ECMA
Résultats expérimentaux

Barreaux Alexis - Le Bozec-Chiffoleau Sulian

dépôt github du projet:
https://github.com/alexisbarreaux/projet_ECMA

Barreaux Alexis - Le Bozec-Chiffoleau Sulian

1)Modèles utilisés

a) Statique

Le modèle statique est présent dans le fichier staticModel.jl. Il contient un modèle équivalent à celui proposé dans notre partie théorique, qui correspond à la correction sans les inégalités triangulaires redondantes.

b) Dual

Le modèle dual est présent dans le fichier dualModel.jl.

c) Plans coupants

Le modèle des plans coupants est présent dans cutsModel.jl. Les sous problèmes se trouvent dans les fonctions firstSubProblem et secondKSubProblem respectivements. On utilise une boucle while pour faire tourner le modèle et itérativement ajouter les nouvelles contraintes après la résolution si certaines sont violées.

d) Branch and cut

Le modèle du branch and cut se trouve dans brandAndCutModel.jl. Du fait de la gestion différente de la limite de temps entre les plans coupants et les autres modèles, les sous-problèmes sont ici définis dans des fonctions très proches mais avec un nom différent (branchFirstSubProblem et branchSecondKSubProblem). Le callback est présent dans la fonction avec le modèle principal et se nomme myCallback.

e) Heuristiques

L'heuristique a pour but d'obtenir rapidement des solutions de la meilleure qualité possible, mais sans garantie d'optimalité. Elle se trouve dans le fichier heuristic.jl. Afin d'évaluer la qualité des solutions renvoyées par l'heuristique nous avons besoin de vérifier leur faisabilité et leur score dans le cas robuste, ce qui nécessite un traitement supplémentaire une fois la solution obtenue.

i) Faisabilité et score dans le cas robuste

Afin de vérifier la faisabilité d'une solution dans le cas robuste nous avons implémenté l'algorithme suivant :

check_feasibility:
Entrée : K listes de sommets (où K est le nombre de clusters pour une instance donnée).
Sortie : Un booléen : VRAI si la solution est faisable et FAUX sinon.
Algorithme :
Pour chaque cluster k faire :

- Initialiser le poids robuste $P_{robuste} = 0$ et l'erreur totale $E = 0$

- Trier les sommets contenus dans le cluster par ordre de poids (w_v) décroissants.
- Pour chaque sommet v pris dans l'ordre faire
  - $e = max(W_v, W - E)$
  - $E \mathrel{+}= e$
  - $P_{robuste} \mathrel{+}= w_v \times (1 + e)$
  - Si $P_{robuste} > B$ alors renvoyer FAUX, sinon continuer

Renvoyer VRAI

Cet algorithme a une complexité dans le pire des cas en $O(n \times log(n))$, où n est le nombre de sommets, il est donc polynomial.

Ensuite nous avons implémenté l'algorithme suivant afin de calculer le score de notre solution dans le cas robuste :

compute_worst_case:
Entrée : K listes de sommets (où K est le nombre de clusters pour une instance donnée).
Sortie : Un float : le score robuste de la solution.
Algorithme :
Initialiser le coût statique $C_{statique} = 0$
Initialiser le coût robuste $C_{robuste} = 0$
Initialiser l'ensemble des distances supplémentaires $D_{sup} = \{\}$
Pour chaque couple de sommets d'un même cluster (i, j) : $i < j$ faire :

- $C_{statique} \mathrel{+}= distance[i][j]$, où $distance$ est la matrice des distances statiques du problème
- Ajouter l'élément $l_h[i] + l_h[j]$ à D_{sup}

Trier les éléments de D_{sup} par valeurs décroissantes
Initialiser la somme des erreurs $E = 0$
Pour chaque élément d de D_{sup} pris dans l'ordre, faire

- $e = max(3, L - E)$
- $E \mathrel{+}= e$
- $C_{robuste} \mathrel{+}= e \times d$

Renvoyer $C_{statique} + C_{robuste}$

Cet algorithme a une complexité dans le pire des cas en $O(n^2 \times log(n))$, où n est le nombre de sommets, il est donc polynomial.

ii) Obtention d'une solution réalisable

On peut remarquer que notre problème s'intéresse à deux grandeurs dont les incertitudes sont indépendantes, en effet l'incertitude sur les distances n'a aucune relation avec l'incertitude sur les poids. Ainsi pour la création d'une heuristique on peut se focaliser sur l'une ou l'autre des deux grandeurs. En revanche, le critère des poids est une contrainte

et doit à tout prix être vérifiée. On a ainsi implémenté une heuristique ayant pour seul but de construire une solution qui soit réalisable, sans prendre en compte les distances.

Cette heuristique se base sur un algorithme de liste, inspiré des algorithmes utilisés en ordonnancement de tâches sur des machines.

construct_feasible_solution:
Entrée : Un nombre de clusters K, une liste ordonnée de sommets $L_{sommets}$
Sortie : K listes disjointes de sommets, dont l'union contient tous les sommets de $L_{sommets}$
Algorithme :
Pour chaque sommet v de $L_{sommets}$ pris dans l'ordre, faire

- Pour chaque cluster k faire
    - calculer le poids robuste de $L_k \cup \{v\}$, où L_k est la liste des sommets actuellements présents dans le cluster k
- Ajouter v au cluster k_{min} dont le poids robuste ainsi calculé est le plus faible

Renvoyer la liste des sommets présents dans chaque cluster

Cet algorithme a une complexité dans le pire des cas en $O(n^2 \times log(n))$, où n est le nombre de sommets, il est donc polynomial.

Cette heuristique a pour but de minimiser le maximum des poids robustes de chaque cluster, avec pour objectif d'avoir ce maximum en dessous de B. Cette condition n'est toutefois pas certaine d'être respectée puisqu'il s'agit d'un algorithme sans garantie d'optimalité.

