

Projet Méta-heuristique (MPRO) : Couverture connexe minimum dans les réseaux des capteurs

Groleaz Lucas
Houdayer Antoine

31 octobre 2016

Table des matières

I	Sujet	2
1	Modélisation	2
2	Reformulation	4
3	Détails d'implémentation	4
II	Approche par chaînes de destruction	6
4	Heuristique	6
4.1	Heuristique déterministe	6
4.2	Heuristique aléatoire	8
5	Les chaînes de destruction	9
5.1	Définition	9
5.2	Propriétés	9
5.2.1	Chaînes de destruction et solutions non dégénérées . .	9
5.2.2	Chaînes équivalentes	11

6	Voisinage induit	12
6.1	Définition et propriétés	12
6.2	Exemple	13

Première partie

Sujet

1 Modélisation

Dans tous le problème, on recherche à placer des capteurs sur une grille de façon optimale. Autrement dit, on peut placer des capteur sur les points à coordonnées entières qui sont à l'intérieur d'une zone prédéfinie. Dans toute la suite on travaillera sur des grilles carrées, pour alléger les notations. On notera n la dimension de la grille. Ainsi chaque sommet v de la grille est un élément de $V = \llbracket 1, n \rrbracket \times \llbracket 1, n \rrbracket$. On notera $S \subset V$ l'ensemble des somment sur lesquels on a choisi de placer un capteur, c'est notre façon d'encoder la solution.

Chaque capteur est doté d'un rayon de captation R_{capt} , et d'un rayon de communication $R_{comm} \geq R_{capt}$. On souhaite utiliser un minimum de capteurs tout en respectant deux contraintes.

- Contrainte de connexité : La grille est doté d'un puits, point de coordonnées $(1; 1)$. Un capteur peut communiquer avec tout capteur ou avec le puits, à condition qu'il soit séparé de ce dernier d'une distance inférieure ou égale à R_{comm} . Tout capteur doit pouvoir communiquer avec le puits, soit directement, soit par l'intermédiaire d'autres capteurs.
- Contrainte de couverture : Tout sommet de la grille, excepté le puits, doit être à une distance inférieure ou égal à R_{capt} du capteur le plus proche.

N.B : Il n'est pas interdit de couvrir le puits (ce qui est même inévitable si $R_{capt} = R_{comm}$), ou d'y placer un capteur, mais l'expérience montre qu'il n'est pas intéressant en pratique de placer un capteur dans un coin de la grille. Comme le puits est un point particulier, on doit appliquer un traitement particulier aux capteurs que l'on place dessus. Notamment, si on enlève un capteur situé sur le puits, les autre capteurs continuent de communiquer avec le puits. De plus avec les nombreuses rotations et symétries possibles, il faudrait que toutes les solutions optimales aient des capteurs aux quatre coins pour qu'interdire un capteur sur le puits soit impactant. Ainsi, nous

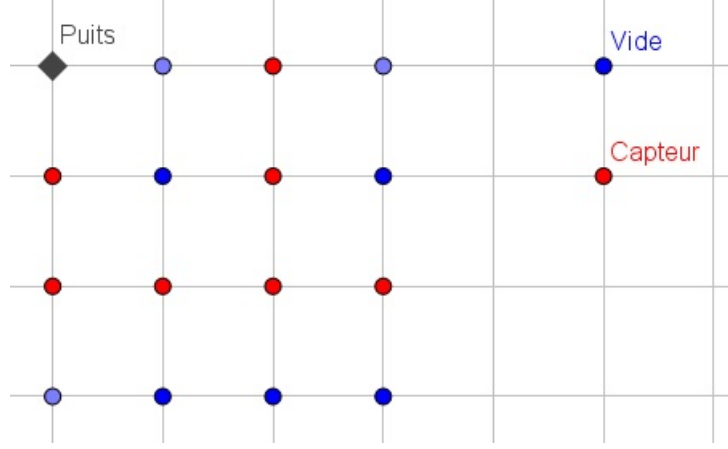


FIGURE 1 – Exemple de solution réalisable pour $n = 4$, $R_{capt} = R_{comm} = 1$, ici $|S| = 7$

avons pris le parti de ne pas mettre de capteur sur le puits. Bien sur, il faut toujours envisager cette option dans les parties théoriques.

