SIMPLICIAL COMPLEXES FOR TOPOLOGICAL DATA ANALYSIS

5 février 2022

Haozhe Cheng

ÉCOLE POLYTECHNIQUE

IP PARIS

TABLE DES MATIÈRES

1
INTRODUCTION

La Homologie Persistante est l'un des utils les plus importants dans le domaine d'analyse topologique des données. Les complexes filtrés sont au cœur du la construction des diagrammes persistants. Le Complexe de Čech est la complexe la plus naturelle à construire. Pourtant, elle s'avère de grandir d'une manière très rapide de façon à ce qu'il soit impossible de la calculer pour un nuage de points grand de dimension élevée. Le Complexe d'Alpha est un équivalent, dans le sens topologique, de le complexe Čech qui est plus facile à calculer. Dans la suite, nous nous restreignons dans le calcul de ces deux complexes en 2D et 3D. Nous illustrons aussi la différence entre eux. Le Complexe de Rips est un complexe approximative de le Complexe de Čech. Elle est construite à partir d'un graphe donnée. Avant de la calculer, on souhaite quand même simplifier le graphe sans changer sa propriété topologique. Dans la denière partie, nous présenterons l'algorithme que nous avons conçu pour répondre à ce besoin.

2
SPHÈRE MINIMUM

Cette question est un problème de type LP (generalized Linear Program), nous avons adapté l'algorithme proposée par Matoušek, Sharir, et Welzl [1] pour la résoudre. L'idée de cet algorithme consiste à tirer au hasard un point du P, on fait une récurence sans ce point dans P (on suppose alors que ce point est dans la sphère). Si l'hypothèse n'est pas vérifiée, on refaire la récurence avec ce point dans R. Ce program est de complexité linéaire. [1] L'exactitude peut être montrée par le raisonnement qu'un point est soit dans une sphère soit sur la surface.

Algorithm 1 Sphère Minimum

Input: P les points à examiner, R les points à la surface de la sphère
Outout: Sphère Minimum

```
function SphereMinimum(P, R)
    return cas de base
    p ← RandomPop(P)
    S ← SphereMinimum(P, R)
    if S contains p then
        return S
    end if
    return S ← SphereMinimum(P, R ∪ {p})
end function
SphereMinimum(Points, ∅)
```

Nous détaillerons le cas de base en 3D :

- **1 Point** La sphère Minimum est celle centrée sur ce point dont le diamètre égale à 0.
- **2 Points** La sphère Minimum est celle centrée sur le centre de ces deux points dont le diamètre égale à la distance entre ces deux points.
- **3 Points** En supposant que ces 3 points ne sont pas colinéaires, la sphère Minimum est celle dont le centre peut être considéré comme l'intersection des deux plans médiateurs et le plan défini par ces trois points. Supposons les trois points sont $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3)$, on n'a qu'a besoin de résoudre le système d'équation suivant :

$$\begin{cases} 2 \times (x_1 - x_2) + 2 \times (y_1 - y_2) + 2 \times (z_1 - z_2) + (x_2^2 + y_2^2 + z_2^2 - x_1^2 - y_1^2 - z_1^2) = 0 \\ 2 \times (x_2 - x_3) + 2 \times (y_2 - y_3) + 2 \times (z_2 - z_3) + (x_3^2 + y_3^2 + z_3^2 - x_2^2 - y_2^2 - z_2^2) = 0 \\ a + b + c - a \times x_1 - b \times y_1 - c \times z_1 = 0 \end{cases}$$

avec

$$\begin{cases} a = (y_2 - y_1) \times (z_3 - z_1) - (z_2 - z_1) \times (y_3 - y_1) \\ b = (z_2 - z_1) \times (x_3 - x_1) - (x_2 - x_1) \times (z_3 - z_1) \\ c = (x_2 - x_1) \times (y_3 - y_1) - (y_2 - y_1) \times (x_3 - x_1) \end{cases}$$

