MultiObjective MetaHeuristics
MultiObjective Two-Stage Uncapacitated Facility Location Problems

Adrien Callico, Sullivan Bitho

Université de Nantes — LINA, UMR CNRS 6241
UFR Sciences – 2 rue de la Houssinière BP92208, F44322 Nantes cédex 03 – France

---

Table des matières

1. Introduction

Le problème de localisation de services en deux phases sans contrainte de capacité (*TSUFLP*) est une variante du problème de localisation de services. Il consiste, dans un premier temps, à ouvrir un sous ensemble de sites de niveau 1 auxquels des clients seront affectés (phase 1). Dans un second temps, on sélectionne un sous ensemble de sites de niveau 2 auxquelles les sites de niveau 1 ouverts seront affectés (phase 2). L'objectif est initialement de minimiser les coûts d'ouvertures et d'affectations. Le problème existe sous diverses variantes et les contraintes varient en fonction. Les variantes impliquent par exemple l'ajout d'un second objectif, l'ajout de contraintes comme les capacités maximales d'affectations.

L'organisation de cette archive ainsi que les instructions de test sont détaillées dans le fichier `README.txt`.

2. Problème traité

Dans cet exercice, nous traitons la variante du problème TSUFLP avec une seconde fonction objectif dite *pull*. Cette variante inclut également une contrainte de capacité Q sur les concentrateurs de niveau 1. Son modèle (basé sur le modèle de *Chardaire et al* [1]) est le suivant :

Données :

- $I = \{1, \ldots, nI\}$, l'ensemble des emplacements des terminaux ;
- $J = \{1, \ldots, nJ\}$, l'ensemble des emplacements possibles des concentrateurs de niveau 1 ;
- $K = \{1, \ldots, nK\}$, l'ensemble des emplacements possibles des concentrateurs de niveau 2 ;
- $Q \in \mathbb{N}$, $Q \geq 1$, est la capacité des concentrateurs de niveau 1 ;
- c_{ij} est le coût d'assignation du terminal $i \in I$ au concentrateur de niveau 1 $j \in J$;
- d_{ij} est la distance entre le terminal $i \in I$ et le concentrateur de niveau 1 $j \in J$;
- b_{jk} est le coût d'installation d'un concentrateur de niveau 1 $j \in J$ et de le connecter au concentrateur de niveau 2 $k \in K$;
- s_k est le coût d'installation d'un concentrateur de niveau 2 $k \in K$.

Variables de décision :

- $x_{ij} \in \{0, 1\}$, $i \in I$, $j \in J$ est égal à 1 si le terminal $i \in I$ est affecté au concentrateur de niveau 1 $j \in J$, 0 sinon ;
- $y_{jk} \in \{0, 1\}$, $j \in J$, $k \in K$ est égal à 1 si le concentrateur de niveau 1 $j \in J$ est connecté au concentrateur de niveau 2 $k \in K$, 0 sinon ;
- $z_k \in \{0, 1\}$, $k \in K$ est égal à 1 si le concentrateur de niveau 2 $k \in K$ est ouvert, 0 sinon.

Modèle :

$$\min \quad f^1(x,y,z) = \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{j\in J}\sum_{k\in K} b_{jk}y_{jk} + \sum_{k\in K} s_k z_k \tag{1}$$

$$\min \quad f^2(x) = \max\{\, d_{ij}x_{ij} \mid i \in I,\ j \in J\} \tag{2}$$

$$\text{s/t} \quad \sum_{j\in J} x_{ij} = 1 \qquad \forall i \in I \tag{3}$$

$$x_{ij} \le \sum_{k\in K} y_{ik} \qquad \forall i \in I,\ \forall j \in J \tag{4}$$

$$y_{jk} \le z_k \qquad \forall j \in J,\ \forall k \in K \tag{5}$$

$$\sum_{k\in K} y_{jk} \le 1 \qquad \forall j \in J \tag{6}$$

$$\sum_{i\in I} x_{ij} \le Q \qquad \forall j \in J \tag{7}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I,\ \forall j \in J \tag{8}$$

$$y_{jk} \in \{0,1\} \qquad \forall j \in J,\ \forall k \in K \tag{9}$$

$$z_k \in \{0,1\} \qquad \forall k \in K \tag{10}$$

(1) minimise les coûts
(2) minimise la distance maximale entre les terminaux et les concentrateurs de niveau 1 ouverts
(3,4) chaque terminal est assigné à exactement un concentrateur de niveau 1, qui est lui-même assigné à un concentrateur de niveau 2
(5,6) chaque concentrateur de niveau 1 est assigné à au plus un concentrateur de niveau 2
(7) chaque concentrateur de niveau 1 est relié à au plus Q terminaux.
(8,9,10) les variables sont binaires.