2 Reformulation

Nous avons adopté une reformulation du problème à base de graphes, un pour chaque contrainte. On note $p = (1; 1)$, le puits. Pour une solution S donnée, non nécessairement réalisable, on définit les ensembles d'arêtes $E_{capt}(S)$, l'ensemble des liens de captations, et $E_{comm}(S)$ l'ensemble des liens de communication. Formellement on a :

$$E_{capt}(S) = \{\{v_1, v_2\} | v_1 \in V \setminus p, v_2 \in S \setminus v_1, \Delta(v_1, v_2) \leq R_{capt}\}$$

$$E_{comm}(S) = \{\{v_1, v_2\} | v_1, v_2 \in S \cup \{p\}, v_1 \neq v_2, \Delta(v_1, v_2) \leq R_{comm}\}$$

On peut alors construire les graphes correspondants :

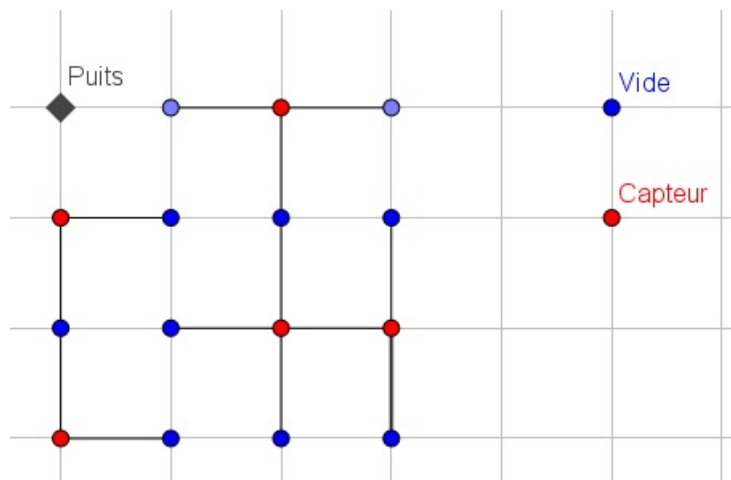
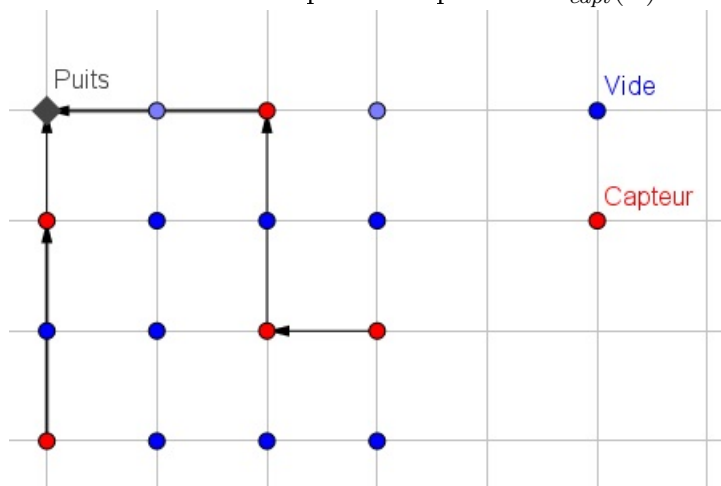
$$G_{capt}(S) = (V, E_{capt}(S))$$

$$G_{comm}(S) = (S \cup p, E_{comm}(S))$$

Ci-dessous, une illustration dans le cas $n = 4$, $R_{capt} = 1$, $R_{comm} = 2$. On a laissé tous les sommets de S sur le schéma pour plus de lisibilité.

