30.12.2020 r.

Projekt grupowy

algorytmy geometryczne

Wieloboki Voronoi

Opracowali:

Paweł Lewkowicz

Mateusz Słuszniak

1. Wstęp

Celem ćwiczenia było zaimplementowanie dwóch algorytmów wyznaczania wieloboków Voronoi dla danych zbiorów punktów na płaszczyźnie z wykorzystaniem metryki euklidesowej. Etapy realizacji algorytmu przedstawiono w formie wizualizacji graficznej. Porównano również czasy działania obu algorytmów, a następnie przedstawiono wyniki zbiorcze dla różnych typów danych wejściowych.

2. Specyfikacja techniczna

W przeprowadzonym projekcie dla zaimplementowanego algorytmu zastosowano precyzję obliczeń na poziomie 10^{-15}. Dokładność porównywania współrzędnych wierzchołków i ich klasyfikacja została zrealizowana na poziomie 10^{-9}. Obliczenia były przeprowadzane na architekturze systemu 64-bitowego oraz procesorze IC i7-8750H. Algorytmy były uruchamiane w Pythonie wersji 3.9.1.

3. Opis algorytmu Fortune'a

Program na wejściu dostaje starcie zbiór punktów o współrzędnych x oraz y na płaszczyźnie. Następnie na ich podstawie tworzy podział na takie obszary, że każdy punkt znajdujący się w obrębie danego obszaru ma najmniejszą odległość do punktu ze zbioru punktów startowych, reprezentującego dany obszar.

Do skonstruowania takich wieloboków skorzystano z algorytmu Fortune'a. Ideą samego działania algorytmu jest przechodzenie „miotły" z góry na dół po współrzędnych y punktów, będącymi zdarzeniami. Nad miotłą występuje ciąg łuków parabolicznych tak zwana linia brzegowa. Każdy taki łuk reprezentuje jeden punkt nad miotłą, który ma obszar nieograniczony od dołu. Łuki te tworzą krzywą,

która jest równo oddalona od miotły oraz punktów, stowarzyszonych z łukami. Wszystkie proste, które występują nad łukami są już niemodyfikowalne do końca działania algorytmu.

Kolejne przejścia miotły w dół następują wtedy, gdy napotkamy na punkt będący zdarzeniem punktowym, bądź okręgowym. Zdarzenia te są przechowywane w strukturze zdarzeń, będącą kolejką priorytetową po współrzędnej y. Zdarzeniami punktowymi są wierzchołki startowe, natomiast zdarzenia okręgowe są dodawane w trakcie przechodzenia miotły w dół. Są one tworzone w momencie „zanikania" łuków, w tym momencie są także tworzone nowe skończone obszary diagramu Voronoi.

W strukturze stanu diagramu Voronoi występują obszary związane z punktami startowymi, a z każdym takim obszarem związane są odpowiednie wierzchołki graniczne oraz podwójnie łączona lista krawędzi granicznych. Podwójnie łączona lista krawędzi oznacza, że każda krawędź obszarów jest reprezentowana przez dwie półproste, które mają wierzchołki początkowe i końcowe oraz przeciwne zwroty. Niektóre obszary zawierają krawędzie nieograniczone (półproste), zatem posiadają tylko jedną reprezentującą krawędź i nie posiadają wierzchołka końcowego. Na potrzeby wizualizacji tworzy się przecięcia takich krawędzi w jakimś większym obszarze, zawierającym wszystkie punkty startowe i w ten sposób powstają wierzchołki końcowe tych półkrawędzi.

Struktura stanu łuków reprezentowana jest jako drzewo czerwono-czarne, aby móc wystarczająco szybko wstawiać i usuwać kolejne łuki. Każdy taki łuk zawiera również wskaźnik do półkrawędzi, za którą odpowiada dany łuk z lewej i prawej strony. W momencie tworzenia się nowego łuku, tworzymy dwie półkrawędzi, o początku w wierzchołku będącym początkiem tego łuku. Z kolei za każdym razem, gdy usuwamy łuk, obie półkrawędzie zbiegają do tego łuku i są to końce dwóch półprostych.

