PROJ - Rapport d'implémentation

Ronan Le Prat, Aurélien Arnoux

Février 2026

Contents

1 Introduction

Ce rapport présente les choix d'implémentations effectués dans le cadre d'un projet d'optimisation robuste du MPRO.

Le problème que l'on cherche à résoudre est un problème de clustering robuste, dont une description formelle (ainsi que de la version statique) est accessible ici : https://github.com/Renelle29/MPRO-PROJ/blob/main/PROJ.pdf

Une des difficultés principale du problème, est que la valeur des clusters croit de manière quadratique avec le nombre d'éléments qu'ils contiennent. Il s'agit donc de trouver un compromis entre regrouper les éléments proches les uns des autres, et créer des clusters de tailles similaires.

Dans l'ensemble du rapport, on évoquera les instances par la valeur de leurs paramètres. `I(n,K)` désignera une instance à `n` sommets, pour laquelle on s'autorise `K` clusters. On fera l'hypothèse que la 'difficulté' des instances est croissante en `n` et `K`, et donc qu'une méthode qui résout `I(n,k)` résout aussi à priori `I(n',k')` pour `n' <= n` et `K' <= K` (dès lors que `K` est petit devant `n`).

2 Implémentations

Les algorithmes ont été implémentés en Julia, et utilisent le solver Gurobi.

Le code, les données, et les instructions d'utilisations sont accessibles via le lien suivant : https://github.com/Renelle29/MPRO-PROJ

2.1 Version statique [VS]

Avant de s'intéresser au cas robuste, on s'est intéressé au cas statique qui présente de nombreux intérêts.

- Il est plus simple à résoudre,
- Toute borne inférieure du problème statique est une borne inférieure du problème robuste, car une solution réalisable du problème robuste est aussi une solution réalisable du problème statique, de valeur plus faible pour le problème statique que pour le problème robuste.
- Il est structurellement proche du problème robuste. Ainsi les méthodes présentées ci-dessous afin d'améliorer les performances de résolution, s'appliqueront de manière similaire pour le cas statique et robuste.

L'algorithme statique résout à l'optimum en moins de 2 minutes les instances jusqu'à `I(40,3)`.

2.2 Version robuste [VR]

Dans la suite du rapport on compare les performances de trois implémentations permettant de résoudre le problème robuste.

2.2.1 Plans Coupants - Sans callbacks [VR-P]

La première méthode est une méthode itérative par plans coupants. Elle fonctionne de la manière suivante.

1. On résout un problème (à variables binaires) principal.
2. On résout des sous-problèmes pour s'assurer de la validité de notre solution. On vérifie qu'on a ni surestimé l'objectif, ni trop rempli un cluster.
    - Si aucune contrainte n'est violée, on a atteint l'optimum.
    - Sinon on rajoute autant de coupes que de contraintes violées, et on reprend au point 1.

En l'état, c'est une méthode très peu efficace, puisqu'on doit possiblement résoudre le problème principal un nombre important de fois (qui est au moins aussi difficile que le problème statique).

Les sous-problèmes sont faciles à résoudre. On a implémenté pour chacun une méthode de résolution via un programme linéaire, et une méthode de calcul explicite en $O(n^2)$, sans qu'utiliser l'une ou l'autre des méthodes n'impacte significativement le temps de calcul total (le temps de résolution des sous-problèmes étant négligeable devant celui du problème principal).

2.2.2 Plans Coupants - Avec callbacks [VR-P-C]

Pour pallier aux problèmes décrits précédemment, notamment pour éviter de résoudre le problème principal à l'optimum un grand nombre de fois, on a également implémenté la méthode de plans coupants en utilisant les possibilités de callbacks du solver Gurobi.

En particulier, via cette méthode, on a fait en sorte de résoudre les sous-problèmes pour chaque solution entière renvoyée par le solver au cours de son exploration de l'arbre des solutions ; puis d'intégrer des coupes en ce nœud dans le cas où la solution entière renvoyée violerait une ou des contraintes robustes.

On réduit via cette méthode grandement le nombre de calculs nécessaires. La méthode obtenue est bien plus efficace que celle via plans coupants sans callbacks. Elle permet en outre de spécifier facilement une durée maximum de résolution.

2.2.3 Dualisation [VR-D]

La dernière méthode exacte implémentée est une méthode via dualisation, qui consiste à dualiser les contraintes robustes, afin d'obtenir une formulation MILP unique résolvant le problème robuste. La nouvelle formulation contient à peu près deux fois plus de variables et de contraintes que le problème initial (mais on n'augmente pas le nombre de variables entières).

