Simplicial complexes for topological data analysis

Yorgo Chamoun, Emmanuel Gnabeyeu

October 10, 2022

1 Introduction

L'ère du Big data est caractérisée par ce que nous pouvons appeler le déluge des données, l'accumulation massive et le nombre pléthorique. Les données sont générées massivement de nos jours par divers secteurs d'activité: la science, l'académie, l'industrie, la vie quotidienne ou le public en général. Souvent, ces données se présentent sous la forme de nuages de points échantillonnés dans des espaces à haute (ou infinie) dimension. Ils ne sont généralement pas distribués uniformément dans l'espace d'intégration mais portent une certaine structure géométrique qui reflète des propriétés importantes des "systèmes" dont ils ont été extraits ou ont été générés.
Ces données sont très utilisées notamment dans l'apprentissage et l'IA en général. Cependant, avant cela, pour plus de finesse, il convient de faire un préprocessing ou analyse de ces données. Il se pose la question de modélisation et gestion des données géométriques (espaces d'images, clustering, vers une réduction de dimension?) et plus particulièrement topologique (espace continu). C'est cette ambition que se donne la partie des maths appliquées et de l'informatique appelé Topological Data Analysis abrégée TDA dont le rôle est l'analyse d'ensembles de données à l'aide de techniques issues de la topologie. L'extraction d'informations à partir d'ensembles de données hautement dimensionnelles, incomplètes et bruyantes est généralement un défi. La TDA fournit un cadre général pour analyser de telles données d'une manière qui est insensible à la métrique particulière choisie et fournit une réduction de la dimensionnalité et une robustesse au bruit.

2 Objectif et motivation

Nous devons utiliser des outils de sélection de modèle pour choisir un complexe simplicial dans une filtration donnée.
Une stratégie naturelle pour déduire des informations topologiques pour une forme inconnue à partir d'un nuage de points est de considérer les décalages du nuage de points.
Cependant, les ensembles non discrets tels que les offsets, ainsi que les formes mathématiques continues telles que les courbes, les surfaces et, plus généralement,

les collecteurs, ne peuvent pas être facilement codés sous forme de structures discrètes finies. Les complexes simplifiés (complexe simplicial) sont donc utilisés en géométrie informatique pour approximer de telles formes. Ils peuvent être considérés comme des généralisations des graphes de voisinage. Il s'agit pour nous d'analyse de vastes réseaux de graphes (pouvant être donc ainsi vu comme un ensemble continu de points ou une union d'objets continus) par l'approche du complexe qui est la généralisation en dimension grande de cette notion même de graphe. Travailler directement avec une union d'objets continus étant difficile, nous la remplaçons donc par son nerf, c'est-à-dire un complexe simplicial et plus généralement un $k-$simplexe dont les arêtes sont les segments reliant deux sommets dont les boules de filtration se croisent. En utilisant des algorithmes de programmation linéaire, nous nous sommes donnés pour ambition d'écrire des fonctions en python nous générer les simplexes de dimension au plus k (entier) et de filtration au plus l pour divers modèles d'un ensemble de points (complexe de Čech, $\alpha-$complexe, complex Rips).

