AI09 - Team Orienteering Problem

BENYAGOUB Imene, BOUZAR Massil

1 Introduction

du joueur i pour le profil de stratégies mixtes $\mathbf{s}$ est :

$$u_i(\mathbf{s}) = \sum_{a \in A} u_i(a) \cdot \prod_{j \in N} s_j(a_j)$$

Le Team Orienteering Problem (TOP) représente un défi complexe en optimisation combinatoire, dérivant du problème de tournées de véhicules. Il s'agit de maximiser le profit d'une tournée, chaque véhicule desservant un certain nombre de clients sans excéder un temps de trajet fixé. Le TOP présente des applications pratiques dans divers domaines tels que la logistique, le tourisme et la gestion des ressources.

2 Formalisation du problème

Nous considérons une flotte F de m véhicules, chacun ayant un temps de parcours limite L. L'ensemble des clients est représenté par un graphe complet $G = (V, E)$ où $V = \{1, \ldots, n\} \cup \{d, a\}$, avec d et a représentant respectivement les points de départ et d'arrivée de chaque véhicule.

Chaque client i est associé à un profit P_i, récoltable au plus une fois par un véhicule de la flotte. Le temps de trajet C_{ij} est défini pour chaque arc (i, j) de E, et il est pris en compte dans la limite de temps de parcours du véhicule empruntant cet arc.

Il n'est pas obligatoire de visiter tous les clients. L'objectif est d'attribuer un itinéraire à chaque véhicule de la flotte afin de maximiser le profit total.

3 Formulation linéaire en nombres entiers

Nous présentons la formulation linéaire en nombres entiers suivante, formalisée par Racha El-Hajj, Duc-Cuong Dang et Aziz Moukrim [El-Hajj et al., 2013].

Dans la suite, V^-, V_d, V_a et V désignent respectivement les ensembles $\{1, \ldots, n\}$, $V^- \cup \{d\}$, $V^- \cup \{a\}$ et $V^- \cup \{d, a\}$. De plus, les variables de décision y_{ir} et x_{ijr} sont définies comme suit : $y_{ir} = 1$ si le client i est servi par le véhicule r et 0 sinon ; $x_{ijr} = 1$ si l'arc (i, j) est utilisé par le véhicule r pour visiter le client i puis le client j, et 0 sinon.

1. Maximiser la somme des profits effectivement collectés :

$$\max \sum_{i \in V^-} \sum_{r \in F} y_{ir} \cdot P_i \quad (1)$$

2. Chaque client est servi au plus par un véhicule :

$$\sum_{r \in F} y_{ir} \leq 1, \quad \forall i \in V^- \quad (2)$$

3. L'ensemble des tournées est connecté :

$$\sum_{j \in V_a} x_{djr} = \sum_{j \in V_d} x_{jar} = 1 \quad (3)$$

$$\sum_{\substack{j \in V_a \\ j \neq k}} x_{kir} = \sum_{\substack{j \in V_d \\ j \neq k}} x_{jkr} = y_{kr} \quad \forall k \in V^-, \forall r \in F \quad (4)$$

4. Le temps de trajet est limité pour chaque véhicule :

$$\sum_{i \in V_d} \sum_{\substack{j \in V_a \\ j \neq i}} C_{ij} x_{ijr} \leq L \quad \forall r \in F \quad (5)$$

5. Il ne peut pas y avoir de sous tours :

$$\sum_{(i,j) \in U \times U} x_{ijr} \leq |U| - 1, \quad \forall U \subseteq V^-, |U| \geq 2, \quad \forall r \in F \quad (6)$$

6. Contraintes d'intégrité :

$$x_{ijr} \in \{0, 1\}, \quad \forall i \in V, \forall j \in V, \forall r \in F \quad (7)$$

$$y_{ir} \in \{0, 1\}, \quad \forall i \in V^-, \quad \forall r \in F \quad (8)$$

4 Choix d'heuristique

Dans le cadre des Problèmes de Tournées de Véhicules (PTV ou VRP), trois heuristiques différentes ont été présentées en cours :

1. Clarke and Wright :
   — Initialisation : n tours d'un seul client
   — Itération : Tester toutes les fusions de paires de tournées
   — Effectuer la fusion de gain maximal si elle existe, sinon l'algorithme termine
   Avantages : Minimise le coût et le nombre de véhicules utilisés

2. Gillet et Miller :
   — Trier les clients par angle polaire croissant par rapport au dépôt
   — A partir d'un client de départ, établir des tournées en utilisant l'heuristique PPV (compatibles avec la capacité des véhicules)
   — Améliorer les tournée avec 2-opt
   Inconvénient : Bons résultats lorsque dépôt central

