Projet MPRO - Modélisation papier

Tadeo Delapalme, Dimitri de Saint Guilhem

Février 2025

1 Introduction

Nous avons implémenté quatre méthodes de résolution pour le VRP robuste avec contrainte de capacité, une méthode heuristique et trois méthodes fondées sur une modélisation compacte du problème : dualisation de la robustesse, plans coupants et branch and cut. Nous avons essayé de raffiner ces trois méthodes en proposant une heuristique pour résoudre le sous problème de branch and cut et plans coupants, ainsi qu'en fournissant une solution initiale au solver.

2 Heuristique

2.1 Algorithme de Clarke et Wright

L'algorithme de Clarke et Wright (CW) nous permet d'obtenir rapidement une première solution admissible au problème. L'algorithme est concu pour la version statique du VRP, et il a été adapté en prenant pour chaque arc (i, j) la borne supérieure du temps de trajet, c'est à dire :

$$t'_{i,j} = t_{i,j} + (\hat{t}_i + \hat{t}_j) + 2\hat{t}_i\hat{t}_j$$

au lieu de :

$$t'_{i,j} = t_{i,j} + \delta^1_{i,j}(\hat{t}_i + \hat{t}_j) + \delta^2_{i,j}\hat{t}_i\hat{t}_j$$

avec $(\delta^1, \delta^2) \in R$, où :

$$R = \{(\delta^1, \delta^2) \in [0,1] \times [0,2] : \sum_{(i,j)\in A} \delta^1_{ij} \leq T, \sum_{(i,j)\in A} \delta^2_{ij} \leq T^2\}$$

L'algorithme fonctionne de la facon suivante :

- Initialisation : Chaque client est desservi par un véhicule distinct (autant de routes qu'il y a de clients).
- Calcul des économies : Pour chaque paire de routes, on calcule l'économie (ou gain) qui résulterait de leur fusion :
$$e_{i,j} = t'_{i,0} + t'_{0,j} - t'_{i,j}$$
- Fusion des routes : Fusion des deux routes qui offrent la plus grande économie, à condition que cela ne viole pas les contraintes de capacité du véhicule.
- Répétition : Après la fusion, mise à jour la liste des routes et des fusions possibles et répétition de l'étape de fusion.
- Terminaison : L'algorithme se termine lorsqu'il n'y a plus de fusions possibles.

2.2 Prolongation par recherche locale

2.2.1 Recherche 2-opt sur les sous-tours

Afin d'améliorer la valeur de l'objectif, nous avons appliqué une recherche 2-opt (suppression de 2 arrêtes et reconnexion de la seule autre manière possible).

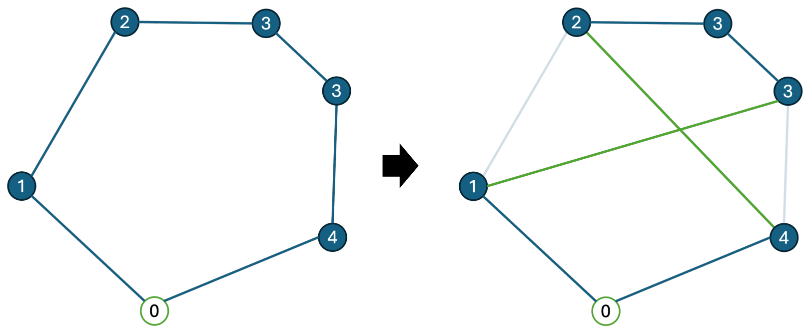

Figure 1: Échange 2-opt sur un sous-tour

L'algorithme fonctionne de la façon suivante :

Algorithm 1 Recherche 2-opt sur les sous-tours

```
function EXPLO_2OPT(solution_de_départ)
    solution_courante ← solution_de_départ
    for route in solution_courante do
        amélioration ← True
        while amélioration do
            amélioration ← False
            voisins ← voisins_2opt(route)
            if coût(meilleur_voisin) < coût(solution_courante) then
                solution_courante ← meilleur_voisin
                amélioration ← True
            end if
        end while
    end for
    return solution_courante
end function
```

Le coût d'une route est calculé de la même manière qu'avec l'algorithme de Clarke et Wright, c'est à dire avec la borne supérieure du temps de trajet.

2.2.2 Recherche 3-opt sur les sous-tours

Le même algorithme à été appliqué avec une recherche sur les voisins 3-opt au lieu de 2-opt.

