Sorbonne Université
Master ANDROIDE

Rapport de projet : Élicitation incrémentale et recherche locale pour le problème du sac à dos multi-objectifs

MADMC

Jules Bouton Popper
Amélie Sun

Année universitaire 2023-2024

Table des matières

Chapitre 1

Première méthode

1.1 Présentation

Dans ce projet, nous allons nous intéresser à la résolution du problème du sac-à-dos multi-objectifs. On cherche à maximiser une fonction d'agrégation dont les paramètres sont connus uniquement du décideur. Dans la première méthode, on commencera par faire une approximation du front de Pareto par recherche locale auquel nous appliquerons ensuite la procédure d'élicitation incrémentale. Pour essayer de minimiser le nombre de questions, nous avons implémenté la Current Solution Strategy (CSS) [1].

1.2 Choix d'implémentation

Pour PLS, nous nous sommes basés sur le travail réalisé en TME. Nous prenons le voisinage échange 1-1 dans lequel on retire un objet de la solution courante et on ajoute un objet qui n'était pas dans la solution. Nous avons implémenté la structure de données ND-tree [2] afin de gérer efficacement la mise à jour de l'approximation du front de Pareto. De plus, nous avons l'option d'utiliser une liste triée pour stocker le front, qui est plus rapide que ND-tree pour 2 objectifs. Pour l'élicitation, nous avons implémenté les trois fonctions d'agrégations du sujet, à savoir la somme pondérée, OWA avec des poids décroissant et l'intégrale de Choquet avec des capacités super-modulaires. Nous calculons les n^2 PMR, mais nous avons parallélisé ces calculs afin d'accélérer notre implémentation.

1.3 Analyse des résultats

Dans cette section, on compare le nombre de questions posées afin de trouver la solution optimale parmi celles trouvées par la recherche locale, ainsi que l'évolution du regret minimax en fonction du nombre de questions. On pourra le visualiser sur deux graphiques, l'un montrant pour chaque décideur l'évolution et l'autre montrant une vue en moyenne. Pour chaque fonction, nous avons répété l'expérience sur 20 jeux de paramètres tirés aléatoirement, sauf pour $d = 3, n = 20$, pour lequel on a tiré 100 décideurs. On note à chaque fois le nombre d'objets, le nombre de critères et la taille du front de Pareto obtenu ($|X|$). Les instances considérées sont des sous-instances du fichier donné dans le projet, avec les objets et les critères choisis de manière aléatoire. Tout les calculs ont été réalisés sur `ppti-gpu-1` accessible aux étudiants, et qui est équipée d'un Intel Xeon Gold 6152.

	Φ	$\#Q$	Temps (s)
$d = 3, n = 20, \|X\| = 121$	WS	8 ± 2	13 ± 3
	OWA_m	3 ± 0	7 ± 0
	C_{sm}	13 ± 3	24 ± 6
$d = 3, n = 60, \|X\| = 966$	WS	11 ± 3	348 ± 141
	OWA_m	5 ± 2	81 ± 21
	C_{sm}	11 ± 3	237 ± 68
$d = 4, n = 35, \|X\| = 1134$	WS	16 ± 4	865 ± 337
	OWA_m	6 ± 1	134 ± 10
	C_{sm}	29 ± 7	2120 ± 774
$d = 5, n = 25, \|X\| = 407$	WS	19 ± 6	186 ± 80
	OWA_m	3 ± 1	16 ± 2
	C_{sm}	25 ± 4	415 ± 95

FIGURE 1.1 – Comparaison du nombre de questions et du temps d'exécution

On constate dans le tableau 1.1 que le nombre de questions posées pour Choquet et la somme pondérée sont du même ordre tandis que pour l'OWA il y en a beaucoup moins. Cela s'explique par le fait que l'on a aucune notion de préférence pour les solutions de compromis avec la somme pondérée d'une part, et d'autre part qu'il y a peu de solutions OWA-optimales différentes pour des poids décroissants, ce qui n'est pas vrai pour Choquet avec des capacités super-modulaires (de par sa flexibilité). On peut voir sur les figures 1.2 et 1.12 que le MMR décroît fortement au début de la procédure puis converge plus lentement vers une valeur négative. De plus, on a pu observer sur nos expériences que le MMR diminuait lentement quand on avait trouvé la solution optimale (autrement dit que le regret réel atteignait 0), les questions restantes servant plus à s'assurer que c'est bien la solution optimale. On converge donc assez rapidement vers la bonne solution, et si on stoppait la procédure préemptivement, on aurait de grandes chances d'avoir une solution optimale. On pourrait borner l'erreur par le MMR de l'itération d'arrêt.

