Algorytmy wyznaczania otoczki wypukłej

Dokumentacja techniczna projektu

Adam Dyda
Norbert Wolniak
Grudzień 2020

1. Wprowadzenie.

W ramach projektu zaimplementowaliśmy algorytmy do wyznaczania otoczki wypukłej w przestrzeni dwuwymiarowej. Algorytmy : Przyrostowy , Górna i dolna otoczka, Grahama, Jarvisa, Dziel i rządź, QuickHull i Chana.

2. Informacje techniczne.

Projekt został napisany w oprogramowania Jupyter Notebook przy użyciu proponowanego narzędzia graficznego, które napisane jest w języku Python3 i wykorzystuje bibliotekę MatPlotLib. Poszczególne są implementowane poprzez odpowiadające im funkcje do każdego algorytmu funkcje w dwóch wersjach, zwracającej otoczkę oraz zwracającej wykres i kolejne kroki wykonania algorytmu.

3. Funkcje pomocnicze.

- collection_to_points(collection)
  - Funkcja wykorzystywana do zamiany punktów zawierających się w obiekcie klasy PointsCollection na listę tych punktów
  - Argumenty:
    - collection - kolekcja punktów, klasy PointsCollection
  - Wartość zwracana:
    - lista punktów

- det(p,q,r)
  - Wyznacznik 3 x 3
  - Argumenty:
    -p- pierwszy punkt w przestrzeni dwuwymiarowej
    -q- drugi punkt w przestrzeni dwuwymiarowej
    -r- trzeci punkt w przestrzeni dwuwymiarowej
  - Wartość zwracana:
    Funkcja zwraca wartość wyznacznika

- orientation(p, q, r, e)
    - Orientacja punktu względem wektora <p, q>
    - Argumenty:
      -p- pierwszy punkt w przestrzeni dwuwymiarowej
      -q- drugi punkt w przestrzeni dwuwymiarowej
      -r- trzeci punkt w przestrzeni dwuwymiarowej
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca wartość -1 gdy r jest cw względem wektora <p,q>,
      wartość 1 gdy r jest ccw względem wektora <p,q>
      i wartość 0 gdy r jest współliniowy względem wektora <p,q>

- alpha(p, q, r, e)
    - Funkcja pomocnicza do wyznaczania kolejności punktów potrzebnych do sortowania ccw
    - Argumenty:
      -p- pierwszy punkt w przestrzeni dwuwymiarowej
      -q- drugi punkt w przestrzeni dwuwymiarowej
      -r- trzeci punkt w przestrzeni dwuwymiarowej
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca wartości -1, 0, 1 zależnie od orientacji punktu r

- sorted_ccw(convex_hull, point, e)
    - Funkcja sortująca punkty należące do otoczki w kolejności ccw
    - Argumenty:
      -convex_hull- otoczka wypukła
      -point- punkt w przestrzeni dwuwymiarowej względem którego następuje sortowanie
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca listę posortowanych punktów

- distance(p,q)
    - Funkcja wyznacza kwadrat dystansu między dwoma punktami (służy do porównywania odległości)
    - Argumenty:
      -p- pierwszy punkt w przestrzeni dwuwymiarowej
      -q- drugi punkt w przestrzeni dwuwymiarowej
    - Wartość zwracana:
      - kwadrat odległości między punktami

4. Funkcje pomocnicze: Algorytm Przyrostowy

- lies_inside(convex_hull, point, e)
    - Funkcja sprawdzająca czy punkt leży wewnątrz zbioru
    - Argumenty:
      -convex_hull- otoczka wypukła w kolejności punktów ccw
      -point- punkt w przestrzeni dwuwymiarowej
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca wartość logiczną True/False
- compute_tangent_points(convex_hull, point, e)
    - Funkcja obliczająca indeksy punktów leżących na dwóch stycznych punktu do zbioru
    - Argumenty:
      -convex_hull- otoczka wypukła w kolejności ccw
      -point- punkt w przestrzeni dwuwymiarowej względem którego obliczane są styczne
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca dwuelementową listę indeksów

5. Funkcje pomocnicze: Algorytm Grahama

- delete_collinear(S,p,e)
    - Funkcja służy do usuwania punktów wspóliniowych wzgledem danego punkty
    - Argumenty:
      -S- zbiór punktów do sprawdzenia
      -q- punkt względem którego usuwamy punkty
    - Wartość zwracana:
      - zbiór S po usunięciu punktów