3. Scatter Search

Scatter Search est une métaheuristique à population originellement proposée ([2]) pour les problèmes mono-objectifs. Ici, nous avons mis en place une version multi-objectif proposée en 2021 ([3]).
Scatter Search est divisée en cinq étapes distinctes, décrites ci-après. Nous ne rappellerons pas en détails les principes de GRASP, Tabu Search et Path Relinking, mais seulement les choix que nous avons fait dans leurs mises en place.

3.1. Diversification

La première étape consiste à créer rapidement une population initiale, à la fois diversifiée et d'assez bonne qualité : GRASP était tout indiqué.

Nous avons fixé l'hyper-paramètre $\alpha = 0.4$ pour un bon rapport diversité/qualité. Pour la construction gloutonne d'une solution, nous choisissons à chaque itération un concentrateur de niveau 1 au sens de GRASP (un concentrateur au hasard parmi la Restricted List), on lui affecte les Q meilleurs terminaux au sens de la fonction objectif, et on lui affecte le concentrateur de niveau 2 le moins coûteux (prenant en compte le fait que chaque concentrateur de niveau 2 soit ouvert ou non). Nous itérons ainsi jusqu'à ce que tous les terminaux soient affectés (soit $\lceil \frac{nI}{Q} \rceil$) itérations). Pour accélérer le choix des terminaux à affecter, nous trions

au préalable les terminaux par distance et par coût pour chaque concentrateur de niveau 1.

Comme Sanchez le suggère, nous avons défini 2 fonctions d'utilité, g_1 et g_2 qui, pour chaque objectif, calcule la variation de la valeur de l'objectif pour cette solution si on lui ajoute ce concentrateur.
Ce que nous avons décidé pour diversifier au maximum la population est de ne pas seulement créer des solutions "lead" par chacun des objectifs, mais selon 5 combinaisons linéaires des objectifs :

$$f_\lambda = \lambda f^1 + (1-\lambda) f^2 \ \lambda \in \{1, 0.75, 0.5, 0.25, 0\}$$

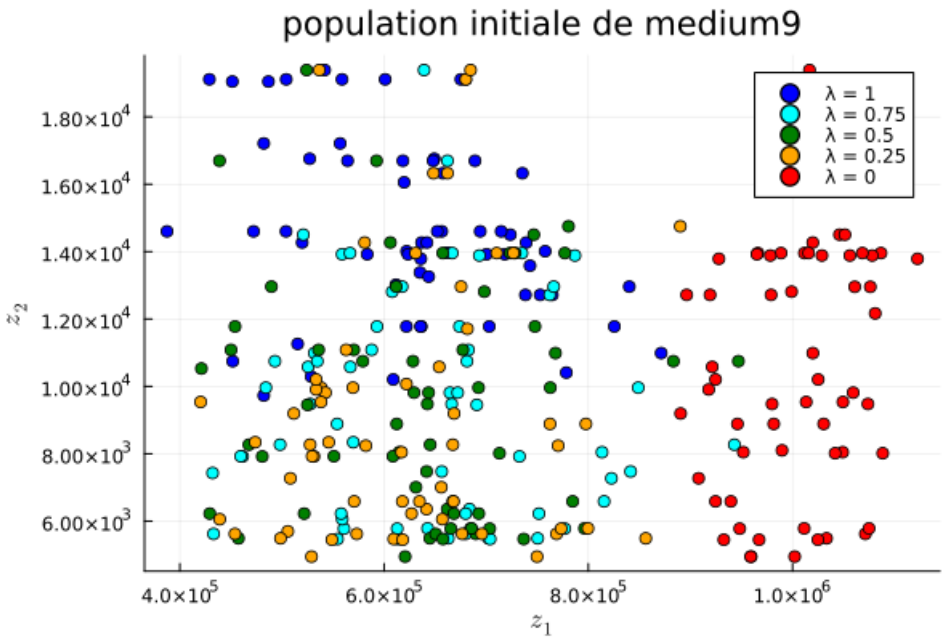


FIGURE 1 – Population initiale

On remarque les combinaisons linéaires permettent bien d'aller chercher des solutions de compromis intéressantes. Evidemment, on stocke tout de suite les solutions non-dominées dans une archive de type SkipList.
Enfin, la génération de 300 solutions prend moins d'une seconde pour les instances small et medium, quelques secondes pour les instances larges.

3.2. Intensification

La méthode d'intensification suggérée par Sanchez et que nous avons mise en place est la méthode de recherche locale Tabu Search.

Trois paramètres sont à décider pour cette métaheuristique : la taille et la nature de la liste tabou, le critère d'arrêt, et les mouvements définissant le voisinage.

