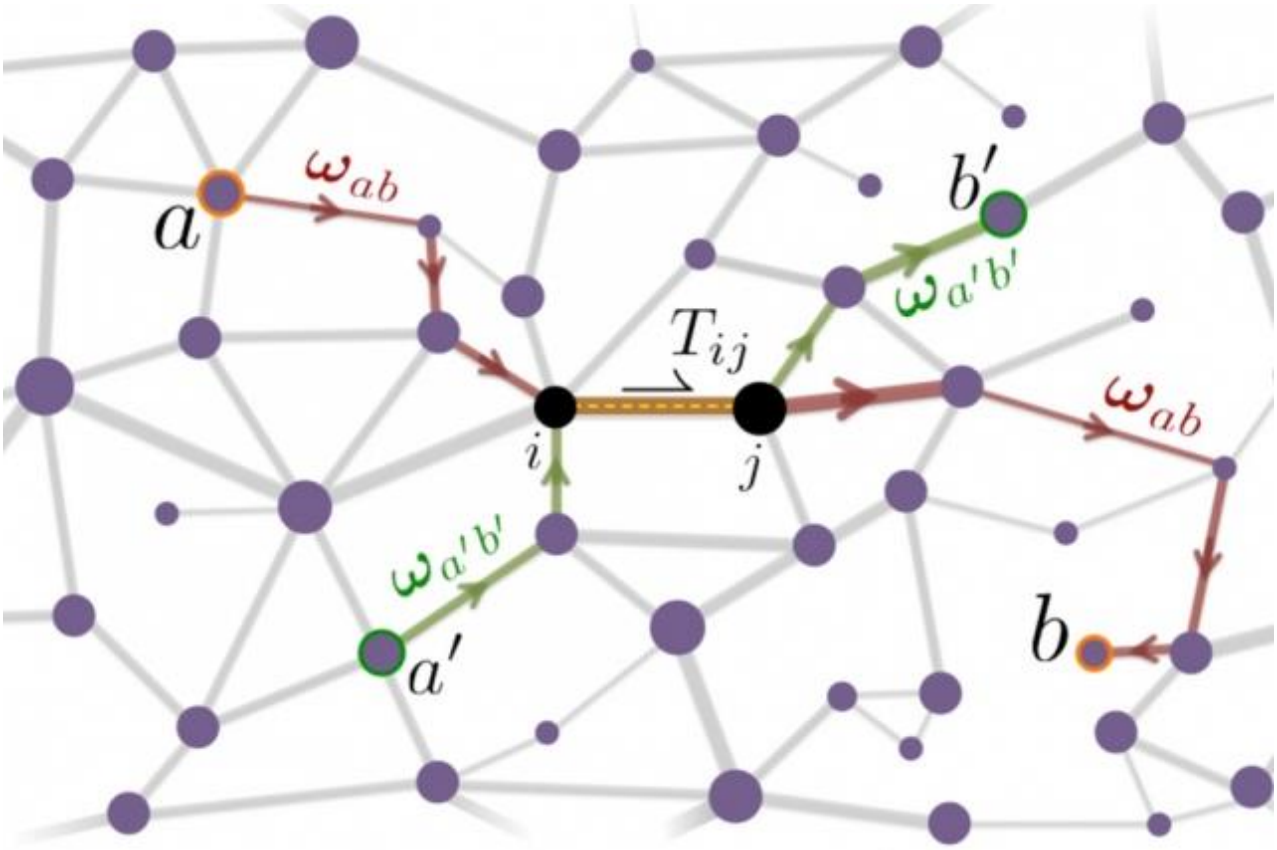

RAPPORT DE PROJET

Tournées d'accompagnants

NEIL FARMER & BRANDON JEDD ARRIUS JIPPIU
UTBM

Table des matières

1) Contexte

1.1) Rappel du problème

L'objectif du projet est de trouver une « bonne » solution d'attribution des interfaces à des missions pour un jeu de données. Cette solution va évidemment dépendre du trajet parcouru par les interfaces (qu'il faudra minimiser) mais devra prendre en compte aussi d'autres paramètres, comme :

- leur temps de pause du déjeuner,
- ou si la spécialité de l'interface est la même que celle de la formation, etc.

