Table des matières

1 Partie théorique

1.1 Modélisation statique

Soit $G = (V, A)$ un graphe orienté, un sommet origine $s \in V$ et un sommet destination $t \in V$, chaque arc $(ij) \in A$ a une durée de trajet d_{ij}, de plus, chaque sommet a un poids p_i. Le problème statique consiste à trouver un plus court chemin d'un sommet s à un sommet t, où la somme pondérée des sommets appartenant au chemin est inférieure ou égale à un nombre entier S.

On utilise les variables suivantes :

$$x_{ij} = 1 \text{ ssi l'arc (ij) est emprunté}$$

$$y_i = 1 \text{ ssi le sommet i est emprunté}$$

$$\begin{cases} \min\limits_{x} \sum\limits_{ij \in A} d_{ij} \cdot x_{ij} \\ \quad s.c. : \\ \qquad \sum\limits_{i \in V} p_i \cdot y_i \leqslant S & \text{(contrainte de sac à dos)} \\ \qquad \sum\limits_{i:ij \in A} x_{ij} = \sum\limits_{i:ji \in A} x_{ji} \; \forall \, j \in V \backslash \{s,t\} & \text{(autant d'arcs entrants que d'arcs sortants)} \\ \qquad \sum\limits_{j:sj \in A} x_{s,j} = 1 & \text{(un arc sortant de s)} \\ \qquad \sum\limits_{i:it \in A} x_{i,t} = 1 & \text{(un arc entrant en t)} \\ \qquad \sum\limits_{j:ij \in A} x_{ij} = y_i \; \forall \, i \in V \backslash \{s\} & \text{(chaque sommet appartenant au chemin est passé une seule fois)} \\ \qquad \sum\limits_{j:ji \in A} x_{ji} = y_i \; \forall \, i \in V \backslash \{t\} & \text{(chaque sommet appartenant au chemin est passé une seule fois)} \\ \qquad x_{ij} \in \{0,1\} \; \forall \, ij \in A \\ \qquad y_i \in \{0,1\} \; \forall \, i \in V \end{cases}$$

1.2 Modélisation robuste

La version robuste de ce problème considère différents scénarios étant donné l'incertitude de certains de ses paramètres :

Incertitude sur les durées : $d^1_{ij} \in \mathcal{U}^1$

$$\mathcal{U}^1 = \left\{ \{d^1_{ij} = d_{ij} \cdot (1 + \delta^1_{ij})\}_{ij \in A} \middle| \sum_{ij \in A} \delta^1_{ij} \leqslant d_1, \; \delta^1_{ij} \in [0, D_{ij}] \; \forall \, ij \in A \right\}$$

Incertitude sur les poids : $p^2_i \in \mathcal{U}^2$

$$\mathcal{U}^2 = \left\{ \{p^2_i = p_i + \delta^2_i \cdot \hat{p_i}\}_{i \in V} \middle| \sum_{i \in V} \delta^2_i \leqslant d_2, \; \delta^2_i \in [0, 2] \; \forall \, i \in V \right\}$$

Le problème robuste s'écrit de la façon suivante :

$$
\left\{
\begin{array}{l}
\min\limits_{(x,y)} \max\limits_{\delta^1_{ij}} \sum\limits_{ij\in A} d_{ij}\cdot(1+\delta^1_{ij})\cdot x_{ij} \\
\quad s.c. : \\
\qquad \left.
\begin{array}{l}
\sum\limits_{i:ij\in A} x_{ij} = \sum\limits_{i:ji\in A} x_{ji} \;\forall\, j\in V\backslash\{s,t\}, \\
\sum\limits_{j\in V} x_{sj} = 1, \\
\sum\limits_{i\in V} x_{it} = 1, \\
\sum\limits_{j:ij\in A} x_{ij} = y_i \;\forall\, i\in V\backslash\{s\}, \\
\sum\limits_{j:ji\in A} x_{ji} = y_i \;\forall\, i\in V\backslash\{t\}, \\
x_{ij}\in\{0,1\}\;\forall\,(ij)\in A,\; y_i\in\{0,1\}\;\forall\, i\in V,
\end{array}
\right\}\mathcal{X}^{comb} \\
\qquad \left.
\begin{array}{l}
\left.
\begin{array}{l}
\sum\limits_{ij\in A}\delta^1_{ij}\leqslant d_1, \\
\delta^1_{ij}\in[0,D_{ij}]\;\forall\,(i,j)\in A,
\end{array}
\right\}\mathcal{U}^1 \\
\left.
\begin{array}{l}
\sum\limits_{i\in V}(p_i+\delta^2_i\cdot\hat{p_i})\cdot y_i\leqslant S, \\
\sum\limits_{i\in V}\delta^2_i\leqslant d_2, \\
\delta^2_i\in[0,2]\;\forall\, i\in V,
\end{array}
\right\}\mathcal{U}^2
\end{array}
\right\}\mathcal{X}^{num}
\end{array}
\right.
$$

1.3 Plans coupants et LazyCallback

1.3.1 Robustesse dans les contraintes

$$
\left\{
\begin{array}{l}
\min\limits_{x,y,z} \quad z \\
\quad s.c. : \\
\qquad \sum\limits_{ij\in A} d_{ij}\cdot(1+\delta^1_{ij})\cdot x_{ij}\leqslant z, \\
\qquad (x,y)\in\mathcal{X}^{comb}, \\
\qquad (x,y)\in\mathcal{X}^{num}
\end{array}
\right.
$$

1.3.2 Problème maître

A la première itération, on fixe

$$\mathcal{U}^{1*}=\varnothing \qquad \mathcal{U}^{2*}=\varnothing$$

On résoudra donc le problème maître suivant :

$$
\left\{
\begin{array}{l}
\min\limits_{x,y,z} \quad z \\
\quad s.c. : \\
\qquad (x,y)\in\mathcal{X}^{comb}
\end{array}
\right. \tag{MP}
$$

C'est-à-dire :

$$\begin{cases} \min\limits_{x,y,z} \quad z \\ \quad s.c. : \\ \qquad \sum\limits_{i:ij\in A} x_{ij} = \sum\limits_{i:ji\in A} x_{ji} \ \forall\, j \in V\backslash\{s,t\}, \\ \qquad \sum\limits_{j\in V} x_{sj} = 1, \\ \qquad \sum\limits_{i\in V} x_{it} = 1, \\ \qquad \sum\limits_{j:ij\in A} x_{ij} = y_i \ \forall\, i \in V\backslash\{s\}, \\ \qquad \sum\limits_{j:ji\in A} x_{ji} = y_i \ \forall\, i \in V\backslash\{t\}, \\ \qquad x \in \{0,1\}^{n^2}, y \in \{0,1\}^n \end{cases} \tag{MP}$$