N.B. : Sur le graphe de G_{comm} , on a représenté les arêtes par des flèches, pour illustrer la notion de puits. Cependant, le graphe n'est pas intrinsèquement orienté, et il pourrait tout à fait présenter des cycles.

Le problème peut alors se reformuler de la façon suivante :

FIGURE 2 – Graphe de captation $G_{capt}(S)$ FIGURE 3 – Graphe de communication $G_{comm}(S)$

- Minimiser $|S|$.
- Contrainte de connexité : $G_{comm}(S)$ est connexe.
- Contrainte de couverture : $G_{capt}(S)$ n'a pas de sommet isolé autre que le puits.

3 Détails d'implémentation

Nous avons choisi d'utiliser le langage `C++` pour tenter d'implémenter une méthode de résolution approchée du problème. Nous avons tâché d'utiliser une implémentation du problème alliant simplicité d'utilisation et

efficacité. Pour cela, nous utilisons un codage double des solutions :

Pour garder en mémoire quels sont les sommets sur lesquels un capteur est placé, le sommets captés, et les capteurs appartenant à la composant connexe du puits dans $G_{comm}(S)$, nous utilisons trois tableaux de bits de type bitset. Leurs principaux avantages sont leur faible occupation de mémoire, mais surtout la possibilité de les manipuler grâce à des opérations binaires. En contrepartie, leur taille doit être fixée à la compilation. Nous les avons déclarés de taille 2500, pour traiter des grilles ayant moins de 2500 sommets. Nous aurions du utiliser des `vector<bool>` pour pouvoir traiter des problèmes de taille arbitraire, mais en perdant la possibilité de faire des opérations binaires.

Nous gardons également en mémoire en permanence, les graphes $G_{comm}(S)$ et $G_{capt}(S)$, encodés comme étant pour chaque sommet, une liste de ses voisins dans le graphe, c'est-à-dire comme un tableau de listes de sommets. Pour être tout à fait rigoureux, plutôt que la structure "list" de la bibliothèque standard, nous avons créé une structure nommée GreedyList constitué d'un tableau pré-alloué associé à un compteur d'éléments. Ainsi, on évite les coûts de désallocation et allocation, principalement lors de la copie d'objets, gagnant un temps non négligeable. Ces graphes sont tout à fait adaptés à la structure du problème et permettent d'être performant lors du calcul d'un nouvel état obtenu par suppression ou ajout d'un capteur. Il suffit pour cela de précalculer les deux graphes suivants :

$$G_{capt} = G_{capt}(V)$$

le graphe de tous les liens de captation possibles, et

$$G_{comm} = G_{comm}(V)$$

le graphe de tous les liens de communication possibles. Un gros avantage de cette méthode est que l'on s'affranchit en partie de la définition initiale du problème. Pour peu que l'on puisse calculer les graphes G_{comm} et G_{capt} , on peut facilement adapter le programme pour travailler sur des grilles (finies) de forme arbitraire, une disposition des points arbitraire, des calculs de distance différents, comme une portée différente suivant la position du capteur etc...

D'un point de vue plus pratique, sont codées les fonctions d'ajout et de suppression de capteur, qui gèrent l'utilisation et l'intégrité des objets mentionnés ci-dessus. On est alors libre d'utiliser ces fonctions et les dits objets pour travailler sur le problème. Notons que la fonction de suppression de capteur est relativement gourmande en calcul, puisqu'elle effectue un parcours en profondeur du graphe $G_{comm}(S)$ pour déterminer quelle est la nouvelle composante connexe du puits.

Nous avons alors essayé deux approches du problème très différentes.

Deuxième partie

Approche par chaînes de destruction

Dans toute cette partie, on illustrera les méthodes présentées sur l'instance (P) du problème : $n = 4$, $R_{capt} = R_{comm} = 1$.

