Rapport de Métaheuristiques

Ronan Le Prat

3 novembre 2025

https://github.com/Renelle29/MH

Résumé

Ce rapport présente les résultats obtenus dans le cadre d'un projet d'implémentation d'heuristiques, descentes, et métaheuristique pour le problème de localisation discrète.

Table des matières

1 Présentation du problème

Dans l'ensemble du rapport on s'intéresse au **Problème de localisation discrète**.
On défini tout au long du problème les paramètres suivants :

$$N \text{ : Nombre de clients}$$
$$M \text{ : Nombre d'usines}$$
$$f_j \text{ : Coût d'installation de l'usine } j$$
$$c_{ij} \text{ : Distance du client } i \text{ à l'usine } j$$

Pour définir le problème, on pose les variables suivantes :

$$y_j \text{ : Variable binaire valant 1 si on ouvre l'usine } j \text{ , et 0 sinon.}$$
$$x_{ij} \text{ : Variable binaire valant 1 si on affecte le client } i \text{ sur l'usine } j \text{ , et 0 sinon.}$$

On formalise le problème de la manière suivante :

$$\begin{aligned}
\min \quad & f(x,y) = \sum_{j=1}^{M} \left(\sum_{i=1}^{N} c_{ij} x_{ij} + f_j y_j \right) & \\
\text{s.c} \quad & \sum_{j=1}^{M} x_{ij} = 1 & i = 1, \dots, N \\
& x_{ij} \leq y_j & i = 1, \dots, N;\ j = 1 \dots M \\
& x_{ij} \in \{0,1\} & i = 1, \dots, N;\ j = 1 \dots M \\
& y_j \in \{0,1\} & i = 1, \dots, N
\end{aligned}$$

1.1 Remarque préliminaire

Dans le cadre du problème sans contraintes de capacités, on peut effectuer le constat suivant :

Étant donné une liste d'usines ouvertes, l'affectation optimale des clients pour cette liste est celle qui affecte chaque client à l'usine ouverte la plus proche.

Ainsi, chercher une solution au problème initial, revient simplement à chercher la liste des usines à ouvrir, i.e. y_{opt} suffit à caractériser la solution optimale.

2 Heuristiques

2.1 Présentations des heuristiques

On présente dans la suite trois heuristiques, par ordre de complexité croissante.

2.1.1 Plus proche usine [PPU]

Pour cette heuristique, on affecte simplement chaque client à l'usine la plus proche, et on ouvre toutes les usines qui ont un client affecté.

$$x_{ij} = 1 \text{ si } j = argmin_{j=1...M}\ cij$$

2.1.2 Construction Gloutonne [CG]

Pour cette heuristique, on choisit d'abord d'ouvrir une seule usine, celle dont le coût total est minimum lorsque lui affecte tous les clients.

$$y_j = 1 \text{ si } j = argmin_{j=1...M} \sum_{i=1}^{N}(c_{ij}) + f_j$$

On itère ensuite sur toutes les autres usines non-encore ouvertes, et on regarde si l'ouvrir permet de réduire le coût, auquel cas on l'ouvre.

2.1.3 Couverture Minimum Itérée [CMI]

Couverture Minimum

Pour cette heuristique, on va transformer notre problème initial en un problème de couverture. On rappelle la définition d'un problème de couverture ci-dessous. On défini les paramètres suivants :

I : Liste d'objets

J : Nombre d'ensembles d'objets

S_j : Ensemble d'objets j

c_j : Coût de l'ensemble d'objets j

On défini les variables suivantes :

x_j : Variable binaire valant 1 si on sélectionne l'ensemble j , et 0 sinon.

On peut formaliser le problème de la manière suivante :

$$\begin{aligned} \min \quad & \sum_{j \in J} c_j\, x_j \\ \text{s.c} \quad & \sum_{j : i \in S_j} x_j \geq 1, \qquad \forall i \in I \\ & x_j \in \{0,1\}, \qquad \forall j \in J \end{aligned}$$

Le problème de couverture de poids minimum est NP-Difficile, on utilise donc une heuristique afin d'obtenir une solution approchée. On utilise l'heuristique gloutonne qui consiste à sélectionner successivement les ensembles dont le coût par objet ajouté est le plus faible. C'est une heuristique à garanti de performance de facteur : $H_{|I|}$ - $|I|$-ème terme de la série harmonique. Asymptotiquement, c'est donc une

heuristique qui donne une solution de valeur au pire $\log(|I|)$ plus grande.