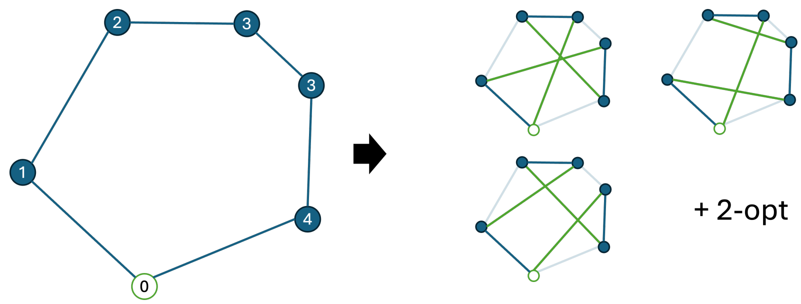

Figure 2: Échange 3-opt sur un sous-tour

2.2.3 Recherche 2-opt entre tours

Nous avons ensuite implémenté un algorithme permettant de faire des permutation entre routes, à l'aide d'échanges 2-opt entre tours.

L'algorithme fonctionne de la façon suivante :

Algorithm 2 Recherche 2-opt entre les sous-tours

```
function EXPLO_2OPT_ENTRE_TOURS(solution_de_départ)
    solution_courante ← solution de l'algorithme 1
    amélioration ← True
    while amélioration do
        amélioration ← False
        for (route_i, route_j) in solution_courante do
            for 1 < k < len(route_i) − 1 and 1 < l < len(route_j) − 1 do
                voisin_1, voisin_2 ← échange_2opt(route_i, route_j, k, l)
                voisin_1 ← Explo_2opt(voisin_1)
                voisin_2 ← Explo_2opt(voisin_2)
                if coût(voisin_1) < coût(voisin_2) then
                    meilleur_voisin_courant ← voisin_1
                else
                    meilleur_voisin_courant ← voisin_2
                end if
                if coût(meilleur_voisin_courant) < coût(solution_courante) then
                    meilleur_voisin ← meilleur_voisin_courant
                    amélioration ← True
                end if
            end for
            solution_courante ← meilleur_voisin
        end for
    end while
    return solution_courante
end function
```

Étant donné que l'algorithme 2 considère maintenant des solutions complètes et non des sous-tours, il est possible de calculer le coût réel des solutions explorées en résolvant le sous-problème de la méthode des plans coupants :

$$\begin{aligned}
\max_{\delta^1,\delta^2} \quad & \sum_{(i,j)\in A} \left(t_{ij} + \delta^1_{ij}(\hat{t}_i + \hat{t}_j) + \delta^2_{ij}\hat{t}_i\hat{t}_j\right) x^*_{ij} \\
\text{s.t.} \quad & \sum_{(i,j)\in A} \delta^1_{ij} \le T \\
& \sum_{(i,j)\in A} \delta^2_{ij} \le T^2 \\
& \delta^1_{ij} \in [0,1], \delta^2_{ij} \in [0,2] \qquad\qquad \forall ij \in A
\end{aligned}$$

Ce problème peut se résoudre soit comme un PL, soit de manière heuristique (cf. 4.3), ce qui ne rend pas très lourd le calcul du coût total réel d'une solution. Nous avons donc implémenté une version utilisant la borne supérieur des temps de trajets, et une version utilisant le coût réel.

2.3 Comparaison des heuristiques

2.3.1 Méthode de test

Les heuristiques ont été testées sur toutes les instances proposées sans limites de temps, et un gap à été calculé à partir de la relaxation continue de la modélisation duale.

2.3.2 Résultats

Temps de calcul pour les heuristiques qui effectuent des échanges entre sous tours plusieurs ordres de grandeur au dessus des autres méthodes.

Performance en terme de gap similaires entre les méthodes. meilleurs résultats avec real cost

pas de garantie sur le temps de calcul, plage de variation assez importante (possibilité de mettre une limite de temps)

peu adapté aux instances non euclidiennes (ne regarde pas les routes dans les 2 sens)

La borne inf étant mauvaise, le gap trouvé ne correspond pas du tout au gap avec l'optimalité

3 Méthode duale

On a implémenté la formulation duale suivante en Julia en la résolvant avec CPLEX.Nous n'avons ajouté de coupe pour obtenir les résultats de l'algorithme de dualisation.

Cependant, nous avons essayé d'initialiser la résolution par CPLEX avec une solution x fournie par une heuristique. Nous n'avons pas donné les autres variables.