4 Heuristique

4.1 Heuristique déterministe

Dans cette approche, on construit tout simplement une solution S en positionnant des capteurs sur la totalité de la grille, pour avoir une solution réalisable. On tente alors de retirer les capteurs un à un lorsque c'est possible sans compromettre la réalisabilité de la solution. Il s'agit pour cette raison d'une heuristique destructrice. Illustrons la méthode sur (P) :

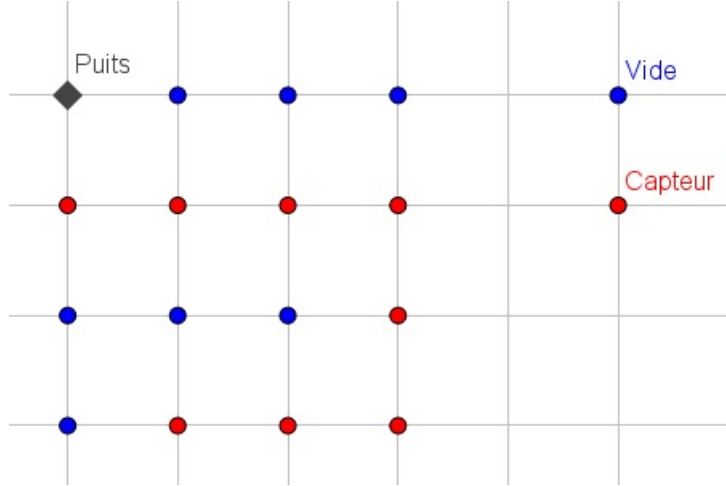


FIGURE 4 – Résultat de l'heuristique de destruction sur (P)

Procédons dans l'ordre, ligne par ligne : on peut enlever la première ligne sans problème, cependant, on doit alors conserver $(2; 1)$ pour ne pas déconnecter le reste des capteurs. On doit également garder le reste de la seconde

ligne pour continuer à capter la première. On peut alors retirer la troisième ligne, à l'exception de (3; 4) qui est alors nécessaire pour connecter la dernière ligne. On peut ensuite enlever (4, 1). On constate qu'on est alors obligés de garder le reste de la dernière ligne.

Si l'on note les sommets par leur indice $k = n(i - 1) + (j - 1)$ plutôt que par leurs coordonnées $(i; j)$, pour plus de concision, on obtient

$$S = \{4, 5, 6, 7, 11, 13, 14, 15\}$$

A première vue, cette heuristique présente de quelques défauts, et assez peu de qualités. Tout d'abord cette heuristique est chronophage : on essaye de retirer un à un les capteurs (initialement au nombre de n^2), ce qui implique d'effectuer un parcours en profondeur des sommets restants pour vérifier la connexité. Le coût de construction d'une telle solution est donc de $O(n^4)$. Deuxièmement, la structure induite par une telle construction est peu désirable en pratique, sauf justement dans le cas $R_{capt} = R_{comm} = 1$, on obtient en général des solutions de qualité médiocre. Elle a toutefois l'avantage d'être implémentée en seulement quatre lignes de code. Elle a également l'avantage de garantir que la solution obtenue est réalisable.

4.2 Heuristique aléatoire

On peut apporter à cette heuristique une très simple amélioration, visant à briser la structure induite par la méthode déterministe. Pour cela, il suffit de remarquer que la solution obtenue est très dépendante de l'ordre dans lequel on a essayé de retirer les sommets (ligne par ligne dans l'exemple). On choisit alors au hasard un ordre dans lequel on va retirer les capteurs, c'est-à-dire une permutation des sommets. Par exemple si on tire au hasard la permutation suivante :

$$0, 10, 11, 12, 13, 5, 8, 15, 9, 6, 4, 3, 14, 1, 2, 7$$

on obtiendra le résultat représenté sur la figure ci-dessous.

