# Dossier d'Analyse Conception

Adrien Burgun, Oscar Dewasmes

UTBM - IT45 - Printemps 2022

# Table des matières

# 1 Introduction

Ce document est un rapport d'analyse conception préparant la réalisation du projet d'IT45 printemps 2022 à l'UTBM. Ce projet est réslisé par *Adrien Burgun* et *Oscar Dewasmes*. L'objectif est de mettre en place un algorithme de recherche optimisation pour résoudre un problème d'emplois du temps pour les intervenants du SESSAD.

Le code source du projet ainsi que de ce documents sont disponibles sur *Github* :
github.com/kalharko/it45-project

# 2 Analyse

## 2.1 Paramètres

Nous avons identifié les paramètres suivants :

— $n$ le nombre d'agents du SESSAD
— $m$ le nombre de missions à réaliser dans une semaine
— $v$ la vitesse de déplacement des agents
— $CA_i$ la compétence que possède l'agent i
— $SA_i$ la spécialité de l'agent i
— $CM_j$ la compétence nécessaire à la mission i
— $DM_j$ la date du début de la mission $j$
— $FM_j$ la date de la fin de la mission $j$
— $VH_i$ le volume horaire de l'agent i

## 2.2 Variables

Nous avons décidé de représenter une solution par une matrice binaire, dans la quelle chaque colonne représente une mission et chaque ligne représente un agent.

| | $m_0$ | $m_1$ | ... | $m_o$ |
|---|---|---|---|---|
| $a_0$ | 1 | 0 | ... | 1 |
| $a_1$ | 0 | 1 | ... | 0 |
| $\vdots$ | $\vdots$ | $\vdots$ | | $\vdots$ |
| $a_n$ | 0 | 0 | ... | 0 |

## 2.3 Objectifs

— Harmoniser la charge de travail et la distance parcourue entre les agents du SESSAD.
— Minimiser le nombre d'affectation qui où l'agent n'a pas la bonne spécialité.
— Minimiser le nombre d'heures supplémentaires, le nombre d'heures perdus et la distance totale parcouru.

## 2.4 Contraintes

Pour une solution $S \in \mathcal{M}_{n\times o}(\{0,1\})$, on a les contraintes suivantes :

1. Une mission ne peut être assignée qu'à un intervenant ayant la même compétence : $\forall(i,j), S_{ij} \Rightarrow CM_i = CA_j$
2. Chaque intervenant ne peut réaliser qu'une mission à la fois : $\forall i, \forall(j_1,j_2), S_{ij_1} \wedge S_{ij_2} \Rightarrow DM_{j_1} \geq FM_{j_2} + v\Delta_{j_1j_2} \vee DM_{j_2} \geq FM_{j_1} + v\Delta_{j_1j_2}$
3. Une mission est réalisée par un et un seul intervenant : $\forall i, \sum_{j=0}^{n} S_{ij} = 1$
4. Accorder à chaque intervenant au moins une heure de pause midi par jour entre midi et 14h. Pour vérifier cette condition, on regarde l'heure de fin de la dernière mission avant

13h et l'heure de début de la première mission après 13h, et on s'assure qu'il y a au moins une heure entre les deux (en prenant compte du temps de trajet) :

$$\begin{aligned}\delta^d_{i,t} &= \min_j(DM_j | S_{ij} \wedge \text{heure}(DM_j) \geq 13 \wedge \text{jour}(DM_j) = t)\\ \delta^f_{i,t} &= \max_j(FM_j + v\Delta_{j,j+1} | S_{ij} \wedge \text{heure}(FM_j) \leq 13 \wedge \text{jour}(FM_j) = t)\\ \forall(i,t),\ \delta^d_{i,t} - \delta^f_{i,t} &\geq 1\end{aligned}$$

5. Respecter les heures maximales à travailler par semaine ($VH_i$, pour lesquelles on compte le temps de la mission ainsi que le temps de trajet) :

$$\begin{aligned}\forall(i,t), T(i,t) = &\sum_{j=0}^{n-1}\{FM_j - DM_j \mid S_{ij} \wedge \text{jour}(FM_j) = t\}\\ &+ \sum_{j=0,j'=0}^{\min(j,j')\leq n-1} \left\{v\Delta_{jj'} \mid j \neq j' \wedge S_{ij} \wedge S_{ij'} \wedge \text{jour}(FM_j) = t\right\}\\ &+ v\Delta_{0,jmin} \quad \text{Avec} : jmin = \arg\min_j \{DM_j \mid S_{ij} \wedge \text{jour}(FM_j) = t\}\\ &+ v\Delta_{jmax,0} \quad \text{Avec} : jmax = \arg\max_j \{FM_j \mid S_{ij} \wedge \text{jour}(FM_j) = t\}\\ \forall i, \sum_t T(i,t) &\leq VH_i\end{aligned}$$

