INF6102 - Métaheuristiques appliquées au génie informatique

Devoir 2 - Optimisation des soins à domicile

Guillaume Thibault
g.thibault@polymtl.ca

Guillaume Blanché
guillaume.blanche@polymtl.ca

27 mars 2023

1 Description du problème

Soit $G = (N, A)$ un graphe complet et orienté avec $N = \{0, 1, ..., n\}$. N est l'ensemble des nœuds du graphe, le nœud 0 représente le nœud de départ et A est une matrice de taille $N \times N$ où $a_{ij} \in A$ représente le temps nécessaire pour atteindre le nœud (patient) j à partir du nœud (patient) i. De plus, pour chaque patient, il existe une fenêtre temporelle $[l_i, u_i]$ où le patient $i \in N$ est disponible. Cette fenêtre temporelle indique que le patient i ne peut être visité ni avant l_i, ni après u_i. On admet cependant un temps d'attente pour le médecin, c'est-à-dire qu'il lui est possible d'arriver chez un patient avant le début de sa disponibilité à condition d'attendre qu'il soit disponible. Il n'est donc pas possible de quitter un patient i avant le début de sa disponibilité l_i. On suppose également que le temps de traitement d'un patient (c'est-à-dire le temps que le médecin passe au domicile du patient, une fois que ce dernier est disponible) est nul.

Ainsi, étant donné une tournée P, l'heure de départ D_{p_k} du patient p_k est $D_{p_k} = max(A_{p_k}, l_{p_k})$ où $A_{p_k} = D_{p_{k-1}} + a_{p_{k-1}p_k}$.

En d'autres termes, le départ d'un patient p_k est la valeur maximale entre :

- La borne inférieure de sa disponibilité (l_{p_k})
- La valeur de départ du patient précédent ($D_{p_{k-1}}$) + le temps de déplacement pour atteindre le patient actuel ($a_{p_{k-1}p_k}$).

Une solution au problème est représentée par $P = (p_0 = 0, p_1, ..., p_n, p_{n+1} = 0)$ où $(p_1, p_2, ..., p_n)$ est une permutation des noeuds de N En d'autres termes, c'est un cycle hamiltonien où p_k est l'index du patient visité à la k^{th} position de la tournée.

Mathématiquement, nous avons le modèle suivant :

$$\text{minimize } D_{p_{n+1}}$$

$$\text{Subject to } \sum_{k=o}^{n+1} \omega\left(p_k\right) = 0$$

Où

$$\omega\left(p_k\right) = 1 \text{ if } A_{p_k} > u_{p_k} \text{ else } 0$$

$$A_{p_{k+1}} = \max(A_{p_k}, l_{p_k}) + a_{p_k p_{k+1}}$$

La contrainte $\sum_{k=o}^{n+1} \omega\left(p_k\right) = 0$ assure que toutes les contraintes sur les fenêtres temporelles sont satisfaites. k=0 Notez qu'une solution valide doit 1. visiter tous les patients une et une seule fois, et 2. respecter les contraintes sur les fenêtres temporelles.

2 Méthode de résolution

2.1 Réimplémentation de l'état de l'art en matière de résolution de ce problème

Dans le cadre de notre devoir, afin de faire une étude comparative des méta-heuristiques, ainsi que pour mieux appréhender les techniques avancées de résolution de problèmes tels que celle de l'optimisation par colonie de fourmis, nous avons procédé à la réimplémentation de Beam-ACO proposée par Manuel Lopez-Ibanez et Christian Blum [LIB10]. Cette méthode de résolution est particulièrement efficace pour le problème du voyageur de commerce avec contraintes temporelles. L'approche repose sur une hybridation entre l'algorithme de recherche en faisceau et l'algorithme de construction probabiliste de colonies de fourmis. Cette démarche nous permettra de disposer d'une référence de base pour évaluer les performances de nos propres méta-heuristiques.

