AVIGNON UNIVERSITÉ

Résolution du problème « Maximum within-clusters distance » en utilisant la Recherche taboue

Etudiant
Abdoulaye SAYOUTI SOULEYMANE

8 février 2021

Master d'Informatique
Intelligence Artificiel

UE TECHNIQUES D'APPRENTISSAGE AUTOMATIQUE
UCE 3 OPTIMISATION NON-LINEAIRE

Responsables
Rosa FIGUEIREDO

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Problème

1.1 Description

Le problème de clustering d'un ensemble d'objets se pose dans de nombreuses disciplines. En raison de la grande diversité des applications, ce problème se décline en de nombreuses variantes. Les principales La différence entre ces problèmes réside dans la fonction objective. Cependant, on sait très peu de choses sur les mérites de ces algorithmes. Même des questions simples concernant la complexité de calcul de la plupart des problèmes de clustering n'ont pas encore trouvé de réponse. Dans cet projet, nous implémentons l'algorithme de recherche taboue pour le problème MMD (Minimizing the Maximum within-clusters Distance) qui consiste à minimiser la distane maximale entre deux noeuds à l'intérieur du même cluster.

Rao (1971) [4] a proposé une formulation mathématique pour l'analyse de ce problème de clustering dont l'objectif est de trouver une partition à partir d'un ensemble de noeuds qui minimise le maximum de la distance entre deux noeuds d'un même cluster. Ce problème est également connu sous le nom de minimisation du plus grand diamètre parmi les clusters. La formulation mathématique est décrite par (1)-(4) comme présenté à la Figure 1 :

Parameters

N – number of objects of the data set;
M – number of clusters of the final partition;
d_{ij} – distance between objects i and j;

Variables

x_{ik} – binary variable that relates object i to cluster k ($x_{ik} = 1$, if object i is in cluster k; 0 otherwise);
Z – value of the largest diameter among the M clusters (continuum variable);

$$\text{Min} \quad Z \tag{1}$$

$$\text{subject to} \quad d_{ij}x_{ik} + d_{ij}x_{jk} - Z \leq d_{ij}$$
$$i = 1, \ldots, N-1; \quad j = i+1, \ldots, N; \quad k = 1, \ldots, M; \tag{2}$$

$$\sum_{k=1}^{M} x_{ik} = 1 \quad i = 1, \ldots, N; \tag{3}$$

$$x_{ik} \in \{0, 1\}; Z \geq 0 \quad i = 1, \ldots, N; \quad k = 1, \ldots, M. \tag{4}$$

The objective function (1) minimizes the value of Z that represents the maximum within clusters distance. Constraints (2) ensure that the value of the largest cluster diameter is lesser than or equal to the Z variable value. Constraints (3) impose that each object belongs to just one cluster. Constraints (4) enforce the non-negativity of the variable Z and the binary nature of the other variables.

Figure 1. Formulation du problème MMD.

1.2 Exemple

Considérons une configuration de 5 noeuds et 2 clusters avec une matrice de distances indiquées à la Figure 2. La solution optimale est présentée à la Figure 3 qui présente la configuration qui minimise le maximum de la distance entre deux noeuds au sein du même cluster. La valeur de la fonction objective est 8, qui correspond à la distance entre les noeuds 1 et 3.

	1	2	3	4	5
1	INF	5	8	8	10
2	5	INF	7	7	8
3	8	7	INF		8
4	8	7	8	INF	3
5	10	8	8	3	INF

Figure 2. Matrice de distances.

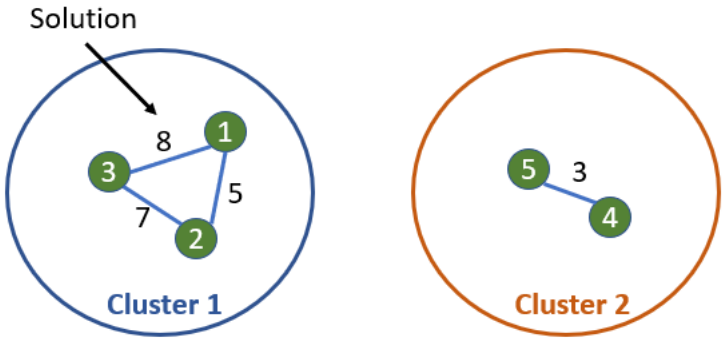

Figure 3. Solution.

