

Porównanie metod konstrukcji diagramu Woronoja

Bartosz Hanc, Jakub Pawlina

3 stycznia 2023

Spis treści

0.1	Część techniczna	1
0.1.1	Wprowadzanie danych wejściowych	1
0.1.2	Wizualizacja wyników	1
0.2	Opis problemu	2
0.3	Konstrukcja diagramu Woronoja jako grafu dualnego do triangulacji Delaunay'a wyznaczonej algorytmem Bowyer-Watsona	2
0.3.1	Opis algorytmu	2
0.3.2	Wydajność zaimplementowanego algorytmu	4
0.3.3	Graficzna prezentacja wyników działania algorytmu	4
0.3.4	Wizualizacja etapów działania algorytmu	5
0.4	Algorytm przybliżony brute force	5
0.4.1	Opis algorytmu	5
0.4.2	Graficzna prezentacja wyników działania algorytmu	6

0.1 Część techniczna

Projekt został wykonany w środowisku Jupyter Notebook w języku Python z wykorzystaniem dostarczonego na laboratoriach narzędzia graficznego. Wszystkie dostępne funkcjonalności programu zostały opisane w notatniku Jupyter, natomiast ich uruchomienie jest możliwe poprzez uruchomienie odpowiedniej komórki w notatniku. Do implementacji samych algorytmów wykorzystano jedynie bibliotekę standardową Pythona.

0.1.1 Wprowadzanie danych wejściowych

Danymi wejściowymi jest lista punktów reprezentowana przez pary typu `(float, float)`. Oprócz bezpośredniego wpisania listy punktów możliwe jest zadawanie punktów za pomocą myszki, korzystając z interfejsu graficznego. Wprowadzone w taki sposób dane można następnie zapisać do pliku `.json` poprzez uruchomienie odpowiedniej komórki. Możliwe jest następnie wczytanie danych wejściowych z pliku `.json` korzystając z funkcji `loadPoints("name.json")` przez podanie nazwy pliku znajdującego się w tym samym folderze co notatnik Jupyter. Dodatkowo istnieje również opcja wygenerowania n losowych punktów należących do zadanego obszaru płaszczyzny $[a; b] \times [a; b]$ poprzez wywołanie funkcji `genRndPoints(n, a, b)`.

0.1.2 Wizualizacja wyników

Notatnik Jupyter zawiera również zaimplementowane funkcje `showResults1(points)` oraz `showResults2(points)` wizualizujące wyniki działania odpowiednio pierwszego i drugiego zaimplementowanego algorytmu.

Dostępna jest również interaktywna wizualizacja algorytmu prezentująca graficznie kolejne etapy jego działania. Wizualizację można uruchomić poprzez uruchomienie odpowiedniej komórki w notatniku.

0.2 Opis problemu

Diagramem Woronoja generowanym przez zbiór punktów $S := \{p_1, \dots, p_n\} \subset \mathbb{R}^2$ nazywamy podział płaszczyzny \mathbb{R}^2 na obszary H_1, \dots, H_n parami rozłączne, takie że dla każdego H_i zachodzi $\forall p \in H_i : d(p, p_i) = \min \{d(p, p_j) \mid p_j \in S\}$, gdzie $d : \mathbb{R}^2 \times \mathbb{R}^2 \mapsto \mathbb{R}_+$ jest metryką na płaszczyźnie \mathbb{R}^2 .

Celem projektu było zaimplementowanie dwóch różnych algorytmów wyznaczających diagram Woronoja zadanej chmury punktów na płaszczyźnie euklidesowej.