Transformation

Pour transformer le problème, on intègre un paramètre supplémentaire d_{max}, et pour chaque usine on considère l'ensemble S_j constitué des clients se trouvant à une distance au plus d_{max} de l'usine j. On pose pour chaque S_j son coût $c_j = f_j$. On trouve (si elle existe) une couverture des clients via l'heuristique ci-dessus, et on ouvre les usines associées aux ensembles couvrants. On trouve ensuite la valeur de la solution obtenue, en assignant chaque client à l'usine la plus proche parmi celles qui sont ouvertes.

Itérations

Pour obtenir une solution de bonne valeur, il faut appliquer l'heuristique pour différentes valeurs de d_{max}. On tire aléatoirement N_{max} valeurs parmi l'ensemble des distances (c_{ij}), on obtient une solution heuristique comme décrit ci-dessous pour chacune de ces valeurs, et on garde finalement la meilleure.

2.2 Évaluation des heuristiques

On présente les résultat obtenu pour chacune des instances et des heuristiques dans le tableau suivant :

Instance		PPU	PPU	PPU	CG	CG	CG	CMI	CMI	CMI
	Optimum	*Valeur*	*Temps*	*Gap*	*Valeur*	*Temps*	*Gap*	*Valeur*	*Temps*	*Gap*
cap71	932615.750	**950470.188**	**0.00**	**1.9**	1055018.062	0.00	13.1	1003841.375	0.02	7.6
cap72	977799.400	1025470.188	0.00	4.9	1070568.000	0.00	9.5	**1013841.375**	**0.00**	**3.7**
cap73	1010641.45	1100470.188	0.00	8.9	2472891.925	0.00	144.7	**1023841.375**	**0.00**	**1.3**
cap74	1034976.97	1212970.188	0.00	17.2	2480391.925	0.00	139.7	**1038841.375**	**0.01**	**0.4**
cap101	796648.437	832291.150	0.00	4.5	**816027.763**	**0.00**	**2.4**	916164.350	0.02	15.0
cap102	854704.200	952291.150	0.00	11.4	**866731.700**	**0.00**	**1.4**	926164.350	0.02	8.4
cap103	893782.112	1072291.150	0.00	20.0	**913719.088**	**0.00**	**2.2**	936164.350	0.02	4.7
cap104	928941.750	1252291.150	0.00	34.8	978152.875	0.00	5.3	**951164.350**	**0.02**	**2.4**
cap131	793439.562	991571.450	0.00	25.0	**831980.438**	**0.00**	**4.9**	916164.350	0.03	15.5
cap132	851495.325	1236571.450	0.00	45.2	**900749.550**	**0.00**	**5.8**	926164.350	0.03	8.8
cap133	893076.712	1481571.450	0.00	65.9	**903785.975**	**0.00**	**1.2**	936164.350	0.03	4.8
cap134	928941.750	1849071.450	0.00	99.1	**943616.488**	**0.00**	**1.6**	951164.350	0.03	2.4
capa	17156454.478	1.8e8	0.00	956.2	20953628.890	0.01	22.1	**17869311.014**	**0.87**	**4.2**
capb	12979071.582	7.5e7	0.00	478.8	15855952.171	0.01	22.2	**13581464.696**	**1.13**	**4.6**
capc	11505594.329	5.5e7	0.00	386.2	12693322.974	0.01	10.3	**11701680.498**	**0.97**	**1.7**

Les deux heuristiques [CG] et [CMI] semblent compétitives et renvoie chacune les meilleurs résultats sur 7 des 15 instances. En revanche l'heuristique [PPU], trop naïve, ne renvoie la meilleure valeur que sur la première instance.

Statistiques d'évaluation des heuristiques	PPU	CG	CMI
Durée d'exécution moyenne (s)	0.00	0.00	0.23
Erreur moyenne (%)	143.99	25.76	5.70
Ecart-type sur l'erreur (%)	267.34	47.76	4.61
Erreur médiane (%)	24.97	5.78	4.64
Erreur minimum (%)	1.91	1.20	0.37
Erreur maximum (%)	956.16	144.69	15.47

L'heuristique [CMI] est celle qui se comporte le mieux de manière générale, elle renvoie toujours une valeur relativement proche de l'optimum, là ou il arrive parfois que [CG] renvoie une valeur très éloignée de l'objectif (par exemple pour *cap73* et *cap74*).