En terme de performances, la méthode par dualisation s'est systématiquement révélée être la plus performantes des méthodes pour le problème robuste.

3 Amélioration des performances

Afin d'améliorer les performances des algorithmes naïf implémentés, nous avons cherché à explorer différentes pistes décrites ci-dessous.

3.1 Bonne solution initiale heuristique

Sur les instances difficiles, le solveur a des difficultés a déterminer une bonne solution entière initiale (voir même à trouver ne serait-ce qu'une solution entière). Nous avons donc implémenté une heuristique gloutonne qui fourni une première solution entière admissible au solveur (ce qui permet à terme de couper plus de branches et d'accélérer l'exploration de l'arbre).

L'heuristique gloutonne implémentée va insérer à une itération i donnée, l'élément dont le **regret** r_v^i est le plus important. Intuitivement le **regret** représente combien cela nous coûterait d'insérer l'élément v à une itération ultérieure. On le définit comme la différence des **coûts d'insertion** pour les deux clusters dont les coûts d'insertion $c_{v,k}^i$ sont les plus faibles.

On définit formellement les coûts d'insertion de la manière suivante :

- Dans le cas statique :

$$c_{v,k}^i = \begin{cases} \sum_{u \in K_k^i} d_{v,u} & \text{si } \sum_{u \in K_k^i} w_u + w_v \leq B, \\ +\infty & \text{sinon} \end{cases}$$

- Dans le cas robuste :

$$c_{v,k}^i = \begin{cases} \sum_{u \in K_k^i} d_{v,u} + max_{u \in K_k^i} 3(\hat{l}_u + \hat{l}_v) & \text{si } \sum_{u \in K_k^i} w_u(1+W_u) + w_v(1+W_v) \leq B, \\ +\infty & \text{sinon} \end{cases}$$

L'heuristique est très dépendante de l'ordre dans lequel on va lui présenter les éléments (car pour les $K-1$ premières itérations, le regret de tous les éléments est nul). On va ainsi recalculer l'heuristique pour un grand nombre d'ordres aléatoires des éléments de l'ordre de `MaxIter=10000` itérations, et garder la solution finalement obtenue.

L'algorithme complet à une complexité totale de l'ordre de `O(MaxIter.K.n²)`, qui reste raisonnable pour toutes les instances proposées (exécution en moins de 10 secondes sur l'ensemble des instances).

En pratique l'heuristique semble très performante, et résout à l'optimum ou presque la grande partie des instances pour lesquelles nous avons réussi à démontrer l'optimalité.

Instance	Solve Time Heuristic	Nodes Heuristic	Solve Time No Heuristic	Nodes No Heuristic
10_ulysses_3.tsp	0.07	159	0.08	203
10_ulysses_6.tsp	0.09	243	0.08	199
10_ulysses_9.tsp	0.01	1	0.03	1
14_burma_3.tsp	0.15	188	0.1	170
14_burma_6.tsp	0.19	422	0.14	370
14_burma_9.tsp	0.43	1741	0.26	1605
22_ulysses_3.tsp	2.23	1400	1.57	2301
22_ulysses_6.tsp	302.86	452095	365.14	515596
26_eil_3.tsp	8.31	1819	9.54	2067
30_eil_3.tsp	43.28	6568	30.04	3959
34_pr_3.tsp	185.39	55555	328.73	52138
48_att_3.tsp	126.36	2644	131.71	2085

Table 1: Comparaison des performances de résolution du problème robuste avec et sans heuristique

3.2 Amélioration de la borne obtenue par relaxation linéaire

Un autre problème que nous avons rencontré était lié à l'obtention d'une bonne borne inférieure. En effet, le problème tel qu'il était posé initialement, admettait pour sa version relâchée une solution optimale de valeur 0 dans laquelle $y_{v,k} = \frac{1}{K}$ et $x_{u,v} = 0$.

Nous avons donc cherché à ajouter des inégalités valides qui permettent d'aboutir à une meilleure valeur de relaxation. On a ainsi ajouté une inégalité valide qui découle d'une borne inférieure m sur la somme des $x_{u,v}$:

$$\sum_{1 \leq u < v \leq n} x_{u,v} \geq m$$

Avec :

$$m = r\frac{(q+1)(q+2)}{2} + (K-r)\frac{q(q+1)}{2}$$

et q et r le quotient et reste dans la division euclidienne de n par K : $n = qK + r$. Ainsi m est le nombre minimum de couples d'éléments appartenant aux mêmes clusters, dans le cas où tous les clusters sont de même taille (plus ou moins 1).