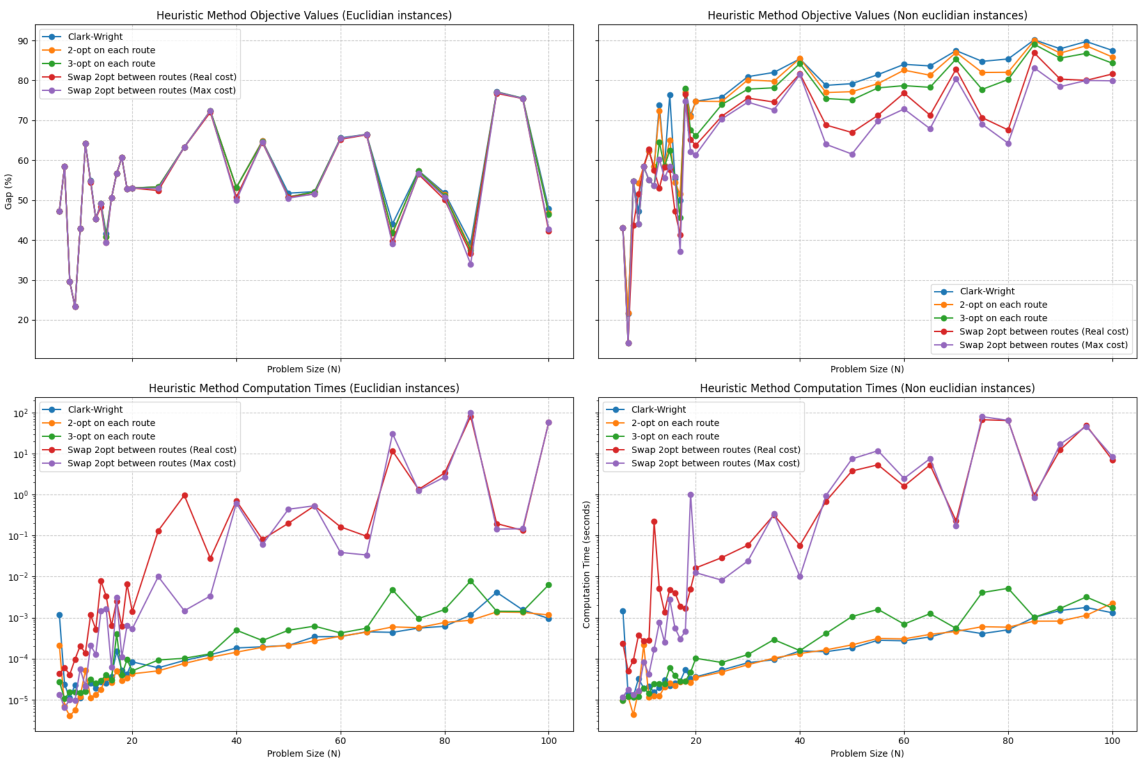

Figure 3: Temps de calcul et gap des différentes heuristiques testées sur les instances proposées

$$
\begin{aligned}
\min_{x,u,\lambda^1,\lambda^2,\mu^1,\mu^2} \; & \sum_{ij\in A} t_{ij}x_{ij} + \lambda^1 T + \lambda^2 T^2 + \sum_{ij\in A} \mu^1_{ij} + 2\mu^2_{ij} \\
\text{s.t.} \quad & \sum_{i\in V} x_{ij} = 1, && \forall j \in V\setminus\{1\}, \\
& \sum_{j\in V} x_{ij} = 1, && \forall i \in V\setminus\{1\}, \\
& u_j - u_i \geq d_j - C(1-x_{ij}), && \forall (i,j) \in \{(i,j)\in V\setminus\{1\} : i\neq j, d_i + d_j \leq C\}, \\
& \lambda^1 + \mu^1_{ij} \geq (\hat{t}_i + \hat{t}_j)x_{ij} && \forall ij \in A, \\
& \lambda^2 + \mu^2_{ij} \geq \hat{t}_i\hat{t}_j x_{ij} && \forall ij \in A, \\
& \\
& \lambda^1 \geq 0, \lambda^2 \geq 0 \\
& \mu^1_{ij} \geq 0, \mu^2_{ij} \geq 0 && \forall ij \in A, \\
& d_i \leq u_i \leq C, && \forall i \in V\setminus\{1\}, \\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A.
\end{aligned}
$$

4 Résolution par plans coupants et branch-and-cut

4.1 Plans coupants

Nous avons résolu le problème maître suivant qui est un PLNE avec CPLEX.

