Polygonisation simple

Kamel Naroun

30 août 2013

Résumé

Etant donné un ensemble de points, il existe un certain nombre de polygones ayant ces points comme sommets. Mais existe-t-il toujours un polygone simple parmi ceux-là ? Comment est celui de périmètre minimal ? Nous allons tenter de répondre à ces questions et voir quels traitements algorithmiques, nous pouvons faire pour y répondre, en utilisant le langage Python.

Table des matières

1 Introduction

Soit un ensemble donné de points. Trouver un polygone simple (non croisé) qui passe par tous les points : est-ce possible dans tous les cas ? Ce problème se nomme *polygonisation simple*. Nous verrons que sauf cas particulier cela est toujours possible. Nous verrons aussi un algorithme qui peut suffire de preuve.

On peut également s'interesser au polygone de périmètre minimal. Ce problème-ci est exactement le problème du voyageur de commerce. En anglais, c'est *traveling salesman problem* et TSP en abrégé. L'énoncé du problème du voyageur de commerce est le suivant :

> *étant donné n points (des « villes ») et les distances séparant chaque point, trouver un chemin de longueur totale minimale qui passe exactement une fois par chaque point et revienne au point de départ.*

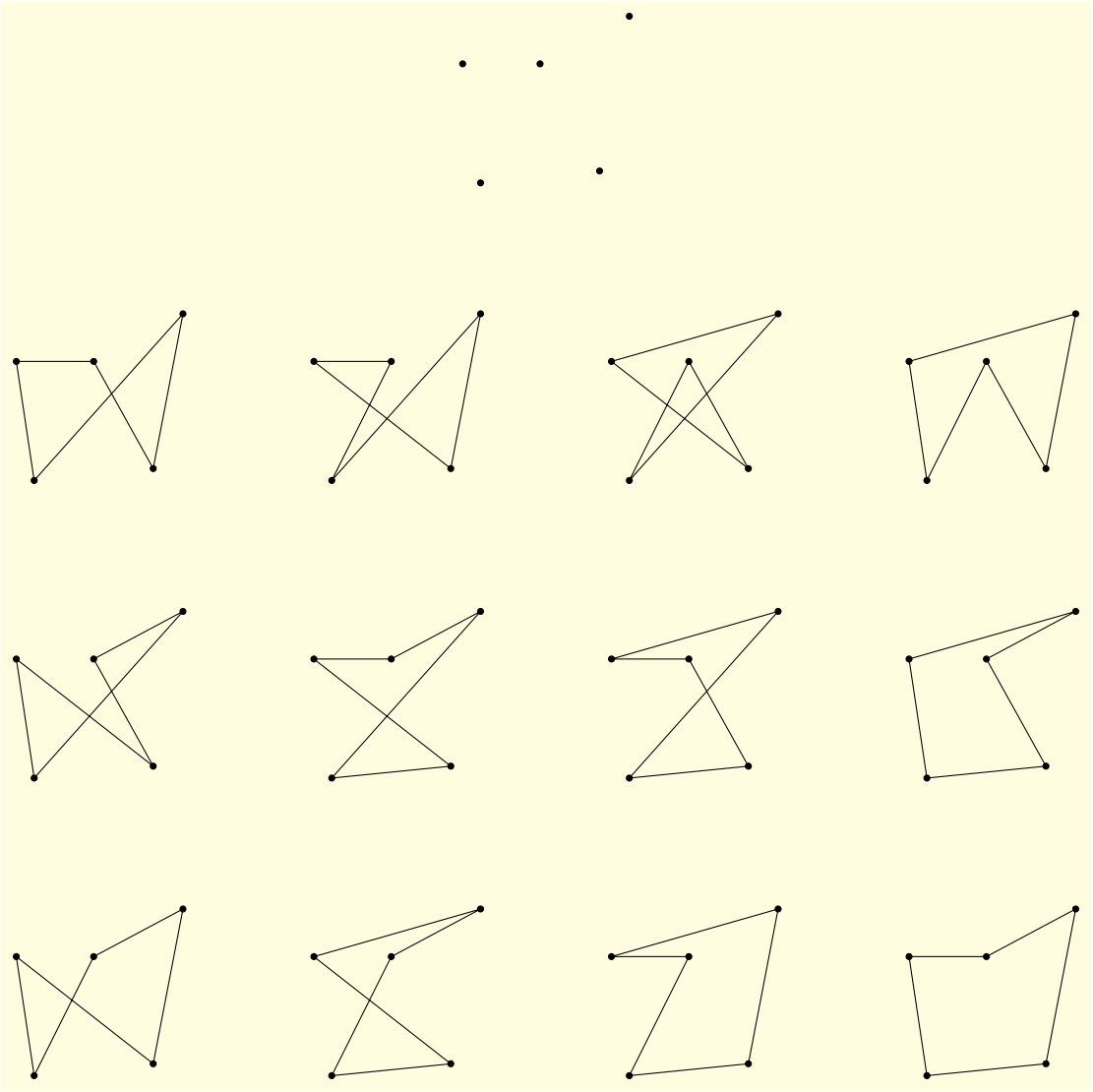
Trouver une solution algorithmique acceptable à ce problème, d'énoncé simple, est plus compliqué qu'il n'y parait. En effet, il fait partie de la classe des problèmes qui sont dits NP-complets. Ce problème peut aussi être étudié dans le cadre de la théorie des graphes : cela consiste à trouver un cycle hamiltonien de coût minimal. Mais nous préférons le voir ici sous un aspect plus géométrique et nous verrons entre autres si ce polygone de périmètre minimal est simple.