2 Méthode de solution

2.1 Recherche taboue standard

La recherche taboue (TS en anglais) est une méthode méta-heuristique proposée à l'origine par Glover (1986) [1]. Le TS a été principalement proposée et développée pour des problèmes d'optimisation combinatoire et a prouvé qu'elle permettait de traiter divers problèmes (Rego et Alidaee, 2005) [5]. La principale caractéristique de la TS est l'utilisation d'une mémoire adaptative et d'une exploration réactive. Le TS simple combine une procédure de recherche locale avec des règles anticycliques basées sur la mémoire pour éviter que la recherche ne soit piégée dans des solutions optimales locales. Plus précisément, TS évite de revenir à des solutions récemment visitées en construisant une liste qui est une liste appelée Liste taboue (TL en anglais). TS génère de nombreuses solutions d'essai dans un voisinage de la solution actuelle et sélectionne la meilleure parmi toutes. Le processus de génération de solutions d'essai est composé de manière à éviter de générer une solution d'essai qui a déjà été visitée récemment. La meilleure solution d'essai parmi les solutions générées deviendra une solution actuelle. TS peut accepter des mouvements de descente pour éviter de se retrouver piégé dans l'optimum local. Le TS peut être arrêté si le nombre d'itérations sans amélioration dépasse un nombre maximum d'itérations prédéterminé.

2.2 Recherche taboue pour le problème MMD

La version modifiée de la recherche taboue pour les problèmes de clustering proposé proposé par Salwani Abdullah et al. [2] est proposé dans cette section pour le problème MMD . Certaines notations sont introduites ici avant que l'explication de l'algorithme ne soit donnée.

Soit A un tableau de dimension n dont le i^{eme} élément (A_i) est un nombre qui représente le cluster auquel le i^{eme} noeud est attribué. Ainsi, $A_i = j$ si le noeud i appartient au cluster C, pour tout $i = 1, 2., n$ et $C = 1, 2., k$. Par exemple, considérons le tableau suivant (A) pour $n = 10$, $k = 3$. Notez que n est le nombre de noeuds et k est le nombre de clusters.

2	3	1	1	2	3	1	3	2	2

Le premier noeud est affecté au deuxième cluster et le deuxième noeud est affecté au troisième cluster et ainsi de suite. Étant donné le tableau (A), la minimisation de la fonction objectif Z peut être définie comme :

$$min\ Z$$
$$\text{avec } Z = max\ (d_{ij}),\ i,j \in C \text{ et } i \neq j$$

Désignons par A_b, A_t et A_c respectivement la meilleure solution, la solution d'un voisin et la solution courantes et Z_b, Z_t et Z_c le coût correspondantes de la meilleure solution, de la solution du voisin et de la solution courante. L'algorithme fonctionne avec une solution courante A_c et ensuite, en appliquant des structures de voisinage, nous générons une solution voisine A_t. La meilleure solution trouvée jusqu'à présent, désignée par A_b, sera toujours conservée tout au long du processus de recherche. Deux listes taboues sont utilisées pour éviter un cycle (se retrouver avec la même solution) tout au long du processus de recherche.

2.3 Création des structures de voisinage

Il existe plusieurs stratégies pour générer la solution voisine. Dans ce projet, trois méthodes de voisinage différentes ont été utilisées :

2.3.1 Méthode d'échange

Ou **Swap method** en anglais consiste à sélectionnez un noeud au hasard dans un cluster et un autre noeud dans un autre cluster, puis les échangé. Il est décrit comme suit : Étant donné une solution courante A_c, sélectionnez deux noeuds au hasard $A_c(i)$ et $A_c(j)$ qui ne sont pas dans la même cluster, puis attribuez $A_c(i)$ au cluster auquel appartient $A_c(j)$ et vice versa.