4. Pseudokod algorytmu Fortune'a

Algorytm ***VoronoiDiagram***(*P*)

Dane wejściowe:
zbiór punktów *P = (x, y)*, gdzie *x, y* to współrzędne punktów

Dane wyjściowe:
diagram Voronoi ograniczony ramką jako podwójnie łączona lista krawędzi *E*

1. Inicjalizacja kolejki zdarzeń *Q* wraz ze zdarzeniami punktowymi, inicjalizacja pustej struktury drzewa *T* oraz pustej listy krawędzi podwójnie łączonych *E*

2. **While** Q nie jest pusta:
   do usuń zdarzenie z największą współrzędną y z kolejki *Q*
   if zdarzenie jest punktowe i wystąpiło w punkcie *s*:
   then ***HandleSiteEvent***(*s*)
   else ***HandleCircleEvent***(*c*), gdzie *c* jest liściem, przedstawiającym zanikający łuk
3. Węzły wewnętrzne nadal obecne w *T* przechowują półproste, które nie są jeszcze dokończone. Należy obliczyć zakresy ramki zawierającej wszystkie wierzchołki startowe w jej wnętrzu. Następnie wyznaczyć miejsca przecięcia się niedokończonych półprostych z ramką i zaktualizować listę krawędzi podwójnie łączonych.
4. Przejdź przez wszystkie półkrawędzie podwójnie łączonej listy krawędzi, aby dodać rekordy komórki oraz wskaźniki półkrawędzi do każdej z komórki oraz każdą komórkę przypisać odpowiednim krawędziom.

HandleSiteEvent(*s*)

1. Jeżeli *T* jest pusta, wstaw do niej s, tak aby był pojedynczym liściem.
   W przeciwnym razie kontynuuj kroki 2-5.
2. Wyszukaj w *T* łuk *q* znajdujący się pionowo powyżej *s*. Jeżeli liść reprezentujący q ma wskaźnik do zdarzenia okręgowego w *Q*, to zdarzenie, to jest to fałszywym alarmem i należy go usunąć z *Q*
3. Zastąp liść *T*, który reprezentuje *q*, poddrzewem posiadającym trzy liście. Środkowy liść przechowuje nowy obszar *s*, a dwa pozostałe liście przechowują obszar *u*, które było pierwotnie przechowywane w *q*. Przechowuj krotki *(s, u)* i *(u, s)* reprezentujące nowe punkty przerwania w nowych węzłach wewnętrznych. Wykonaj operacje zrównoważenia na *T*, jeśli to konieczne.
4. Utwórz nowe rekordy półkrawędzi w strukturze diagramu Voronoi dla oddzielenia krawędzi *s* oraz *u*, które będą śledzone przez dwa nowe punkty przerwania.
5. Sprawdź trzy kolejne łuki, w których nowy łuk przecina lewy łuk, aby zobaczyć, czy punkty przerwania są zbieżne. Jeżeli tak, to wstaw zdarzenie okręgowe do *Q*, a następnie dodaj wskaźniki między węzłem *T* a węzłem *Q*. Zrób to samo dla trójki, gdzie nowy łuk jest prawym łukiem.

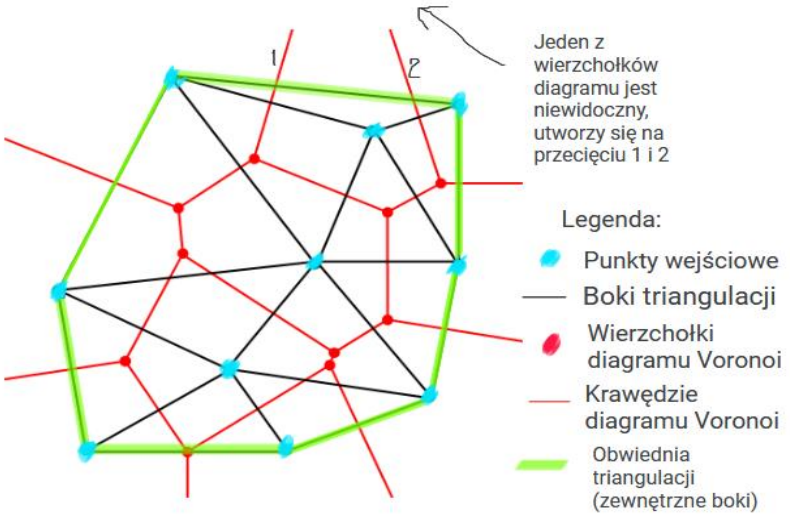
HandleCircleEvent(*c*)