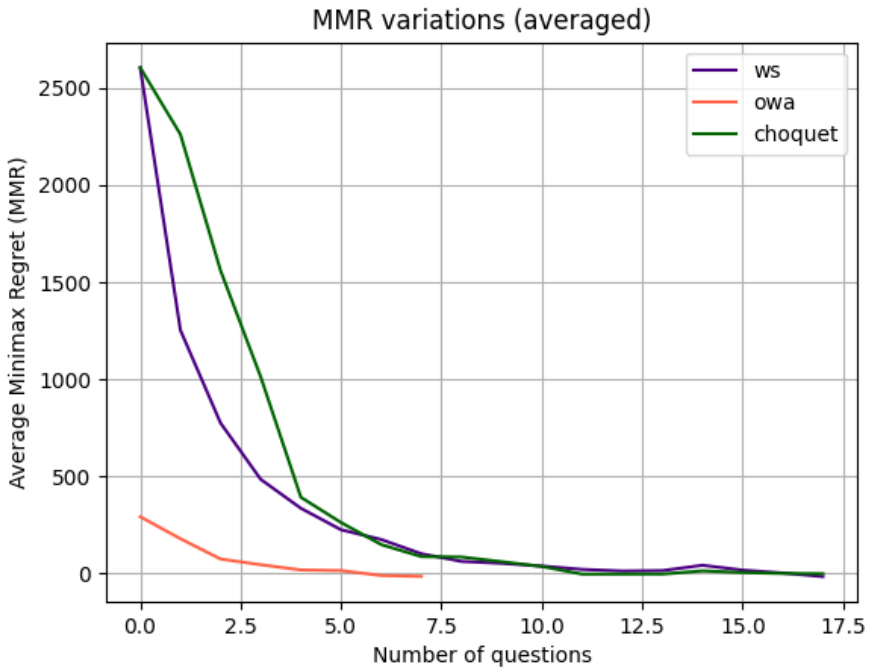

FIGURE 1.2 – Variation moyenne du MMR (d=3, n=60, $|X|$=966)

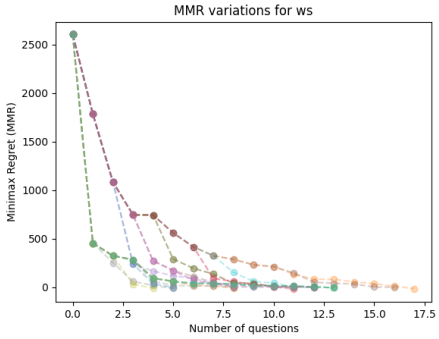

FIGURE 1.3 – Somme pondérée

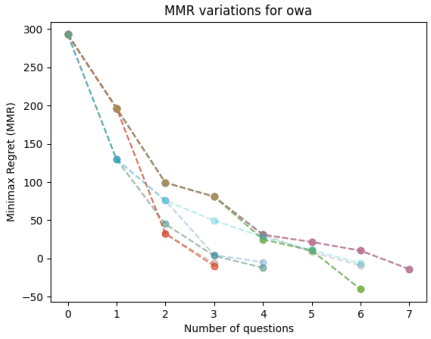

FIGURE 1.4 – OWA

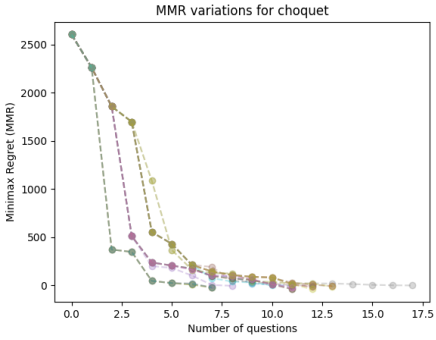

FIGURE 1.5 – Choquet

FIGURE 1.6 – Variations individuelles du MMR (d=3, n=60, $|X|$=966)

