

ST7-OSTP - Planification quotidienne d'une équipe mobile (DecisionBrain)

Cléa Chataigner, Tom Collignon, Antoine de Bouillé, Olivia Moyal, Victor Neyt

1^{er} avril 2022

Sommaire

1	Premier modèle exact simplifié	2
1.1	Formalisation du problème	2
1.1.1	Objectif	2
1.1.2	Variables de décision	2
1.1.3	Paramètres	3
1.1.4	Contraintes	3
1.2	Implémentation	5
1.2.1	Instances d'entrée	5
1.2.2	Constantes à récupérer	5
1.2.3	Variables	5
1.2.4	Code	5
1.3	Analyse des résultats	6
2	Deuxième modèle exact étendu	9
2.1	Formalisation du problème	9
2.1.1	Objectif	9
2.1.2	Variables de décision	9
2.1.3	Paramètres	10
2.1.4	Contraintes	10
2.2	Implémentation	12
2.2.1	Instances d'entrée	12
2.2.2	Constantes à récupérer	12
2.2.3	Variables	13
2.2.4	Code	13
2.3	Analyse des résultats	13
3	Résolution approchée	16
3.1	Introduction aux métahéuristiques	16
3.2	Méthode par recuit simulé	16
3.2.1	Explication de la méthode et application à notre situation	16
3.2.2	PseudoCode de la méthode du recuit simulé	17
3.2.3	PseudoCode de la fonction pour calculer les voisins d'un planning	17
3.3	Analyse des résultats	18
3.4	Méthode de l'évolution différentielle	18

Résumé

This report for Decision brain presents our project in ST7 to optimally assign tasks to agents in specific cities, respecting availability constraints among others. We present our results obtained with Gurobi and a metaheuristic simulated annealing method.

Introduction

DecisionBrain est une start-up développant des solutions d'optimisation et d'aide à la décision pour la logistique, l'industrie, la planification de main d'oeuvre et la chaîne d'approvisionnement (supply chain) sur tous les continents. Dans le cadre d'une mission réalisée pour le compte d'un client, nous intervenons afin de réaliser une partie du projet : un outil de planification d'une équipe de techniciens mobiles pour créer les routes quotidiennes de 500 techniciens devant réaliser quelques 10 000 tâches. L'entreprise TuttoBene fait donc appel à nous afin de lui développer le dit logiciel d'aide à la décision. L'activité de cette entreprise consiste à envoyer des techniciens chez des clients (particuliers ou entreprises) pour réaliser des tâches (e.g. entretien de chaudière, réparation électrique, chauffage). Ce rapport présente le développement de l'algorithme d'optimisation sur lequel s'appuie le logiciel.

Chapitre 1

Premier modèle exact simplifié

1.1 Formalisation du problème

Nous considérerons une version simplifiée du problème prenant compte des caractéristiques suivantes :

- l'objectif est de minimiser les coûts de transports ;
- tous les employés partent d'un même lieu dépôt ;
- chaque employé est disponible pendant un seul intervalle de temps ;
- aucune période d'indisponibilité des employés n'est considérée (notamment la pause midi) ;
- les lieux des tâches sont ouverts sur un seul créneau dans la journée ;
- les tâches ne peuvent être affectées qu'aux employés ayant un niveau de compétence suffisant.

1.1.1 Objectif

Notre objectif est de minimiser les coûts de transport. Nous utilisons pour cela la formalisation du Vehicle Routing Problem with Time Windows (VRPTW)[1]. Nous définissons notre problème sur le graphe $G = (V, A)$, avec comme sommets les tâches à effectuer et le dépôt, et avec K l'ensemble des employés (la table *Employees*). Le dépôt est représenté par les noeuds 0 et $n+1$.

Le graphe est complet (toutes les tâches sont reliées entre elles), pondéré, et orienté. Nous avons pour chaque arc les pondérations c_{ij} égales au temps que l'on met à parcourir l'arc (i, j) ($= \frac{d_{ij}}{50}$ car les employés se déplacent en voiture à une vitesse moyenne de 50km/h).

1.1.2 Variables de décision

Nous disposons des variables de flux :

$$x_{ijk} = \begin{cases} 1 & \text{si l'arc } (i, j) \text{ est utilisé par l'employé } k \\ 0 & \text{sinon} \end{cases} \quad \text{avec } (i, j) \in A, i \neq j, k \in K$$

En notant n_o le nombre d'employés, et n_t le nombre de tâches, i peut donc prendre les valeurs de 0 à n_t (aucun employé ne peut sortir du dépôt $n_t + 1$ donc nous ne considérerons pas l'arc $(n_t + 1, j)$) et j de 1 à $n_t + 1$ (aucun employé ne peut rentrer au dépôt 0 donc nous ne considérerons pas l'arc $(i, 0)$), et k de 1 à n_o . Nous imposons $i \neq j$ pour les variables x_{ijk} car l'employé k ne peut pas aller du noeud i au noeud i .

Nous avons également les variables continues w_i , $i \in \llbracket 1, n_t \rrbracket$, qui indiquent à quelle heure (en minutes) l'employé attribué à la tâche i la commence.

Nous pouvons alors reformuler notre objectif comme suit :

$$\boxed{\text{Minimize} \sum_{k=1}^{n_o} \sum_{i=0}^{n_t} \sum_{\substack{j=1 \\ j \neq i}}^{n_t+1} c_{ij} x_{ijk}}$$

1.1.3 Paramètres

Nous listons ci-dessous l'ensemble des paramètres utiles à l'expression des contraintes.

- d_k et f_k les heures de début et de fin de l'employé k (pour $k \in \llbracket 1, n_o \rrbracket$) ;
- s_i la durée de la tâche i (pour $i \in \llbracket 1, n_t \rrbracket$) ;
- q_k le niveau de l'employé k (pour $k \in \llbracket 1, n_o \rrbracket$) ;
- m_i le niveau minimal requis pour la tâche i (pour $i \in \llbracket 1, n_t \rrbracket$) ;
- a_i et b_i les heures d'ouverture et de fermeture respectivement de la tâche i (pour $i \in \llbracket 1, n_t \rrbracket$).

1.1.4 Contraintes

Dépôt

Chaque employé doit retourner au dépôt s'il en est parti :

$$\forall k \in K, \quad \sum_{j=1}^{n_t+1} x_{0jk} = \sum_{i=0}^{n_t} x_{i(n_t+1)k} \quad (1.1)$$

Débit

Si un employé fait la tâche i , donc 'rentre' au noeud i , il doit en ressortir, sauf pour les noeuds correspondant au dépôt :

$$\sum_{\substack{j=1 \\ j \neq i}}^{n_t+1} x_{ijk} = \sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket \quad (1.2)$$

Attribution des tâches

Une tâche doit être visitée exactement une fois (1 seul employé peut effectuer la tâche) :

$$\sum_{k=1}^{n_o} \sum_{\substack{j=1 \\ j \neq i}}^{n_t+1} x_{ijk} = 1 \quad \forall i \in \llbracket 1, n_t \rrbracket \quad (1.3)$$

Heures de travail

Les employés ne peuvent travailler que sur les horaires où ils sont disponibles. Nous avons :

$$(\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik})(d_k + c_{0i}) \leq (\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik}) * w_i \leq (\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik})(f_k - s_i - c_{i(n_t+1)}) \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket$$

Nous devons multiplier par $(\sum_{j=0, j \neq i}^{n_t} x_{jik})$ car cela n'est vrai que si l'employé k effectue la tâche i , donc qu'il 'entre' dans le noeud i .

