université de BORDEAUX

Collège Sciences et Technologies
UF Mathématiques et Interactions

Multi-period facility location problem

Martin Debouté
Lin Hirwa Shema

M2 CMI OPTIM / ROAD

2022–2023

Engagement de non plagiat

Nous déclarons être pleinement conscient que le plagiat de documents ou d'une partie d'un document publiés sur toutes formes de support, y compris l'internet, constitue une violation des droits d'auteur ainsi qu'une fraude caractérisée.

En conséquence, nous nous engageons à citer toutes les sources que nous avons utilisées pour produire et écrire ce rapport.

Fait à Talence le 14 janvier 2023

Signature

Martin Debouté
Lin Hirwa Shema

Table des matières

Description du problème

Le problème de localisation de services considéré dans ce projet est multi-périodes. En plus de devoir définir sur quels sites installer des services, il faut décider la période à laquelle le service commencera à être délivré. A la fin de chaque période de temps, une proportion croissante de groupes d'usagers (ou clients selon le contexte) doivent être couverts par un service, de façon à ce qu'à la fin de l'horizon, tous soient couverts. Une fois qu'un service est rendu accessible à un groupe d'usagers, il ne peut bien sûr pas être interrompu dans les périodes suivantes. De même, une fois un service installé sur un site, il l'est pour tout le reste de l'horizon de temps.

Soit $\mathcal{I}$ l'ensemble des sites possibles, $\mathcal{J}$ l'ensemble des groupe d'usagers, $\mathcal{T}$ l'ensemble (ordonné) des périodes ($\mathcal{T} := \{1, ..., T\}$). On note $f_i^t \geqslant 0$ le coût d'ouverture du service sur le site $i \in \mathcal{I}$ à la période $t \in \mathcal{T}$. Ce coût prend en compte l'installation et le fonctionnement (le maintien) du service sur les périodes suivantes. Un objectif de niveau de service doit être respecté tout au long de l'horizon de temps. Précisément, à la fin de la période $t \in \mathcal{T}$, il faut couvrir au moins $n^t \in \{1, ..., |\mathcal{J}|\}$ groupes d'usagers. Pour tout $t \in T$, on a $n^t < n^{t+1}$ et on a aussi $n^T = |\mathcal{J}|$. De plus, il faut ouvrir à chaque période t exactement $p^t \in \mathbb{N}^*$ nouveaux sites. Cette dernière restriction relève aussi de questions budgétaires. On note aussi $c_{ij}^t \geqslant 0$ le coût d'affectation du groupe d'usagers $j \in \mathcal{J}$ au site $i \in \mathcal{I}$ à la période $t \in \mathcal{T}$. Un groupe d'usagers peut être couvert par (affecté à) des sites différents selon les périodes. A chaque période de temps où ce groupe est couvert, le coût d'affectation (qui dépend du site auquel il est affecté) doit être payé.

L'objectif de ce problème est de minimiser la somme du coût d'installation des services sur les différent sites et du coût d'affectation des groupes d'usagers aux sites ouverts.

Réponses aux questions

Question 1.

Le problème de localisation de services à plusieurs périodes est NP-difficile parce qu'il est une généralisation du problème de localisation de site classique, qui est connu pour être NP-difficile. Cela signifie qu'un algorithme pour résoudre le problème de localisation de services à plusieurs périodes doit être capable de résoudre toute instance du problème de localisation de site classique en un temps raisonnable. Cependant, puisque le problème de localisation de site classique est NP-difficile, cela signifie qu'aucun tel algorithme ne peut exister à moins que $P = NP$. Par conséquent, le problème de localisation de services à plusieurs périodes est également NP-difficile.

Question 2.

Variables de décisions :

— $x_{ij}^t = \begin{cases} 1, \text{ si on affecte le groupe d'usagers } j \in \mathcal{J} \text{ au site } i \in \mathcal{I} \text{ à la période } t \in \mathcal{T}, \\ 0, \text{ sinon} \end{cases}$

— $y_i^t = \begin{cases} 1, \text{ si on ouvre le site } i \in \mathcal{I} \text{ à la période } t \in \mathcal{T}, \\ 0, \text{ sinon} \end{cases}$

— $z_i^t = \begin{cases} 1, \text{ si le site } i \in \mathcal{I} \text{ est ouvert (disponible) à la période } t \in \mathcal{T}, \\ 0, \text{ sinon} \end{cases}$

$$\min \sum_{i\in\mathcal{I}}\sum_{t\in\mathcal{T}} f_i^t y_i^t + \sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}}\sum_{t\in\mathcal{T}} c_{ij}^t x_{ij}^t$$

$$s.c. \sum_{i\in\mathcal{I}} y_i^t = p^t, \qquad \forall t \in \mathcal{T} \tag{1}$$

$$z_i^1 = y_i^1, \qquad \forall i \in \mathcal{I} \tag{2}$$

$$z_i^t = y_i^t + z_i^{t-1}, \qquad \forall i \in \mathcal{I},\ \forall t \in \mathcal{T} \setminus \{1\} \tag{3}$$

$$\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{J}} x_{ij}^t \geqslant n^t, \qquad \forall t \in \mathcal{T} \tag{4}$$

$$\sum_{i\in\mathcal{I}} x_{ij}^t - \sum_{i\in\mathcal{I}} x_{ij}^{t-1} \geqslant 0, \qquad \forall j \in \mathcal{J},\ \forall t \in \mathcal{T} \setminus \{1\} \tag{5}$$

$$x_{ij}^t \leqslant z_i^t, \qquad \forall t \in \mathcal{T},\ \forall j \in \mathcal{J},\ \forall i \in \mathcal{I} \tag{6}$$

$$\sum_{i\in\mathcal{I}} x_{ij}^t \leqslant 1, \qquad \forall t \in \mathcal{T},\ \forall j \in \mathcal{J} \tag{7}$$

$$\sum_{t\in\mathcal{T}} y_i^t \leqslant 1, \qquad \forall i \in \mathcal{I} \tag{8}$$

$$x_{ij}^t \in \{0,1\}, \qquad \forall i \in \mathcal{I},\ \forall j \in \mathcal{J},\ \forall t \in \mathcal{T} \tag{9}$$

$$y_i^t \in \{0,1\}, \qquad \forall i \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{10}$$

$$z_i^t \in \{0,1\}, \qquad \forall i \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{11}$$

L'objectif de ce problème est de minimiser la somme du coût d'installation des services sur les différent sites et du coût d'affectation des groupes d'usagers aux sites ouverts. La contrainte (1) assure qu'on ouvre exactement p^t sites pour chaque période $t \in T$. Les contraintes (2) et (3) définissent la variable z par rapport à la variable y. La contrainte (4) assure qu'au moins n^t groupes d'usagers sont couverts pour chaque période $t \in T$, et la contrainte (5) assure le fait que si un groupe d'usagers est couvert à une période, alors il doit aussi l'être dans les suivantes. La contrainte (6) assure qu'un groupe d'usagers ne peut être affecté à un site que si celui-ci est ouvert. La contrainte (7)assure le fait qu'on ne va pas affecter un groupe d'usagers à plusieurs sites dans la même période, et la (8) assure qu'un site ne peut être ouvert qu'une fois. Enfin, les contraintes (9) à (11) définissent les domaines de définitions de nos variables de décisions.