3 Voisinages

3.1 k-Hamming [k-H]

Pour un entier k fixé, et un vecteur $y \in \{0,1\}^M$, on considère le voisinage $V_k(y)$ constitué de l'ensemble des vecteurs que l'on peut obtenir en complémentant k-bits de y.

C'est-à-dire qu'à partir d'une liste d'usines ouvertes, on choisit d'ouvrir ou de fermer k usines.

4 Descentes

On présente ci-dessous les résultats pour trois méthodes de descentes, à partir du voisinage présenté ci-dessous.

4.1 Présentation des descentes

4.1.1 Exploration complète d'un voisinage [C1-H]

Dans cette méthode, on se contente d'explorer un seul voisinage à partir d'une solution initiale, le voisinage 1-Hamming. On cherche la meilleure solution au sein de ce voisinage, puis on ré-itère à partir de cette nouvelle solution jusqu'à ce qu'on ne trouve plus de meilleure solution.

4.1.2 Exploration complète de plusieurs voisinages [CP-H]

On propose ici une méthode à voisinage variables, à exploration complète de voisinage. Le fonctionnement est le même que pour [C1-H] hormis le fait qu'on va désormais explorer successivement les voisinage 1-Hamming, 2-Hamming et 3-Hamming, jusqu'à ce qu'on ne trouve plus de meilleure solution.

On s'arrête à 3, car au-delà l'espace des solutions explose combinatoirement, et il n'est pas possible en pratique, d'explorer suffisamment rapidement des voisinages pour $k > 3$.

4.1.3 Exploration aléatoire de plusieurs voisinages [AP-H]

Pour palier au problème de l'explosion combinatoire évoquée ci-dessus. On définit une valeur tps_{max} correspondant à la durée d'exécution de l'algorithme sans amélioration. On va générer pour chaque voisinage un nombre $K_{voisins}$ d'éléments dans chaque voisinage, et évaluer la valeur de la solution pour chacun de ces éléments. Dès qu'on trouve un meilleur élément, on recommence à partir du premier voisinage et on remet le chronomètre à 0. On s'arrête quand on arrive à cours de temps.

4.2 Évaluation des descentes

Pour l'évaluation ci-dessous, on fait tourner en parallèle trois descentes à partir de chacune des valeurs obtenues par les trois heuristiques ci-dessus. On renvoie le meilleur des trois résultats, et le temps mis pour obtenir solution (ou pour fermer le gap avec l'optimum).

Instance		C1-H	C1-H	C1H	CP-H	CP-H	CPH	AP-H	AP-H	APH
	Optimum	*Valeur*	*Temps*	*Gap*	*Valeur*	*Temps*	*Gap*	*Valeur*	*Temps*	*Gap*
cap71	932615.750	932615.750	0.01	0.0	932615.750	0.01	0.0	932615.750	0.35	0.0
cap72	977799.400	977799.400	0.01	0.0	977799.400	0.01	0.0	977799.400	0.18	0.0
cap73	1010641.45	1010641.450	0.01	0.0	1010641.450	0.00	0.0	1010641.450	0.58	0.0
cap74	1034976.97	1034976.975	0.00	0.0	1034976.975	0.00	0.0	1034976.975	0.21	0.0
cap101	796648.437	796648.438	0.01	0.0	796648.438	0.24	0.0	796648.438	5.93	0.0
cap102	854704.200	854704.200	0.00	0.0	854704.200	0.00	0.0	854704.200	0.32	0.0
cap103	893782.112	893782.113	0.00	0.0	893782.113	0.12	0.0	893782.113	5.47	0.0
cap104	928941.750	**934586.975**	**0.01**	**0.6**	928941.750	0.02	0.0	928941.750	0.97	0.0
cap131	793439.562	793439.563	0.09	0.0	793439.563	1.04	0.0	793439.563	11.31	0.0
cap132	851495.325	851495.325	0.10	0.0	851495.325	0.13	0.0	851495.325	6.22	0.0
cap133	893076.712	893076.713	0.02	0.0	893076.713	1.02	0.0	**893782.113**	**13.14**	**0.1**
cap134	928941.750	**934586.975**	**0.11**	**0.6**	928941.750	0.11	0.0	928941.750	4.18	0.0
capa	17156454.47	**1.7413e7**	**0.05**	**1.5**	17156454.478	2.00	0.0	**17604051.07**	**7.45**	**2.6**
capb	12979071.58	**1.3071e7**	**0.10**	**0.7**	12979071.581	30.43	0.0	**13418881.27**	**6.92**	**3.4**
capc	11505594.32	**1.1606e7**	**0.05**	**0.9**	**11509361.66**	**85.79**	**0.0**	**11632123.46**	**6.91**	**1.1**