Cette contrainte n'est pas linéaire, il faut donc la linéariser. Pour cela on introduit $H = 24 * 60 = 1440$ qui jouera le rôle d'une constante très grande par rapport aux données du problème. On considère la première inégalité :

$$\left(\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \right) (d_k + c_{0i}) \leq \left(\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \right) * w_i \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket$$

On obtient alors :

$$\left(\sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \right) (d_k + c_{0i} - w_i) \leq 0 \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket$$

Cette contrainte est donc équivalente à :

$$(d_k + c_{0i} - w_i) \leq \left(1 - \sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \right) * H \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket \quad (1.4)$$

En effet, on a $\sum_{j=0, j \neq i}^{n_t} x_{jik} = 0$ ou 1 . Si $\sum_{j=0, j \neq i}^{n_t} x_{jik} = 0$, l'inégalité sur le terme $d_k + c_{0i} - w_i$ ne fournit aucune information, on veut donc que ce soit la même chose pour l'inégalité équivalente, ce qui est fait (en fixant H suffisamment grand) avec $d_k + c_{0i} - w_i \leq H$ qui ne fournit aucune information. De plus, si $\sum_{j=0, j \neq i}^{n_t} x_{jik} = 1$ on a bien la négativité de $d_k + c_{0i} - w_i$. On fait de même pour la seconde inégalité et on obtient la contrainte équivalente suivante :

$$(w_i - f_k + s_i + c_{i(n+1)}) \leq \left(1 - \sum_{\substack{j=0 \\ j \neq i}}^{n_t} x_{jik} \right) * H \quad \forall k \in K, i \in \llbracket 1, n_t \rrbracket \quad (1.5)$$

Compétences

La tâche doit être réalisée par quelqu'un qui en a les compétences. Pour une tâche fixée i , nous ne savons pas quel agent k intervient mais nous savons que pour cet agent k nous devons avoir $m_i \leq q_k$. Pour obtenir ce bon indice k , nous devons donc utiliser les doubles sommes suivantes :

$$\sum_{k=1}^{n_o} \sum_{\substack{j=1 \\ j \neq i}}^{n_t+1} x_{ijk} m_i \leq \sum_{k=1}^{n_o} \sum_{\substack{j=1 \\ j \neq i}}^{n_t+1} x_{ijk} q_k \quad \forall i \in \llbracket 1, n_t \rrbracket \quad (1.6)$$

Durée d'une tâche

Si l'employé k va de la tâche i à la tâche j , nous avons $w_i + s_i + c_{ij} \leq w_j$.

D'où :

$$\left(\sum_{k=1}^{n_o} x_{ijk} \right) (w_i + s_i + c_{ij} - w_j) \leq 0 \quad \forall i \in \llbracket 1, n_t \rrbracket, j \in \llbracket 1, n_t \rrbracket$$

Cette contrainte n'est pas linéaire, il faut donc la linéariser. Pour cela on introduit $H = 1440$ qui jouera le rôle d'une constante très grande par rapport aux données du problème et nous obtenons la contrainte équivalente :

$$w_i + s_i + c_{ij} - w_j \leq \left(1 - \sum_{k=1}^{n_o} x_{ijk} \right) * H \quad \forall i \in \llbracket 1, n_t \rrbracket, j \in \llbracket 1, n_t \rrbracket \text{ and } j \neq i \quad (1.7)$$

Ouverture d'une tâche

Les tâches ne sont ouvertes qu'à un seul créneau dans la journée. Nous avons :

$$a_i \leq w_i \leq b_i - s_i \quad \forall i \in \llbracket 1, n_t \rrbracket \quad (1.8)$$

1.2 Implémentation

1.2.1 Instances d'entrée

Chaque instance d'entrée est composée des feuilles suivantes :

- *Employees* : liste des employés. Pour chacun est indiqué son prénom, les coordonnées GPS de son lieu de résidence (dans cette première version, elles sont toutes égales aux coordonnées du dépôt), son domaine (cette colonne vaut tout le temps 'Oenology', donc ne sera pas utile au cours du projet), son niveau de compétences et ses horaires de début et de fin de travail ;
- *Unavailabilities* : liste des indisponibilités des employés. Pour chaque employé concerné on y retrouve les horaires de ses périodes d'indisponibilités et les coordonnées GPS du lieu où il se trouvera pendant ce temps. Nous n'utiliserons pas cette table dans cette première version, car nous considérons que chaque employée est disponible pendant un seul intervalle de temps (il n'y a pas de 'trous') ;
- *Tasks* : liste des tâches. Pour chaque tâche est indiqué son numéro, les coordonnées GPS du lieu où la réaliser, sa durée, le domaine de compétence et le niveau requis et les horaires d'ouverture pendant lesquels elle peut être réalisée ;
- *Tasks Unavailabilities* : liste des indisponibilités des tâches. Pour chaque tâche concernée, on y retrouve chaque créneau pendant lequel elle ne peut pas être effectuée, à l'intérieur des horaires données dans la feuille Tasks. Nous n'utiliserons pas cette table dans cette première version, car nous considérons les lieux des tâches ouverts sur un seul créneau dans la journée (il n'y a pas de 'trous').

Nous transformons chaque feuille en DataFrame Pandas.

1.2.2 Constantes à récupérer

Nous avons besoin, en comparant avec nos contraintes évoquées ci-dessus, de récupérer les informations suivantes, ordonnées selon des structures bien spécifiques :

- La matrice C : la matrice des coûts de transports c_{ij} , en calculant grâce aux coordonnées géographiques la distance entre la tâche i et la tâche j et en divisant ensuite par 50 (les techniciens se déplacent en voiture à une vitesse moyenne de 50km/h), grâce aux tables *Tasks* et *Employees* (car nous devons aussi calculer la distance entre les tâches et le dépôt) ;
- Le vecteur de compétences M , qui associe à chaque tâche i le niveau requis m_i , grâce à la table *Tasks* ;
- Les vecteurs d'ouverture et de fermeture des tâches, respectivement A et B, qui à chaque tâche i associent le début a_i et la fin b_i , grâce à la table *Tasks* ;
- Le vecteur de durée de tâche S, qui à chaque tâche i associe la durée s_i , grâce à la table *Tasks* ;
- Le vecteur du niveau des agents Q, qui à chaque agent k associe son niveau de compétences q_k , grâce à la table *Employees* ;
- Les vecteurs d'horaires de début et de fin de travail des agents, respectivement D et F qui à chaque agent k associent le temps de début d_k et de fin f_k , grâce à la table *Employees*.

1.2.3 Variables

Nous devons alors introduire via Gurobi les variables binaires x_{ijk} et les variables continues w_i .

1.2.4 Code

Il faut se référer au Jupyter Notebook correspondant pour voir le code associé.

1.3 Analyse des résultats

Nous présentons ici les temps d'exécution de notre algorithme sur les différentes instances, sur un MacBook Pro, avec un processeur 2 GHz Intel Core i5 double cœur et une mémoire 8 Go 1867 MHz LPDDR3, en comparant également les principales caractéristiques des instances en entrée.