2.3.2 Méthode unique

Ou **Single method** en anglais la méthode la plus simple. Un noeud est déplacé à chaque fois d'un cluster à un autre. Il est décrit comme suit : Étant donné une solution courante A_c, sélectionner aléatoirement $A_c(i)$, attribuer X qui est défini comme un entier généré aléatoirement dans la plage $[1, k]$, (k = nombre de clusters) et non égal au cluster auquel appartient l'élément $A_c(i)$. Ensuite $A_c(i)$ = X.

2.3.3 Méthode du seuil

Ou **Threshold method** en anglais est une méthode de seuil de probabilité couramment utilisé pour établir les solutions voisines (d'essai) dans la recherche taboue basée sur l'algorithme de clustering (Al-Sultan, 1995 [6]; Liu et al., 2008 [3]). Il permet de modérer le bouleversement de la solution actuelle. Plus la valeur du seuil de probabilité est élevée, moins le bouleversement est autorisé et, par conséquent, les solutions d'essai sont plus similaires à la solution actuelle et vice versa. La méthode de seuil de probabilité peut être décrit comme suit : Étant donné une solution courante A_c et un seuil de probabilité P, pour ($I = 1,...,n$ où n = nombre de noeuds), générer un nombre aléatoire $R \sim u$ (0, 1). Si $R < P$, alors $A_t(i)$ = $A_c(i)$, c'est-à-dire qu'aucune modification n'est effectuée ; sinon, générer aléatoirement un nombre entier (L) dans la plage $[1, k]$ qui n'est pas égal au cluster d'éléments $A_c(i)$, et attribué $A_c(i)$ = L.

2.4 L'algorithme

Le pseudo-code de l'algorithme de recherche taboue pour le problème MMD est présenté par l'Algorithme 1. Cet algorithme est structuré en plusieurs étapes :

- **Étape 1 : Initialisation**
  Définir A comme une solution initiale obtenue à partir d'une méthode d'heuristique constructive et Z comme la valeur de la fonction objective. Soit ensuite A_b = A et A_c = A, puis Z_b = Z, Z_c = Z avec A_c la solution courante. Fixez les valeurs des paramètres suivants : $method$ (méthode de voisinage), $MTLS$ (taille de la liste tabu des solutions),P (seuil de probabilité), NTS (nombre de solutions de voisinage) et que $ITMAX$ soit le nombre maximum d'itérations qui est fixé comme critère de terminaison. Soit TLL (longueur de la liste de taboues) = 0 et passer à l'étape 2. Pour ce projet, l'heuristique constructive utilisée est l'initialisation de façon aléatoire des clusters.
- **Étape 2 : Choisir la méthode de voisinage**
  Sélectionnez $method$ au hasard. Si $method$ est taboue, répétez l'étape 2, sinon passez à l'étape 3.
- **Étape 3 :** Utiliser de A_c pour générer un certain nombre de solutions voisin $A_{t1}, A_{t2}, ..., A_{tNTS} \in trials$ et évaluer les valeurs correspondantes des fonctions objectives $Z_{t1}, Z_{t2}, ..., Z_{tNTS} \in Z_{trials}$. Passer ensuite à l'étape 4.
- **Étape 4 :** Ordonner les solutions dans un ordre croissant ($Z_{t1}, Z_{t2}, ..., Z_{tNTS}$) : Si Z_{t1} n'est pas tabu, ou tabu mais $Z_{t1} < Z_b$, alors A_c = A_{t1} et Z_c = Z_{t1} et passez à l'étape 5 ; sinon A_c = A_{tL}, Z_c = Z_{tL}, où Z_{tL} est la meilleure fonction objective de $[Z_{t1}, Z_{t2}, ..., Z_{tNTS}]$ qui n'est pas tabu et passez à l'étape 5. Si toutes les solutions de voisinage sont taboues, retourné à l'étape 3.
- **Étape 5 :** Insérez A_c à la fin de la deuxième liste taboue (qui conserve les solutions visitées) et $TLL = TLL + 1$ (si $TLL = MTLS + 1$, supprimez le premier élément de la liste et $TLL = TLL - 1$). Si $Z_b > Z_c$, A_b = A_c et Z_b = Z_c. Si la solution ne s'améliore pas après un certain nombre d'itérations (dans ce travail, nous avons fixé à 10) alors insérez$method$ dans la première liste taboue (qui conserve les methodes de voisinage) et retourner à l'étape 2.
- **Étape 6 :** Vérifiez la condition d'arrêt : Si la condition d'arrêt est rempli, (A_b sera la meilleure solution trouvée et Z_b est la valeur de la meilleure fonction objective correspondante) ; sinon, retourner à l'étape 2