2 Notations (pour bien se comprendre)

Dans tout le document, nous utilisons les notations suivantes :

- Soit n un entier naturel supérieur ou égal à 3.
- Soient A_i pour i dans $[\![1\ ;\ n]\!]$ des points. On considére qu'ils sont tous distincts.
- On note $\mathcal{S}$ l'ensemble de points $\{A_i \mid i \in [\![1\ ;\ n]\!]\}$.
- Pour tout polygone $\mathcal{P}$, on note $som(\mathcal{P})$ l'ensemble des sommets de $\mathcal{P}$.
- On note $\Pi(\mathcal{S})$ l'ensemble de tous les polygones $\mathcal{P}$ tels que
  - $som(\mathcal{P}) \subset \mathcal{S}$,
  - et tout point de $\mathcal{S} \setminus som(\mathcal{P})$ appartient à un côté de $\mathcal{P}$.

Exemple 1 *Ici $n = 5$. Pour un ensemble $\mathcal{S}$ donné, voici les tous les polygones de $\Pi(\mathcal{S})$:*



3 Généralités

Commençons par nous intéresser au cardinal de $\Pi(\mathcal{S})$. Il est facile de voir tous les polygones possibles pour $n = 3$ ou $n = 4$. Dans le cas général, la propriété suivante nous donne le résultat.

Propriété 1 *On a*

$$card(\Pi(\mathcal{S})) = \frac{(n-1)!}{2}$$

Preuve *Tous les polygones de $\Pi(\mathcal{S})$ passent par A_1 et peuvent donc s'écrire sous la forme*

$$A_1 A_{\sigma(2)} \dots A_{\sigma(n)}$$

où σ est une permutation de $[\![2\ ;\ n]\!]$. Le nombre de telles permutations est $(n-1)!$. Mais en utilisant cette notation, on compte deux fois chaque polygone, une fois parcouru dans un sens et une fois dans l'autre sens. ■

Comme l'illustre le tableau ci-dessous, le nombre de polygones augmente très rapidement.

n	3	4	5	6	7	10	100	1000
$\frac{(n-1)!}{2}$	1	3	12	60	360	181 440	$\approx 4{,}66 \times 10^{55}$	$\approx 2{,}01 \times 10^{2564}$

Comme on peut le voir sur les figures, certains des polygones, (la majorité ?) ont des côtés qui se croisent. Ce qui nous amène à la définition suivante.

Définition 1 (Polygone simple) *On dit qu'un polygone est simple si pour tous côtés c, c' du polygone, on a*

$$c \text{ et } c' \text{ non consécutifs} \Longrightarrow c \cap c' = \emptyset$$

Plus simplement, un polygone simple est un polygone dont les côtés ne se croisent pas.

Définition 2 (Polygone croisé) *Un polygone croisé est un polygone qui n'est pas simple.*

Théorème 1

Si les points de $\mathcal{S}$ ne sont pas tous alignés, alors $\Pi(\mathcal{S})$ contient au moins un polygone simple.