$$
\begin{aligned}
\min_{x,u,\Theta} \; & \Theta \\
\text{s.t.} \quad & \sum_{(i,j)\in A} \left(t_{ij} + \delta^1_{ij}(\hat{t}_i + \hat{t}_j) + \delta^2_{ij}\hat{t}_i\hat{t}_j\right) x_{ij} - \Theta \leq 0, && \forall \delta^1, \delta^2 \in R, \\
& \sum_{i\in V} x_{ij} = 1, && \forall j \in V\setminus\{1\}, \\
& \sum_{j\in V} x_{ij} = 1, && \forall i \in V\setminus\{1\}, \\
& u_j - u_i \geq d_j - C(1-x_{ij}), && \forall (i,j) \in \{(i,j)\in V\setminus\{1\} : i\neq j, d_i + d_j \leq C\}, \\
& d_i \leq u_i \leq C, && \forall i \in V\setminus\{1\}, \\
& x_{ij} \in \{0,1\}, && \forall (i,j) \in A.
\end{aligned}
$$

Où

$$
R = \{(\delta^1, \delta^2) \in [0,1]\times[0,2] : \sum_{(i,j)\in A} \delta^1_{ij} \leq T, \sum_{(i,j)\in A} \delta^2_{ij} \leq T^2\}
$$

.

Puis à chaque solution x^* du problème maître, nous résolvons avec CPLEX le sous-problème qui est un PL pour trouver le pire scénario pour la tournée $x*$. Ceci nous donne $(\delta^{1*}, \delta^{2*})$ qui nous permet d'ajouter une contrainte dans le problème maître.

$$
\begin{aligned}
\max_{\delta^1,\delta^2} \quad & \sum_{(i,j)\in A} \left(t_{ij} + \delta^1_{ij}(\hat{t}_i + \hat{t}_j) + \delta^2_{ij}\hat{t}_i\hat{t}_j\right) x^*_{ij} \\
\text{s.t.} \quad & \sum_{(i,j)\in A} \delta^1_{ij} \leq T \\
& \sum_{(i,j)\in A} \delta^2_{ij} \leq T^2 \\
& \delta^1_{ij} \in [0,1], \delta^2_{ij} \in [0,2] \qquad\qquad \forall ij \in A
\end{aligned}
$$

Le défaut de la méthode est qu'il faut recréer et résoudre un PLNE pour le problème maître à chaque itération de l'algorithme.

4.2 Branch-and-cut

Nous corrigeons le défaut de l'algorithme des plans coupants en ne créeant qu'un unique PLNE pour le problème maître. A chaque tournée réalisable entière x rencontrée, on résout le sous-problème pour la tournée x ce qui nous donne une nouvelle contrainte. On l'ajoute au problème par un LazyCallback dans CPLEX.

Nous n'avons pas ajouté d'autres coupes et avons plutôt cherché à accélérer la résolution du sous-problème.

4.3 Heuristique pour la résolution du sous problème

Nous remarquons que le sous-problème a une formulation en 4.1 qui est totalement séparable pour les variables δ^1 et δ^2 et que pour chacune de ces variables nous avons une formulation qui correspond au PL d'un problème de sac-a-dos. On le résout alors de la manière suivante pour δ^1 en s'appuyant sur le fait que T est entier:

- ranger $(\hat{t}_i + \hat{t}_j)_{ij}$ par ordre décroissant.
- Avec cet ordre sur les (i,j), fixer les T premiers $\delta^1_{ij} = 1$ et les autres à 0.

On procède de la même manière pour δ^2, en les rangeant par $(\hat{t}_i\hat{t}_j)_{ij}$ décroissant et en prenant les $\lfloor \frac{T^2}{2} \rfloor$ premiers que l'on fixe à 2 et le suivant à 1 ou 0 selon la parité de T^2.

Cet algorithme exacte est polynomial et plus efficace que l'utilisation de CPLEX por résoudre le PL. On passe de 2% du temps d'exécution dédié à la résolution du sous-problème à 0.1% . Seulement comme c'est la résolution du problème maître qui prend le plus de temps, on ne voit pas de nettes améliorations sur la performance des algorithmes.

5 Résultats

5.1 Performances

Pour les résultats, nous avons comparé la meilleure heuristique (swap 2opt between routes), les plans coupants et le branch-and-cut en utilisant l'algorithme présenté pour la résolution du sous problème ainsi que la dualisation et la dualisation ws (warm start) initialisée avec une tournée fournie par l'heuristique précédente. Nous avons cherché à résoudre les instances en 600s.