Bien sûr, cela ne garantit pas l'obtention d'une meilleure solution mais donne effectivement des solutions meilleures en moyenne que celles de l'heuristique déterministe, pour une instance quelconque. En annexe est fourni un jeu de résultats obtenus sur les instances de test.

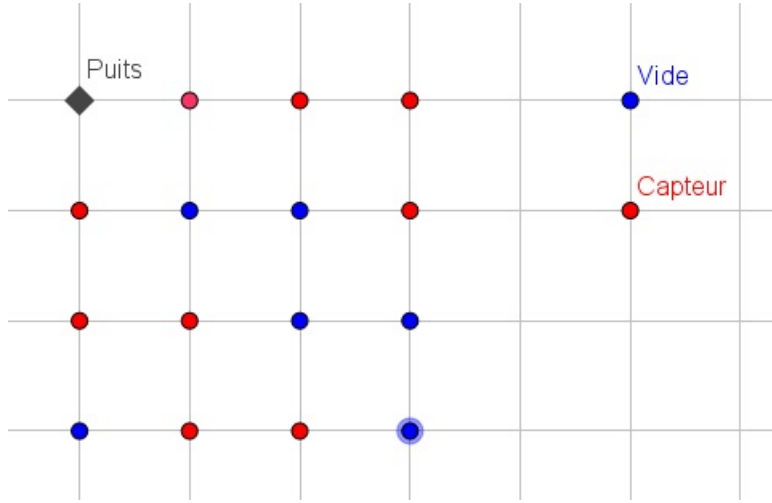


FIGURE 5 – Résultat pour une permutation au hasard

5 Les chaînes de destruction

5.1 Définition

Le procédé décrit ci-dessus a été conçu avec le problème présent en tête, mais on pourrait tout à fait l'appliquer à n'importe quel problème d'optimisation binaire dans lequel on veut minimiser le nombre de bits à 1 dans le vecteur solution, dans le sens où diminuer le nombre de bits à 1 fait décroître la fonction objectif. Ici ce nombre de bits à 1 correspondant directement au nombre de capteurs utilisés. On pourrait aussi adapter le procédé pour la maximisation en partant d'un vecteur nul, et en mettant progressivement des bits à 1. Par exemple, l'algorithme glouton du problème de sac à dos classique consiste en fait à appliquer cette méthode en utilisant une permutation obtenue grâce à un heuristique d'utilité.

Cela permet d'espérer qu'améliorer l'heuristique puisse consister à trouver une "bonne" permutation.

Nous appellerons chaîne de destruction, et on notera l , toute permutation des sommets. On notera $S(l)$ la solution engendrée par l , c'est-à-dire la solution obtenue en essayant de retirer les capteurs d'une grille remplie dans l'ordre de l .

5.2 Propriétés

5.2.1 Chaînes de destruction et solutions non dégénérées

Définitions et énoncé On appelle solution dégénérée une solution dans laquelle il est possible de retirer un capteur sans violer de contrainte. En particulier, toute solution optimale est non dégénérée.

Soit S une solution quelconque, réalisable ou non. On définit alors l^c comme étant la liste dans un ordre arbitraire des indices k pour lesquels un capteur est présent. De même on définit l^e comme étant la liste dans un ordre arbitraire des indices k pour lesquels il n'y a pas de capteur. On note $l^e + l^c$ la concaténation de ces deux listes (avec l^e en premier). Par définition $l^e + l^c$ contient une et une seule fois tous les entier compris entre 0 et $n^2 - 1$, et est donc bien une chaîne de destruction. On observe alors, dans notre problème mais pas nécessairement dans un cas plus large, les propriétés suivantes

Toute chaîne de destruction engendre une solution non dégénérée

$$S = S(l^e + l^c) \Leftrightarrow S \text{ est non dégénérée}$$

Démonstrations Commençons par montrer la première propriété, par l'absurde.