0.3 Konstrukcja diagramu Woronoja jako grafu dualnego do triangulacji Delaunay'a wyznaczonej algorytmem Bowyer-Watsona

0.3.1 Opis algorytmu

Jako jedną z dwóch wymaganych metod konstrukcji diagramów Woronoja wybrano metodę konstrukcji korzystającą z dualności między triangulacją Delaunay'a, a diagramem Woronoja. Schemat postępowania jest wówczas następujący: wyznaczamy najpierw triangulację Delaunay'a chmury punktów, a następnie na jej podstawie tworzymy krawędzie diagramu Woronoja. Do wyznaczenia triangulacji Delaunay'a wykorzystano algorytm Bowyer-Watsona. Ideę zaproponowanej metody można przedstawić w postaci elementarnego pseudokodu*:

```
funkcja BowyerWatson:
    points := lista zadanych punktów
    triangles := pusty zbiór trójkątów triangulacji

    Znajdź super-prostokąt pokrywający całkowicie chmurę punktów.
    Dodaj do triangles dwa super-trójkąty powstałe z podziału
    super-prostokąta jedną z jego przekątnych.

    Dla każdego punktu p z points:
        Znajdź wszystkie trójkąty T, takie że okrąg opisany na T zawiera
        punkt p i dodaj je do zbioru badTriangles.

        Znajdź krawędzie wielokąta, który powstałby po usunięciu trójkątów
        z badTriangles i dodaj je do zbioru polygon.

    Usuń z triangles wszystkie trójkąty znajdujące się w badTriangles.

    Dokonaj triangulacji powstałej wielokątnej wnęki polygon.

    Usuń z triangles wszystkie trójkąty, które zawierają wierzchołek
    super-trójkąta.

    Oblicz diagram Woronoja jako graf dualny do triangulacji Delaunaya
    zawartej w triangles.
```

Dokładny opis implementacji w języku Python poszczególnych fragmentów powyższego pseudokodu został opisany poniżej.

• Wykorzystane struktury danych

- Wejściowe punkty są przechowywane w liście `points`, która zawiera pary typu `(float, float)` określające współrzędne kolejnych punktów
- Struktura `triangles` przechowuje trójki postaci `(int, int, int)` określające wierzchołki trójkąta jako indeksy punktów w liście `points`. Struktura `triangles` została zaimplementowana jako pythonowy `set()`.

*Przedstawiony pseudokod z niewielkimi modyfikacjami został zaczerpnięty z artykułu „[Bowyer-Watson algorithm](#)” na Wikipedii.

- Struktura `edges` przechowuje pary typu key-value, gdzie kluczami są pary typu `(int, int)` określające krawędź triangulacji jako indeksy punktów w liście `points`, natomiast wartościami są zbiory zawierające trójkąty, które zawierają krawędź będącą kluczem. Trójkąty te są reprezentowane tak jak w strukturze `triangles` tj. w postaci trójek typu `(int, int, int)`. Struktura `edges` została zaimplementowana jako pythonowy słownik.
- **Konstrukcja superprostokąta pokrywającego chmurę punktów.** Superprostokąt obliczamy poprzez znalezienie maksymalnej i minimalnej współrzędnej x i y chmury punktów, obliczenie wielkości $l := \max(|y_{\max} - y_{\min}|, |x_{\max} - x_{\min}|)$, a następnie odpowiednio dodanie i odjęcie l do/od x_{\max} , y_{\max} i x_{\min} , y_{\min} . Parametr l określa wielkość obszaru, do którego ograniczamy obliczany diagram Woronoja.
- **Poszukiwanie trójkątów, których okrąg opisany zawiera dany punkt p .** Dla każdego trójkąta $T(A, B, C)$ znajdującego się aktualnie w zbiorze `triangles` obliczamy współrzędne (S_x, S_y) środka okręgu opisanego na nim, korzystając ze wzorów

$$S_x = \frac{1}{D} [(A_x^2 + A_y^2)(B_y - C_y) + (B_x^2 + B_y^2)(C_y - A_y) + (C_x^2 + C_y^2)(A_y - B_y)]$$

$$S_y = \frac{1}{D} [(A_x^2 + A_y^2)(C_x - B_x) + (B_x^2 + B_y^2)(A_x - C_x) + (C_x^2 + C_y^2)(B_x - A_x)]$$

gdzie $D := 2[A_x(B_y - C_y) + B_x(C_y - A_y) + C_x(A_y - B_y)]$ oraz promień tego okręgu korzystając ze wzoru

$$R = \frac{abc}{4\sqrt{s(s-a)(s-b)(s-c)}},$$

gdzie a, b, c to długości boków trójkąta, a $s := \frac{1}{2}(a + b + c)$. Jeśli odległość punktu p od środka `centre(A, B, C)` jest mniejsza lub równa R to dodajemy trójkąt $T(A, B, C)$ do zbioru `badTriangles` (`badTriangles` jest implementowany jako pythonowy `set()`). Równość sprawdzana jest przez warunek $|\text{dist}(p, \text{centre}) - R| < \text{err}$, gdzie `err` to tolerancja dla zera.