3 Résultats principaux et commentaires

3.1 Question 1

L'outil le plus important pour la définition de tels structures ou modèles de points évoqués ci-dessus est le problème de la plus petite boule englobante (en anglais, minimal enclosing ball(MEB). Il s'agit d'un problème mathématique consistant à calculer la plus petite boule (de l'espace 3D, ou le plus petit cercle du plan) qui contient tous les points d'un ensemble donné dans l'espace (resp dans le plan) euclidien. Plus généralement, c'est la boule de plus petit rayon telle que tous les objets/points sont entièrement contenus dans celle-ci. Ce rayon est appelé filtration.
Le bloc de construction ou la base d'obtention des paramètres d'une telle boule a été le calcul du centre circonférentiel d'un ensemble non dégénéré de points p_0,..., p_k.
Nous avons écrit ce centre comme le barycentre de ces points avec des poids inconnues. En remarquant que le fait d'être équidistant de p_0 et de p_i s'écrit comme une équation linéaire, la résolution d'une telle équation grâce au solveur linalg.solve() de numpy permet de trouver ces poids barycentriques et puis déduire le centre et le rayon de la sphère circonscrite.
La difficulté est de traduire "X est équidistant de chaque point et se trouve dans leur enveloppe convexe" en un système d'équations linéaires. Nous allons écrire le système en dimension 3 pour plus de clareté, mais l'implémentation se fera en dimension quelconque. Soit O l'origine du repère. Commençons par traduire "X est équidistant de A et B". Le milieu C de AB est donné par $\overrightarrow{OC} = (\overrightarrow{OA} + \overrightarrow{OB})/2$. X est équidistant de A et B si et seulement si

$\overrightarrow{AB}.\overrightarrow{CX} = \overrightarrow{0}$. En combinant les deux, on obtient:

$$\begin{cases} (x_B - x_A)(x - \frac{x_B + x_A}{2}) = 0 \\ (y_B - y_A)(y - \frac{y_B + y_A}{2}) = 0 \\ (z_B - z_A)(y - \frac{z_B + z_A}{2}) = 0 \end{cases}$$

A présent, "X est dans l'enveloppe convexe de points de n points" se traduit par l'existence de $n-1$ constantes a_i telles que $\sum_i a_i \leq 1$, avec

$$\begin{cases} x = \sum_i a_i x_i + (1 - \sum_i a_i) x_n \\ y = \sum_i a_i y_i + (1 - \sum_i a_i) y_n \\ z = \sum_i a_i z_i + (1 - \sum_i a_i) z_n \end{cases}$$

Au final, on traduit "X est équidistant de chaque point et se trouve dans leur enveloppe convexe" pour, par exemple, 4 points A, B, C et D par une équation matricielle:

$$A. \begin{pmatrix} x \\ y \\ z \\ a_1 \\ a_2 \\ a_3 \end{pmatrix} = \begin{pmatrix} \frac{x_B^2 - x_A^2 + y_B^2 - y_A^2 + z_B^2 - z_A^2}{2} \\ \frac{x_C^2 - x_A^2 + y_C^2 - y_A^2 + z_C^2 - z_A^2}{2} \\ \frac{x_D^2 - x_A^2 + y_D^2 - y_A^2 + z_D^2 - z_A^2}{2} \\ x_D \\ y_D \\ z_D \end{pmatrix}$$

avec

$$A = \begin{pmatrix} x_B - x_A & y_B - y_A & z_B - z_A & 0 & 0 & 0 \\ x_C - x_A & y_C - y_A & z_C - z_A & 0 & 0 & 0 \\ x_D - x_A & y_D - y_A & z_D - z_A & 0 & 0 & 0 \\ 1 & 0 & 0 & x_D - x_A & x_D - x_B & x_D - x_B \\ 0 & 1 & 0 & y_D - y_A & y_D - y_B & y_D - y_B \\ 0 & 0 & 1 & z_D - z_A & z_D - z_B & z_D - z_B \end{pmatrix}$$

On a bien le même nombre de lignes et de colonnes, égale à la somme de la dimension de l'espace et du nombre de points moins 1. La matrice peut ne pas être inversible, ce que nous prenons en compte dans notre algorithme. Si le nombre de points est inférieur ou égal à la dimension, la solution est la plus petite boule ayant ces points sur sa frontière, car son centre est dans leur enveloppe convexe, et c'est bien ce que nous cherchons.