Soit l une chaîne de destruction telle que $S(l)$ soit dégénérée. On appelle k l'indice d'un capteur de $S(l)$ que l'on peut retirer sans violer de contrainte. Puisqu'un capteur est présent en k , cela signifie que lorsqu'on a rencontré k dans la chaîne de destruction, le capteur était inamovible. La contrainte qui aurait été violée est nécessairement la contrainte de connexité. En effet, k est amovible dans $S(l)$, donc k n'est pas nécessaire à la couverture même après avoir enlevé plus de capteurs. On en déduit que k est devenu amovible après avoir retiré la totalité de la composante connexe qu'il aurait déconnecté fût-il enlevé. Soit k' le capteur dont la retraite rendit k amovible, c'est-à-dire le dernier capteur de la dite composante connexe. k est donc nécessaire à ce que k' soit connecté, aucun autre capteur ne communique avec k' . Cependant on sait que k n'est pas nécessaire à la couverture de k' , même après retrait du capteur de k' , c'est-à-dire qu'un capteur en k'' capte k' , sans assurer la connexion de k' . C'est absurde, car $R_{capt} \leq R_{comm}$. $S(l)$ est donc non dégénérée.

On remarque que la propriété cruciale du problème assurant le fonctionnement des chaînes de destruction est

$$G_{capt} \subset G_{comm}$$

Montrons désormais la seconde propriété. Le sens direct est une conséquence immédiate de la première propriété. Il nous faut donc montrer la réciproque.

Expliquons d'abord l'essence de cette propriété. On souhaite montrer que si on dispose d'une solution non dégénérée S , on est capable de construire une chaîne de destruction qui l'engendre. Le raisonnement est le suivant : je part d'une grille remplie, et je souhaite supprimer des capteurs un à un pour aboutir à S , en passant uniquement par des solutions réalisables. Il va de soi que si j'arrive à enlever tous les capteurs de l^e , j'aurais atteint S , et je ne pourrais enlever aucun capteur de l^c puisque S est non dégénérée. Il suffit donc de montrer que quel que soit l'ordre dans lequel j'essaie d'enlever les capteurs de l^e , je ne passerai que par des solutions réalisables. Nous allons pour cela démontrer le cheminement inverse : si je part de S et que j'ajoute les capteurs manquants dans un ordre arbitraire, je ne passerai que par des solutions réalisables. Récursivement, il suffit donc de montrer qu'en ajoutant un capteur à une solution réalisable, j'obtiens encore une solution réalisable.

Soit S une solution réalisable quelconque. Montrons qu'en ajoutant un capteur à S on obtient toujours une solution réalisable (mais nécessairement dégénérée il va de soi).

Trivialement, la contrainte de couverture est toujours vérifiée. En effet, ajouter un capteur à S a pour effet d'ajouter des arêtes dans $G_{capt}(S)$. Ce graphe n'ayant aucun point isolé sauf le puits avant ajout d'arêtes vérifiera toujours la propriété après ajout.

Montrons maintenant que $G_{comm}(S)$ sera toujours connexe après ajout d'un sommet à S . Soit k l'indice du sommet ajouté. S est réalisable donc avant ajout de k , k est déjà couvert, i.e. possède un lien de captation avec un autre capteur. Or, $G_{capt} \subset G_{comm}$, cette même arête connectera donc le nouveau capteur au graphe.

La seconde propriété est donc démontrée.

Remarque : Je remarquais précédemment que notre implémentation permettait de traiter des variantes avec d'autres définitions de distance. On voit ici que cette propriété (et les méthodes qui en découlent) n'est vraie que si à chaque possible lien de captation correspond un lien de communication, i.e. $G_{capt} \subset G_{comm}$. En particulier, il serait impossible de travailler avec cette heuristique si on avait $R_{comm} < R_{capt}$.