C'est pourquoi nous avons dû lancer cet algorithme avec plusieurs ordres différents afin d'obtenir une solution réalisable pour le plus d'instances possible :

1) Sommets ordonnés par w_v décroissants
2) Sommets ordonnés par w_v décroissants puis par W_v décroissants
3) Sommets ordonnés par w_v décroissants puis par W_v croissants

Avec ces trois ordres différents, seule une instance ne nous permet pas d'obtenir de solution réalisable (52_berlin_9.tsp).

En pratique, on teste les 3 ordres pour chacune des instances et on garde la solution réalisable avec le plus petit score calculé par **compute_worst_case**.

iii) Amélioration de la solution réalisable avec une recherche locale

Afin d'obtenir une solution avec un meilleur score robuste que la solution réalisable construite par l'heuristique de la section précédente, nous avons implémenté un algorithme de recherche local initialisé avec la solution réalisable.

local_search:
Entrée : K listes de sommets (où K est le nombre de clusters pour une instance donnée)
Sortie : K listes de sommets
Algorithme :
Initialiser le score courant $S_{current} = V$, où V est le score robuste de la solution en entrée
Initialiser le critère d'arrêt $Stop = false$
Tant que $Stop == false$

- Pour chaque couple de sommets de clusters différents $(i, j) : i < j$ dans la solution courante, faire
    - calculer le score robuste de la solution courante en permutant les sommets i et j, si cette permutation permet d'obtenir une solution réalisable
- Sélectionner le couple (i, j) donnant le meilleur score robuste S_{best}
- Si $S_{best} < S_{current}$ alors $S_{current} = S_{best}$ et permuter i et j dans la solution courante
  Sinon, $Stop = true$

Renvoyer la solution courante

Cet algorithme converge vers une solution qu'il ne peut plus améliorer, et sa complexité est en $O(V \times n^4 \times log(n))$, où V est le score robuste de la solution en entrée, il est donc pseudo-polynomial.

En pratique, il est possible d'exploiter le score de la solution courante afin de calculer plus efficacement le nouveau score obtenu en permutant deux sommets, cela permet d'accélérer l'algorithme de recherche locale.

iv) Heuristique finale

L'heuristique que nous utilisons afin d'obtenir nos solutions sur les instances est donc une combinaison des algorithmes **construct_feasible_solution** et **local_search**.

En effet, pour chaque instance, nous lançons **construct_feasible_solution** 3 fois, une fois pour chaque ordre cité dans la section ii), puis nous sélectionnons la meilleure solution réalisable.

S'il existe une telle solution réalisable alors nous l'améliorons en appelant par dessus l'algorithme **local_search**.

2)Choix de méthodologie pour les résultats

a) Lenteurs d'exécution

i) Constat sur la lenteur

Un problème majeur pour étudier ce problème et nos résultats a été que les instances deviennent extrêmement rapidement trop lentes à faire tourner. En effet, atteindre l'optimal semble très vite irréalisable en l'état. Seule une poignée d'instances sont possibles à résoudre en dessous de la minute et le temps de résolution s'envole ensuite : pour 26_eil_6 déjà il faut plus de 12 min et pour 30_eil_6 c'est près de 5 heures qui sont nécessaires. Même les instances avec seulement trois parties dans le partitionnement ne sont pas réalistes à résoudre : 38_rat_3 demande déjà plus d'une heure et demie.

Instance	Optimal	Time	Value
10_ulysses_3.tsp	true	0.16	54.35
10_ulysses_6.tsp	true	0.15	7.22
10_ulysses_9.tsp	true	0.13	0.72
14_burma_3.tsp	true	0.27	66.21
14_burma_6.tsp	true	0.3	17.96
14_burma_9.tsp	true	0.31	4.72
22_ulysses_3.tsp	true	0.85	284.21
22_ulysses_6.tsp	true	9.6	82.63
22_ulysses_9.tsp	true	2.24	26.22
26_eil_3.tsp	true	33.46	1850.42
26_eil_6.tsp	true	827.05	608.72
26_eil_9.tsp	true	1019.75	317.76
30_eil_3.tsp	true	214.6	2693.05
30_eil_6.tsp	true	17989.92	868.39
34_pr_3.tsp	true	236.83	493480.25
38_rat_3.tsp	true	6161.7	6505.45

Extrait des instances résolues de manière optimale en statique (lien vers le fichier)

ii) Réécriture du problème statique

Pour résoudre ce problème de lenteur, nous avons notamment essayé d'utiliser une modélisation différente en statique, avec un objectif quadratique mais binaire pour voir si CPLEX arrive mieux à le résoudre (fichier staticModel_2.jl). A partir de là on aurait pu essayer de l'adapter à une version robuste. Or sur des petites instances déjà il ne semble pas que cela apporte de gain et même que cela ralentisse le processus.

iii) Bornes sur les instances

Nous avons aussi tenté de voir si les instances étaient suffisamment contraintes pour nous donner des informations certaines, comme une borne sur le nombre d'éléments pouvant être mis dans une partie. C'est le sujet du sous-modèle présent dans partWeightsModel.jl qui cherche à minimiser, en ne prenant en compte que la contrainte de sac à dos sur le poids des parties, le nombre d'éléments dans la première partie (choix arbitraire et équivalent à prendre n'importe qu'elle autre). Le nombre d'éléments dans cette partie offre une borne sur le nombre d'éléments qu'on est obligé au minimum de mettre par instance pour ne pas dépasser les contraintes de poids. Hélas, cela offre des résultats assez décevants : seule une petite moitié des instances sont ainsi contraintes (voir le json des résultats). Surtout, cela va logiquement plus contraindre les instances avec peu de parties possibles (les "_3") qui sont déjà les plus simples à résoudre. Par contre c'est un modèle très rapide à résoudre et dont on peut stocker le résultat pour plus tard, ainsi le json où sont stockés les résultats pour la totalité des instances peut se générer en 30. Dans le cas où on a une borne cependant, cela permet de gagner du temps. Ainsi, on peut citer le cas de 34_pr_3 où sans la borne on a une valeur au bout de 5 minutes mais encore un gap de 7%, alors qu'avec la borne en moins de 4 minutes la même valeur est prouvée être optimale. Pour 38_pr_3 aussi par exemple, la valeur réalisable est améliorée de 7% environ et le gap de 10%. Pour 40_eil_3 par contre, la valeur réalisable trouvée est moins bonne, avec un gap plus faible d'1% par contre. Voir le fichier sans la borne et celui avec pour d'autres résultats. Bien qu'intéressante, cette borne ne paraît donc pas suffisante pour être exploitée systématiquement et on ne l'utilise pas dans la suite des résultats.

