Étude de cas : k-couverture connexe minimum dans les réseaux de capteurs

BATY Léo, BRUNOD-INDRIGO Luca

3 novembre 2020

Table des matières

1 Calcul de bornes inférieures

Dans cette section, on pose le problème sous la forme d'un programme linéaire en nombres entiers (PLNE) et on en déduit des bornes inférieures par relaxation linéaire.

1.1 Notations

Voici tout d'abord les différentes notations qui seront utilisées dans la suite :

- constante de connexité $k \in \{1, 2, 3\}$
- rayons de captation et de communication$R^{capt} \leq R^{com}$
- T ensemble des cibles (targets), de cardinal n
  - $\rightarrow$ $t \in T$, coordonnées (x_t, y_t)
  - $\rightarrow$ $t, t' \in T, t \neq t', \Delta_{t,t'} = \sqrt{(x_t - x_{t'})^2 + (y_t - y_{t'})^2}$
- puits s de coordonnées (x_s, y_s)
- $E^{capt} = \{(t, t') \in T^2 \mid t \neq t', \Delta_{t,t'} \leq R^{capt}\}$
  - $\rightarrow$ $E^{capt}_t = \{t' \in T |\ (t, t') \in E^{capt}\}$
  - $\rightarrow$ **Graphe de captation** : $\boxed{G^{capt} = (T, E^{capt})}$
- $E^{com} = \{(t, t') \in (T \cup \{s\})^2 |\ t \neq t', \Delta_{t,t'} \leq R^{com}\}$
  - $\rightarrow$ $E^{com}_t = \{t' \in T |\ (t, t') \in E^{com}\}$
  - $\rightarrow$ **Graphe de communication** : $\boxed{G^{com} = (T \cup \{s\}, E^{com})}$
  - $\rightarrow$ On note $\mathcal{D}(E^{com})$ l'ensemble des arrêtes orientées contenant les (t, t') et (t', t).
- $\forall t \in T$, $\delta_t = \mathbb{1}_{\{\text{capteur sur la cible } t\}}$
- $\forall e \in E^{com}$, $x_e = \mathbb{1}_{\{e \text{ dans l'arbre de communication}\}}$

1.2 Modèle PLNE

On décrit une solution à l'aide des variables de décision suivantes :

- $\forall t \in T$, $\delta_t = \mathbb{1}_{\{\text{capteur sur la cible } t\}}$
- $\forall e \in E^{com}$, $x_e = \mathbb{1}_{\{e \text{ dans l'arbre de communication}\}}$ (l'arbre de communication est un arbre couvrant toutes les cibles possédant un capteur dans le graphe de communication G^{capt})
- $\forall a \in \mathcal{D}(E^{com})$, $y_a \in \mathbb{R}_+$: on définit un flot sur $\mathcal{D}(E^{com})$, qui part de s, de valeur égale au nombre de capteurs placés, et dont chaque cibles possédant un capteur réceptionne une unité.

Afin que les contraintes définies ci-après soient valables aussi pour le puits s on fixe $\delta_s = 1$:

- On minimise le nombre de capteurs placés : (1a)
- k-connexité : (1b)
- nombre d'arêtes de l'arbre fixé à n :(1c)
- Une arête n'appartient pas à l'arbre couvrant si au moins une de ses deux extrémités ne possède pas de capteur : (1d), et (1e)

- Pour chaque cible, la différence entre le flot entrant et le flot sortant vaut 1 si elle possède un capteur, et 0 sinon : (1f)
- Le différence entre le flot sortant et le flot entrant dans le puits est égal au nombre de cibles placées : (1g)
- On interdit le flot sur les arcs dont l'arête associées n'est pas dans l'arbre de communication entre les capteurs : (1h) et (1i)

Voici le PLNE complet obtenu :

$$\min_{\delta, x, f} \quad \sum_{t \in T} \delta_t \tag{1a}$$

$$\text{s.t.} \quad \sum_{t' \in E_t^{capt}} \delta_{t'} \geq k, \qquad \forall t \in T \tag{1b}$$

$$\sum_{e \in E^{com}} x_e = n, \tag{1c}$$

$$x_e \leq \delta_t, \qquad \forall e = (t, t') \in E^{com} \tag{1d}$$

$$x_e \leq \delta_{t'}, \qquad \forall e = (t, t') \in E^{com} \tag{1e}$$

$$\sum_{t' \in E_t^{com}} y_{(t',t)} - \sum_{t' \in E_t^{com}} y_{(t,t')} = \delta_t, \qquad \forall t \in T \tag{1f}$$

