Geometria Obliczeniowa

Dokumentacja projektu

Wyszukiwanie najmniejszego okręgu i prostokąta na chmurze punktów

Jakub Stępak

Grudzień 2016

Spis treści

1 Opis projektu **2**
1.1 Problem . . . 2
1.2 Wykorzystanie otoczki wypukłej . . . 2

2 Wyszukiwanie najmniejszego okręgu **3**
2.1 Obserwacja . . . 3
2.2 Algorytm Appleta - pseudokod . . . 4
2.3 Algorytm Appleta - implementacja . . . 4
2.4 Wynik . . . 6

3 Wyszukiwanie najmniejszego prostokąta **7**
3.1 Obserwacja . . . 7
3.2 Wyszukiwanie najmniejszego prostokąta - pseudokod . . . 8
3.3 Wyszukiwanie najmniejszego prostokąta - implementacja . . . 8

4 Obsługa programu **10**
4.1 Wymagania . . . 10
4.2 Uruchomienie . . . 10
4.3 Struktura repozytorium . . . 11
4.4 Licencja . . . 11

5 Bibliografia **11**

1 Opis projektu

1.1 Problem

Zadanie polegało na napisaniu programu, który, na zadanej chmurze punktów, będzie znajdywał:

- najmniejszy **okrąg** ją zawierący,
- prostokąt ją zawierający o minimalnym **polu**,
- prostokąt ją zawierający o minimalnym **obwodzie**.

1.2 Wykorzystanie otoczki wypukłej

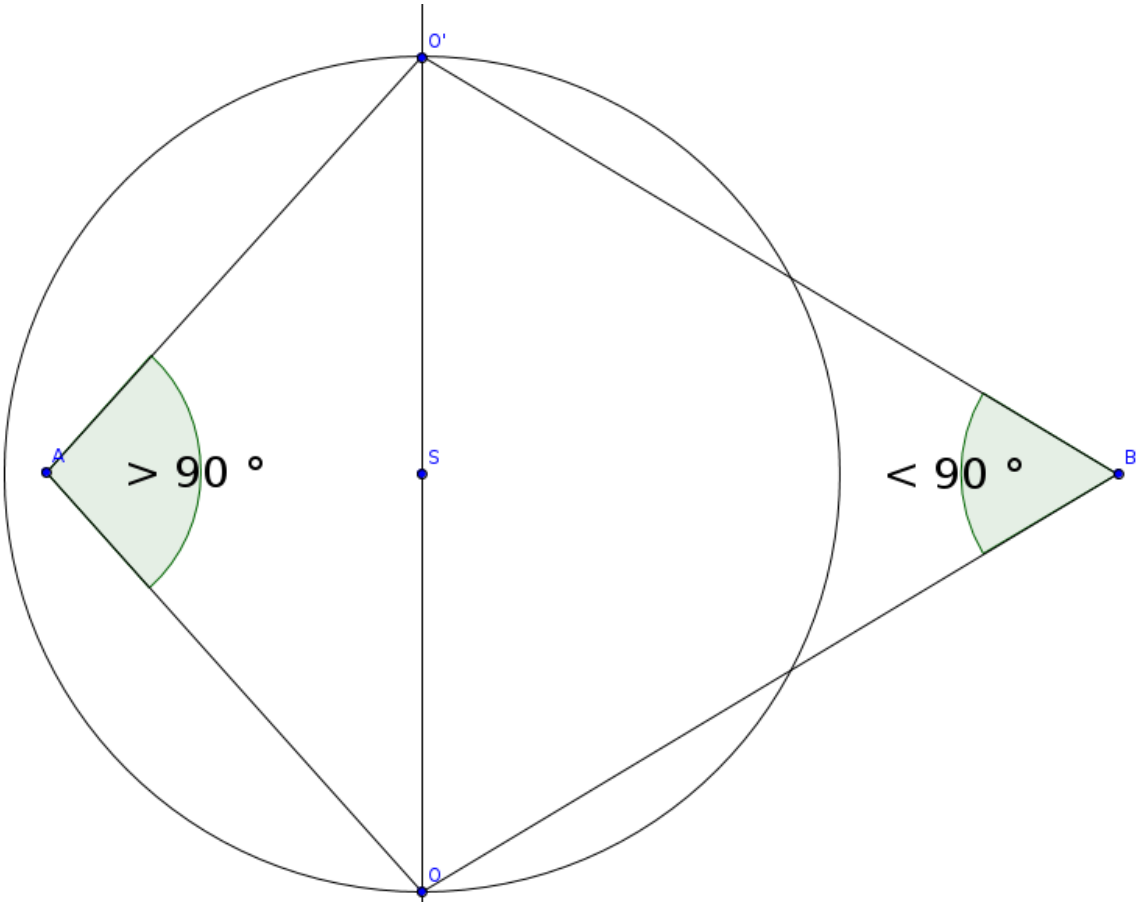
Wszystkie rozwiązania będą opierały się o wyznaczoną otoczkę wypukłą zadanej chmury punktów.