Sans grande surprise, la méthode de descente qui résout le moins d'instances à l'optimum est [C1-H]. En effet, il s'agit d'une méthode pour laquelle on se contente d'explorer de petits voisinages, ce qui rend la possibilité finale de fermer le gap entre l'optimum et la solution obtenue difficile.

La méthode [AP-H] ne semble pas particulièrement bien fonctionner sur ce problème (en tout cas avec l'implémentation proposée). On constate en effet, que sur les instances les plus difficiles, pour lesquelles on s'attendrait à ce que cette méthode soit la plus performante, on obtient au contraire les pires résultats des trois méthodes. Cela peut s'expliquer par plusieurs raisons :

— La valeur de t_{max} choisie est de 5 secondes, ce qui peut s'avérer trop faible pour explorer suffisamment les voisinages.

— On met autant d'effort à explorer les proches voisinages, que les voisinages plus éloignés. Il pourrait être judicieux de commencer par explorer de manière importante les voisinages proches, avant d'étendre à d'autres voisinages une fois que la solution n'a pas été améliorée depuis un certain temps.

La méthode [CP-H] est globalement très performante sur toutes les instances fournies. Elle parvient à fermer le gap sur toutes les instances, à l'exception de la dernière (pour laquelle le gap final est très faible - environ 0.04% d'erreur). Les temps d'exécution restent raisonnables, même sur les instances les plus grandes (en sachant que sur ces grandes instances, on effectue une recherche complète. On pourrait s'arrêter avant la fin de la recherche et obtenir une solution intermédiaire moins bonne - mais au moins aussi bonne que celle de [C1-H] pour la même durée d'exécution).

Statistiques d'évaluation des heuristiques	C1-H	CP-H	AP-H
Durée d'exécution moyenne (s)	0.04	8.06	4.68
Erreur moyenne (%)	0.29	0.00	0.48
Ecart-type sur l'erreur (%)	0.46	0.01	1.07
Erreur médiane (%)	0.00	0.00	0.00
Erreur minimum (%)	0.00	0.00	0.00
Erreur maximum (%)	1.50	0.03	3.39

On retrouve de manière agrégée les résultats discutés précédemment. Parmi les trois méthodes proposées, [CP-H] est celle qu'il est préférable de choisir pour le problème de location discrète non-contraint.

5 Métaheuristique

5.1 Recuit Simulé [RC]

On présente ci-dessous l'implémentation d'une méta-heuristique : le recuit simulé. Au vu des résultat obtenu par la descente [CP-H], on s'attend difficilement à obtenir de meilleurs résultat avec cette métaheuristique sur les instances proposées. On pourra néanmoins comparer les performances de cette métaheuristique avec celles de [AP-H].

5.2 Évaluation de la métaheuristique

On évalue le recuit simulé avec les paramètres suivants :