1.3.3 Sous-problèmes

On résout le problème maître et on obtient une solution $(x^\star, y^\star), z^\star$. Les sous-problèmes nécessaires à la résolution du problème robuste par plans coupants sont les suivants :

$$\begin{cases} \max\limits_{\delta^1_{ij}} \quad \sum\limits_{(ij)\in A} d_{ij}\cdot(1+\delta^1_{ij})\cdot x^\star_{ij} \\ \quad s.c. : \\ \qquad \sum\limits_{(ij)\in A} \delta^1_{ij} \leqslant d_1 \\ \qquad \delta^1_{ij} \in [0, D_{ij}],\ \forall (ij) \in A \end{cases} \tag{SPo}$$

$$\begin{cases} \max\limits_{\delta^2_i} \quad \sum\limits_{i\in V} (p_i + \delta^2_i \cdot \hat{p_i})\cdot y^\star_i \\ \quad s.c. : \\ \qquad \sum\limits_{i\in V} \delta^2_i \leqslant d_2 \\ \qquad \delta^2_i \in [0,2],\ \forall i \in V \end{cases} \tag{SP2}$$

1.3.4 Solution optimale

On résout SPo et SP2 et on obtient des solutions $\delta^{1\star}$ et $\delta^{2\star}$.
Une solution $(x^\star, y^\star), z^\star$ du problème maître est optimale si

$$\begin{cases} \sum\limits_{(ij)\in A} d_{ij}\cdot(1+\delta^{1\star}_{ij})\cdot x^\star_{ij} \leqslant z^\star \\ \sum\limits_{i\in V} (p_i + \delta^{2\star}_i \cdot \hat{p_i})\cdot y^\star_i \leqslant S \end{cases}$$

1.3.5 Coupes à ajouter

Si l'une des conditions ci-dessus n'est pas respectées, il faudra alors ajouter l'une ou deux des coupes ci-dessous (en fonction desquelles ne sont pas respectées) :

$$\begin{cases} \sum\limits_{(ij)\in A} d_{ij}\cdot(1+\delta^{1\star}_{ij})\cdot x_{ij} \leqslant z \\ \sum\limits_{i\in V} (p_i + \delta^{2\star}_i \cdot \hat{p_i})\cdot y_i \leqslant S \end{cases}$$

On itère ce processus jusqu'à obtenir une solution réalisable. Celle-ci sera alors optimale.
Une façon d'améliorer l'algorithme consiste à utiliser des callbacks afin d'ajouter les coupes au fur et à mesure plutôt que de démarrer à zéro à chaque ajout de coupe.

1.4 Dualisation

Rappelons la formulation du problème robuste :

$$
\left\{
\begin{array}{l}
\min\limits_{(x,y)} \max\limits_{\delta^1_{ij}} \sum\limits_{ij\in A} d_{ij}\cdot(1+\delta^1_{ij})\cdot x_{ij} \\
\quad s.c. : \\
\qquad \left.
\begin{array}{l}
\sum\limits_{i:ij\in A} x_{ij} = \sum\limits_{i:ji\in A} x_{ji} \;\forall\, j\in V\backslash\{s,t\}, \\
\sum\limits_{j\in V} x_{sj} = 1, \\
\sum\limits_{i\in V} x_{it} = 1, \\
\sum\limits_{j:ij\in A} x_{ij} = y_i \;\forall\, i\in V\backslash\{s\}, \\
\sum\limits_{j:ji\in A} x_{ji} = y_i \;\forall\, i\in V\backslash\{t\}, \\
x_{ij}\in\{0,1\}\;\forall\,(ij)\in A,\; y_i\in\{0,1\}\;\forall\, i\in V,
\end{array}
\right\} \mathcal{X}^{comb} \\
\qquad \left.
\begin{array}{l}
\left.
\begin{array}{l}
\sum\limits_{ij\in A} \delta^1_{ij} \leqslant d_1, \\
\delta^1_{ij}\in[0,D_{ij}]\;\forall\,(i,j)\in A,
\end{array}
\right\} \mathcal{U}^1 \\
\left.
\begin{array}{l}
\sum\limits_{i\in V}(p_i+\delta^2_i\cdot\hat{p_i})\cdot y_i \leqslant S, \\
\sum\limits_{i\in V}\delta^2_i \leqslant d_2, \\
\delta^2_i\in[0,2]\;\forall\, i\in V,
\end{array}
\right\} \mathcal{U}^2
\end{array}
\right\} \mathcal{X}^{num}
\end{array}
\right.
$$

1.4.1 Reformulation de la fonction objectif

$$
\begin{aligned}
&\min_{(x,y)} \max_{\delta^1_{ij}} \sum_{ij\in A} d_{ij}\cdot(1+\delta^1_{ij})\cdot x_{ij} \\
=&\min_{(x,y)} \max_{\delta^1_{ij}} \sum_{ij\in A} d_{ij}\cdot x_{ij} + \sum_{ij\in A} d_{ij}\cdot\delta^1_{ij}\cdot x_{ij} \\
=&\min_{(x,y)} \sum_{ij\in A} d_{ij}\cdot x_{ij} + \max_{\delta^1_{ij}} \sum_{ij\in A} d_{ij}\cdot\delta^1_{ij}\cdot x_{ij}
\end{aligned}
$$

Le problème devient alors :

$$
\left\{
\begin{array}{l}
\min\limits_{(x,y)} \sum\limits_{ij\in A} d_{ij}\cdot x_{ij} + \max\limits_{\delta^1_{ij}\in\mathcal{U}^1} \sum\limits_{ij\in A} d_{ij}\cdot\delta^1_{ij}\cdot x_{ij} \\
\quad s.c. : \\
\qquad x\in\mathcal{X}^{comb}, \\
\qquad x\in\mathcal{X}^{num}
\end{array}
\right.
$$

1.4.2 Problème interne lié aux variables δ^1_{ij}

$$
\left\{
\begin{array}{l}
\max\limits_{\delta^1_{ij}} \sum\limits_{ij\in A} d_{ij}\cdot\delta^1_{ij}\cdot x_{ij} \\
\quad s.c. : \\
\qquad \sum\limits_{ij\in A}\delta^1_{ij} \leqslant d_1 \\
\qquad \delta^1_{ij} \leqslant D_{ij} \qquad \forall ij\in A \\
\qquad \delta^1_{ij} \geqslant 0
\end{array}
\right.
$$

Remarque : x n'est pas considéré comme une variable ici.