Question 3.

Si un groupe d'usagers $j \in \mathcal{J}$ est couvert pour la première fois à la période $t \in \mathcal{T}$, alors à chaque période $t' \in \{t, ..., T\}$ il sera affecté au site $i \in \mathcal{I}^{t'}$ qui minimise le coût d'affectation $c_{ij}^{t'}$. Il en découle que le coût total d'affectation du groupe d'usagers j à partir de la période t sera égal à $d_j^t = \sum_{t'=t}^{T} (\min_{i \in \mathcal{I}^{t'}} c_{ij}^{t'})$

Question 4.

Variables de décisions :

— $z_j^t = \begin{cases} 1, \text{ si le groupe d'usagers } j \in \mathcal{J} \text{ est couvert pour la première fois à la période } t \in \mathcal{T}, \\ 0, \text{ sinon} \end{cases}$

On pose $n^0 = 0$ et $\mathcal{I}^0 = \varnothing$.

$$\sum_{t \in \mathcal{T}} \sum_{i \in \mathcal{I}^t \setminus \mathcal{I}^{t-1}} f_i^t + \min \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}} d_j^t z_j^t$$

$$s.c. \sum_{j \in \mathcal{J}} z_j^t = n^t - n^{t-1} \qquad \forall t \in \mathcal{T} \qquad (12)$$

$$\sum_{t \in \mathcal{T}} z_j^t = 1 \qquad \forall j \in \mathcal{J} \qquad (13)$$

$$z_j^t \in \{0, 1\} \qquad \forall j \in \mathcal{J},\ \forall t \in \mathcal{T} \qquad (14)$$

Ici l'objectif est de minimiser le coût total de couverture des groupes d'usagers. Le coût d'ouverture des sites est connu donc on le place en dehors de la minimisation. Nous n'avons pas de contraintes sur l'ouverture des sites car les sites ouverts sont supposés connus. Ce sont les contraintes (4) - (7) du premier modèle qu'on doit adapter ici. Notre objectif assure que les clients seront toujours affectés aux sites disponibles, c'est pour cette raison que nous n'avons pas à adapter la contrainte (6). De plus, la contrainte (5) est impliquée par la définition et l'usage de la variable z_j^t car dès qu'elle prend la valeur 1 pour un $t \in \mathcal{T}$ et $j \in \mathcal{J}$ donné, ce groupe est couvert pour le reste de l'horizon du temps. La contrainte (7) est remplacée par la contrainte (12) qui dit que chaque groupe d'usagers est couvert une seule fois dans l'horizon de temps. Cette contrainte est valide car $n^T = |\mathcal{J}|$, ce qui indique bien que tous les groupes d'usagers seront couverts durant toutes les périodes de temps données. La contrainte (4) est remplacée par la contrainte (13) qui assure le fait que le nombre de groupes d'usagers couverts pour la première fois avant l'instant $t \in \mathcal{T}$ (et donc à t) soit égal à n^t.

Question 5.

Soit A la matrice de contraintes associée au modèle de la question 4. Cette matrice a $|\mathcal{T}| + |\mathcal{J}|$ lignes et $|\mathcal{T}| \times |\mathcal{J}|$ colonnes et se décrit comme telle (on notera $T := |\mathcal{T}|$ et $J := |\mathcal{J}|$) :

$\forall l \in [\![1; T + J]\!], \forall c \in [\![1; T \times J]\!]$,

$$(A_{lc}) = \begin{cases} 1, \text{ si } l \in [\![1; T]\!] & \text{et } c \in [\![(l-1)J + 1; (l-1)J + J]\!] \\ 1, \text{ si } l \in [\![T + 1; T + J]\!] & \text{et } c \in \{(l - T)j | j \in \mathcal{J}\} \\ 0, \text{ sinon} \end{cases}$$

On peut partitionner la matrice A en deux ensembles disjoints B et C tels que :

$(B_{lc}) = (A_{lc}), \forall c, \forall l \in [\![1; T]\!]$ et $(C_{l-T,c}) = (A_{lc}), \forall c, \forall l \in [\![T + 1; T + J]\!]$.

Vérifions que la **proposition 1** est vérifiée pour notre matrice A et ses partitions B et C :

— On voit que dans la matrice B, chaque élément qui vaut 1 se trouvent dans des colonnes disjointes. En effet, $\forall l, k \in [\![1; T]\!]$ t.q. $l < k$, on a $(l-1)J + J < (k-1)J + J$, d'où $c_l \cap c_k = \varnothing$. Chaque colonne de B a au plus 1 élément non nul. On constate aussi dans la matrice C que les ensembles $\{(l - T)j | \forall j \in \mathcal{J}\}$ sont disjoints pour tout $l \in [\![T + 1; T + J]\!]$. Donc chaque colonne de C a au plus 1 élément non nul. On en déduit que chaque colonne de A a au plus 2 éléments non nuls.

— Chaque élément de A vaut effectivement 0 ou 1.

— On a vu que chaque colonne de B contient au plus un élément non nul et chaque colonne de C contient au plus un élément non nul. Donc si une colonne de A contient deux éléments, c'est que l'un est dans B et l'autre est dans C. Donc quoi qu'il en soit, si deux éléments d'une colonne de A ont le même signe, alors la ligne de l'un est dans B, l'autre dans C.

— Il n'y a pas d'éléments de signes opposés dans A. Donc toutes contraintes sur les éléments de signes opposés sont respectées par défaut.

On peut donc conclure que la matrice A est totalement unimodulaire. Puisque la matrice A est totalement unimodulaire et $n^t - n^{t-1} \in \mathbb{N}$ pour tout $t \in \mathcal{T}$, on en déduit que toute solution optimale de la relaxation linéaire de notre modèle de la question 4 est entière.

Question 6.

Une heuristique possible pour notre problème serait la suivante :

On commence par calculer avec l'algorithme glouton (1) les ensembles I^t en ouvrant à chaque période $t \in \mathcal{T}$ les p^t sites non ouverts dans les périodes précédentes dont les coûts f_i^t sont minimaux.

