TME : Recherche locale Pareto

Durant ce TME, vous allez mettre en œuvre une recherche locale Pareto pour générer une approximation de bonne qualité de l'ensemble des solutions efficaces du problème du sac à dos bi-objectifs.
La formulation du problème du sac à dos bi-objectifs est la suivante :

$$\max f(x) = \left(\sum_{i=1}^{n} v_i^1 x_i, \sum_{i=1}^{n} v_i^2 x_i \right)$$

$$\begin{cases} \sum_{i=1}^{n} w_i x_i & \leq W \\ x_i \in \{0,1\} & \forall i \in \{1,\dots,n\} \end{cases}$$

Avec :

- n : nombre d'objets
- $W > 0$: capacité du sac à dos
- $w_i > 0$: poids d'un objet i
- $v_i^1 > 0$: profit selon l'objectif 1 d'un objet i
- $v_i^2 > 0$: profit selon l'objectif 2 d'un objet i

Une solution est représentée par un vecteur $x = (x_1, \dots, x_n)$ de variables de décision x_i telles que $x_i = 1$ si l'objet i est inclus dans la solution et 0 sinon. Une solution réalisable doit satisfaire la contrainte de capacité du sac à dos, c'est-à-dire $\sum_{i=1}^{n} w_i x_i \leq W$. La valeur d'une solution réalisable x est donnée par un vecteur à deux dimensions donnant la valeur des deux objectifs : $\left(\sum_{i=1}^{n} v_i^1 x_i, \sum_{i=1}^{n} v_i^2 x_i \right)$.
Pour n'importe quelle instance de ce problème, le but est de générer une bonne approximation de l'ensemble des solutions efficaces.

Le langage de programmation utilisé est libre, mais quelques fonctions, codées en Python, vous sont données (disponible sur Moodle, vous n'êtes pas obligés de les utiliser). A la fin du TME, vous soumettrez votre code et un fichier contenant un résumé des résultats obtenus.

Exercice 1 – Lecture des instances

Vous trouverez sur Moodle différentes instances, de taille variant de 100 objets à 700 objets. Les données des instances sont contenues dans les fichiers d'extension .dat. Dans ces fichiers, pour chaque objet est donné son poids w, son profit v_1 et son profit v_2. La capacité W du sac à dos est donnée en fin de fichier. Vous trouverez également pour chaque instance l'ensemble Y_N des points non-dominés (fichiers d'extension .eff), généré avec une méthode exacte. Par exemple l'instance 2KP100-TA-0, de taille 100, comprend 125 points non-dominés. Ces ensembles vous seront utiles pour mesurer la qualité des approximations que vous allez générer avec la recherche locale Pareto.

Question 1 — Coder une fonction de lecture des instances et une fonction permettant de lire les ensembles de points non-dominés (voir `read_file.py`).

Question 2 — Coder une fonction permettant de représenter les ensembles de points non-dominées dans l'espace des objectifs (voir `main.py`).

Exercice 2 – Indicateurs de qualité

Pour mesurer la qualité d'une approximation des solutions efficaces, vous allez utiliser deux indicateurs de qualité :

- Proportion P_{Y_N} de points non-dominés générés : $P_{Y_N} = \frac{|\widetilde{Y}_N \cap Y_N|}{|Y_N|}$, avec Y_N l'ensemble des points non-dominés et $\widetilde{Y}_N$ une approximation des points non-dominés.
- Indicateur de distance moyenne D_M : cet indicateur mesure la distance moyenne entre les points non-dominés et les points de l'approximation. Si on considère n'importe quelle mesure de distance $d(y^1, y^2)$ entre deux points y^1 et y^2 dans l'espace des objectifs, on peut définir la distance $d'(\widetilde{Y}_N, y^1)$ entre un point $y^1 \in Y_N$ et les points de $\widetilde{Y}_N$ de la façon suivante : $d'(\widetilde{Y}_N, y^1) = \min_{y^2 \in \widetilde{Y}_N} d(y^2, y^1)$. Cette distance est donc égale à la distance minimale entre un point $y^1 \in Y_N$ et les points de l'approximation $\widetilde{Y}_N$. La distance moyenne D_M entre $\widetilde{Y}_N$ et Y_N est ensuite définie de la façon suivante :

  $$D_M(\widetilde{Y}_N, Y_N) = \frac{1}{|Y_N|} \sum_{y^1 \in Y_N} d'(\widetilde{Y}_N, y^1)$$

  Pour mesure de distance, vous utiliserez la distance Euclidienne pondérée :

  $$d(y^1, y^2) = \sqrt{p_1(y^1_1 - y^2_1)^2 + p_2(y^1_2 - y^2_2)^2}$$

  avec p_1 et p_2 des poids permettant de prendre en compte l'étendue des valeurs des objectifs : $p_k = \frac{1}{y^I_k - y^N_k}, k \in \{1, 2\}$, avec y^N le point Nadir et y^I le point idéal.