Pour la dualisation ws, elle n'accorde à l'algorithme de dualisation que le temps restant des 600s non utilisée par l'heuristique.

On voit sur la figure 4 que l'algorithme de dualisation résout le plus d'instances mais c'est en réalité comparable avec dualisation ws, puisque ce dernier trouve la solution mais ne parvient pas à prouver son optimalité dans le laps de temps restant. Cet algorithme retourne toujours les valeurs les plus faibles mais pas nécessairement les meilleurs gaps. Ces derniers sont calculés avec la borne inférieure fournie par l'algorithme et non avec la meilleure connue.

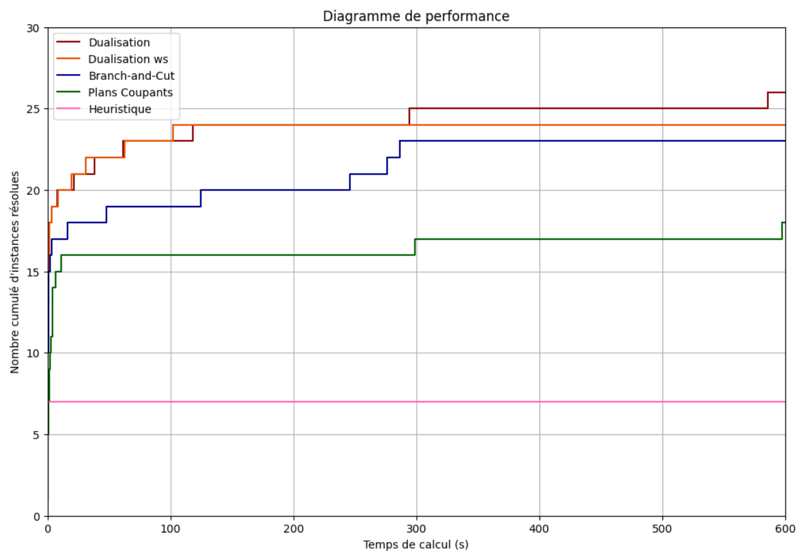

Figure 4: Diagramme de performances

La première distinction à faire sur la 5 est entre la courbre heuristique eucl gap et heuristique eucl gap dualisation. La première est le gap donné par l'heuristique et la deuxième est le gap entre le solution de l'heuristique et la borne fournie par l'aglorithme de dualisation. On remarque donc que sur les instances euclidiennes l'heuristique fournit les meilleurs résultats à égalité avec dualisation ws (qui l'utilise). La différence entre les deux provient de la borne inférieure utilisée. En revanche pour les instances non euclidiennes l'heuristique ne fournit plus les meilleurs résultats et l'algorithme de dualisation initialisé par l'heuristique fournit les meilleurs résultats.

Ces résultats ne sont pas étonnants puisque l'heuristique se comporte différement si l'instance est euclidienne donc on pouvait s'attendre à ce que son utilisation pour l'algorithme de dualisation soit efficace.

On voit également que l'algorithme des plans coupants et de branch-and-cut ne sont pas aussi performants. On pouvait s'y attendre pour les plans coupants. Concernant le branch-and-cut peut-être que l'utilisation d'un unique thread imposé par l'utilisation de callback ralentit trop le solveur.

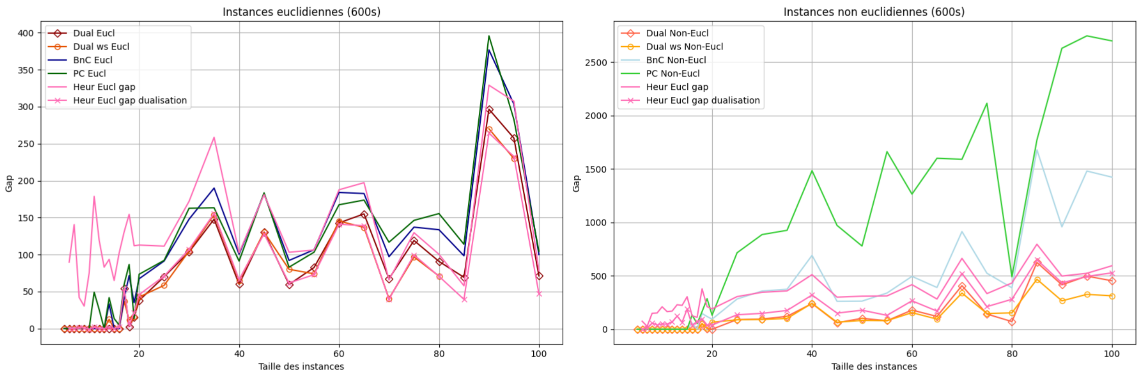

Figure 5: Comparaison des gaps.