Le problème du voyageur de commerce avec fenêtres de temps est reconnu comme étant difficile pour trouver une solution initialement valide. Pour y remédier, la méthodologie de construction consiste à générer un certain nombre de solutions indépendantes par l'utilisation d'une recherche probabiliste des candidats potentiels à chaque étape d'une itération. La partie de recherche en faisceau implique la conservation d'un certain nombre des meilleures solutions partielles à chaque étape de la construction, en vue d'une extension ultérieure. Ces solutions partielles sont obtenues à partir d'un échantillonnage stochastique combinant un phénomène de phéromone issu de l'optimisation par colonies de fourmis, ainsi que des heuristiques visant à quantifier les bénéfices de la visite d'un client potentiel, correspondant à la priorisation des clients les plus proches, des clients ayant les fenêtres de disponibilité ce fermant le tôt ainsi que les clients ayant la fenêtre de disponibilité commençant le plus tôt. Cette méthode permet d'éviter l'évaluation coûteuse de tous les clients potentiels à chaque étape de la construction.

L'une des caractéristiques distinctives de cette approche par rapport aux autres algorithmes réside dans sa capacité à autoriser la construction de solutions infaisables, sans recourir à des termes de pénalité. Cette méthode utilise un moyen de comparaison de différentes solutions, y compris celles qui sont potentiellement infaisables. Ainsi, l'algorithme se concentre initialement sur la minimisation du nombre de violations de contraintes, et en cas d'un nombre identique de violations de contraintes, il procède à une comparaison des coûts de la tournée. Une fois la solution initiale construite, une méthode de recherche locale 2-opt est exécutée afin d'améliorer la solution.

La principale différence entre l'algorithme que nous avons implémenté et l'implémentation originale de Beam-ACO réside dans l'utilisation du langage de programmation Python au lieu de C++ et l'absence de calcul parallèle ainsi que l'utilisation de certaines méthodes de mise en cache pour éviter les coûts d'évaluation pendant la recherche locale. Cependant, ces différences ont entraîné une baisse notable de la performance de notre algorithme, ne permettant pas d'atteindre les mêmes résultats que ceux présentés dans l'article.

2.2 Implémentation d'un algorithme génétique

Les algorithmes génétiques font partie de la classe des algorithmes évolutionnistes : ils utilisent la notion de sélection naturelle et l'appliquent sur une population de solutions potentielles à un problème. Les algorithmes génétiques simulent ce processus évolutionnaire en 3 phases : la sélection, la mutation et la recombinaison de la population. Le choix d'un algorithme génétique pour résoudre ce problème est motivé par le fait que l'espace de solution du problème est très contraint par les fenêtres de temps.

2.2.1 Fonction d'évaluation

L'objectif de la fonction d'évaluation est d'orienter la recherche vers des meilleures solutions, ce qui en fait donc un élément de conception essentiel. Dans le problème du TSPTW, la fonction d'évaluation doit prendre en compte à la fois le coût total de la solution et le nombre de violations de fenêtres temporelles. Nous avons alors choisi une fonction d'évaluation simple sous la forme de la somme du coût de la tournée et d'un terme de pénalité pour chaque fenêtre temporelle non respectée. Il faut

cependant privilégier des solutions valides, et pour cela nous avons simplement pondéré lourdement le poids du nombre de violations de fenêtres temporelles. Nous avons alors mis cette fonction d'évaluation sous la forme : $-(\alpha N_{collisions} + D_{p_{n+1}})$ avec α un hyperparamètre dont la valeur doit être très grande devant $D_{p_{n+1}}$.

2.2.2 Description des opérateurs génétiques

La première étape de l'algorithme consiste à générer la population. Nous avons envisagé pour cela deux possibilités : d'une part, la génération de solutions de mauvaise qualité de façon totalement aléatoire, et d'autre part la génération de solution de meilleure qualité à l'aide des travaux présentés précédemment, en utilisant le Beam-ACO. Nous avons finalement privilégié la première méthode car c'est celle qui donnait les meilleurs résultats en pratique : cela est dû à la meilleure diversification permise par la création de solutions complètement aléatoires et à son coût de calcul nettement inférieur.