- **Poszukiwanie krawędzi wielokątnej wnęki.** Dla każdego trójkąta $T(A, B, C)$ znajdującego się w zbiorze `badTriangles` przypisujemy $a, b, c = (A, B), (B, C), (C, A)$, a następnie dla każdej krawędzi e z (a, b, c) obliczamy moc zbioru $(\text{edges}[e] \cap \text{badTriangles}) - \{T\}$ (ponieważ $\#\text{edges}[e] \leq 2$ oraz złożoność znalezienia części wspólnej $s \cap t$ to $O(\min(\#s, \#t))$, a złożoność znalezienia różnicy $s - t$ to $O(\#s)$, więc obliczenie mocy zbioru w powyższy sposób zajmuje $O(1)$ operacji). Jeśli moc powyższego zbioru wynosi 0 to dana krawędź e nie jest krawędzią wspólną żadnych dwóch trójkątów ze zbioru `badTriangles`, czyli jest krawędzią poszukiwanego wielokąta i dodajemy ją do zbioru krawędzi `polygon` (`polygon` jest implementowany jako pythonowy `set()`)
- **Usuwanie nieprawidłowych trójkątów.** Dla każdego trójkąta $T(A, B, C)$ znajdującego się w zbiorze `badTriangles` przypisujemy $a, b, c = (A, B), (B, C), (C, A)$. Usuwamy T ze zbioru `triangles` korzystając z metody `.remove()` (o złożoności czasowej $O(1)$), a następnie aktualizujemy strukturę `edges`: dla każdej krawędzi e z (a, b, c) usuwamy trójkąt T ze zbioru `edges[e]` korzystając z metody `.remove()` i jeśli po usunięciu zbiór ten jest pusty usuwamy klucz e ze słownika `edges` korzystając z metody `del` (o złożoności $O(1)$).
- **Triangulacja wielokątnej wnęki.** Dla każdej krawędzi $e = (q, r)$ ze zbioru `polygon` dodajemy do zbioru `triangles` trójkąt $T(p, q, r)$ i aktualizujemy strukturę `edges`: przypisujemy $a, b, c = (p, q), (q, r), (r, p)$, a następnie dla każdej krawędzi e z (a, b, c) wykonujemy:
 - jeśli klucz e istnieje już w `edges` (sprawdzamy to w $O(1)$, korzystając z metody `.get()`) to dodajemy do zbioru `edges[e]` trójkąt $T(p, q, r)$;
 - jeśli klucz e nie istnieje, to go tworzymy i wstawiamy do pustego zbioru `edges[e]` trójkąt $T(p, q, r)$.

- **Usuwanie trójkątów zawierających wierzchołek początkowego supertrójkąta.** Dla każdego trójkąta $T(A, B, C)$ znajdującego się w zbiorze `triangles` sprawdzamy, czy A lub B lub C są wierzchołkami supertrójkąta. Jeśli tak to dodajemy T do zbioru `toRemove`. Następnie dla każdego trójkąta T ze zbioru `toRemove` usuwamy T ze zbioru `triangles` korzystając z metody `.remove()`.
- **Obliczenie diagramu Woronoja dualnego do wyznaczonej triangulacji Delaunay’a.** Diagram Woronoja jest reprezentowany przez listę `voronoi` zawierającą krawędzie go tworzące. Dla każdej krawędzi e będącej kluczem w `edges` przypisujemy $Ts = edges[e] \cap triangles$ ($O(1)$). Jeśli $\#Ts = 2$ to krawędź e jest krawędzią wspólną dwóch trójkątów $T_1(A_1, B_1, C_1)$, $T_2(A_2, B_2, C_2)$ zatem do listy `voronoi` dodajemy krawędź $[centre(A_1, B_1, C_1), centre(A_2, B_2, C_2)]$. Jeśli $\#Ts = 1$ to krawędź e należy do otoczki wypukłej chmury punktów i krawędź diagramu Woronoja jest fragmentem prostej prostopadłej do e . W celach wizualizacji możemy jednak utworzyć w tym przypadku krawędź pomiędzy trójkątem T_1 z Ts i trójkątem T_2 zawierającym wierzchołek supertrójkąta, przy czym oba trójkąty T_1 i T_2 znajdują się w zbiorze `edges[e]`.