— Nous avons décidé d'avoir une taille de liste tabou petite afin d'éviter les cycles. Pour chaque instance, elle est fixée à $\lceil 0.1 \times nJ \rceil$ itérations consécutives sans améliorer la meilleure solution rencontrée. Notre liste tabou stock des mouvements tabous.

— le critère d'arrêt est, selon un paramètre $k \in [0, 1]$, $k \times nJ$. Dans la section résultats, nous discutons de l'intérêt de chacune des valeurs 0.1, 0.5 et 0.9.

Nous avons considéré deux mouvements, "shift" et "swap".

— shift consiste à changer l'affectation d'un terminal vers un autre concentrateur de niveau 1, ouvert ou fermé (respectant la contrainte de capacité)

— swap échange l'affectation de tous les terminaux affecté à un concentrateur de niveau 1 vers un concentrateur jusqu'alors fermé. On ferme ensuite le concentrateur initial et ouvrons le concentrateur de destination.

Ainsi, nous stockons dans notre liste tabou le concentrateur qui vient d'être fermé : on ne pourra le rouvrir qu'après *tenure* itérations. Nous faisons bien sûr les ajustements nécessaires pour les concentrateurs de niveau 2 (les fermer ou les ouvrir si besoin).
Nous avons alors constaté expérimentalement que shift avait un voisinage associé bien trop grand, et ne présentait pas de modification assez significative à la solution. Swap en revanche est bien plus performant sur ces deux points (taille de voisinage correcte et modifie assez la solution).

Enfin, nous appliquons une stratégie de "first improvement" : on parcourt le voisinage en s'arrêtant à la première solution qui améliore l'objectif considéré. Si aucun voisin n'améliore la solution courante, alors on ouvre le concentrateur tabou qui détériore le moins la valeur de l'objectif considéré (si la liste tabou est vide, on prend le voisin qui détériore le moins la valeur de l'objectif considéré).

Cette stratégie de first improvment n'explore pas tout le voisinage, mais elle permet un temps d'exécution de quelques centièmes de secondes sur les instances small et medium.
De plus, nous utilisons le critère d'aspiration suivant : si effectuer un swap avec un concentrateur tabou améliore la valeur de l'objectif considéré de la meilleure solution rencontrée, alors on l'ajoute à la solution.

Nous appliquons la recherche tabou une première fois sur la population initiale créée avec GRASP : la première moitié (pour les solutions créées avec $\lambda = 1$,$\lambda = 0.75$ et la moitié de celles avec $\lambda = 0.5$) selon l'objectif 1, et l'autre moitié selon l'objectif 2 :

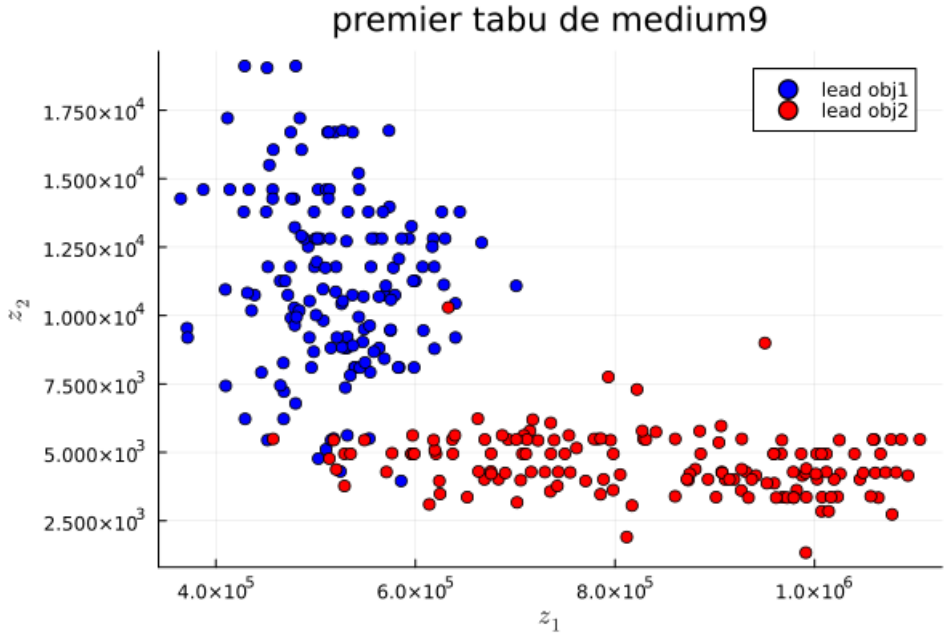


FIGURE 2 – Population initiale améliorée une première fois avec Tabu Search

Les nouvelles solutions obtenues sont ajoutées à notre archive.