3.2 Question 2

Le complexe de Čech est un complexe simplicial abstrait construit à partir d'un ensemble de points dans un espace métrique(ici l'espace euclidien). Étant donné un ensemble fini de points X et un réel r positif, le complexe de Čech $C(r)$ est l'ensemble des simplexes (généralisation du triangle en dimension quelconque) tels que les boules de rayon r et de centre les points X ont une intersection non vide. Il peut être vu comme le nerf de l'ensemble des boules de rayon r centrées sur les points de X (il est équivalent homotopiquement à l'union de ces boules).

Le seul point d'ombre est le lien entre MEB et valeur de filtration. La réponse est donnée par le

Lemme 1 *Le rayon de la MEB d'un ensemble de points est égal à la valeur de filtration du simplexe généré par ces points.*

Preuve: Lorsque la valeur de filtration est atteinte, les boules centrées en chaque sommet ont au moins un point en commun. Ces points se trouvent à l'intérieur de chaque boule, donc sont à une distance inférieure ou égale au rayon de ces sommets. Donc chacun est le centre d'une boule de rayon cette distance qui contient tous les points. Comme la valeur de filtration est le plus petit rayon pour lequel cela est vrai, alors c'est bien le rayon de la MEB. Ceci montre par la même occasion que le point commun pour la valeur de filtration est unique.

3.3 Question 3

Pour optimiser l'algorithme naif, nous avons pensé à définir une notion de $d-$voisin pour les $k-$simplexes.

Définition 1 *Un sommet est voisin d'un $k-$simplexe s'il n'appartient pas aux sommets de ce simplexe et se trouve à une distance au plus d de chaque sommet du simplexe.*

En effet, si k points forment un simplexe pour une valeur de filtration d, alors les seuls candidats pour former un $(k+1)-$simplexe avec ces sommets sont les voisins du simplexe. Nous avons choisi cette définition car la liste des voisins est facile à mettre à jour: les voisins d'un simplexe formé à partir d'un simplexe de dimension inférieur et d'un sommet supplémentaire sont exactement les sommets qui sont voisins du simplexe et du sommet. Nous avons donc implémenté la liste des voisins sous forme de *sets* pour optimiser l'opération d'intersection. De plus, nous avons stocké les simplexes de même dimension dans un même dictionnaire et avons créé une liste de dictionnaire parallèle pour stocker et faciliter la mise à jour des voisins.

Remarque 1 *Nous aurions pu utiliser une autre définition de voisins plus restrictive, comme: un sommet est voisin d'un $k-$simplexe s'il forme avec tous $k-1$ sommets de ce simplexe un $k-$simplexe. Cette définition est valide mais est plus difficile à mettre à jour, donc nous avons décidé d'utiliser celle ci-dessus comme compromis entre nombre de tests et difficulté de mise à jour.*

Nous comparons les temps d'exécution des méthodes naive et non-naive pour un nombre n de points générés aléatoirement et pour différentes valeurs de filtration l. L'algorithme non-naive est évidemment plus rapide, et le devient de plus en plus lorsque l est petit, ce qui est logique puisque cela réduit le nombre de voisins et donc de comparaisons à chaque étape. (voir le tableau ci-dessous).

	valeurs de n	valeurs de l	valeurs de t1	valeurs de t2
0	4.0	3.5	0.002808	0.000948
1	4.0	4.5	0.002019	0.001091
2	4.0	5.5	0.001886	0.000716
3	4.0	6.5	0.001933	0.000764
4	5.0	3.5	0.004623	0.001109
5	5.0	4.5	0.004615	0.001096
6	5.0	5.5	0.004842	0.001151
7	5.0	6.5	0.004767	0.001186
8	6.0	3.5	0.009876	0.001581
9	6.0	4.5	0.009900	0.001607
10	6.0	5.5	0.010000	0.001629
11	6.0	6.5	0.010838	0.003221
12	7.0	3.5	0.019199	0.002326
13	7.0	4.5	0.019595	0.002208
14	7.0	5.5	0.019567	0.002223
15	7.0	6.5	0.019272	0.002224
16	8.0	3.5	0.033740	0.002804
17	8.0	4.5	0.034060	0.002889
18	8.0	5.5	0.034066	0.002809
19	8.0	6.5	0.033801	0.002823
20	9.0	3.5	0.055946	0.003674
21	9.0	4.5	0.055772	0.003658
22	9.0	5.5	0.056409	0.003774
23	9.0	6.5	0.056096	0.004483