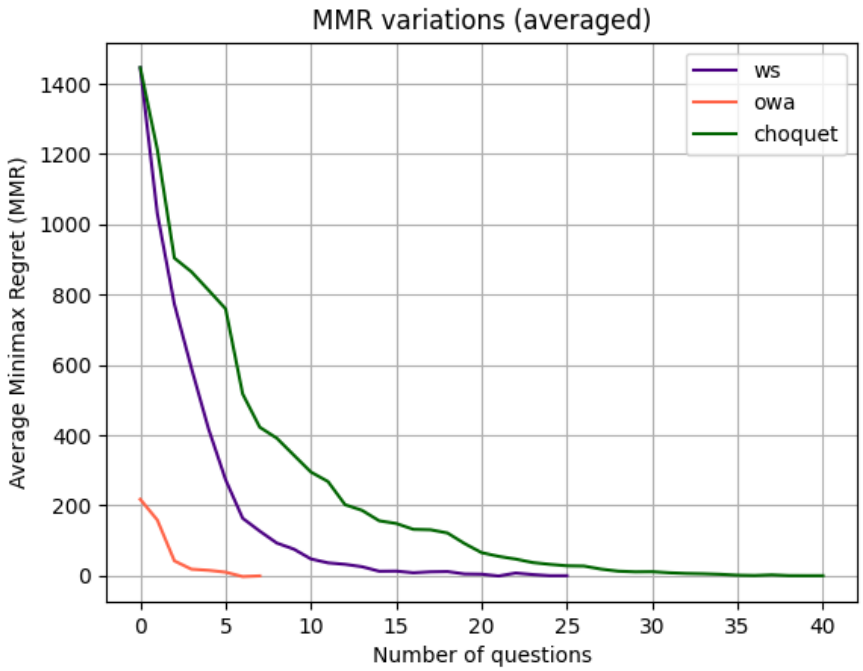

FIGURE 1.7 – Variation moyenne du MMR (d=4, n=35, $|X|$=1134)

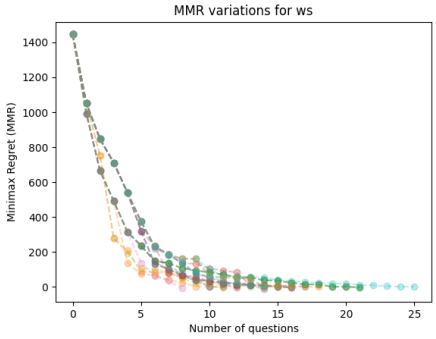

FIGURE 1.8 – Somme pondérée

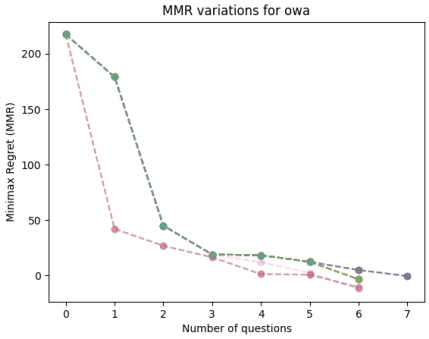

FIGURE 1.9 – OWA

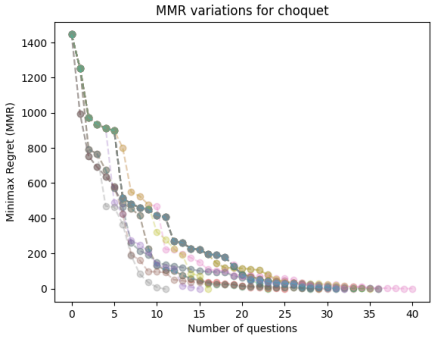

FIGURE 1.10 – Choquet

FIGURE 1.11 – Variations individuelles du MMR (d=4, n=35, $|X|$=1134)