3.3. Ensembles de solutions de référence

L'idée centrale de Scatter Search est l'utilisation d'ensembles de solutions de référence. Dans le cas multiobjectif, il y en a un par objectif. Ils sont disjoints et ont une taille fixée β (hyper paramètre) et sont construits de la façon suivante :

— Les $\beta/2$ premières solutions de chaque ensemble de référence sont les solutions de la population qui ont les meilleures valeurs pour l'objectif correspondant.

— les $\beta/2$ autres solutions sont les solutions les plus éloignées de celles déjà présentes dans l'ensemble de référence, au sens de la distance suivante :

$$d(S_i, RefSet) = \min_{S_j \in RefSet} d(S_i, S_j)$$

où $d(S_i, S_j)$ est le nombre de concentrateurs de niveau 1 et 2 qui sont ouverts dans S_i mais pas dans S_j, et vice-versa.

3.4. Combinaison de solutions

L'intérêt de ces ensembles de références est d'avoir des solutions diversifiées et de bonne qualité pour chacun des objectifs, pour pouvoir effectuer des path-relinking entre chaque paire (S_i, S_j) où S_i n'a jamais été combinée avec S_j dans les itérations précédentes, et où S_i et S_j proviennent d'ensembles de référence différents.
On voit ici toute l'importance de l'hyper paramètre β, car il va directement influer sur le nombre de paires (β^k, où k est le nombre d'objectifs) où l'on va appliquer le path-relinking. Sanchez avait choisi $\beta = 6$ pour avoir 108 paires dans le cas où toutes les solutions de références sont nouvelles (ce qui comparable à l'effort fourni des Scatter Search en mono-objectif). Nous avons alors choisi $\beta = 10$ pour avoir 100 paires, toujours

dans le cas où toutes les solutions de référence sont nouvelles.

Pour le path-relinking, nous procédons comme suit :

— On identifie les concentrateurs de niveau 1 à ouvrir et fermer dans la solution de départ par rapport à la solution d'arrivée. On effectue ensuite des swaps successifs jusqu'à ce qu'il n'y ait plus de concentrateur de niveau 1 à fermer ou ouvrir, en stockant la solution à chaque swap.

— on fait de même avec les concentrateurs de niveau 2

— puis on vérifie les assignations des terminaux aux concentrateurs de niveau 1 : on les réaffecte un par un. Afin de limiter le nombre important de solutions intermédiaires, nous avons décidé de limiter le nombre de solutions archivées lors de chaque étape du path relinking. On archive une solution non pas à chaque réaffectation d'un terminal, mais après en avoir réaffecté la moitié. On ne stocke ainsi qu'une solution à chaque étape du path relinking.

— on fait de même pour la réaffectation des concentrateurs de niveau 1 à ceux du niveau 2.

Ensuite, sur chaque solution intermédiaire stockée, on applique Tabu Search selon chacun des deux objectifs.

On mettra alors les ensembles de référence à jour comme suit :

— pour chaque solution issue des path-relinking et pour chaque refSet, on regarde d'abord si elle ne fait pas partie de l'autre refSet.

— Ensuite, si elle domine au moins une autre solution de son refSet d'origine, alors elle prend la place d'une solution qu'elle domine selon l'objectif du refSet. La solution remplacée est la plus proche au sens de la distance entre deux solutions précédemment définie.

Le Scatter Search s'arrête lorsqu'on ne peut plus mettre à jour les refSets.

3.5. Diversification et intensification supplémentaire

Nos résultats ne présentant pas une diversité satisfaisante (cf section Résultats), nous avons eu l'idée d'ajouter des opérations de "*crossover*" non pas inter-solutions mais intra-solution : cette opération consiste à intervertir la moitié des terminaux affectés à un concentrateur c_1 avec la moitié des terminaux affecté à un autre concentrateur c_2. Nous effectuons ces opérations sur les meilleurs solutions ($\approx 25\%$) obtenues après path relinking. Nous reviendrons sur l'intérêt de ce "crossover" (avec et sans Tabu Search) dans la section Résultats.
Nous avons aussi essayé de ré-appliquer Tabu Search sur les solutions obtenues après crossover si celles-ci étaient meilleures qu'avant crossover selon au moins un des deux objectifs.

3.6. Archivage des solutions

Nous avons créé notre propre structure de stockage des solutions non dominées en implémentant une archive de type SkipList. La SkipList s'avère particulièrement efficace dans un contexte d'archivage de solutions, car les opérations de recherches et de mises à jour (insertions/suppressions) y sont courantes.