Preuve *Voir algorithme du code Python 3.* ■

Mais on verra dans la section suivante qu'il existe une preuve plus directe, à l'aide du polygone de périmètre minimal.

4 Le polygone de périmètre minimal

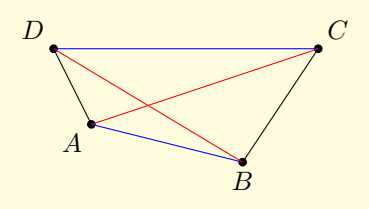
L'ensemble $\Pi(\mathcal{S})$ étant de cardinal fini, il contient bien un polygone qui a un périmètre inférieur ou égal [1] à celui des autres. Ce polygone de périmètre minimal, que peut-on en dire ?

Théorème 2

Dans $\Pi(\mathcal{S})$, le polygone de périmètre minimal est simple.

Pour démontrer ce théorème, nous allons utiliser le lemme suivant.

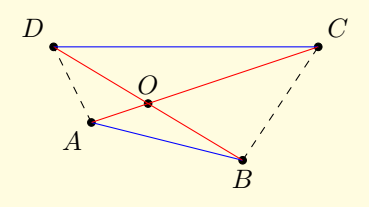
Lemme 1 *Soit $ABDC$ un quadrilatère croisé.*



Alors on a :

$$AB + CD < AC + BD$$

Preuve (du lemme) *Soit O le point de croisement de ce quadrilatère.*



D'après l'inégalité triangulaire dans le triangle OAB, on a

$$AB < AO + BO$$

Et d'après l'inégalité triangulaire dans le triangle OCD, on a

$$CD < OC + OD$$

Puisque $AO + OC = AC$ et $BO + OD = BD$, en additionnant les deux inégalités précédentes, on obtient l'inégalité recherchée. ■

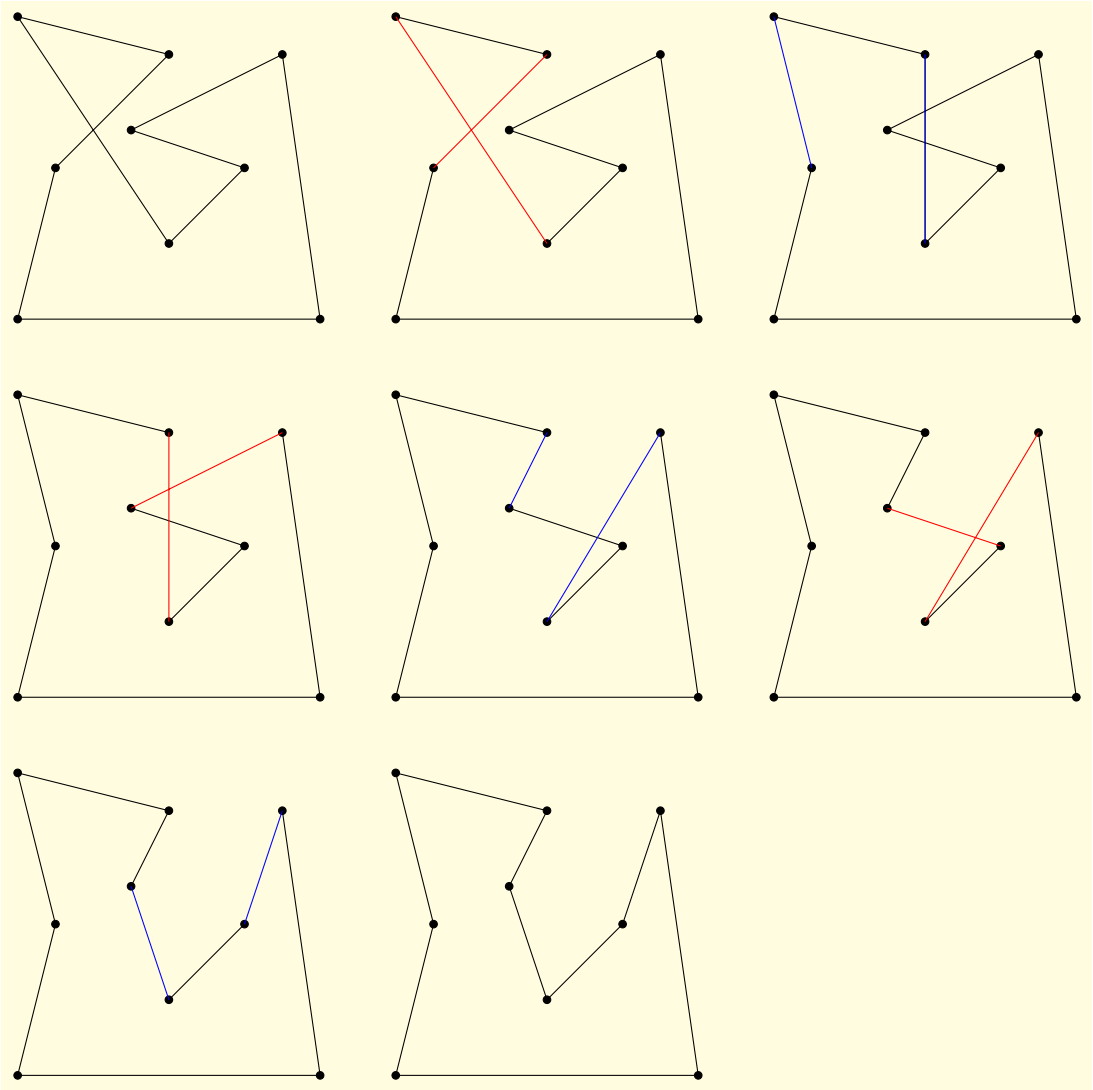
Preuve (du théorème) *Supposons que le polygone de périmètre minimal soit croisé. Alors on peut appliquer un décroisement et obtenir un polygone de périmètre strictement inférieur, ce qui est contradictoire. Donc le polygone de périmètre minimal est simple.* ■