Sur les figures **??**, on compare les performances des différentes méthodes exactes sur les instances euclidiennes et non-euclidiennes. On remarque qu'en général les gaps sont plus important pour les instances non-euclidiennes. Etant donné la présence de symétries pour les instances euclidiennes, celles-ci sont traitées par CPLEX et améliore la convergence des méthodes. En revanche lorsque les instances sont résolues, elles le sont plus rapidement pour les instances non-euclidiennes. Peut-être que le temps de traiter les symétries ralentit l'algorithme ou alors les valeurs des paramètres étant plus élevées pour les instance euclidiennes, il y a davantage de branchement à réaliser.

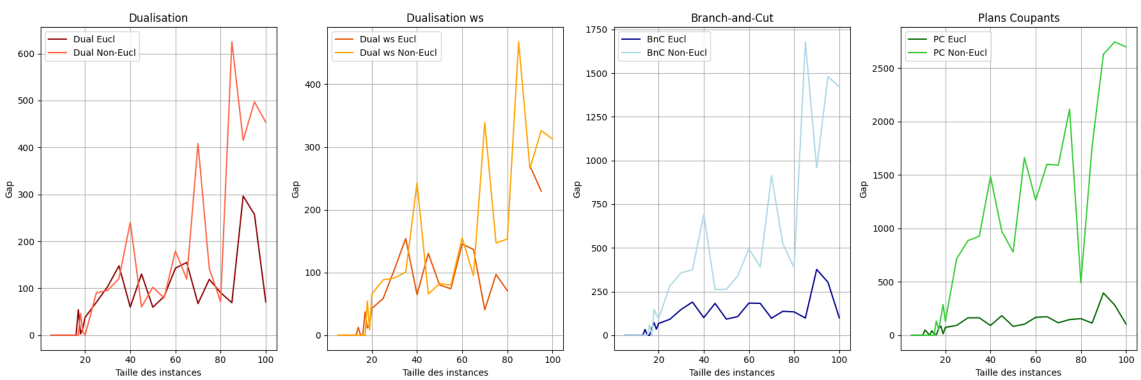

Figure 6: Comparaison des gaps méthodes exactes sur instances euclidiennes et non-euclidiennes.

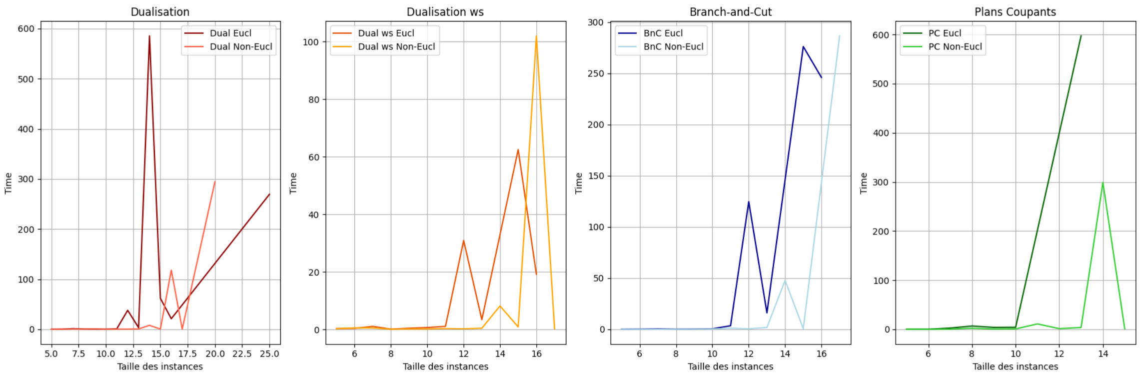

Figure 7: Comparaison des temps des méthodes exactes sur instances euclidiennes et non-euclidiennes.

