Rapport de projet : méta-heuristiques

Yuli Daune Funato, Thibaut Milhaud

Automne 2020

1 Présentation et modélisation du problème

1.1 Présentation

Le problème auquel nous devons proposer une solution se présente de la façon suivante :

Problème Étant donnés :

- I un ensemble de n points de $\mathbb{R}^2$;
- $p \in \mathbb{R}^2$
- $k \in \mathbb{N}$
- $K, R_{capt}, R_{com} \in \mathbb{N}$.

L'objectif est de calculer un ensemble $I' \subset I$ tel que

$$\forall i \in I', \exists i_1 ... i_n, i_1 = p \wedge i_n = i \wedge \forall j < n, d(i-j, i_{j+1}) \leq R_{com}$$

et

$$\forall i \in I, \exists i_1 ... i_k, \forall j \leq k, d(i_j, i) \leq R_{capt}$$

1.2 Modélisation

Nous avons choisi de représenter le problème à l'aide de deux graphes non orientés : un graphe COUV=(V, E_{Couv}) dit de couverture et un graphe COM=(V, E_{Com}) dit de communication.

- Les sommets des graphes sont les cibles à capter
- Dans le graphe de couverture, deux sommets sont reliés par une arête si et seulement si ils sont situés à une distance inférieure ou égale au rayon de captation R_{capt}
- Dans le graphe de communication, deux sommets sont reliés par une arête si et seulement si ils sont situés à une distance inférieure ou égale au rayon de communication R_{com}

Avec cette modélisation, le problème de k-couverture étudié se rapporte à trouver un sous-ensemble de sommets S tel que :

$$\begin{aligned} &\forall v \in V, \exists v_1 ... v_k \in \{v\} \cup N_{couv}(v), \forall i \leq k, v_i \in S \\ &\text{Le sous-graphe de COM induit par } S \cup \{p\} \text{ est connexe} \end{aligned} \tag{1}$$

où $N_G(v)$ désigne les voisins de v dans G et p désigne le sommet puits.

2 Algorithme semi-glouton pour la solution de base

L'algorithme que nous avons retenu pour la construction d'une solution gloutonne est le suivant :

Algorithm 1: algorithme glouton pour la solution de base

Result: un ensemble de sommets S correspondant à une solution de base pour la k-couverture

S=∅
L=V
while *L non vide* **do**
  Trier les sommets de L par valeur décroissante;
  Ajouter hd(L) à S;
  Retirer hd(L) de L;
  Actualiser les valeurs des sommets de L;
end
if *sous-graphe de COM induit par S non connexe* **then**
  Ajouter les plus courts chemins entre les composantes connexes
end

La valeur d'un sommet est égale au nombre de couvertures qu'apporte la pose d'un capteur sur ce sommet. Par exemple, si l'on considère le graphe complet à quatre sommets K_4, pour k=1, tous les sommets ont originellement une valeur de 3. Après la pose d'un premier capteur, il ne reste plus qu'un sommet avec une valeur non nulle (celui qui n'est pas voisin de là où on a posé le capteur).

3 Méta-heuristique choisie : Colonie de fourmis

Lorsque nous avons découvert le sujet, nous avons d'abord pensé aux méta-heuristiques de voisinage multiple et de recuit simulé, car le problème semblait s'y prêter. À l'inverse, nous avons convenu que les méta-heuristiques génétique et de colonie de fourmis étaient inadaptés, notamment cette dernière que nous ne connaissions qu'appliquée à des problèmes de plus court chemin.
Cependant, nous avons trouvé un article de 2004 ("Ant Colony Optimization for Multiple Knapsack Problem and Model Bias") qui décrit l'adaptation de l'algorithme de la colonie de fourmis au problème du sac à dos.

Nous avons choisi d'appliquer une méta-heuristique "colonie de fourmis" car elle permet un temps de calcul rapide et que nous n'avions jamais travaillé dessus. Voici, à titre de rappel, l'algorithme pour le problème du voyageur de commerce.