6. Funkcje pomocnicze: QuickHull

- distance_to_line(p,A,B)
    - Funkcja służy do wyznaczania odległości między punktem a odcinkiem
    - Argumenty:
      -p - punkt którego odległość od prostej mierzymy
      -A - punkt będący początkiem odcinka
      -B - punkt będący końcem odcinka
    - Wartość zwracana:

- odległość punktu p od odcinka AB

7. Funkcje pomocnicze: Algorytm Chana

- min_angle(a,b,c)
    - Funkcja służy do sprawdzania który punkt po połączeniu z danym punktem tworzy mniejszy kąt
    - Argumenty:
      -a - punkt względem którego wyznaczamy kąt
      -b - pierwszy punkt testowy
      -B - drugi punkt testowy
    - Wartość zwracana:
      - jeden z punktów wejściowych b lub c
- split(P,m)
    - Funkcja służy do podziału zbioru punktów na zbiory o określonej ilości punktów
    - Argumenty:
      -P -zbiór punktów do podzielenie
      -m - ilość punktów w każdym wynikowym zbiorze
    - Wartość zwracana:
      - Lista zbiorów punktów
- jarvis_next_point(p,S):
    - Funkcja służy wyznaczenia punktu z danego zbioru tworzącego z danym punktem największy kąt
    - Argumenty:
      -p - punkt względem którego szukamy punktu
      -S - zbiór w którym szukamy punktu
    - Wartość zwracana:
      - punkt który tworzy z punktem wejściowym największy kąt

8. Algorytm Przyrostowy

-
    - Funkcja implementująca algorytm przyrostowy
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
      -e- tolerancja dla zera
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do otoczki

9. Algorytm Górna i dolna otoczka

- upper_lower_algorithm(S)
    - Funkcja implementująca algorytm górnej i dolnej otoczki

- Argumenty:
  -S - zbiór punktów na których wyznaczamy otoczkę
- Wartość zwracana:
  Funkcja zwraca listę punktów należących do otoczki

10. Algorytm Grahama

- graham_algorithm(S,e):
  - Funkcja implementująca algorytm Grahama
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca listę punktów należących do otoczki

11. Algorytm Jarvisa

- jarvis_algorithm(S, e)
  - Funkcja implementująca algorytm Jarvisa
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
    -e- tolerancja dla zera
  - Wartość zwracana:
    Funkcja zwraca listę punktów należących do otoczki

12. Algorytm Dziel i rządź

- divide_and_conquer_algorithm(S,k):
  - Funkcja inicjalizująca algorytm Dziel i rządź
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
    -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
  - Wartość zwracana:
    Funkcja zwraca listę punktów należących do otoczki

- divide_and_conquer(points, k):
  - Funkcja dzieląca zbiór na kolejne zbiory A i B
  - Argumenty:
    -points- zbiór punktów do podziału
    -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
  - Wartość zwracana:
    Funkcja zwraca wartość zwracaną funkcji merge_convex_hulls(A,B)

- merge_convex_hulls(A,B):
  - Funkcja łącząca dwie otoczki wypukłe A i B
  - Argumenty:
    -A- zbiór punktów otoczki leżących po lewej stronie
    -B- zbiór punktów otoczki leżących po prawej stronie

- Wartość zwracana:
  Funkcja zwraca listę punktów należących do wyznaczonej tymczasowej otoczki wypukłej

13. QuickHull

- quickhull(S)
  - funkcja inicjalizująca algorytm quickhull do wyznaczania otoczki wypukłej
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej
- quickhull_step(A,B,S)
  - funkcja wyznaczające kolejne punkty otoczki wykonywana rekurencyjnie
  - Argumenty:
    -A - jeden ze skrajnych punktów w rozpatrywanym zbiorze
    -B - drugi ze skrajnych punktów w rozpatrywanych zbiorze
    -S - rozpatrywany zbiór
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej w rozpatrywanym zbiorze

14. Algorytm Chana

- chan_algorithm(S)
  - funkcja zwracająca otoczke wypukła za pomocą algorytmu Chana
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
  - Wartość zwracana:
    -Lista punktów należących do otoczki wypukłej

15. Algorytm Przyrostowy Wizualizacja

- plot_incremental_algorithm(S, e):
  - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
  - Argumenty:
    -S - zbiór punktów na których wyznaczamy otoczkę
    -e- tolerancja dla zera
  - Wartość zwracana:
    -Obiekt klasy Plot() służący do wizualizacji

16. Algorytm Górna i dolna otoczka Wizualizacja

- plot_upper_lower_algorithm(S):
  - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
  - Argumenty:

-S - zbiór punktów na których wyznaczamy otoczkę
    - Wartość zwracana:
      -Obiekt klasy Plot() służący do wizualizacji