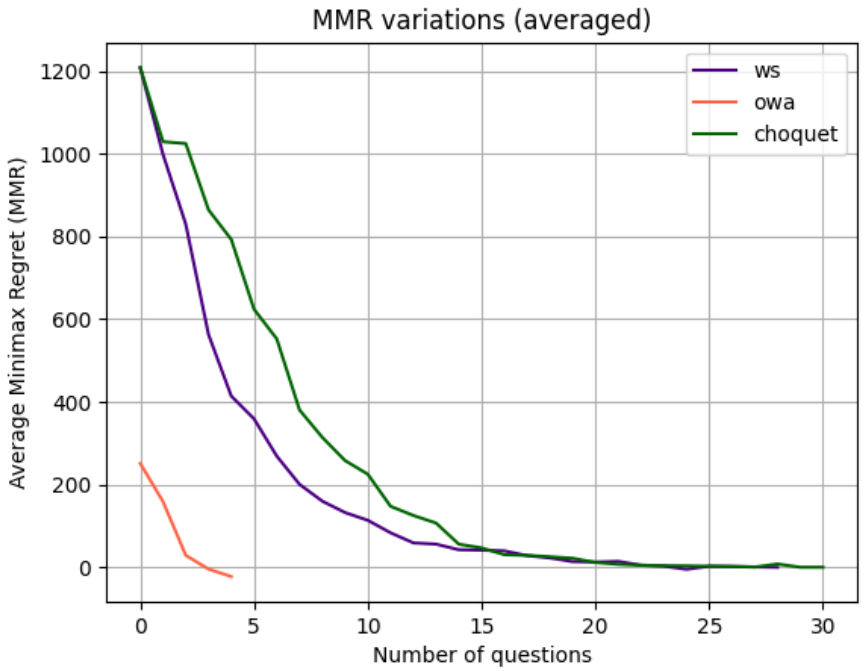

FIGURE 1.12 – Variation moyenne du MMR (d=5, n=25, $|X|$=407)

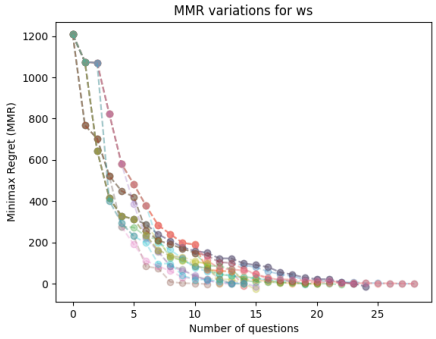

FIGURE 1.13 – Somme pondérée

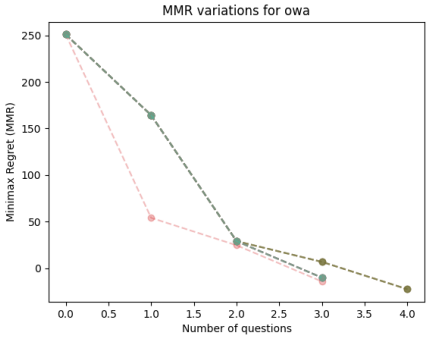

FIGURE 1.14 – OWA

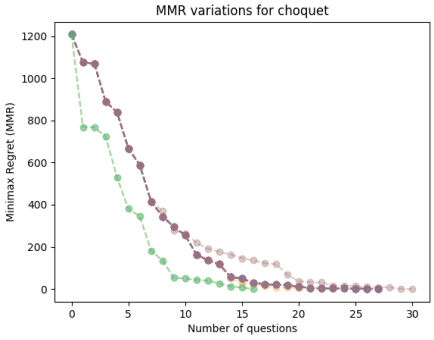

FIGURE 1.15 – Choquet

FIGURE 1.16 – Variations individuelles du MMR (d=5, n=25, $|X|$=407)

Chapitre 2

Deuxième méthode

2.1 Présentation

La deuxième méthode [3] reprend l'idée de la recherche locale, mais contrairement à la première on prendra cette fois-ci à chaque itération la solution préférée du décideur, trouvée par élicitation incrémentale.

2.2 Choix d'implémentation

Pour l'initialisation, nous utilisons une solution trouvée avec un algorithme glouton, puis génère le voisinage échange 1-1 comme dans la première méthode et en prenant que les solutions dans le front de Pareto (trouvées grâce à la structure NDTree). Sur ce front de Pareto, nous appliquerons la procédure incrémentale en utilisant les trois fonctions d'agrégations implémentées auparavant et on réitère sur une solution voisine minimisant le regret maximal jusqu'à trouver une solution (aucune amélioration locale) ou le nombre maximal d'itérations atteinte.