En faisant ainsi on améliore sensiblement les bornes inférieures obtenues par relaxation. La relaxation obtenue à la racine à pour valeur la somme des m plus petites distances entre deux éléments. Cela reste malgré tout une borne assez faible, dans le sens où elle ne prend pas en compte la structure des clusters.

Instance	Solve Time No Symetries	Nodes No Symetries	Solve Time Symetries	Nodes Symetries
10_ulysses_3.tsp	0.07	159	0.06	235
10_ulysses_6.tsp	0.09	243	0.09	219
10_ulysses_9.tsp	0.01	1	0.0	0
14_burma_3.tsp	0.15	188	0.18	107
14_burma_6.tsp	0.19	422	0.15	492
14_burma_9.tsp	0.43	1741	0.37	2311
22_ulysses_3.tsp	2.23	1400	3.55	2137
22_ulysses_6.tsp	302.86	452095	400.02	263348
26_eil_3.tsp	8.31	1819	323.92	64913
30_eil_3.tsp	43.28	6568	400.01	43931
34_pr_3.tsp	185.39	55555	400.06	7528
48_att_3.tsp	126.36	2644	400.01	2181

Table 2: Comparaison des performances de résolution du problème robuste avec et sans coupe

3.3 Suppression de symétries

Dans le problème proposé, les clusters sont indifférents, et leur ordre importe peu. Ainsi pour une solution donnée, il existe $K!$ solutions symétriques de même valeur pour lesquelles l'ordre des clusters à simplement été permuté.

On a ajouté les contraintes suivantes qui permettent de retirer certaines symétries :

$$y_{i,k} = 0, \forall k > i$$

Cela permet de réduire l'exploration de l'arbre des solutions. Certaines symétries demeurent néanmoins, mais les expressions nécessaires pour les retirer deviennent trop compliquées pour que les retirer en vaille la peine.

Instance	Solve Time No Symetries	Nodes No Symetries	Solve Time Symetries	Nodes Symetries
10_ulysses_3.tsp	0.07	159	0.07	123
10_ulysses_6.tsp	0.09	243	0.08	33
10_ulysses_9.tsp	0.01	1	0.02	1
14_burma_3.tsp	0.15	188	0.16	200
14_burma_6.tsp	0.19	422	0.19	88
14_burma_9.tsp	0.43	1741	0.17	174
22_ulysses_3.tsp	2.23	1400	1.23	1623
22_ulysses_6.tsp	302.86	452095	12.8	38450
26_eil_3.tsp	8.31	1819	4.54	827
30_eil_3.tsp	43.28	6568	24.38	3532
34_pr_3.tsp	185.39	55555	191.25	8769
48_att_3.tsp	126.36	2644	84.15	2449

Table 3: Comparaison des performances de résolution du problème robuste avec et sans symétries

4 Comparaison des méthodes

4.1 Comparaison aggrégée des performances

Dans la suite, on comparera essentiellement les performances obtenues par la méthode par dualisation, et la méthode par plans coupants avec callbacks.

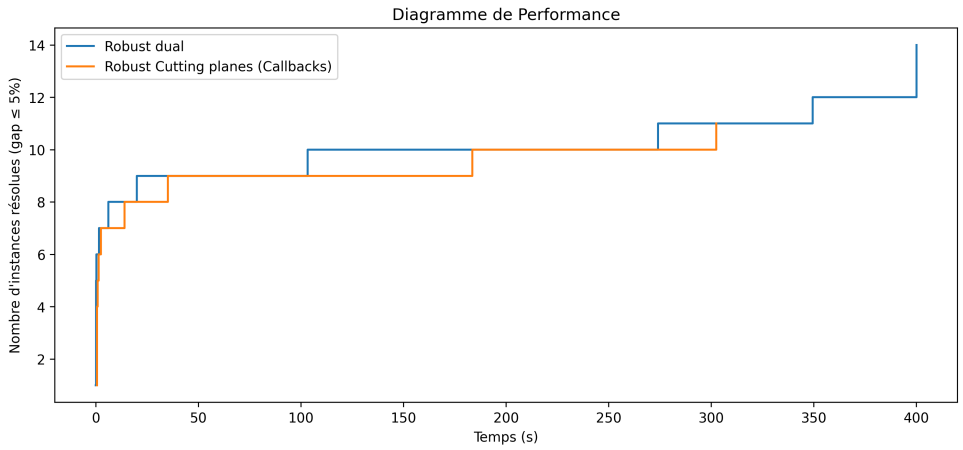

Figure 1: Nombre d'instances résolues à l'optimum en moins de 400 secondes par les méthodes duales et cutting planes avec callbacks.