$$\sum_{t \in E_s^{com}} y_{(s,t)} - \sum_{t \in E_s^{com}} y_{(t,s)} = \sum_{t \in T} \delta_t, \tag{1g}$$

$$y_{(t,t')} \leq n \times x_e, \qquad \forall e = (t, t') \in E^{com} \tag{1h}$$

$$y_{(t',t)} \leq n \times x_e, \qquad \forall e = (t, t') \in E^{com} \tag{1i}$$

$$\delta_t \in \{0, 1\}, \qquad \forall t \in T \tag{1j}$$

$$x_e \in \{0, 1\}, \qquad \forall e \in E^{com} \tag{1k}$$

$$y_a \geq 0, \qquad \forall a \in \mathcal{D}(E^{com}) \tag{1l}$$

1.3 Relaxation linéaire et obtention des bornes inférieures

Nous avons implémenté le PLNE en Julia en utilisant la librairie JuMP (librairie de programmation mathématique) couplée avec le solveur Gurobi (licence académique), afin de résoudre la relaxation linéaire de ce problème, voire la résolution exacte par Branch and Bound pour les plus petites instance. Nous obtenons exactement les mêmes bornes inférieures que celles fournies avec les instances, sauf pour les plus grosses instances pour lesquelles le temps de calcul était trop important pour pouvoir les calculer sur nos ordinateurs.

2 Heuristiques

2.1 Heuristique par captations de coût minimal successives

Cette première heuristique a été utilisée au cours du projet pour générer les solutions de départ des métaheuristiques. Cet algorithme construit de façon déterministe une solution admissible à partir

d'une "graine" qui peut être générée aléatoirement. Les solutions données par cette heuristique se sont révélées être d'une qualité satisfaisante, au prix d'un coût en calcul plus élevé que d'autres méthodes de construction plus naturelles.

Algorithm 1 Heuristique par captations de coût minimal successives

Input :
$(T, G^{capt}, G^{com}, k)$: une instance du problème
L : une liste contenant k fois chaque cible de T dans un ordre quelconque

Output : $N \subset T$: le sous-ensemble de cibles sur lesquelles placer des capteurs

$N = \emptyset$
for all $t \in L$ **do**
  Choisir t' tel que $(t, t') \in E^{capt}$ et qui minimise le plus court chemin vers $s \cup N$ dans G^{com}
  $E^{capt} = E^{capt} \setminus (t, t')$
  Ajouter à N les cibles situées sur le plus court chemin de t' à $s \cup N$ dans G^{com}
end for
Renvoyer N

L'idée de l'algorithme est fondée sur le fait que, dans toute solution admissible, toute cible a dans G^{capt} au moins k voisins distincs qui portent un capteur.
La liste L fournie en entrée sert de "graine", elle contient k fois l'indice de chaque cible. À chaque itération de la boucle principale de l'algorithme, il va être fait en sorte que la cible t corresponant à l'indice dans L ait au moins un capteur parmi ses voisins dans G^{capt}. Pour cela, des capteurs supplémentaires vont être placés de façon à relier dans G^{com} un voisin de la cible à l'ensemble formé par le puits et les capteurs déjà présents. Le nombre de capteur ajoutés à chaque itération est minimisé en choisissant un voisin de la cible au plus près d'un capteur déjà placé ou du puits et en plaçant les capteurs additionnels sur un plus court chemin vers cet ensemble. Afin que le voisin choisi ne soit pas réutilisé pour la même cible lors d'une prochaine itération, l'arète les reliant est retirée de G^{capt}.
Il convient de remarquer qu'au terme de chaque itération, l'ensemble formé par le puits et les capteurs est connexe dans G^{com}. À la fin de l'algorithme, il est donc assuré que chaque cible a au moins k capteurs parmi ses voisins dans le graphe G^{capt} de l'instance et que ces capteurs sont reliés au puits dans G^{com}.

Une implémentation naturelle de cette heuristique consiste à recalculer à chaque itération les plus courts chemins depuis le puits dans le graphe G^{com} muni de poids binaires choisis. Cela revient à appliquer $k|T|$ fois l'algorithme de Dijkstra, soit une complexité en $O(k|T|^3)$.
Cependant, au prix du précalcul en $O(|T|^3)$ des plus courts chemins entre toute paire de cibles ainsi que des plus courts chemins vers le puits dans G^{com} muni de poids unitaires, cette complexité peut être réduite à $O(k|T|^2)$.