6. Respecter les heures maximales à travailler par jour : $\forall(i,t), T(i,t) \leq 10$
7. Respecter la limite des heures supplémentaires autorisées : $\forall i, \sum_t \max\{T-8, 0\} = 10$
8. Respecter l'amplitude de la journée de travail de chaque intervenant :

$$\begin{aligned}\forall(i,t),\ &\max_j \{FM_j \mid S_{ij} \wedge \text{jour}(FM_j)\}\\ &- \min_j \{DM_j \mid S_{ij} \wedge \text{jour}(DM_j)\}\\ &+ v\Delta_{0,jmin} + v\Delta_{jmax,0} \leq 12\end{aligned}$$

# 3 Conception

Nous avons décidé de réaliser ce projet en c.

## 3.1 Algorithme

Nous allons implémenter l'algorithme du *recuit*. Pour cela nous allons avoir besoin de déterminer une solution finale, une fonction d'évaluation et un moyen de déterminer les voisins d'une solution.

### 3.1.1 Solution initiale

Pour pouvoir initier l'algorithme du *recuit*, il faut déterminer une solution initiale. Pour cela, nous utilisons une forme relaxée des règles :

— Une mission peut ne pas être réalisée
— Deux missions pour un agent peuvent se chevaucher
— On peut dépasser les heures maximales à travailler par jour
— On peut dépasser la limite des heures supplémentaires autorisées
— On peut ne pas respecter l'amplitude de la journée de travail

On commence par affecter à tous les agent du SESSAD une mission par jour qui convient parfaitement à leur compétences et leur spécialité. Puis nous répétons le processus jusqu'à ce que l'on ne puisse plus faire d'affectation. De la même manière, on affecte ensuite aux agents du SESSAD des missions non optimales où leur spécialité ne correspond pas. Lorsque l'on ne peut à nouveau plus faire d'affectation, on vérifie si les conditions sont toutes remplies. Si elles le sont, alors on prend cette solution comme solution initiale et on procède à la suite de l'algorithme.

Si une ou plusieurs conditions ne sont pas remplies, alors on procède à chercher une solution initiale par algorithme génétique :

— Le génome est constitué de $n$ gènes $g_j \in Agents \cup \{\varnothing\}$
— On convertit chaque génome $g$ en une solution $S$ avec la relation $S_{ij} \Leftrightarrow g_j = i$
— On donne un score à chaque individu : si la solution $S$ ne remplit pas les règles relaxées, alors il obtient une pénalité de $\infty$ ; sinon, elle obtient une pénalité proportionnelle au temps dépassant les limites des règles strictes et au nombre de missions non-réalisées
— On élimine tous les individus ayant une pénalité de $\infty$, ainsi que les individus ayant la plus haute pénalité
— Une partie des meilleurs individus obtient le droit de se reproduire : l'opérateur de croisement revient à choisir un jour $t$ et à échanger les gènes de ce jour : $\{g_j \mid \text{jour}(FM_j) = t\}$
— L'étape de mutation se fait en modifiant un ou plusieurs gènes
— Un faible nombre d'individus neufs sont introduits, afin de maintenir la diversité du bassin génétique

Une fois que le meilleur individu a un score nul, on le choisit comme solution initiale.

### 3.1.2 Fonction d'évaluation

La fonction d'évaluation se sépare en 2 étapes, la vérification de la validité de la solution puis l'évaluation de la solution par rapport aux objectifs. Pour vérifier la validité de la solution, nous allons vérifier les contraintes une à unes.

Pour évaluer la solution nous pouvons prendre en compte ses caractéristiques suivantes :
- $c_0$ = Nombre de mission assignées
- $c_1$ = Nombre de missions assignées à un agent qui n'a pas la bonne spécialité
- $c_2$ = Nombre d'heures supplémentaires
- $c_3$ = Distance totale parcourue
- $\vdots$

Ainsi le score d'une solution serait calculé ainsi :

$$score = \lambda_0 c_0 + \lambda_1 c_1 + \lambda_2 c_2 + \cdots + \lambda_n c_n$$

Avec $\lambda_i$ l'importance que nous assignons au critère $i$. Par exemple le nombre d'heure supplémentaire est un critère que nous voulons beaucoup minimiser, ainsi son $\lambda$ associé serait une grande valeur négative.