Algorithme 1 : résout le problème d'ouverture de sites

```
Data : an instance of the problem
Result : the list of sites to open each period
open_sites ← [[0 * |T|] * |I|];
unopened_sites ← [i for i in I];
foreach t in T do
    sites ← [(i, f_i^t) for i in I];
    sort(sites by f_i^t);
    foreach i in {0, ..., p^t − 1} do
        foreach t' in {t', ..., |T|} do
            open_sites[sites[i][0]][t'] ← 1;
            remove(sites[i][0] in unopened_sites);
        end
    end
end
return open_sites;
```

Ensuite, on calcule avec l'algorithme glouton (2) les d_j^t pour tout $t \in \mathcal{T}$ et $j \in \mathcal{J}$ à partir des $\mathcal{I}^t$ grâce à la formule de la question 3.

Algorithme 2 : obtient le coût d'affectation de chaque groupe d'usagers pour chaque période

```
Data : an instance of the problem, open_sites
Result : the list of the assignement costs
assignment_cost ← [[0 * |T|] * |J|];
foreach j in J do
    foreach t in T do
        assignment_cost[j][t] ← sum(
            min(c[i][j][t'] for i in I, if open_sites[i][t'] == 1)
            for t' in {t', ..., T});
    end
end
return assignment_cost;
```

Puis, il nous suffit de calculer pour chaque groupe d'usagers la période à laquelle il est couvert pour la première fois en couvrant à chaque période $t \in \mathcal{T}$ les $n^t - n^{t-1}$ groupes d'usagers non couverts de coûts d_j^t minimaux. Ceci se fait simplement avec l'algorithme glouton (3) décrit ci-dessous.

Algorithme 3 : résout le problème d'assignement des usagers aux sites

Data : an instance of the problem, *assignement_cost*
Result : the list of assignement

```
assignments ← [[0 * |T|] * |J|];
foreach t in T do
    customers ← [(j, assignement_cost[j][t]) for j in J];
    sort(customers by assignement_cost[j][t]);
    foreach k in {0, ..., n^t − n^{t−1} − 1} do
        assignments[customers[k][0]][t] = 1;
        J ← J ∖ {customers[k][0]}
    end
end
return assignments;
```

Finalement, il ne reste plus qu'à construire la solution complète en affectant chaque groupe d'usagers au site ouvert de coût minimal à partir de la période où ce groupe est couvert. L'algorithme est immédiat.

Question 7.

On introduit des noeuds l^t pour chaque période $t \in \mathcal{T}$ qui produisent $n^t - n^{t-1}$ flot (en posant $n^0 = 0$) et des noeuds u_j pour chaque groupe d'usagers $j \in \mathcal{J}$ qui eux demandent une unité de flot.

Parmi les noeuds intermédiaires, on a :

— Les noeuds sl_i^t pour chaque site $i \in \mathcal{I}$ et chaque période $t \in \mathcal{T}$. Ces noeuds sont connecté aux noeuds l^t par des arcs (l^t, sl_i^t) de coût 0 et de capacité $(n^t - n^{t-1}) \sum_{t'=1}^{t} \bar{y}_i^{t'}$ pour tout $i \in \mathcal{I}$ et $t \in \mathcal{T}$.

— Les noeuds ul_j^t pour chaque groupe d'usagers $j \in \mathcal{J}$ et chaque période $t \in \mathcal{T}$. Ces noeuds sont connectés aux noeuds sl_i^t par des arcs (sl_i^t, ul_j^t) de coût c_{ij}^t et de capacité 1 pour tout $i \in \mathcal{I}$, $j \in \mathcal{J}$ et $t \in \mathcal{T}$.

— Les noeuds sul_{ij}^t pour chaque site $i \in \mathcal{I}$, chaque groupe d'usagers $j \in \mathcal{J}$ et chaque période $t \in \mathcal{T} \setminus \{1\}$. Ces noeuds sont connectés aux noeuds ul_j^{t-1} par des arcs (ul_j^{t-1}, sul_{ij}^t) de coût 0 et de capacité $\sum_{t'=1}^{t} \bar{y}_i^{t'}$ pour tout $i \in \mathcal{I}$, $j \in \mathcal{J}$ et $t \in \mathcal{T} \setminus \{1\}$. De plus, ces noeuds se connectent aux noeuds ul_j^t avec les arcs (sul_{ij}^t, ul_j^t) de coût c_{ij}^t et de capacité 1 pour tout $i \in \mathcal{I}$, $j \in \mathcal{J}$ et $t \in \mathcal{T}$.

Enfin, les noeuds $ul_j^{|\mathcal{T}|}$ et u_j sont reliés par les arcs $(ul_j^{|\mathcal{T}|}, u_j)$ de coût 0 et de capacité 1 pour tout $j \in \mathcal{J}$.