Program posiada moduł do wyznaczania otoczki wypukłej algorytmem Grahama. Istnieje możliwość łatwego dodania innego algorytmu do wyznaczania otoczki wypukłej i użycie go zamiast dostarczonego.

2 Wyszukiwanie najmniejszego okręgu

Do znalezienia najmniejszego okręgu używam **algorytmu Appleta**.

2.1 Obserwacja



Do zrozumienia działania algorytmu Appleta przydatne będzie zauważenie, że jeśli na średnicy okręgu oprzemy kąt, to:

- jeśli wierzchołek kąta znajduje się wewnątrz tego okręgu, jego miara jest większa niż 90°
- jeśli wierzchołek kąta znajduje się na zewnątrz tego okręgu, jego miara jest mniejsza niż 90°

2.2 Algorytm Appleta - pseudokod

1. Wybierzmy dowolną krawędź otoczki $S = [P_1, P_2]$.
2. Dla każdego wierzchołka $P_0 \neq P_1, P_2$, obliczamy $\measuredangle P_1 P_0 P_2$.
3. Najmniejszy znaleziony kąt oznaczmy przez α, a wierzchołek przy którym występuje przez V:
   (a) Jeśli $\alpha > 90^\circ$ to rozwiązaniem jest okrąg opisany na S.
   (b) Jeśli $\alpha < 90^\circ$ sprawdzamy pozostałe kąty $\triangle P_1 V P_2$:
      i. Jeśli żaden nie jest rozwarty, to rozwiązaniem jest okrąg opisany na $\triangle P_1 V P_2$.
      ii. Jeśli któryś z kątów jest rozwarty, krawędź naprzeciwko niego staje się nowym S. Wracamy do punktu 2.

2.3 Algorytm Appleta - implementacja

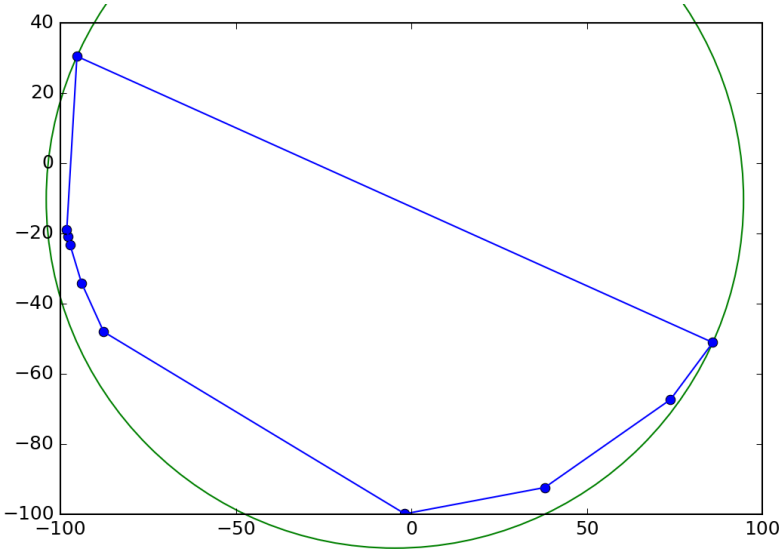
Na kolejnej stronie znajduje się listing metody *run* w klasie *Applet* implementującej algorytm Appleta.