1. Usuń liść *c*, który reprezentuje znikający łuk *q* z *T*. Zaktualizuj krotki reprezentujące punkty przerwania w węzłach wewnętrznych. Wykonaj operacje zrównoważenia na *T*, jeśli to konieczne. Usuń wszystkie zdarzenia okręgowe, dotyczące łuków *q* z *Q*; można je znaleźć przy pomocy wskaźników na poprzednika i następnika *c* w *T*. (Zdarzenie w okręgu, gdzie *q* jest środkowym łukiem, jest obecnie obsługiwane i zostało już usunięte z *Q*.)
2. Dodaj środek okręgu, wywołującego zdarzenie jako rekord wierzchołka do listy krawędzi podwójnie łączonych *E*, przechowującej tworzony diagram Voronoi. Utwórz dwa rekordy półkrawędzi, odpowiadających nowemu punktowi przerwania linii brzegowej. Ustaw odpowiednio wskaźniki między nimi. Dołącz trzy nowe rekordy do rekordów półkrawędzi, które kończą się na wierzchołku.
3. Sprawdź nową trójkę kolejnych łuków, w których poprzedni lewy sąsiad *q* jest łukiem środkowym, aby zobaczyć, czy dwa punkty przerwania potrójnego zbiegają się. Jeżeli tak, wstaw odpowiednie zdarzenie okręgowe do *Q* i ustaw wskaźniki między nowym zdarzeniem okręgowym w *Q* a odpowiadającym mu liściem w *T*. Zrób to samo dla trójki, gdzie poprzedni prawy sąsiad jest środkowym łukiem.

5. Algorytm oparty o triangulację Delaunay'a

Drugi z algorytmów znajdujący diagram Voronoi jest oparty na triangulacji Delaunay'a. Implementacje triangulacji Delaunay'a wzięliśmy z modułu scipy.spatial, który z kolei korzysta z innej biblioteki, a mianowicie Qhull (biblioteka do algorytmów geometrycznych). Mając daną triangulacje w postaci trójek punktów, które tworzą trójkąt triangulacji, można w relatywnie prosty sposób uzyskać diagram Voronoi'a. Należy dla każdego z trójkątów znaleźć ich środki okręgów opisanych. Środki te stanowią wierzchołki diagramu Voronoi'a. Następnie należy łączyć ze sobą te wierzchołki diagramu, dla których bok pojawiał się w obydwu trójkątach (trójkątach, których środkami są wierzchołki diagramu, patrz rysunek poniżej). Część boków będzie pojawiała się tylko raz, będą to boki stanowiące obwiednie triangulacji. Wówczas od punktu, który jest środkiem okręgu opisanego na trójkącie zawierającym dany bok, należy poprowadzić półprostą. Półprosta ta będzie prostopadła do „osamotnionego" boku trójkąta.

Rys. 1 Diagram Voronoi (czerwone kropki i krawędzie) uzyskany na podstawie triangulacji Delaunay'a



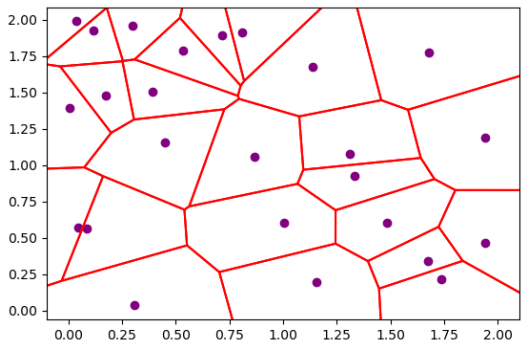
6. Zbiory punktów

W celu sprawdzenia działania algorytmu przygotowano różne zbiory punktów startowych. Każdy zbiór składał się z kilkunastu wierzchołków. Są to następujące zbiory:

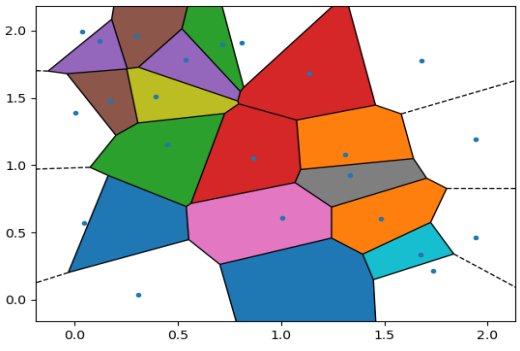
- losowo wygenerowane punkty z zadanego obszaru, rozmieszczone losowo z rozkładu jednostajnego
- równomiernie rozmieszczone punkty, tworzące kratę
- równomiernie rozmieszczone punkty ze względu na kąt na okręgu
- równomiernie rozmieszczone punkty ze względu na kąt na okręgu wraz z punktem środkowym okręgu
- równomiernie rozmieszczone punkty na odcinku
- losowo rozmieszczone punkty na odcinku

Dla tych zbiorów przygotowano wizualizację działania zaimplementowanego algorytmu Fortune'a oraz algorytmu, korzystającego z biblioteki *scipy.spatial*.