3. Beasley :
   — Définir un "tour géant" pour tous les clients en utilisant une heuristique quelconque du PVC
   — Découper ce tour en tournées en suivant l'algorithme fourni

Pour les instances étudiées :
— Il n'y a pas systématiquement de points de départ/arrivée centraux par rapport aux clients
— Le service de tous les clients n'est pas obligatoire (définition du TOP)

Ainsi, nous choisissons d'utiliser l'heuristique de Clarke and Wright plutôt que celles de Gillet et Miller ou Beasley. Cela relève d'un choix personnel, ces heuristiques auraient également pu être adaptées au problème du TOP.

5 Implémentation

5.1 Instances

Nous utilisons dans notre implémentations les instances proposées par Chao [Chao and Golden, 1993][Chao et al., 1996] et Tsiligirides [Tsiligiridis, 1984].

5.2 Adaptation de l'algorithme Clarke and Wright pour le TOP

Pour implémenter la solution Clarke and Wright, nous nous basons sur un algorithme proposé par Javier Panadero et Angel A. Juan en 2020 [Panadero et al., 2019].

Algorithme Le fonctionnement général de l'algorithme est le suivant :

```
clarke_and_wright_TOP :

    // générer n routes composées du point de départ, d'un client et du point d'arrivée
    solution := generer_routes_initiales(instance)

    // heuristique, liste de gains décroissants
    liste_gains := générer_liste_gains(instance, alpha)

    // boucle d'exécution
    Tant_que liste_gains non vide :

        // choisir le premier arc de la liste de gains (connecte deux routes)
        arc = selectionner_prochain_arc(liste_gains)

        // retrouver les deux routes
        iIndex, iRoute = trouver_iRoute(arc)
        jIndex, jRoute = trouver_jRoute(arc)

        // fusionner les deux routes
        nouvelle_route = fusionner_routes(iRoute, jRoute)
        nouveau_temps_trajet = calculer_temps_trajet(nouvelle_route)

        // s'assurer que la fusion est valide (temps ne dépasse pas tmax)
        Si fusion_valide(nouveau_temps_trajet, tmax) est vrai :
            solution := maj_solution(nouvelle_route, iRoute, jRoute, solution)
        Fin_si

        supprimer_arc_liste_gains(arc)

    Fin_tant_que

    // retenir les m routes au profit le plus haut (m nombre de véhicules disponibles)
    solution = classer_routes_par_profit(solution)
    solution = supprimer_routes(solution, m)

    retourner solution
```

Heuristique L'heuristique utilisée pour la fusion des nœuds résulte d'une adaptation des gains de l'algorithme de Clarke et Wright. La formule de calcul de gain est donnée comme suit :

$$s'_{ij} = \alpha \cdot s_{ij} + (1 - \alpha)(u_i + u_j)$$

Le terme s_{ij} représente le gain original de l'algorithme de Clarke et Wright pour fusionner les clients i et j.

Le terme (u_i+u_j) quant à lui, tient compte des profits individuels des clients i et j. Plus précisément, $(u_i + u_j)$ représente la somme des profits associés à ces clients.

Le paramètre α est défini de manière à représenter l'hétérogénéité des profits des différents clients de l'instance. Plus l'hétérogénéinité est élevée, plus α est proche de zéro. Ainsi, en ajustant α, on peut influencer la façon dont l'algorithme priorise la réduction des trajets par rapport à la maximisation du profit global. Lorsque α est proche de zéro, l'accent est davantage mis sur les profits individuels des clients, favorisant potentiellement ceux avec des profits plus élevés. En revanche, avec α proche de un, l'accent est davantage mis sur la réduction des trajets sans prendre en compte autant les différences de profits entre les clients.

Complexité La complexité de l'algorithme proposé est de l'ordre de $O(n^3)$: la boucle *Tant_ que* itère sur une liste de dimension n^2 et effectue au pire des opérations de l'ordre de n (exemple : *trouver_ iRoute*).

5.3 Amélioration 2-opt

Nous avons choisi d'apporter une amélioration à l'algorithme présenté en y ajoutant l'algorithme 2-opt. A chaque itération de la boucle *Tant_ que*, si la fusion des deux routes sélectionnées est valide, l'algorithme 2-opt est appelé sur la nouvelle route. Cela permet de réarranger l'ordre des clients au sein de cette route et minimiser le temps de trajet. Ainsi, les possibilités de fusion pour les prochains tours sont maximisées et les profits potentiels également.