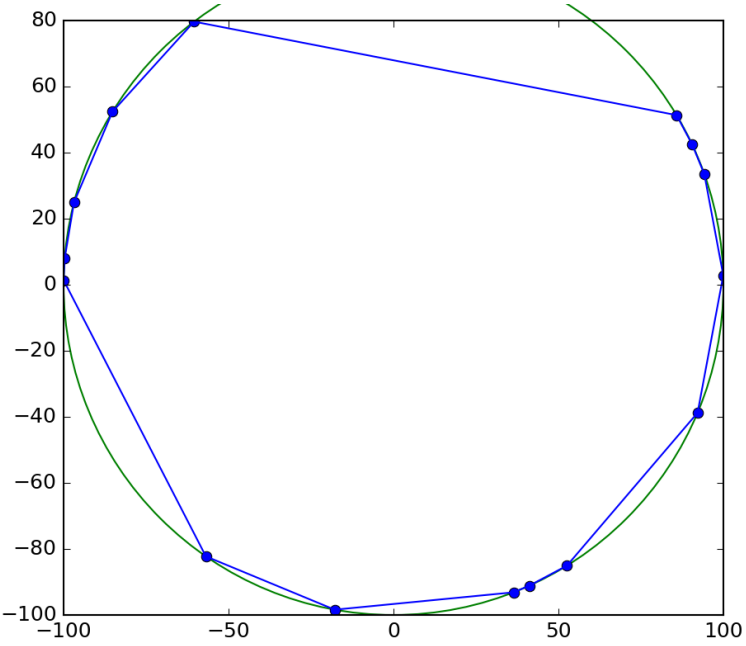
Implementacja algorytmu Appleta:

```
def run(self):
    hull = copy(self.hull)
    p0 = hull[0]
    p1 = hull[1]
    other_points = hull[2:-1]

    while True:
        other_points = sorted(
                other_points,
                key=lambda point: cosinus(p0, point, p1),
                reverse=True
        )

        v = other_points[0]
        cos = cosinus(p0, v, p1)

        self.plot_stage(hull, p0, p1, v)

        if cos < 0:
            v = p0.vector(p1)
            s = p0.add_vector(v.divide(2))

            xy = (s.x, s.y)
            radius = v.divide(2).length
            self.plot_circle(hull, radius, xy)
            break

        if cosinus(p0, p1, v) > 0 and cosinus(v, p0, p1) > 0:
            x = complex(p0.x, p0.y)
            y = complex(p1.x, p1.y)
            z = complex(v.x, v.y)
            w = z - x
            w /= y - x
            c = (x - y) * (w - abs(w) ** 2) / 2j / w.imag - x

            xy = (-c.real, -c.imag)
            radius = abs(c + x)
            self.plot_circle(hull, radius, xy)
            break

        if cosinus(p0, p1, v) < 0:
            other_points.remove(v)
            other_points.append(p1)
            p1 = v
        if cosinus(v, p0, p1) < 0:
            other_points.remove(v)
            other_points.append(p0)
            p0 = v
```

2.4 Wynik

Przykładowy wynik działania programu:

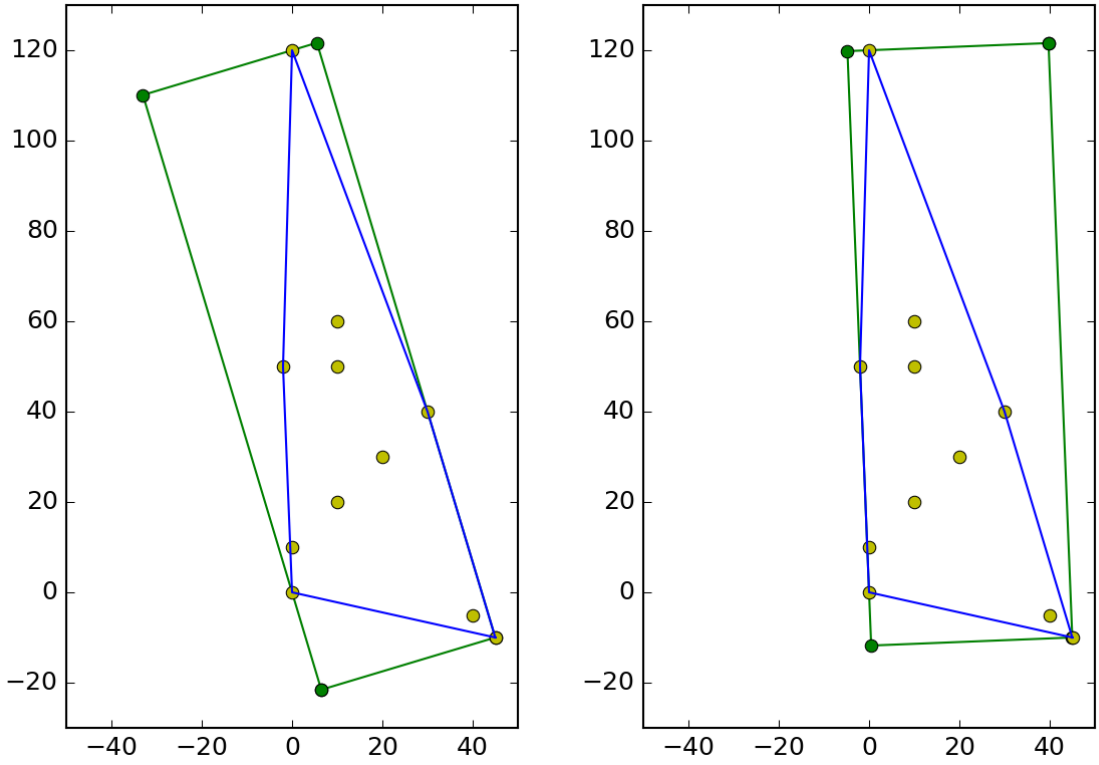




3 Wyszukiwanie najmniejszego prostokąta

3.1 Obserwacja

Najmniejszy prostokąt można definiować jako prostokąt o najmniejszym obwodzie lub jako prostokąt o najmniejszym polu. Nie zawsze (choć dość często) będą to te same prostokąty:



Powyżej po lewej znajduje się prostokąt o najmniejszym polu na tej chmurze punktów, po prawej o najmniejszym obwodzie na tej samej chmurze.

