop 150 Ite 100 Op 50	Pop 150 Ite 150 Op 50	Pop 150 Ite 200 Op 50
Gap	97,14	97,25	97,46
Temps	1418	2117	2802
NR	0	0	0

Test	Pop 200 Ite 100 Op 50	Pop 200 Ite 150 Op 50	Pop 200 Ite 200 Op 50
Gap	97,21	97,36	97,66
Temps	1931	2873	3839
NR	0	0	0

B.4 Opérateurs de voisinages

Test	Pop 100 Ite 100 Op Ech 50	Pop 100 Ite 100 Op ChgSpe 50	Pop 100 Ite 100 Op CrsList 50
Gap	5,05	7,19	12,17
Temps	85	370	107
NR	7	9	11

Test	Pop 100 Ite 100 Op Amel 50	Pop 150 Ite 100 Op Ech 50	Pop 150 Ite 100 Op ChgSpe 50
Gap	10,40	5,05	6,71
Temps	517	145	561
NR	5	6	10

Test	Pop 150 Ite 100 Op CrsList 50	Pop 150 Ite 100 Op Amel 50	Pop 100 Ite 100 Op Amel 0
Gap	11,91	9,80	3,72
Temps	175	792	482
NR	11	5	4

Test	Pop 100 Ite 100 Op CrsList 0	Pop 100 Ite 100 Op ChgSpe 0	Pop 100 Ite 100 Op Ech 0
Gap	4,00	3,62	3,99
Temps	905	649	896
NR	0	3	2

Test	Pop 150 Ite 100 Op Amel 0	Pop 150 Ite 100 Op CrsList 0	Pop 150 Ite 100 Op ChgSpe 0
Gap	3,49	3,76	3,31
Temps	754	1378	1003
NR	5	0	2

Test	Pop 150 Ite 100 Op Ech 0	Pop 100 Ite 100 Ech/CrsList 50	Pop 100 Ite 100 Ech/Amel 50
Gap	3,64	4,66	4,80
Temps	1381	212	777
NR	2	6	2

Test	Pop100 Ite100 Ech/ChgSpe50	Pop100 Ite100 ChgSpe/CrsList50	Pop100 Ite100 ChgSpe/Amel50
Gap	3,97	5,93	6,00
Temps	539	561	1752
NR	6	8	2

Test	Pop 100 Ite 100 Op CrsList/Amel 50
Gap	5,96
Temps	1170
NR	6

B.5 Paramètre booléen

Test	Strict true	Strict false
Gap	96,35	96,76
Temps	1737	1732

B.6 Tests supplémentaires

Test	Pop 50 Ite 100 Op 50	Pop 100 Ite 50 Op 50	Pop 100 Ite 100 Op 25
Gap	95,91	95,55	95,94
Temps	426	437	454
NR	1	1	1