Instances	Temps d'exécution (sec)	Nombre d'employés	Nombre de tâches
GuineaGolf	0,15	1	9
Bordeaux	3	2	10
Poland	259,04	2	19
Italy	771,71	3	21
Finland	> 20 000	4	25

Sur les Instances en Finlande, malheureusement, notre algorithme prenait trop de temps à s'exécuter et nous n'avons pas réussi à trouver de solution. De même, nous voyons dans le tableau ci-dessus que sur les instances en Italy, l'algorithme a également pris beaucoup de temps à s'exécuter. Nous pensons donc que notre modèle actuel n'est pas performant quand il y a beaucoup de données. Plus précisément, nous allons donner un ordre de grandeur du nombre de contraintes en fonction du nombre de tâches et du nombre d'employés, pour voir ce qui influe le plus sur notre modèle :

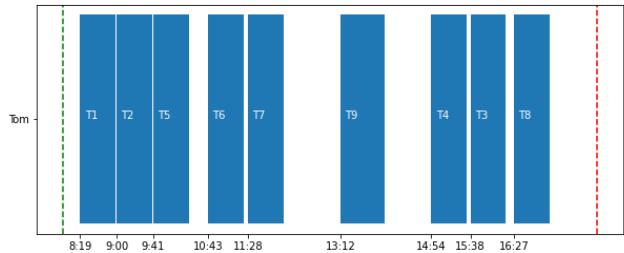
- La contrainte de dépôt crée K contraintes ;
- La contrainte de débit crée Kx_{nt} contraintes ;
- La contrainte d'attribution des tâches crée n_t contraintes ;
- La contrainte des heures de travail crée $2xKx_{nt}$ contraintes ;
- La contrainte des compétences crée n_t contraintes ;
- La contrainte de durée d'une tâche crée n_tx_{nt} contraintes ;
- La contrainte d'ouverture d'une tâche crée n_t contraintes.

Nous avons donc finalement $K + 3n_t + 2 * K * n_t + n_t^2$ contraintes, soit un nombre de contraintes en $\mathcal{O}(n_t^2 + Kn_t)$. En général, nous avons $K \ll n_t$ donc nous pouvons considérer que le nombre de contraintes est en $\mathcal{O}(n_t^2)$.

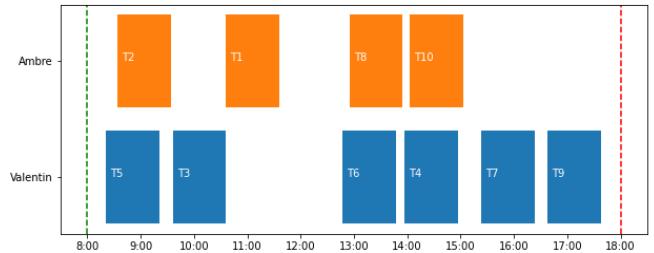
Instances	Temps d'exécution (sec)	Nombre de contraintes
GuineaGolf	0,15	127
Bordeaux	3	172
Poland	259,04	496
Italy	771,71	633
Finland	> 3 600	904

Nous mettons ci-dessous les emplois du temps des employés retournés par notre algorithme sur chaque Instance.

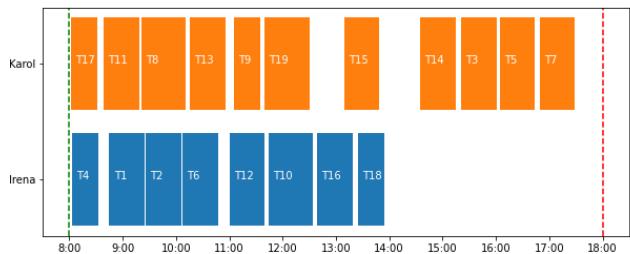
Pour les résultats cartographiques, nous mettons ici des captures d'écran mais nous conseillons de se référer aux fichiers .html sur lesquels plus d'informations sont données.



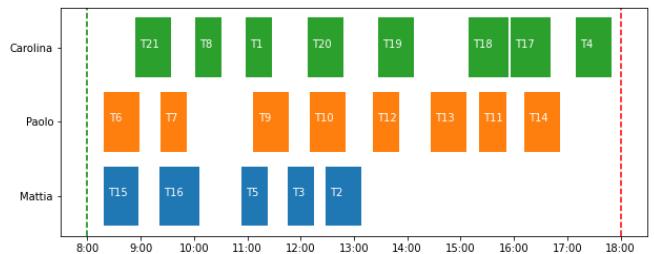
(a) Golfe de Guinée



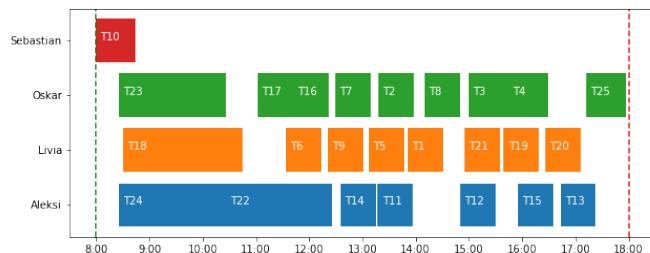
(b) Bordeaux



(c) Pologne

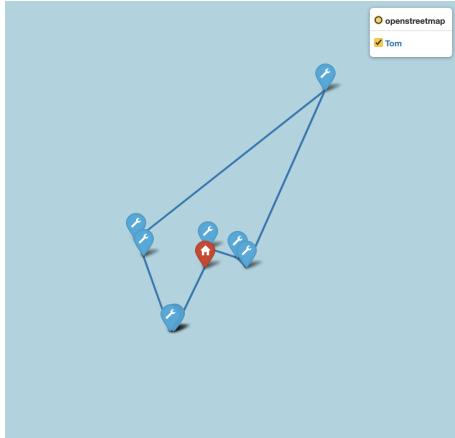


(d) Italie

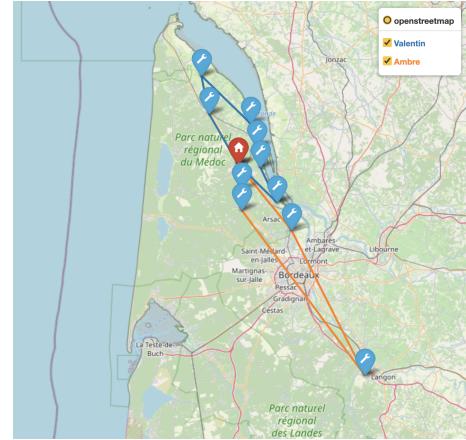


(e) Finlande

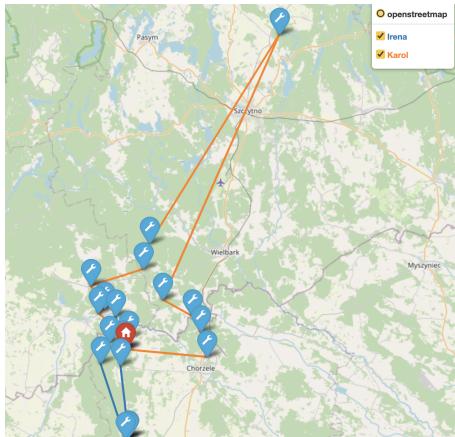
FIGURE 1.1 – Emplois du temps par instance



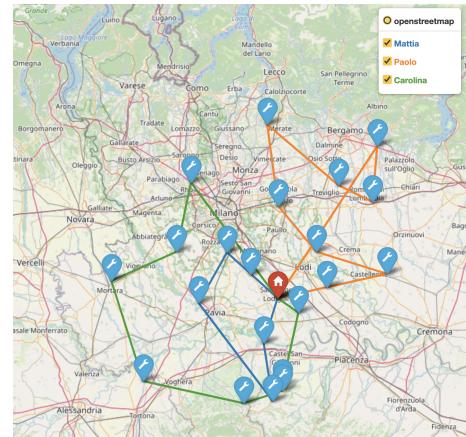
(a) Golfe de Guinée



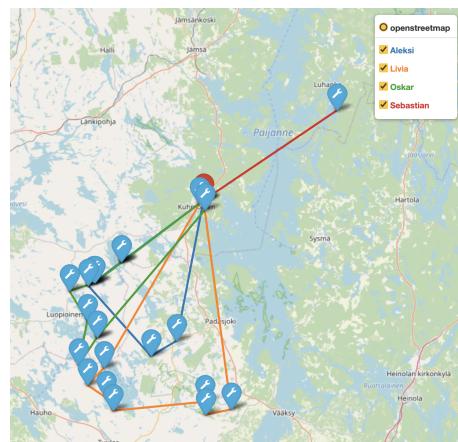
(b) Bordeaux



(c) Pologne



(d) Italie



(e) Finlande

FIGURE 1.2 – Résultats cartographiques

Chapitre 2

Deuxième modèle exact étendu

2.1 Formalisation du problème

Nous ajoutons au modèle précédent les points suivants :

- l'objectif est maintenant de maximiser la durée totale des tâches réalisées à coût opérationnel minimum ;
- chaque employé part en début de journée et rentre en fin de journée en un lieu qui lui est propre ;
- chaque employé possède une pause midi d'une durée d'1h à placer entre 12h et 14h, dans un lieu quelconque ;
- certains employés peuvent avoir des périodes d'indisponibilité dans la journée, associées à des lieux précisés. Les employés doivent se trouver en le lieu donné au début de la période et le quitter à la fin ;
- les lieux de certaines tâches peuvent être ouverts sur plusieurs créneaux disjoints dans la journée.

