Rapport d'Algorithmes collaboratifs et applications

Localisation Robot, Colonie de Fourmi

‘ Application-Solver

Bus Allocation Problem

Élève de B1a :

Jinhan ZHOU

Yuexuan KONG

Chargé de TD :

Alexandre SAIDI

Date du Rapport : 27 Février 2021

Année 2020-2021

Table des matières

1 Introduction **2**

2 Modélisation du problème **2**
2.1 Algorithme *ACO* adapté 2
2.2 Modélisation des paramètres supplémentaires 2

3 Accélération du calcul **3**
3.1 Calcul Matriciel 3
3.2 Algorithme Génétique 3

4 Design d'interface algorithme-client **4**

5 Conclusion **4**

1 Introduction

Le problème $BAP(Bus\ Allocation\ Problem)$ s'agit d'une optimisation de chemins. Nous devons concevoir une allocation de bus en considérant les conditions réelles, y compris la fluidité, le temps d'attente, les correspondances, le nombre de passagers, la capacité des bus, etc. Les algorithmes comme algorithme ACO et algorithme génétique peuvent nous donneront une idée de base sur notre modélisation. Et l'utilisation pratique nous demandera aussi de considérer la vitesse de calcul et l'interface de communication avec les utilisateurs. Nous vous les précisons au fur et à mesure.

2 Modélisation du problème

2.1 Algorithme ACO adapté

Le problème se distingue de le problème à l'origine de l'algorithme $ACO(Ants\ Colonies\ Optimisation)$. Il nous permet de choisir une arrête centrale qui peut faire passer plusieurs de chemins de bus. Cela implique que nous pouvons encore nous profiter de l'algorithme ACO en ajoutant des petites modifications.

L'algorithme ACO s'occupe d'une optimisation du chemin passant tous les points une fois et seulement une fois. Son principe imite simplement l'intelligence de la colonie de fourmis : Des fourmis cherchent leurs propre chemins par eux-même. Chaque fois quand les fourmis choisissent un chemin, ce chemin va marqué par leur *pheromene*. Les fourmis préfèrent de choisir les chemins avec plus de *pheromene* d'autrui. Les chemins plus longs font les fourmis prendre plus de temps à parcourir et alors avec les répétitions, les chemins plus courts vont marqué plus de *pheromene*.

L'algorithme utilise l'inverse du longueur de chemin pour présenter l'actualisation de *pheromene*. Et des objets $fourmis$ définies de même façon et grouper les efforts de chacuns par l'actualisation multilatérale de leurs *pheromene*.

Cependant, pour notre problème, il faut ajouter l'effet d'arrête centrale. Le choix d'arrête centrale est simplement : l'arrête qui possède une somme minimale des distances avec d'autres arrêtes. Ce choix correspond à la fois la minimisation de distance globale et la réalité des centres commerciaux. Et pour adapter à la particularité de l'arrête centrale, nous pouvons configurer à nouveau la définition de la distance entre les arrêtes i et j, D_{ij}, par la formule en dessous :

$$D_{ij} = min(d_{2cj}, d_{2ij}) \tag{1}$$

Où, l'arrête c est l'arrête central et d_2 est la distance euclidienne. Cette définition nous permet de faire le problème BAP revenir au problème ACO, puisque nous planifier la transition vers l'arrête prochaine soit avec l'arrête dernière, soit avec l'arrête centrale, d'après le principe de minimisation.

2.2 Modélisation des paramètres supplémentaires

La particularité du problème BAP se présente aussi sur la considération complémentaire incluant la fluidité, le temps d'attente, les correspondances, le nombre de passagers, la capacité des bus, etc. Il s'agit d'une convention concrète. Nous notons alors ces symboles :

v - la fluidité

f - la fréquence de bus

τ - le temps d'attente

N^c - la capacité des bus

N^{up} - le nombre de passagers à monter

N^{down} - le nombre de passagers à descendre

Où, ces quatre premiers paramètres dépendent des conditions des chemins de bus et ces deux derniers dépendent des conditions des arrêtes à partir. Et alors, la distance peut se transformer en coûts temporels T_{ij} par la formule en dessous :

$$T_{ij} = \frac{D_{ij}}{v_{ij}} + \frac{P_{ij}^{attente} + 0.5}{f_{ij}} \tag{2}$$