1.4.3 Problème Dual

Le problème dual du problème interne est le suivant :

$$\begin{cases} \min\limits_{\alpha,\beta} \sum\limits_{ij\in A} D_{ij}\cdot\beta_{ij} + d_1\cdot\alpha \\ \quad s.c. : \\ \qquad \alpha + \beta_{ij} \geqslant d_{ij}\cdot x_{ij} \qquad \forall ij \in A \\ \qquad \alpha \geqslant 0 \\ \qquad \beta_{ij} \geqslant 0 \qquad \forall ij \in A \end{cases}$$

1.4.4 Reformulation des contraintes robustes

La contrainte robuste se réécrit :

$$\begin{aligned} &\sum_{i\in V}(p_i + \delta_i^2\cdot\hat{p_i})\cdot y_i \leqslant S \ \forall \delta^2 \in \mathcal{U}^2 \\ =&\begin{cases} \sum\limits_{i\in V} p_i\cdot y_i + \max\limits_{\delta^2}\sum\limits_{i\in V}\delta_i^2\cdot\hat{p_i}\cdot y_i \leqslant S \\ \quad s.c. : \\ \qquad \sum\limits_{i\in V}\delta_i^2 \leqslant d_2 \\ \qquad \delta_i^2 \leqslant 2 \ \forall i \in V \\ \qquad \delta_i^2 \geqslant 0 \ \forall i \in V \end{cases} \end{aligned}$$

1.4.5 Problème interne lié aux variables δ_j^2

Il s'agit alors de résoudre le problème interne suivant :

$$\begin{cases} \max\limits_{\delta_i^2} \sum\limits_{i\in V}\delta_i^2\cdot\hat{p_i}\cdot y_i \\ \quad s.c. : \\ \qquad \sum\limits_{i\in V}\delta_i^2 \leqslant d_2 \\ \qquad \delta_i^2 \leqslant 2 \qquad \forall i \in V \\ \qquad \delta_i^2 \geqslant 0 \qquad \forall i \in V \end{cases}$$

1.4.6 Problème Dual

Son dual s'écrit :

$$\begin{cases} \min\limits_{\gamma,\varepsilon} \sum\limits_{i\in V} 2\cdot\varepsilon_i + d_2\cdot\gamma \\ \quad s.c. : \\ \qquad \gamma + \varepsilon_i \geqslant \hat{p_i}\cdot y_i \qquad \forall i \in V \\ \qquad \gamma \geqslant 0 \\ \qquad \varepsilon_i \geqslant 0 \qquad \forall i \in V \end{cases}$$

Remarque : y n'est pas considéré comme une variable ici.

1.4.7 PLNE du problème robuste

On remplace l'objectif par la nouvelle expression trouvée en 1.4.1. Par dualité forte et 1.4.3, on a :

$$\left\{\begin{array}{l}\max\limits_{\delta^1_{ij}} \sum\limits_{ij\in A} d_{ij}\cdot\delta^1_{ij}\cdot x_{ij}\\ \quad s.c.:\\ \qquad \sum\limits_{ij\in A}\delta^1_{ij}\leqslant d_1\\ \qquad \delta^1_{ij}\leqslant D_{ij}\qquad \forall ij\in A\\ \qquad \delta^1_{ij}\geqslant 0\end{array}\right. = \left\{\begin{array}{l}\min\limits_{\alpha,\beta} \sum\limits_{ij\in A} D_{ij}\cdot\beta_{ij}+d_1\cdot\alpha\\ \quad s.c.:\\ \qquad \alpha+\beta_{ij}\geqslant d_{ij}\cdot x_{ij}\qquad \forall ij\in A\\ \qquad \alpha\geqslant 0\\ \qquad \beta_{ij}\geqslant 0\qquad \forall ij\in A\end{array}\right.$$

De même, par dualité forte et 1.4.6, on a :

$$\left\{\begin{array}{l}\max\limits_{\delta^2_i} \sum\limits_{i\in V} \delta^2_i\cdot\hat{p}_i\cdot y_i\\ \quad s.c.:\\ \qquad \sum\limits_{i\in V}\delta^2_i\leqslant d_2\\ \qquad \delta^2_i\leqslant 2\qquad \forall i\in V\\ \qquad \delta^2_i\geqslant 0\qquad \forall i\in V\end{array}\right. = \left\{\begin{array}{l}\min\limits_{\gamma,\varepsilon} \sum\limits_{i\in V} 2\cdot\varepsilon_i+d_2\cdot\gamma\\ \quad s.c.:\\ \qquad \gamma+\varepsilon_i\geqslant \hat{p}_i\cdot y_i\qquad \forall i\in V\\ \qquad \gamma\geqslant 0\\ \qquad \varepsilon_i\geqslant 0\qquad \forall i\in V\end{array}\right.$$

$$\left\{\begin{array}{l}\min\limits_{(x,y),(\alpha,\beta)} \sum\limits_{ij\in A} d_{ij}\cdot x_{ij}+\sum\limits_{ij\in A} D_{ij}\cdot\beta_{ij}+d_1\cdot\alpha\\ \quad s.c.:\\ \left.\begin{array}{l}\qquad \sum\limits_{i:ij\in A} x_{ij}=\sum\limits_{i:ji\in A} x_{ji}\ \forall\, j\in V\backslash\{s,t\},\\ \qquad \sum\limits_{j\in V} x_{sj}=1,\\ \qquad \sum\limits_{i\in V} x_{it}=1,\\ \qquad \sum\limits_{j:ij\in A} x_{ij}=y_i\ \forall\, i\in V\backslash\{s\},\\ \qquad \sum\limits_{j:ji\in A} x_{ji}=y_i\ \forall\, i\in V\backslash\{t\},\\ \qquad x_{ij}\in\{0,1\}\ \forall\,(ij)\in A, y_i\in\{0,1\}\ \forall\, i\in V,\end{array}\right\}\mathcal{X}^{comb}\\ \left.\begin{array}{l}\left.\begin{array}{l}\qquad \alpha+\beta_{ij}\geqslant d_{ij}\cdot x_{ij}\ \forall\, ij\in A\\ \qquad \alpha\geqslant 0,\ \beta_{ij}\geqslant 0\ \forall\, ij\in A\end{array}\right\}\mathcal{U}^1\\ \left.\begin{array}{l}\qquad \sum\limits_{i\in V} p_i\cdot y_i+\sum\limits_{i\in V} 2\varepsilon_i+d_2\cdot\gamma\leqslant S,\\ \qquad \gamma+\varepsilon_i\geqslant \hat{p}_i\cdot y_i\ \forall\, i\in V\\ \qquad \gamma\geqslant 0,\ \varepsilon_i\geqslant 0\ \forall\, i\in V\end{array}\right\}\mathcal{U}^2\end{array}\right\}\mathcal{X}^{num}\end{array}\right.$$

2 Implémentation

Etant donnée les modélisations données précedemment, il est intéressant d'implémenter ces différentes méthodes afin de les comparer. On dispose d'une batterie d'instances sur lesquelles on teste chacune des méthodes : Plans coupants, LazyCallback, Dualisation ainsi qu'une heuristique.