2.1.1 Objectif

Notre objectif est de maximiser la durée totale des tâches réalisées à coût opérationnel minimum. Nous utilisons la même formalisation (VRPTW) : notre graphe $G = (V, A)$ a maintenant comme sommets les tâches à effectuer, les maisons des employés et les lieux de leurs indisponibilités.

Tous les noeuds sont reliés entre eux, le graphe est donc toujours complet. Il est également toujours pondéré, et orienté. Nous avons pour chaque arc les pondérations c_{ij} égales au temps que l'on met à parcourir l'arc (i,j) ($= \frac{d_{ij}}{50}$ car les employés se déplacent en voiture à une vitesse moyenne de 50km/h).

2.1.2 Variables de décision

Nous disposons des mêmes variables de flux x_{ijk} .

Pour l'employé k ($k \in K$) nous définissons l'ensemble N_k qui regroupe l'ensemble des tâches, sa maison H_k et l'ensemble des lieux de ses indisponibilités qui lui sont propres. Nous notons également pour la suite du problème T l'ensemble des tâches, $H = \{H_k, \forall k \in K\}$ l'ensemble des maisons des employés et $U = \{U_k, \forall k \in K\}$ l'ensemble des indisponibilités par employé. Nous avons donc, à k fixé ($k \in K$), $(i, j) \in (N_k)^2$. Nous imposons toujours $i \neq j$ pour les variables x_{ijk} car l'employé k ne peut pas aller du noeud i au noeud i .

Nous avons également toujours les mêmes variables de temps w_i , $i \in T \cup U$.

Enfin, nous introduisons les variables p_k qui, pour chaque employé k ($k \in K$), représente l'heure de début de sa pause déjeuner.

Nous pouvons alors reformuler nos objectifs comme suit :

Objectif de maximisation des durées des tâches :

$$\text{Maximize} \sum_{k \in K} \sum_{i \in T} \sum_{\substack{j \in N_k \\ j \neq i}} s_i x_{ijk}$$

Objectif de minimisation des distances parcourues :

$$\text{Minimize} \sum_{k \in K} \sum_{i \in N_k} \sum_{\substack{j \in N_k \\ j \neq i}} c_{ij} x_{ijk}$$

Nous pondérons chaque objectif avec α_1 et α_2 .

Pour les durées des tâches, nous estimons que l'entreprise gagne 50€ par heure de tâche effectuée, soit $\frac{50}{60} \approx 0.8$ €/minute.
Nous estimons qu'un trajet d'1h à 50 km/h coûte 5€, soit $\frac{5}{60} \approx 0.08$ €/minute. D'où notre objectif final :

$$\text{Maximize } 0.8 * \sum_{k \in K} \sum_{i \in T} \sum_{\substack{j \in N_k \\ j \neq i}} s_i x_{ijk} - 0.08 * \sum_{k \in K} \sum_{i \in N_k} \sum_{\substack{j \in N_k \\ j \neq i}} c_{ij} x_{ijk}$$

2.1.3 Paramètres

Nous listons ci-dessous l'ensemble des paramètres utiles à l'expression des nouvelles contraintes, tout en gardant nos notations précédentes pour les niveaux des tâches et des employés.

— d_k et f_k les heures de début et de fin de l'employé k .

Nous notons n_k le nombre d'indisponibilités de l'employé k . Nous noterons d_k^l et f_k^l les heures de début et de fin de l'indisponibilité l , pour $l \in \llbracket 1, n_k \rrbracket$;

— s_i la durée de la "tâche" i ($i \in T \cup U$) ;

— a_i et b_i les heures d'ouverture et de fermeture respectivement de la tâche i ($i \in T \cup U$)

Si la tâche est ouverte sur n_i intervalles disjoints nous noterons a_i^r et b_i^r pour $r \in \llbracket 1, n_i \rrbracket$ les heures de début et de fin de l'intervalle r sur lequel la tâche i est ouverte.

2.1.4 Contraintes

Débit

Si un employé rentre au noeud i , il doit en ressortir :

$$\sum_{\substack{j \in N_k \\ j \neq i}} x_{jik} = \sum_{\substack{j \in N_k \\ j \neq i}} x_{ijk} \quad \forall k \in K, i \in N_k \quad (2.1)$$

Attribution

Une tâche doit être visitée au maximum une fois :

$$\sum_{k \in K} \sum_{\substack{j \in N_k \\ j \neq i}} x_{ijk} \leq 1 \quad \forall i \in T \quad (2.2)$$

Nous devons également nous assurer que l'agent k effectue ses indisponibilités :

$$\sum_{\substack{j \in N_k \\ j \neq i}} x_{jik} = 1 \quad \forall k \in K, i \in U_k \quad (2.3)$$

Compétences

La tâche doit être réalisée par quelqu'un qui en a les compétences. Nous avons :

$$\sum_{k \in K} \sum_{\substack{j \in N_k \\ j \neq i}} x_{ijk} m_i \leq \sum_{k \in K} \sum_{\substack{j \in N_k \\ j \neq i}} x_{ijk} q_k \quad \forall i \in T \quad (2.4)$$

Heures de travail

Les employés ne peuvent travailler que sur les horaires où ils sont disponibles. Nous avons les mêmes contraintes qu'en partie 1 pour les heures de début et de fin. Nous les réécrivons ici pour les effets de bord. Comme d'habitude, nous introduisons $H = 1440$ qui jouera le rôle d'une constante très grande par rapport aux données du problème.

Nous décidons de ne pas considérer les cas où l'employé commencerait par une indisponibilité (donc irait de sa maison à une indisponibilité), ou finirait par une indisponibilité (donc irait d'une indisponibilité à sa maison) car il peut arriver que l'employé n'ait pas le temps d'aller à ses indisponibilités sur ses heures de travail (pour une indisponibilité qui finirait à la fin de la journée et qui ne serait pas chez l'employé par exemple).

Cela nous donne les contraintes suivantes :

$$(d_k + c_{H_k i} - w_i) \leq (1 - \sum_{\substack{j \in N_k \\ j \neq i}} x_{jik}) * H \quad \forall k \in K, i \in T \quad (2.5)$$

$$(w_i - f_k + s_i + c_{iH_k}) \leq (1 - \sum_{\substack{j \in N_k \\ j \neq i}} x_{jik}) * H \quad \forall k \in K, i \in T \quad (2.6)$$

Pause Déjeuner

La pause déjeuner doit être entre 12h et 14h pour chaque employé, nous avons donc :

$$720 \leq p_k \leq 780 \quad \forall k \in K$$

Chaque employé ne peut pas travailler pendant sa pause déjeuner. Nous avons besoin de savoir si le noeud i est visité juste avant la pause déjeuner de l'employé. Nous introduisons donc des nouvelles variables de décision, $\gamma_{i,k}$: $\gamma_{i,k} = 1$ si la tâche i est la dernière tâche effectuée par l'agent k avant sa pause déjeuner, $\gamma_{i,k} = 0$ sinon, pour $i \in T \cup U$.