Input : A (Première solution obtenu par heuristique constructive)
Input : Z (coût de la solution)
Input : $MTLS$ (Taille maximale de la liste taboue)
Input : NTS (Nombre de solution de voisinage à générer)
Input : p (Seuil de probabilité)
Input : $ITMAX$ (Nombre maximale d'itération)
Output : A_b (Meilleure solution)

$NTLS \leftarrow 2$
$A_b \leftarrow A$
$A_c \leftarrow A$
$Z_b \leftarrow Z$
$method \leftarrow$Choisir aléatoirement la première méthode de voisinage
$count \leftarrow 0$
for $i \leftarrow 1$ *to* $ITMAX$ **do**
  if $count$ *== 10* **then**
    if *Taille de la liste taboue des méthodes de voisinage ==* $NTLS$ **then**
      Retirer la première méthode de voisinage de la liste taboue.
    end if
    Insérer l'encienne méthode utiliser dans la liste taboue des méthodes de voisinage.
    $method \leftarrow$Choisir aléatoirement une méthode de voisinage ne se trouvavant pas dans la liste taboue des méthodes de voisinage.
    $count \leftarrow 0$
  end if
  $trials \leftarrow$Générer NTS solutions candidates voisines à la solution courante A_c
  $Ztrials \leftarrow$Calculer le coût de chaque solution candidate.
  Triller les solutions $trials$ dans l'ordre croissant des coûts $Ztrials$.
  if $trials[1]$ *non taboue* ***OU*** *(*$trials[1]$ *taboue* ***ET*** $Ztrials[1] > Z_b$*)* **then**
    $A_c \leftarrow trials[1]$
    $Z_c \leftarrow Ztrials[1]$
  else
    $A_c \leftarrow$Prochaine solution candidate non taboue.
    $Z_c \leftarrow$Coût de la rochaine solution candidate non taboue.
    if *Toute les autres solution candidate sont taboue* **then**
      Retourner au point **14**
    end if
  end if
  Ajouter la solution courante A_c à la liste taboue des solutions.
  if *La taille de la liste taboue des solutions >* $MTLS$ **then**
    Retirer la première solution de de la liste taboue des solutions.
  end if
  if $Z_b > Z_c$ **then**
    $Z_b \leftarrow Z_c$
    $A_b \leftarrow A_c$
  else
    $count \leftarrow count + 1$
  end if
end for
Return A_b

Algorithme 1. Pseudo-code du Recherche taboue sur le problème MMD

3 Implémentation

Dans cette section, nous allons présenté la phase d'implémentation de l'agorithme.

3.1 Machine utilisée

La machine utilisée pour l'implémentation et l'expérimentation possède les caractéristiques suivantes :

- **PC :** HP ElitBook 745 G3
- **System :** Windows 10 Pro
- **Processeur :** AMD PRO A10-8700B R6, 10 Compute Cores 4C+6G, 1800 Mhz, 2 Core(s), 4 Logical Processor(s)

3.2 Language de programmation choisi

Le langage de programmation utilisé est le langage python. Phyton a été choisi pour sa simplicité d'utilisation. Il fournit également des bibliothèques efficaces pour la manipulation des structures de données.

3.3 Structures de données utilisées

En ce qui concerne les structures de données utilisées, deux structures ont été principalement choisies. La première est une matrice pour conserver les distances entre les différents noeuds. La seconde est une simple liste pour représenter les listes taboues et les solutions.