3.2 Wyszukiwanie najmniejszego prostokąta - pseudokod

Powtarzaj dla każdej krawędzi otoczki:

1. „Obróć" otoczką, „kładąc" ją na kolejnej krawędzi na osi OX.
2. Oblicz pole i obwód prostokąta utworzonego przez skrajne punkty (z największą i najmniejszą współrzędną x i y)
3. Zapamiętaj, który prostokąty był najmniejszy.

3.3 Wyszukiwanie najmniejszego prostokąta - implementacja

Na kolejnej stronie znajduje się listing metody *run* w klasie *Rectangle* implementującej algorytm wyszukiwania najmniejszego prostokąta.

Implementacja algorytmu znajdowania najmniejszego prostokąta:

```
def run(self, mode='area'):
    hull = self.hull + [self.hull[1]]

    min_area = 10 ** 10
    min_perimeter = 10 ** 10
    coords = ()

    for i in xrange(len(hull) - 1):
        center = hull[i]
        t = center.vector(Point(0, 0))
        hull = [p.add_vector(t) for p in hull]

        center = hull[i]
        next_point = hull[i + 1]

        cos = cosinus(next_point, center, Point(1, 0))
        hull = [p.rotate(cos) for p in hull]

        self.plot_stage(hull, mode)

        minx, maxx, maxy = 0, 0, 0
        for p in hull:
            if p.x > maxx:
                maxx = p.x
            if p.x < minx:
                minx = p.x
            if p.y > maxy:
                maxy = p.y

        area = maxy * (maxx - minx)
        perimeter = maxy + maxx - minx
        color = 'ro-'
        if (mode == 'area' and area < min_area) or \
           (mode == 'perimeter' and perimeter < min_perimeter):
            min_area = area
            min_perimeter = perimeter
            coords = minx, maxx, maxy, i
            color = 'go-'

        self.plot_stage_with_rectangle(color, hull, maxx, maxy, minx, mode)

    minx, maxx, maxy, i = coords
```

4 Obsługa programu

4.1 Wymagania

Program napisany jest w języku Python 2.7.
Do uruchomienia programu potrzebna jest biblioteka *matplotlib*.

4.2 Uruchomienie

Repozytorium z kodem znajduje się pod adresem `https://github.com/jakubste/geo-project`. Można je stamtąd pobrać jako archiwum i wypakować na swoim komputerze lub sklonować używając Gita.

Każdy z algorytmów można zobaczyć urachamiając skrypty

```
$ python applet.py
$ python rectangle.py
```

Nic nie stoi również na przeszkodzie, żeby użyć algorytmów w innym programie, importując odpowiednie klasy w swoim projekcie. Prawdopodobnie będzie przy tym konieczne usunięcie z działania programu zapisywania poszczególnych kroków programu jako obrazków oraz dostosowanie zwracanych przez program wartości.

```
from applet import Applet


Applet(hull, points).run()
```

```
from rectangle import SmallestRectangle


SmallestRectangle(hull, points).run('area')
SmallestRectangle(hull, points).run('perimeter')
```

4.3 Struktura repozytorium

Poszczególne pliki lub katalogi zawierają:

- appet.py – Algorytm Appleta,
- rectangle.py – algorytm wyznaczania najmniejszego prostokąta,
- hull.py – dostarczony algorytm Grahama do wyznaczania otoczki wypukłej,
- points_generator.py – różne generatory losowych punktów,
- prezentacja/ – prezentację z zajęć wraz z przykładem działania programu (obrazkami wygenerowanymi przez program),
- dokumentacja/ – niniejszy dokument.

4.4 Licencja

Projekt udostępniony jest publicznie na warunkach licencji GNU General Public Licence v.3.

5 Bibliografia

- Bartosz Sądel – Wyznaczanie minimalnego okręgu i prostokąta zawierającego chmurę punktów 2D
  `https://prezi.com/ehey3lpea0dy/wyznaczanie-minimalnego-okregu-i-prostokata-zawierajacego-ch/`