iv) Contrainte triviale pour les symétries

Ensuite, nous avons testé la contrainte très simple d'imposer qu'un élément soit dans une partie (@constraint(model, y[1,1] ==1)). C'est en effet toujours possible au départ pour un élément et cela casse légèrement les symétries. Bien que cela permettent un gain en théorie, cela paraissait avoir un effet négatif au départ, mais pour le dual nous avons comparé sur les premières instances et finalement ça a semblé positif. D'une part, on arrivait en moyenne à des meilleures solutions à temps équivalent et d'autre part, on avait un gap plus faible à valeur supérieure ou égale. En tout cas, il n'y avait pas de pertes importantes a priori. C'est ce que l'on voit dans les captures ci-dessous:

Instance	Optimal	Time	Value	Gap
22_ulysses_9.tsp	false	301.78	64.97	1
26_eil_3.tsp	true	156.8	2297.63	0
26_eil_6.tsp	false	300.1	1054.41	0.42
26_eil_9.tsp	false	300.11	825.3	1
30_eil_3.tsp	false	300.06	3113.81	0.33
30_eil_6.tsp	false	300.1	1405.74	0.62
30_eil_9.tsp	false	300.48	851.47	1
34_pr_3.tsp	false	300.16	755306.15	0.47

Extrait des instances de la résolution par le dual en 5 minutes maximum sans la contrainte (lien vers le fichier)

Instance	Optimal	Time	Value	Gap
22_ulysses_9.tsp	false	300.58	64.97	0.95
26_eil_3.tsp	true	137.18	2297.63	0
26_eil_6.tsp	false	300.03	1085.79	0.46
26_eil_9.tsp	false	300.03	836.83	0.87
30_eil_3.tsp	false	300.02	3155.29	0.25
30_eil_6.tsp	false	265.99	1376.96	0.58
30_eil_9.tsp	false	300.03	749.98	0.94
34_pr_3.tsp	false	300.01	669328.02	0.25

Extrait des instances de la résolution par le dual en 5 minutes avec la contrainte (lien vers le fichier)

Cette contrainte a donc été utilisée dans la suite.

v) Warm start

Nous avons aussi essayé de brancher notre heuristique sur le dual pour fournir des solutions réalisables au départ, comme nous avons constaté que générer une solution de départ semble assez difficile sur le problème : ainsi il faut quelques minutes sur les plus grosses instances déjà pour générer une première solution réalisable. Pour ce faire nous avons utilisé l'option start de la définition de variable : "@variable(model, x[i=1:n, j=i+1:n], Bin, start=x_heur[i,j])" où x_heur est ici le x associé à la solution réalisable de l'heuristique par exemple. Néanmoins, encore une fois les résultats sont décevants. Dans le fichier dual_warm_comparison.csv, on a comparé l'exécution du dual en 5 minutes avec et sans warm start. On regarde l'écart relatif à la solution sans warm start en valeur et en gap (ie (sans - avec) / sans). Donc plus l'écart est grand et positif, plus la solution avec warm start améliore la résolution et à l'inverse plus elle est négative moins elle aide. On peut voir dans ce fichier que, si ponctuellement le warm start arrive à faire jusqu'à deux fois mieux que sans, sur 100_kroA_9 on a ainsi 98% d'amélioration, on a en moyenne entre +5% et -5% de modifications par rapport à la valeur trouvée par le dual de base. Surtout, il n'y a pas de logique apparente sur quelle instance sera meilleure et quelle instance sera pire. Enfin, sur 400_rd_9 on multiplie par plus de 21 la solution avec le warm start et sur deux instances on n'est même plus réalisable.

Tous ces différents essais ne permettant pas de changer drastiquement le temps d'exécution, nous avons donc fait le choix de restreindre les durées de nos exécutions.

b) Choix des durées d'exécution

i) Impact d'une limite de temps en statique

Pour savoir ce que la réduction du temps d'exécution implique sur nos instances, on peut comparer les valeurs réalisables obtenues entre un temps d'exécution de 5 minutes et un autre de 30 minutes. On évalue pour cela l'écart relatif entre la solution en 5 minutes et celle en 30 minutes, on obtient le tableau suivant :

Instance	Optimal	Relative_difference
44_lin_9.tsp	false	0.16
38_rat_6.tsp	false	0.09
202_gr_3.tsp	false	0.08
38_rat_3.tsp	true	0.08
70_st_6.tsp	false	0.05
48_att_9.tsp	false	0.05
48_att_6.tsp	false	0.05
26_eil_9.tsp	true	0.04
202_gr_9.tsp	false	0.04
70_st_3.tsp	false	0.04
34_pr_6.tsp	false	0.04
80_gr_6.tsp	false	0.04
100_kroA_3.tsp	false	0.04
44_lin_6.tsp	false	0.03
44_lin_3.tsp	false	0.03
100_kroA_6.tsp	false	0.03
52_berlin_3.tsp	false	0.03
34_pr_9.tsp	false	0.03

Extrait du fichier contenant les écart relatifs trié par différence relative décroissante pour le modèle classique (lien vers le fichier)

On peut voir dans ce tableau que seules 4 instances ont des écarts relatifs de plus de 5% et qu'une moitié est en dessous de 2%. De plus, sur l'instance 30_eil_6 où on a trouvé l'opt en 5h, on est à moins d'1% de la solution optimale en 5 minutes avec un écart absolu de l'ordre de 3. Pourtant le gap est encore à 42% pour CPLEX (33% avec la borne citée plus haut). Le problème est donc de réussir à couper efficacement nos solutions pour éviter ce gap très élevé à proximité de l'optimal, car il semble qu'en static on obtient rapidement déjà une solution efficace.

ii) Impact sur le dual robuste

Par acquis de conscience, nous avons fait la même étude sur les premières instances pour les résultats avec le modèle dual.