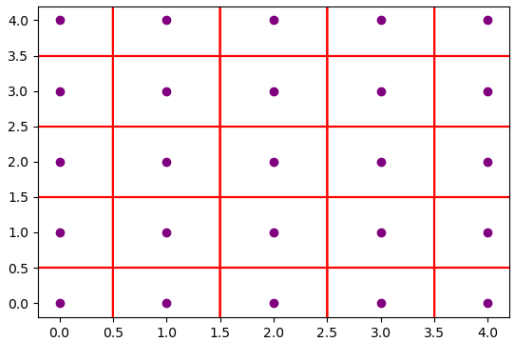
7. Wizualizacja



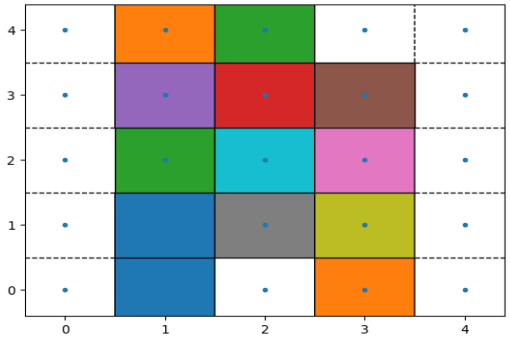
Wykres_1



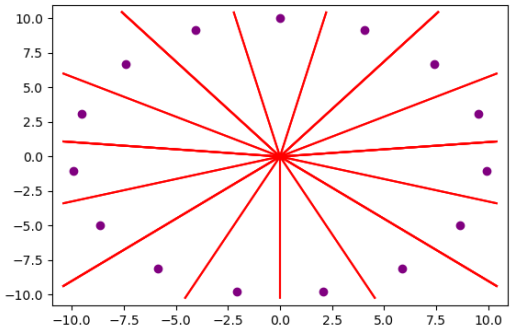
Wykres_2



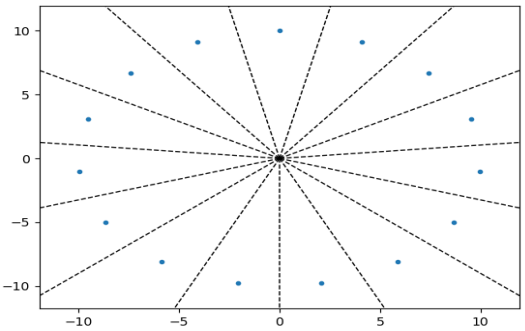
Wykres_3



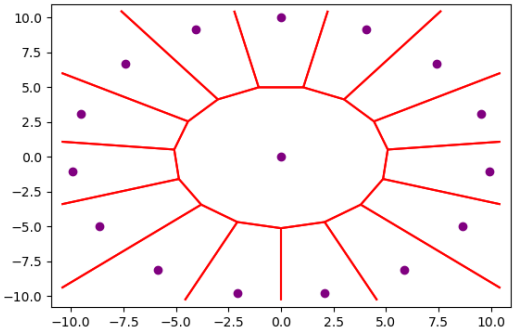
Wykres_4



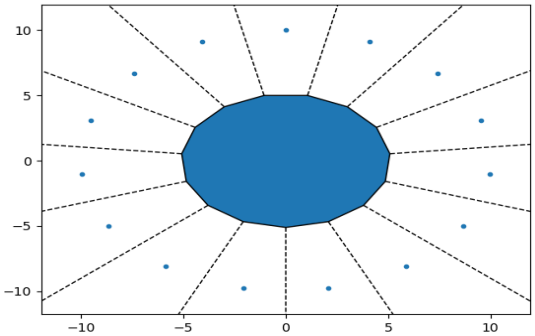
Wykres_5



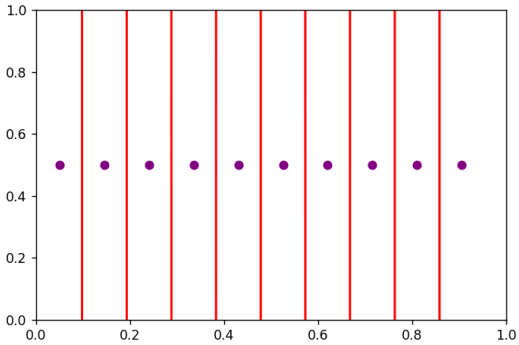
Wykres_6



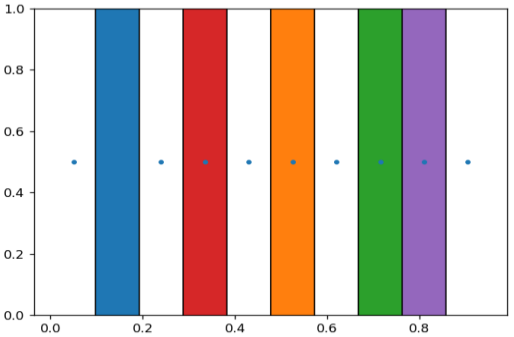
Wykres_7



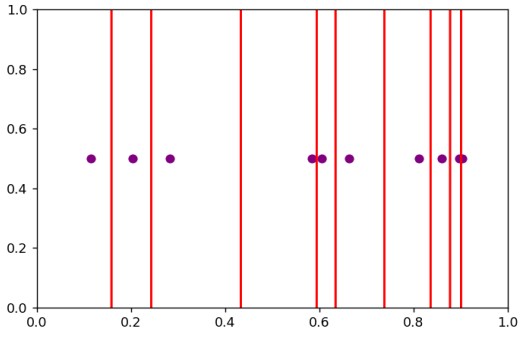
Wykres_8



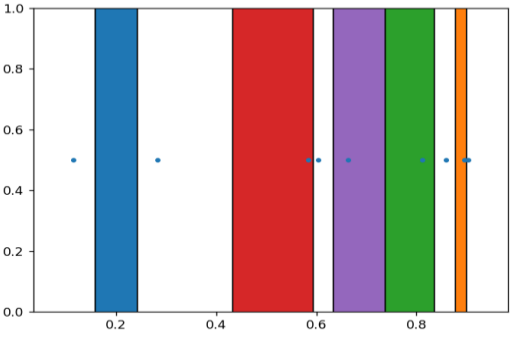
Wykres_9



Wykres_10



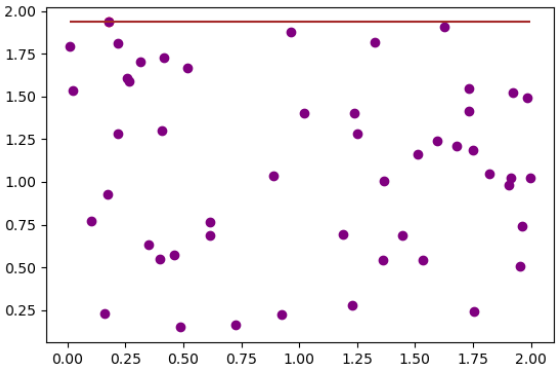
Wykres_11