{*début*}
  Pour $t = 1, \ldots, t_{max}$ **Faire**
    Pour Chaque fourmi $k = 1, \ldots, m$ **Faire**
      {Construction d'un tour}
      Choisir une ville au hasard
      Pour Pour chaque ville non visitée i **Faire**
        Choisir une ville $j \in J_i^k$ selon la règle de déplacement.
      FinPour
      {Mise à jour des phéromones : dépôt}
      Dépôt de $\Delta\tau_{ij}^k(t)$ sur le trajet $T^k(t)$
    FinPour
    {Mise à jour des phéromones : évaporation}
    Évaporer les pistes
  FinPour
  Retourner le meilleur tour rencontré
{*fin*}

3.1 Adaptation de l'algorithme

Voici notre algorithme, adapté au problème de la k-couverture :

Algorithm 2: Algo de Colonie de fourmis pour k-couverture

Result: un ensemble de sommets S correspondant à une solution pour la k-couverture
{On pose S : meilleure solution rencontrée}
while *Critères d'arrêt non remplis* **do**
  for $k \leq$ *nombre de fourmis* **do**
    Construire une solution S'
    Déposer les phéromones sur les sommets sélectionnés
    Actualiser S si $|S'| \leq |S|$
  end
  Évaporation des phéromones
end

Les paramètres à modifier incluent donc :

- Le nombre de fourmis de la colonie;
- La façon dont les phéromones sont déposés à la fin de chaque passage de la colonie;
- L'évaporation des phéromones;

- La procédure de construction de la solution à partir des pistes de phéromones;
- Les critères d'arrêt.

Comme critère d'arrêt, nous avons choisi de fixer un nombre d'itérations max, défini lors de l'appel de la fonction.

3.2 Algorithme de construction des solutions

Pour construire sa solution S, chaque fourmi va procéder de manière "gloutone" : chaque fourmi commence avec $S = \emptyset$ et ajoute les sommets 1 par 1 parmi les candidats valides. Un candidat x et dit valide si il est adjacent à S et $x \notin S$.

Pour pouvoir générer un ensemble de solution un peu varié, nous avons ajouté de l'aléa dans le choix du prochain sommet à ajouter. De plus le phéromones laissées par les fourmis lors des itérations précédentes entrent également en jeu lors choix du prochain sommet.

Ainsi la sélection du prochain sommet s'effectue de la façon suivante :

Algorithm 3: Choix du voisin

Result: le prochain sommet qui sera ajouté dans la solution
$CS[n_v + 1] = [0 \dots 0]$ le tableau de scores cumulés
for $1 \leq v \leq n_v$ **do**
  $CS[v] = CS[v-1] +$ score$(v) +$ phéromones(v)
$x = \text{rand}(CS[n_v])$
L'indice du voisin que l'on cherche est i tel que $x \in [CS[i], CS[i+1]]$
On obtient i par dichotomie sur CS qui est trié.

Une fois le sommet choisi, il faut l'ajouter à la solution courante et mettre à jour les voisins. Et recommencer jusqu'à ce que tout les sommets soient couverts par k capteurs.

Cette méthode de construction permet de garantir que l'ensemble S qui définit la solution courante est toujours connexe (vis à vis du graphe COM) et nous évite donc d'avoir à vérifier ce point.

3.3 Mise à jours du niveau de phéromone sur les sommets

L'actualisation des niveaux de phéromones se fait avec un barycentre qui dépend d'un paramètre $\rho \in [0; 1]$:

$$pherom_i(v) = \rho * pherom_{i-1}(v) + (1 - \rho) * newpherom(i, v)$$

où $pherom_n(v)$ est la quantité de phéromones sur le sommet v après n itérations, et $newpherom(i, v)$ est la quantité de phéromones qui a été déposée sur le sommet v à l'itération n.

4 Résultats et analyse

4.1 Expérience

Pour un ensemble de paramètres donné, nous lançons notre algorithme N fois, et relevons la solution minimale trouvée (en nombre de capteurs posés) ainsi que la distribution en quartiles.

Ceci étant dit, notre algorithme, pour des paramètre fixé ne varie que des peu selon les exécutions. En effet pour $N = 100$ les écarts observés entre le min et le troisième quartile étaient de l'ordre de 1 ou 2. Nous présenterons donc par la suite une seule valeur et non pas 3.