Ze względu na przeszukiwanie wszystkich trójkątów w strukturze `triangles` w celu utworzenia zbioru `badTriangles` złożoność czasowa opisanego algorytmu wynosi $O(n^2)$, gdzie n to liczba wprowadzonych punktów. Taki sposób znajdowania nieprawidłowych trójkątów jest jednak bardzo czytelny w kodzie w języku Python i wymaga istotnie mniej implementacji. Sama procedura przekształcania triangulacji Delaunay’a w dualny do niej diagram Woronoja jest realizowana w czasie liniowym.

0.3.2 Wydajność zaimplementowanego algorytmu

Złożoność czasowa zaimplementowanego algorytmu wynosi $O(n^2)$. W Tabeli 1 zebrano uśrednione czasy działania programu (bez wizualizacji) dla wygenerowanych losowo zbiorów punktów o coraz większej liczności. Testy przeprowadzono na komputerze z procesorem Intel Core i5-8600k 3.60 GHz i systemem operacyjnym Windows 11. Dla $n < 1000$ program oblicza diagram Woronoja w rozsądnym czasie, jednak ze względu na odbiegającą od wzorcowej złożoność nie może zostać wykorzystany do przetwarzania bardzo dużych chmur punktów. Jednocześnie ze względu na krótką i przejrzystą implementację nadaje się do wizualizacji algorytmu służącej objaśnieniu działania algorytmu.

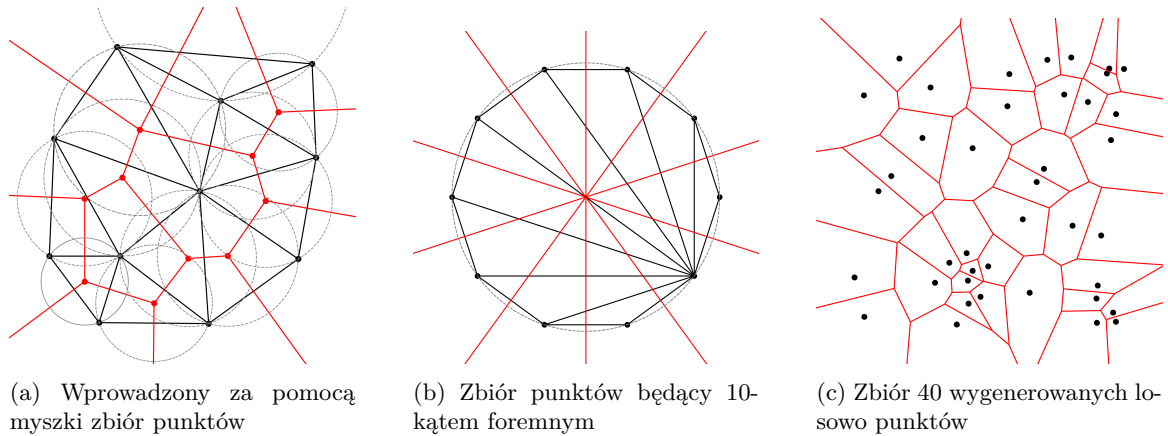
Liczba punktów n	Czas [s]
10	0.002
50	0.013
100	0.047
500	1.08
1000	4.24
1500	9.57

Tabela 1: Czasy działania programu dla losowo wygenerowanych zbiorów punktów

0.3.3 Graficzna prezentacja wyników działania algorytmu

Na Rysunku 1 przedstawiono wygenerowane wizualizacje wyznaczonych diagramów Woronoja dla trzech różnych zbiorów punktów. Czarne punkty oznaczają wprowadzoną chmurę punktów, czarne krawędzie to krawędzie triangulacji, czerwone punkty to środki okręgów opisanych na trójkątach z triangulacji, natomiast czerwone krawędzie to krawędzie diagramu Woronoja.