Vous mesurerez également le temps de résolution (CPU en secondes).
Pour chaque instance, l'algorithme de résolution sera exécuté 10 fois et vous calculerez la moyenne des différentes indicateurs (P_{Y_N}, D_M et Temps) sur ces 10 exécutions.
N'hésitez pas également à représenter les ensembles de points potiellement non-dominés dans l'espace des objectifs, obtenus par votre méthode, et de les comparer avec les fronts de Pareto (exacts).

Question 3 — Coder les fonctions permettant de calculer les valeurs des indicateurs P_{Y_N} et D_M d'une approximation $\widetilde{Y}_N$ (voir `indicators.py`).

Exercice 3 – Génération aléatoire

Avant de programmer la méthode PLS, et afin de se familiariser avec le problème, vous allez coder dans un premier temps une méthode générant m solutions générées aléatoirement (les objets mis dans le sac sont choisis au hasard jusqu'à ce qu'il ne soit plus possible d'en ajouter). Les solutions trouvées sont filtrées au fur et à mesure afin de ne garder que les solutions non-dominées. La procédure de mise à jour (`MiseAJour`(X, x)) de la liste des solutions mutuellement non-dominées sera très simple : un parcours complet de la liste X sera réalisé afin de déterminer si la solution x doit être ajoutée ou pas. De plus, si la solution x est ajoutée, toutes les solutions de X dominées par x seront retirées de la liste. Cette procédure renverra un booléan égal à Vrai si la solution a été ajoutée dans la liste.
Calculer les indicateurs de qualité de l'approximation obtenue. Représenter également les points non-dominés obtenus et comparer-les au front de Pareto (voir `main.py` où ils vous restent à code la fonction de mise à jour).

Exercice 4 – Recherche locale Pareto (PLS1)

Vous allez maintenant coder la recherche locale Pareto telle que vue au cours, dont le pseudo-code est rappelé ci-dessous.

Algorithm 1 Recherche Locale Pareto (PLS)

Paramètres $\downarrow$: une population initiale P_0, une fonction de voisinage $V(x)$.
Paramètres $\uparrow$: une approximation $\widetilde{X}_E$ de l'ensemble des solutions efficaces 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 $p' \in V(p)$ **do**
      --| si p' n'est pas dominé par p
      if $f(p) \not\preceq f(p')$ **then**
        if `MiseAJour`$(\widetilde{X}_E \updownarrow, p' \downarrow)$ **then**
          `MiseAJour`$(P_a \updownarrow, p' \downarrow)$
  --| P est composée des nouvelles solutions potentiellement efficaces qui viennent d'être trouvées
  $P \leftarrow P_a$
  --| Réinitialisation de P_a
  $P_a \leftarrow \varnothing$

Pour appliquer PLS au problème du sac à dos bi-objectifs, vous allez avoir besoin de définir trois éléments :

1. Une population initiale constituée de solutions réalisables mutuellement non dominées.
2. Une fonction de voisinage $V(x)$ permettant de générer un ensemble de solutions réalisables proches d'une solution x.
3. Une procédure de mise à jour d'un ensemble de solutions mutuellement non dominées.

La première version de PLS (nommée PLS1) que vous allez implémenter sera très simple :

1. La population initiale sera constituée de plusieurs solutions réalisables obtenues à l'aide d'un simple algorithme glouton. Pour cela on utilise un rapport de performance $R(i)$ pour un objet i, égal à :
$$\frac{q_1 v_i^1 + (1-q_1) v_i^2}{w_i}$$
avec q_1 un poids (compris dans $[0,1]$) permettant de pondérer l'importance donnée au premier et deuxième profit de l'objet. Une solution sera obtenue en sélectionnant les objets itérativement par ordre décroissant du rapport de performance. Cette opération sera répétée m fois, pour m valeurs différentes de q_1. Vous pourrez utiliser une valeur de m égale à 100.
2. La fonction de voisinage $V(x)$ consiste à retirer un objet de la solution courante x et à ajouter un nouvel objet (échange 1-1). Toutes les combinaisons 1-1 seront testées. Attention tous les échanges 1-1 ne sont pas forcément réalisables (à cause de la contrainte de capacité).

