Rapport de projet

MU5IN256 - Modèles et Algorithmes pour la Décision Multicritères et Collective

Élicitation incrémentale et recherche locale pour le problème du sac à dos multi-objectifs

Master d'informatique spécialité androide

Deuxième année

Année universitaire 2021 - 2022

Étudiants :

Vincent Fu
Yuhao Liu

1. Implémentation pour les procédures de résolution

Voici les différentes fonctions implémentées pour les deux procédures de résolution :

— Une classe `QuadTree` contenue dans le fichier tree.py permettant de gérer un front de points non mutuellement dominés sous forme d'arbre. Afin de vérifier que le `QuadTree` ne fait pas d'erreur calculs, la fonction `test_dominance` vérifie si le front ne contient pas de points dominés.
Pour le vérifier, entrez dans le terminal : `python tree.py`

— Des fonctions utiles (lire les données, calculer les valeurs, etc ...) pour le fonctionnement des autres fonctions. Ces fonctions sont contenues dans le fichier utils.py

— La fonction `pls` contenue dans le fichier pls.py effectuant l'algorithme *Pareto local search*. Afin de vérifier que le PLS ne fait pas d'erreur de calculs, les fichiers 2K100-TA-0.dat et 2K100-TA-0.eff ont été utilisés pour calculer la proportion des points approximés Pareto-optimale pour le problème. On obtient des proportions autour de 30 % à 70 % sur tout l'instance (200 objets et 2 critères).
Pour le vérifier, entrez dans le terminal : `python pls.py`
À noter qu'en lançant cette commande, 3 PLS sont effectués afin de vérifier toutes les méthodes : un PLS classique pour l'instance 2K100-TA-0.dat, un PLS à éliciation pour le dernier front des points approximés et un PLS à élicitation incrémentale pour le fichier 2K200-TA-0.dat.

— Une fonction d'affichage graphique `vizualize` contenue dans le fichier visualiation.py permettant d'observer graphiquement l'évolution des points obtenus par PLS et l'élicitation.

— Les fonctions génératrices de poids (somme pondérée, OWA, Choquet), une classe de décideur `decision_maker` et les fonctions pour l'élicitation (les programmes linéaires pour le regret minimax le Ω-filtre) sont contenues dans le fichier elicitation.py.

— Un pragramme linéaire `true_sol` contenu dans solution.py permettant de calculer la solution exacte préférée du décideur selon l'agrégateur utilisé.

— Enfin, un programme générique d'exécution main.py pour l'interaction sur terminal pour obtenir les résultats des instances avec les paramètres de l'utilisateur (voir le fichier README.md).

— Une fonction de traçage de graphe `draw_graph_mmr` contenue dans le fichier graph.py pour observer l'évolution du regret minimax en du nombre de questions posées. Le graphe obtenu est mmr_graph.png.
À noter que le multiprocessing a été utilisé pour obtenir plus rapidement les résultats (graphe et tableau).

2. Résultats des procédures de résolution

Voici les différentes fonctions implémentées pour obtenir les résultats :

— Une fonction de traçage de graphe `draw_graph_mmr` contenue dans le fichier graph.py pour observer l'évolution du regret minimax en du nombre de questions posées. Le graphe obtenu est mmr_graph.png.

— Une fonction `experiment` contenue dans resultat.py qui effectue les 2 procédures de résolutions selon les paramètres du problème et enregistre ces résultats dans le dossier logs sous le fichier :
<AGREGATEUR>_<STRATEGIE>_<NB_OBJETS>_N<NB_CRITERES>.log
où AGREGATEUR est LW (somme pondérée), OWA ou CHOQ (intégrale de Choquet)
STRAGEGIE est RANDOM ou CSS (current solution strategy).
Les résultats sont sous le format suivant : 5 valeurs dont :

- La première est soit c ou e : c pour une élicitation à la fin du PLS, e pour une élicitation incrémentale.
- La deuxième est le nombre de jeux effectués pour faire une moyenne des résultats.
- La troisième est le temps moyen d'exécution.
- La quatrième est le nombre moyen de questions posées.
- La cinquième est l'écart relatif moyen de la valeur de la fonction objectif par rapport à l'optimum.

À noter que le multiprocessing a été utilisé pour obtenir plus rapidement les résultats (graphe et tableau).

Pour une agrégation à somme pondérée, voici l'évolution du regret minimax en fonction du nombre de questions (voir FIGURE 1) :

Le graphe ci-dessus montre que plus la taille des instances est grande, plus il est nécessaire de poser plus de questions afin de discriminer les solutions. Le minimax regret décroît toujours en fonction du nombre de questions posées. En particulier, le nombre de questions posées est relativement faible par rapport à la taille des solutions à comparer. L'élicitation par le minimax regret est donc efficace.