C. Planification des tâches

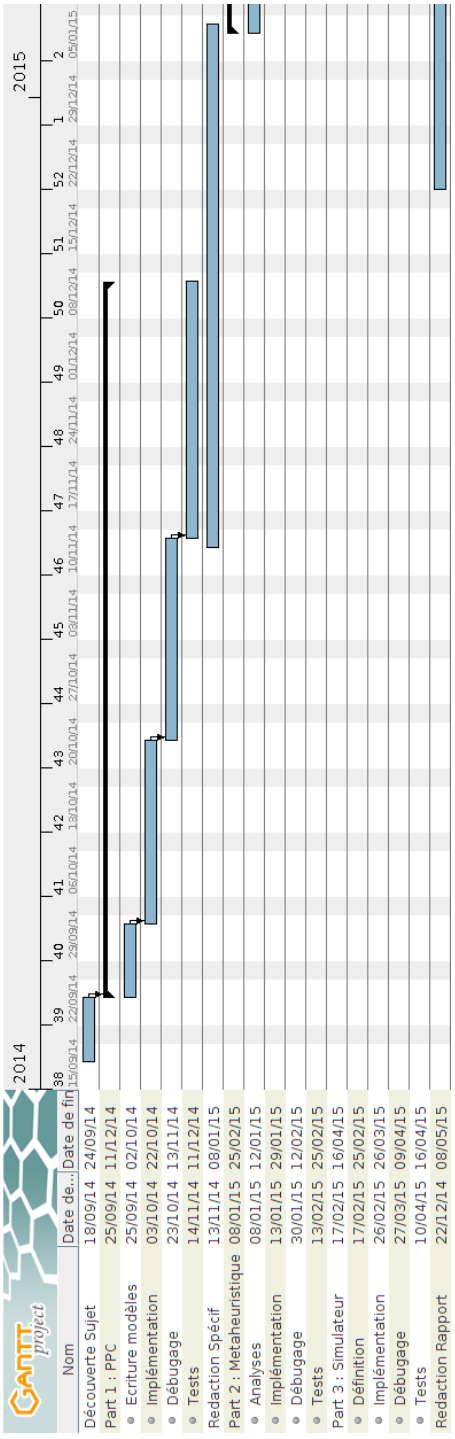


FIGURE C.1 – Planification du projet - 1er semestre

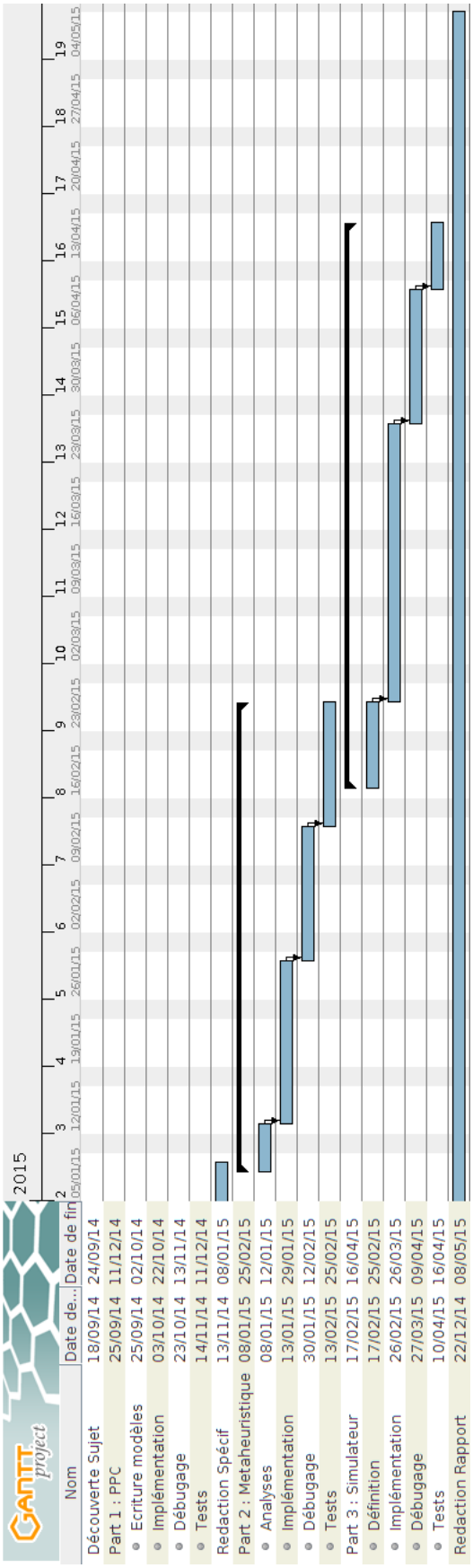


FIGURE C.2 – Planification du projet - 2ème semestre

Master Surgical Scheduling Problem

Département Informatique
5e année
2014 - 2015

Rapport de Projet de Fin d'Études

Résumé : Rapport du projet de fin d'études de Katy DUFEUTRELLE intitulé "Master Surgical Scheduling Problem" qui traite de la programmation d'interventions dans des salles opératoires. Ce projet se décompose en 3 parties dont deux de méthodes de résolution (Programmation par contraintes et Méta-heuristique) et le développement d'un simulateur visant à évaluer les solutions trouvées par les résolutions.

Mots clefs : MSSP, Ordonnancement, Blocs opératoires, Programmation par contraintes, Méta-heuristique, Simulateur.

Abstract: Graduation project report by Katy Dufeutrelle entitled "Master Surgical Scheduling Problem" dealing with the planning of interventions in operating rooms. This project is divided into three parts including two resolution methods (Constraint Programming and Meta-heuristic) and development of a simulator to evaluate the solutions found by the resolutions.

Keywords: MSSP, Scheduling, Operating rooms, Constraint programming, Meta-heuristic, Simulator.

Encadrant
Yannick KERGOSIEN
yannick.kergosien@univ-tours.fr

Étudiant
Katy DUFEUTRELLE
katy.dufeutrelle@etu.univ-tours.fr

Université François-Rabelais, Tours

DI5 2014 - 2015