Algorithme

```
clarke_and_wright_TOP_2-opt :

    // ... algorithme existant

    // boucle d'exécution
    Tant_que liste_gains non vide :

        // ... algorithme existant

        // s'assurer que la fusion est valide
        Si fusion_valide(nouveau_temps_trajet, tmax) est vrai :
            2-opt_route = 2_opt(nouvelle_route)
            solution := maj_solution(2-opt_route, iRoute, jRoute, solution)
        Fin_si

        supprimer_arc_liste_gains(arc)

    Fin_tant_que

    // ... algorithme existant

    retourner solution
```

Complexité L'ajout de 2-opt augmente la complexité de l'algorithme original. Dans le pire des cas, 2-opt est de complexité $O(n^2)$, ce qui résulte en un algorithme de Clarke and Wright pour le TOP de complexité $O(n^4)$

Résultats Le fichier *final_ output_ results.csv* compare les résultats entre les profits obtenus sans et avec 2-opt. Nous pouvons remarquer que dans une grande partie des cas, une amélioration du profit total est permis par l'ajout de 2-opt. Un extrait des améliorations effectives est donné dans le tableau suivant :

Instance	Résultat (profit)	Résultat 2-opt
p6.2.h.txt	1128	1284
p6.2.i.txt	1176	1386
p6.2.j.txt	1536	1548
p6.2.k.txt	1572	1578
p6.2.l.txt	1572	1704
p6.2.m.txt	1572	2376
p6.2.n.txt	1806	2430
p6.3.m.txt	1632	1872
p6.3.n.txt	1716	2046
p5.2.e.txt	130	155

TABLE 1 – Résultats comparés sans et avec 2-opt

Remarques L'ajout de 2-opt est plus ou moins efficace selon les groupes d'instance. Il arrive également que son ajout n'apporte aucun profit aux solutions initiales.

5.4 Amélioration or-opt

Dans la même logique, nous avons également implémenté une version permettant une amélioration des routes avec or-opt. Ce dernier est également appelé à chaque itération de la boucle *Tant_ que* sur la nouvelle route fusionnée si elle existe.

Complexité L'ajout de or-opt augmente la complexité de l'algorithme original à $O(n^4)$, or-opt ayant une complexité de $O(n^2)$.

Résultats De la même manière que pour 2-opt, un extrait des améliorations effectives est donné dans le tableau suivant :

Instance	Résultat (profit)	Résultat 2-opt	Résultat or-opt
p6.2.l.txt	1572	1704	1710
p6.2.m.txt	1572	2376	1806
p6.2.n.txt	1806	2430	1668
p6.3.m.txt	1632	1872	1632
p5.2.h.txt	435	485	500
p5.3.l.txt	625	690	705
p5.3.m.txt	525	605	620

TABLE 2 – Résultats comparés sans et avec or-opt

Remarques Nous remarquons que selon le groupe d'instances, or-opt est plus efficace que 2-opt et vice-versa. Le choix entre ces deux algorithmes d'optimisation est dépendant de chaque instance, en particulier de leur taille et de l'organisation des clients.

6 Conclusion

En conclusion, il existe des adaptations des algorithmes utilisés pour le VRP pour les différentes variantes de ce dernier, dont le TOP fait partie. Nous avons choisi d'implémenter une adaptation de l'algorithme de Clarke et Wright et de l'améliorer en utilisant 2-opt et or-opt sur un ensemble d'instances connues.

Références

I-Ming Chao and Bruce L. Golden. Algorithms and solutions to multi-level vehicle routing problems. 1993. URL https://api.semanticscholar.org/CorpusID:111701856.

I-Ming Chao, Bruce L. Golden, and Edward A. Wasil. A fast and effective heuristic for the orienteering problem. *European Journal of Operational Research*, 88(3) :475–489, 1996. ISSN 0377-2217. doi : https://doi.org/10.1016/0377-2217(95)00035-6. URL https://www.sciencedirect.com/science/article/pii/0377221795000356.

Racha El-Hajj, Duc-Cuong Dang, and Aziz Moukrim. Un algorithme de branch-and-cut pour la résolution du problème de tournées sélectives. 02 2013.

Javier Panadero, Christopher Bayliss, Angel Juan, and Christine Currie. Maximizing reward from a team of surveillance drones : a simheuristic approach to the stochastic team orienteering problem. *European J of Industrial Engineering*, 14, 12 2019. doi : 10.1504/EJIE.2020.108581.

Theodore Tsiligiridis. Heuristic methods applied to orienteering. *Journal of the Operational Research Society*, 35 :797–809, 09 1984. doi : 10.1057/jors.1984.162.