### 3.1.3 Déterminer les voisins d'une solution

Pour déterminer une solution voisine à la solution actuel, une mission et un agent sont choisis au hasard. Cet agent remplace celui qui était affecté précédemment à la mission. Si cette nouvelle affectation n'est pas valide, une nouvelle mission et un nouvel agent sont tirés au hasard jusqu'à ce que une affectation valide apparaisse.

### 3.1.4 Algorithme du recuit

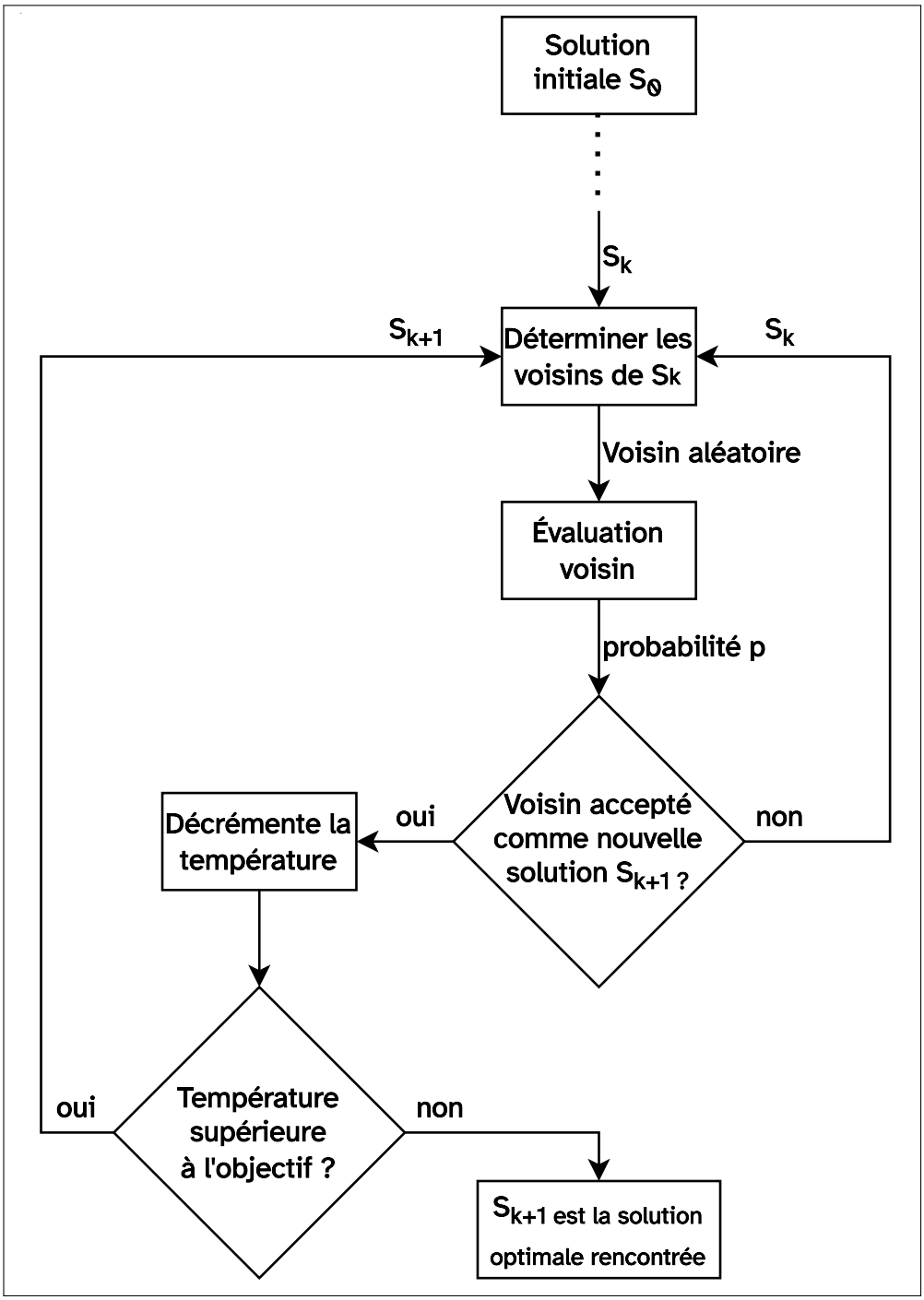



*Figure 1 : Flow chart représentant l'exécution de l'algorithme du recuit*

Le flow chart ci-dessus représente les étapes constituant l'algorithme du *recuit*. À cause de l'étape de sélection d'un voisin aléatoire et l'acceptation de la prochaine solution basé sur une probabilité ; cet algorithme peut donner différents résultats. C'est pour cela qu'il est appliqué plusieurs fois à un même problème, pour contrecarrer la possibilité d'aberrances probabilistes qui nous éloigneraient de la solution optimale.