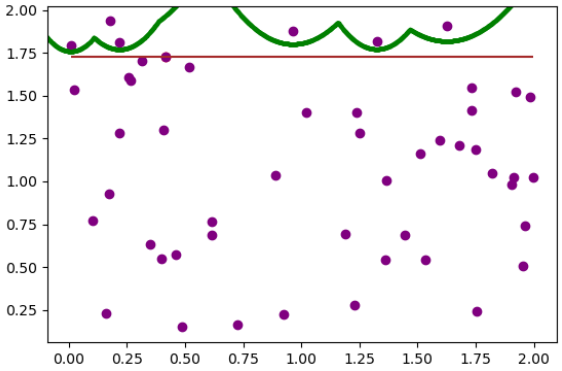
Wykres_12

Wykresy o numerach nieparzystych przedstawiają efekt końcowy działania zaimplementowanego programu, a o numerach parzystych korzystającego z biblioteki *scipy.spatial*. Ze względu na to, że niektóre obszary algorytm klasyfikował jako nieograniczone, nie zostały one pokolorowane.

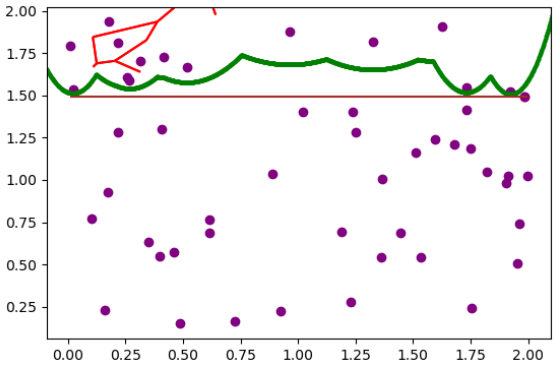
Aby lepiej przedstawić działanie samego algorytmu, stworzono również sceny, które są tworzone w trakcie natrafienia na zdarzenie punktowe. Na wykresach 13-20 przedstawiono kolejno niektóre z tych scen dla losowo wygenerowanych 50 punktów.