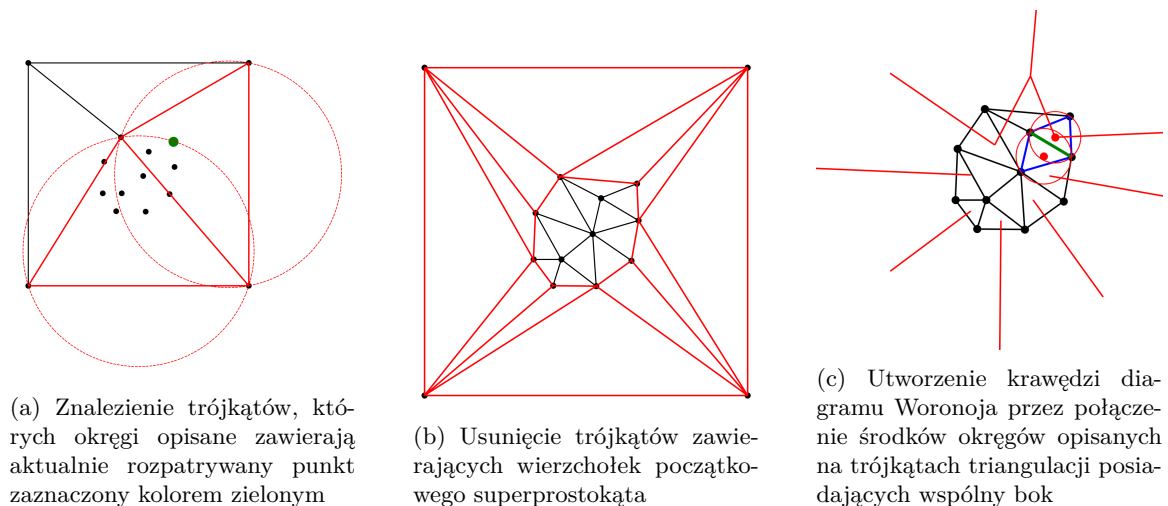
Zaimplementowany algorytm przetestowano na różnych zbiorach punktów. W szczególności sprawdzono, czy w przypadku zbioru punktów, dla których więcej niż 3 z nich leżą na jednym okręgu nie występują problemy związane z testem przynależności do koła opisanego. Nie stwierdzono występowania żadnych problemów i jak widać na zamieszczonym przykładzie: dla 10-kąta foremego triangulacja Delaunay’a i diagram Woronoja zostały wyznaczone poprawnie.



Rysunek 1: Wizualizacja diagramu Woronoja dla trzech różnych zbiorów punktów

0.3.4 Wizualizacja etapów działania algorytmu

Pierwsza część wizualizacji prezentuje algorytm wyznaczania triangulacji Delaunay'a zadanej chmury punktów, natomiast druga – przekształcenie uzyskanej triangulacji w dualny do niej diagram Woronoja. Na Rysunku 2 zamieszczono wybrane fragmenty wizualizacji algorytmu wraz z opisem.



