Nicolas Veber – 2036057 / Aladji Warren – 2035541 / Louis Sérey – 1846142

Résolution par recherche locale

Nous avons opté pour ce problème à un algorithme génétique. Voici le pseudocode :

```
N = 50
M = 100
Création d'une population aléatoire initiale de taille N
Tant que M générations n'ont pas eu lieu :
        Isoler la meilleur moitié de la génération précédente
        Construire la nouvelle génération avec :
                La meilleur solution
                N*60% de solutions issues de croisements
                N*30% de solutions issues de mutations
                Le reste avec des nouvelles solutions aléatoire
Retourner la meilleur solution
```

Pour la création d'une population aléatoire initiale, nous allumons aléatoirement les générateurs pour une solution donnée. Il faut faire attention à vérifier qu'au moins 1 générateur est allumé. Nous avons décidé pour ce lab de prendre une population de 50 solutions.

Ensuite nous avons décidé aussi de faire rouler l'algorithme sur 100 générations. On a remarqué que pour 25 générateurs et 100 machines, l'algorithme obtenait sa meilleur solution en moyenne vers la 11-ème génération. On a mis 100 générations pour avoir une marge, au cas où l'aléatoire de notre algorithme n'est pas en notre faveur. Avec un temps d'exécution total en moyenne inférieur à 6 secondes, on a trouvé ça bien.

On a décidé de garder les 25 meilleurs solutions, soit 50% de la population, pour servir de base à la génération futur. Les solutions sont triées par ordre croissant en cout.

Ensuite pour retrouver une population de 50 solutions, nous gardons la meilleur solution. Puis nous ajoutons 30 (soit N*60%) solutions issues de croisement. Ensuite nous ajoutons 15 (soit N*30%) solutions issues de mutation. Enfin nous complétons avec de nouvelles solutions générées aléatoirement.

Pour les croisement, nous sélectionnons 2 solutions parents selon une distribution betavariate. On a fait plusieurs tests avec les paramètres de cette distribution et on a trouvé que alpha = 1 et beta = 0.3 fonctionne bien pour nos besoin. On a ainsi beaucoup de chance de choisir une bonne solution mais quand même une chance non nulle de choisir un solution « moyenne ». Le croisement consiste à prendre une section du parent 2 et de la swaper avec la même section chez le parent 1. En pratique, on a : enfant = parent1[0 : borne_inf] + parent2[borne_inf : borne_sup] + parent1[borne_sup : self.n_generator]

Pour la mutation, on choisit quelle solution va être mutée avec la même distribution betavariate. Les mutations consistent à inverser l'état d'un générateur aléatoire dans la solution. La solution choisie a 60% de chance de subir 1 mutation, 30% d'en subir 2 et 10% d'en subir 3 d'un coup.

Enfin, on retourne la meilleur solution

Source :
Youtube, Tutorial : Introduction to Genetic Algorithm n application on Traveling Sales Man Problem (TSP), https://www.youtube.com/watch?v=3GAfjE_ChRI&t=561s