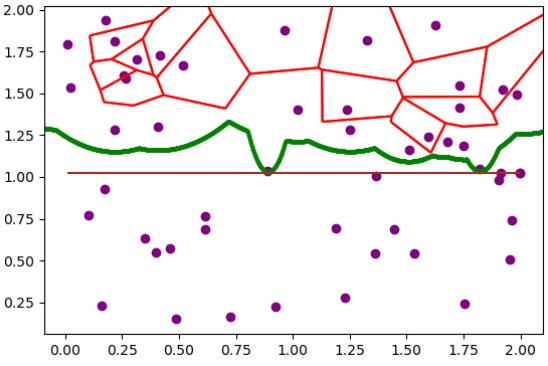
Wykres_13



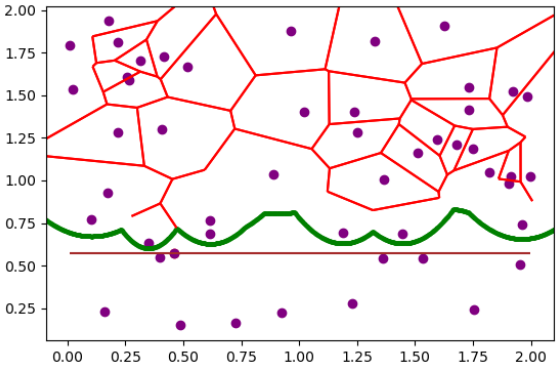
Wykres_14



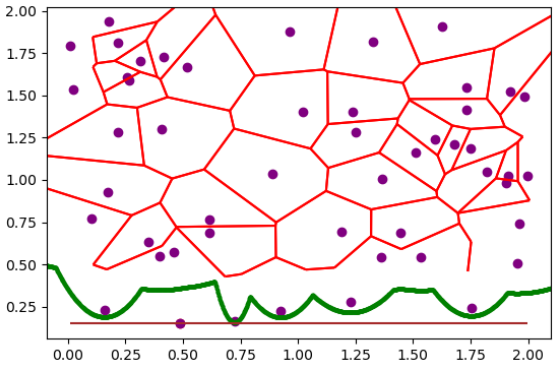
Wykres_15



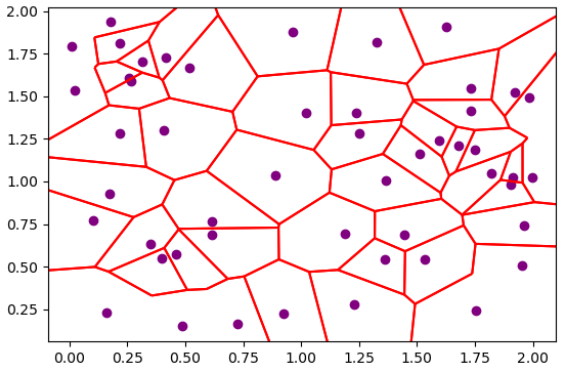
Wykres_16



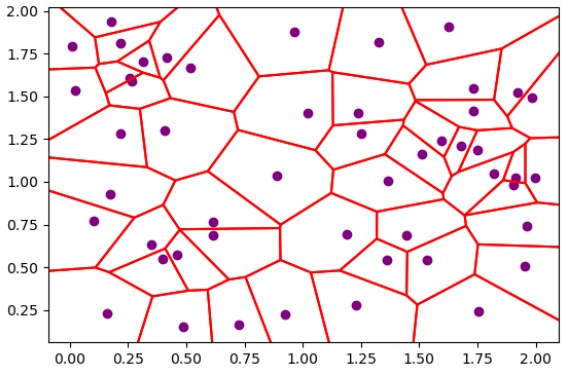
Wykres_17



Wykres_18



Wykres_19

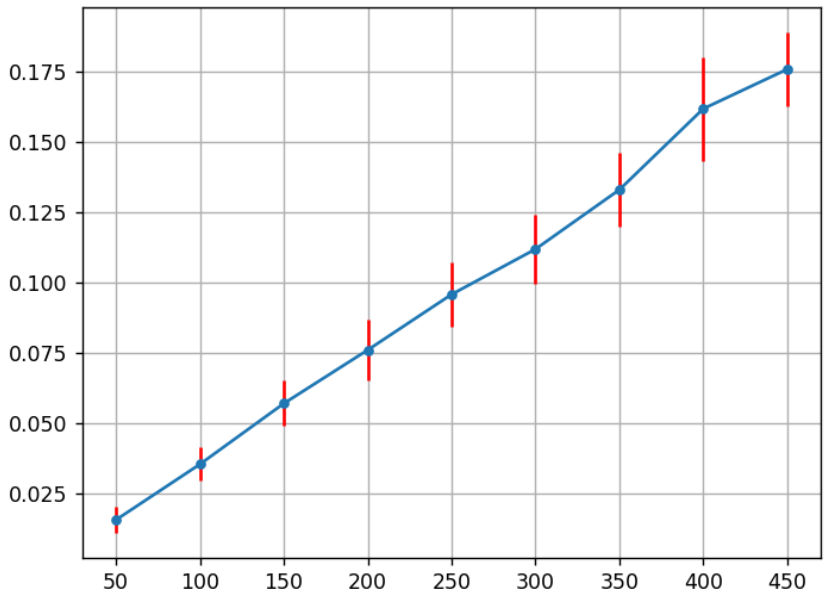


Wykres_20

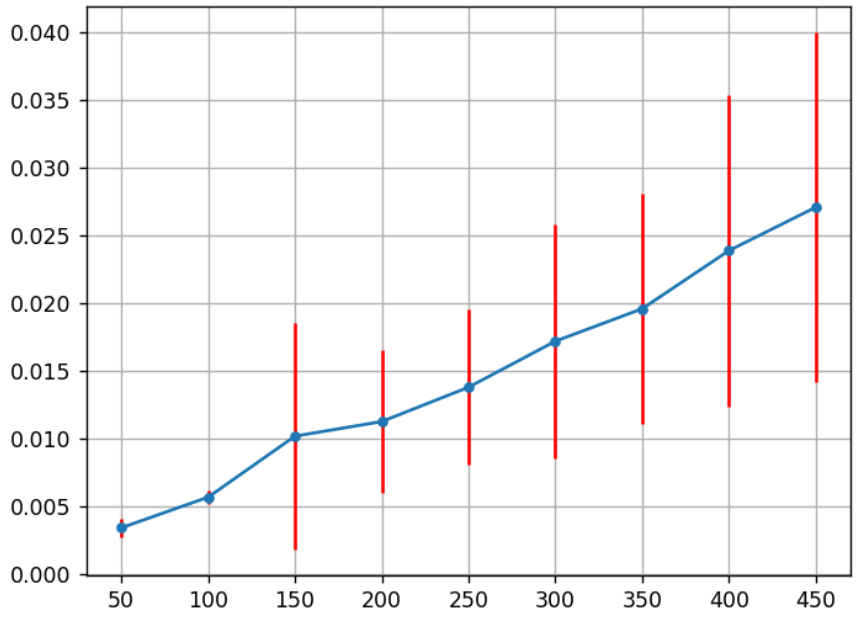
Brązowa linia pozioma reprezentuje „miotłę”, punkty fioletowe są wierzchołkami startowymi, a czerwone krawędzie są krawędziami Voronoi. Zielona krzywa (linia brzegowa) przedstawia łuki, które znajdują się aktualnie w strukturze. Pomiędzy wykresami 19-20 następuje zaktualizowanie wszystkich niedokończonych półkrawędzi, a następnie przedstawieni finalnego diagramu.