2.1 Plans coupants et LazyCallback

Algorithme 1 : Algorithme de plans coupants

Entrées :

- graphe G, Sommet départ s, Sommet arrivée t
- problème maître (MP), sous-problème (SPo,SP2)
- données de robustesse

Initialiser $\mathcal{U}^{1*} = \varnothing \qquad \mathcal{U}^{2*} = \varnothing$

Initialiser SPoValue=0, SP2=0, robustValue=0, $\hat{x}_{ij} = 0 \ \forall i,j \in A$, $\hat{y}_i = 0 \ \forall i \in V$, $\hat{\delta}^1_{ij} = 0 \ \forall i,j \in A$, $\hat{\delta}^2_i = 0 \ \forall i \in V$

tant que *SPoValue>robustValue || SP2Value>S* **faire**

  si *SPoValue>robustValue* **alors**

    $$(MP) \leftarrow \sum_{(ij)\in A} d_{ij} \cdot (1 + \hat{\delta}^1_{ij}) \cdot x_{ij} \leqslant z$$

  fin

  si *SP2Value>S* **alors**

    $$(MP) \leftarrow \sum_{i \in V} (p_i + \hat{\delta}^2_i \cdot \hat{p_i}) \cdot y_i \leqslant S$$

  fin

  Résoudre (MP)

  robustValue = z

  $\hat{x}_{ij} = x^\star_{ij} \ \forall i,j \in A$

  $\hat{y}_i = y^\star_i \ \forall i \in V$

  Résoudre (SPo)

  SPoValue = $\sum_{(ij)\in A} d_{ij} \cdot (1 + \delta^1_{ij}) \cdot x^\star_{ij}$

  $\hat{\delta}^1_{ij} = \delta^{1\star}_{ij} \ \forall i,j \in A$

  Résoudre (SP2)

  SP2Value = $\sum_{i \in V} (p_i + \delta^2_i \cdot \hat{p_i}) \cdot y^\star_i$

  $\hat{\delta}^2_i = \delta^{2\star}_i \ \forall i \in V$

fin

Pour l'algorithme de Branch & Cut, nous utilisons la fonction fournie par les solveurs d'optimisation appelée "Call back", qui nous permet d'ajouter les coupes de l'algorithme du plan de coupe à chaque nœud de la résolution de branchement et de coupe, ce qui permet de gagner du temps.

2.2 Heuristique

Il s'agit de trouver un plus court chemin. L'heuristique s'inspire donc de **l'algorithme $A^\star$** de plus court chemin.

2.2.1 L'algorithme $A^\star$

Cet algorithme est très rapide pour trouver un plus court chemin avec obstacles dans le cas où une information sur l'arrivée (Orientation Nord, Sud, Est, Ouest) est connue. L'idée est de calculer le coût d'un chemin depuis `s` en passant par le sommet `i` et jusqu'à `t` pour chaque sommet, en retenant son prédécesseur. Le noeud suivant de score le plus faible est alors choisi. Usuellement, il y a deux fonctions `g` et `h` qui permettent de calculer le score `f=g+h` :

- `g(i)` est la distance effective du noeud `s` au noeud `i`
- `h(i)` est la distance estimée entre le noeud `i` et le noeud `t`. Usuellement, on utilise la distance euclidienne qui est une borne inférieure de la distance réelle en cas de présence d'obstacles.

Données

- G : le graphe
- `s` : sommet de départ
- `t` : sommet d'arrivée
- `openList` : liste des sommets à traiter
- `closedList` : liste des sommets traités
- `h(i)` : distance estimée du sommet `i` au sommet d'arrivée `t`
- `g(i)` : distance réelle du sommet de départ `s` au sommet `i`
- `f(i) = h(i) + g(i)`
- `parent(i)` : parent du sommet `i`

Algorithme 2 : Algorithme $A^\star$

```
1  Initialiser closedList à vide;
2  Initialiser openList à vide;
3  Ajouter s à openList;
4  tant que openList non vide faire
5  |  Retirer le sommet u de openList tel que f(u) soit le plus petit;
6  |  si u == t alors
7  |  |  Retourner le chemin;
8  |  fin
9  |  sinon
10 |  |  Ajouter u à closedList;
11 |  fin
12 |  pour chaque successeur i de u faire
13 |  |  h(i) : estimation de la distance de i à t;
14 |  |  g_tmp ← g(u) + arc(u,i);
15 |  |  f_tmp ← g(u) + h(u);
16 |  |  si i est déjà dans openList avec un meilleur f(i) alors
17 |  |  |  On passe au voisin suivant;
18 |  |  fin
19 |  |  si i est déjà dans closedList avec un meilleur f(i) alors
20 |  |  |  On passe au voisin suivant;
21 |  |  fin
22 |  |  parent(i) ← u;
23 |  |  g(i) ← g_tmp;
24 |  |  g(i) ← f_tmp;
25 |  |  openList ← openList ∪ i;
26 |  |  closedList ← closedList \ i;
27 |  fin
28 |  closedList ← closedList \ u;
29 fin
```

2.2.2 Fonctions de score utilisées

- h(i) : Dans notre cas, cette fonction est fixée à 0. On aurait pu estimer la distance au noeud final par une heuristique gloutonne par exemple.
- g(i) : C'est la somme des durées sur les arcs et des poids aux sommets. Afin de prendre en compte la contrainte de robustesse, la recherche de plus court chemin se fait en saturant tous les arcs et noeuds du coût maximal possible ($\delta^1_{ij} = D_{ij}$ et $\delta^2_i = 2\hat{p}_i$).

2.2.3 Post-traitement

On obtient donc un plus court chemin dans le graphe augmenté des durées et poids maximums. Notons $P = \{i_1, i_2, \dots, i_l\}$ ce chemin. Afin d'obtenir une solution réalisable, on effectue donc le post-traitement 3 à la sortie de l'algorithme.

Algorithme 3 : Post-traitement

si $\sum_{i \in P} 2 * \hat{p}_i > d_2$ **alors**

  Trouver $(\delta_i)_{i \in P} = \arg\max \left\{ \sum_{i \in P} \delta^2_i \,\middle|\, \sum_{i \in P} \delta^2_i \leqslant d_2, \forall i \in P \; \delta^2_i \leqslant 2 \right\}$

fin

si $\sum_{(ij) \in P} D_{ij} > d_1$ **alors**

  Trouver $(\delta_{ij})_{(ij) \in P} = \arg\max \left\{ \sum_{(ij) \in P} \delta^1_{ij} \,\middle|\, \sum_{ij} \delta^1_{ij} \leqslant d_1, \forall (ij) \in P \; \delta^1_{ij} \leqslant D_{ij} \right\}$

fin

Il s'agit de 2 petits programmes linéaires continus qui sont résolus avec la bibliothèque CPLEX.

3 Résultats obtenus

comment the results of the table

Nous présentons les résultats obtenus dans un tableau (ci-joint) avec les temps d'exécution de chaque méthode. Une colonne PR (prix de robustesse) a été ajoutée en relation avec les différences entre les solutions robustes trouvées et les solutions du modèle statique. Pour toutes les méthodes, nous avons imposé une limite de temps d'exécution de 100 secondes. Un GAP a également été calculé par rapport à la solution optimale ou à la meilleure solution trouvée au moment de l'arrêt ou de la coupure d'exclusion des algorithmes. Pour le calcul du GAP de l'heuristique, nous utilisons la valeur optimale de chaque instance des autres méthodes. En raison de la taille de la plupart des instances, les algorithmes de résolution exacte sont coupés avant la limite de temps (Killed) car ils ont besoin d'une grande quantité de mémoire informatique et ne peuvent donc pas trouver de solution.