4. Résultats numériques

4.1. Environnement et instructions

L'environnement machine dans lequel nous avons réalisé nos tests présente les caractéristiques suivantes :

Système d'exploitation Ubuntu 20.04

Processeur Intel Core™ i5-7300HQ CPU @ 2.50GHz × 4

Carte Graphique Mesa Intel HD Graphics 630 (KBL GT2)

RAM 8 Go

Les instructions de test sont détaillées dans le `README.txt`.

4.2. Construction des instances

Nous nous sommes basés en grande partie sur les instances utilisées par Sanchez, en faisant les ajustements suivants :

- Nous séparons les "facility" de l'instance : les $4/5$ premières facilities seront nos concentrateurs de niveau 1, et les $1/5$ derniers seront nos concentrateurs de niveau 2.
- Nous utilisons la distance $d(x, y) = (x_1 - y_1)^2 + (x_2 - y_2)^2$ pour mesurer les distances entre les terminaux et concentrateurs de niveau 1.
- les coûts b_{jk} seront égaux aux distances entre les concentrateurs de niveau 1 et de niveau 2
- les coûts c_{ij} seront tirés au hasard (seed fixée) entre le minimum et le maximum des distances entre les terminaux et les concentrateurs de niveau 1
- de même pour les coûts z_k, tirés au hasard entre le minimum et le maximum des distances entre les concentrateurs de niveau 1 et 2
- nous arrondissons à l'inférieur tous les coûts et distances pour avoir des coûts entiers, et ainsi utiliser $\epsilon = 1$ dans la résolution avec vOpt (voir ci-après)

Les instances "small" donnent ainsi des instances TSUFLP de taille $nI = 50$, $nJ = 16$ et $nK = 4$, tandis que les instances medium sont de taille $nI = 100$, $nJ = 40$ et $nK = 10$.

Nous avons aussi créé une instance sur une partie de la ville d'Angers, reprenant la méthode de Prof. X. Gandibleux :
Nous avons converti les coordonnées des terminaux pour pouvoir calculer les distances, les coordonnées du dataset [1] étant en WGS 84 / PseudoMercator, srid 3857, et ceux du site `http://overpass-turbo.eu/` en latitude/longitude. La méthode de génération des coûts est la même que les instances de Sanchez, et nous avons multiplié par 1000 les distances car celles-ci étaient en km (et nous avions besoin de les arrondir pour pouvoir utilier l'ϵ-contrainte avec $\epsilon = 1$).

1. `https://www.data.gouv.fr/fr/datasets/ma-connexion-internet/`

4.3. vOpt

Pour évaluer la qualité de nos résultats, nous avons choisi de nous comparer avec l'ensemble Y_N calculé par vOpt Solver [4]avec la méthode ϵ-contrainte, et le solveur Gurobi. Pour ce faire, nous avons linéarisé le modèle de départ ainsi :

— ajout d'une variable réelle $Z \geq 0$;

— remplacement de f^2 par $f_L^2 = \min Z$

— ajout des contraintes $Z \geq x_{ij} d_{ij},\ \forall i \in I,\ \forall j \in J$

Les moyennes des temps de résolution par vOpt avec Gurobi en ϵ-contrainte sont les suivantes :

— sur les instances small : 219.390 secondes

— sur les instances medium : 1881.351 secondes

4.4. Choix des hyper-paramètres

— β : Pour avoir un maximum de diversité sans exploser le temps de calcul, nous avons suivi l'avis de Sanchez et pris $\beta = 10$ pour avoir une centaine de paires de solutions au début.

— Critère d'arrêt pour la recherche tabou : Nous avons suivi Sanchez sur le fait d'avoir un nombre maximal d'itérations pour la recherche tabou, et de l'indexer selon le nombre de concentrateurs de niveau 1. Nous avons testé un nombre maximal d'itérations égal à $k \times nJ$ pour $k \in \{0.1, 0.5, 0.9\}$

— Nous avons décidé d'avoir une liste tabou de petite taille, mais qui soit elle aussi indexée sur le nombre de concentrateurs de niveau 1. Nous l'avons alors fixée à $0.1 \times nJ$.

4.5. Etude du paramètre k

Nous avons étudié l'influence de k sur :

— les temps de résolution des instances

— la couverture de dominance qu'il apporte par rapport à une autre valeur de k

Ci-dessous les résultats de nos tests pour montrer l'influence du paramètre k sur les temps de résolution des instances :

k	**taille instance**	**temps resolution**
0.1	small	2.895
	medium	76.445
0.5	small	7.310
	medium	113.371
0.9	small	11.807
	medium	189.643

TABLE 1 – Influence du paramètre k sur les temps de résolution

Ci-dessous les résultats de nos tests sur l'influence du paramètre k sur la couverture de dominance :

taille instance	k1	k2	k1 domine k2 sur (%) de solution
small	0.9	0.5	41.734
	0.9	0.1	46.561
	0.5	0.1	38.658
medium	0.9	0.5	37.039
	0.9	0.1	30.501
	0.5	0.1	40.031

TABLE 2 – Influence du paramètre k sur la mesure de couverture

On constate qu'une valeur de k élevée ne fournit pas un ensemble Y_N qui domine largement les autres.