5.2 Tableaux

Résultats des algorithmes (temps limité à 600s). Le prix de la robustesse (PR) est caclulé avec les résultats de l'algorithme de dualisation. Il est exact si l'instance est résolue et majoré par la valeur indiqué si non (val(dualisation) - borne inf(static))/borne inf(static). ‘

Instance	PR	Plans coupants		Branch-and-cut		Dualisation		Heuristique	
		Time	Gap	Time	Gap	Time	Gap	Time	Gap
n_5 instance	8.8%	0.1s	0%	0.01s	0%	0.01s	0%		
n_5_euclidean-false	17.4%	0.2s	0%	0.08s	0%	0.03s	0%		
n_5_euclidean-true	10.4%	0.2s	0%	0.03s	0%	0.03s	0%		
n_6_euclidean-false	29.5%	0.1s	0%	0.11s	0%	0.19s	0%		
n_6_euclidean-true	2.2%	0.2s	0%	0.19s	0%	0.2s	0%		
n_7_euclidean-false	11.3%	0.3s	0%	0.002s	0%	0.36s	0%		
n_7_euclidean-true	5%	2.5s	0%	0.4s	0%	1.27s	0%		
n_8_euclidean-false	40%	2.2	0%	0.07s	0%	0.37s	0%		
n_8_euclidean-true	5.7%	6.7s	0%	0.1s	0%	0.5s	0%		
n_9_euclidean-false	15.3%	0.5s	0%	0.2s	0%	0.73s	0%		
n_9_euclidean-true	18.5%	3.8s	0%	0.27s	0%	0.37s	0%		
n_10_euclidean-false	22.6%	0.8s	0%	0.2s	0%	0.1s	0%		
n_10_euclidean-true	12.2%	4.1s	0%	0.42s	0%	0.14s	0%		
n_11_euclidean-false	34.2%	10.9s	0%	1.02s	0%	0.31s	0%		
n_11_euclidean-true	14.8%	600s	49%	3.43s	0%	0.91s	0%		
n_12_euclidean-false	15%	1.3s	0%	0.45s	0%	0.29s	0%		
n_12_euclidean-true	11.8%	600s	26%	125s	0%	38s	0%		
n_13_euclidean-false	20.9%	3.7s	0%	1.73s	0%	0.67s	0%		
n_13_euclidean-true	19.7%	597s	0%	16s	0%	3.3s	0%		
n_14_euclidean-false	36.9%	299s	0%	48s	0%	7.9s	0%		
n_14_euclidean-true	37.2%	600s	42%	600s	33%	586s	0%		
n_15_euclidean-false	14.7%	1.2s	0%	0.51s	0%	0.3s	0%		
n_15_euclidean-true	4.8%	600s	13%	276s	0%	62s	0%		
n_16_euclidean-false	134%	600s	133%	600s	58%	118s	0%		
n_16_euclidean-true	8.2%	600s	4.8%	246s	0%	21s	0%		
n_17_euclidean-false	199%	600s	52%	287s	0%	0.34s	0%		
n_17_euclidean-true	71.6%	600s	60%	600s	41%	600s	54%		
n_18_euclidean-false	216%	600s	166%	600s	147%	603s	46%		
n_18_euclidean-true	20.9%	600s	87%	600s	71%	600s	2.75%		
n_19_euclidean-false	327%	600s	286%	600s	119%	600s	6.77%		
n_19_euclidean-true	32.9%	600s	14%	600s	35%	600s	15%		
n_20_euclidean-false	232%	600s	131%	600s	94%	294s	0%		
n_20_euclidean-true	76.9%	603s	73%	600s	67%	652s	38%		
n_25_euclidean-false	773%	727s	716%	600s	283%	601s	90%		
n_25_euclidean-true	95.5%	600s	92%	600s	91%	269s	70%		
n_30_euclidean-false	875%	600s	886%	600s	357%	639s	95%		
n_30_euclidean-true	163%	600s	163%	600s	148%	600s	103%		
n_35_euclidean-false	1246%	600s	926%	600s	374%	622s	120%		
n_35_euclidean-true	165%	601s	163%	600s	190%	600s	147%		
...	...	...	...	...	...	...	...	...	...