Ce problème est un problème d'affectation.

Le scénario impose que l'interface se rende dans un premier temps chez l'apprenant, puis qu'ils se déplacent ensemble jusqu'au lieu de la formation, sur lequel l'intervenant réalise sa prestation. A la fin de la formation, l'interface raccompagne l'apprenant chez lui. Enfin, pour des raisons de simplification du modèle, on considéra que l'interface rentre à son domicile.

1.2) Solution envisagée

Une première solution envisagée est de venir séparer le problème en deux types de contraintes :

- d'une part celles qui doivent être respectées pour des raisons physiques (une interface ne peut pas avoir deux missions en même temps) ou pour des raisons légales (une interface ne peut pas travailler plus de 48h sur une même semaine en France),
- d'autre part, les contraintes qu'il serait préférable de respecter mais sans obligation. On viendra tester chaque catégorie de contraintes dans une fonction différente.

Le choix de la méta-heuristique se porte sur la recherche tabou qui a l'avantage de converger très vite, d'être peu coûteux en temps et en mémoire. Il faut donc aussi choisir un algorithme pour développer la solution courante. On va choisir l'algorithme des plus proches voisins qui permettra de donner une solution acceptable. Cependant, on ne peut apporter aucune garantie quant aux solutions que peut donner l'algorithme des plus proches voisins et, dans le cas moyen, la solution qui en ressortira sera tout au plus correcte, mais cela permettra d'avoir une solution quasi constante à la fin de la recherche tabou.

2) Implémentation du problème

2.1) Distance et solution

Le jeu de données nous donnant uniquement des coordonnées, la première étape, bien que triviale, est de calculer la matrice des solutions. A noter cependant que tous les coefficients ne seront pas intéressants. Seuls seront retenus les coefficients allant du domicile d'une interface au domicile d'un apprenant et celui du domicile de l'apprenant à son lieu de formation. Ces distances nous intéresseront plus tard pour des raisons différentes.

La solution est entrée dans un tableau à 1 dimension, où chaque « case » correspond à l'affectation d'une mission à une interface. On implémente aussi une fonction pour lire ces solutions qui viendra nous indiquer la distance entre l'interface et l'apprenant et si la spécialité de l'interface correspond à celle de la formation.

2.2) Fonction objectif

La fonction sera évidemment une des composantes les plus importantes de la bonne résolution ou non du problème. Cette fonction sera au moins aussi importante que la recherche tabou. Une « bonne » fonction objectif fera en sorte que toutes les contraintes soient respectées en fonction de l'importance qu'on leur porte. Chaque contrainte pourra plus ou moins pénaliser la solution trouvée.

On va se poser plusieurs questions sur cette fonction objectif. Que va-t-elle évaluer au départ ? Que va-t-on pénaliser et comment ?

Tout d'abord, on fera une première évaluation de la solution qui ne sera rien de plus que le calcul de la distance entre le domicile de l'interface et celui de l'apprenant. En effet, ce sera la seule distance de tout le parcours de l'interface qui va venir varier, car peu importe l'attribution de la mission, la distance entre l'interface et le lieu d'apprentissage sera toujours strictement la même. Evidemment, cette seule évaluation n'est pas très, voire pas du tout, intéressante. Elle permet uniquement de donner la base nécessaire à l'implémentation de la vraie fonction d'évaluation.

La première fonction ne prenant pas encore en compte les contraintes obligatoires, on crée une fonction qui teste une par une ces contraintes. Si toutes sont respectées, la fonction renvoie 1, sinon elle renvoie un code d'erreur (qui est un nombre négatif) qui peut permettre de connaitre l'erreur rencontrée. Bien que peu d'utilité à première vue, cela peut révéler des problèmes lors de la conception ou le problème d'un jeu de données qui ne serait pas réalisable. Cette fonction doit réaliser tout test suivant :

- L'interface dispose de la compétence requise (Codage LPC ou Langue des signes)
- Moins de 10h de travail par jour[1]
- Moins de 48h de travail en une semaine
- Aucune mission au même moment
- Une amplitude 12h maximum en une journée