4.6. Intérêt du "crossover"

Nous nous sommes rendus compte que, depuis leur affectation pendant GRASP, les terminaux étaient groupés par "blocs" à un concentrateur de niveau 1 (qu'on changeait via les swaps) et ces blocs n'étaient jamais scindés. Cela restreignait la diversité de nos solutions, et pour y pallier, nous nous sommes inspirés du "crossover" des algorithmes génétiques :

— Pour les meilleures solutions ($\approx$ 10 à 25% pour chaque objectif) obtenues après les path-relinking, nous sélectionnons quelques concentrateurs de niveau 1 (environ 20%) et nous créons deux nouvelles solutions, en interchangeant pour chaque paire de ces concentrateurs la première moitié (resp. la deuxième moitié) des terminaux affectés à l'un et l'autre de ces concentrateurs.

— Nous avons aussi tenté d'appliquer la recherche tabou sur ces nouvelles solutions si elles sont prometteuses (si la valeur d'une des deux fonctions objectifs est meilleure que la solution de départ).

Pour chaque valeur de k, nous avons essayé les trois stratégies sur chacune des instances :

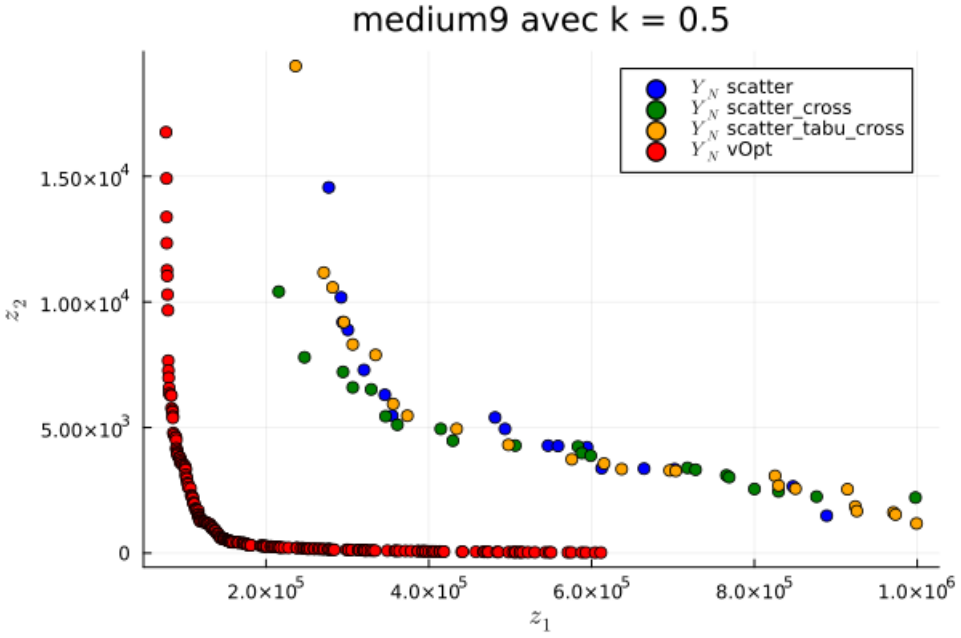


FIGURE 3 – Comparaison des différentes stratégies

Nous allons comparer ces trois stratégies entre elles selon trois critères :

— le temps moyen d'exécution :

— la distance moyenne des points à l'ensemble Y_N calculé par vOpt ;

— les coverage measures

Temps d'exécution (pour $k = 0.5$)

Stratégie	**taille instance**	**temps résolution**
Scatter Search	small	7.31
	medium	113.370
"crossover"	small	7.542
	medium	133.335
"crossover" + tabu	small	8.164
	medium	148.132

TABLE 3 – Temps de résolution pour les différentes stratégies

On remarque que le temps de résolution n'explose pas : il reste stable pour les instances small, et augmente légèrement pour les instances medium.

Distances moyennes aux Y_N calculés par vOpt (pour $k = 0.5$) :

Stratégie	**taille instance**	**distance moyenne à Y_N ($\times 10^{11}$)**
Scatter Search	small	0.671
	medium	2.691
"crossover"	small	0.846
	medium	3.138
"crossover" + tabu	small	0.748
	medium	2.479

TABLE 4 – Distances moyennes aux Y_N pour les différentes stratégies

On peut alors constater que :

— sur les instances small, le crossover avec ou sans tabu n'est pas intéressant

— sur les instances medium, le crossover est intéressant, mais seulement couplé au tabu

— le crossover seul n'est jamais pertinent.