Les quatre méthodes résolvent la plupart des instances jusqu'à 250 sommets. Au delà, les méthodes de plans coupants et callback n'obtiennent pas de solutions dans le temps imparti. La méthode heuristique donne systématiquement une réponse en moins d'1 seconde. En effet, si la première solution rencontrée n'est pas réalisable suivant la contrainte de poids, alors l'heuristique ne renvoie pas de solution et termine. Lorsqu'elle trouve une solution, celle-ci est de coût très supérieur à la solution trouvée par les autres méthodes. Notons que, pour l'heuristique, lorsque le GAP est d'exactement 100%, c'est que le temps imparti a été dépassé sans renvoyer de solution.

Pour améliorer la compréhension des résultats, nous construisons un graphique de performance des résultats en fonction du temps d'exécution de la résolution des instances (figure 1).

On observe le nombre d'instances résolues en fonction du temps de résolution pour chacune des méthodes proposées. Les méthodes de plans coupants et callback semblent être les moins efficaces puisqu'elles résolvent moins de 30 instances. Notons néanmoins que ces méthodes ne renvoient une solution que si elle est optimale, ce qui explique les faibles performances apparentes de celles-ci. La méthode par dualisation résout environ 50 instances mais avec des temps d'exécution très longs. Enfin, la métode heuristique est la plus performante sur ce diagramme puisque les instances sont résolues très rapidement. Néanmoins, on paye cela dans la valeur des

solutions renvoyées qui sont souvent bien moins bonnes que pour les autres méthodes. Un point intéressant de l'heuristique est son utilisation comme solution réalisable initiale dans la dualisation.

	callBack		cuttingPlane		dual		heuristic		
Instance	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**PR**
20_BAY	0.41	7.14%	4.16	7.14%	0.06	0.0	1.36	0.0%	0.19%
20_COL	0.29	0.0%	2.06	0.0%	0.05	0.0	0.0	12.83%	0.29%
20_NY	0.12	0.0%	0.2	0.0%	0.03	0.0	0.0	132.73%	0.18%
40_BAY	0.93	0.0%	0.56	0.0%	0.16	0.0	0.0	33.08%	0.49%
40_NY	0.35	0.0%	0.46	0.0%	0.14	0.0	0.0	121.26%	0.28%
40_COL	14.27	6.65%	100.53	0.0%	0.56	0.0	0.0	1.66%	0.28%
60_BAY	1.44	0.0%	1.76	0.0%	0.55	0.0	0.0	251.0%	0.07%
60_COL	1.08	0.0%	0.72	0.0%	0.56	0.0	0.02	237.98%	0.21%
60_NY	1.72	0.0%	1.28	0.0%	0.7	0.0	0.0	177.16%	0.31%
80_BAY	3.36	0.0%	3.94	0.0%	1.92	0.0	0.01	82.42%	0.23%
80_NY	3.44	0.0%	4.59	0.0%	1.75	0.0	0.0	177.3%	0.31%
80_COL	2.67	0.0%	3.59	0.0%	1.56	0.0	0.01	105.25%	0.26%
100_NY	8.42	0.0%	7.3	0.0%	3.17	0.0	0.0	100%	0.28%
100_BAY	12.45	0.0%	10.1	0.0%	3.33	0.0	0.0	82.7%	0.23%
100_COL	8.66	0.0%	8.86	0.0%	3.14	0.0	0.0	318.33%	0.18%
120_NY	7.81	0.0%	8.67	0.0%	6.48	0.0	0.0	100%	0.26%
120_COL	11.87	0.0%	18.4	0.0%	6.33	0.0	0.0	100%	0.18%
120_BAY	21.49	0.0%	39.39	0.0%	6.53	0.0	0.0	65.54%	0.22%
140_BAY	32.5	0.0%	36.72	0.0%	12.15	0.0	0.01	112.6%	0.22%
140_NY	23.37	0.0%	31.36	0.0%	11.9	0.0	0.01	38.86%	0.5%
140_COL	26.46	0.0%	35.59	0.0%	11.81	0.0	0.0	309.64%	0.2%
160_NY	32.18	0.0%	56.35	0.0%	25.51	0.0	0.0	434.89%	0.21%
160_COL	54.02	0.0%	74.65	0.0%	24.2	0.0	0.0	309.81%	0.2%
160_BAY	50.68	0.0%	70.41	0.0%	21.85	0.0	0.26	91.28%	0.27%
180_NY	100.42	0.0%	92.66	4.74%	33.21	0.0	0.02	460.72%	0.16%
180_COL	82.43	0.0%	85.5	0.0%	32.27	0.0	0.02	100%	0.52%
180_BAY	101.4	0.0%	104.9	0.0%	33.19	0.0	0.02	104.81%	0.19%
200_NY	84.59	0.0%	58.95	0.0%	49.46	0.0	0.0	50.07%	0.32%
200_COL	79.97	0.0%	51.34	0.0%	49.19	0.0	0.0	100%	0.4%
200_BAY	100.97	0.0%	108.08	0.0%	69.88	0.0	0.02	105.02%	0.19%
250_NY	106.06	0.0%	106.28	0.0%	128.62	0.0	0.0	201.84%	0.11%
250_COL	108.62	0.0%	106.3	0.0%	120.77	0.0	0.0	213.12%	0.19%
250_BAY	101.13	0.0%	106.26	17.03%	123.68	0.0	0.0	189.59%	0.0%
300_NY	109.02	100%	111.58	0.0%	258.5	0.0	0.02	239.86%	0.0%