Instance	Optimal	Relative_difference
70_st_3.tsp	false	0.33
48_att_9.tsp	false	0.25
52_berlin_9.tsp	false	0.22
38_rat_9.tsp	false	0.22
80_gr_6.tsp	false	0.2
48_att_3.tsp	false	0.16
40_eil_3.tsp	false	0.15
44_lin_9.tsp	false	0.14
48_att_6.tsp	false	0.14
70_st_6.tsp	false	0.14
44_lin_3.tsp	false	0.13
80_gr_3.tsp	false	0.1
70_st_9.tsp	false	0.09
26_eil_9.tsp	false	0.07
100_kroA_3.tsp	false	0.07

Extrait du fichier contenant les écart relatifs trié par différence relative décroissante pour le modèle robuste dual (lien vers le fichier)

Cette fois on voit des écarts plus grands apparaître, à noter que par manque de temps on n'a que les instances jusqu'à 100_… qui ont été évaluées en 30 min en dual. Toutefois, en 17% du temps on a déjà en moyenne plus de 85% de la solution. Or, comme faire tourner toutes les instances 5 minutes représente déjà 4h30 de temps d'exécution par modèle, c'est le choix que nous avons fait dans la suite pour pouvoir évaluer nos modifications et disposer de résultats sur toutes les instances.

D'autant plus que nous avons essayé également de jouer avec les paramètres de génération de coupes de CPLEX (CPX_PARAM_CLIQUES, CPX_PARAM_COVERS, CPX_PARAM_ZEROHALFCUTS, CPX_PARAM_MIRCUTS) pour fermer plus vite le gap (résultats sur le problème statique et le dual) mais cela fournissait des résultats incertains :

Instance	Optimal	Difference	Gap_difference
34_pr_3.tsp	false	0.02	-0.02
34_pr_6.tsp	false	-0.09	-0.08
34_pr_9.tsp	false	-0.02	-0.01
38_rat_3.tsp	false	0.09	0.11
38_rat_6.tsp	false	-0.05	-0.02
38_rat_9.tsp	false	0.12	0.02
40_eil_3.tsp	false	0.13	0.22
40_eil_6.tsp	false	0.04	0.02
40_eil_9.tsp	false	-0.08	0
44_lin_3.tsp	false	0.12	0.11
44_lin_6.tsp	false	-0.03	0.01
44_lin_9.tsp	false	0.15	0.04
48_att_3.tsp	false	0.1	0.13
48_att_6.tsp	false	-0.02	0.02
48_att_9.tsp	false	0.06	0.01
52_berlin_3.tsp	false	0.03	0.01
52_berlin_6.tsp	false	0.01	-0.02
52_berlin_9.tsp	false	0.07	0.01
70_st_3.tsp	false	0.08	0.03
70_st_6.tsp	false	0.09	0.02
70_st_9.tsp	false	0.35	0
80_gr_6.tsp	false	0.06	0
100_kroA_6.tsp	false	0.33	0.01

Extrait de la comparaison des résultats sur 5 minutes d'exécution pour le dual avec et sans les paramètres de coupes renforcées (lien vers le fichier)

Dans le tableau ci-dessus la troisième colonne correspond à la différence relative de valeur trouvée sans et avec les paramètres de coupe et la quatrième à la différence de gap. Des valeurs négatives signifient donc que la version avec les coupes est moins bonne et à l'inverse des valeurs positives qu'elle est meilleure. On observe que plus les instances sont grandes, plus il semble que les coupes sont utiles, sans pour autant pouvoir parler de régularité. On voit que la valeur peut parfois être bien meilleure avec les coupes (jusqu'à plus de 30% de gain) sans que le gap n'ait beaucoup évalué une fois cette solution réalisable trouvée (l'écart de gap est très faible par exemple pour 100_kro_6.tsp). Donc ce que nous voulions obtenir avec ces coupes n'a pas fonctionné.

En fin de compte nous nous sommes donc contentés de garder le modèle tel quel et de garder des temps d'exécutions courts de 5 minutes, faute de mieux.

3)Résultats

a) Discussion sur les méthodes

Des méthodes non heuristiques implémentées, la plus efficace est de loin la méthode duale. Elle est déjà plus rapide : elle réussit à trouver des solutions réalisables pour la quasi totalité des instances en 5 minutes environ (53/54) là où le branch and cut n'y arrive que pour la moitié environ (26/54) et les plans coupants sur encore moins d'instances (7/54). Ensuite, à durée équivalente elle est plus proche de l'optimal comme on peut le voir dans le tableau ci-dessous.