‘

- **4 Points** En supposant que ces 4 points ne sont pas sur le même plan, la sphère Minimum est celle définie par ces quatre points dont le centre peut être considérée comme l'intersection des trois plans médiateurs. Supposons les quatre points sont $(x_1, y_1, z_1), (x_2, y_2, z_2), (x_3, y_3, z_3), (x_4, y_4, z_4)$, on n'a qu'a besoin de résoudre le système d'équation suivant :

$$\begin{cases} 2 \times (x_1 - x_2) + 2 \times (y_1 - y_2) + 2 \times (z_1 - z_2) + (x_2^2 + y_2^2 + z_2^2 - x_1^2 - y_1^2 - z_1^2) = 0 \\ 2 \times (x_2 - x_3) + 2 \times (y_2 - y_3) + 2 \times (z_2 - z_3) + (x_3^2 + y_3^2 + z_3^2 - x_2^2 - y_2^2 - z_2^2) = 0 \\ 2 \times (x_3 - x_4) + 2 \times (y_3 - y_4) + 2 \times (z_3 - z_4) + (x_4^2 + y_4^2 + z_4^2 - x_3^2 - y_3^2 - z_3^2) = 0 \end{cases}$$

3
COMPLEXE DE ČECH

Définition Le complexe de Čech de valeur de filtration l est une ensemble des simplexes telle que toutes les sphères de rayon l centrées sur les sommets d'un simplexe ont un point commun.

Ce qui revient à dire que le rayon de la sphère circonscrite de cette simplexe est plus petit que l vu que le premier endroit que les sphères rencontrent est le centre de cette sphère circonscrite. En parcourant tous les simplexes et calculer leur sphère circonscrite, la complexité est $O(\sum_{i=0}^{n} iC_n^i)$, où n est le nombre de points.

En tenant compte du fait que la valeur de filtration d'un simplexe est plus grande que toutes les valeurs de filtration de ses sous-simplexes. Nous proposons donc de maintenir une liste L

des simplexes dont la valeur de filtration est plus grande que l. Si un simplexe contient un sous-simplexe dans L, au lieu de calculer sphère circonscrite, nous pourrons dire directement qu'elle ne vérifie pas le contraint.

Nous n'ajoutons que des simplexes qui ne contiennent pas de sous-simplexes de valeur de filtration plus grande que l dedans. Nous pourrons prouver qu'après un certain temps, L ne change plus. Dans ce cas, *contains* est de la complexité constante. Nous avons donc une meilleure complexité que calculer la sphère circonscrite chaque fois.

Parce que, la complexité de $Issubset$ est $O(len(tup))$. La complexité de *contains* est alors totalement déterminés par L.

Algorithm 2 Contains

Input: s un simplexe, L liste des simplexes de valeur de filtration plus grande que l
Outout: True si l'une des simplexes dans la liste L est un sous-simplexe de s

```
function CONTAINS(s, L)
    for tup ∈ L do
        if tup.ISSUBSET(s) then
            return True
        end if
    end for
    return False
end function
```

4
COMPLEXE D'ALPHA

Définition Le complexe d'Alpha est un sous-complexe de Delaunay d'une valeur de filtration α. Un simplexe dans le complexe de Delaunay fait partie de le complexe d'Alpha si et seulement si :

- ▷ La sphère circonscrite de ce simplexe a un rayon inférieur à α, et elle ne contient pas de sommet (Gabriel) .
- ▷ Ce simplexe fait partie d'un autre simplexe dans le complexe de Delaunay dont la valeur de filtration est plus petite que α.