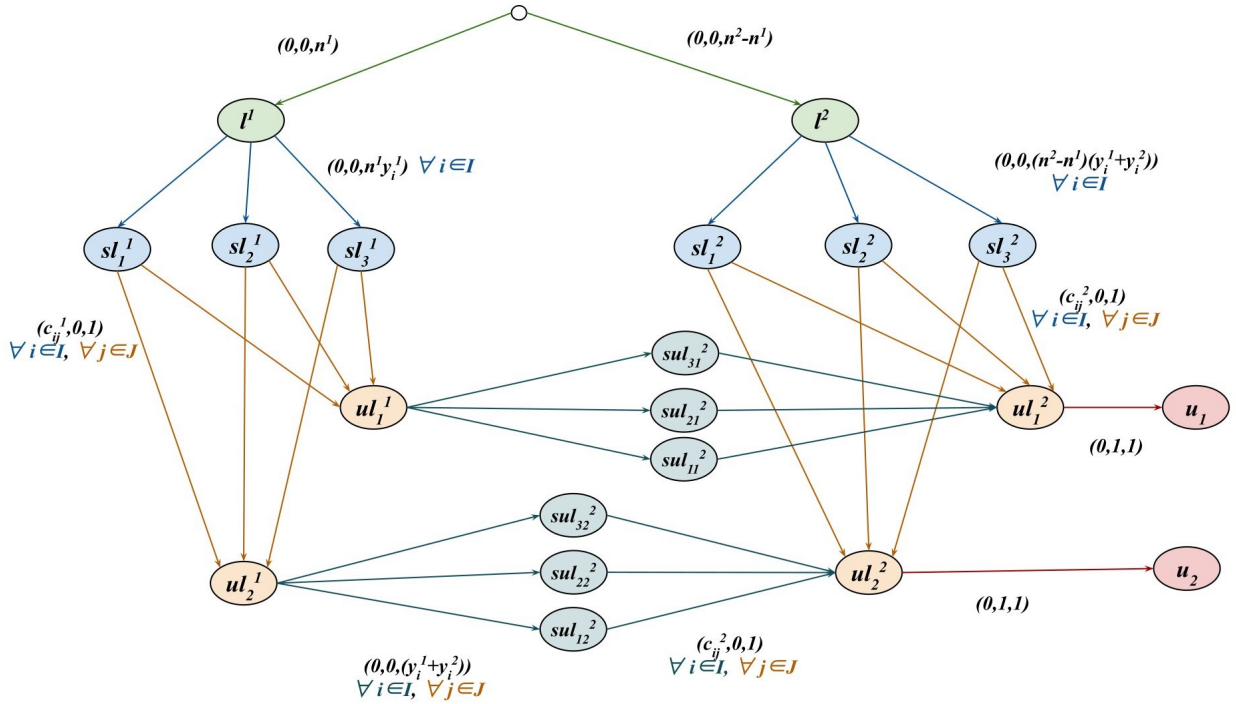

FIGURE 1 – Modèle de flot pour une instance avec 2 périodes, 3 sites et 2 groupes d'usagers (ici $y_i^t = \bar{y}_i^t$)

On a maintenant tous nos noeuds $\mathcal{V}$ et arcs $\mathcal{A}$ de notre graphe $G(\mathcal{V}, \mathcal{A})$. Un exemple d'un tel graphe donné sur la figure 1 pour une instance avec 2 périodes de temps, 3 sites potentiels et 2 groupes d'usagers. Pour chaque arc on a ses données entre parenthèse sous la forme (cout, borne inf., borne sup). Sur cette figure on a modéliser les production de l^t comme des capacités sur les arcs sortant d'un noeud d'origine et les demandes de u_j comme des bornes inférieures sur le flot entrant.

On peut modéliser notre problème comme un problème de flot de coût minimum avec le modèle suivant ($cout_{u,v}$ et $capacite_{u,v}$ désignent respectivement le coût et capacité de l'arc $(u, v) \in \mathcal{A}$) :

Variables de décisions :

— $x_{u,v} \in \mathbb{N}$ = quantité de flot circulant sur l'arc $(u, v) \in \mathcal{V}$

$$\sum_{t\in\mathcal{T}}\sum_{i\in\mathcal{I}} f_i^t \bar{y}_i^t + \min \sum_{(u,v)\in\mathcal{A}} cout_{u,v} x_{u,v}$$

$$s.c. \sum_{i\in\mathcal{I}} x_{l^t, sl_i^t} = n^t - n^{t-1} \qquad \forall t \in \mathcal{T} \qquad (15)$$

$$\sum_{t\in\mathcal{T}} x_{ul_i^{\mathcal{T}}, u_j} = 1 \qquad \forall j \in \mathcal{J} \qquad (16)$$

$$\sum_{\substack{v\in\mathcal{V}\\(v,u)\in\mathcal{A}}} x_{v,u} = \sum_{\substack{v\in\mathcal{V}\\(u,v)\in\mathcal{A}}} x_{u,v} \qquad \forall u \in \mathcal{V} \setminus \{l^t | \forall t \in \mathcal{T}, u_j | \forall j \in \mathcal{J}\} \qquad (17)$$

$$x_{u,v} \leqslant capacite_{u,v} \qquad \forall (u, v) \in \mathcal{A} \qquad (18)$$

$$x_{u,v} \geqslant 0 \qquad \forall (u, v) \in \mathcal{A} \qquad (19)$$

Le coût de notre solution est donné par le coût total d'ouverture des sites plus le coût total du flot dans notre graphe qu'on essaie de minimiser. La contrainte (15) assure que le flot sortant de chaque noeud l^t est de $n^t - n^{t-1}$, la contrainte (16) nous dit que le flot entrant dans chaque noeud u_j est de 1, et la contrainte (17) assure que pour tous les autres noeuds, leur flot entrant est égal à leur flot sortant. Les contraintes (18) et (19) limitent le flot de chaque arc entre 0 et sa capacité, sachant que la capacité de certains arcs dépend des valeurs des $\bar{y}_i^t$ comme décrit lors de la création du graphe.

Question 8.

Pour résoudre le problème de localisation des services sur plusieurs périodes à l'aide de la décomposition de Benders, nous pouvons suivre les étapes suivantes :

— (1) Définir le problème maître : ce problème est constitué des variables de décision pour l'ouverture des sites à chaque pas de temps, ainsi que de la fonction objectif de minimisation du coût total d'ouverture des sites et de service des utilisateurs.

— (2) Définir le sous-problème : ce problème comprend les variables de décision permettant d'affecter des groupes d'utilisateurs aux sites pour chaque pas de temps, ainsi que la fonction objective de minimisation du coût total de service des utilisateurs.

— (3) Résoudre le problème principal afin de déterminer les variables de décision pour l'ouverture de sites à chaque pas de temps.

— (4) Résoudre le sous-problème et générer une coupe de Benders : pour chaque pas de temps, nous pouvons utiliser la solution du sous-problème pour générer une coupe Benders, qui représente le coût optimal de service des utilisateurs à ce pas de temps. La coupe est déduite des valeurs duales du programme linéaire.

— (5) Mettre à jour le problème principal : nous pouvons ajouter la coupe de Benders au problème principal en tant que contrainte, garantissant ainsi que le coût total de service des utilisateurs à chaque pas de temps est minimisé.

— (6) Itérer : nous pouvons répéter les étapes (3) à (5) jusqu'à ce que le problème principal converge vers une solution optimale pour le problème de localisation de service multi-période.

Question 9.

On peut adapter notre modèle pour que chaque groupe soit servi par le même site tout au long du temps en modifiant la contrainte (5) pour qu'elle devienne :

$$x_{ij}^t - x_{ij}^{t-1} \geqslant 0, \qquad \forall i \in \mathcal{I},\ \forall j \in \mathcal{J},\ \forall t \in \mathcal{T} \setminus \{1\} \tag{20}$$

Question 10.

Pour cette question, on suppose que le coût $h_{ii'}$ est un coût fixe d'envoi de ressources du site i au site i'. On ne se soucie pas de savoir quelle quantité de ressources est envoyé de i à i' ou de savoir combien de destinataires recevront ces ressources. Cela signifie que peu importe le nombre de destinataires qui reçoivent leurs ressources depuis i', le coût d'envoi des ressources de i à i' reste $h_{ii'}$ dans tous les cas. On modifie notre modèle de la question 2 comme suit.