	callBack		cuttingPlane		dual		heuristic		
Instance	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**PR**
300_COL	107.11	100%	110.7	0.0%	256.02	0.0	0.03	471.46%	0.0%
300_BAY	108.19	100%	110.87	0.0%	256.24	0.0	0.02	127.18%	0.0%
350_NY	134.28	100%	176.27	100%	473.67	0.0	0.03	156.05%	0.35%
350_COL	138.77	100%	143.77	100%	484.8	0.0	0.02	240.66%	0.24%
350_BAY	131.44	100%	134.46	100%	470.98	0.0	0.02	82.7%	0.25%
400_BAY	120.32	100%	339.16	100%	840.0	0.0	0.02	159.41%	0.24%
400_NY	119.55	100%	239.86	100%	896.05	0.0	0.02	129.07%	0.35%
400_COL	117.74	100%	246.32	100%	913.21	0.0	0.01	142.65%	0.49%
450_NY	196.83	100%	403.19	100%	1412.15	0.0	0.02	171.35%	0.28%
450_COL	194.08	100%	400.63	100%	1483.05	0.0	0.02	142.83%	0.49%
450_BAY	191.31	100%	389.78	100%	892.74	0.0	0.05	159.69%	0.24%
500_NY	303.25	100%	632.05	100%	3174.47	0.0	0.02	298.18%	0.23%
500_COL	303.29	100%	601.66	100%	2534.89	0.0	0.04	143.01%	0.49%
500_BAY	307.96	100%	598.6	100%	2313.79	0.0	0.05	274.55%	0.2%
550_NY	443.76	100%	890.17	100%	3424.34	0.0	0.02	304.37%	-
550_COL	441.42	100%	890.17	100%	3962.39	0.0	0.04	143.24%	-
550_BAY	420.87	100%	936.67	100%	4083.87	0.0	0.05	198.86%	-
600_NY	642.2	100%	1307.93	100%	-	-	0.03	0.0%	-
600_COL	630.03	100%	1285.71	100%	-	-	0.05	0.0%	-
600_BAY	585.53	100%	1274.45	100%	-	-	0.09	0.0%	-
650_NY	884.75	100%	1830.78	100%	-	-	0.05	0.0%	-
650_COL	898.35	100%	1820.46	100%	-	-	0.07	0.0%	-
650_BAY	888.84	100%	1773.85	100%	-	-	0.05	0.0%	-
700_NY	2729.83	100%	2570.98	100%	-	-	0.05	0.0%	-
700_COL	1223.32	100%	2478.9	100%	-	-	0.06	0.0%	-
700_BAY	1198.2	100%	2484.92	100%	-	-	0.06	0.0%	-
750_COL	-	-	3174.38	100%	-	-	0.1	0.0%	-
750_BAY	-	-	3443.77	100%	-	-	0.08	0.0%	-
750_NY	-	-	-	-	-	-	0.05	0.0%	-
800_NY	-	-	-	-	-	-	0.06	0.0%	-
800_COL	-	-	-	-	-	-	0.11	0.0%	-
800_BAY	-	-	-	-	-	-	0.12	0.0%	-
850_COL	-	-	-	-	-	-	0.13	0.0%	-
850_BAY	-	-	-	-	-	-	0.1	0.0%	-
850_NY	-	-	-	-	-	-	0.08	0.0%	-

	callBack		cuttingPlane		dual		heuristic		
Instance	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**PR**
900_BAY	-	-	-	-	-	-	0.1	0.0%	-
900_COL	-	-	-	-	-	-	0.14	0.0%	-
900_NY	-	-	-	-	-	-	0.08	0.0%	-
950_NY	-	-	-	-	-	-	0.08	0.0%	-
950_COL	-	-	-	-	-	-	0.12	0.0%	-
950_BAY	-	-	-	-	-	-	0.13	0.0%	-
1000_BAY	-	-	-	-	-	-	0.35	0.0%	-
1000_NY	-	-	-	-	-	-	0.11	0.0%	-
1000_COL	-	-	-	-	-	-	0.16	0.0%	-
1100_BAY	-	-	-	-	-	-	0.06	100%	-
1100_NY	-	-	-	-	-	-	0.09	0.0%	-
1100_COL	-	-	-	-	-	-	0.18	0.0%	-
1200_COL	-	-	-	-	-	-	0.18	0.0%	-
1200_NY	-	-	-	-	-	-	0.05	0.0%	-
1200_BAY	-	-	-	-	-	-	0.03	100%	-
1300_BAY	-	-	-	-	-	-	0.08	100%	-
1300_NY	-	-	-	-	-	-	0.07	100%	-
1300_COL	-	-	-	-	-	-	0.22	100%	-
1400_NY	-	-	-	-	-	-	0.06	100%	-
1400_COL	-	-	-	-	-	-	0.27	100%	-
1400_BAY	-	-	-	-	-	-	0.03	100%	-
1500_BAY	-	-	-	-	-	-	0.1	100%	-
1500_NY	-	-	-	-	-	-	0.07	0.0%	-
1500_COL	-	-	-	-	-	-	0.45	100%	-
1600_NY	-	-	-	-	-	-	0.06	100%	-
1600_BAY	-	-	-	-	-	-	0.11	100%	-
1600_COL	-	-	-	-	-	-	0.33	0.0%	-
1700_NY	-	-	-	-	-	-	0.09	100%	-
1700_BAY	-	-	-	-	-	-	0.12	100%	-
1700_COL	-	-	-	-	-	-	0.46	100%	-
1800_BAY	-	-	-	-	-	-	0.21	100%	-
1800_COL	-	-	-	-	-	-	0.42	0.0%	-
1800_NY	-	-	-	-	-	-	0.07	100%	-
1900_NY	-	-	-	-	-	-	0.11	100%	-
1900_BAY	-	-	-	-	-	-	0.14	100%	-

	callBack		cuttingPlane		dual		heuristic		
Instance	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**Temps (s)**	**GAP**	**PR**
1900_COL	-	-	-	-	-	-	0.52	0.0%	-
2000_COL	-	-	-	-	-	-	0.5	100%	-
2000_BAY	-	-	-	-	-	-	0.22	100%	-
2000_NY	-	-	-	-	-	-	0.27	0.0%	-
2100_BAY	-	-	-	-	-	-	0.13	100%	-
2100_COL	-	-	-	-	-	-	0.59	0.0%	-
2100_NY	-	-	-	-	-	-	0.28	100%	-
2200_NY	-	-	-	-	-	-	0.35	100%	-
2200_COL	-	-	-	-	-	-	0.73	0.0%	-
2200_BAY	-	-	-	-	-	-	0.19	100%	-
2300_BAY	-	-	-	-	-	-	0.19	100%	-
2300_COL	-	-	-	-	-	-	0.79	0.0%	-
2300_NY	-	-	-	-	-	-	0.37	100%	-
2400_COL	-	-	-	-	-	-	0.85	100%	-
2400_NY	-	-	-	-	-	-	0.1	0.0%	-
2400_BAY	-	-	-	-	-	-	0.17	100%	-
2500_COL	-	-	-	-	-	-	0.84	0.0%	-
2500_BAY	-	-	-	-	-	-	0.5	100%	-
2500_NY	-	-	-	-	-	-	0.14	0.0%	-

4 Conclusion

Le problème du plus court chemin robuste est un problème multi-application, qui permet de faire face à diverses adversités pouvant être produites par des agents externes, permettant ainsi la modélisation d'une solution résiliente et conservatrice.

Dans ce travail, ce problème a été modélisé mathématiquement et quatre méthodes de résolution ont été implémentées, parmi lesquelles nous soulignons la rapidité des heuristiques utilisées, ainsi que l'efficacité dans de petites instances des méthodes exactes, où les solutions trouvées sont optimales.

Une façon possible d'améliorer les résultats obtenus est d'utiliser un mélange de ces méthodes pour augmenter la vitesse, l'étude de la modélisation mathématique, entre autres.