4.2 Résultats

4.2.1 k=1

Dataset	Rcapt	Rcom	Majorant	Temps (s)
15-100	1	1	604	1,4
15-100	1	2	313	2
15-100	2	2	180	0,9
15-100	2	3	115	0,8
18-100	1	1	1013	1,3
18-100	1	2	555	2,1
18-100	2	2	201	1,1
18-100	2	3	185	0,9
12-100	1	1	88	0,2
12-100	1	2	66	0,3
12-100	2	2	29	0,2
12-100	2	3	23	0,2
7-4	1	1	30	0,01
7-4	1	2	24	0,01
7-4	2	2	9	0,01
7-4	2	3	7	0,01
8-10	1	1	47	0,06
8-10	1	2	32	0,06
8-10	2	2	13	0,03
8-10	2	3	10	0,03
15-20	1	1	550	0,4
15-20	1	2	248	0,8
15-20	2	2	153	0,4
15-20	2	3	110	0,4

4.2.2 k=2

Dataset	Rcapt	Rcom	Majorant	Temps (s)
15-100	1	1	357	1,5
15-100	1	2	285	2,3
15-100	2	2	187	1,1
15-100	2	3	178	1,1
18-100	1	1	434	1,3
18-100	1	2	393	2,4
18-100	2	2	173	1,3
18-100	2	3	123	1,3
12-100	1	1	137	0,3
12-100	1	2	134	0,4
12-100	2	2	44	0,3
12-100	2	3	41	0,3
7-4	1	1	49	0,01
7-4	1	2	45	0,01
7-4	2	2	15	0,01
7-4	2	3	13	0,01
8-10	1	1	65	0,04
8-10	1	2	59	0,04
8-10	2	2	19	0,07
8-10	2	3	19	0,04
15-20	1	1	241	0,5
15-20	1	2	236	0,9
15-20	2	2	142	0,5
15-20	2	3	84	0,5

4.2.3 k=3

Dataset	Rcapt	Rcom	Majorant	Temps (s)
15-100	1	1	446	1,6
15-100	1	2	348	2,5
15-100	2	2	230	1,3
15-100	2	3	144	1,3
18-100	1	1	483	1,4
18-100	1	2	438	2,5
18-100	2	2	145	1,4
18-100	2	3	143	1,5
12-100	1	1	190	0,3
12-100	1	2	187	0,5
12-100	2	2	61	0,3
12-100	2	3	55	0,3
7-4	1	1	68	0,01
7-4	1	2	65	0,01
7-4	2	2	19	0,01
7-4	2	3	18	0,01
8-10	1	1	90	0,06
8-10	1	2	90	0,06
8-10	2	2	31	0,07
8-10	2	3	29	0,06
15-20	1	1	349	0,5
15-20	1	2	321	0,9
15-20	2	2	152	0,5
15-20	2	3	104	0,6

4.3 Analyse

4.3.1 Ajustement des paramètres

- La valeur de ρ (paramètre d'évaporation) optimale semble être autour de 0,7.
- Augmenter significativement le nombre d'itérations dans le critère d'arrêt ne semble pas améliorer la qualité de la solution, surtout dans les problèmes de petite taille.
- Augmenter le nombre de fourmis ne semble pas améliorer significativement la qualité de la solution, il y a un "nombre d'équilibre" qui dépend de la taille du problème (32 fourmis ne font pas vraiment mieux que 8 pour 7-4, le plus petit dataset, mais ce n'est pas le cas pour les plus gros).
- Faire varier le coefficient de proportionnalité du dépôt de phéromones ne change pas grand chose aux solutions trouvées

4.3.2 Analyse globale

On remarque que dans la plupart des instances, la méta-heuristique converge rapidement et peine à faire significativement mieux que la solution gloutonne. En particulier, pour k=1, certains résultats sont désastreux (notamment pour les gros problèmes). De manière générale, en général, notre approche réagit mal à l'augmentation de taille car les solutions semblent rester trop proches de ce que fournit l'approche gloutonne. Voici plusieurs pistes d'explication :

- La procédure gloutonne est particulièrement efficace.
- Notre adaptation de la méta-heuristique des fourmis est inadaptée au problème.
- La méta-heuristique des fourmis est inadaptée au problème.
- Nous n'avons pas réussi à ajuster les paramètres de façon efficace.

4.3.3 Pistes d'amélioration

- Notre code est déjà rapide, mais l'algorithme de colonie de fourmis se prête particulièrement bien à la parallélisation de code
- Mettre en place une meilleure procédure d'amélioration locale à la fin de chaque passage de la colonie, ou plus généralement de plus "perturber" la solution en cours pour avoir moins de chance de se coincer dans un optimum local.