Les heuristiques quant à elles fournissent des résultats globalement moins bons, mais à partir des instances de taille 70 on voit apparaître des gaps plus faibles, en particulier logiquement pour la version avec recherche locale.

b) Instances résolues de manières optimales

On regroupe ci-dessous pour chaque méthode non heuristique les instances résolues de manière optimale en 5 minutes maximum.

i) Statique

Instance	Optimal	Time	Value	Gap
10_ulysses_3.tsp	true	0.17	54.35	0
10_ulysses_6.tsp	true	0.07	7.22	0
10_ulysses_9.tsp	true	0.07	0.72	0
14_burma_3.tsp	true	0.19	66.21	0
14_burma_6.tsp	true	0.14	17.96	0
14_burma_9.tsp	true	0.14	4.72	0
22_ulysses_3.tsp	true	0.58	284.21	0
22_ulysses_6.tsp	true	6.95	82.63	0
22_ulysses_9.tsp	true	1.6	26.22	0
26_eil_3.tsp	true	18.61	1850.42	0
30_eil_3.tsp	true	180.27	2693.05	0

Solutions optimales en statique, tirées de static_5.csv

ii) Dual

Instance	Optimal	Time	Value	Gap
10_ulysses_3.tsp	true	0.16	137	0
10_ulysses_6.tsp	true	0.07	55.12	0
10_ulysses_9.tsp	true	0.04	33.29	0
14_burma_3.tsp	true	0.16	93.39	0
14_burma_6.tsp	true	0.12	42.74	0
22_ulysses_3.tsp	true	2.57	358.64	0
22_ulysses_6.tsp	true	62.38	116.53	0
26_eil_3.tsp	true	137.18	2297.63	0

Solutions optimales avec le dual, tirées de dual_5.csv

iii) Branch and cut

Instance	Optimal	Time	Value	Gap
10_ulysses_3.tsp	true	2.58	137	0
10_ulysses_6.tsp	true	2.61	55.12	0
10_ulysses_9.tsp	true	7.28	33.29	0
14_burma_3.tsp	true	1.86	93.39	0
14_burma_6.tsp	true	14.36	42.74	0
22_ulysses_3.tsp	true	28.92	358.64	0

Solutions optimales avec le branch and cut, tirées de branch_and_cut_5.csv

iv) Plans coupants

Instance	Optimal	Time	Value	Gap
10_ulysses_3.tsp	true	1.48	137	0
10_ulysses_6.tsp	true	0.55	55.12	0
10_ulysses_9.tsp	true	0.82	33.29	0
14_burma_3.tsp	true	1.88	93.39	0
14_burma_6.tsp	true	1.15	42.74	0
14_burma_9.tsp	true	2.63	20.76	0
22_ulysses_3.tsp	true	53.38	358.64	0

Solutions optimales avec les plans coupants, tirées de cut_5.csv

c) Tableau des résultats

Le tableau des résultats est disponible au format csv sur ce lien. A noter pour comprendre son contenu que:

- pour PR:
  - une valeur seule en % correspond au coût réel entre l'optimal statique et l'optimal robuste quand il est connu,
  - une valeur préfixée de "<=" signifie qu'on connaît l'optimal statique mais seulement une valeur réalisable robuste et tout ce qu'on peut dire est que le PR sera inférieur à cela,
  - une valeur préfixée de "(?)" correspond au cas où ni le robuste ni le statique ne sont optimaux et on en est réduit à comparer deux valeurs réalisables sans garantie.
- pour le temps:
  - quand on a "---" cela signifie que qu'il n'y a pas eu de solution réalisables dans le temps imparti,
  - si un temps dépasse de manière importante les 300 secondes, c'est lié au fait qu'une opération "unitaire" a débuté avant 300 secondes strictement et n'a pu être terminée que longtemps après.
- pour le gap:
  - si on a "---" pour le temps et le gap c'est encore une fois que la solution retournée n'était pas réalisable au terme du temps,
  - s on a "---" pour le gap seulement, c'est que le gap était de 1 exactement et donc que la borne pouvait être aussi loin que l'on veut de la valeur réalisable trouvée et donc on n'est pas en mesure de retrouver exactement la borne à partir de la valeur et du gap, car on n'a stocké que les gaps et non les bornes dans nos tableaux. Plutôt que de dire des choses fausses, on préfère donc mettre un gap vide.

Instance	PR	Dual_time	Dual_gap	Branch_time	Branch_gap	Plans_time	Plans_gap	Heur_time	Heur_gap	Heur_local_time	Heur_local_gap
10_ulysses_3.tsp	152.0%	0.16s	-0.0%	2.58s	0.0%	1.48s	-0.0%	0.18s	17.1%	2.2s	13.8%
10_ulysses_6.tsp	663.2%	0.07s	-0.0%	2.61s	0.0%	0.55s	0.0%	0.18s	59.8%	3.47s	-0.0%
10_ulysses_9.tsp	4522.1%	0.04s	0.0%	7.28s	0.0%	0.82s	0.0%	0.23s	22.6%	3.33s	0.0%
14_burma_3.tsp	41.0%	0.16s	-0.0%	1.86s	0.0%	1.88s	0.0%	0.9s	92.3%	2.39s	75.3%
14_burma_6.tsp	137.9%	0.12s	0.0%	14.36s	0.0%	1.15s	0.0%	0.25s	60.4%	4.72s	14.5%
14_burma_9.tsp	339.4%	300.02s	0.0%	300.03s	0.0%	2.63s	0.0%	0.37s	108.2%	9.77s	31.7%
22_ulysses_3.tsp	26.2%	2.57s	0.0%	28.92s	0.0%	53.38s	0.0%	0.27s	48.7%	17.36s	25.4%
22_ulysses_6.tsp	41.0%	62.38s	0.0%	300.01s	27.2%	---	---	0.88s	112.0%	34.21s	45.6%
22_ulysses_9.tsp	<= 147.8%	300.58s	94.9%	300.02s	99.5%	---	---	1.41s	387.4%	36.37s	101.4%
26_eil_3.tsp	24.2%	137.18s	0.0%	300.0s	5.3%	---	---	0.65s	49.9%	40.32s	11.3%
26_eil_6.tsp	(?) 77.5%	300.03s	45.7%	300.02s	59.8%	---	---	0.59s	111.9%	51.7s	54.9%
26_eil_9.tsp	(?) 152.8%	300.03s	64.4%	300.01s	99.3%	---	---	0.94s	135.6%	54.73s	67.1%
30_eil_3.tsp	<= 17.2%	300.02s	25.0%	300.02s	42.6%	---	---	0.43s	66.7%	70.11s	34.0%
30_eil_6.tsp	(?) 58.0%	265.99s	50.5%	300.01s	95.6%	---	---	0.76s	117.2%	53.49s	75.1%
30_eil_9.tsp	(?) 50.4%	300.03s	32.2%	300.02s	99.8%	---	---	1.22s	117.2%	90.88s	68.7%
34_pr_3.tsp	(?) 35.6%	300.01s	25.2%	300.02s	48.6%	---	---	0.54s	75.5%	93.24s	49.3%
34_pr_6.tsp	(?) 85.9%	300.55s	53.5%	300.01s	98.4%	---	---	0.96s	139.2%	209.77s	57.4%
34_pr_9.tsp	(?) 124.4%	300.15s	63.2%	300.01s	100.0%	---	---	1.38s	118.7%	222.33s	40.1%
38_rat_3.tsp	(?) 35.0%	300.09s	41.5%	300.01s	59.9%	---	---	0.55s	86.0%	159.57s	43.4%
38_rat_6.tsp	(?) 67.5%	300.36s	53.6%	300.0s	98.4%	---	---	1.03s	110.6%	271.86s	56.2%
38_rat_9.tsp	(?) 167.0%	300.89s	98.0%	---	---	---	---	1.61s	118.0%	292.63s	69.6%

