RAPPORT DE PROJET:

Élicitation incrémentale et recherche locale pour le problème de sélection multi-objectifs

Réalisé par:
Madina TRAORÉ

MU5IN254 - Modèles et algorithmes pour la décision multicritère et collective

M2 Informatique - Spécialité ANDROIDE

Année universitaire 2019-2020

Sommaire

Introduction

Ce projet a été réalisé dans le cadre de l'unité d'enseignement Modèles et algorithmes pour la décision multicritère et collective. L'objectif était d'implémenter différentes procédures de résolutions afin de déterminer la solution préférée d'un Décideur pour le problème de sélection multi-objectifs que nous détaillerons dans la première section de ce rapport. La première procédure de résolution consistait à appliquer une recherche locale de Pareto pour obtenir une approximation des points non-dominés puis de déterminer la solution préférée du Décideur parmi ces points à l'aide d'une procédure d'élicitation incrémentale. La deuxième consistait à combiner élicitation incrémentale et recherche locale de Pareto afin de réduire le nombre de solutions générées et pouvoir résoudre des problèmes de plus grande taille. Dans les sections suivantes nous présenterons de manière plus détaillée les deux procédures de résolution implémentées, nous exposerons les résultats obtenus et comparerons les deux méthodes de résolution. Le code associé à ce rapport est regroupé dans deux fichiers : un Notebook **projet.ipynb** permettant de tester et visualiser les résultats de l'ensemble des méthodes implémentées et un fichier annexe python **ndtree.py** dont nous parlerons dans la quatrième section.

1. Présentation du problème

Le problème auquel nous nous intéressons est la version suivante du problème de sélection multi-objectifs :

Données :

- n objets
- un entier k
- chaque objet i est valué par un vecteur c^i à p dimensions $(c_1^i, ..., c_p^i)$

Solutions réalisables : tout sous-ensemble de k objets, caractérisé par un vecteur $x = (x_1, ..., x_n)$ ($x_i = 1$ si l'objet i est sélectionné, 0 sinon).

Objectif : Déterminer une solution réalisable x maximisant une fonction d'agrégation paramétrée représentant les préférences d'un Décideur sur les solutions.La fonction d'agrégation est de la forme $\psi_\omega(y(x)) = \psi_\omega(y_1(x), ..., y_p(x))$, avec :

- ω : paramètre initialement inconnu parmi l'ensemble Ω des paramètres initialement admissibles ($\omega \in \Omega$)
- $y(x)$: évaluation d'une solution x donnée par $(\sum_{i=1}^n c_i^1 x_i, ..., \sum_{i=1}^n c_i^p x_i)$.

On utilisera comme fonction d'agrégation une fonction linéaire en ses paramètres (somme pondérée, OWA ou intégrale de Choquet par exemple).

2. Première procédure de résolution

La première procédure de résolution est une procédure en deux temps. Premièrement, nous déterminons une approximation des points Pareto non-dominés en appliquant une recherche locale de Pareto (*PLS* ou *Pareto Local Search* en anglais). Dans un second temps, nous déterminons la solution préférée du Décideur parmi les points non-dominés trouvés par PLS à l'aide d'une procédure d'élicitation incrémentale.

Implémentation de la recherche locale de Pareto[1]

Dans notre méthode **PLS()**, nous commençons par générer aléatoirement une solution admissible x à l'aide de la méthode **generate_random_solution()**. Nous générons ensuite tous les voisins de x avec la méthode **generate_neighbours()** qui utilise comme fonction de voisinage un échange 1-1 : un objet est retiré de la sélection, un nouvel objet est ajouté à la sélection. Nous déterminons les solutions non-dominées parmi x et ses voisins à l'aide de la méthode **non_dominated_solutions()**. Tant que nous trouvons de nouvelles solutions non-dominées (i.e. tant que nous n'avons pas atteint un optimum local), nous déterminons les voisins de chaque nouvelle solution non-dominée trouvée puis nous regardons s'il existe de nouvelles solutions non-dominées parmi celles-ci.

[1] Nous nous sommes appuyés sur le pseudo-code de la figure 1 pour l'implémenter.