# 4 Réalisation

Par rapport à notre analyse initiale, nous avons ajusté notre approche sur les points suivants :

- Évaluation multi-objetif en cascade au lieu de moyenne pondérée.
- La solution est maintenant représenté par une liste d'assignation $x_i = j$ où la mission $i$ est assigné à l'agent $j$.
- Certaines contraintes, comme l'assignation unique d'une solution, sont directement prises en compte par la forme de la solution.
- Pour palier à la nature aléatoire du *recuit*, l'optimisation peut être lancée plusieurs fois pour garder la meilleur solution.

Nous avons aussi instauré des test unitaires en utilisant la librairie *unity*. Cela nous a permit de s'assurer de la qualité du code que nous avons écrit.

La dernière mesure que nous avons prise pour s'assurer de la qualité du code est l'utilisation de *valgrind* qui permet la détection des fuites de mémoire.

# 5 Résultats

Notre programme est capable de trouver une solution initiale pour les 3 instances du problèmes fournies.

Sur l'instance « 45-4 », nous obtenons comme solution :

| mission | agent | mission | agent |
|---|---|---|---|
| 1 | 1 | 26 | 2 |
| 2 | 3 | 27 | 1 |
| 3 | 1 | 28 | 3 |
| 4 | 1 | 29 | 1 |
| 5 | 3 | 30 | 3 |
| 6 | 0 | 31 | 0 |
| 7 | 2 | 32 | 2 |
| 8 | 2 | 33 | 0 |
| 9 | 0 | 34 | 2 |
| 10 | 1 | 35 | 0 |
| 11 | 3 | 36 | 1 |
| 12 | 1 | 37 | 3 |
| 13 | 3 | 38 | 3 |
| 14 | 2 | 39 | 3 |
| 15 | 0 | 40 | 1 |
| 16 | 0 | 41 | 2 |
| 17 | 2 | 42 | 0 |
| 18 | 0 | 43 | 2 |
| 19 | 3 | 44 | 0 |
| 20 | 1 | 45 | 2 |
| 21 | 3 | | |
| 22 | 1 | | |
| 23 | 2 | | |
| 24 | 0 | | |
| 25 | 0 | | |

TABLE 1 – Une solution finale pour 45-4

La distance totale parcourue est de 423.7 km et 28 affectations ont la mauvaise spécialité. Les scores à optimiser ont pour valeur :

— $f_{\text{employees}} = 0.587$
— $f_{\text{students}} = 60.0$
— $f_{\text{SESSAD}} = 484.87$

Pour le problème « 100-10 », on obtient une solution avec une distance totale parcourue de 1096 km et 72 affectations avec la mauvaise spécialité. Les scores sont :

— $f_{\text{employees}} = 1.129$
— $f_{\text{students}} = 72.0$
— $f_{\text{SESSAD}} = 911.37$

### 5.1 Taux de succès des fonctions d'initialisation

Nous avons mesuré le taux de succès des deux fonctions d'initialisation (l'heuristique « naïve »et l'algorithme génétique), en fonction du nombre d'agents et du nombre de missions. Plus précisément, nous mesurons

$$\text{perf} = \mathbb{E}(\text{valid}(f(X, \text{Seed})) \mid \varepsilon)$$

Avec $X$ un problème suivant une loi uniforme pour les compétences, spécialisations, horaires et durées ; $f$ la fonction donnant une solution initiale à mesurer et $\varepsilon = (\texttt{n\_agents}, \texttt{n\_missions})$

Pour $f_1 = \texttt{build\_naive}$ (heuristique seulement), on obtient la distribution suivante :

