Thien-Kim Loisel-Luu 1834378
Souhaila Mellouk 1835144
Mourad Younes 1832387

Q1 - Résolution locale

Pour l'exercice concernant la recherche locale, nous avons passé beaucoup du temps à réfléchir pour bien choisir notre fonction d'évaluation et notre fonction heuristique, soit la fonction de voisinage, la fonction de validité, la fonction de sélection ainsi que le critère d'arrêt. Après plusieurs essais, nous avons trouvé une solution qui nous paraît assez bien.

Nous avons tout d'abord décider de commencer l'algorithme avec un état aléatoire. Il s'agit d'une procédure assez simple qui est souvent efficace.

Pour la **fonction de voisinage**, nous avons décidé de récupérer tous les états ayant un seul changement. C'est-à-dire où il n'y a qu'une seule machine qui change de générateur à la fois. La simple raison est le fait que changer plusieurs machines à la fois était plus coûteux à calculer et plus difficile à définir.

Ensuite, pour la **fonction d'évaluation**, nous avons décidé de choisir le nombre de machine reliée au générateur le plus coûteux. En effet, on a remarqué que l'activation d'un générateur était beaucoup plus coûteux que le coût variable qui correspond à la distance euclidienne. Donc Il fallait nécessairement trouver un moyen d'enlever les générateurs les plus coûteux et garder ceux les moins coûteux.

Pour la **fonction de validité**, nous avons simplement choisi les voisins où le nombre de machines reliées au générateur le plus coûteux était inférieur à l'état actuel.

Enfin, pour la **fonction de sélection**, nous avons choisi le voisin pour lequel le coût total était le plus petit (donc le meilleur voisin). En répétant plusieurs fois ces fonctions, on remarque que petit à petit on désactive les générateurs les plus coûteux et le coût total diminue considérablement.

Il faut noter qu'on a choisi un **critère d'arrêt** égale à 5 fois le nombre de machine pour essayer d'avoir le meilleur résultat possible. Après plusieurs tentatives, nous avons remarqué que l'algorithme était parfois coincé entre deux états et restait coincé dans un minimum local. C'est pour cette raison que nous avons utilisé le **Tabu search**, qui est une liste qui retient chaque générateur qu'on désactive. De cette façon, à la fin de notre algorithme, nous obtenons un coût beaucoup plus petit que le coût de l'algorithme naïve, puisqu'on finit par enlever tous les générateurs en ne laissant que celui ayant le coût le moins élevé.