2.2 Heuristique gloutonne avec ordre sur les cibles

Cette seconde heuristique suit un principe plus simple et est plus rapide que la précédente. Elle construit une solution admissible de façon déterministe à partir d'un ordre sur les cibles. Contraire-

ment à l'heuristique par captations de coût minimal successives qui fournit généralement des solutions de bonne qualité en tirant aléatoirement des "graines", cet algorithme donne globalement de très mauvais résultats lorsqu'il génère des solutions à partir d'ordres aléatoires.
En contrepartie, l'ensemble des solutions admissibles atteignables est a priori plus vaste que pour la première heuristique qui se contraint à placer des capteurs sur des plus courts chemins.

Algorithm 2 Heuristique gloutonne avec ordre sur les cibles

Input :
$(T, G^{capt}, G^{com}, k)$: une instance du problème
$\prec$: une relation définissant un ordre sur les cibles

Output : $N \subset T$: le sous-ensemble de cibles sur lesquelles placer des capteurs

$N = \emptyset$
$Q = \{t \in T | (s,t) \in E^{com}\}$
$U = \emptyset$
while Toutes les cibles n'ont pas au moins k voisins appartenant à N dans G^{capt} **do**
  if $Q \setminus U \neq \emptyset$ **then**
    Choisir t dans $Q \setminus U$ maximal pour la relation d'ordre $\prec$
    if t a au moins un voisin dans G^{capt} ayant moins de k voisins dans N **then**
      $Q = Q \setminus \{t\}$
      $Q = Q \cup \{t' | (t,t') \in E^{com}\}$
      $N = N \cup \{t\}$
    else
      $U = U \cup \{t\}$
    end if
  else
    Choisir t dans $Q \cap U$ maximal pour la relation d'ordre $\prec$
    $Q = Q \setminus \{t\}$
    $Q = Q \cup \{t' | (t,t') \in E^{com}\}$
    $N = N \cup \{t\}$
  end if
end while
Renvoyer N

L'idée de l'algorithme est de construire un réseau connexe dans G^{com} en plaçant des capteurs de proche en proche à partir du puits. À chaque itération, les cibles candidates pour recevoir un nouveau capteur sont celles situées dans le voisinage du puits ou des capteurs déjà placés dans G^{com}. La cible choisie pour placer le nouveau capteur est celle la mieux classée dans l'ordre donné en entrée. Néanmoins, pour éviter de placer des capteurs "inutiles", si la cible choisie a tous ses voisins dans G^{capt} déjà couverts par au moins k capteurs, elle est mise "en attente" et ne sera reconsidérée que s'il n'y a plus aucune cible "utile" parmi les candidates.
L'algorithme s'arrête lorsque toutes les cibles sont couvertes par au moins k capteurs. Le réseau des capteurs placés étant connexe dans G^{com} par construction, la solution obtenue est bien admissible.

En utilisant des structures de données adaptées pour tenir le compte du nombre de capteurs couvrant chaque cible, cet algorithme peut être implémenté avec une complexité en $O(|T|^2)$.

3 Structure de voisinage

Étant donné que nous disposons d'une heuristique rapide et capable d'explorer l'espace des solutions admissibles en jouant sur le paramètre d'ordre donné en entrée, nous avons choisi de la mettre à profit pour élaborer une structure de voisinage.
Nous avons donc opté pour un voisinage qui pourrait être qualifié d'indirect en travaillant non pas sur les solutions elles-mêmes mais sur des vecteurs définissant des ordres sur les cibles. En procédant de la sorte, nous avons pu nous ramener à des transformations élémentaires très simples consistant à permuter les composantes des vecteurs. Ces permutations ont l'avantage de pouvoir être effectuées sans se soucier des contraintes du problème initial puisque tout vecteur d'ordre mène à une solution admissible par l'intermédiaire de l'heuristique gloutonne.

4 Métaheuristiques

4.1 Recuit simulé

Afin d'améliorer les solutions obtenues à l'aide de l'heuristique 1, nous avons mis en œuvre une métaheuristique de type recuit simulé comme vu en cours, en utilisant des 2-opt dans la structure de voisinage définie dans la section précédente. Nous avons paramétré le recuit simulé de la manière suivante :

- Paramètre multiplicateur de **décroissance de la température** : $\phi = 0.95$
- **Nombre d'itérations** par palier de température : number_iterations = 10000
- **Température initiale** : nous avons fixé un paramètre correspond à la probabilité initiale de garder une modification qui dégrade l'objectif à initial_keep_probability = 0.8. Puis en calculant une moyenne empirique de la valeur de dégradation d'une solution, on en déduit une valeur de la température initiale : $T_0 = \dfrac{-\Delta_{mean}}{\ln(\text{initial\_keep\_probability})}$
- **Température finale** : de manière similaire au calcul la température initiale on a choisi final_keep_probability $= 1e-10$ qui correspond à la probabilité finale d'accepter une solution qui augmente l'objectif de 1. On obtient donc $T_{final} = \dfrac{-1}{\ln(\text{final\_keep\_probability})}$