Ensuite, à partir d'une population donnée, nous générons des solutions filles selon une fonction de *crossover*. L'objectif est ici de croiser les informations issues des solutions parentes pour en proposer une nouvelle. Nous avons utilisé le m-point-crossover pour générer ces nouvelles solutions. Cet opérateur consiste à sélectionner m points distincts sur la longueur des solutions parentes et à échanger les segments entre les points correspondants pour créer deux nouvelles solutions descendantes. Il faut noter que m est un hyperparamètre à fixer : des valeurs de m élevées conduisent généralement à une plus grande diversification tandis que des valeurs faibles permettent une grande exploitation des solutions parentales. Pour équilibrer la diversification et l'intensification, nous choisissons cette valeur m aléatoirement sur une étendue de valeurs possibles définie.

Une fois ces solutions filles générées, un mécanisme de diversification des algorithmes génétiques consiste à altérer aléatoirement la solution avec une faible probabilité : il s'agit de la *mutation*. Nous avons choisi l'opérateur 2-swap : il consiste à sélectionner au hasard deux positions distinctes dans la solution et à en échanger leurs valeurs, ce qui préserve les contraintes dures de notre problème. Si cet opérateur est très simple de par son fonctionnement, il est important d'équilibrer soigneusement son utilisation en fixant le taux de mutation afin d'explorer efficacement l'espace de recherche.

A partir de cet ensemble, il est alors nécessaire de réaliser une *sélection* dans les nouvelles solutions, afin de conserver une population de bonne qualité et de taille fixe. Notre algorithme génétique réalise cette sélection sous la forme d'un tournoi. La sélection par tournoi est une méthode largement utilisée dans les algorithmes génétiques. Elle consiste à sélectionner aléatoirement un sous-ensemble de solutions candidates dans la population et à choisir les meilleurs individus de ce sous-ensemble comme parent pour la génération suivante. Dans chaque tournoi, les solutions sont comparées à l'aide de la fonction d'évaluation. La taille du sous-ensemble ainsi que le nombre de candidats acceptés par tournoi sont donc deux hyperparamètres à fixer. Le processus de sélection est répété jusqu'à ce que le nombre de parents souhaité soit atteint. La sélection par tournoi présente plusieurs avantages par rapport à d'autres méthodes de sélection. Elle est simple à mettre en oeuvre, elle garantit que les individus sélectionnés sont parmi les plus aptes et elle permet une certaine diversité puisque les individus sélectionnés dans chaque tournoi peuvent être différents.

La stratégie de redémarrage est un autre élément important dans la conception de méta-heuristiques. Elle est utilisée pour sortir des optima locaux et explorer de nouvelles régions de l'espace de recherche. Dans notre algorithme, cette stratégie est déclenchée lorsque qu'une population ne permet pas d'observer d'amélioration durant un certain nombre de générations. L'idée consiste ainsi à redémarrer la recherche sur une population aléatoire lorsque la recherche aboutit à un minima local.

2.3 Implémentation d'un algorithme glouton

Les algorithmes gloutons sont utiles pour trouver des solutions de bonne qualité très rapidement et ils sont de plus faciles à mettre en oeuvre. Les algorithmes gloutons sont donc particulièrement intéressants lorsque l'espace de recherche est trop grand pour une recherche exhaustive, même si ils ne garantissent pas l'obtention de la solution optimale. Nous avons implémenté un algorithme glouton (Algorithme 4) qui permet de trouver très rapidement la solution optimale sur les instances A_4

et B_20. L'idée de cet algorithme est la suivante : explorer les noeuds dans l'ordre de leur fenêtre d'ouverture. A chaque étape, l'algorithme sélectionne le noeud suivant qui satisfait la contrainte de temps et dont l'heure d'ouverture est la plus proche. L'algorithme se termine lorsque tous les noeuds ont été visités ou lorsqu'il ne reste plus aucun noeud satisfaisant la contrainte de temps.

2.4 Formalisation des algorithmes implémentés

2.4.1 Beam-ACO [LIB10]

Algorithm 1 Beam-ACO pour le problème TSPTW

```
best_solution ← null
restart_best_solution ← null
η = 0.5
bs_update ← false
while CPU time left ≠ 0 do
    potential_solution ← probabilisticBeamSearch()          ▷ Voir algorithme 2
    potential_solution ← localSearch(potential_solution)    ▷ 2-opt algorithme
    if potential_solution < best_solution then
        best_solution ← potential_solution
    end if
    if potential_solution < restart_best_solution then
        restart_best_solution ← potential_solution
    end if
    cf ← computeConvergenceFactor(η)
    if bs_update = trueandcf > 0.99 then
        η = 0.5
        restart_best_solution ← null
        bs_update ← false
    else
        if cf > 0.99 then
            bs_update ← true
        end if
        pheromoneUpdate(cf, bs_update, η, potential_solution, best_solution, restart_best_solution)
    end if
end while
return best_solution
```

