Algorithmes pour la persistence topologique

Vincent Billaut, Armand Boschin, Théo Matussière

Topological Data Analysis, INF563

Lecture du code

Nous avons utilisé Python 3.5 pour coder les algorithmes. Nous utilisons deux classes principales : `Simplex` et `Filtration`, et une classe auxiliaire qui implémente une *sparse-matrix* de booléens, `SparseBool`.

Ces trois classes sont en annexes.

`Simplex` est une classe très simple, on cacule la frontière du simplex directement à l'enregistrement.

`Filtration` est la classe qui permet tous les calculs, toutes les méthodes ont en commun une table de hashage du nom du simplex vers son numéro de colonne/ligne dans la matrice des frontières, partagée elle aussi. Une fois la réduction effectuée on garde aussi la table des pivots de chaque colonne, utile pour le calcul du code-barre.

`SparseBool` est la classe qui implémente la structure de matrice *sparse*, en utilisant un simple dictionnaire ayant pour clé le numéro d'une colonne, et pour valeur l'ensemble (`set` en python) des indices de ligne pour lequel la cellule contient un 1. On performe les calculs en faisant une simple différence symétrique - un `XOR`.

Performances

Voilà le tableau des performances de notre implémentation. Lorsque les parties ne s'additionnent pas au total, c'est que le fichier a pris un temps significatif à être lu (la différence, simplement).

Filtration	Temps total	`boundary_matrix`	`reduce`	`barcode`
2-sphere	0.004s	0.003s (74%)	0.000s (10%)	0.000s (7%)
3-sphere	0.004s	0.002s (64%)	0.000s (9%)	0.000s (5%)
4-sphere	0.005s	0.003s (65%)	0.000s (11%)	0.000s (8%)
...	...	...	...	...
9-sphere	0.056s	0.011s (20%)	0.012s (21%)	0.003s (5%)
filtration_A	204.274s	1.996s (0%)	196.888s (96%)	0.527s (0%)

filtration_B	9.036s	0.518s (5%)	7.193s (79%)	0.131s (1%)
filtration_C	23.670s	0.839s (3%)	20.499s (86%)	0.221s (0%)
filtration_D	752.958s	12.465s (1%)	702.956s(93%)	3.280s (0%)

Commentaire sur la complexité

Nos algorithmes ont une complexité théorique en O(*m*^3) **dans le pire des cas**, *m* étant le nombre de simplexes de la filtration.

Les rapports de tailles (*m*) entre les quatre filtrations A, B, C et D sont les suivants:

- A : 4 (m=480k)
- B : 1 (m=110k)
- C : 2 (m=180k)
- D : 25 (m=2.7m)

On voit que le temps effectif de calcul se rapproche plus d'une loi linéaire que d'une loi cubique. Ceci est dû à notre implémentation avec des matrices creuses. Celles-ci permettent d'exploiter la puissance des objets `Set` en `python` et de réduire les parcours de matrices aux seules cases remplies.

On passe de m^2 cases à *m*, et l'accès est également plus rapide.

Interprétations

Nos résultats pour les objets "classiques" sont très satisfaisants.

Nous présentons les *barcodes*, et dans le cas du **ruban de Moebius**, du **tore**, du **plan projectif** et de la **bouteille de Klein**, le schéma simplicial, en annexe (B).

Les d_sphères et d_boules

L'ajout des arêtes ne laisse qu'une seule composante connexe et crée des cycles (dimension 1). L'ajout de faces (dimension 2) "tue" ces cycles et crée des vides (dimension 3). Ainsi de suite...

On remarque que la seule différence entre les *barcodes* d'une d_sphère et d'une (d+1)_boule est la composante de plus grande dimension (en haut sur le graphique) qui est "tuée" par l'ajout du volume (de dimension d+1) à l'intérieur de la sphère.

Bouteille de Klein, Ruban de Moebius, Plan projectif et Tore

Dans tous les cas, on observe la persistence, en plus d'une composante connexe, d'au moins un cycle (dimension 1), ce qui est cohérent avec la géométrie de la forme : il existe des "poignées".

Toutes les formes à l'exception du ruban de Moebius comportent un creux (enferment un vide de dimension 3), d'où la composante de H2 qui perdure.

Filtrations simulées

Conclusion commune sur la structure topologique de chacune (A,B,C,D)

Chacune des formes est unique. Les *barcodes* donnent des indices sur la particularité de chacune d'entre elles. En regardant la photographie du barcode à t=inf, on peut néanmoins tirer des conclusions communes vis-à-vis des quatre formes, d'un point de vue global.