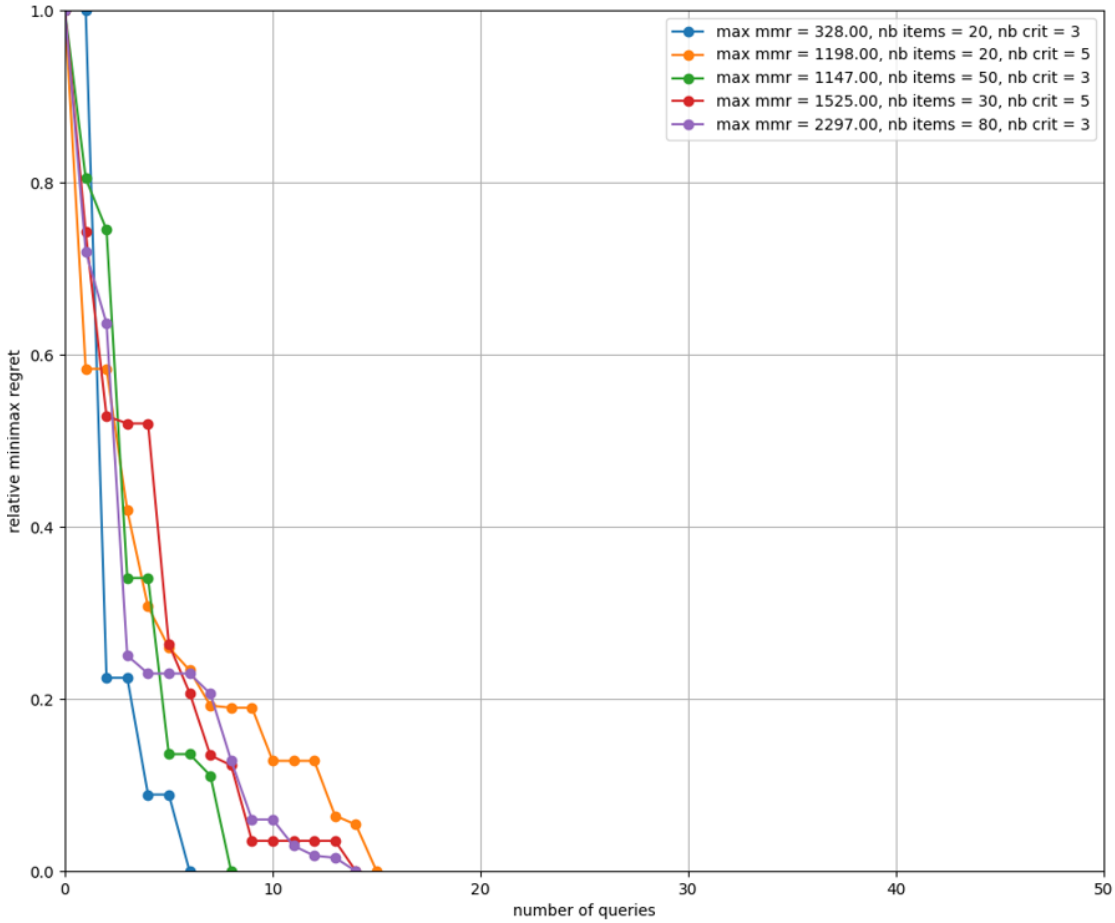

FIGURE 1 – *Évolution du minimax regret relatif en fonction du nombre de questions posées pour une stratégie de questionnement aléatoire*

Voici à présent les résultats des deux procédures en terme de temps d'exécution en s, du nombre de questions posées pendant toute la procédure et ainsi que l'écart relatif à l'optimum (voir les TABLE [1]) :

D'après les résultats obtenus, la procédure 2 est toujours plus efficace que la procédure 1 en terme de temps d'exécution (très efficace même). En contrepartie, la procédure 2 nécessite généralement un peu plus de questions à poser au décideur. Par ailleurs, pour toutes les instances ayant le nombre d'objets inférieures à 50, l'écart de la solution agrégée approximée par rapport à la solution agrégée exacte est extrêmement faible (en dessous des 1 %). Enfin, les performances des procédures pour l'agrégateur OWA est généralement meilleures que celles de la somme pondérée en terme de temps d'exécution et du nombre de questions posées tandis que l'agrégateur d'intégrale de Choquet est le plus mauvais et nécessite un temps d'exécution plus long et beaucoup plus de questions.

1. / indique que le temps d'exécution pour calculer une moyenne a été trop long pour extraire les résultats. Dans les fichiers .log s'ils existent, le / est représenté par des 0 en résultat pour le time, queries et gap.