Algorithme 1 `Recherche Locale Pareto`

Paramètres $\downarrow$: une population initiale P_0, une fonction de voisinage $\mathcal{N}(x)$.
Paramètres $\uparrow$: une approximation $\widetilde{X}_E$ de l'ensemble efficace X_E.

--| Initialisation de $\widetilde{X}_E$ et d'une population P avec la population initiale P_0
$\widetilde{X}_E \leftarrow P_0$
$P \leftarrow P_0$
--| Initialisation d'une population auxiliaire P_a
$P_a \leftarrow \varnothing$
while $P \neq \varnothing$ **do**
    --| Génération de tous les voisins p' de chaque solution $p \in P$
    for all $p \in P$ **do**
        for all $p' \in \mathcal{N}(p)$ **do**
            if $f(p) \not\preceq_P f(p')$ **then**
                if `Update`$(\widetilde{X}_E \updownarrow, p' \downarrow)$ **then**
                `UPdate`$(P_a \updownarrow, p' \downarrow)$
    --| P est composé des nouvelles solutions potentiellement efficaces
    $P \leftarrow P_a$
    --| Ré-initialisation de P_a
    $P_a \leftarrow \varnothing$

Figure 1: Pseudo-code de l'algorithme de recherche locale de Pareto

Implémentation de l'élicitation incrémentale

Le principe de l'élicitation incrémentale est le suivant :

- ▷ On dispose d'un ensemble $\mathcal{X}$ de solutions
- ▷ On cherche à déterminer les coefficients d'un vecteur de poids ω qui devra représenter au mieux les préférences du Décideur
- ▷ On pose des questions du type "Quelle solution préférez-vous entre x_1 et x_2 ?" ($x_1 \succcurlyeq x_2$?) au Décideur
- ▷ S'il répond qu'il préfère x_1 à x_2 ($x_1 \succcurlyeq x_2$), on restreint l'ensemble Ω de poids possibles aux poids ω tels que $\psi_\omega(y(x_1)) \geq \psi_\omega(y(x_2))$ avec ψ une fonction d'aggrégation linéaire en ses paramètres telle que la somme pondérée, OWA ou l'intégrale de Choquet

En pratique, nous procédons de la manière suivante (méthode **incremental_elicitation()**) :

1. On fixe le regret minimax à $+\infty$
2. On calcule les PMR (*Pairwise Max Regrets*) à l'aide de Gurobi pour chaque couple de solutions. Cela nous permet de calculer les regrets maximaux (MR) puis les regrets minimax (MMR).
3. Afin de déterminer une bonne question à poser au Décideur (une question qui nous apportera beaucoup d'information sur ses préférences) nous utilisons la stratégie CSS (*Current Solution Strategy*). Elle consiste à comparer une solution x_p^* choisie arbitrairement dans $\underset{x \in \mathcal{X}}{\operatorname{argmin}} MR(x, \mathcal{X})$ (solution minimisant le regret maximal) à une solution y_p^* choisie au hasard dans $\underset{y \in \mathcal{X}}{\operatorname{argmax}} PMR(x_p^*, y)$ (solution maximisant le regret

paire à paire). De cette manière, le regret minimax diminuera à chaque itération et on se rapprochera alors progressivement du *mmr_threshold*.

4. On simule la réponse du Décideur à cette question en calculant $\psi_\omega(y(x_p^*))$ et $\psi_\omega(y(y_p^*))$, ω étant un vecteur de poids généré aléatoirement et représentant ses préférences
5. Si $\psi_\omega(y(x_p^*)) \geq \psi_\omega(y(y_p^*))$ (i.e. $x_p^* \succcurlyeq y_p^*$) alors on ajoute cette contrainte au modèle Gurobi permettant de calculer les PMR. On restreint ainsi l'ensemble Ω de poids possibles aux poids ω tels que $\psi_\omega(y(x_p^*)) \geq \psi_\omega(y(y_p^*))$ et les calculs de la prochaine itération prendront en compte cette nouvelle information.
6. Si le regret minimax est supérieur au seuil *mmr_threshold*, on recommence à partir de l'étape 2. Sinon on renvoie la solution trouvée (solution minimisant le regret minimax).