Elles sont :

- connexes (car une seule composante d'H0 perdure)
- sans poignées
- sans creux

Il semble que la filtration A a moins de composantes dans H2 que C ou D par exemple, ce qui laisse penser une structure moins "massique" et plus "planaire", c'est-à-dire formée de faces qui créent peu de creux (dimension 3).

Annexes

A - Code

Pour printer les *barcodes* des filtrations A, B, C et D, il est nécessaire d'adapter la valeur du `THRESHOLD` dans le code.

De plus, nous avons mis un paramètre dans le programme `test.py` pour permettre d'afficher en échelle logarithmique.

Un autre paramètre permet de plotter directement dans un fichier.

Nous avons réussi à afficher les *barcodes* des filtrations A et B, mais à force de bidouillage sur le code (la ligne qui permet de filtrer en fonction du seuil est différente : une prend en compte l'échelle logarithmique en amont, l'autre non).

```
class SparseBool:

    def __init__(self):
        self.d = dict()

    def set(self, row, col):
        try:
            self.d[col].add(row)
        except KeyError:
            self.d[col] = set([row])

    def add(self, colA, colB):
        try:
            self.d[colA].symmetric_difference_update(self.colB)
```

```
        except KeyError:
            pass


class Simplex:
    """ Basic simplex class with val as its time of appearance, its dimension,
        and vert as the indices of its members
    """

    def __init__(self, val, dim, vert):
        # Time of appearance
        self.val = val
        # Dimension of the simplex
        self.dim = dim
        # Vertices of the simplex
        self.vert = vert
        # Its boundary
        self.boundary = [
            "-".join(map(str, sorted(b)))
            for b in combinations(vert.split("-"), len(self.vert.split("-")) - 1)
        ]

    def __str__(self):
        return "val: {}; dim: {}; vert: {}, boundary: {}".format(
            self.val, self.dim, self.vert, self.boundary)

    def __repr__(self):
        return self.__str__()


class Filtration:

    def __init__(self, filepath):

        filtration = list()

        text_file = open(filepath, "r")
        for line in text_file:
            line = line.split()
            simplex = Simplex(
                float(line[0]),
                int(line[1]),
                str(line[2]) if len(line) == 3
                    else "-".join(map(str, sorted(line[2:]))))
            )
            filtration.append(simplex)
        text_file.close

        self.filtration = sorted(filtration, key=lambda x: x.val)
        self.to_idx = dict(
            [(s.vert, i) for i, s in enumerate(self.filtration)])

    def boundary_matrix(self):

        matrix = SparseBool()
```

```
        for column, simplex in tqdm(enumerate(self.filtration)):
            if simplex.dim == 0:
                continue
            for name in simplex.boundary:
                row = self.to_idx[name]
                matrix.set(row, column)
        self.bm = matrix

    def reduce(self):

        pivots_dic = dict()
        nb_cols = self.bm.shape[1]

        for column in tqdm(range(nb_cols)):
            row = find_pivot(self.bm, column)
            while row in pivots_dic.keys():
                self.bm.add(pivots_dic[row])
                row = find_pivot(self.bm, column)
            if row:
                pivots_dic[row] = column

        self.pivots = pivots_dic

    def barcode(self):
        bars = list()
        for col in tqdm(range(self.bm.shape[1])):
            if not self.bm.d.get(col) and col not in self.pivots.keys():
                bars.append((col, np.inf))
        bars.extend(self.pivots.items())
        self.barcode = bars
        self.bc_clean = [
            (
                self.filtration[a].dim,
                self.filtration[a].val,
                (self.filtration[b].val if b != np.inf else np.inf)
            )
            for a, b in bars
        ]
```

Annexe B:

1) Sphères et boules :

Pour rappel, nous prenons les notations qui font de la sphère usuelle une 2_sphere et de la boule correspondante (usuelle en 3D) une 3_ball.