3.4 Paramétrage de la méthode

L'algorithme de recherche taboue implémenté dans ce projet est défini comme suite : $tabu\_search(A, Z, NTS, MTLS, ITMAX, p, k, dist\_matrix)$. Où $dist\_matrix$ représente la matrice distances et les autres paramètres ont été décrits dans la section 2.4.

Plusieurs autres algorithmes ont été implémentés dans le projet pour assurer le bon fonctionnement du code. Les commentaires nécessaires ont été ajoutés aux codes pour faciliter leur compréhension.

3.5 Testes réalisés et solutions obtenues

L'algorithme a été testé sur plusieurs instances et les résultats sont présenté dans le Tableau 1. Les instances considérées sont les suivantes :

- Instance de 30 noeuds avec 3 clusters.
- Instance de 50 noeuds avec 3 clusters en deux versions.
- Instance de 50 noeuds avec 5 clusters en deux versions.
- Instance de 50 noeuds avec 10 clusters en deux versions.

Les paramètres de l'algorithme ont été modifié à plusieurs reprise sans réel amélioration des résultats. J'ai donc décidé de garder les valeurs des paramètre comme suuggérer dans [2]. Les valeurs ainsi choisis sont les suivantes :

- Nombre de solution de voisinage à générer NTS = 20
- Taille maximale de la liste taboue $MTLS$ = 20
- Nombre maximale d'itération $ITMAX$ = 300
- Seuil de probabilité p = 0.95

Après mûr réflexion, il se pourrai que le problème vienne des intances sur lesquels l'algorithme est testé.

Instances	Solution optimale	Heuristique constructive	Recherche taboue
30 Noeuds et 3 Clusters	986	9858	8246
50 Noeuds et 3 Clusters V1	993	9970	9275
50 Noeuds et 3 Clusters V2	998	9961	9377
50 Noeuds et 5 Clusters V1	999	9996	8976
50 Noeuds et 5 Clusters V2	990	9925	8913
50 Noeuds et 10 Clusters V1	979	9970	7719
50 Noeuds et 10 Clusters V2	994	9888	7621

Table 1. Résultats des tests. Solution optimale, Première solution avec l'heuristique constructive et la solution finale obtenue par la recherche taboue.

4 Conclusion

Pour conclure, je pense avoir largement atteint les objectifs du projet en implémentant correctement l'algorithme de recherche taboue et en implémentant également les trois méthodes de voisinage. Cependant, une étude approfondie doit être réalisée pour comprendre la raison pour laquelle les résultats obtenus sont loin de la solution optimale. Pour le moment je pense que le problème doit se situer au niveau des instances générées.

Références

[1] Fred Glover. « Future Paths for Integer Programming and Links to Artificial Intelligence." Computers & Operations Research 13, 533-549 ». In : *Computers & Operations Research* 13 (jan. 1986), p. 533-549. doi : 10.1016/0305-0548(86)90048-1.

[2] Adnan Kharroushe, Salwani Abdullah et Mohd Nazr. « A Modified Tabu Search Approach for the Clustering Problem ». In : *Journal of Applied Sciences* 11 (déc. 2011), p. 3447-3453. doi : 10.3923/jas.2011.3447.3453.

[3] Yongguo Liu, Zhang Yi, Hong Wu, Mao Ye et Kefei Chen. « A tabu search approach for the minimum sum-of-squares clustering problem ». In : *Information Sciences* 178 (juin 2008), p. 2680-2704. doi : 10.1016/j.ins.2008.01.022.

[4] M. Rao. « Cluster Analysis and Mathematical Programming ». In : *Mathematical Programming* 79 (oct. 1969), p. 30. doi : 10.2307/2283542.

[5] Cesar Rego et Bahram Alidaee. « Metaheuristic optimization via memory and evolution. Tabu search and scatter search ». In : 30 (jan. 2005).

[6] Khaled Al-Sultan. « A Tabu search approach to the clustering problem ». In : *Pattern Recognition* 28 (sept. 1995), p. 1443-1451. doi : 10.1016/0031-3203(95)00022-R.