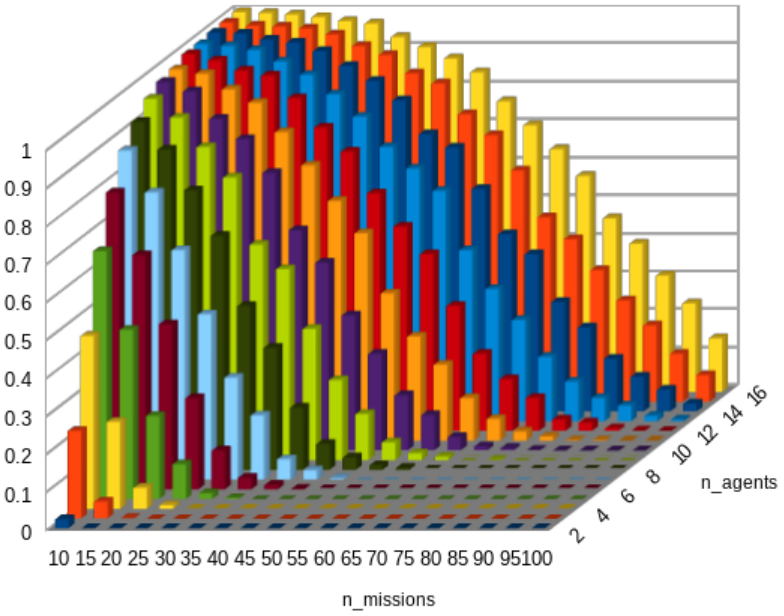



FIGURE 1 – Taux de succès de `build_naive`

Pour $f_2 = \texttt{build\_initial\_solution}$ (algorithme génétique), on obtient la distribution suivante :

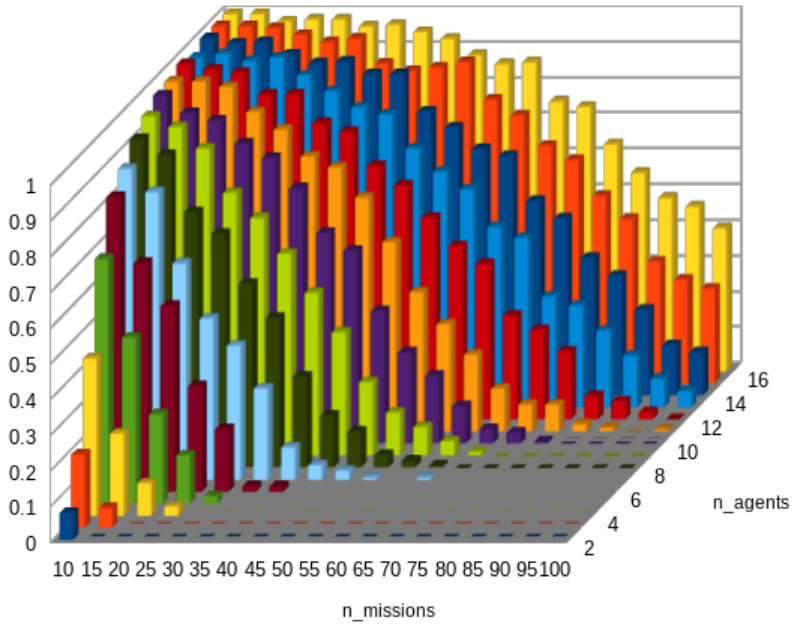



FIGURE 2 – Taux de succès de `build_initial_solution`

Et voici la distribution de $\mathbb{E}(\neg\text{valid}(f_1(X, \text{Seed})) \wedge \text{valid}(f_2(X, \text{Seed})) \mid \varepsilon)$, qui représente la

contribution de l'algorithme génétique à trouver une solution initiale :

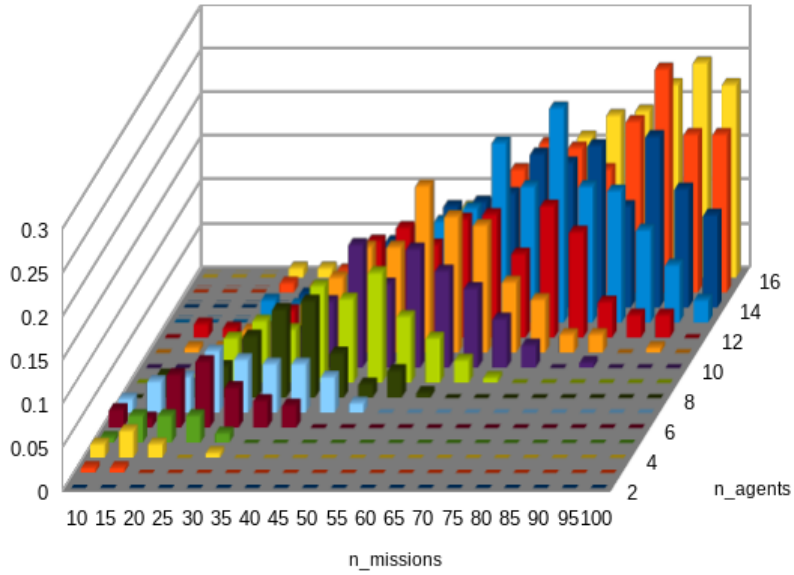



FIGURE 3 – Différence du taux de succès de `build_naive` et `build_initial_solution` (échelle ré-ajustée)

## 5.2 Évolution des objectifs au cours de la résolution

Ci dessous est représenté l'évolution des objectifs au fil des itérations. Les barres verticales représentent le changement d'objectif dans la cascade.