On a donc démontré que pour chaque solution non dégénérée (et a fortiori toute solution optimale) on pouvait construire facilement une chaîne de destruction qui la génère, et donc qu'il en existe une. On peut donc en toute légitimité encoder une solution non dégénérée (et donc a fortiori optimale) par une de ses chaînes de destruction. Moralement on démontre que les chaînes de destruction sont adaptés pour résoudre le problème que l'on se pose.

5.2.2 Chaînes équivalentes

Lorsqu'on code une solution quelconque, pas forcément réalisable, avec son vecteur binaire, il découle immédiatement que pour une grille de dimension n , il existe 2^{n^2} solutions possible. De plus, seulement une partie infime de ces solutions, les solutions non dégénérées, peut être associée à une chaîne de destruction.

Or, une chaîne de destruction étant essentiellement une permutation, il existe $n! \gg 2^{n^2}$ chaînes de destruction. On en déduit qu'il existe nécessairement pour chaque chaîne de destruction un nombre colossal de chaînes de destruction équivalentes, c'est-à-dire engendrant la même solution.

En particulier, on a vu que toute chaîne de destruction engendre une solution non dégénérée, et que de cette solution on peut reconstruire une chaîne l'engendrant, en utilisant la partition S et $V \setminus S$, l'ordre étant sans importance.

D'un point de vue encodage d'une solution, on peut donc voir une chaîne de destruction comme une partition des sommets entre S et $V \setminus S$ (il faut cependant appliquer la chaîne de destruction pour connaître la partition qu'elle encode).

6 Voisinage induit

6.1 Définition et propriétés

Lorsque qu'on cherche une structure de voisinage, on cherche un façon qui fait sens de modifier légèrement le vecteur qui encode la solution. Par exemple dans le problème du sac à dos, on peut essayer d'ajouter un objet dans le sac, quitte à appliquer après une heuristique de réparation. Dans notre problème on souhaiterait par exemple déplacer légèrement un capteur, ou en supprimer quelques uns. Il n'est cependant pas facile de faire une heuristique de réparation assez efficace pour avoir une meilleure solution après.

C'est là qu'est la force des chaînes de destruction : en appliquant une légère modification à une chaîne de destruction et en l'appliquant, on obtient une solution S légèrement modifiée, réalisable mais surtout non dégénérée ! Autrement dit, en un sens, on peut facilement créer un voisinage dans le lequel toutes les solutions sont "de qualité".

Le voisinage utilisé par la suite est le suivant : Soit S une solution. On construit les suites l^e et l^c , puis on les mélange aléatoirement. On choisit au hasard un élément de l^c , qu'on retire de l^c pour le mettre au début de l^e . On appelle l'^e et l'^c les suites ainsi obtenues. On construit alors une nouvelle solution S' à partir de $l' = l'^e + l'^c$. En faisant cela, on choisit en fait au hasard

un capteur et on met son indice au début de la chaîne de destruction, on va donc faire en sorte de ne pas mettre de capteur à cet endroit. La nouvelle chaîne l' ne peut donc pas générer S . Au vu de la quantité de permutations équivalentes, cela aurait pu être un risque.

Remarques :

Sans que ce soit gênant, on peut noter que S n'est pas nécessairement dans le voisinage de S' .

Par ailleurs, toujours sans que ce soit gênant, si S est dégénérée, il se peut que $S(l)$ avec $l = l^e + l^c$ ne soit pas "proche" de S . Il va alors de soit que S' n'est pas nécessairement "proche" de S .

Dans toute la suite on travaillera de toute façon uniquement sur des solutions non dégénérées.