3.4 Question 4

L'$\alpha-$complexe est le complexe simplicial dans lequel au lieu d'avoir une boule autour de chaque point, on ne garde que la partie de la boule qui est dans la cellule de Voronoï de ce point. Sauf aux frontières, cela supprime les redondances dans le recouvrement, sans changer l'union.
Nous utilisons un algorithme de type Seidel pour trouver la plus petite boule ayant les sommets en question sur sa frontière. La base B et l'ensemble S des points sont confondus, et la fonction f représente le rayon de la plus petite boule ayant l'ensemble des points de B sur sa frontière (correspondant à l'algorithme de la question 1). On a toujours $f(B) \leq f(B \cup \{x\})$. L'idée est simple: on part des sommets donnés en entrée, on construit la plus petite boule qui a ces sommets sur sa frontière, et on teste si les autres sommets sont à l'intérieur de cette boule. Quand on en trouve un à l'intérieur, on construit, si elle existe, la boule ayant les sommets initiaux et ce nouveau sommet sur sa frontière. Si elle n'existe pas, on renvoie -1. Si elle existe, il faut, avant de continuer à tester les sommets restant, s'assurer que ceux déjà testés sont bien à l'extérieur de cette nouvelle boule, car son rayon est strictement supérieur à celui de la précédente. On continue jusqu'à avoir testé tous les sommets ou arriver à un système insoluble.

Reste à expliciter le lien entre cette boule et l'$\alpha-$filtration.

Lemme 2 *L'$\alpha-$filtration d'un ensemble de sommets, si elle existe, est égale au rayon de la plus petite boule vide ayant ces sommets sur sa frontière.*

Preuve: Supposons qu'il existe une boule vide avec les sommets en question sur sa frontière. Son centre est équidistant de ces sommets, et aucun autre sommet n'est plus proche de lui, donc il appartient à la frontière des cellules de Voronoi de ces sommets, et les boules de rayon égal au rayon de cette boule centrées en ces sommets ce touchent en ce points. Le raisonnement inverse donne la réciproque. L'$\alpha-$filtration étant le plus petit rayon pour lequel cela se produit, on a bien le résultat.