3. Deuxième procédure de résolution

La deuxième procédure de résolution appelée Recherche Locale Interactive (ou *Interactive Local Search* en anglais) est une procédure combinant élicitation incrémentale et recherche locale de Pareto.

Implémentation de la Recherche Locale Interactive[2]

La méthode **interactive_local_search()** se décompose de la manière suivante :

1. On génère aléatoirement une solution admissible x à l'aide de la méthode **generate_random_solution()**
2. On génère l'ensemble N des voisins de x avec la méthode **generate_neighbours()**
3. On détermine la meilleure solution $x^* \in N \cup \{x\}$ à l'aide de la méthode **incremental_elicitation()** décrite dans la section précédente et on garde en mémoire les solutions dominées déjà rencontrées pour ne pas les traiter plusieurs fois. Si $x^* \in N$, on recommence à partir de l'étape 2 en générant cette fois-ci les voisins de x^*. Sinon, la procédure s'arrête car un optimum local (x^*) a été trouvé

[2]Nous nous sommes appuyés sur le pseudo-code de la figure 2 pour l'implémenter.

Algorithm 1: ILS

IN $\downarrow$ P: a MOCO problem; δ_1, δ_2: thresholds; f_ω: an aggregator with unknown parameters; Θ: a set of preference statements; m: number of initial solutions.

--| Initialization of the admissible parameters:
$\Omega_\Theta \leftarrow \{\omega : \forall (a,b) \in \Theta, f_\omega(a) \leq f_\omega(b)\}$
--| Generation of m initial solutions:
$X_0 \leftarrow$ `Select&Optimize`(P, Ω_Θ, m)
--| Determination of the starting solution:
while $MMR(X_0, \Omega_\Theta) > \delta_1$ **do**
  --| Ask the DM to compare two solutions in X_0:
  $(x, x') \leftarrow$ `Query`(X_0)
  --| Update preference information:
  $\Theta \leftarrow \Theta \cup \{(y(x), y(x'))\}$
  $\Omega_\Theta \leftarrow \{\omega : \forall (a,b) \in \Theta, f_\omega(a) \leq f_\omega(b)\}$
end while
$x^* \leftarrow$ `Select`$(\arg\min_{x \in X_0} MR(x, X_0, \Omega_\Theta))$
--| Interactive Local Search:
improve $\leftarrow$ **true**
while improve **do**
  $X^* \leftarrow$ `Neighbors`$(P, x^*) \cup \{x^*\}$
  while $MMR(X^*, \Omega_\Theta) > \delta_2$ **do**
    --| Ask the DM to compare two solutions in X^*:
    $(x, x') \leftarrow$ `Query`(X^*)
    --| Update preference information:
    $\Theta \leftarrow \Theta \cup \{(y(x), y(x'))\}$
    $\Omega_\Theta \leftarrow \{\omega : \forall (a,b) \in \Theta, f_\omega(a) \leq f_\omega(b)\}$
  end while
  --| Move to another solution:
  if $MR(x^*, X^*, \Omega_\Theta) > \delta_2$ **then**
    $x^* \leftarrow$ `Select`$(\arg\min_{x \in X^*} MR(x, X^*, \Omega_\Theta))$
  else
    improve $\leftarrow$ **false**
  end if
end while
return x^*

Figure 2: Pseudo-code de l'algorithme Interactive Local Search

4. Utilisation d'un ND-Tree

La recherche locale de Pareto nécessitant de nombreuses mises à jour d'ensembles de solutions non-dominées, nous avons implémenté une version utilisant un ND-Tree pour chacune des méthodes concernées (**PLS_nd_tree** et **interactive_local_search_nd_tree**) . Le code contenant l'implémentation de la structure ND-Tree se trouve dans le fichier **ndtree.py** et s'appuie sur l'article *ND-Tree-based update : a Fast Algorithm for the Dynamic Non-Dominance Problem* de Thibaut Lust et Andrzej Jaszkiewicz ([3]).