Pour garantir cette définition des $\gamma_{i,k}$, nous devons nous assurer que les contraintes suivantes soient respectées :

$$\sum_{k \in K} \gamma_{i,k} \leq 1 \quad \forall i \in T \cup U \quad (2.7)$$

$$\gamma_{i,k} \leq \sum_{\substack{j \in N_k \\ j \neq i}} x_{jik} \quad \forall k \in K, i \in T \cup U \quad (2.8)$$

Nous obtenons alors, pour la contrainte de la pause déjeuner, après linéarisation, :

$$(w_i + s_i - p_k) \leq (2 - \sum_{\substack{j \in N_k \\ j \neq i}} x_{jik} - \gamma_{i,k}) * H \quad \forall k \in K, i \in T \cup U \quad (2.9)$$

$$(p_k + 60 - w_i) \leq (1 + \gamma_{i,k} - \sum_{\substack{j \in N_k \\ j \neq i}} x_{jik}) * H \quad \forall k \in K, i \in T \cup U \quad (2.10)$$

Durée d'une tâche

Si l'employé k va du noeud i au noeud j, nous avons $w_i + s_i + c_{ij} \leq w_j$. Pour le cas particulier où la tâche i serait juste avant la pause déjeuner, nous devons nous assurer que $w_i + s_i + c_{ij} + 60 \leq w_j$.

D'où :

$$\left(\sum_{k \in K} x_{ijk} \right) (w_i + s_i + c_{ij} + 60 * \sum_{k \in K} \gamma_{i,k} - w_j) \leq 0 \quad \forall i, j \in T \bigcup U$$

Cette contrainte n'est pas linéaire, il faut donc la linéariser. Pour cela on introduit de nouveau $H = 1440$ qui jouera le rôle d'une constante très grande par rapport aux données du problème et nous obtenons la contrainte équivalente :

$$w_i + s_i + c_{ij} + 60 * \sum_{k \in K} \gamma_{i,k} - w_j \leq (1 - \sum_{k \in K} x_{ijk}) * H \quad \forall i, j \in T \bigcup U \text{ and } j \neq i \quad (2.11)$$

Ouverture d'une tâche

Désormais, les "tâches" peuvent être ouvertes sur plusieurs créneaux disjoints pendant la journée. Pour une indisponibilité, nous fabriquons artificiellement un créneau qui correspond exactement à son heure d'ouverture et de fermeture.

La modélisation de la contrainte d'ouverture d'une tâche est complexifiée. Pour cela nous ajoutons de nouvelles variables de décisions au problème. On note n_i le nombre de créneaux distincts de la "tâche" i. On définit ensuite β_i^r la variable binaire qui est égale à 1 si et seulement si la tâche i est effectuée au créneau r avec $r \in \llbracket 1, n_i \rrbracket$.

On construit alors de nouvelles contraintes.

Une première contrainte s'assure que, à i fixé, la somme des β_i^r est égale à 1 si et seulement si la tâche est traitée au créneau r. On relie donc cette somme à nos variables de décisions du problème (les x_{ijk}) :

$$\sum_{k \in K} \sum_{\substack{j \in N_k \\ j \neq i}} x_{ijk} = \sum_{r=1}^{n_i} \beta_i^r \quad \forall i \in T \bigcup U \quad (2.12)$$

Ensuite on veut s'assurer qu'un employé commencera la tâche uniquement après le début d'un créneau de disponibilité, si la tâche est effectuée sur ce créneau. Après linéarisation, on obtient :

$$a_i^r - w_i \leq (1 - \beta_i^r) * H \quad \forall r \in \llbracket 1, n_i \rrbracket, \forall i \in T \bigcup U \quad (2.13)$$

Enfin on veut s'assurer qu'un employé finira la tâche uniquement avant la fin d'un créneau de disponibilité, si la tâche est effectuée sur ce créneau. Après linéarisation, on obtient :

$$w_i - b_i^r + s_i \leq (1 - \beta_i^r) * H \quad \forall r \in \llbracket 1, n_i \rrbracket, \forall i \in T \bigcup U \quad (2.14)$$

2.2 Implémentation

2.2.1 Instances d'entrée

Rien ne change pour les instances d'entrée par rapport au premier modèle.

2.2.2 Constantes à récupérer

Nous listons ici les informations à récupérer depuis nos dataframes, nécessaires à l'expression des contraintes évoquées ci-dessus :

- Le vecteur de compétences M ;
- Le vecteur du niveau des agents Q ;

- Un vecteur d'ouverture des tâches et des indisponibilités O qui à chaque 'tâche' i associe une liste d'intervalles sur lesquelles la 'tâche' est ouverte, grâce aux tables *Tasks*, *TasksUnavailabilities* et *Unavailabilities* ;
- Le vecteur de durée de chaque tâche et de chaque indisponibilité S ;
- Les vecteurs d'horaires de début et de fin de travail des agents, respectivement D et F ;
- La matrice C des coûts de transports c_{ij} , entre le noeud i et le noeud j (un noeud pouvant être une maison, une tâche ou une indisponibilité) grâce aux tables *Tasks*, *Employes*, *Unavailabilities*.

2.2.3 Variables

Nous devons alors introduire via Gurobi les variables binaires x_{ijk} , les variables entières w_i , les variables entières p_k et les variables binaires de linéarisation β_i^r et γ_i .

2.2.4 Code

Il faut se référer au Jupyter Notebook correspondant pour voir le code associé.

2.3 Analyse des résultats

Nous présentons ici les temps d'exécution de notre algorithme sur les différentes instances, sur un MacBook Pro, avec un processeur 2 GHz Intel Core i5 double cœur et une mémoire 8 Go 1867 MHz LPDDR3, en comparant également les principales caractéristiques des instances en entrée.

Instances	Temps d'exécution (sec)	Nb d'employés	Indisponibilités (employés)	Nb de tâches	Indisponibilités (tâches)	Écart à l'optimalité
Bordeaux	129	2	1	10	1	0
Poland	> 3 600	2	1	19	1	0.359183
Australia	13	3	1	16	6	0
Spain	> 3 600	3	2	27	5	0.650228
Austria	> 3 600	5	3	31	5	0.394834

Sur les Instances en Autriche, Espagne et Pologne, malheureusement, notre algorithme prenait trop de temps à s'exécuter et nous n'avons pas réussi à trouver de solution 'complète'. Nous présentons ci-dessous les résultats quand nous arrêtons l'optimisation après une 1h d'exécution. Nous pensons donc que notre modèle actuel n'est pas performant quand il y a beaucoup de données. Plus précisément, comme pour la première partie, nous allons donner un ordre de grandeur du nombre de contraintes en fonction du nombre de tâches et du nombre d'employés, pour voir ce qui influe le plus sur notre modèle :

- La contrainte de débit crée $K + n_t + \text{card}(U)$ contraintes ;
- La contrainte d'attribution des tâches crée n_t contraintes ;
- La contrainte des compétences crée n_t contraintes ;
- La contrainte des heures de travail crée $2xKxn_t$ contraintes ;
- La contrainte de la pause déjeuner crée $2xKx(n_t + \text{card}(U))$ contraintes ;
- La contrainte de durée d'une tâche crée $(n_t + \text{card}(U))x(n_t + \text{card}(U))$ contraintes ;
- La contrainte d'ouverture d'une tâche crée $(n_t + \text{card}(U) + 2 * n_i * (n_t + \text{card}(U)))$ contraintes.