‘

Instance	PR	Plans coupants		Branch-and-cut		Dualisation		Heuristique	
		Time	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**
n_40_euclidean-false	1214%	642s	1483%	600s	691%	600s	240%		
n_40_euclidean-true	92%	600s	91%	600s	100%	600s	60%		
n_45_euclidean-false	699%	601s	971%	600s	261%	601s	60%		
n_45_euclidean-true	185%	602s	183%	600s	182%	600s	130%		
n_50_euclidean-false	614%	600s	779%	600s	263%	600s	102%		
n_50_euclidean-true	83%	601s	83%	600s	92%	606s	60%		
n_55_euclidean-false	1336	601s	1662%	600s	339%	600s	80%		
n_55_euclidean-true	110%	600s	103%	600s	107%	600s	82%		
n_60_euclidean-false	1254%	600s	1264%	600s	495%	601s	179%		
n_60_euclidean-true	164%	600s	167%	600s	184%	600s	142%		
n_65_euclidean-false	1305%	600s	1599%	600s	390%	600s	119%		
n_65_euclidean-true	183%	600s	174%	600s	183%	605s	155%		
n_70_euclidean-false	1748%	603s	1590%	600s	914%	601s	408%		
n_70_euclidean-true	126%	601s	117%	600s	97%	621s	64%		
n_75_euclidean-false	1533%	601s	2114%	600s	523%	600s	142%		
n_75_euclidean-true	138%	611s	146%	600s	137%	601s	119%		
n_80_euclidean-false	398%	601s	491%	600s	388%	606s	71%		
n_80_euclidean-true	154%	604s	155%	600s	134%	602s	90%		
n_85_euclidean-false	2014%	602s	1771%	600s	1679%	602s	624%		
n_85_euclidean-true	131%	630s	114%	600s	98%	604s	69%		
n_90_euclidean-false	2400%	602s	2629%	600s	958%	600s	415%		
n_90_euclidean-true	395%	602s	396%	600s	377%	611s	296%		
n_95_euclidean-false	2578%	602s	2745%	600s	1480%	600s	497%		
n_95_euclidean-true	292%	602s	283%	600s	303%	600s	257%		
n_100_euclidean-false	2458%	601s	2697%	600s	1422%	600s	453%		
n_100_euclidean-true	81%	601s	104%	600s	100%	600s	71%		

5.3 Solutions

Instance	Solution
instance_n5	(1,2),(1,3,5,4)
n_5-_euclidean-false	(1,4,3,2),(1,5)
n_5_euclidean-true	(1,3,4),(1,5,2)
n_6-_euclidean-false	(1,2),(1,4,3),(1,6,5)
n_6_euclidean-true	(1,4,3),(1,5,6,2)
n_7-_euclidean-false	(1,5,7,4),(1,6,2,3)
n_7_euclidean-true	(1,5,2),(1,6,3,4,7)
n_8-_euclidean-false	(1,3,5),(1,6,4,8),(1,7,2)
n_8_euclidean-true	(1,3),(1,5,2),(1,7,4,6,8)
n_9-_euclidean-false	(1,4,2),(1,5,8,9),(1,7,3,6)
n_9_euclidean-true	(1,3,7),(1,4,8,6,2),(1,5),(1,9)
n_10-_euclidean-false	(1,2,8,3),(1,6,7,5),(1,10,4,9)
n_10_euclidean-true	(1,2,6,5,7),(1,3,4,8),(1,10,9)
n_11-_euclidean-false	(1,3,6,8,9),(1,4,2,7),(1,5,10,11)
n_11_euclidean-true	(1,2,4),(1,3,7,6,10),(1,5,8,11),(1,9)
n_12-_euclidean-false	(1,2,11),(1,3,10,5,6),(1,4,9,8,7,12)
n_12_euclidean-true	(1,6,12,9),(1,7,4,10,11),(1,8,5,3,2)
n_13-_euclidean-false	(1,6,11,13,5,7),(1,9,8,4,3),(1,10,2,12)
n_13_euclidean-true	(1,5,2),(1,11,3),(1,12,8,4),(1,13,7,10,9,6)
n_14-_euclidean-false	(1,5,3),(1,11,6,8),(1,12,4,9,2,7),(1,13,10,14)
n_14_euclidean-true	(1,4,7,13),(1,6,5),(1,8,10,2),(1,14,3,12,9,11)
n_15-_euclidean-false	(1,5,6,13,8,7,4),(1,9,12,3,2,10),(1,15,14,11)
n_15_euclidean-true	(1,3),(1,12,8,6,13),(1,14,4,10,2),(1,15,7,9,11,5)
n_16-_euclidean-false	(1,4,10,9),(1,5,15,6),(1,7,2,16,12),(1,13,11,3,14,8)
n_16_euclidean-true	(1,6,5),(1,8,15,2),(1,11,9,4),(1,12,7,3),(1,14,10),(1,16,13)
n_17-_euclidean-false	(1,2,15),(1,6,16,14),(1,7,9,17,3,5),(1,8,12,4),(1,11),(1,13,10)
n_20-_euclidean-false	(1,2,9,19),(1,10,13,14,15,8,4,3,5),(1,17,12,11,6,7),(1,20,18,16)