Tableau des résultats dans resultats.csv, (1/3)

Instance	PR	Dual_time	Dual_gap	Branch_time	Branch_gap	Plans_time	Plans_gap	Heur_time	Heur_gap	Heur_local_time	Heur_local_gap
40_eil_3.tsp	(?) 21.6%	301.01s	66.8%	300.01s	68.8%	---	---	0.62s	96.9%	187.68s	51.2%
40_eil_6.tsp	(?) 71.6%	300.33s	78.8%	300.02s	82.0%	---	---	1.12s	115.4%	265.03s	68.6%
40_eil_9.tsp	(?) 111.5%	300.55s	98.5%	---	---	---	---	1.61s	108.0%	220.65s	58.1%
44_lin_3.tsp	(?) 33.6%	300.31s	66.2%	300.0s	61.1%	---	---	0.68s	93.3%	276.01s	60.0%
44_lin_6.tsp	(?) 86.5%	300.33s	84.6%	---	---	---	---	1.4s	127.3%	326.58s	72.8%
44_lin_9.tsp	(?) 122.1%	300.46s	100.0%	---	---	---	---	1.97s	161.0%	322.11s	68.0%
48_att_3.tsp	(?) 25.7%	300.02s	58.2%	300.02s	71.0%	---	---	0.9s	105.7%	291.27s	40.5%
48_att_6.tsp	(?) 96.8%	300.12s	87.4%	---	---	---	---	4.73s	150.2%	204.63s	89.0%
48_att_9.tsp	(?) 60.6%	300.09s	90.8%	---	---	---	---	2.35s	148.2%	322.45s	26.5%
52_berlin_3.tsp	(?) 15.8%	300.06s	54.1%	300.02s	77.1%	---	---	0.93s	97.4%	300.13s	57.4%
52_berlin_6.tsp	(?) 41.7%	300.21s	82.5%	---	---	---	---	1.74s	161.5%	300.09s	106.3%
52_berlin_9.tsp	(?) 142.0%	300.06s	96.8%	---	---	---	---	---	---	---	---
70_st_3.tsp	(?) 21.0%	300.08s	73.2%	300.02s	88.2%	---	---	1.42s	83.3%	300.2s	63.5%
70_st_6.tsp	(?) 93.1%	300.06s	95.5%	---	---	---	---	2.83s	89.8%	300.23s	82.5%
70_st_9.tsp	(?) 108.7%	300.07s	99.9%	---	---	---	---	4.32s	97.9%	300.26s	69.7%
80_gr_3.tsp	(?) 28.8%	300.15s	84.6%	---	---	---	---	1.83s	106.6%	300.08s	103.4%
80_gr_6.tsp	(?) 102.7%	300.04s	95.8%	---	---	---	---	3.59s	96.3%	300.17s	96.3%
80_gr_9.tsp	(?) 138.9%	300.08s	100.0%	---	---	---	---	5.34s	107.2%	300.5s	100.9%
100_kroA_3.tsp	(?) 28.9%	300.13s	90.5%	---	---	---	---	2.7s	111.6%	300.18s	110.4%
100_kroA_6.tsp	(?) 85.2%	300.62s	97.6%	---	---	---	---	5.3s	97.8%	300.26s	92.2%
100_kroA_9.tsp	(?) 123.0%	300.08s	---	---	---	---	---	7.89s	---	300.15s	---

Tableau des résultats dans resultats.csv, (2/3)