De manière générale, la méthode par dualisation s'est révélée plus efficace que la méthode par plans coupants. Elle résout plus rapidement à l'optimum les instances résolues, et permet d'obtenir de meilleures bornes inférieures quand l'optimum n'est pas atteint.

Métrique	Dualisation	Plans Coupants
Erreur Moyenne	11%	15%
Ecart-type à la moyenne	14%	17%
Erreur Maximale	42%	44%

Table 4: Résumé aggrégé des méthodes par dualisation et plans coupants.

4.2 Tableau de performances

Le tableau donne les temps de calculs des résolutions par dualisation robuste, par plans coupants avec call back et par l'heursitique gloutonne. On donne également pour chaque instance le gap entre la valeur renvoyées par chaque méthode et la meilleure borne inférieure parmi celles obtenues au cours des résolutions robustes et statiques. Enfin PR donne le gap entre la valeur statique et la meilleure valeur robuste obtenue.

Instance	PR	Dual Time	Dual Gap	CP+CB Time	CP+CB Gap	Greedy Time	Greedy Gap
10_ulysses_3.tsp	152.04%	0.08	0.00%	0.59	0.00%	0.00	2.08%
10_ulysses_6.tsp	663.22%	0.08	0.00%	0.61	0.00%	0.00	0.00%
10_ulysses_9.tsp	4522.09%	0.01	0.00%	0.56	0.00%	0.00	0.00%
14_burma_3.tsp	41.04%	0.11	0.00%	0.65	0.00%	0.00	0.00%
14_burma_6.tsp	137.95%	0.16	0.00%	0.86	0.00%	0.00	0.00%
14_burma_9.tsp	339.43%	0.28	0.00%	1.47	0.00%	0.00	15.59%
22_ulysses_3.tsp	26.19%	1.61	0.00%	2.51	0.00%	0.02	0.12%
22_ulysses_6.tsp	28.23%	349.45	0.00%	400.01	8.90%	0.00	8.90%
22_ulysses_9.tsp	147.83%	400.00	41.52%	400.01	47.73%	0.00	47.73%
26_eil_3.tsp	17.85%	6.06	0.00%	14.02	0.00%	0.00	0.00%
26_eil_6.tsp	25.48%	400.06	14.54%	400.01	23.75%	0.00	23.75%
26_eil_9.tsp	38.47%	400.02	31.30%	400.03	44.78%	0.00	48.13%
30_eil_3.tsp	11.63%	20.02	0.00%	35.08	0.00%	0.00	0.00%
30_eil_6.tsp	22.82%	400.00	24.53%	400.02	29.16%	0.00	28.60%
30_eil_9.tsp	34.73%	20.01	30.74%	400.01	34.89%	0.00	37.49%
34_pr_3.tsp	24.32%	274.05	0.00%	302.54	0.01%	0.00	0.01%
34_pr_6.tsp	30.41%	400.01	34.41%	400.02	41.50%	0.00	39.44%
34_pr_9.tsp	64.63%	20.01	49.23%	400.01	51.06%	0.00	49.67%
38_rat_3.tsp	24.13%	400.01	1.83%	400.02	11.72%	0.00	12.91%
38_rat_6.tsp	50.92%	20.01	38.28%	400.03	38.89%	0.00	38.28%
38_rat_9.tsp	56.80%	20.03	48.97%	400.04	48.16%	0.00	50.00%
40_eil_3.tsp	16.23%	400.01	4.44%	400.01	11.13%	0.00	11.13%
40_eil_6.tsp	47.89%	20.01	33.85%	400.02	33.83%	0.02	32.70%
40_eil_9.tsp	63.30%	20.11	44.03%	400.35	42.47%	0.00	43.42%
44_lin_3.tsp	20.44%	400.01	14.95%	400.02	22.24%	0.00	22.37%
44_lin_6.tsp	67.76%	20.06	45.13%	400.02	45.84%	0.00	44.86%
44_lin_9.tsp	59.49%	20.01	49.26%	400.05	49.15%	0.00	49.81%
48_att_3.tsp	9.03%	103.36	0.00%	183.60	0.00%	0.00	2.96%
48_att_6.tsp	38.75%	400.10	33.18%	400.02	42.22%	0.00	38.57%
48_att_9.tsp	67.36%	20.02	47.45%	400.03	47.97%	0.00	47.37%
52_berlin_3.tsp	11.23%	400.03	18.38%	400.02	21.38%	0.00	21.85%
52_berlin_6.tsp	37.35%	20.03	49.44%	400.03	48.47%	0.00	49.20%
52_berlin_9.tsp	83.97%	20.02	59.73%	400.02	57.74%	0.00	59.41%
70_st_3.tsp	8.67%	20.02	16.36%	400.03	16.36%	0.00	16.36%
70_st_6.tsp	29.63%	20.02	29.77%	400.06	29.68%	0.00	29.28%
70_st_9.tsp	55.75%	20.04	36.21%	400.05	36.81%	0.00	37.85%
80_gr_3.tsp	9.62%	20.04	22.06%	400.03	22.06%	0.00	22.06%
80_gr_6.tsp	28.87%	20.05	29.80%	400.24	29.44%	0.00	30.39%
80_gr_9.tsp	63.14%	20.03	42.06%	400.07	41.01%	0.00	41.21%
100_kroA_3.tsp	5.97%	20.07	17.30%	400.02	17.30%	0.00	17.30%
100_kroA_6.tsp	18.06%	20.07	24.67%	400.04	24.55%	0.00	24.55%
100_kroA_9.tsp	42.56%	20.08	32.35%	400.49	33.59%	0.00	33.79%
202_gr_3.tsp	7.37%	20.10	38.28%	400.04	38.20%	0.00	38.44%
202_gr_6.tsp	86.93%	–	–	–	–	–	–
202_gr_9.tsp	21.71%	20.10	55.48%	400.07	55.50%	0.00	55.82%
318_lin_3.tsp	39.66%	–	–	–	–	–	–
318_lin_6.tsp	7.56%	20.10	25.79%	400.10	25.22%	0.02	25.40%
318_lin_9.tsp	18.45%	20.23	31.52%	400.47	31.71%	0.04	31.25%
400_rd_3.tsp	0.69%	20.18	21.82%	400.12	21.82%	0.05	21.82%
400_rd_6.tsp	3.11%	20.30	100.00%	–	–	0.03	100.00%
400_rd_9.tsp	6.92%	20.18	100.00%	–	–	0.03	100.00%
532_att_3.tsp	11.91%	20.14	41.10%	400.20	40.95%	0.08	40.93%
532_att_6.tsp	14.86%	20.23	100.00%	–	–	0.08	100.00%
532_att_9.tsp	23.57%	20.41	100.00%	–	–	0.09	100.00%