Variables de décisions additionnelles :

— $d_{ii'}^t = \begin{cases} 1, \text{ si des ressources sont envoyées du site } i \in \mathcal{I} \cup \{r\} \text{ au site } i' \in \mathcal{I} \text{ à la période } t \in \mathcal{T}, \\ 0, \text{ sinon} \end{cases}$

$$\min \sum_{i \in \mathcal{I} \cup \{r\}} \sum_{i' \in \mathcal{I}} \sum_{t \in \mathcal{T}} h_{ii'} d_{ii'}^t + \sum_{i \in \mathcal{I}} \sum_{t \in \mathcal{T}} f_i^t y_i^t + \sum_{i \in \mathcal{I}} \sum_{j \in \mathcal{J}} \sum_{t \in \mathcal{T}} c_{ij}^t x_{ij}^t$$

$$s.c. \sum_{i \in \mathcal{I} \cup \{r\}} d_{ii'}^t = z_{i'}^t \qquad \forall i' \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{21}$$

$$z_r^t = 1 \qquad \forall t \in \mathcal{T} \tag{22}$$

$$d_{ii'}^t \leqslant \frac{1}{2}(z_i^t + z_{i'}^t) \qquad \forall i \in \mathcal{I} \cup \{r\},\ \forall i' \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{23}$$

$$(|\mathcal{I}| - 1) \sum_{\substack{i \in \mathcal{I} \cup \{r\} \\ i \neq i'}} d_{ii'}^t \geqslant \sum_{\substack{i'' \in \mathcal{I} \\ i'' \neq i'}} d_{i'i''}^t \qquad \forall i' \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{24}$$

$$+ \text{ les contraintes de flots} \tag{25}$$

$$(1) - (11)$$

$$d_{ii'}^t \in \{0, 1\} \qquad \forall i \in \mathcal{I} \cup \{r\}, \forall i' \in \mathcal{I},\ \forall t \in \mathcal{T} \tag{26}$$

Le coût total de transportation des ressources se voit ajouté à l'objectif. On a aussi les contraintes (21) - (26) ajoutées en plus des contraintes (1) - (11) du modèle initial. La contrainte (21) signifie que chaque site reçoit ses ressources d'une seule source et seulement s'il est ouvert. La contrainte (22) assure le fait que l'organisme central r est considéré comme toujours ouvert. La contrainte (23) nous dit que pour toute période $t \in \mathcal{T}$, un site $i' \in \mathcal{I}$ ne peut recevoir des ressources de $i \in \mathcal{I} \cup \{r\}$ que si les deux sites sont ouverts. La contrainte (24) assure le fait que si un site $i' \in \mathcal{I}$ envoi des ressources à d'autres sites, alors il doit lui-même recevoir des ressources (sauf bien sûr le site r qui n'est pas contraint). Puisque le site i' peut envoyer des ressources à plusieurs autre sites mais ne peut avoir qu'un seul site qui le précède, on fixe un M qui prend la valeur $|\mathcal{I}| - 1$, car dans le pire cas i' envoi des ressources à tous les autres sites sauf lui-même. La contrainte (26) quant à elle, pose les domaines de définitions de nos variables d.

Expérimentation et résultats

L'ensemble du code réalisé et utilisé pour effectuer ces tests comparatifs est disponible en libre accès sur ce dépôt Github : https://github.com/mdeboute/multi-period-flp. L'implémentation a été faite en Python et les tests ont étés effectués sur le serveur `infini2` du CREMI [1] possédant deux processeurs Intel Xeon X5570 64 bits de quatre coeurs chacun, cadencés à 3.2 GHz pour 55 Go de mémoire vive. La totalité des 16 coeurs ont étés utilisés pour effectuer nos tests. Nous avons décidé de fixer un temps limite de 300 secondes soit 5 minutes pour la résolution de chaque instance. Enfin, notre modèle de programmation linéaire en nombres entiers fut implémenté avec Python-MIP [2] et le solveur que nous avons utilisé pour le résoudre était Gurobi [3] 9.1.0.

Rappelons que le jeu d'instance testé comporte des instances dites « *Single* » signifiant que pour elles il n'y a qu'un seul site à ouvrir à chaque période de temps, et des instances dites « *Multi* » signifiant que pour elles il y a plusieurs sites à ouvrir à chaque période de temps. Nous avons fixé une tolérance de 10^{-4} pour le *gap* d'optimalité, *i.e* qu'une solution sera considéré optimale *ssi* son *gap* $\leqslant 0.0001\%$.

Résultats des tests sur les instances « *Single* »