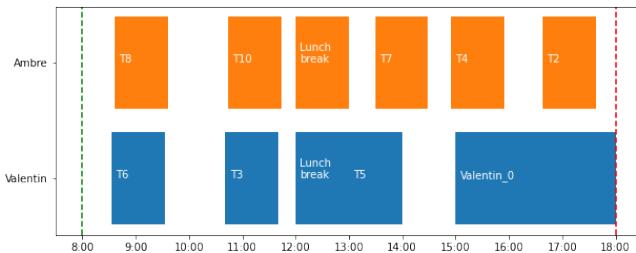
Nous avons donc finalement $K + 4n_t + 2\text{card}(U) + 4 * K * n_t + 2 * K * \text{card}(U) + (n_t + \text{card}(U))^2 + 2 * n_i * (n_t + \text{card}(U))$ contraintes, soit un nombre de contraintes en $\mathcal{O}((K + n_i) * n_t + (K + n_i) * \text{card}(U) + (n_t + \text{card}(U))^2)$. Cela nous donne, en utilisant la formule exacte, 2 281 contraintes pour l'instance Autriche ! En général, nous pouvons considérer que le nombre

de contraintes est en $\mathcal{O}((n_t + \text{card}(U))^2)$.

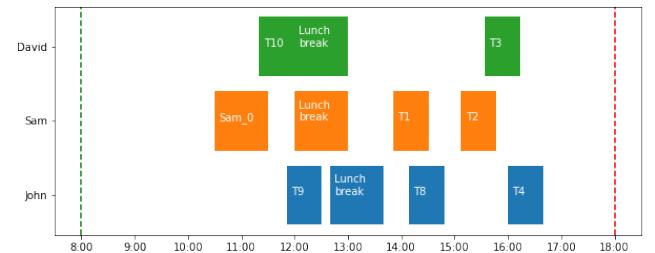
Instances	Temps d'exécution (sec)	Nombre de contraintes
Bordeaux	129	271
Poland	> 3 600	676
Australia	13	760
Spain	> 3 600	1 582
Austria	> 3 600	2 281

Nous voyons que sur l'instance Pologne, bien qu'il y ait un nombre de contraintes relativement faible, le solveur Gurobi n'arrive quand même pas à trouver la solution optimale. Au delà du nombre de contraintes, la difficulté de résolution peut donc venir également : des fenêtres de disponibilité des tâches, de celles des agents, de la durée des tâches, de la difficulté à prouver l'optimalité même lorsque le solveur trouve la solution optimale rapidement...

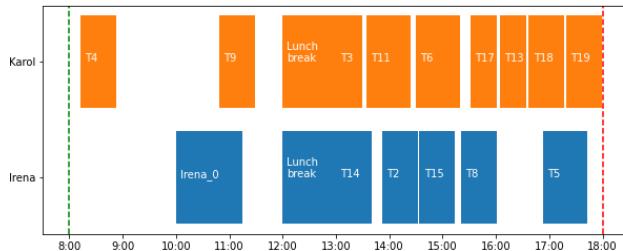
Nous mettons ci-dessous les emplois du temps des employés retournés par notre algorithme sur chaque Instance.



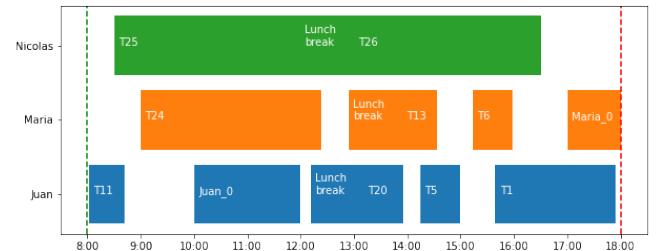
(a) Bordeaux



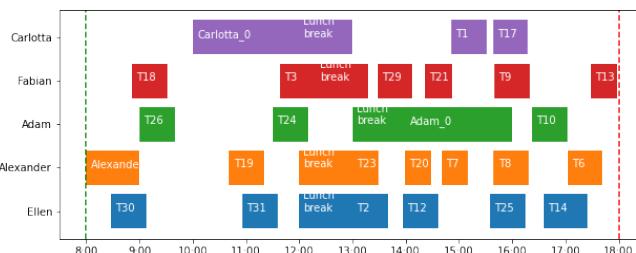
(b) Australie



(c) Pologne



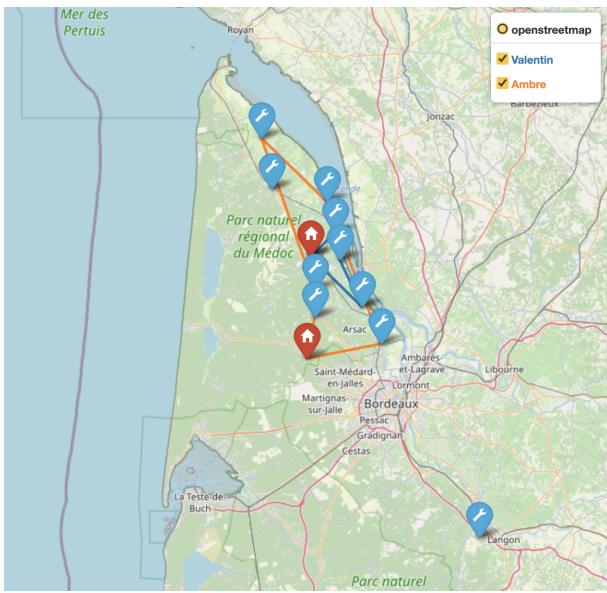
(d) Espagne



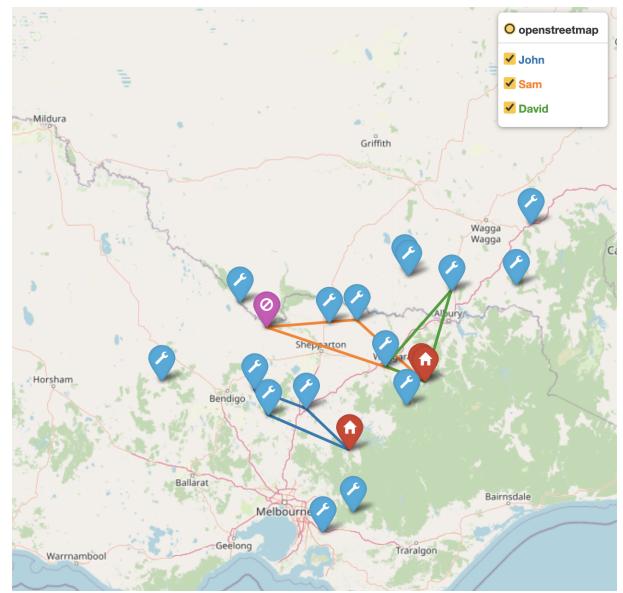
(e) Autriche

FIGURE 2.1 – Emplois du temps par instance

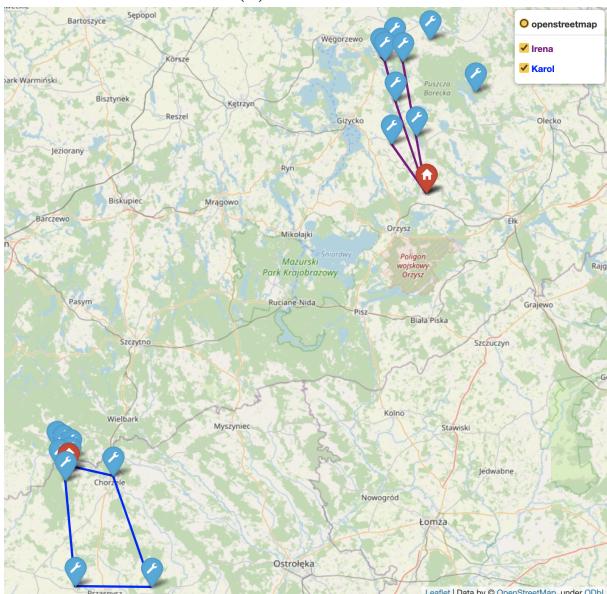
Pour les résultats cartographiques, nous mettons ici des captures d'écran mais nous conseillons de se référer aux fichiers .html sur lesquels plus d'informations sont données.