17. Algorytm Grahama Wizualizacja

- plot_graham_algorithm(S)
    - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
    - Wartość zwracana:
      -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot

18. Algorytm Jarvisa Wizualizacja

- plot_jarvis_algorithm(S)
    - Funkcja służąca do rysowania wykresu wyznaczonej otoczki i wizualizacji kolejnych kroków algorytmu
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
    - Wartość zwracana:
      -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot

19. Algorytm Dziel i rządź Wizualizcja

- plot_divide_and_conquer_algorithm(S,k):
    - Funkcja inicjalizująca algorytm Dziel i rządź do rysowania otoczki wypukłej
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
      -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
    - Wartość zwracana:
      -rysunek zbioru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot

- plot_divide_and_conquer(points, k, plot, S):
    - Funkcja dzieląca zbiór na kolejne zbiory A i B
    - Argumenty:
      -points- zbiór punktów do podziału
      -k- stała k oznaczająca maksymalny rozmiar rozdzielonego zbioru
      -plot- obiekt klasy Plot()
      -S- zbiór początkowy punktów

    - Wartość zwracana:
      Funkcja zwraca wartość zwracaną funkcji plot_merge_convex_hulls(A,B)

- plot_merge_convex_hulls(A,B, plot, S):
    - Funkcja łącząca dwie otoczki wypukłe A i B
    - Argumenty:
      -A- zbiór punktów otoczki leżących po lewej stronie
      -B- zbiór punktów otoczki leżących po prawej stronie
      -plot- obiekt klasy Plot()
      -S- zbiór początkowy punktów
    - Wartość zwracana:
      Funkcja zwraca listę punktów należących do wyznaczonej tymczasowej otoczki wypukłej

20. QuickHull Wizualizacja

- plot_quickhull(S)
    - funkcja inicjalizująca algorytm quickhull do rysowania otoczki wypukłej
    - Argumenty:
      -S - zbiór punktów na których wyznaczamy otoczkę
    - Wartość zwracana:
      -rysunek zboru oraz wykresu otoczki z poszczególnymi krokami wykonania algorytmu, obiekt klasy Plot
- plot_quickhull_step(points,A,B,S,plot)
    - funkcja służąca do rysowania otoczki wypukłej wyznaczonej na danym zbiorze, wykonywana rekurencyjnie
    - Argumenty:
      -points- pierwotny zbiór punktów na których wyznaczmy otoczkę
      -A - jeden ze skrajnych punktów w rozpatrywanym zbiorze
      -B - drugi ze skrajnych punktów w rozpatrywanych zbiorze
      -S - rozpatrywany zbiór
      -plot - obiekt klasy Plot, rysunek na którym rysujemy wykres otoczki
    - Wartość zwracana:
      -Lista punktów należących do otoczki w rozpatrywanym zbiorze

21. Algorytm Chana Wizualizacja

- plot_chan_algorithm(S)
    - funkcja służy do rysowania otoczki wyznaczonej na danym zbiorze punktów za pomocą algorytmu Chana
    - Argumenty:
      -S - zbiór punktów na którym wyznaczamy otoczkę
    - Wartość zwracana:
      -obiekt typu Plot, rysunek zawierający wykres wyznaczonej otoczki