1. Dans certains cas, il peut y avoir plusieurs polygones de périmètre minimal.

5 Décroiser un polygone (ou défaire les noeuds)

Pour simplifier, j'appelle "décroiser un polygone" le fait de remplacer deux côtés qui se croisent par deux côtés qui ne se croisent pas. Cette opération met en jeu 4 points qui sont les sommets d'un quadrilatère dans lequel on remplace les diagonales par deux côtés opposés. Cette opération permet toujours de diminuer strictement le périmètre du polygone. C'est facile à démontrer en utilisant l'inégalité triangulaire.

A chaque fois qu'on décroise un polygone, cela diminue strictement son périmètre. Voici sur un exemple, ce que cela donne. En rouge les côtés qui se croisent que l'on décroise et en bleu les côtés décroisés.



Comme on le voit sur cet exemple, le fait de décroiser deux côtés peut provoquer d'autes croisements. Mais en poursuivant les décroisements, on est sûr d'obtenir un polygone simple.

Ce principe donne d'ailleurs un algorithme pour obtenir un polygone non croisé en partant d'un polygone quelconque croisé (peu importe le nombre de croisements). Il est possible de trouver un polygone (passant évidemment par les mêmes sommets) non croisé de périmètre strictement inférieur. Pour cela, il suffit de décroiser le polygone autant de fois que nécessaire. Et cela se fera un en nombre fini d'étapes (pas de boucle sans fin) car cela diminue strictement le périmètre dans un ensemble fini de périmètres. Mais cet algorithme ne donnera pas forcément le chemin le plus court.

6 Quand la convexité s'en mêle

Prendre en compte la convexité peut donner des résultats intéressants. En effet, il existe un algorithme très connu de complexité faible ($o(n \ln n)$) qui permet d'obtenir l'enveloppe convexe d'un ensemble de points. Ce dernier porte le nom de *Parcours de Graham*.

Définition 3 (polygone convexe) *Un polygone est dit convexe si pour tout couple de points appartenant à l'intérieur du polygone, alors le segment d'extrémités ces deux points est aussi contenu dans le polygone.*

Propriété 2 *Un polygone convexe est simple.*

Définition 4 (enveloppe convexe) *Soit $\mathcal{S}$ un ensemble de points. L'enveloppe convexe de $\mathcal{S}$ est le polygone dont les sommets appartiennent à $\mathcal{S}$ et qui contient tous les points de $\mathcal{S}$.*

Théorème 3

Les points de $\mathcal{S}$ appartenant à l'enveloppe convexe de $\mathcal{S}$ restent dans le même ordre dans tout polygone simple de $\Pi(\mathcal{S})$.