Le ND-Tree maintient à jour une archive Y_N de points Pareto non-dominés de la manière suivante :

Lorsque l'on ajoute un point y au ND-Tree, on commence par regarder s'il est dominé ou non

par un des points de Y_N en parcourant les noeuds n de l'arbre tels que y soit comparable à l'approximation du point idéal et/ou l'approximation du point nadir associées à n. Si y est dominé par l'approximation du point nadir associé à n, il est rejeté. S'il domine l'approximation du point idéal associé à n, ce sont n et tout son sous-arbre qui sont rejetés. Sinon, cela signifie que y est situé dans le rectangle dont le point inférieur gauche est l'approximation du point nadir et le point supérieur droit est l'approximation du point idéal. Dans ce cas, si n est un noeud interne, l'algorithme de mise à jour est appelé de manière récursive pour chacun de ses enfants. Si c'est une feuille, il est possible que y domine ou soit dominé par des points de n et on doit donc le comparer à l'ensemble des points contenus dans n. Si y est dominé par un de ces points, il est rejeté et si y domine un de ces points, ce point est retiré de n. S'il s'avère que y est Pareto non-dominé, il est inséré à la feuille de l'arbre la plus proche de lui en terme de distance euclidienne au segment reliant les approximations des points nadir et idéal de la feuille. Dans ce cas, on met à jour les valeurs des approximations des points nadir et idéal de la feuille à laquelle y vient d'être ajouté. Si le nombre de points stockés dans la feuille devient supérieur à la valeur maximale de points pouvant être stockés dans un noeud (valeur définie au préalable), on sépare le noeud en un nombre prédéfini de noeuds qui deviendront alors ses enfants. On fait en sorte que chaque nouveau noeud contienne des points qui se ressemblent entre eux plus qu'ils ne ressemblent aux autres points grâce à un calcul de barycentre.

5. Étude et comparaison des procédures de résolution

Les tets numériques effectués dans cette section ont tous été réalisés sur 20 jeux de paramètres différents en utilisant comme fonction d'agrégation une somme pondérée.

Première procédure de résolution : évolution du regret minimax en fonction du nombre de questions posées

La stratégie CSS utilisée pour déterminer la question posée au Décideur décrite dans la section 2 nous assure que les regrets minimax (MMR) seront toujours décroissants. Comme nous pouvons le voir dans la figure 3, nous observons bien ce résultat en pratique :

```
Nombre de questions posées : 0
MMR = 45.0

Nombre de questions posées : 1
MMR = 2.6296296296296298

Nombre de questions posées : 2
MMR = 0.0

Vecteur représentant les préférences du Décideur : [0.39219054630517103, 0.6078094536948291]

Solution trouvée par élicitation incrémentale: [0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1]

Valeur de cette solution : [585, 809]

Nombre de questions posées au Décideur : 2
```

Figure 3: Résultat d'une exécution de la méthode **incremental_elicitation()** pour $n = 20$, $p = 2$

Pourcentage de points non-dominés trouvés par PLS et efficacité du ND-Tree implémenté

Résultats pour **PLS()** (obtenus avec la méthode **quality_measure_PLS()**) pour différents jeux de paramètres :

```
Paramètres : n = 10, p = 2
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.016335487365722656 s

Paramètres : n = 10, p = 3
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.13049626350402832 s

Paramètres : n = 10, p = 4
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.4041154384613037 s

Paramètres : n = 10, p = 5
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :1.5451209545135498 s

Paramètres : n = 10, p = 6
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :1.9071440696716309 s

Paramètres : n = 20, p = 2
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :2.2120144367218018 s

Paramètres : n = 20, p = 3
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :19.176265716552734 s
```

Résultats pour **PLS_nd_tree()** (obtenus avec la méthode **quality_measure_PLS_nd_tree()**) pour différents jeux de paramètres :

```
Paramètres : n = 10, p = 2
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.006983518600463867s

Paramètres : n = 10, p = 3
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.03470325469970703s

Paramètres : n = 10, p = 4
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.06851077079772949s

Paramètres : n = 10, p = 5
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.4919319152832031s

Paramètres : n = 10, p = 6
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.5048227310180664s

Paramètres : n = 20, p = 2
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :0.5822663307189941s

Paramètres : n = 20, p = 3
Pourcentage de points non-dominés obtenus par PLS : 100.0 %
Temps de calcul :8.417462348937988s
```

On remarque que, pour tous les jeux de paramètres testés, on trouve l'ensemble des points Pareto non-dominés. Par ailleurs, le ND-Tree permet d'avoir une PLS au moins deux fois plus efficace en terme de temps de calcul.