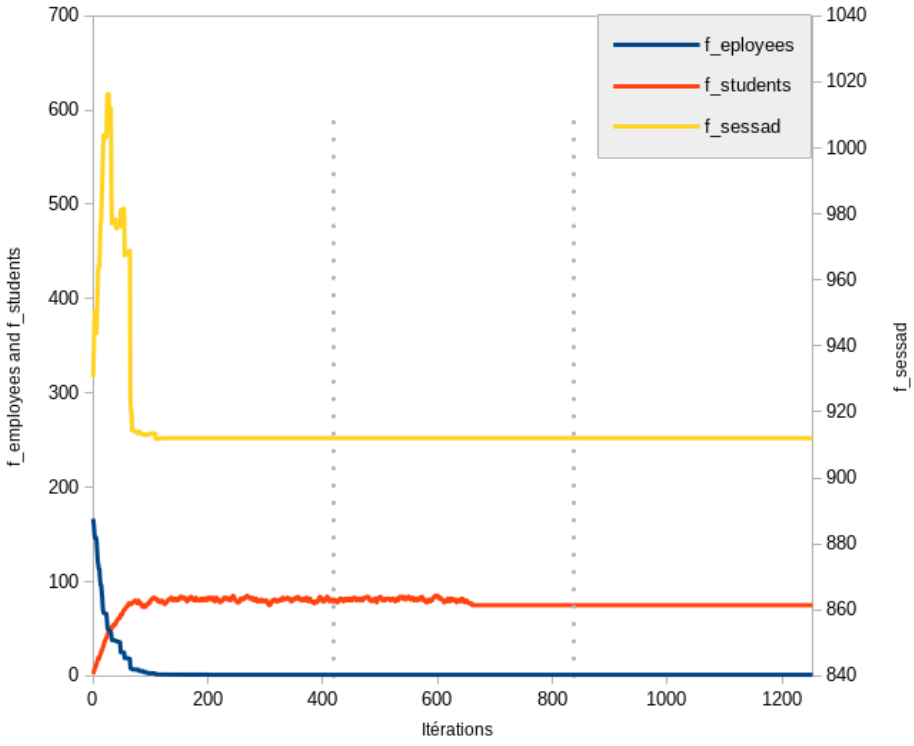



FIGURE 4 – Évolution des objectifs au fil des itérations

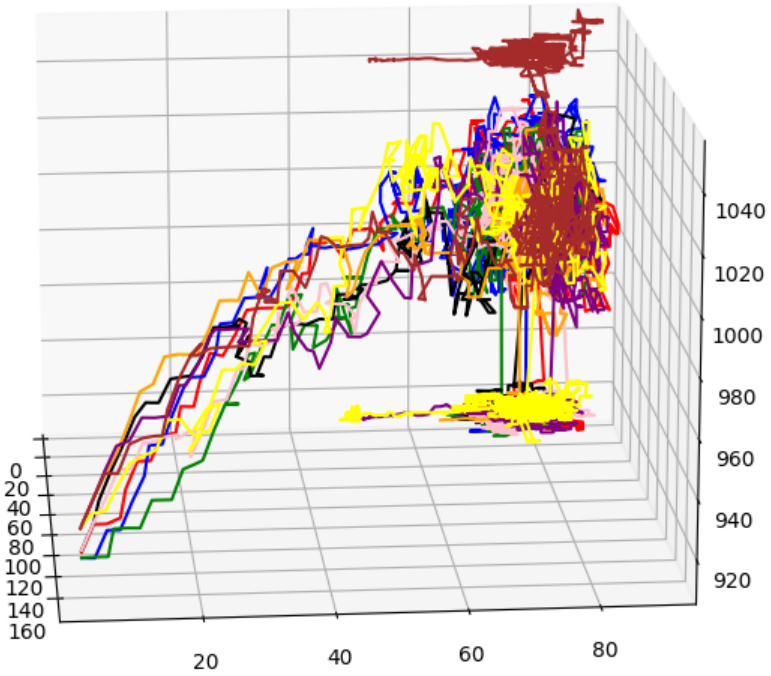



FIGURE 5 – Représentation du chemin suivi par 10 solutions initiales différentes lors de l'optimisation

### 5.3 Visualisations de la solution

FIGURE 6 – Chemin tracé par les agents dans la solution initiale ($\Sigma = 1063.87 \pm 15.15$ km)