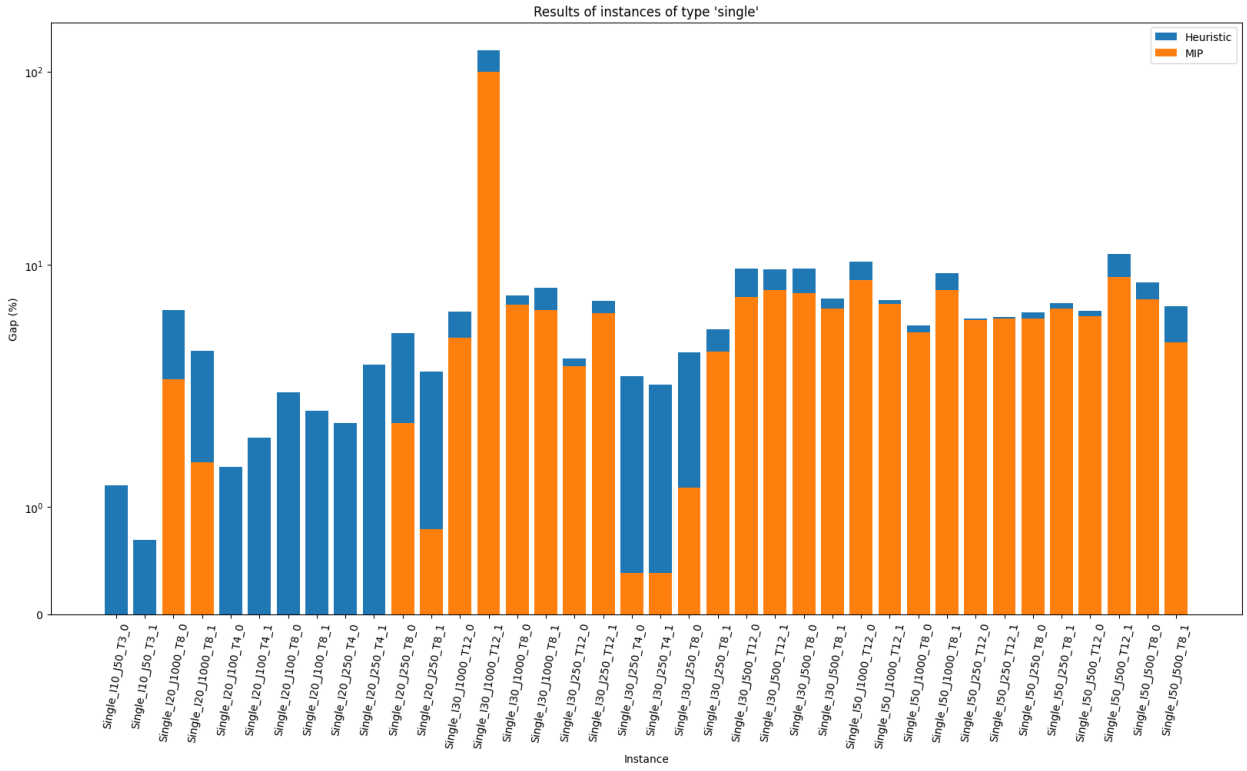

FIGURE 2 – Graphique comparatif des performances des deux méthodes de résolution sur les instances de type « *Single* »

1. Centre de Ressources pour l'Enseignement des Mathématiques et de l'Informatique
2. https://www.python-mip.com
3. https://www.gurobi.com/solutions/gurobi-optimizer/

On constate que le solveur MIP [4] Gurobi trouve 8 solutions optimales parmi les instances les plus petites mais a plus de mal dès que $|\mathcal{J}| > 250$ et $|\mathcal{T}| > 4$. Notre heuristique quant à elle performe bien sur ces mêmes instances, en trouvant des solutions ayant des coûts très proches de celles trouvées avec le MIP. Notons qu'il est particulièrement difficile de trouver une bonne solution pour l'instance `Single_I30_J1000_T12_1` puisque dans les deux cas, MIP et heuristique ont trouvés une solution avec un *gap* d'optimalité supérieur à 100% comme on peut le constater sur le graphique ci-dessus (2) et sur les tableaux de résultats (1 & 2). Notons aussi que l'heuristique trouve une solution vraiment proche de l'optimale pour l'instance `Single_I10_J50_T3_1` puisque son gap est de 0.6913%.

Instance	**Runtime (sec)**	**Obj. Value**	**Gap (%)**
Single_I10_J50_T3_0	0.0	23484	1.2008
Single_I10_J50_T3_1	0.0	24301	0.6913
Single_I20_J1000_T8_0	0.18	548961	5.8013
Single_I20_J1000_T8_1	0.17	529254	3.5657
Single_I20_J100_T4_0	0.01	45097	1.3748
Single_I20_J100_T4_1	0.0	44157	1.6419
Single_I20_J100_T8_0	0.02	81883	2.1812
Single_I20_J100_T8_1	0.02	80797	1.8924
Single_I20_J250_T4_0	0.01	87633	1.7847
Single_I20_J250_T4_1	0.01	90916	3.0259
Single_I20_J250_T8_0	0.04	161142	4.4366
Single_I20_J250_T8_1	0.04	156920	2.7931
Single_I30_J1000_T12_0	0.47	776119	5.7209
Single_I30_J1000_T12_1	0.48	793688	**128.7386**
Single_I30_J1000_T8_0	0.21	535028	6.9304
Single_I30_J1000_T8_1	0.22	525009	7.5711
Single_I30_J250_T12_0	0.13	221491	3.2546
Single_I30_J250_T12_1	0.12	229222	6.5018
Single_I30_J250_T4_0	0.01	88461	2.6562
Single_I30_J250_T4_1	0.01	86863	2.3942
Single_I30_J250_T8_0	0.05	154812	3.5248
Single_I30_J250_T8_1	0.05	158698	4.6273
Single_I30_J500_T12_0	0.24	401471	9.575
Single_I30_J500_T12_1	0.24	413236	9.487
Single_I30_J500_T8_0	0.11	287171	9.5118
Single_I30_J500_T8_1	0.11	288331	6.6907
Single_I50_J1000_T12_0	0.63	773459	10.4023
Single_I50_J1000_T12_1	0.62	767560	6.5409
Single_I50_J1000_T8_0	0.28	512626	4.8296
Single_I50_J1000_T8_1	0.27	532711	9.0621
Single_I50_J250_T12_0	0.16	229611	5.2701
Single_I50_J250_T12_1	0.16	228604	5.3463
Single_I50_J250_T8_0	0.07	157450	5.6469
Single_I50_J250_T8_1	0.07	160494	6.3209
Single_I50_J500_T12_0	0.31	405883	5.7797
Single_I50_J500_T12_1	0.32	409894	11.3689
Single_I50_J500_T8_0	0.15	282693	8.1112
Single_I50_J500_T8_1	0.13	284348	6.0763

TABLE 1 – Résultats des instances de type « *Single* » pour l'heuristique

4. *Mixed Integer Programming*

On peut voir dans le tableau (1) que notre heuristique est très rapide, ne dépassant pas la seconde avec une moyenne de 0.16 secondes d'exécution. De plus, on constate qu'elle trouve des solutions qui ont un coût relativement proche de celle de la solution optimale avec un *gap* moyen de 8.48%. Pour calculer ce *gap* (%) nous avons simplement calculé l'écart entre la valeur de la solution du MIP et celle de l'heuristique puis il nous a suffit de sommer les pourcentages (et nous parlons bien des centièmes). Autrement dit, $heuristic\_gap = 100 \times \frac{abs(mip\_obj - heuristic\_obj)}{mip\_obj} + mip\_gap$. Le MIP solveur quant à lui à temps d'execution moyen de 228.15 secondes pour un gap moyen de 6.23%.