8. Analiza czasowa

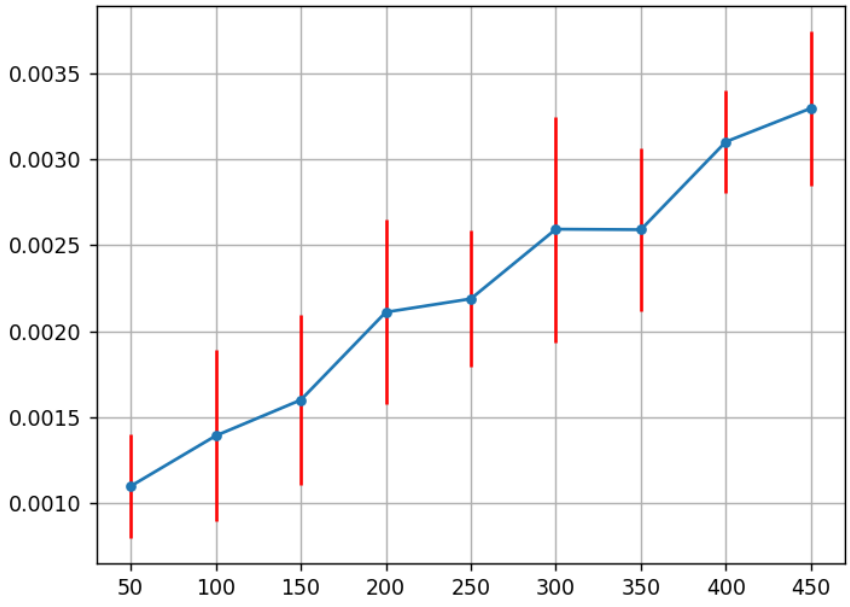
Aby można było porównać szybkość działania obu algorytmów, przetestowano czas ich działania dla losowo rozmieszczonych punktów w ilości: 50, 100, 150, 200, 250, 300, 350, 400, 450. Na wykresach 21-23 przedstawiono wykresy czasu w sekundach potrzebnego do wykonania się algorytmu od ilości punktów startowych. *Wykres_21* przedstawia szybkość zaimplementowanego algorytmu algorytmu Fortune’a, *wykres_22* szybkość algorytmu z triangulacją Delaunay’a, a wykres_23 szybkość algorytmu, wykorzystujący bibliotekę *scipy.spatial*.



Wykres_21



Wykres_22



Wykres_23

Dla każdej ilości punktów startowych wykonano 50 testów, a następnie zaznaczono na wykresie średni wynik, a następnie wyznaczono jego odchylenie standardowe. Czerwonymi liniami zaznaczono zakres jednego odchylenia standardowego.

Algorytm Fortune'a ma złożoność O(N log N). Na trzech wykresach wyniki układają się w sposób liniowo-logarytmiczny. Badając czasy tych algorytmów, można zauważyć, że algorytm Fortune'a jest najwolniejszy dla takich zbiorów punktów, algorytm z triangulacją jest 1 rząd wielkości szybszy, a biblioteczny 2 rzędy wielkości szybszy od Fortune'a.

Przyglądają się tym wynikom bardzo łatwo znaleźć rozsądne wytłumaczenie dla takiego właśnie rezultatu wyników czasowych. Algorytm Fortune'a, który był całkowicie napisany przez nas osiąga najsłabszy czas. Wynika to przede wszystkim z tego, że nie posiada optymalizacji bibliotecznych jak reszta algorytmów. Część algorytmu która decyduje o ostatecznej złożoności (N log N) jest napisana własnoręcznie, co przy braku maksymalnej optymalizacji od razu podbija złożoność o pewną stałą. Drugim najszybszym algorytmem jest program opierający się o znalezienie triangulacji Delaunaya, a następnie diagramu Voronoi. Poradził on sobie szybciej od poprzedniego, ponieważ część algorytmu, która decydowała o złożoności (triangulacja jest O(N log N)) była implementacją biblioteczną. Zamiana grafu triangulacji na diagram Voronoi jest liniowe. Ostatnia z implementacji robi to samo co poprzednia implementacja, tylko, że jest w całości wziętą implementacją biblioteczną. Optymalizacja na każdym kroku algorytmu sprawia, że działa on naturalnie najszybciej. Co warte podkreślenia, czasy dział algorytmów są wraz z n rosną niemal identycznie. Zatem czas zbija tutaj stała, zaś w notacji O algorytmy mają taką samą złożoność czyli N log N.

9. Uwagi i wnioski

W trakcie pisania algorytmu natrafiliśmy na serie wykresów, które zawierały źle sklasyfikowane pojedyncze krawędzie lub półkrawędzie bądź program się nie kompilował. Było to najbardziej zauważalne dla zbiorów punktów ułożonych regularnie. Wynikało to między innymi z tego, że niektóre punkty miały taką samą współrzędną x lub y, a program nie był w stanie rozstrzygnąć, nad jakim dokładnie łukiem leżą nowe zdarzenia, ponieważ wtedy łuki zanikając, miały bardzo małe szerokości. Właśnie pod tymi łukami pojawiały się wtedy nowe punkty, które trzeba było sklasyfikować do odpowiedniego łuku.

Problem ten rozwiązaliśmy w następujący sposób. Dla równomiernie sklasyfikowanych punktów dodawaliśmy do współrzędnych x oraz y tych punktów małe losowe epsilony z zakresu $(0, 10^{-5})$. dzięki temu nie występowały już żadne anomalie w klasyfikacji, a jednocześnie nie miało to większego wpływu na wykres, ponieważ różnice te były na nim niedostrzegalne.