3. La procédure de mise à jour (`MiseAJour`(X, x)) de la liste des solutions mutuellement non-dominées utilisera un simple parcours de la liste X.

Question 4 — Coder cette première version de PLS. Vérifier bien la convergence de la méthode à travers la taille de la population P : celle-ci doit rapidement augmenter, puis lentement diminuer. La méthode s'arrête lorsque la fonction de voisinage ne permet plus de trouver de nouvelles solutions non-dominées.
Donner dans un tableau les différents résultats obtenus pour les différents indicateurs de qualité par cette méthode.

Exercice 5 – Amélioration de la fonction de mise à jour (PLS2)

Comme vu au cours, il est possible de facilement améliorer la procédure de mise à jour d'une liste de solutions mutuellement non-dominées. En effet, avec deux objectifs, le tri des valeurs selon un des objectifs maintient le tri des valeurs selon le deuxième objectif. Utiliser cette propriété pour améliorer la procédure de mise à jour. Attention, cela ne doit pas améliorer les indicateurs de qualité de PLS1, mais uniquement réduire fortement les temps de calcul.

Question 5 — Coder cette deuxième version de PLS (PLS2). Noter dans un tableau les différents résultats obtenus pour les différents indicateurs de qualité par cette méthode.

Exercice 6 – Amélioration de la fonction de voisinage (PLS3)

Une dernière façon d'améliorer la méthode PLS est d'améliorer la fonction de voisinage. En effet dans la fonction de voisinage initiale uniquement un seul objet est retiré. On pourrait donc envisager d'en retirer plusieurs, mais le nombre de combinaisons possibles augmenterait considérablement. Une façon simple d'effectuer des échanges plus conséquents d'objets est de procéder de la façon suivante :

1. Une liste L_1, de taille L, d'objets candidats à être retirés (donc des objets présents dans la solution courante x) est formée, à l'aide du rapport de performance R (les objets les moins bons pour le rapport R sont mis dans L_1). Un poids q_1 généré aléatoirement en début de procédure de voisinage sera pris en compte pour le calcul du rapport R.
2. Une liste L_2, de taille L également, d'objets candidats à être ajoutés (donc des objets absents de la solution courante x) est formée, à l'aide du rapport de performance R (les meilleurs objets pour le rapport R sont mis dans L_2).
3. Les deux listes L_1 et L_2 sont ensuite fusionnées et les objets présents dans cette liste forment une nouvelle instance du sac à dos bi-objectifs, avec comme nouvelle contrainte de capacité une valeur égale à $W - \sum_{i=1, i \notin L1}^{n} w_i x_i$. Si L reste petit, cette instance est également de petite taille (taille $2*L$) et une méthode exacte peut être utilisée pour sa résolution. Vous pouvez utiliser une simple méthode d'énumération complète des solutions réalisables ou alors une méthode basée sur la programmation dynamique. Il est également possible d'utiliser une valeur plus grande pour L, mais dans ce cas, pour limiter les temps de calcul, il sera nécessaire de développer une méthode heuristique pour la résolution de l'instance issue du voisinage (qui pourrait être une version simple de PLS).
4. Les solutions efficaces (ou potentiellement efficaces) de cette mini-instance sont ensuite combinées avec la solution courante x afin d'obtenir l'ensemble des voisins.

Question 6 — Coder ce nouveau voisinage et intégrer le dans la dernière version de PLS, afin d'obtenir une nouvelle version nommée PLS4.

Exercice 7 – Résulats finaux

Vous présenterez les résultats obtenus sous forme d'un tableau présentant les résultats obtenus (temps de résolution, indicateur D_M et pourcentage P_{Y_N} de points non-dominées trouvés) par les versions de l'adaptation de PLS (PLS1, PLS2, PLS3, PLS4) pour différentes tailles d'instances du problème du sac à dos bi-objectifs.