[1] Source : https://www.service-public.fr/particuliers/vosdroits/F1911

Du fait de ce scenario, on prend en compte le temps de trajet avant et après la mission, et aucune autre mission ne peut se trouver au cours de cette période. Il faut aussi remarquer que prenant en compte le temps de trajet, une mission peut tout à fait commencer un jour et finir le lendemain dû à un long trajet. Le calcul étant différent pour calculer l'heure de départ et l'heure d'arrivée, on se retrouve avec 4 fonctions pour le trajet (2 pour les heures, 2 pour les jours). De plus, les distances n'ayant pas d'unité, on leur attribue un temps (en minutes) par unité de distance pour avoir une estimation du temps de trajet. On viendra ajouter (ou soustraire pour l'heure de départ) ce temps de trajet à l'heure de fin (ou début) de formation, qu'on arrondie à l'entier le plus grand (ou le plus petit). En effet, les heures du jeu de données étant en « int », le plus simple est de rester en entier aussi pour les comparaisons, et les manipulations en général.
Cependant on ne prend pas en compte le temps de trajet dans les heures de travail ou dans l'amplitude.

Le test pour la compétence regarde la compétence requise pour la mission, puis le test la compare à celle de l'interface.

Le calcul du temps de travail ne prend donc en compte que les heures de la formation.

Cette fonction sera appelée par la fonction objectif qui retournera l'évaluation. Si la fonction objectif observe une solution impossible, elle renvoie aussi « -1 » pour indiquer que la solution n'est pas valide.

On va maintenant réfléchir comment implémenter les pénalités. La pénalité aura pour but de chercher d'autres solutions viables qui respectent les conditions. Ces pénalités sont la distance supplémentaire que l'on accepte qu'une interface réalise pour remplacer l'interface à laquelle on a attribué temporairement la mission, à condition que la nouvelle interface respecte cette contrainte. C'est-à-dire que si on a une pénalité en dur de 200 unités pour le non-respect de la pause le midi, cela signifie que l'on est prêt à attribuer la mission à une autre interface dans un rayon de 200 unités à condition que celle-ci ait une pause le midi.

Une autre question qui peut se poser est de savoir si on doit avoir une pénalité en pourcentage du trajet réalisé ou en une distance qui sera la même pour tous. Le choix sera fait de prendre en pourcentage de la distance, pour ne pas sur-pénaliser les petites distances et sous-pénaliser les grandes distances.

Les pénalités vont porter sur 4 points :

- La spécialité
- La pause le midi
- L'homogénéité de l'attribution
- Les heures supplémentaires
- La répartition des distances
- La répartition du temps de travail.

3) Recherche tabou

3.1) Rappel sur la recherche tabou

La recherche tabou est une méthode de recherche d'optimal approché. Son principe fondateur est assez simple, on recherche dans le voisinage de la solution, une solution qui minimise la fonction d'évaluation. Chaque mouvement est mémorisé dans une liste FIFO (appelée liste Tabou), qui vient interdire pendant un certain nombre d'itérations ce mouvement. Cela permet notamment de sortir d'un minimum local, car on peut dégrader la solution.

Cette méta-heuristique a l'avantage de converger rapidement et de tendre vers une « bonne » solution. Cependant, rien ne peut garantir l'optimalité de la solution. Il y a même de très grandes chances que la solution trouvée ne soit pas optimale.

3.2) Solution courante

Ayant pour fonction de recherche une méta-heuristique, il faut une solution de départ (appelée solution courante). On va implémenter la méthode des plus proches voisins. Cette méthode cherche pour chaque formation l'interface la plus proche qui permet une solution valide. Cette solution n'est pas la plus pertinente, mais elle peut aussi peut être très proche d'une bonne solution.

3.3) Implémentation de la recherche tabou

On va implémenter 3 fonctions :
- L'une qui va chercher la meilleure solution dans le voisinage,
- L'une pour diversifier la solution,
- et l'une qui va gérer la liste tabou, les itérations, …

La première fonction va donc venir gérer la liste tabou, les itérations (via une boucle while). Mais on lui implémente aussi un critère de diversification. Ce critère se gère par une 2^{e} fonction.