3.5 Question 5

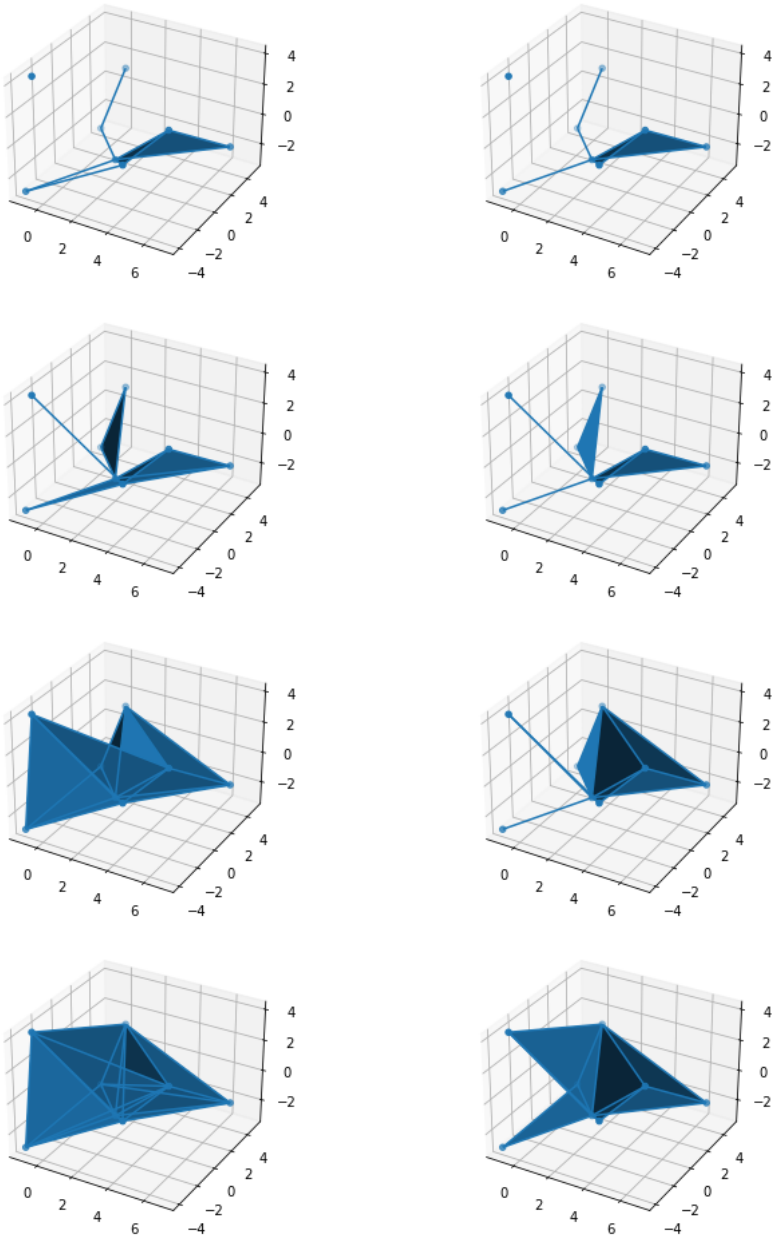
Nous allons illustrer la différence entre le complexe de Čech et l'$\alpha-$complexe en traçant quelques figures. Notons d'abord une difficulté rencontrée lors de nos essais: lorsque la liste des triangles à tracer était vide, la fonction "ax.plot_trisurf" effectuait une triangularisation par défaut, et traçait donc des triangles entre les sommets. Nous avons résolu le problème en ajoutant une condition "if len(simplexes[2]) !=0" avant l'appel de cette fonction.

Ci-dessous les figures promises. Les sommets sont (5,0,1), (-1,-3,4), (-1,-4,-3), (-1,4,-3), (4,-3,0), (0,5,1), (7,3,-1) et (2,0,-2). A gauche le complexe de Čech et à droite l'$\alpha-$complexe, pour des filtrations d'au plus (de haut en bas) 3, 3.5, 4 et 4.5.

Remarque 2 *C'est évidemment la notion de voisin définie plus haut qui permet d'avoir un algorithme aussi efficace. Une libraire qui optimise ces opérations aurait donc nécessairement amélioré notre algorithme.*

3.6 Question 6

Le complexe de Rips possède les mêmes sommets et arêtes que le complexe de Čech et la valeur de filtration des arêtes est encore la moitié de leur longueur. Cependant, pour les simplexes de dimension supérieure, il s'agit d'un complexe drapeau (ou complexe de cliques), c'est-à-dire que k + 1 points définissent un k-simplexe du complexe de Rips de paramètre r si et seulement si le graphe complet sur ces k + 1 sommets est dans le complexe. En d'autres termes, comme pour le complexe de Čech, tous les (k + 1)-tuples de points définissent un k-simplexe du complexe de Rips pour r suffisamment grand, mais la valeur de filtration est maintenant la moitié du diamètre du simplexe, quelque chose de beaucoup plus facile à calculer que le rayon du MEB.
L'objectif a été de transformer un graphe d'entrée en un nouveau graphe plus petit, sans temps d'exécution ni utilisation de mémoire énormes. Nous avons commencé par un algorithme assez naif: on trie les arêtes par ordre croissant de filtration, et à chaque étape on ajoute toutes les arêtes ayant la même filtration,