En sortie du recuit, on applique en plus une recherche locale. Voici les résultats finaux obtenus, et leur comparaison avec les bornes inférieures :

TABLE 1 – Comparaison entre : résultats du recuit simulé | bornes inférieures

$(\mathbf{k}, \mathbf{R^{capt}}, \mathbf{R^{com}})$	**150-7-4**	**225-8-10**	**625-12-100**	**900-15-20**	**1500-15-100**	**1500-18-100**
(1, 1, 1)	31 | 21	45 | 26	89 | 54	137 | 82	138 | 80	198 | 113
(1, 1, 2)	19 | 19	25 | 25	55 | 53	83 | 80	88 | 80	121 | 112
(1, 2, 2)	8 | 6	11 | 8	24 | 16	37 | 23	37 | 23	55 | 32
(1, 2, 3)	6 | 6	8 | 8	18 | 16	27 | 22	28 | 23	39 | 31
(2, 1, 1)	44 | 40	56 | 50	119 | 106	185 | 162	189 | 159	261 | 225
(2, 1, 2)	39 | 39	50 |50	109 | 106	166 | 161	174 | 159	239 | 225
(2, 2, 2)	12 | 11	16 | 15	34 | 31	52 | 44	53 | 45	73 | 62
(2, 2, 3)	11 | 11	16 | 15	32 | 31	49 | 44	52 | 45	69 | 62
(3, 1, 1)	62 | 60	79 | 78	166 | 162	253 | 246	260 | 239	355 | 341
(3, 1, 2)	60 | 60	78 | 78	166 | 162	251 | 245	257 | 239	355 | 341
(3, 2, 2)	17 | 17	23 | 23	49 | 46	74 | 66	75 | 67	103 | 93
(3, 2, 3)	17 | 17	23 | 23	48 | 46	72 | 66	75 | 67	105 | 93

Remarque : pour l'instance `1500-18-100`, on obtient une solution à 103 pour le jeu de paramètres $(3, 2, 2)$ et 105 pour le jeu de paramètres $(3, 2, 3)$. Toute solution de $(3, 2, 2)$ étant admissible pour $(3, 2, 3)$, on a donc aussi une solution à 103 pour $(3, 2, 3)$.

4.2 Recherche à voisinages multiples

Afin d'affiner et d'améliorer de manière très légère les résultats du recuit simulé, nous avons implémenté une recherche à voisinages multiples, qui part des solutions du recuit, et qui alterne entre des 2-opt, 3-opt, et 4-opt, cela pendant 500000 itérations. On a observé une amélioration des solutions qui varie entre 0 et 5. Voici une table résumant les résultats obtenus, avec en couleur les jeux de paramètres/instances pour lesquels on a amélioré la solution d'au moins 1.

TABLE 2 – Comparaison entre : résultats finaux | bornes inférieures

$(\mathbf{k}, \mathbf{R^{capt}}, \mathbf{R^{com}})$	**150-7-4**	**225-8-10**	**625-12-100**	**900-15-20**	**1500-15-100**	**1500-18-100**
(1, 1, 1)	31 | 21	45 | 26	88 | 54	137 | 82	137 | 80	196 | 113
(1, 1, 2)	19 | 19	25 | 25	55 | 53	82 | 80	87 | 80	121 | 112
(1, 2, 2)	8 | 6	11 | 8	24 | 16	36 | 23	36 | 23	53 | 32
(1, 2, 3)	6 | 6	8 | 8	18 | 16	26 | 22	27 | 23	38 | 31
(2, 1, 1)	44 | 40	56 | 50	117 | 106	183 | 162	184 | 159	259 | 225
(2, 1, 2)	39 | 39	50 |50	109 | 106	165 | 161	170 | 159	237 | 225
(2, 2, 2)	12 | 11	16 | 15	34 | 31	50 | 44	53 | 45	71 | 62
(2, 2, 3)	11 | 11	15 | 15	32 | 31	48 | 44	50 | 45	69 | 62
(3, 1, 1)	61 | 60	79 | 78	165 | 162	253 | 246	255 | 239	354 | 341
(3, 1, 2)	60 | 60	78 | 78	165 | 162	249 | 245	253 | 239	352 | 341
(3, 2, 2)	17 | 17	23 | 23	49 | 46	73 | 66	75 | 67	102 | 93
(3, 2, 3)	17 | 17	23 | 23	48 | 46	72 | 66	74 | 67	102 | 93