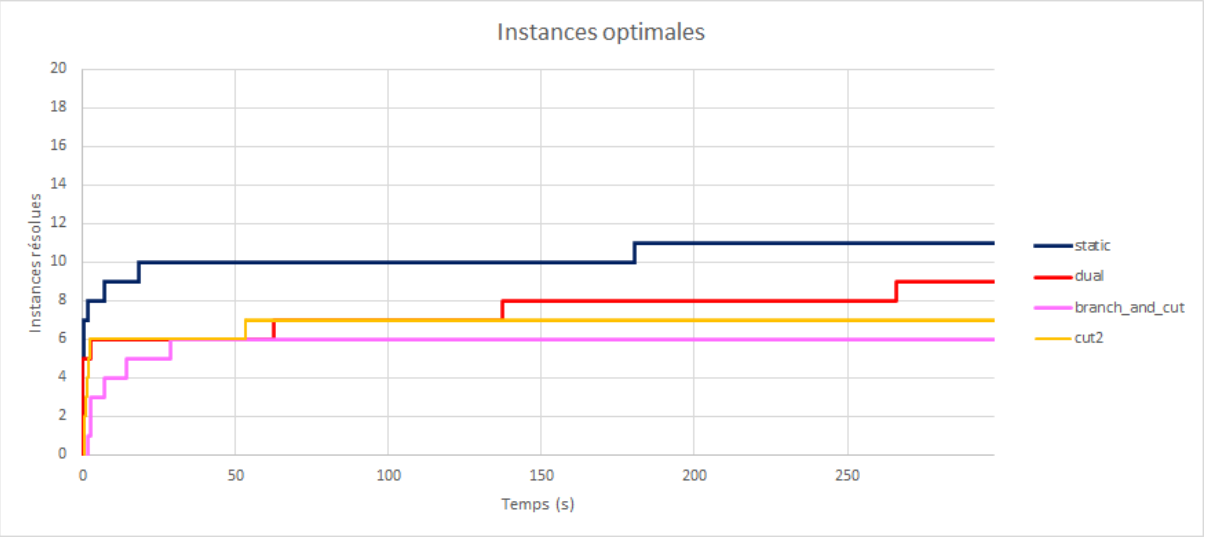
Instance	PR	Dual_time	Dual_gap	Branch_time	Branch_gap	Plans_time	Plans_gap	Heur_time	Heur_gap	Heur_local_time	Heur_local_gap
202_gr_3.tsp	(?) 14.9%	300.08s	99.5%	---	---	---	---	9.84s	-69.5%	300.41s	-69.5%
202_gr_6.tsp	(?) 34.0%	300.1s	99.5%	---	---	---	---	19.45s	72.7%	514.72s	71.8%
202_gr_9.tsp	(?) 59.0%	300.16s	99.7%	---	---	---	---	29.29s	100.2%	300.22s	100.2%
318_lin_3.tsp	(?) 32.6%	300.31s	99.3%	---	---	---	---	23.43s	91.6%	300.37s	89.8%
318_lin_6.tsp	(?) 41.2%	300.44s	99.7%	---	---	---	---	46.41s	94.2%	300.45s	93.1%
318_lin_9.tsp	(?) 94.5%	300.48s	---	---	---	---	---	69.56s	---	300.34s	---
400_rd_3.tsp	(?) -0.7%	300.26s	99.7%	---	---	---	---	37.1s	86.7%	300.23s	85.5%
400_rd_6.tsp	(?) -1.4%	---	---	---	---	---	---	74.23s	---	300.57s	---
400_rd_9.tsp	(?) -3.1%	300.62s	100.0%	---	---	---	---	109.99s	100.0%	355.29s	99.1%
532_att_3.tsp	(?) 26.4%	317.63s	99.8%	---	---	---	---	62.85s	86.6%	300.3s	86.0%
532_att_6.tsp	(?) 19.2%	359.46s	---	---	---	---	---	125.95s	---	406.99s	---
532_att_9.tsp	(?) 35.2%	330.28s	100.0%	---	---	---	---	189.13s	55.7%	418.42s	54.6%

Tableau des résultats dans resultats.csv, (3/3)

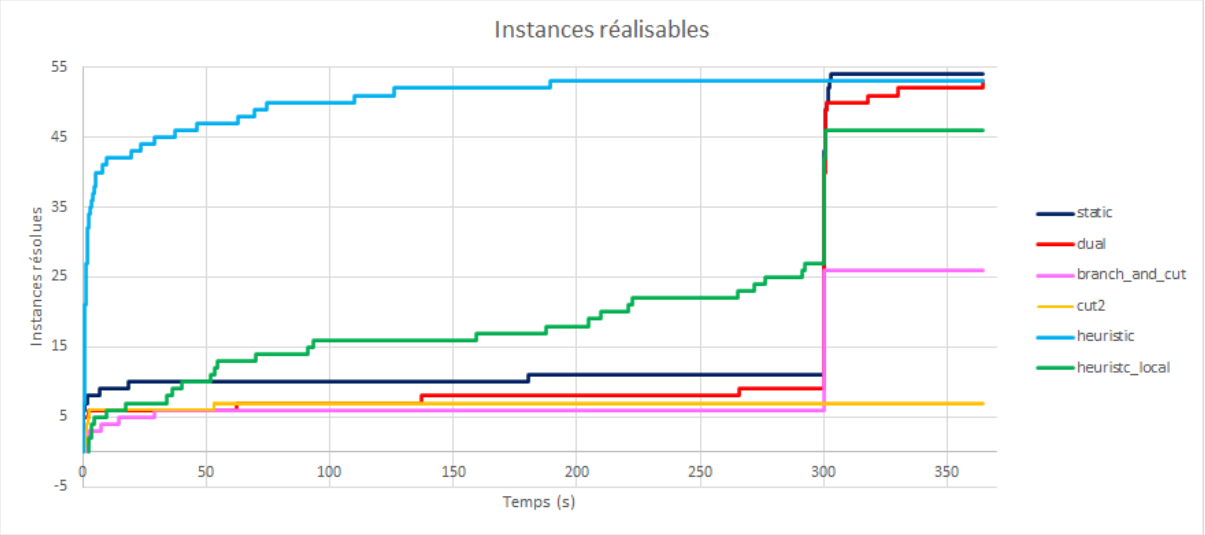
d) Diagramme de performances

Les deux diagrammes ci-dessous sont tirés du tableur diagram.xlsx.

i) Instances optimales



ii) Instances réalisables



4)Annexes

Comme demandé, on présente ici les meilleures valeurs d'objectif obtenues pour les premières instances. Pour être homogène on compare les résultats avec des temps d'exécution maximal de 5 minutes comme indiqué plus tôt.

a) Statique

Solving 10_ulysses_3.tsp in static mode.
Success, nodes : 0, Time : 0.107 Value : 54.3548
Part 1 Any[1, 5, 8]
Part 2 Any[6, 7, 9, 10]
Part 3 Any[2, 3, 4]

Solving 10_ulysses_6.tsp in static mode.
Success, nodes : 0, Time : 0.067 Value : 7.222
Part 1 Any[1, 8]
Part 2 Any[6, 7]
Part 3 Any[5]
Part 4 Any[2, 3]
Part 5 Any[9, 10]
Part 6 Any[4]