6.2 Exemple

Soit la solution non dégénérée obtenue avec l'heuristique déterministe. On rappelle qu'on avait alors

$$l = (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

$$S(l) = \{4, 5, 6, 7, 11, 13, 14, 15\}$$

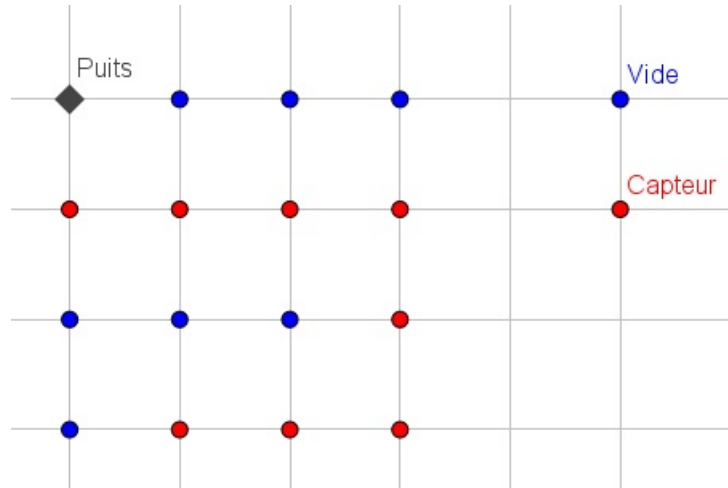


FIGURE 6 – Résultat de l'heuristique de destruction sur (P)

On construit donc

$$l^e = (0, 1, 2, 3, 8, 9, 10, 12)$$

$$l^c = (4, 5, 6, 7, 11, 13, 14, 15)$$

On mélange aléatoirement

$$l^e = (8, 0, 9, 10, 3, 1, 12, 2)$$

$$l^c = (13, 7, 5, 6, 15, 11, 14, 4)$$

On prend un sommet de l^c qu'on met au début de l^e , par exemple 15, on impose alors de ne pas mettre de capteur dans le coin inférieur droit.

$$l'^e = (15, 8, 0, 9, 10, 3, 1, 12, 2)$$

$$l'^c = (13, 7, 5, 6, 11, 14, 4, 12)$$

On obtient alors

$$l' = (15, 8, 0, 9, 10, 3, 1, 12, 2, 13, 7, 5, 6, 11, 14, 4)$$

A première vue, il pourrait sembler que cette chaîne de destruction est très différente de la chaîne ordonnée, pourtant en appliquant la chaîne, on obtient

$$S' = \{4, 5, 6, 7, 10, 13, 14\}$$

Non seulement on est sur une solution très proche, mais elle utilise même un capteur en moins.

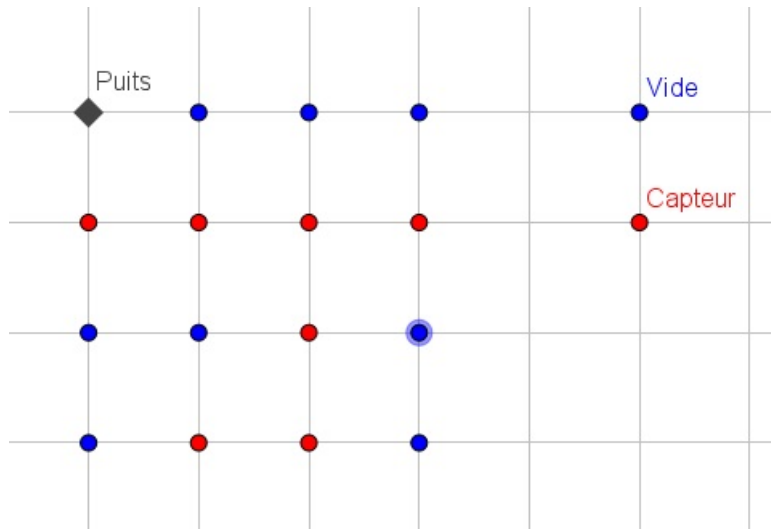


FIGURE 7 – Voisinage de la solution déterministe

Remarques : La taille du voisinage est démesurée, il est déconseillé d'appliquer des technique de parcours méthodique telles que la recherche de voisinages améliorants, ou la recherche avec tabous. Dans la section suivante, on l'utilise dans l'algorithme du recuit simulé.