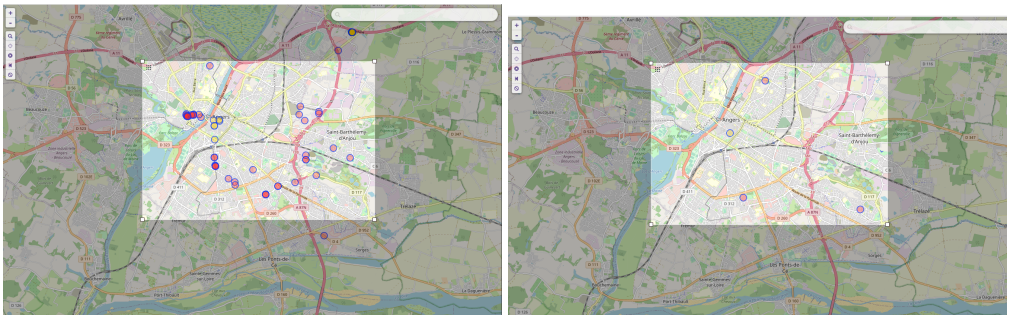
Moyennes des coverage measures entre les stratégies (pour $k = 0.5$) :

Taille instance	Stratégie 1	Stratégie 2	Strat 1 $\preceq$ Strat 2 (%)
small	crossover + tabu	crossover	34.988
	crossover + tabu	SS	24.925
	crossover	SS	34.083
medium	crossover + tabu	crossover	49.548
	crossover + tabu	SS	45.792
	crossover	SS	30.816

TABLE 5 – Influence du paramètre k sur la mesure de couverture

4.7. Instance "Angers"

Nous nous sommes construit une instance sur une partie de la ville d'Angers, récupérant les emplacements des "bases immeubles" du Maine-et-Loire (fichier data/terms_49.txt). Nous sélectionnons alors les terminaux correspondant à la zone que nous avons choisie sur overpass-turbo.net (cf figure 4).



(a) Concentrateurs de niveau 1 (b) Concentrateurs de niveau 2

FIGURE 4 – Concentrateurs instance "Angers"

Nous l'avons alors dimensionnée à $nI = 100$, $nJ = 25$ et $nK = 4$ (initialement $nI = 200$, $nJ = 71$ et $nK = 4$ mais le temps d'exécution était trop long, pour notre Scatter Search et pour vOpt)
On observe sur la figure **??**que notre Scatter Search, peu importe la stratégie, est moins performante que sur les instances de Sanchez. Néanmoins, les stratégies utilisant des opérations de cross-overs contribuent à améliorer significativement notre ensemble bornant supérieur.

4.8. Instance "large"

Nous avons également confronté notre Scatter Search aux instances larges de Sanchez afin de vérifier que notre implémentation confirme ses performances à l'échelle supérieure.

Cependant, il s'avère que le temps d'exécution explose, même avec un critère d'arrêt petit ($k = 0.05$) pour la recherche tabou :

— Scatter Search : 762.38 secondes
— Scatter Search + crossover : 655.99 secondes
— Scatter Search + tabu : 1818.37 secondes

On constate sur la figure 6 que l'utilisation du crossover (avec et sans tabu) n'est de plus pas pertinente pour cette instance.
Nous n'avons pas pu nous comparer au Y_N calculé par vOpt, le temps de résolution étant bien long.