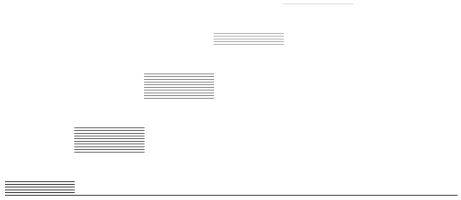
Barcode for the 2_sphere

Barcode for the 3_ball

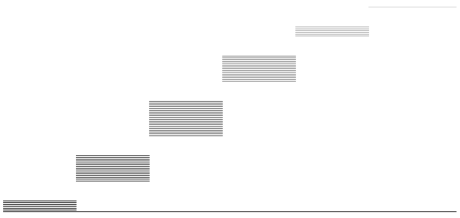
Barcode for the 3_sphere

Barcode for the 4_ball

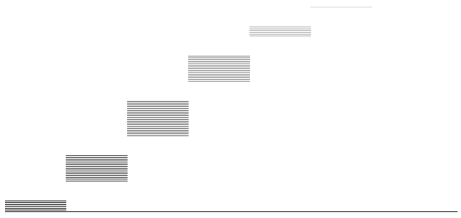
Barcode for the 4_sphere



Barcode for the 5_ball



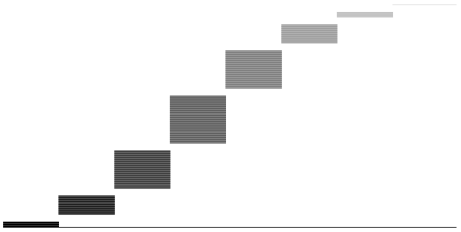
Barcode for the 5_sphere



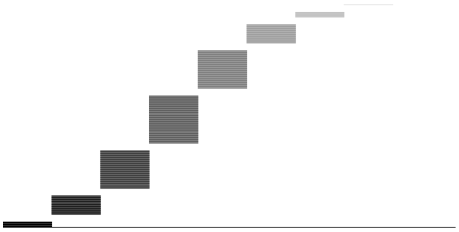
Barcode for the 6_ball

Barcode for the 6_sphere

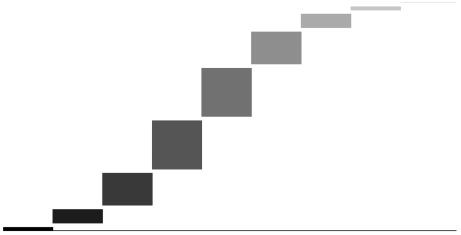
Barcode for the 7_ball



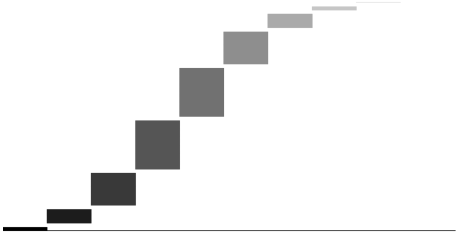
Barcode for the 7_sphere



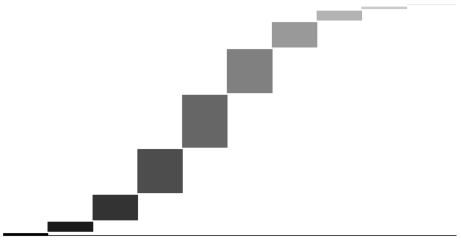
Barcode for the 8_ball



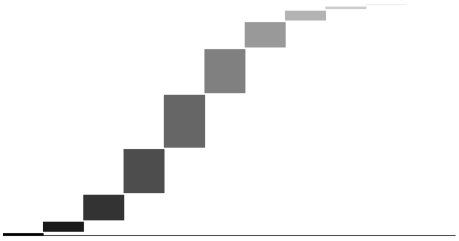
Barcode for the 8_sphere



Barcode for the 9_ball



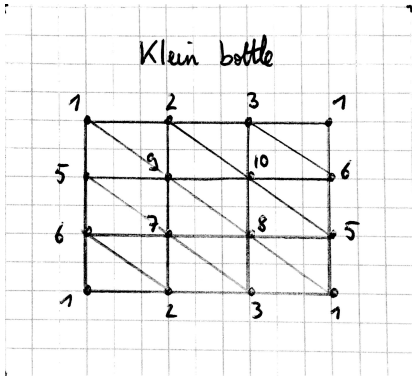
Barcode for the 9_sphere



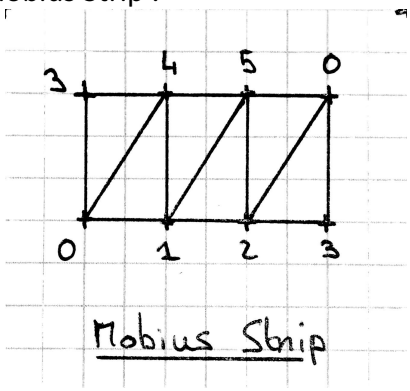
Barcode for the 10_ball

2) Autres filtrations:

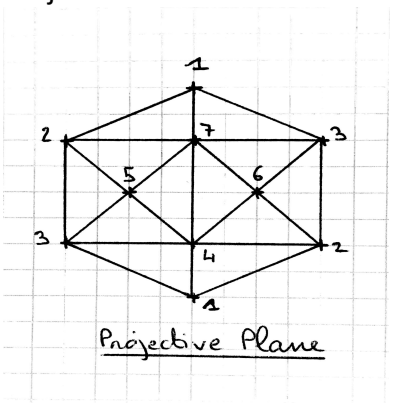
Bouteille de klein:



Mobius Strip :



Projective Plane :



Torus :