Rysunek 2: Wybrane fragmenty wizualizacji etapów algorytmu

0.4 Algorytm przybliżony brute force

0.4.1 Opis algorytmu

Jako drugą metodę zaimplementowano algorytm brute force wyznaczający w przybliżony sposób wieloboki Woronoja dla zadanej chmury punktów. Algorytm wprowadza na płaszczyźnie dyskretną kratę \mathbb{Z}_+^2 i dla każdego punktu kratowego znajduje punkt z zadanej chmury, dla którego odległość (w sensie metryki euklidesowej) jest minimalna. Algorytm zaimplementowany w języku Python jest niezwykle krótki, dlatego zamieszczono go poniżej w całości.

```
def ApproxVoronoi(points, N): #  $O(n \cdot N^2)$ 
    n = len(points)
    P = [i for i in range(n)]
    mx, my = min(points, key=lambda x: x[0])[0], min(points, key=lambda x: x[1])[1]
```

```

Mx, My = max(points, key=lambda x: x[0])[0], max(points, key=lambda x: x[1])[1]
a = .15 * max(Mx - mx, My - my)
mx -= a
my -= a
Mx += a
My += a
hx, hy = (Mx - mx) / N, (My - my) / N
grid = [[min(P, key=lambda x: d(points[x], (mx+i*hx, my+j*hy))) for j in
         range(N)] for i in range(N)]

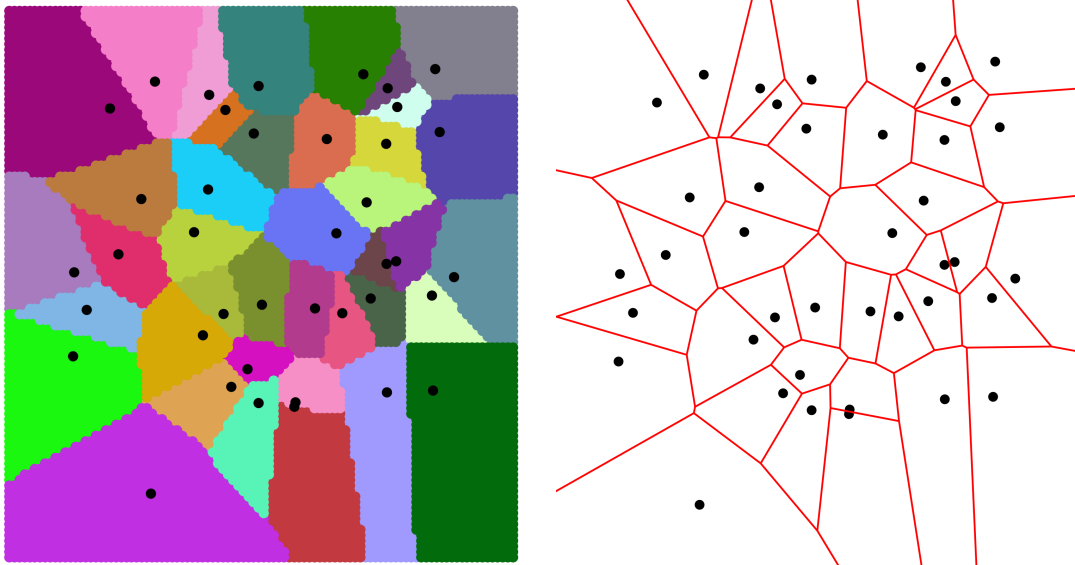
return grid, mx, my, hx, hy

```

Krata jest reprezentowana przez tablicę dwuwymiarową `grid[N][N]`, której wartościami są indeksy punktów z zadanej chmury `points`. Liczba naturalna N^2 określa liczbę punktów kratowych i jednocześnie wpływa bezpośrednio na dokładność wyznaczonego podziału. Obszar, dla którego wprowadzana jest krata, jest ograniczany do prostokąta $((m_x, m_y), (m_x, M_y), (M_x, M_y), (M_x, m_y))$, gdzie m_x, m_y, M_x, M_y to odpowiednio przeskalowane minimalne i maksymalne współrzędne punktów zadanej chmury. Odległość między sąsiednimi punktami kratowymi wynosi wówczas $h_x = (M_x - m_x)/N$ (w poziomie) i $h_y = (M_y - m_y)/N$ (w pionie). Złożoność czasowa działania tego algorytmu to $O(nN^2)$, czyli dla ustalonej liczby punktów kratowych jest liniowa względem wielkości wprowadzonej chmury punktów.

0.4.2 Graficzna prezentacja wyników działania algorytmu

Na Rysunku 3 przedstawiono wygenerowaną za pomocą algorytmu aproksymacyjnego dla 10^4 punktów kratowych wizualizację diagramu Woronoja dla 40 losowo wygenerowanych punktów oraz diagram dokładny wyznaczony opisanym wcześniej algorytmem Bowyera–Watsona.



Rysunek 3: Przybliżony diagram Woronoja 40 losowych punktów wyznaczony algorytmem aproksymacyjnym dla 10^4 punktów kratowych oraz diagram dokładny wyznaczony algorytmem Bowyera–Watsona