Où, $P_{ij}^{attente}$ est le nombre moyen de bus à attente et 0.5 modélise la probabilité à attendre le premier bus arrivé. Il y a totalement N_i^{up} passagers à monter et $N_{ij}^{available} = N_i^{down} + N_{ij}^c$ passagers à descendre. Chaque $N_{ij}^{available}$ passagers peuvent prendre le même bus, qui implique que les premiers $N_{ij}^{available} = N_i^{down} + N_{ij}^c$ passagers prennent le premier bus arrivé, ces deuxièmes attendent une intervalle $t_{ij} = \frac{1}{f_{ij}}$ pour le deuxième bus arrivé, etc. Et alors nous remarque que nous pouvons séparer le nombre de passagers sous la forme $N_i^{up} = N_{ij}^{available} N_{ij}^a + N_{ij}^b$, où $0 \leq N_{ij}^b < N_{ij}^{available}$. Nous obtenons alors la formule pour $P_{ij}^{attente}$:

$$P_{ij}^{attente} = \frac{\frac{N_{ij}^a (N_{ij}^a - 1)}{2} + N_{ij}^b}{N_i^{up}} \tag{3}$$

Cette formule peut bien décrire l'influence de chaque paramètre. Nous ne nions pas les inconvenants de ce modèle, mais aucun modèle est strictement correct. Ce modèle peut déjà correspond à l'utilisation pratique.

Avec cette formule, les paramètres revient alors au système d'algorithme *ACO*.

3 Accélération du calcul

3.1 Calcul Matriciel

Pour adapter à l'utilisation pratique, il faut faire attention aux défauts de l'algorithme *ACO*. Pour améliorer la vitesse de calcul, le calcul matriciel est une bonne solution. Le module *numpy* de *Python* ou *Matlab* exécutent l'algorithme une ligne après une autre. Le calcul matriciel, en revanche, rend ces étapes en parallèle dans une étape. Et alors, nous préférons écrire les itérations par le calcul matriciel.

3.2 Algorithme Génétique

La profondeur de l'algorithme *ACO* est un compromis de la vitesse du calcul et de la précision. Pour passer cet obstacle, nous devons penser à améliorer le processus de l'algorithme *ACO*.

L'algorithme génétique est une stratégie courante aujourd'hui. L'idée principale est la transfert de deux objets parents vers un enfants. Cette méthode peut bien réduire le nombre d'itérations et éviter le minimum local.

Pour l'algorithme *ACO* adaptée, nous laissons l'enfant à reprendre un certain nombre de chemins inférieur à un moitié de nombre d'arrêtes. Et l'enfant prend une autre stratégie et apprendre les autre arrêtes à partir de la dernière arrête apprise. Les arrêtes non-découvertes reste à découvrir par l'algorithme *ACO* adaptée.

4 Design d'interface algorithme-client

La conception d'interface algorithme-client est aussi un point technique important pour transformer l'algorithme en une application.

Nous utilisons le module *tkinter* pour desiger le *GUI*. En dehors des applications basiques, y compris l'ajout des arrêtes, la suppressions des arrêtes et calcul, il faut encore donner la possibilité pour définir les arrêtes et les chemins. Pour une convention, les définitions des arrêtes marques à côté des arrêtes et celles de chemins définis à gauche ou à droite au milieu de chemin : à gauche si l'arrête à partir est plus haut que l'arrête à arriver. Nous pouvons configurer à nouveau aussi la distance de chemin pour adapter la différence de distances aller-retour.

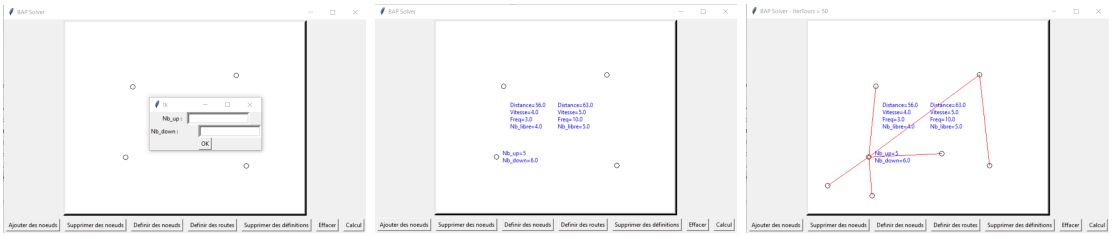

FIGURE 4.1 – Illustration de *GUI*

5 Conclusion

Les algorithmes *ACO* et génétique sont considérablement puissants pour les problèmes d'optimisation du routage. Le problème *BAP* est bon exemple : Nous devons redéfinir certaines notations pour faire le problème revenir au problème à l'origine, par exemple, en changeant la définition de distance géométrique et de coût temporel de chemins. D'ailleurs, les améliorations des algorithmes pour la vitesse de calcul et aussi de l'interface algorithme-client nous permettent d'envelopper notre algorithme plus acceptable pour les vrais utilisateurs.