— Température initial :

$$T_{init} = \frac{max(max_{ij}(c_{ij}), max_j(f_j))}{1000}$$

— Coefficient de diminution de la température :

$$\alpha = 0.999$$

— Temps maximum d'exécution depuis la dernière amélioration :

$$\Delta_{max} = 5\,\text{sec}$$

Instance Name		RC	RC	RC
	Valeur Optimale	*Valeur*	*Temps (sec)*	*Gap (%)*
cap71	932615.750	932615.750	0.00	0.0
cap72	977799.400	977799.400	0.00	0.0
cap73	1010641.450	1010641.450	0.00	0.0
cap74	1034976.975	1034976.975	0.00	0.0
cap101	796648.437	796648.438	5.12	0.0
cap102	854704.200	854704.200	0.00	0.0
cap103	893782.112	893782.113	5.03	0.0
cap104	928941.750	928941.750	0.02	0.0
cap131	793439.562	793439.563	5.05	0.0
cap132	851495.325	851495.325	0.02	0.0
cap133	893076.712	**893732.975**	**5.04**	**0.1**
cap134	928941.750	928941.750	0.01	0.0
capa	17156454.478	**17413325.081**	**5.02**	**1.5**
capb	12979071.582	**13103172.562**	**5.06**	**1.0**
capc	11505594.329	**11605955.742**	**5.02**	**0.9**

La méthode se révèle globalement performante, et s'exécute rapidement avec les paramètres utilisés. Elle ne parvient pas à renvoyer la solution optimale pour 4 des instances les plus difficiles.

Si on compare par rapport aux méthodes précédentes, on constate que :

— La méthode fonctionne mieux que [AP-H], ce qui semble confirmer que considérer les voisinages proches est préférable pour ce problème.
— La méthode est compétitive avec [C1-H], avec un profil de solution a peu près similaire. C'est assez logique, puisqu'une fois que la température a suffisamment diminué, [RC] devient une simple exploration du voisinage 1-hamming de la solution courante.
— Les performances sont plus faibles par rapport à [CP-H], qui reste la méthode la plus performante jusqu'à présent.

Statistiques d'évaluation de la métaheuristique	RC
Durée d'exécution moyenne (s)	2.36
Erreur moyenne (%)	0.23
Ecart-type sur l'erreur (%)	0.47
Erreur médiane (%)	0.00
Erreur minimum (%)	0.00
Erreur maximum (%)	1.50

En moyenne, les résultats de [RC] sont légèrement meilleurs que ceux de [C1-H], mais la méthode prend un peu plus de temps à s'exécuter. Pour améliorer les performances du recuit simulé, on pourrait envisager les pistes suivantes :

— Lancer plusieurs recuits simulés, avec des paramètres initiaux différents, et garder la meilleure solution.
— Augmenter la valeur de température, dès lors que la température a trop chuté et que l'on ne trouve plus de meilleure solution.
— Choisir un voisin parmi des voisinages de taille plus grande (par un exemple un voisin 1/2/3-hamming).

6 Conclusion

Durant ce projet, j'ai proposé différentes heuristiques, méthodes de descentes et métaheuristique, afin d'obtenir une bonne solution au problème de localisation sans capacité.

Deux des trois heuristiques proposées sont performantes, [CG] et [CMI], et permettent d'obtenir rapidement de bonnes solutions initiales pour différentes instances du problème. On a pu constater, qu'on ne sait pas à priori quelle heuristique va retourner la meilleure valeur pour une instance donnée. Ainsi, en pratique, il est intéressant d'exécuter différentes heuristiques en parallèles, afin de maximiser les chances d'obtenir une (ou plusieurs) bonnes solutions initiales, et espérer se rapprocher ainsi de la solution optimale (ou à défaut d'un bon optimum local).

Le voisinage, et les méthodes de descentes associées, ont montré de bonnes performances, en particulier la méthode de descente [CP-H]. Les méthodes [C1-H] et [CP-H] permettent de mettre en avant un compromis entre le temps d'exécution d'une méthode, et la qualité de la solution obtenue. Les moins bonnes performances de [AP-H] montrent que pour le problème de localisation sans capacité, on n'a pas intérêt à explorer des voisinages trop éloignés de la solution courante (car il devient très difficile d'exhiber une meilleure solution dans de tels voisinages).

Enfin la métaheuristique implémentée, le recuit simulé, permet de dégrader la solution courante dans l'espoir d'obtenir une meilleure solution plus tard. Les résultats obtenues pour cette méthode sont prometteurs, et pourraient, sur des instances très difficiles, permettre d'obtenir de meilleures solutions que les méthodes de descente précédentes, notamment en lançant plusieurs recuits en parallèle, ou avec une gestion dynamique de la température. La difficulté de la méthode réside dans le choix des paramètres initiaux (qui influent grandement sur la qualité de la solution finale), et son caractère non-déterministe.