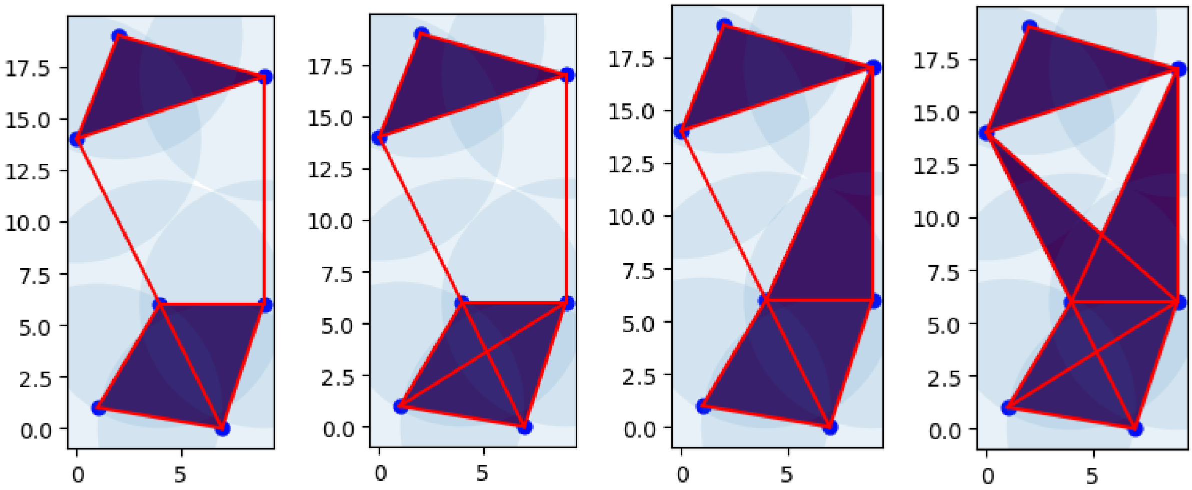
On a vu à partir de cette définition que la valeur de filtration d'un simplexe est le minimum entre celle des autres simplexes dont il fait partie et le rayon de sa sphère circonscrite Gabriel. Donc on ne peut déterminer la valeur de filtration d'un simplexe que quand on a déterminé celle de tous les simplexes de dimension supérieure. D'où nous avons conçu un algorithmes qui trouve d'abord le complexe de Delaunay, et examine ensuite tous les simplexes de Delaunay suivant un ordre de la dimension décroissante. Nous initialisons d'abord la valeur de filtration d'un simplexe comme le minimun des valeurs de filtration des simplexes dont il fait partie s'il n'existe pas de shpère circonscrite Gabriel. Sinon, la valeur de filtration est le rayon de la shpère circonscrite Gabriel. Sachant que ce rayon est la valeur de filtration la plus petite possible pour un simplexe, cet ordre d'assignation est justifié.

Nous nous plaçons dans le pire cas. Deux premières boucles for servent à parcourir tous les simplexes : $O(\sum_{i=0}^{n+1} N_i)$, où N_i est le nombre des simplexes de Delaunay de dimension i, n est la dimension de l'espace de l'espace des sommets. Dans la boucle, nous pourrions calculer la sphère circonscrite qui est $O(i)$ et parcourir tous les sous-simplexes de dimension $i-1$ qui est aussi $O(i)$. En parcourrant, il se peut que nous ayons besoins de calculer sphère circonscrite, qui est $O(i-1)+O(V)$. Pour conclure, la complexité totale est $O(\sum_{i=0}^{n+1} i^2 N_i)$, étant donné que l'opération "trouver" est $O(1)$ dans un table de hachage.

Algorithm 3 Complexe d'Alpha

Input: P Les sommets de le complexe d'Alpha
Outout: Les valeurs de filtration pour tous les simplexes

function AlphaFiltration(P)
  $fil \leftarrow$ Dict()
  $tri \leftarrow$ Tous les simplexes de Delaunay
  for $i =$ dimension de l'espace des sommets $+ 1 \rightarrow 0$ **do**
    for $C \in tri$ et C est de dimension i **do**
      if C n'est pas dans fil **then**
        $fil[C] \leftarrow$ Le rayon du sphère circonscrite (forcément Gabriel par construction d'algorithme)
      end if
      for tup sous-simplexe de C de dimension $i-1$ **do**
        if tup est dans fil **then**
          $fil[tup] \leftarrow$ Min($fil[tup], fil[C]$)
        else
          if La sphère circonscrite de tup n'est pas Gabriel **then**
            $fil[tup] \leftarrow fil[C]$
          end if
        end if
      end for
    end for
  end for
end function