Dans l'optique de la recherche du polygone de périmètre minimal, le résultat précédent est intéressant. En effet, connaissant l'enveloppe convexe de $\mathcal{S}$, on peut alors rechercher le polygone de périmètre minimal parmi un nombre plus restreint de polygones. Et si on a la chance d'avoir tous les points de $\mathcal{S}$ sur l'enveloppe convexe de $\mathcal{S}$, alors la recherche devient triviale, comme l'exprime le résultat suivant.

Théorème 4

Si l'enveloppe convexe de $\mathcal{S}$ contient tous les points de $\mathcal{S}$, alors $\Pi(\mathcal{S})$ ne contient qu'un seul polygone simple et c'est le polygone de périmètre minimal.

7 Algorithmes en Python

Dans les programmes ci-dessous, les points sont représentés par des couples de nombres, les ensembles ou nuages de points par des ensembles (set) de points (donc non ordonnée) et les polygones par une liste de points (donc ordonnée). Par exemple :

```
p1 = (2, 5); p2 = (1, 8); p3 = (4, 3); p4 = (6, 6)
nuage = {p1, p2, p3, p4}
polygone = [p1, p2, p3, p4]
```

Code Python 1 : Obtenir le polygone de périmètre minimal

```
from itertools import permutations
# from math import sqrt # inutile avec pylab
from pylab import *


def perimetre(polygone):
    """ calcule le périmètre d'un polygone """
    n = len(polygone)
    peri = 0
    for k, point in enumerate(polygone):
        (x1, y1) = point
        if k + 1 != n:
            point_suivant = polygone[k+1]
        else:
            point_suivant = polygone[0]
        (x2, y2) = point_suivant
        peri += sqrt((x2 - x1)**2 + (y2 - y1)**2)
    return peri


def polygone_minimal(sommets):
    """ détermine le polygone de périmètre minimal """
    polygone_solution = [point for point in sommets]
    perimetre_solution = perimetre(polygone_solution)
    p0 = sommets.pop()
    chemins = permutations(sommets)
    for chemin in chemins:
        polygone_test = (p0,) + chemin
        perimetre_test = perimetre(polygone_test)
        if perimetre_test < perimetre_solution:
            polygone_solution = polygone_test
            perimetre_solution = perimetre_test
    return polygone_solution


# On commence avec une liste de points (couple de coordonnées)
sommets = {(29,13), (14,17), (6,9), (18,21), (20,6), (34,3), (34,21)}

# Traitement
polygone_solution = polygone_minimal(sommets)
perimetre_solution = perimetre(polygone_solution)

# Affichage de la solution
print("Le polygone de périmètre minimal est", polygone_solution)
print("Et son périmètre est", perimetre_solution)

# Visualisation de la solution
x = [point[0] for point in polygone_solution]
y = [point[1] for point in polygone_solution]
x = x + [x[0]]
y = y + [y[0]]
plot(x, y, 'b-o')
show()
```

Si on se refuse à utiliser la fonction `permutations` du module `itertools`, on peut utiliser le code suivant pour obtenir tous les polygones possibles dans $\Pi(\mathcal{S})$.

Code Python 2 : Obtenir $\Pi(\mathcal{S})$

```
def permutation(sommets):
    """
    Pour un nombre n de sommets, cette fonction donne les (n-1)!/2 listes
    ordonnées de sommets formant un polygone éventuellement croisé.
    Exemple avec sommets = {p1, p2, p3, p4, p5}
    Au debut, permu = [[p1, p2, p3]]
    Puis on ajoute p4, donc
    [p1, p2, p3] donne
    [p1, p4, p2, p3],
    [p1, p2, p4, p3],
    et [p1, p2, p3, p4].
    Pour finir, on ajoute p5, donc
    [p1, p4, p2, p3] donne
    [p1, p5, p4, p2, p3],
    [p1, p4, p5, p2, p3],
    [p1, p4, p2, p5, p3],
    [p1, p4, p2, p3, p5],
    et [p1, p2, p4, p3] donne
    [p1, p5, p2, p4, p3],
    [p1, p2, p5, p4, p3],
    ...
    """
    # On commence par une liste ne comportant que la liste des trois
    # premiers sommets.
    sommets = [point for point in sommets]
    permu = [sommets[:3]]
    # Puis on ajoute les points au fur et à mesure en les intercalant à des
    # places différentes.
    for point in sommets[3:]:
        aux = []
        for liste in permu:
            k = len(liste)
            for i in range(k):
                copie = liste[:]
                copie.insert(i+1, point)
                aux.append(copie)
        permu = aux[:]
    return permu
```