Instance	Runtime (sec)	Obj. Value	Gap (%)
Single_I10_J50_T3_0	0.14	23202	**0.0**
Single_I10_J50_T3_1	0.12	24133	**0.0**
Single_I20_J1000_T8_0	290.56	531134	2.5539
Single_I20_J1000_T8_1	283.29	517860	1.4129
Single_I20_J100_T4_0	3.22	44477	**0.0**
Single_I20_J100_T4_1	4.82	43432	**0.0**
Single_I20_J100_T8_0	212.0	80097	**0.0**
Single_I20_J100_T8_1	21.98	79268	**0.0**
Single_I20_J250_T4_0	211.2	86069	**0.0**
Single_I20_J250_T4_1	108.68	88165	**0.0**
Single_I20_J250_T8_0	300.01	156867	1.7837
Single_I20_J250_T8_1	300.2	153778	0.7908
Single_I30_J1000_T12_0	300.01	764049	4.1657
Single_I30_J1000_T12_1	300.25	1021783	**100.0**
Single_I30_J1000_T8_0	209.58	538790	6.2273
Single_I30_J1000_T8_1	252.8	534269	5.8073
Single_I30_J250_T12_0	300.02	220905	2.99
Single_I30_J250_T12_1	300.01	231221	5.6297
Single_I30_J250_T4_0	299.81	86451	0.384
Single_I30_J250_T4_1	300.08	85118	0.3853
Single_I30_J250_T8_0	300.01	151186	1.1826
Single_I30_J250_T8_1	300.26	157002	3.5586
Single_I30_J500_T12_0	300.03	412524	6.8219
Single_I30_J500_T12_1	164.44	421898	7.3909
Single_I30_J500_T8_0	162.3	293992	7.1366
Single_I30_J500_T8_1	300.01	290472	5.9482
Single_I50_J1000_T12_0	221.91	789867	8.2809
Single_I50_J1000_T12_1	277.72	769770	6.253
Single_I50_J1000_T8_0	292.76	510800	4.4734
Single_I50_J1000_T8_1	211.94	541740	7.3672
Single_I50_J250_T12_0	300.01	229406	5.1808
Single_I50_J250_T12_1	297.49	228851	5.2383
Single_I50_J250_T8_0	218.86	158055	5.2627
Single_I50_J250_T8_1	300.01	161124	5.9284
Single_I50_J500_T12_0	262.13	407385	5.4096
Single_I50_J500_T12_1	280.23	421229	8.6036
Single_I50_J500_T8_0	286.67	286869	6.634
Single_I50_J500_T8_1	194.12	278307	3.9518

TABLE 2 – Résultats des instances de type « *Single* » pour le MIP

Résultats des tests sur les instances « *Multi* »

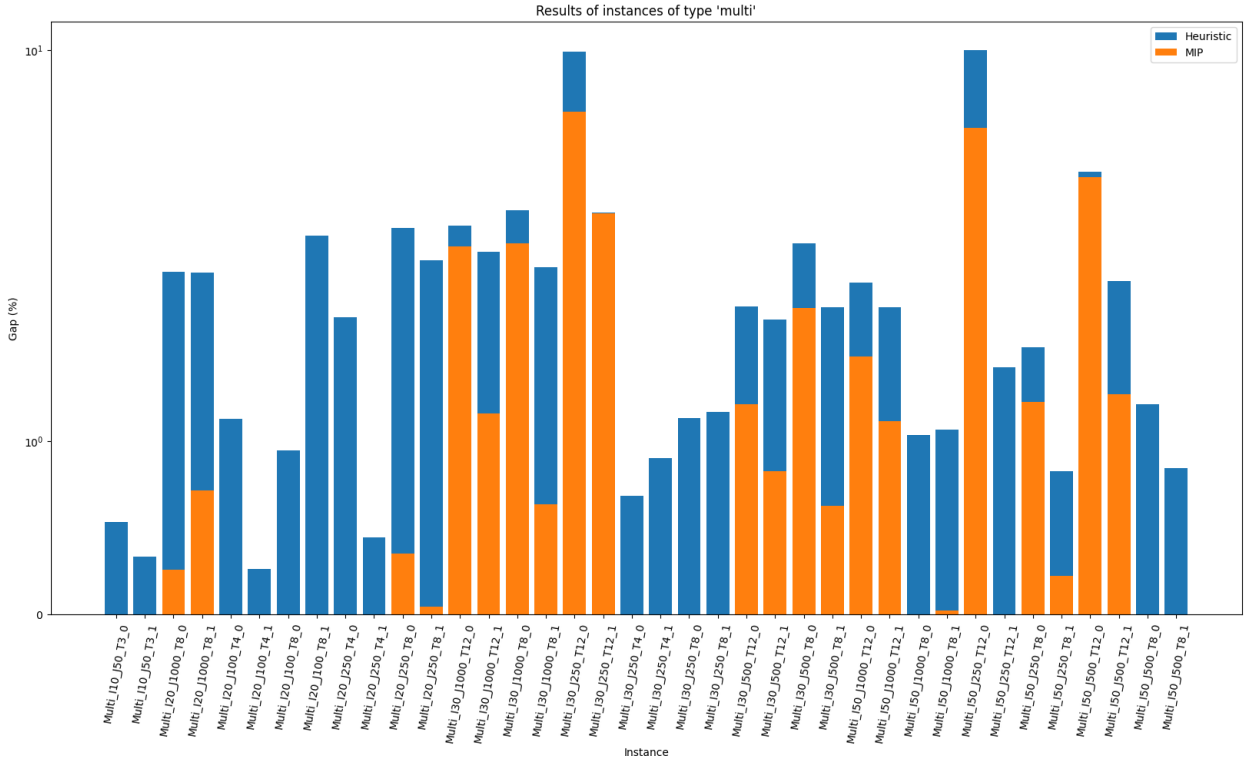

FIGURE 3 – Graphique comparatif des performances des deux méthodes de résolution sur les instances de type « *Multi* »

Tout d'abord on remarque qu'il y a des « *clusters* » d'instances plus compliqués à résoudre optimalement que d'autres, ces derniers sont clairement identifiables sur le graphique ci-dessus.

On constate que sur les instances « *Multi* » le solveur MIP résout 16 instances à l'optimale soit 8 de plus que pour les instances du type « *Single* » (*c.f.* les tableaux de résultats (2 & 4)). Le temps moyen de résolution par instance pour le MIP est de 185.13 secondes soit plus d'1.23 fois plus rapide que sur les instances du type « *Single* ». De plus, le *gap* moyen du MIP est de 0.96% ce qui est une fois de plus bien meilleur que sur les instances du type « *Single* ». En effet, le MIP trouve 9 solutions non optimales avec un *gap* inférieur à 1% contre 3 sur les instances « *Single* ».

L'heuristique quant à elle résout les instances du type « *Multi* » avec un temps moyen de 0.20 secondes soit 1.25 fois plus lentement que sur les instances du premier type. Cependant, son *gap* moyen est moindre avec une valeur de 2.07%. En effet, on note que l'heuristique trouve 9 solutions avec un *gap* inférieur à 1% contre une sur les instances « *Single* » (*c.f.* les tableaux de résultats (1 & 3)). Mais ce n'est pas pour cette raison que l'heuristique est plus performante sur les instances du type « *Multi* », au contraire. En effet, on constate grâce aux graphiques (2 & 3) que les valeurs des solutions trouvées grâce à l'heuristique sont en moyenne bien plus proches de celles trouvées par le MIP pour les instances « *Single* » que pour les instances « *Multi* ». L'heuristique mettant beaucoup moins de temps que le solveur MIP pour trouver une solution, il est bien plus intéressant et avantageux de l'utiliser dans le premier cas de figure que dans le second. Effectivement, il est plus intéressant d'utiliser le solveur MIP pour résoudre les instances du type « *Multi* » puisqu'il résout ces dernières plus rapidement et plus efficacement, avec un *gap* moyen bien plus faible que pour les instances du premier type.