Algorithm 2 Probabilistic Beam Search

```
Randomly define lambda          ▷ Poids des différentes heuristiques qui seront combinées
C ← 0                           ▷ Noeud de départ pour les N solutions à garder
number_of_children_not_in_solution ← N − 1
while number_of_children_not_in_solution > 0 do
    potential_states ← stochasticSampling(C, numbe_sample)   ▷ Voir algorithme 3
    C ← reduce(potential_states, beam_width)
    number_of_children_not_in_solution ← number_of_children_not_in_solution − 1
end while
return bestSolution(C)
```

Algorithm 3 Stochastic Sampling

$q = \text{randomBetween}(0.0, 1.0)$
$\tau = pheromone * \text{sampleHeuristic}()$
if $q <= determinism\_rate$ **then**
  return $\text{argmax}(\tau)$
else
  $p = (\tau / \sum \tau)$
  return $\text{choice}(\tau, probability = p)$
end if

2.4.2 Greedy algorithm

Algorithm 4 Greedy algorithm

$time \leftarrow 0$
$time\_constraints \leftarrow tsptw.time\_windows$
$nodes \leftarrow sort(nodes)$ ▷ Trier les noeuds dans l'ordre des fenêtres ouvrantes
$remaining\_nodes \leftarrow nodes$
$first\_node \leftarrow nodes[0]$
$solution \leftarrow [first\_node]$
$remaining\_nodes.remove(first\_node)$
while $remaining\_nodes$ is not empty **do**
  $next\_node \leftarrow null$
  for $node$ in $remaining\_nodes$ **do**
    if $check\_constraint(last\_node\_in\_solution, node, time, time\_constraints)$ **then**
      $next\_node \leftarrow node$
    end if
  end for
  if $next\_node$ is null **then**
    return $null$
  else
    $solution.add(next\_node)$
    $remaining\_nodes.remove(next\_node)$
    $time \leftarrow max(time + travel\_time, opening\_time(next\_node))$
  end if
end while
$solution.add(first\_node)$
return $solution$

2.5 Résultats obtenus

2.5.1 Beam-ACO

Fourmi	lr	τ_{min}	τ_{max}	% deterministic	Faisceau	mu	Violation	Score
0	-	-	-	0.1	10	5.0	2	730.16
0	-	-	-	0.1	10	5.0	3	731.47
1	0.1	0.001	0.999	0.1	10	5.0	3	700.07
5	0.1	0.001	0.999	0.1	10	5.0	4	700.07
25	0.1	0.001	0.999	0.1	10	5.0	4	690.265
15	0.1	0.050	0.950	0.1	10	5.0	4	696.2145

TABLE 1 – Résultats d'expérimentation avec Beam-ACO pour l'instance B_20.

Fourmi	lr	τ_{min}	τ_{max}	% deterministic	Faisceau	mu	Violation	Score
0	-	-	-	0.2	100	100	2	948.89
0	-	-	-	0.1	10	5.0	3	954.19
1	0.1	0.001	0.999	0.1	10	5.0	4	938.19
5	0.1	0.001	0.999	0.1	10	5.0	5	964.48
25	0.1	0.001	0.999	0.1	10	5.0	6	953.77
15	0.1	0.010	0.990	0.1	10	5.0	6	839.88

TABLE 2 – Résultats d'expérimentation avec Beam-ACO pour l'instance C_28.

Fourmi	lr	τ_{min}	τ_{max}	% deterministic	Faisceau	mu	Violation	Score
0	-	-	-	0.1	10	5.0	43	601.0
1	0.1	0.001	0.999	0.1	10	5.0	43	584.0
5	0.1	0.001	0.999	0.1	10	5.0	24	718.0

TABLE 3 – Résultats d'expérimentation avec Beam-ACO pour l'instance D_60.