Code Python 3 : Obtenir un polygone simple

```
from math import sqrt, acos, pi
from pylab import *

class Point:
    def __init__(self, p):
        (x, y) = p
        self.x = x
        self.y = y

    def rayon(self):
        return sqrt(self.x**2 + self.y**2)

    def angle(self):
        if self.y >= 0:
            return acos(self.x/self.rayon())
        else:
            return 2*pi - acos(self.x/self.rayon())

    def meme_angle_que(self, autrePoint):
        x1, y1 = self.x, self.y
        x2, y2 = autrePoint.x, autrePoint.y
        if x1*y2 - x2*y1 == 0 and (x1*x2 > 0 or y1*y2 > 0):
            return True
        else:
            return False

    def distance_jusqua(self, autrePoint):
        x1, y1 = self.x, self.y
        x2, y2 = autrePoint.x, autrePoint.y
        return sqrt((x2 - x1)**2 + (y2 - y1)**2)

    def translate(self, u, v):
        self.x += u
        self.y += v


def grouper_par_angle(nuage):
    """ Construit une liste (E) de listes (E_i) de points de même angle """
    E = []
    while nuage:
        p0 = nuage.pop()
        E_i = [p0]
        for point in nuage:
            if point.meme_angle_que(p0):
                E_i.append(point)
                nuage.remove(point)
        E.append(E_i)
    return E


def rayon(point):           # fonction utile pour classer suivant rayon
    return point.rayon()


def angle(ens):
    return ens[0].angle()   # fonction utile pour classer suivant angle


def simple(sommets):
    # origine
    p0 = Point(sommets.pop())
    x0, y0 = p0.x, p0.y
    # les autres points
    nuage = {Point((x, y)) for (x, y) in sommets}
    for point in nuage:             # cela correspond à un changement d'origine
        point.translate(-x0, -y0)
    E = grouper_par_angle(nuage)
    E.sort(key=angle)       # les ensembles E_i sont rangés par angle croissant
    for i in range(len(E)-1):
        p0, p1 = E[i][0], E[i+1][0]
        if p1.x*p0.y - p1.y*p0.x >= 0: #
            E = E[i+1:] + E[:i+1]
```

```
            break                                # Cela ne peut se produire qu'une fois
    for E_i in E[:-1]:                           # Dans chaque E_i, les points sont
        E_i.sort(key=rayon)                      # rangés par rayon croissant
    E[-1].sort(key=rayon, reverse=True)          # sauf dans le dernier E_i
    polygone_simple = [Point((x0,y0))]
    for E_i in E:
        for point in E_i:
            point.translate(x0, y0)
            polygone_simple.append(point)
    return polygone_simple


# On commence avec une liste de points (couple de coordonnées)
sommets = {(2, 3), (1, 17), (10, 15), (18, 21), (-20, 6), (2, 34), (34, 21)}
# Traitement
polygone_simple = simple(sommets)
# Affichage
polygone_simple = [(p.x, p.y) for p in polygone_simple]
print('Voici un polygone simple :', polygone_simple)
# Visualisation
x = [point[0] for point in polygone_simple]
y = [point[1] for point in polygone_simple]
x = x + [x[0]]
y = y + [y[0]]
plot(x, y, 'b-o')
show()
```

8 Webographie

- Page Wikipedia sur le problème du voyageur de commerce
- On polygons enclosing point sets de F. Hurtado, C. Merino, D. Oliveros, J. Urrutia et T. Sakai.
- Simple polygonizations de Erik Demaine : quelques résultats sur la polygonisation.
- Page Wikipedia sur le parcours de Graham