2.3 Analyse comparative

Pour la comparaison expérimentale des deux procédures, nous avons tester sur plusieurs dimensions avec le nombre d'objets qui varie et nous avons fait la moyenne de 20 jeux de paramètres différents pour chaque fonction d'agrégation. Contrairement à la première méthode, la deuxième méthode présente un taux d'erreur par rapport à la solution optimale, comme nous pouvons le voir sur le tableau 2.1. Nous pouvons remarquer que le taux d'erreur augmente avec la taille de l'instance pour toutes les fonctions d'agrégations. De plus, dans nos expériences, les taux d'erreurs des différents fonctions sont similaires pour une instance de taille suffisamment grande. Pour le temps d'exécution, la première méthode est plus rapide que la deuxième sur les deux premières instances plus petites (figures 2.2 et 2.3), mais sur les deux plus grandes, la deuxième surpasse la première, surtout avec Choquet (figures 2.4 et 2.5). En générale, les deux méthodes prennent plus de temps avec Choquet, puis avec la somme pondérée et elles sont plus rapide avec OWA. Nous pouvons clairement le voir sur les plus grandes instances (figure 2.6). Pour le nombre de questions, la première méthode en pose toujours moins que la deuxième (figure 2.11). Pareil que pour le temps d'exécution, en générale, les méthodes sont plus performantes avec OWA,

Instance	Φ	Erreur	$\pm\sigma$
$1 : d = 3, n = 20$	WS	11%	2.6
	OWA_m	14%	0.8
	C_{sm}	14%	1.2
$2 : d = 3, n = 40$	WS	15%	2.8
	OWA_m	17%	0.4
	C_{sm}	17%	0.6
$3 : d = 4, n = 30$	WS	17%	2.3
	OWA_m	21%	0.5
	C_{sm}	21%	0.5
$4 : d = 5, n = 20$	WS	20%	2.3
	OWA_m	20%	0.7
	C_{sm}	19%	0.9

FIGURE 2.1 – Comparaison des erreurs relatives à l'optimum pour la deuxième méthode

puis la somme pondérée et pour finir Choquet. De plus, le nombre de questions augmente avec la taille de l'instance.

2.4 Conclusion

Nous avons implémenté avec succès les deux méthodes demandées et les expériences réalisées nous ont permis de les comparer sur différentes instances et fonctions d'agrégation. La première méthode présente l'avantage d'être très proche de l'optimum, grâce à la bonne approximation du front de Pareto. De plus, elle est économe en nombre de questions posées au décideur pendant l'élicitation. En revanche, quand la taille du front de Pareto devient grande, la recherche de la prochaine question est lente et le temps par question explose. La deuxième méthode fait un compromis sur la qualité de la solution et sur le nombre de questions posées afin de réduire ce temps par question. En effet, elle peut nécessiter deux fois plus de questions pour trouver une solution avec une erreur de 20% à l'optimum, mais dix fois plus rapidement que la première méthode. Dans un contexte où le problème est de grande taille, mais que le décideur doit être sollicité le moins possible on pourra préférer la première méthode, en revanche si le décideur ne pose pas cette contrainte et que l'on accepte une solution sous optimale on pourra préférer la rapidité de la seconde méthode. Il est à noter que notre implémentation de CSS pourrait être améliorée, en codant le calcul du max regret comme un problème de recherche et faire descendre $|X|^2$ paires de solutions testées à presque seulement $|X|$.