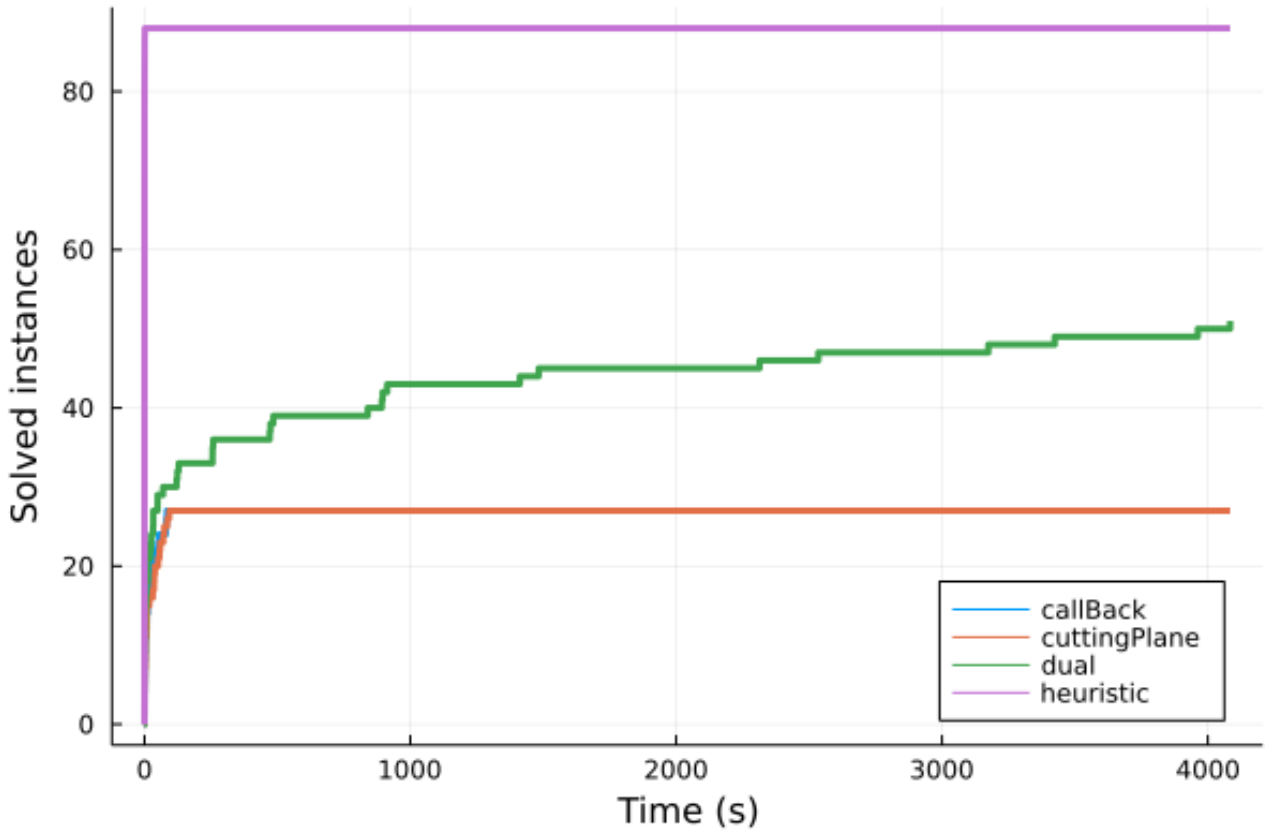

Figure 1 – Diagramme de performance des 4 méthodes utilisées

Annexes

Instance	Noeuds rencontrés	Valeur
20_BAY	[15, 4, 13, 5, 17]	14311
20_COL	[1, 2, 3, 4, 5,]	6305
20_NY	[1, 2, 3, 4,]	4082
40_BAY	[1, 2, 3, 4, 5, 6,]	9356
40_NY	[1, 2, 3, 4, 5, 6,]	7864
40_COL	[1, 2, 3, 4, 5, 6,]	14120
60_BAY	[1, 2, 3, 4, 5, 6,]	2465
60_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]	6281
60_NY	[1, 2, 3, 4, 5, 6, 7, 8,]	11589
80_BAY	[1, 2, 3, 4, 5, 6, 7,]	6009
80_NY	[1, 2, 3, 4, 5, 6, 7, 8,]	11589
80_COL	[1, 2, 3, 4, 5, 6, 7, 8,]	7057
100_NY	[]	7573
100_BAY	[1, 2, 3, 4, 5, 6, 7,]	6009
100_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,]	5348
120_NY	[]	5372
120_COL	[]	5348
120_BAY	[1, 2, 3, 4, 5, 6, 7, 8,]	7468
140_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9,]	7109
140_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]	22818
140_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]	5884
160_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,]	4682
160_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]	5884
160_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9,]	7097
180_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,]	4470
180_COL	[]	15020
180_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9,]	6642
200_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]	20002
200_COL	[]	11629
200_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9,]	6642
250_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	11239
250_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,]	9912
250_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]	6534
300_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	10214

Instance	Noeuds rencontrés	Valeur
300_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	5754
300_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]	9592
350_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,]	15632
350_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,]	8268
350_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,]	11944
400_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	11767
400_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,]	15632
400_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,]	15256
450_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,]	16295
450_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,]	15256
450_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	11767
500_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,]	11990
500_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,]	15256
500_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,]	8669
550_NY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,]	11757
550_COL	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15,]	15256
550_BAY	[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,]	7307
600_NY	[463, 461, 549, 547, 239, 335, 353, 407, 108, 574, 162, 560, 85, 584, 275, 281, 393, 515, 488]	47575
600_COL	[83, 189, 544, 552, 545, 283, 2, 170, 318, 223, 113, 301, 346, 352, 389]	37151
600_BAY	[269, 506, 289, 130, 565, 381, 393, 358, 570, 550, 420, 120, 394, 567, 588]	33133
650_NY	[576, 385, 526, 287, 309, 239, 444, 581, 325, 641, 109, 243, 174, 62, 602, 120, 348] [349, 646, 621]	43510
650_COL	[83, 189, 311, 217, 581, 38, 2, 8, 20, 527, 584, 602, 608, 260, 389]	37205
650_BAY	[269, 506, 253, 623, 611, 287, 393, 540, 535, 550, 60, 365, 394, 642, 588]	33154
700_NY	[576, 385, 526, 539, 364, 359, 324, 360, 325, 392, 508, 306, 182, 62, 687, 435, 619] [620, 478, 621]	43548
700_COL	[83, 189, 311, 217, 545, 218, 2, 1, 668, 89, 317, 192, 281, 260, 389]	37231
700_BAY	[664, 669, 514, 375, 257, 550, 445, 511, 393, 287, 565, 698, 591, 608, 599, 602, 695, 677]	46207
750_COL	[83, 189, 233, 130, 545, 444, 2, 18, 145, 89, 113, 301, 608, 352, 389]	37273
750_BAY	[664, 669, 514, 375, 257, 550, 497, 433, 393, 287, 565, 623, 705, 703, 269, 728, 600, 677]	46244
750_NY	[621, 478, 620, 619, 120, 432, 277, 18, 306, 508, 723, 696, 715, 324, 496, 323, 578, 691, 707]	39448
800_NY	[621, 478, 620, 348, 120, 432, 178, 399, 704, 706, 108, 396, 581, 324, 651, 713, 712, 758, 707]	39471
800_COL	[83, 189, 183, 65, 197, 538, 2, 734, 199, 336, 584, 301, 570, 352, 389]	37304
800_BAY	[588, 642, 628, 566, 372, 632, 339, 795, 765, 287, 721, 680, 686, 748, 766]	38747
850_COL	[213, 188, 836, 199, 283, 733, 285, 845, 305, 667, 493, 543, 710, 666]	28874