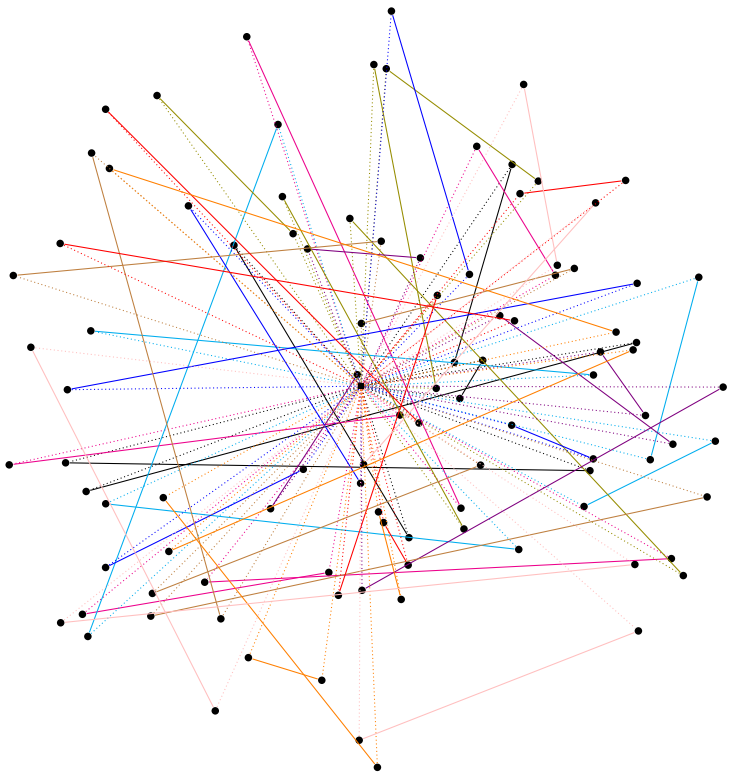
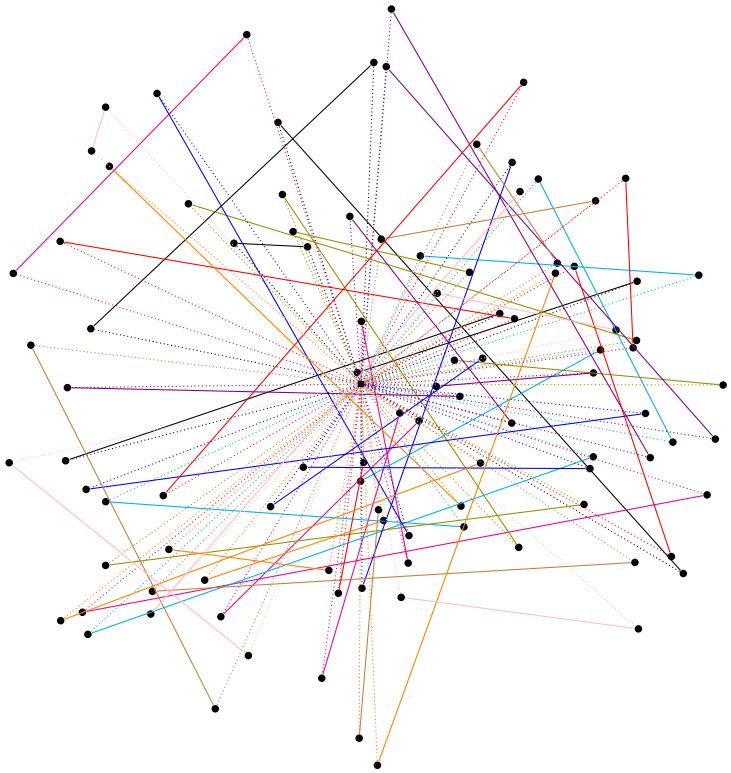


FIGURE 7 – Chemin tracé par les agents dans la solution finale ($\Sigma = 1096.43 \pm 1.16$ km)



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| TRAJ | TP | 1 | lundi | 7:56 | 8:00 | 1 | Distanciel |
| MISS | CM | 1 | lundi | 8:00 | 12:00 | 1 | Distanciel |
| TRAJ | TP | 1 | lundi | 12:00 | 12:03 | 1 | Distanciel |
| TRAJ | TP | 1 | lundi | 13:50 | 14:00 | 1 | Distanciel |
| MISS | CM | 1 | lundi | 14:00 | 17:00 | 1 | Distanciel |
| TRAJ | TP | 1 | lundi | 17:00 | 17:09 | 1 | Distanciel |

FIGURE 8 – Extrait d'un emploi du temps généré par le programme

Pour générer une image à partir de l'emploi du temps créé par le programme, nous utilisons le site web de conversion d'emploi du temps UTBM créé par un élève, d'où les informations superflues dans l'emploi du temps généré par le programme : Générateur d'emploi du temps UTBM