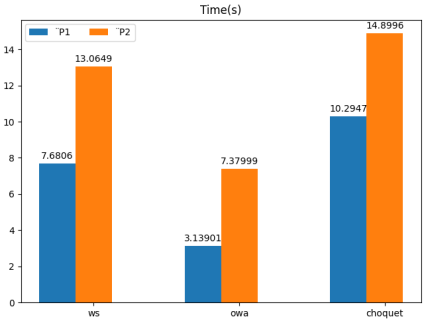

FIGURE 2.2 – Instance 1

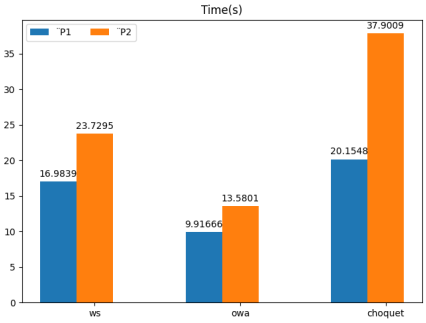

FIGURE 2.3 – Instance 2

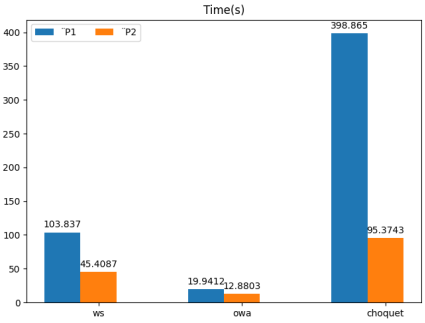

FIGURE 2.4 – Instance 3

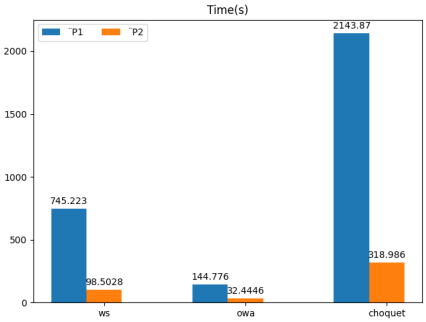

FIGURE 2.5 – Instance 4

FIGURE 2.6 – Comparaison des temps d'exécution

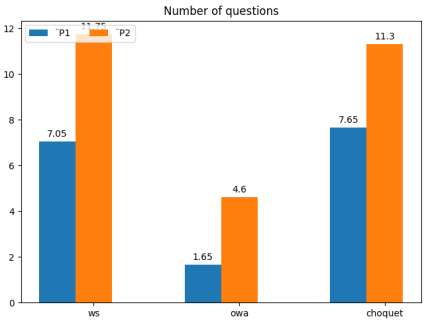

FIGURE 2.7 – Instance 1

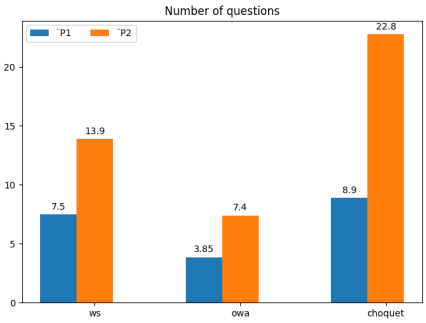

FIGURE 2.8 – Instance 2

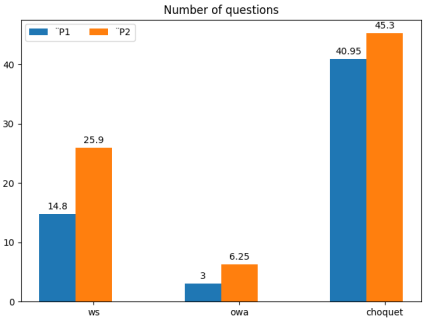

FIGURE 2.9 – Instance 3

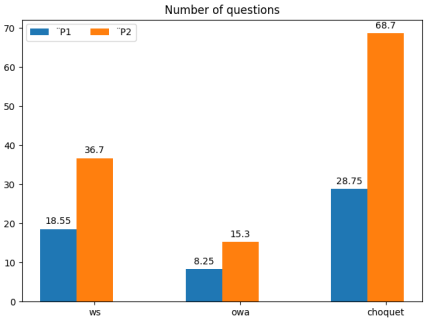

FIGURE 2.10 – Instance 4

FIGURE 2.11 – Comparaison du nombre de questions

Bibliographie

[1] Nawal Benabbou, Christophe Gonzales, Patrice Perny et Paolo Viappiani. « Minimax regret approaches for preference elicitation with rank-dependent aggregators ». In : *EURO journal on Decision processes* 3 (2015), p. 29-64 (page 1).

[2] Andrzej Jaszkiewicz et Thibaut Lust. « ND-tree-based update : a fast algorithm for the dynamic nondominance problem ». In : *IEEE Transactions on Evolutionary Computation* 22.5 (2018), p. 778-791 (page 1).

[3] Nawal Benabbou, Cassandre Leroy et Thibaut Lust. « Regret-Based Elicitation for Solving Multi-Objective Knapsack Problems with Rank-Dependent Aggregators ». In : (2020) (page 6).