Instance	Valeur Robuste
10_ulysses_3.tsp	137.00
10_ulysses_6.tsp	55.12
10_ulysses_9.tsp	33.29
14_burma_3.tsp	93.39
14_burma_6.tsp	42.74
14_burma_9.tsp	21.28
22_ulysses_3.tsp	358.64
22_ulysses_6.tsp	108.45
22_ulysses_9.tsp	62.24
26_eil_3.tsp	2297.63
26_eil_6.tsp	826.63
26_eil_9.tsp	588.48
30_eil_3.tsp	3021.11
30_eil_6.tsp	1129.81
30_eil_9.tsp	661.12
34_pr_3.tsp	646561.44
34_pr_6.tsp	249079.44
34_pr_9.tsp	165011.09
38_rat_3.tsp	7850.15
38_rat_6.tsp	3770.83
38_rat_9.tsp	2306.84
40_eil_3.tsp	6062.27
40_eil_6.tsp	2776.47
40_eil_9.tsp	1743.76
44_lin_3.tsp	150720.03
44_lin_6.tsp	71453.97
44_lin_9.tsp	37213.62
48_att_3.tsp	647503.88
48_att_6.tsp	281650.00
48_att_9.tsp	168925.56
52_berlin_3.tsp	168958.05
52_berlin_6.tsp	71768.45
52_berlin_9.tsp	48707.38
70_st_3.tsp	25540.46
70_st_6.tsp	10425.61
70_st_9.tsp	6344.15
80_gr_3.tsp	21413.54
80_gr_6.tsp	8241.13
80_gr_9.tsp	5316.51
100_kroA_3.tsp	1541208.00
100_kroA_6.tsp	565932.00
100_kroA_9.tsp	345041.51
202_gr_3.tsp	70166.11
202_gr_9.tsp	16154.27
318_lin_6.tsp	6376595.38
318_lin_9.tsp	3551717.15
400_rd_3.tsp	8467238.00
400_rd_6.tsp	2938088.00
400_rd_9.tsp	1614582.00
532_att_3.tsp	87763560.00
532_att_6.tsp	29597690.00
532_att_9.tsp	17379930.00

Table 5: Valeur de la meilleure solution robuste