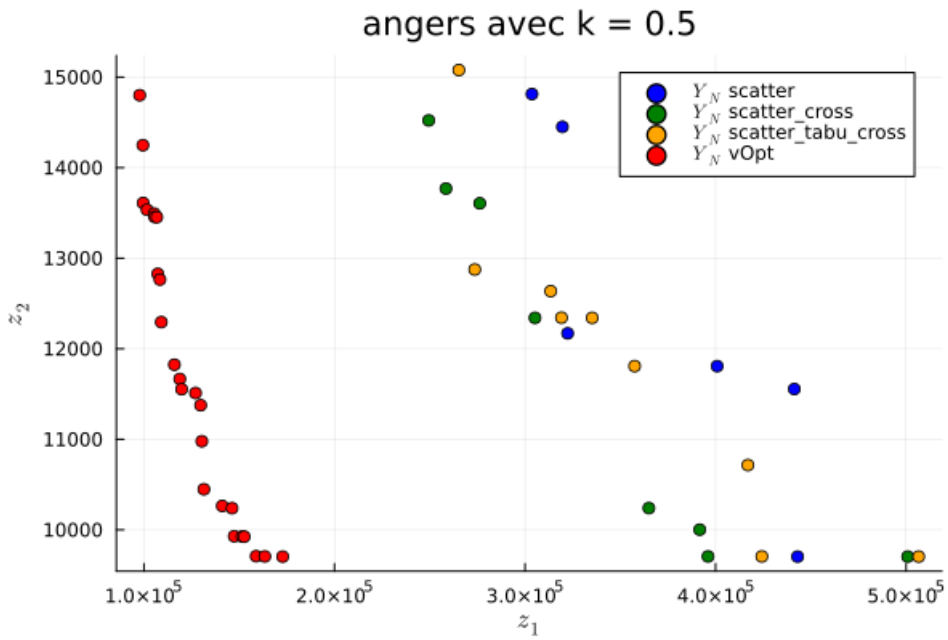


FIGURE 5 – Comparaison des stratégies sur l'instance Angers

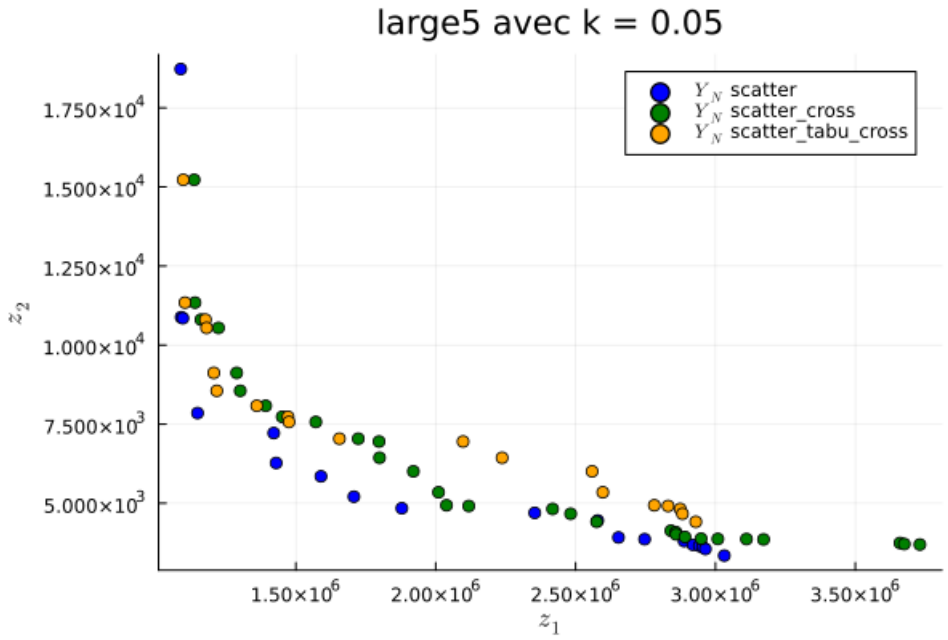


FIGURE 6 – Comparaison des stratégies sur l'instance large5

4.9. Conclusion des expérimentations

Bien que nos tests montrent un intérêt limité de notre "crossover", nos fronts de Pareto nous apparaissent satisfaisants compte tenu de nos temps de résolution : de l'ordre de la seconde pour les instances *small*, et de l'ordre de la minute pour les instances *medium*.

5. Conclusion

Le TSUFLP multi-objectif est un excellent cas d'étude pour évaluer l'impact des métaheuristiques multi-objectifs sur la résolution. La méthode Scatter Search combine de puissantes heuristiques et nous fournie des résultats très intéressants. Dans cet exercice, nous avons explorer différentes stratégies pour pousser toujours plus loin la qualité de notre ensemble bornant inférieur. A l'instar des métaheuristiques mono-objectifs, le *tuning* a une influence majeure sur la résolution. Dans une prochaine étude, nous pourrons chercher à améliorer nos fronts de Pareto en agissant sur le voisinage de Tabou, ou en perturbant davantage les solutions obtenues à chaque itération. Nous pourrons également repenser notre implémentation afin de pouvoir traiter des instances plus grandes.

Références

[1] P. Chardaire, J.-L. Lutton, A. Sutter, Upper and lower bounds for the two-level simple plant location problem, Annals of operations research 86 (1999) 117–140.

[2] F. Glover, A template for scatter search and path relinking, in : European conference on artificial evolution, Springer, 1997, pp. 1–51.

[3] A. López-Sánchez, J. Sánchez-Oro, M. Laguna, A new scatter search design for multiobjective combinatorial optimization with an application to facility location, INFORMS Journal on Computing 33 (2) (2021) 629–642.

[4] A. P. Xavier Gandibleux, Gauthier Soleilhac, voptsolver : an ecosystem for multi-objective linear optimization, in : JuliaCon, 2021, online and everywhere.