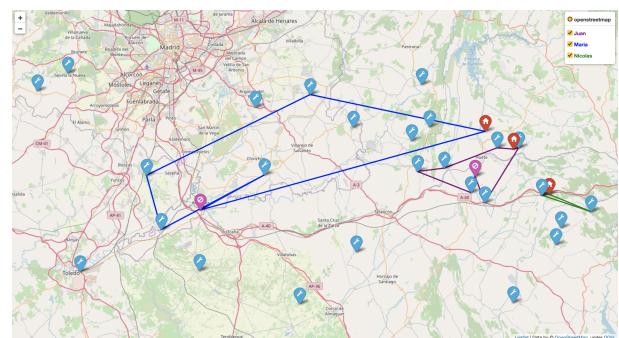
(a) Bordeaux



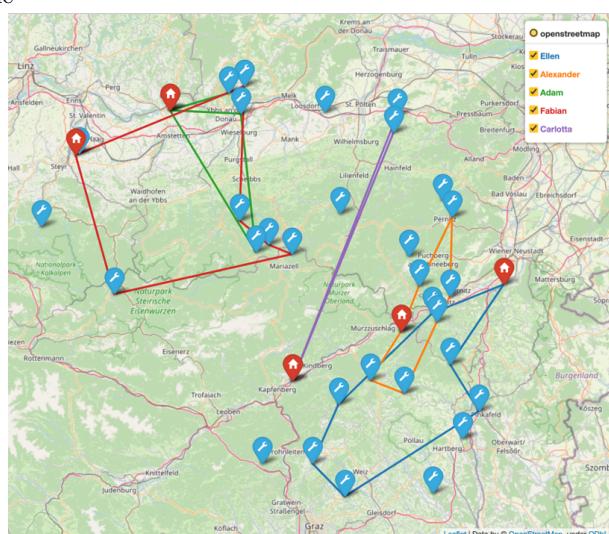
(b) Australie



(c) Pologne



(d) Espagne



(e) Autriche

FIGURE 2.2 – Résultats cartographiques

Chapitre 3

Résolution approchée

3.1 Introduction aux métaheuristiques

Que sont les métaheuristiques ? [3] Il s'agit de méthodes de recherche dédiées aux problèmes d'optimisation difficiles. Contrairement aux heuristiques qui sont souvent adaptées pour résoudre un seul type de problème, les métaheuristiques sont des méthodes générales pouvant s'adapter à un ensemble de problèmes d'optimisation. Lorsque le cardinal de l'ensemble des solutions devient élevé, et que les méthodes de recherche exhaustives ne permettent pas de trouver une solution en un temps raisonnable, les métaheuristiques peuvent être utilisées. De nature déterministe ou stochastique, elles permettent de trouver une solution proche de la solution optimale en un temps de calcul raisonnable. On peut distinguer deux familles de métaheuristiques.

La première contient les heuristiques dites à trajectoire, qui fournissent une unique solution au problème. Parmi elles, nous trouvons la méthode par recuit simulé. C'est cette méthode que nous avons tenté d'implémenter.

La deuxième famille de métaheuristiques est constituée des méthodes à populations, qui fournissent un jeu de solutions (contrairement à la première famille). Pour cette famille, nous avons fait quelques recherches sur la méthode de l'évolution différentielle. Nous nous sommes cependant rapidement rendus compte que cette métaheuristique n'était pas adaptée à notre problème.

3.2 Méthode par recuit simulé

3.2.1 Explication de la méthode et application à notre situation

En 1982, trois chercheurs de la Société IBM — S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi — ont proposé une méthode itérative s'inspirant de phénomènes physiques : la méthode du recuit simulé, directement inspirée de la méthode du recuit utilisée notamment en métallurgie.

En effet, pour la croissance d'un monocristal en métallurgie, la technique du recuit consiste en deux étapes : premièrement, chauffer un matériau à une température T pour lui conférer une énergie élevée E . Deuxièmement, refroidir lentement le matériau en marquant des paliers de température afin d'atteindre l'équilibre thermodynamique local. Cette stratégie permet de conduire à un état solide cristallisé stable qui correspond à un minimum d'énergie.

En appliquant concrètement l'algorithme à notre cas, pour une température donnée T (qui représente un nombre d'itérations), on fait subir une perturbation à notre système (le planning courant), perturbation symbolisée par ΔE . Cette perturbation correspond à l'ajout le déplacement ou la suppression d'une tâche de notre planning courant, pour un agent en particulier. Si cette transformation a pour effet de diminuer la température du système (c'est à dire $\Delta E > 0$) elle est acceptée. Sinon elle est acceptée avec une probabilité $e^{-\Delta E/T}$. La température T est ensuite diminuée par paliers réguliers.

Ainsi le terme $e^{-\Delta E/T}$ correspond à un facteur d'acceptation : lorsque la température est élevée (nombre d'itérations peu élevé), le terme $e^{-\Delta E/T}$ est proche de 1 donc la plupart des mouvements sont acceptés : l'algorithme correspond alors à une marche aléatoire dans l'espace des configurations. A basse température (nombre d'itérations élevé), le terme $e^{-\Delta E/T}$ est proche de 0 et donc seuls les mouvements permettant de diminuer l'énergie sont acceptés.

Le fait de diminuer la température régulièrement permet dans un premier temps de parcourir rapidement l'espace des configurations pour ensuite se concentrer dans une zone susceptible de contenir le maximum global.

L'algorithme s'arrête soit lorsque aucune amélioration n'est observée pendant un nombre d'itérations donné, soit lorsqu'une température minimale est atteinte. Dans notre cas il s'arrête lorsque l'on a atteint un nombre d'itérations suffisant, c'est à dire lorsque T est égal à une température finale fixée.

3.2.2 PseudoCode de la méthode du recuit simulé

Initialisation de T , T_f et du pas régulier α

Initialisation d'un planning d'affectation (vide initialement)

Tant que $T > T_f$:

1. Choix aléatoire d'une tâche à réaliser
2. Choix aléatoire d'un agent à affecter à la tâche
3. Calcul de l'ensemble des voisins possibles du planning
4. Choix aléatoire d'un voisin parmi ceux possibles et calcul de sa fonction de coût E_{voisin} . Calcul de $\Delta E = E_{planning} - E_{voisin}$
5. Choix du nouveau planning :
 - **Si** $\Delta E > 0$, **Alors** le voisin choisi devient le planning
 - **Sinon**, le voisin choisi devient le planning avec une probabilité $e^{-\Delta E/T}$
6. **Décrementer** T de α

Fin Tant que

Renvoyer le dernier planning calculé

3.2.3 PseudoCode de la fonction pour calculer les voisins d'un planning

L'algorithme prend en entrée un planning et une tâche à considérer pour la création des voisins

Initialisation d'une liste vide qui contiendra l'ensemble des voisins possibles

Pour chaque agent **Faire**

1. Calcul de toutes ses disponibilités pour faire la tâche
2. **Pour** chacune de ses disponibilités **Faire**
 - Création d'un planning voisin (planning initial auquel on a ajouté la tâche à une disponibilité)
 - **Fin pour**

Fin pour

Renvoyer la liste de tous les plannings voisins créés

3.3 Analyse des résultats

Nous présentons sur la figure 3.1 les emplois du temps obtenus pour les instances V3.