(a) Alpha,filtration=6 (b) Čech,filtration=6 (c) Alpha,filtration=6.06 (d) Čech,filtration=6.06

Figure 1 – Différence entre le Complexe de Čech et d'Alpha

5 OPTIMISATION DE GRAPHE

5.1 Présentation de la solution

Comme proposé dans le sujet, on sait qu'il y a deux moyens de transformer un graphe sans changer sa persistent barcode : en parcourant les valeurs de filtration, soit une arête de valeur de filtration t est dominée en temp t, on redéfinit t comme la plus petite valeur de filtration qui est plus grande que t ; soit t est la plus grade valeur de filtration dans cette graph, on enlève cette arête.

5.2 Analyse de la complexité

Quand on utilise la matrice adjacente :

- **Filtered** qui va parcourir tous les éléments de la matrice est de $O(V^2)$.
- **IsDominationed** qui va parcourir tous les sommets et tous les voisins d'un sommet ou d'une arête est de $O(V^2)$.
- **ExistDomination** qui va parcourir tous les éléments de la matrice en faisant appel à **Filtered** et **IsDominationed**, est de $O(V^4)$ dans le pire cas.

- **Optimize** Dans le pire cas, toutes les arêtes sont de valeur de filtration différente. Pour calculer une borne assez triviale, on va donc enlever une arête par itération, et toutes les autres valeurs de filtration augmenent "d'un pas". La boucle "while" est alors de $O(V^6)$ en tenant compte de **ExistDomination**. Dans la boucle, la première boucle "for" donne $O(V^2)$, la deuxième boucle "for" donne $O(V^2)$, dans cette boucle, **IsDominationed** et **Filtered** sont de $O(V^2)$, la modification de G est de $O(1)$. la complexité totale est donc $O(V^{12})$. Notons que cette borne est n'est pas la borne supérieure de cet algorithme, ni la complexité moyenne. La situation réalle serait meilleure.

Certes, la liste adjacente aurait des avantages au terme de la complexité. Par contre, les graphes à optimiser sont souvent des graphes denses, Il n'y aurait pas de différences significatives entre deux méthodes d'implémentation.

RÉFÉRENCES

[1] J. Matoušek, M. Sharir, and E. Welzl, "A subexponential bound for linear programming," *Algorithmica*, vol. 16, no. 4-5, pp. 498–516, 1996.

Algorithm 4 Optimisation de Graphe

Input: G Graphe original
Outout: Graph optimizée

```
function Optimize(G)
    while ExistDomination(G) do
        T ← Toutes les valeurs de filtration dans le Graphe G
        for t in T do
            for e in Toutes les arêtes de valeur de filtration t do
                Gfiltered ← Filtered(G, t)
                if IsDominationed(Gfiltered, e) then
                    if t est la valeur la plus gande dans T then
                        Enlever e de G
                    else
                        Augmenter t de sort qu'il soit
                        la plus petite valeur dans T qui est plus grande que t
                    end if
                end if
            end for
        end for
    end while
    return G
end function
function ExistDomination(G)
    for e in Toutes les arêtes do
        Gfiltered ← Filtered(G, t)
        if IsDominationed(Gfiltered, e) then
            return True
        end if
    end for
    return False
end function
function IsDominationed(G, e)
    for v in Tous les sommets do
        if e est dominée par v then
            return True
        end if
    end for
    return False
end function
function Filtered(G, t)
    return G avec toutes les arêtes de valeur de filtration plus grande que t enlevées
end function
```