	Somme pondérée						OWA						Choquet					
	Procédure 1			Procédure 2			Procédure 1			Procédure 2			Procédure 1			Procédure 2		
n	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap
2	0.36	3.7	0.002	0.16	6.8	0.006	0.33	2.0	0.002	0.15	5.6	0.009	0.53	3.6	0.001	0.37	6.8	0.013
3	2.14	6.0	0.000	0.63	8.7	0.004	1.91	2.7	0.002	0.53	7.5	0.012	6.56	8.9	0.000	2.99	11.2	0.007
4	14.96	9.4	0.000	1.85	10.3	0.005	16.56	3.0	0.000	1.64	9.4	0.006	118.05	15.7	0.000	16.78	15.7	0.000
5	299.36	13.3	0.000	6.04	12.3	0.003	351.22	5.8	0.000	7.34	12.1	0.002	/	/	/	75.32	24.6	0.014
6	/	/	/	8.20	29.6	0.006	/	/	/	3.31	10.9	0.016	/	/	/	/	/	/

TABLE 1 – *Résultats des procédures pour une stratégie de questionnement aléatoire pour 25 objets sur une moyenne de 20 instances pour chaque paramètres.*

	Somme pondérée						OWA						Choquet					
	Procédure 1			Procédure 2			Procédure 1			Procédure 2			Procédure 1			Procédure 2		
n	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap
2	3.54	3.0	0.004	0.54	10.8	0.008	4.63	2.9	0.000	0.66	10.9	0.005	4.42	5.1	0.001	1.11	11.2	0.006
3	82.07	7.4	0.000	2.50	13.3	0.005	75.62	3.9	0.000	2.12	13.6	0.004	158.82	11.8	0.000	8.42	15.8	0.005
4	879.62	15.3	0.000	7.77	16.3	0.001	600.40	4.7	0.000	3.53	14.0	0.000	/	/	/	23.99	27.4	0.009
5	/	/	/	20.42	33.6	0.005	/	/	/	10.46	17.7	0.003	/	/	/	169.30	35.8	0.007
6	/	/	/	38.87	39.1	0.003	/	/	/	29.42	20.2	0.004	/	/	/	/	/	/

TABLE 2 – *Résultats des procédures pour une stratégie de questionnement aléatoire pour 50 objets sur une moyenne de 20 instances pour chaque paramètres.*

	Somme pondérée						OWA						Choquet					
	Procédure 1			Procédure 2			Procédure 1			Procédure 2			Procédure 1			Procédure 2		
n	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap
2	0.33	3.0	0.007	0.18	8.0	0.016	0.41	2.0	0.001	0.27	8.7	0.012	0.48	3.5	0.001	0.43	8.2	0.013
3	1.96	5.1	0.004	0.76	11.2	0.013	2.21	2.8	0.000	0.94	11.0	0.011	4.89	7.9	0.004	2.75	12.0	0.014
4	14.51	10.2	0.000	1.57	10.8	0.010	15.61	3.0	0.000	1.81	9.5	0.008	101.27	11.7	0.000	23.96	16.3	0.003
5	308.75	18.2	0.000	8.81	16.9	0.002	335.64	6.5	0.000	4.45	9.4	0.006	/	/	/	88.82	27.4	0.014
6	/	/	/	17.20	35.0	0.007	/	/	/	8.13	14.9	0.009	/	/	/	/	/	/

TABLE 3 – *Résultats des procédures pour une stratégie de questionnement CSS pour 25 objets sur une moyenne de 20 instances pour chaque paramètres.*

	Somme pondérée						OWA						Choquet					
	Procédure 1			Procédure 2			Procédure 1			Procédure 2			Procédure 1			Procédure 2		
n	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap	time	queries	gap
2	3.57	3.2	0.002	0.56	10.5	0.013	4.71	2.4	0.000	0.70	11.3	0.007	4.53	5.0	0.001	1.13	11.8	0.009
3	75.29	7.3	0.000	2.29	13.8	0.003	71.90	3.5	0.000	2.24	13.9	0.002	166.5	11.7	0.000	10.85	17.6	0.002
4	647.25	13.1	0.000	5.49	16.2	0.003	/	/	/	4.15	17.6	0.001	/	/	/	24.54	26.7	0.010
5	/	/	/	18.24	34.5	0.004	/	/	/	15.43	20.2	0.004	/	/	/	152.91	34.9	0.013
6	/	/	/	48.66	47.3	0.002	/	/	/	29.36	21.6	0.003	/	/	/	/	/	/

TABLE 4 – *Résultats des procédures pour une stratégie de questionnement CSS pour 50 objets sur une moyenne de 20 instances pour chaque paramètres.*

Références

[1] Nawal BENABBOU, Cassandre LEROY et Thibaut LUST. "Regret-Based Elicitation for Solving Multi-Objective Knapsack Problems with Rank-Dependent Aggregators". In : *The 24th European Conference on Artificial Intelligence (ECAI'20)*. Saint Jacques de Compostelle, Spain, juin 2020. URL : https://hal.sorbonne-universite.fr/hal-02493998.

[2] Nawal BENABBOU et al. "Minimax Regret Approaches for Preference Elicitation with Rank-Dependent Aggregators". In : *EURO Journal on Decision Processes* 3.1-2 (juin 2015), p. 29-64. DOI : 10.1007/s40070-015-0040-6. URL : https://hal.archives-ouvertes.fr/hal-01170030.