Instance	Runtime (sec)	Obj. Value	Gap (%)
Multi_I10_J50_T3_0	0.0	34509	0.5332
Multi_I10_J50_T3_1	0.0	36641	0.333
Multi_I20_J1000_T8_0	0.19	731218	1.976
Multi_I20_J1000_T8_1	0.2	807798	1.9723
Multi_I20_J100_T4_0	0.01	68701	1.1295
Multi_I20_J100_T4_1	0.01	108776	0.2611
Multi_I20_J100_T8_0	0.02	127679	0.9453
Multi_I20_J100_T8_1	0.02	116535	2.5307
Multi_I20_J250_T4_0	0.01	112459	1.7144
Multi_I20_J250_T4_1	0.02	232129	0.4438
Multi_I20_J250_T8_0	0.05	235372	2.6865
Multi_I20_J250_T8_1	0.05	209595	2.1174
Multi_I30_J1000_T12_0	0.54	1007963	2.7325
Multi_I30_J1000_T12_1	0.59	1224115	2.248
Multi_I30_J1000_T8_0	0.23	716045	3.0629
Multi_I30_J1000_T8_1	0.29	1181039	2.0056
Multi_I30_J250_T12_0	0.13	299654	9.8664
Multi_I30_J250_T12_1	0.14	276580	3.0103
Multi_I30_J250_T4_0	0.02	266514	0.6844
Multi_I30_J250_T4_1	0.02	322845	0.9012
Multi_I30_J250_T8_0	0.07	348154	1.1348
Multi_I30_J250_T8_1	0.07	360819	1.169
Multi_I30_J500_T12_0	0.3	624388	1.775
Multi_I30_J500_T12_1	0.29	625574	1.7002
Multi_I30_J500_T8_0	0.11	395191	2.3896
Multi_I30_J500_T8_1	0.12	400793	1.7737
Multi_I50_J1000_T12_0	0.83	1583818	1.9168
Multi_I50_J1000_T12_1	0.86	1555188	1.774
Multi_I50_J1000_T8_0	0.37	1206507	1.0359
Multi_I50_J1000_T8_1	0.42	1789442	1.0668
Multi_I50_J250_T12_0	0.22	497921	9.9937
Multi_I50_J250_T12_1	0.22	550365	1.4242
Multi_I50_J250_T8_0	0.09	371804	1.5417
Multi_I50_J250_T8_1	0.08	271628	0.8246
Multi_I50_J500_T12_0	0.34	529743	4.0592
Multi_I50_J500_T12_1	0.45	875421	1.9255
Multi_I50_J500_T8_0	0.18	625671	1.2121
Multi_I50_J500_T8_1	0.22	961577	0.845

TABLE 3 – Résultats des instances de type « *Multi* » pour l'heuristique

Instance	Runtime (sec)	Obj. Value	Gap (%)
Multi_I10_J50_T3_0	0.24	34325	**0.0**
Multi_I10_J50_T3_1	0.14	36519	**0.0**
Multi_I20_J1000_T8_0	269.39	718670	0.26
Multi_I20_J1000_T8_1	284.8	797646	0.7156
Multi_I20_J100_T4_0	0.51	67925	**0.0**
Multi_I20_J100_T4_1	0.32	108492	**0.0**
Multi_I20_J100_T8_0	5.89	126472	**0.0**
Multi_I20_J100_T8_1	23.27	113586	**0.0001**
Multi_I20_J250_T4_0	50.92	110531	**0.0**
Multi_I20_J250_T4_1	2.44	231099	**0.0001**
Multi_I20_J250_T8_0	300.02	229872	0.3498
Multi_I20_J250_T8_1	300.03	205247	0.0429
Multi_I30_J1000_T12_0	278.87	1004033	2.3426
Multi_I30_J1000_T12_1	179.72	1210809	1.161
Multi_I30_J1000_T8_0	265.64	711226	2.3899
Multi_I30_J1000_T8_1	300.0	1164861	0.6358
Multi_I30_J250_T12_0	300.03	310252	6.3297
Multi_I30_J250_T12_1	300.02	276620	2.9958
Multi_I30_J250_T4_0	9.26	264690	**0.0**
Multi_I30_J250_T4_1	7.73	319936	**0.0001**
Multi_I30_J250_T8_0	20.31	344203	**0.0**
Multi_I30_J250_T8_1	9.26	356601	**0.0**
Multi_I30_J500_T12_0	300.1	620885	1.214
Multi_I30_J500_T12_1	291.39	620104	0.8258
Multi_I30_J500_T8_0	280.42	392735	1.7681
Multi_I30_J500_T8_1	300.02	396201	0.628
Multi_I50_J1000_T12_0	239.02	1576999	1.4863
Multi_I50_J1000_T12_1	294.76	1544956	1.1161
Multi_I50_J1000_T8_0	201.37	1194010	**0.0001**
Multi_I50_J1000_T8_1	300.0	1770725	0.0208
Multi_I50_J250_T12_0	299.07	519762	5.6073
Multi_I50_J250_T12_1	78.33	542527	**0.0001**
Multi_I50_J250_T8_0	290.98	370637	1.2278
Multi_I50_J250_T8_1	294.09	269988	0.2208
Multi_I50_J500_T12_0	270.13	530544	3.908
Multi_I50_J500_T12_1	300.0	869707	1.2728
Multi_I50_J500_T8_0	180.99	618087	**0.0**
Multi_I50_J500_T8_1	205.55	953452	**0.0**

TABLE 4 – Résultats des instances de type « *Multi* » pour le MIP

Conclusion

Pour conclure, on constate que le solveur MIP trouve en général de meilleurs solutions pour les instances où l'on peut ouvrir plusieurs sites à chaque période de temps plutôt que pour les instances où il n'y a qu'un seul site à ouvrir à chaque période. Cependant, l'heuristique se trouve utile pour trouver de « bonnes » solutions rapidement sur ces dernières mais est moins efficace sur les instances de type « *Multi* ». Le solveur MIP étant surtout efficace sur les instances de petites tailles dans le cas « *Single* », il serait intéressant de développer une méthode de résolution exacte basé sur la décomposition de Benders afin de pouvoir résoudre optimalement des instances plus conséquentes. Nous avons eu des difficultés à répondre aux questions 7 et 8 mais nous pensons avoir compris l'essentiel de ce que l'on nous demandait.