Instance	Noeuds rencontrés	Valeur
850_BAY	[659, 656, 642, 519, 460, 313, 298, 670, 595, 424, 330, 165, 395, 718, 754, 786]	40908
850_NY	[576, 385, 382, 678, 752, 359, 444, 360, 325, 392, 766, 748, 174, 85, 818, 212, 799] [773, 759, 621]	43641
900_BAY	[820, 616, 629, 579, 599, 719, 354, 593, 456, 850, 424, 330, 782, 470, 767, 371, 701, 844, 854]	51585
900_COL	[213, 188, 836, 199, 283, 733, 585, 145, 809, 858, 332, 724, 710, 666]	28912
900_NY	[576, 385, 382, 678, 752, 359, 444, 360, 396, 392, 888, 384, 391, 105, 184, 120, 256] [773, 478, 621]	43671
950_NY	[576, 385, 382, 678, 942, 320, 324, 360, 325, 447, 588, 162, 886, 915, 54, 212, 799] [620, 759, 621]	43712
950_COL	[213, 188, 268, 199, 582, 452, 56, 816, 761, 745, 332, 618, 576, 666]	28944
950_BAY	[820, 616, 925, 530, 540, 926, 452, 593, 218, 438, 424, 330, 720, 797, 341, 371, 432, 844, 854]	51623
1000_BAY	[659, 878, 488, 519, 614, 198, 112, 927, 559, 875, 330, 412, 1000, 918, 698, 786, 993]	45947
1000_NY	[576, 385, 382, 708, 456, 239, 444, 360, 582, 280, 415, 553, 278, 523, 977, 995, 256] [773, 478, 621]	43746
1000_COL	[213, 188, 268, 405, 839, 298, 984, 85, 883, 745, 631, 724, 710, 666]	28986
1100_BAY	Any[]	$+\infty$
1100_NY	[621, 646, 620, 256, 829, 818, 994, 589, 1017, 694, 507, 866, 697, 585, 641, 696, 581] [324, 928, 443, 959, 1095, 1066]	58921
1100_COL	[88, 238, 331, 661, 521, 681, 611, 599, 652, 825, 1043, 528, 696, 529, 1053]	39175
1200_COL	[88, 238, 471, 536, 361, 1114, 406, 793, 972, 1103, 960, 528, 1000, 1001, 1053]	39228
1200_NY	[1066, 1095, 578, 1187, 651, 1153, 916, 396, 838, 585, 697, 866, 855, 1093, 1070, 1117, 1082] [1053, 875, 759, 722, 1144]	67351
1200_BAY	Any[]	$+\infty$
1300_BAY	Any[]	$+\infty$
1300_NY	Any[]	$+\infty$
1300_COL	Any[]	$+\infty$
1400_NY	Any[]	$+\infty$
1400_COL	Any[]	$+\infty$
1400_BAY	Any[]	$+\infty$
1500_BAY	Any[]	$+\infty$
1500_NY	[1144, 621, 646, 1183, 1053, 1254, 1117, 1070, 1044, 507, 866, 1445, 1431, 108, 696, 360, 324] [579, 1235, 1223, 1469, 1470, 1296]	61919
1500_COL	Any[]	$+\infty$
1600_NY	Any[]	$+\infty$
1600_BAY	Any[]	$+\infty$

Instance	Noeuds rencontrés	Va
1600_COL	[88, 538, 471, 188, 759, 409, 1576, 359, 1487, 782, 767, 774, 479, 1398, 1053]	3
1700_NY	Any[]	
1700_BAY	Any[]	
1700_COL	Any[]	
1800_BAY	Any[]	
1800_COL	[88, 931, 1290, 266, 1708, 1352, 1533, 1091, 1601, 1253, 383, 330, 696, 1398, 1053]	3
1800_NY	Any[]	
1900_NY	Any[]	
1900_BAY	Any[]	
1900_COL	[88, 362, 213, 518, 1534, 967, 178, 1510, 899, 930, 1679, 1528, 1674, 1001, 1053]	3
2000_COL	Any[]	
2000_BAY	Any[]	
2000_NY	[1792, 1604, 1271, 1257, 1286, 2000, 1482, 1383, 662, 248, 1490, 1694, 1393, 1661, 1858, 1394, 1401] [1630, 1927, 1976, 1741, 1950, 1973]	5
2100_BAY	Any[]	
2100_COL	[88, 1395, 249, 1007, 904, 1467, 178, 793, 482, 2082, 1043, 1030, 1430, 1924, 2048, 2013]	4
2100_NY	Any[]	
2200_NY	Any[]	
2200_COL	[88, 1297, 1353, 212, 1180, 1635, 861, 1932, 1487, 591, 338, 946, 669, 2006, 1997, 2033, 2153]	4
2200_BAY	Any[]	
2300_BAY	Any[]	
2300_COL	[88, 238, 701, 536, 904, 2050, 1116, 555, 293, 1330, 1669, 1671, 1674, 2032, 2152, 2031, 2227] [2273, 2259, 2272]	4
2300_NY	Any[]	
2400_COL	Any[]	
2400_NY	[1785, 1617, 1701, 1852, 595, 1309, 1556, 1738, 2350, 2368, 1584, 1918, 1414, 1799, 2061, 2369, 1952] [2140, 2319, 2336, 2353, 1954, 1974]	6
2400_BAY	Any[]	
2500_COL	[88, 189, 249, 2015, 1840, 1235, 2014, 138, 567, 2280, 590, 946, 2248, 620, 1636, 1937, 1917] [1888, 2147, 2088]	5
2500_BAY	Any[]	
2500_NY	[1792, 1296, 1746, 1587, 595, 2382, 1959, 2393, 2373, 2029, 1570, 2463, 2112, 1856, 2278, 2374, 1969] [2104, 2316, 2434, 2472, 2317, 1973]	6

Remarque : Dans certaines des solutions, les noeuds rencontrés apparaissent dans l'ordre croissant et non dans l'ordre de parcourt. Nous n'avons pas eu le temps de refaire les tests pour retrouver les solutions complètes.