On va aussi ajouter un critère de diversification. On sait qu'il possible que la recherche tabou n'explore qu'une partie de l'espace de recherche. L'objectif est donc de venir diversifier l'espace de recherche. Il existe énormément de méthodes de diversification. On peut par exemple citer la « diversification radicale » qui force quelques éléments peu fréquents dans la solution. On peut également citer la diversification continue qui dégrade les éléments très fréquents dans la solution. Nous allons passer par une des méthodes les plus basiques. Au bout d'un certain nombre d'itérations sans amélioration, la solution, sera modifiée aléatoirement par une autre solution valide. Aussi, pendant l'élaboration de l'algorithme, nous avons testé 2 méthodes, la première consistait à prendre aléatoirement des éléments de solutions précédentes améliorantes. Cela avait l'avantage que le très bon élément était forcément présent car surreprésenté dans les solutions améliorantes, mais avait comme inconvénient le fait de se placer dans un espace très proche ou se mettre sur une solution déjà visitée.

Enfin la dernière fonction est la fonction de voisinage, qui évalue les solutions pour chaque interface pour chacune des formations à partir de la dernière solution. Si une solution est valide, on va alors faire une évaluation de cette solution. Si celle-ci est la meilleure évaluation de la fonction de

voisinage, on viendra la retourner à la recherche tabou. La fonction de recherche tabou devra ensuite comparer l'évaluation de cette nouvelle solution avec la meilleure évaluation rencontrée. Si la nouvelle évaluation est meilleure, alors on garde la solution en mémoire. La fonction de recherche tabou devra aussi ajouter le mouvement réalisé dans la liste tabou pour « bannir » ce mouvement pour un nombre d'itérations précédemment choisi.

Plusieurs fonctions de voisinage ont été pensée. Une fonction de voisinage qui venait échanger 2 attributions de mission pour des interfaces données. Cependant le problème était que si une interface n'avait pas de mission dans la solution courante, cette interface ne pouvait donc pas être intégrer dans la solution.

On va aussi ajouter un critère d'aspiration à notre algorithme, qui peut s'avérer efficace pour trouver des minimums. Le principe du critère d'aspiration est que si on trouve une solution avec une évaluation strictement inférieure à la meilleure solution trouvée jusque-là, même si le mouvement est tabou, on réalise le mouvement. Cela nous permet d'éviter de ne pas trouver un minimum local plus intéressant, voire l'optimum, sans s'y rendre.

3.4) Les paramètres

Plusieurs paramètres sont modifiables pour faire varier les résultats comme par exemple la pénalité associée à chaque contrainte. Mais aussi les paramètres de la recherche tabou, c'est-à-dire la mémoire tabou, le nombre d'itérations, le nombre d'itérations sans amélioration avant une diversification de la solution. Enfin, le temps en minute par unité de distance peut aussi être modifié.

3.5) Les performances

Dû aux nombreux tests sur la validité de la solution et au calcul des pénalités, il est assez difficile d'effectuer un très grand nombre d'itérations. Cependant, cela ne pose que peu de problèmes, car l'algorithme semble converger très rapidement. En moins d'une dizaine d'itérations et moins d'une quarantaine en doublant de taille le jeu de données, cela dépend évidemment des paramètres testés, et notamment la longueur de la liste tabou se révèlent être un paramètre très important sur le nombre d'itérations avant de converger et la solution trouvée. A noter aussi que le critère de convergence permet parfois d'améliorer la solution, cependant il est plus prudent dans un premier temps de trouver à partir de quelle itération on converge avec une longueur de liste tabou qui semble optimale empiriquement. Une fois ces paramètres trouvés, on peut essayer d'ajuster le critère de diversification. Cela permettra d'éviter de chercher des solutions trop longtemps alors que l'algorithme a déjà convergé (si le nombre d'itérations sans amélioration est trop haut) ou de passer à côté de minimums locaux intéressants (si le nombre d'itérations sans amélioration est trop bas).

4) Amélioration possible

On aurait pu imaginer ajouter un critère d'intensification qui poursuit des recherches dans un espace qui pourrait contenir de meilleures solutions.
On aurait aussi pu rajouter des contraintes afin de coller davantage à la réalité et de tendre vers un résultat qui aurait pu être encore « meilleur ».
Enfin, on a vu précédemment que le programme n'était peut-être pas assez rapide et optimisé, on aurait pu réfléchir plus longuement à la façon de le rendre plus performant en temps.