Comparaison des deux procédures en terme de temps de calcul, d'erreur par rapport à la solution optimale du Décideur et de nombre de questions posées

Résultats pour la première procédure de résolution (obtenus avec la méthode **quality_measure_procedure_1_nd_tree()**) pour différents jeux de paramètres :

```
Paramètres : n = 10, p = 2
Temps moyen de calcul :0.012340712547302245s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 1.0

Paramètres : n = 10, p = 3
Temps moyen de calcul :0.09794524908065796s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 3.4

Paramètres : n = 10, p = 4
Temps moyen de calcul :0.5909860253334045s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 7.2

Paramètres : n = 10, p = 5
Temps moyen de calcul :4.81545969247818s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 9.85

Paramètres : n = 20, p = 2
Temps moyen de calcul :0.6386162996292114s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 3.35
```

On ne parvient pas à traiter de plus grosses instances que l'instance $n = 20$, $p = 2$ en un temps raisonnable avec la première procédure.

Résultats pour la deuxième procédure de résolution (obtenus avec la méthode **quality_measure_procedure_2_nd_tree()**) pour différents jeux de paramètres :

```
Paramètres : n = 10, p = 2
Temps moyen de calcul :0.010389697551727296s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 2.1

Paramètres : n = 10, p = 3
Temps moyen de calcul :0.043032991886138925
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 6.25

Paramètres : n = 10, p = 4
Temps moyen de calcul :0.25358172655105593s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 11.35

Paramètres : n = 10, p = 5
Temps moyen de calcul :1.2767961025238037s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 18.95

Paramètres : n = 10, p = 6
Temps moyen de calcul :2.108030140399933s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 23.45

Paramètres : n = 20, p = 2
Temps moyen de calcul :0.06903607845306396s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 6.75

Paramètres : n = 20, p = 3
Temps moyen de calcul :0.5141715764999389s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 13.6

Paramètres : n = 30, p = 2
Temps moyen de calcul :0.21743197441101075s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 7.6

Paramètres : n = 40, p = 2
Temps moyen de calcul :0.5756325125694275s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 7.1

Paramètres : n = 50, p = 2
Temps moyen de calcul :1.4290114045143127s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 9.95

Paramètres : n = 60, p = 2
Temps moyen de calcul :2.9151734828948976s
Erreur moyenne par rapport à la solution optimale du décideur : 0.0
Nombre moyen de questions posées : 9.85
```

Avec la deuxième procédure de résolution, on parvient donc à traiter des instances beaucoup plus grosses en un temps raisonnable. Néanmoins, on constate que cette procédure nécessite de poser davantage de questions au Décideur que la première. En situation réelle, cela pourrait déranger le Décideur qu'autant de questions lui soit posées même si cela permet de lui recommander sa solution préférée.

Références

[1] Nawal Benabbou, Christophe Gonzales, Patrice Perny, Paolo Viappiani. *Minimax Regret Approachesfor Preference Elicitation with Rank-Dependent Aggregators*. EURO Journal on Decision Processes, 2015, 3 (1-2), pp.29-64.

[2] Nawal Benabbou, Cassandre Leroy, Thibaut Lust, Patrice Perny. *Combining Local Search and Elicitation for Multi-Objective Combinatorial Optimization*. ADT 2019 - 6th International Conference onAlgorithmic Decision Theory, Oct 2019, Durham, NC, United States. hal-02170910

[3] Andrzej Jaszkiewicz, Thibaut Lust. *ND-Tree-based update : a Fast Algorithm for the Dynamic Non-Dominance Problem*. 2018. hal-01900840