Fourmi	lr	τ_{min}	τ_{max}	% deterministic	Faisceau	mu	Violation	Score
0	-	-	-	0.1	10	5.0	85	841.0
1	0.1	0.001	0.999	0.1	10	5.0	86	805.0
5	0.1	0.001	0.999	0.1	10	5.0	57	1181.0

TABLE 4 – Résultats d'expérimentation avec Beam-ACO pour l'instance E_100.

Les résultats obtenus à travers la résolution avec Beam-ACO n'ont pas permis d'atteindre les résultats optimaux. Après enquête, il a été observé que l'algorithme éprouvait des difficultés à trouver une solution valide après l'ajout d'environ quinze nœuds dans la solution. Ce problème pourrait être résolu par deux mécanismes. Le premier implique une implémentation améliorée pour réduire la complexité des calculs, en ajoutant des fonctions d'estimation pour mesurer les métriques qui ne peuvent pas être accélérées, ce qui permettrait de tester beaucoup plus d'options pour la même quantité de ressources et donc d'espérer trouver une meilleure construction de solution initiale. La deuxième option implique la modification des heuristiques utilisées pour la construction de la solution initiale. Dans l'article original, ces heuristiques sont expliquées de manière ou l'interprétation est de mise.

Finalement, nous avons constaté qu'il était plus difficile de trouver les bons hyperparamètres pour les algorithmes de colonie de fourmis. Il a été plus facile d'obtenir des résultats comportant moins de violations en utilisant simplement une recherche par faisceau (en gardant les phéromones à 1 sur toute la recherche, celles-ci étant multipliées par des heuristiques) plutôt qu'en utilisant un nombre X de fourmis.

2.5.2 Algorithme génétique avec recherche gloutonne

Valeurs des hyperparamètres choisis :
— Taille de la population P : 300
— Nombre maximal de générations : 1000
— Nombre de générations sans amélioration : 50
— Taille du tournoi : $T = \lceil \frac{N}{20} \rceil$
— Nombre de candidats acceptés par le tournoi : $\lceil \frac{T}{5} \rceil$
— Etendue de m pour le m-crossover : $2 \leq m \leq 10$
— Taux de mutation : 0.10
— α : 10^{10}

Instance	Violation	Score
A_4	0	117.85
B_20	0	592.06
C_28	2	1040.93
D_60	28	792.0
E_100	73	1314.0

TABLE 5 – Résultats d'expérimentation avec l'algorithme génétique sur les différentes instances

Notre seconde implémentation a été capable de trouver des solutions valides pour les petites instances : A_4 et B_20. Cependant, pour les instances plus grandes, l'algorithme génétique n'a pas été en mesure de produire des solutions valides. Il y a plusieurs raisons possibles qui peuvent expliquer ces difficultés. Tout d'abord, il est possible que ces derniers n'aient pas été suffisamment bien choisis pour s'échapper des minima locaux. Le choix des hyperparamètres est une difficulté propre aux algorithmes génétiques à laquelle nous avons dû faire face. Une solution envisageable pour améliorer cette sélection consisterait à fixer ces valeurs à l'aide d'un modèle d'apprentissage automatique. Une autre possibilité est que les opérateurs génétiques introduits dans notre algorithme ne permettent pas d'explorer suffisamment l'espace de recherche qui est lui même très vaste. Une amélioration possible consisterait à ajouter des mécanismes plus avancés sur les opérateurs simples que nous avons déjà implémentés.

2.6 Conclusion

Après avoir mesuré les performances des deux algorithmes de recherche locale que nous avons implémenté pour résoudre le problème du TSPTW, nous pouvons tirer quelques conclusions. Tout d'abord, aucun de nos deux algorithmes n'a été en mesure de trouver les solutions optimales sur chacune des instances du problème. Cela suggère que des mécanismes plus sophistiqués devraient être introduits pour générer plus facilement des solutions valides et de bonne qualité. Malgré tout, la mise en oeuvre de ces algorithmes et de leurs résultats peuvent fournir des indications sur la complexité du TSPTW et des difficultés liées à sa résolution.

Références

[LIB10] Manuel López-Ibáñez and Christian Blum. Beam-aco for the travelling salesman problem with time windows. *Computers and Operations Research*, 37(9) :1570–1583, 2010.