puis on teste si l'une ou plusieurs d'entre elles sont dominées, auquel cas on les retire du graphe pour les réintroduire à l'étape suivante. Tester si une arête est dominée se fait de la manière suivante: on calcule les voisins de l'arête, puis on teste s'il en existe un qui soit voisin de tous les autres. Nous utilisons une ma-

trice d'adjacence, car les exemples sur lequels nous travaillons sont des graphes complets, et que dans ce cas l'avantage des listes d'adjacence disparaît. Dans ce cas, trouver les voisins d'une arête (soit les sommets voisins à la fois des deux sommets de l'arête), en énumérant les voisins d'un des sommets et en vérifiant s'ils sont voisins de l'autre se fait en au plus $|V|$ étapes, puis tester si un des voisins domine l'arête se fait en $k := max_{u \in V}\{deg(u)\}$ étapes, ce qui fait une compléxité totale en $O(|V| + k^2) = O(k^2)$ puisque $|V| \approx k$. L'algorithme fait ce test pour chaque arête, autant de fois qu'elle est réintroduite, donc nous avons une compléxité en $O(m.|E|.k^2)$ pour l'ensemble de l'algorithme.
Il est assez évident que cet algorithme marche et qu'il donne la solution optimale. Cependant, ses performances en terme de temps d'exécution sont clairement perfectibles. En effet, à chaque fois qu'une arête est dominée, elle est réintroduite à l'étape d'après et subit de nouveau le teste de domination. Certains de ces tests peuvent être évités, grâce à l'obsevation suivante.

Lemme 3 *Si à une étape $i+1$ toutes les arêtes de filtration initiale $i+1$ sont dominées, alors toutes celles héritées des étapes précédentes sont elles aussi dominées à l'étape $i+1$.*

Preuve: Quand on retire les arêtes de filtration initiale $i+1$ (car elles sont dominées), on se retrouve avec la même configuration de graphe qu'à l'étape i, donc les arêtes qui y étaient dominées le restent.

Ceci nous permet d'améliorer notre algorithme en créant un espace de stockage propre aux arêtes héritées, et en ne testant leur domination que si l'une des arêtes initiales n'est pas dominée. Le lemme ci-dessus prouve que cet algorithme est toujours correcte et donne toujours la solution optimale. Le facteur m dans la compléxité est remplacé par le nombre d'étapes ayant une arête initiale non dominée, qui peut se révéler être bien inférieur. Comme illustré par les tests ci-dessous (premier tableau pour le premier algorithme, deuxième pour le deuxième), les performances avec, par exemple, un graphe complet avec sommets aléatoires et filtration de Rips sont nettement améliorées: les distances étant presque toutes différentes, on a $m \approx |E|$ donc une très grande compléxité si on utilise le premier algorithme, mais comme le nombre de simplifications possibles est très important, le deuxième algorithme donne des temps d'exécution tout-à-fait satisfaisants, comme illustré ci-dessous.

Pour n=	t1 pour graph_complet	t1 pour 2n-gon_régulier	t1 points aléatoires dans_un_carée
40	0.007522	0.061487	1.889295
50	0.012479	0.126902	5.412272
60	0.020343	0.247459	13.100724
70	0.031105	0.450604	28.742540
80	0.045095	0.784170	55.321516
90	0.062696	1.184102	101.772915
100	0.084543	1.767104	167.618734
110	0.110774	2.565789	265.563032

Pour n=	t2 pour graph_complet	t2 pour 2n-gon_régulier	t2 points aléatoires dans_un_carée
40	0.007377	0.047415	0.090565
50	0.012274	0.096879	0.220859
60	0.020098	0.187299	0.447292
70	0.030821	0.329861	0.825942
80	0.044710	0.582582	1.446042
90	0.062298	0.870759	2.347033
100	0.083855	1.261665	3.470213
110	0.110616	1.838020	5.105001