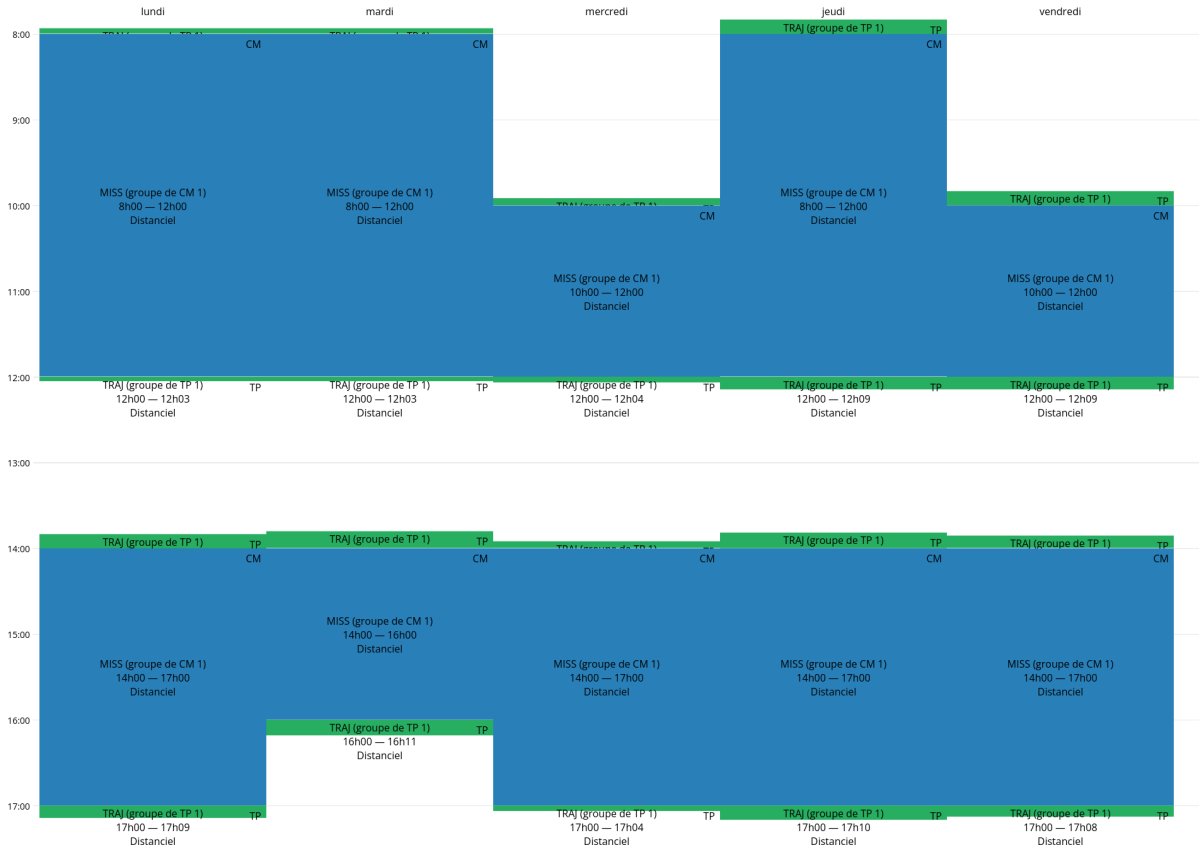



FIGURE 9 – Image rendue de l'emploi du temps d'un agent

# 6 Conclusion

Nous avons pu mettre en place et implémenter un algorithme cherchant à optimiser les affectations d'agents du SESSAD. Notre implémentation utilise une heuristique et un algorithme génétique afin d'obtenir une solution initiale, puis utilise l'algorithme de recuit simulé afin de trouver une solution plus optimale.

Nous n'avons pas rencontré de grandes difficulté lors de ces deux étapes, en dehors des problèmes liés à la gestion manuelle du programme, qui a été écrit en C. La mémoire utilisée par programme a été vérifié avec `valgrind` et des tests unitaires vérifient le bon fonctionnement des différents composants du programme.

Vous trouverez le code source du projet sur *Github* :
github.com/kalharko/it45-project

## 6.1 Pistes d'amélioration

— Lorsque le programme est lancé avec plusieurs itérations pour garder la meilleur solution, seul le premier objectif de la cascade est pris en compte pour choisir la meilleur solution.
— Génération d'instances du problème complètement intégré au programme à des fins de test.
— Plus de tests unitaires.
— Comparer différentes manières de calculer les objectifs.