Solving 10_ulysses_9.tsp in static mode.
Success, nodes : 0, Time : 0.048 Value : 0.7203
Part 1 Any[1, 8]
Part 2 Any[2]
Part 3 Any[3]
Part 4 Any[7]
Part 5 Any[5]
Part 6 Any[10]
Part 7 Any[4]
Part 8 Any[9]
Part 9 Any[6]

Solving 14_burma_3.tsp in static mode.
Success, nodes : 0, Time : 0.174 Value : 66.2137
Part 1 Any[1, 2, 8, 9, 10, 11]
Part 2 Any[3, 4, 12, 14]
Part 3 Any[5, 6, 7, 13]

Solving 14_burma_6.tsp in static mode.
Success, nodes : 456, Time : 0.119 Value : 17.9622
Part 1 Any[1, 2, 8]
Part 2 Any[6, 12, 14]
Part 3 Any[3, 4]
Part 4 Any[7, 13]

Part 5 Any[9, 10, 11]
Part 6 Any[5]

Solving 14_burma_9.tsp in static mode.
Success, nodes : 100, Time : 0.155 Value : 4.7248
Part 1 Any[1, 8]
Part 2 Any[2]
Part 3 Any[3, 14]
Part 4 Any[4]
Part 5 Any[7, 13]
Part 6 Any[6, 12]
Part 7 Any[5]
Part 8 Any[10]
Part 9 Any[9, 11]

Solving 22_ulysses_3.tsp in static mode.
Success, nodes : 473, Time : 0.689 Value : 284.2098
Part 1 Any[1, 2, 3, 4, 8, 16, 17, 18, 22]
Part 2 Any[7, 10, 12, 13, 14, 19, 20, 21]
Part 3 Any[5, 6, 9, 11, 15]

Solving 22_ulysses_6.tsp in static mode.
Success, nodes : 25130, Time : 6.766 Value : 82.631
Part 1 Any[1, 8, 16, 22]
Part 2 Any[7, 12, 13, 14]
Part 3 Any[10, 19, 20, 21]
Part 4 Any[2, 3, 4, 17, 18]
Part 5 Any[5, 6, 15]
Part 6 Any[9, 11]

Solving 22_ulysses_9.tsp in static mode.
Success, nodes : 3058, Time : 0.753 Value : 26.2172
Part 1 Any[1, 8, 16]
Part 2 Any[2, 3, 17]
Part 3 Any[10, 19, 20, 21]
Part 4 Any[11]
Part 5 Any[6, 7]
Part 6 Any[12, 13, 14]
Part 7 Any[9]
Part 8 Any[5, 15]
Part 9 Any[4, 18, 22]

b) Robuste

Solving 10_ulysses_3.tsp in dual mode.
Success, nodes : 0, Time : 0.14 Value : 136.9953
Part 1 Any[1, 2, 3, 10]
Part 2 Any[4, 6, 7, 8]

Part 3 Any[5, 9]

Solving 10_ulysses_6.tsp in dual mode.
Success, nodes : 18, Time : 0.077 Value : 55.1194
Part 1 Any[1, 4, 8]
Part 2 Any[7]
Part 3 Any[2, 3]
Part 4 Any[10]
Part 5 Any[6, 9]
Part 6 Any[5]

Solving 10_ulysses_9.tsp in dual mode.
Success, nodes : 7327, Time : 0.626 Value : 33.2919
Part 1 Any[1]
Part 2 Any[7]
Part 3 Any[2, 3]
Part 4 Any[10]
Part 5 Any[9]
Part 6 Any[5]
Part 7 Any[6]
Part 8 Any[8]
Part 9 Any[4]

Solving 14_burma_3.tsp in dual mode.
Success, nodes : 0, Time : 0.168 Value : 93.39
Part 1 Any[1, 8, 9, 10, 11]
Part 2 Any[3, 4, 5, 6, 7]
Part 3 Any[2, 12, 13, 14]

Solving 14_burma_6.tsp in dual mode.
Success, nodes : 81, Time : 0.194 Value : 42.7406
Part 1 Any[1, 8, 11]
Part 2 Any[6, 12, 14]
Part 3 Any[9, 10]
Part 4 Any[7]
Part 5 Any[2, 13]
Part 6 Any[3, 4, 5]

Solving 14_burma_9.tsp in dual mode.
Success, nodes : 1333107, Time : 300.031 Value : 20.7624
Part 1 Any[1, 8]
Part 2 Any[3, 4]
Part 3 Any[5]
Part 4 Any[6, 12]
Part 5 Any[7]
Part 6 Any[9, 11]
Part 7 Any[2]
Part 8 Any[10]

Part 9 Any[13, 14]

Solving 22_ulysses_3.tsp in dual mode.
Success, nodes : 3820, Time : 3.229 Value : 358.6368
Part 1 Any[1, 2, 3, 4, 8, 16, 17, 18]
Part 2 Any[7, 10, 12, 13, 14, 19, 20, 22]
Part 3 Any[5, 6, 9, 11, 15, 21]

Solving 22_ulysses_6.tsp in dual mode.
Success, nodes : 60087, Time : 50.026 Value : 116.5288
Part 1 Any[1, 8, 16, 22]
Part 2 Any[7, 11]
Part 3 Any[5, 6, 14, 15]
Part 4 Any[2, 3, 4, 17, 18]
Part 5 Any[9, 13, 19]
Part 6 Any[10, 12, 20, 21]

Solving 22_ulysses_9.tsp in dual mode.
Success, nodes : 286231, Time : 300.035 Value : 64.9736
Part 1 Any[1, 16, 22]
Part 2 Any[6, 10]
Part 3 Any[5, 9]
Part 4 Any[7, 13]
Part 5 Any[11, 19]
Part 6 Any[20, 21]
Part 7 Any[12, 14, 15]
Part 8 Any[2, 3, 17]
Part 9 Any[4, 8, 18]