Nous présentons les temps d'exécution de notre algorithme sur les différentes instances, toujours sur un MacBook Pro, avec un processeur 2 GHz Intel Core i5 double cœur et une mémoire 8 Go 1867 MHz LPDDR3. Nous comparons les résultats obtenus sur la météahéuristiche avec 10 000 itérations et les solutions exactes.

Instances	Temps d'exécution du modèle exact (sec)	Temps d'exécution sur la météahéuristiche (sec)	Objectif du modèle exact	Objectif de la météahéuristiche
Bordeaux	129	10.47	-402	-316
Poland	> 3 600	17.73	-	-426
Australia	13	14.15	-331	-51
Spain	> 3 600	18.22	-	-295
Austria	> 3 600	23.78	-	-404
Colombie	-	50.15	-	-1267
Ukraine	-	51.65	-	-1944
Roumanie	-	86.12	-	-3275

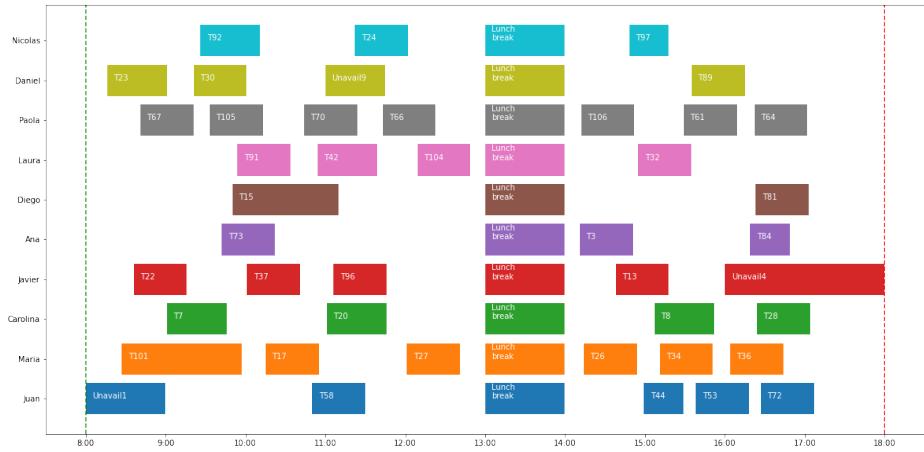
3.4 Méthode de l'évolution différentielle

Cette méthode se base sur les travaux de l'évolution de Darwin. [2] L'algorithme d'évolution différentielle consiste à maintenir une population de solutions candidates soumises à des itérations de recombinaison, d'évaluation et de sélection. L'approche de recombinaison implique la création de nouveaux composants de solutions candidats basés sur la différence pondérée entre deux membres de la population sélectionnés aléatoirement, ajoutés à un troisième membre de la population. Cela perturbe les membres de la population par rapport à la propagation de l'ensemble de la population. En conjonction avec la sélection, l'effet de perturbation auto-organise l'échantillonnage de l'espace du problème, le liant à des zones d'intérêts connues.

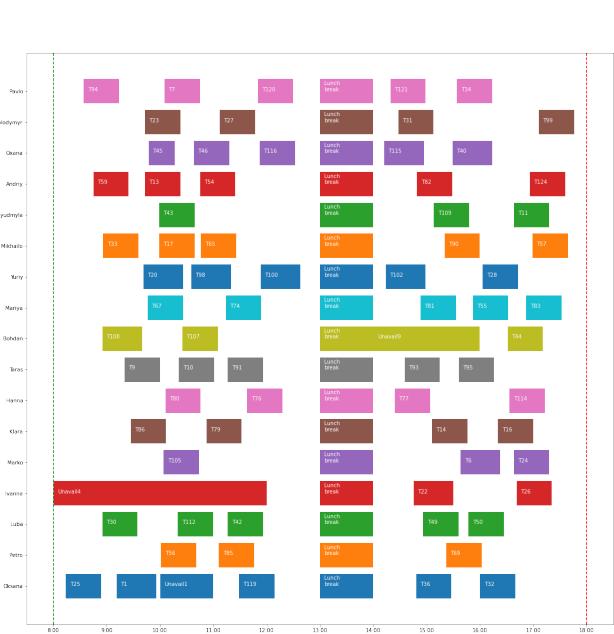
Nous avons souhaité utiliser la bibliothèque Scipy pour utiliser simplement l'algorithme de l'évolution différentielle.

Cependant, de nombreux problèmes nous ont empêchés d'utiliser cette méthode pour notre problème :

- La fonction de la bibliothèque Scipy ne peut pas résoudre pour des variables entières. Il n'est pas possible de régler ce paramètre, il faut donc arrondir les variables binaires dans la fonction objectif.
- Les variables du problème doivent toutes être dans un même vecteur d'entrée, ce qui rend l'implémentation compliquée, sachant que nous avons environ 1 000 variables par instance. Il est compliqué de vérifier les contraintes avec des variables qui n'ont pas de nom mais seulement un indice.
- Nous avons aussi remarqué qu'il est difficile de créer les différentes étapes de l'évolution différentielle. En effet, il est compliqué de choisir les méthodes pour la mutation et le crossover, puisque nos instances d'entrée ne sont pas de tailles compatibles avec celles utilisées usuellement dans la méthode. Il est difficile de choisir une méthodologie.
- Une autre complexité de ce problème consiste à observer que la méthode avec des populations risque de converger vers une solution qui n'est pas optimale. En effet, pour ce problème, les variables de temps augmentent le nombre de possibilités, si bien que les voisins ne vont pas forcément se rapprocher dans l'espace des solutions.



(a) Colombie



(b) Ukraine



(c) Roumanie

FIGURE 3.1 – Emplois du temps par instance

Conclusion

Retour sur notre objectif premier : le client Tutto Bene

Pour notre client Tutto Bene, avec 500 agents et 10 000 tâches, nous conseillons la méthode exacte étendue si le client peut se permettre d'attendre plusieurs jours pour un emploi du temps fixé, pour une semaine par exemple. S'il y a régulièrement des imprévus et que l'activité du client nécessite une actualisation régulière, nous conseillons plutôt la méthode heuristique qui est beaucoup plus rapide, malgré des résultats moins satisfaisants.

Pistes d'amélioration

Notre solution métaheuristique peut également être améliorée pour produire des résultats plus satisfaisants. Par exemple, nous listons ci-dessous les pistes auxquelles nous avons réfléchi :

- Développer un algorithme glouton permettant de trouver une meilleure initialisation au problème et ainsi de trouver une solution plus proche des standards des solveurs exacts, en un temps plus raisonnable.
- Modifier notre création de voisin dans notre métaheuristique en échangeant des tâches entre elles (et non en changeant une seule tâche de place). Faire cela permettra une exploration de plus de possibilités et donc de couvrir plus de possibilités en moins d'itérations.
- Permettre le "glissement temporel" des tâches et de la pause déjeuner. Cette fonctionnalité permettant une relaxation des contraintes imposées par la méthode de meta-heuristique actuelle tout en vérifiant les contraintes générales du problème, n'a pas encore été implémentée.
- Relier l'algorithme à l'API Google Maps afin d'avoir des informations plus précises sur les temps de trajet et ne plus seulement fonctionner à vol d'oiseau.

Remerciements

Enfin, nous tenons à remercier particulièrement les encadrants du projet, Mathieu Lerouge et Mehdi Charles, pour leur encadrement personnalisé et leur réactivité face à nos questions tout au long du projet.

Bibliographie

- [1] Jean-Francois Cordeau. *The VRP with time windows*. Groupe d'études et de recherche en analyse des décisions Montréal, 2000.
- [2] Oussama El Gerari. Contribution à l'amélioration des techniques de la programmation génétique. Université du Littoral Côte d'Opale, 2011.
- [3] Patrick Siarry. *Métaheuristiques*. Eyrolles, 2014.