Rapport du porjet d'Optimisation
One Pizza is all you need

BASUALDO Lautaro et LOI Léo

19 avril 2024

Table des matières

1 Le problème :

Nous ouvrons une pizzeria qui n'a au menu qu'une seule pizza.
Un client viendra dans notre pizzeria uniquement si les deux conditions suivantes sont remplies :

1. Tous les ingrédients qu'il aime sont sur la pizza
2. Aucun des ingrédients qu'il n'aime pas se trouve sur la pizza

Nous devons décider des ingrédients qui iront sur cette pizza afin de maximiser le nombre de clients qui achèteront cette pizza.

2 Petites instances du problème : recherche explicite

Tout d'abord, nous pouvons nous demander ce qu'est une solution au problème.
Nous avons choisit, ici, qu'une solution serait représentée par une liste d'ingredients. Un ingrédient de cette liste est un ingrédient qu'un client à dit aimer ou détester et chaque ingrédient n'apparaît qu'au plus une fois dans la liste.

Ainsi, si nous supposons N le nombre d'ingredients disponibles au total, la liste solution aura entre 0 et N ingrédients.
Mais si nous cherchons à calculer le nombre de solutions totales, la chose se complique un peu.

Soit n le nombre d'ingrédients et k la taille de la liste solution souhaitée, le nombre de combinaisons possible est calculé par la formule suivante :

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Sauf qu'ici, nous cherchons le nombre de solutions totales, ce qui revient à faire le calcul précédent pour toutes les tailles de listes allant de 0 à N.
On obtient ainsi le calcul suivant :

$$\sum_{k=0}^{n}(C_n^k)$$

Par exemple, si nous avons 6 ingrédients, le calcul devient :

$$\begin{aligned}
\sum_{k=0}^{6}(C_6^k) &= \sum_{k=0}^{6}(\frac{6!}{k!(6-k)!}) \\
&= \frac{6!}{0!(6-0)!} + \frac{6!}{1!(6-1)!} + ... + \frac{6!}{5!(6-5)!} + \frac{6!}{6!(6-6)!} \\
&= 1 + 6 + 15 + 20 + 15 + 6 + 1 \\
&= 64
\end{aligned}$$

Nous pouvons, naïvement, essayer de produire un programme permettant de trouver la solution au problème.
Nous pouvons par exemple créer le code suivant :

```
def choix_meilleur(liste, data):
  best = liste
  bestscore = 0
  score = 0
  for i in range(len(liste), 0, -1):
    #on teste toutes les combinaisons d'ingredients de taille i
    for j in combinations(liste, i):
      #pour chaque client
      for cpt in range(1, len(data), 2):
      #si aucun ingredient de j n'est deteste par le client
        if(len(list(set(j).intersection(data[cpt+1]))) == 0):
          #si j contient tous les ingredients aime par le client
          if(len(list(set(j).intersection(data[cpt])))+1 == len(data[cpt])):
            score += 1
    if score >= bestscore:
      best = j
      bestscore = score
      score = 0
  return best
```

3 Grosses instances du problème : métaheuristiques

3.1 Recuit Simulé : fait par Léo

Pour notre recuit simulé, entre chaque solution, nous autorisons trois mouvements différents.

- l'ajout d'un ingrédient qui n'est pas encore dans la solution
- l'échange d'un ingrédient par un autre qui n'est pas encore dans la solution
- la suppression d'un ingrédient de la solution

Dû à certains problèmes, je n'ai pas pû implémenter la suite de l'algorithme. La suite du rapport concerne ce que j'aurais souhaité implémenter
Comme expliqué dans le cour, je comptais faire une 100aine de modifications aléatoires puis de faire la moyenne de variation dans leurs scores. Je comptais faire débuter la température T_0 à 1000 puis de la faire diminuer de 0,9*T_n à chaque fois que soit 120 solutions sont acceptés soit que 1000 solutions ont été tentées J'ai décidé d'arrêter l'algorithme si nous ne constatons plus de changement après que la température ait changé 2 fois. Pour cela, nous avons mis un compteur qui se réinitialise à chaque modification de solution qui détermine l'arrêt des modifications et donc de l'algorithme