22. Porównanie prędkości algorytmów

- Algorytm Przyrostowy
  Dla n punktów wykonujemy poszukiwanie punktów leżących na zewnątrz dotychczasowej otoczki w czasie O(n) i dla każdego z tych punktów znajdujemy styczne do otoczki w czasie O(logn).
  Daje nam to wynikową złożoność O(nlogn)
- Algorytm Górnej i Dolnej otoczki
  Dla n punktów wykonujemy sortowanie w czasie O(nlogn), operacje znajdowania górnej i dolnej otoczki w czasie O(2n).
  Daje nam to wynikową złożoność O(nlogn)
- Algorytm Grahama
  Dla n punktów wykonujemy sortowanie w czasie O(nlogn), operacje wyszukania minimalnego punktu w czasie O(n) oraz iteracja po n-3 punktach w czasie O(n).
  Daje nam to wynikową złożoność O(nlogn).
- Algorytm Jarvisa
  Dla h punktów gdzie h to wielkość otoczki wykonujemy operacje znajdowania następnego punktu należącego do otoczki, operacja ta ma złożoność O(n) daje nam to wynikową złożoność O(n*h) jednak w praktyce w większości przypadków
  h << n więc rzadko złożoność zdegraduje się do O(n^2)
- Algorytm Dziel i Rządź
  Dla n punktów dzielimy zbiór na dwa podzbiory o wielkości n/2 każdy aż do uzyskania podzbioru p o wielkości danej stałej k. Znajdujemy otoczkę danego podzbioru w czasie stałym ze względu na rząd wielkości k w stosunku do rzędu wielkości początkowego problemu n. Łączymy otrzymane otoczki w łącznym czasie O(logn).
  Wynikowa złożoność wynosi O(nlogn).
- Algorytm QuickHull
  Złożoność jak i działanie algorytmu QuickHull są podobne do znanego algorytmu QuickSort, dzielimy zbiór na pół i w każdym rekurencyjnym wywołaniu znajdujemy ekstremalny punkt w czasie O(n).
  W najlepszym wypadku dostaniemy złożoność O(n logn)
  W najgorszym wypadku gdy zbiory na które dzielimy punkty są niezbalansowane algorytm działa w czasie O(n^2)
- Algorytm Chana
  Na złożoność algorytmu Chana składa się:
  Wykonanie algorytmu Grahama dla wszystkich grup punktów: O(nlogm)
  Znalezienie stycznej z punktu do otoczki: O(log n)
  Koszt algorytmu Jarvisa dla r otoczek: O((hn/m) log m)

Wyniki dla testów prędkości

■ - najszybszy

■ - niewiele wolniejszy

Zbiór A

Liczba punktów	50	100	200	500	1000
Algorytm przyrostowy	0.001	0.0019	0.005	0.012	0.036
Algorytm gorna dolna otoczka	**0.00095**	**0.00099**	**0.0019**	**0.0030**	**0.0079**
Algorytm Grahama	0.00099	0.00099	0.0019	0.007	0.013
Algorytm Jarvisa	0.00099	0.006	0.0039	0.006	0.023
Algorytm dziel i rządź	0.001	0.0019	0.0029	0.01	0.025
Algorytm QuickHull	0.001	0.00099	0.002	0.007	0.0079
Algorytm Chana	0.001	0.0029	0.0035	0.012	0.041

Zbiór B

Liczba punktów	50	100	200	500	1000
Algorytm przyrostowy	0.014	0.034	0.14	0.81	2.99
Algorytm gorna dolna otoczka	**0.00099**	**0.00099**	**0.0029**	**0.0060**	**0.010**
Algorytm Grahama	0.0019	0.0020	0.0039	0.018	0.034
Algorytm	0.0090	0.038	0.11	0.761	3.01

Jarvisa					
Algorytm dziel i rządź	0.0039	0.0039	0.017	0.035	0.072
Algorytm QuickHull	0.0030	0.0040	0.011	0.039	0.080
Algorytm Chana	0.013	0.050	0.17	1.27	4.35

Zbiór C

Liczba punktów	50	100	200	500	1000
Algorytm przyrostowy	0.00096	0.0019	0.0060	0.011	0.025
Algorytm gorna dolna otoczka	**0.0**	**0.0030**	**0.0030**	**0.0049**	**0.0049**
Algorytm Grahama	0.0	0.00099	0.0040	0.0060	0.014
Algorytm Jarvisa	0.0010	0.00099	0.0019	0.0050	0.017
Algorytm dziel i rządź	0.00099	0.00099	0.0020	0.0049	0.014
Algorytm QuickHull	0.0	0.00096	0.00096	0.0059	0.0090
Algorytm Chana	0.0020	0.0049	0.0069	0.019	0.032

Zbiór D

Liczba punktów	50	100	200	500	1000
Algorytm przyrostowy	0.001	0.004	0.0079	0.02	0.027
Algorytm gorna dolna otoczka	**0.003**	**0.003**	**0.0039**	**0.009**	**0.011**
Algorytm Grahama	0.001	0.0019	0.008	0.011	0.023

Algorytm Jarvisa	0.0019	0.0029	0.003	0.0069	0.015
Algorytm dziel i rządź	0.0028	0.0059	0.0059	0.012	0.024
Algorytm QuickHull	0.0	0.0039	0.003	0.0059	0.014
Algorytm Chana	0.006	0.006	0.014	0.031	0.063

23. Bibliografia

- https://www.youtube.com/watch?v=EzeYI7p9MjU
- https://en.wikipedia.org/wiki/Chan%27s_algorithm
- https://pl.wikipedia.org/wiki/Quickhull
- https://cs.stackexchange.com/questions/97194/is-binary-search-really-required-in-chans-convex-hull-algorithm
- Mark de Berg - “Computational Geometry - Algorithms and Applications”