Lokalizacja punktu na płaszczyźnie. Metoda Separatorów

Kołotyło Jakub

Nieć Witold

Spis treści

1. Dane Techniczne

Wersja Python: 3.12.4
Wersja matplotlib: 3.9.0

1.1. Komputer 1

Procesor: AMD Ryzen 7 5700U
RAM: 16 GB
Używane środowiska: Anaconda, VSCode
System operacyjny: Windows 11

1.2. Komputer 2

Procesor: Intel core 7 155V
RAM: 16 GB
Używane środowiska: Anaconda, Pycharm
System operacyjny: Windows 11

2. Wstęp

Celem niniejszej pracy jest przedstawienie implementacji i analizy algorytmu lokalizacji punktu na płaszczyźnie za pomocą metody separatorów. Problem lokalizacji punktu na płaszczyźnie polega na określeniu, w którym z rozłącznych obszarów (np. wielokątów lub regionów) znajduje się zadany punkt. Jest to zagadnienie często spotykane w takich dziedzinach, jak grafika komputerowa, systemy informacji geograficznej (GIS) czy robotyka.

Metoda separatorów jest efektywnym podejściem do rozwiązywania tego problemu, opierającym się na podziale płaszczyzny na mniejsze regiony za pomocą linii, prostych lub innych struktur geometrycznych. Dzięki temu możliwe jest zredukowanie złożoności obliczeniowej i przyspieszenie procesu wyszukiwania.

W sprawozdaniu omówiono założenia teoretyczne algorytmu, przedstawiono jego implementację, a także przeanalizowano wyniki uzyskane dla przykładowych danych testowych.

3. Założenia teoretyczne

Wejście:

- Skierowany graf acykliczny w postaci listy sąsiedztwa.
- Słownik zawierający współrzędne punktów grafu.

W algorytmie lista sąsiedztwa jest rozdzielana na dwie listy: lista sąsiedztwa z wierzchołkami tylko powyżej rozważanego oraz sąsiedztwa z wierzchołkami tylko poniżej rozważanego.

Punkty w tablicy są posortowane względem współrzędnej (Y), natomiast w liście sąsiedztwa względem kąta, który tworzą z dodatnią półosią (OX).

Separator:
Łamana, monotoniczna względem kierunku osi (OY), zbudowana z krawędzi grafu.

Wyjście:
Współrzędne punktów tworzących minimalny zamknięty obszar, do którego należy analizowany punkt.

4. Wprowadzanie podziału

Wielokąt oraz podział można wprowadzić za pomocą myszki. Program do manualnego wprowadzania został zrealizowany za pomocą biblioteki *matplotlib* wykonując kod z klasą LineDrawer. Lewy przycisk dodaje krawędź między oznaczonym wierzchołkiem, a klikniętym oraz go oznacza. Jeżeli, żaden wierzchołek nie jest oznaczony, lewy przycisk tylko oznacza wierzchołek. Prawy przycisk odznacza wierzchołek.

Szukany punkt również można podać za pomocą biblioteki *matplotlib*, wykonując kod z klasą PointSelector.

5. Regularyzacja podziału

Algorytm lokalizacji punktu na płaszczyźnie wymaga aby podawany jako argument, graf był regularny. Oznacza to że każdy wierzchołek poza źródłem i ujściem grafu (wierzchołkami: najwyższym i najniższym względem współrzędnej Y) musi posiadać przynajmniej po jednej krawędzi łączącej go z dowolnym wierzchołkiem leżącym powyżej i poniżej danego.

5.1. Założenia teoretyczne

Argumenty podawane do algorytmu:

- słownik punktów posortowanych po współrzędnej Y
- lista sąsiedztwa z wierzchołkami powyżej posortowane według kąta jaki tworzy krawędź z osią OX (od największego kąta)
- lista sąsiedztwa z wierzchołkami poniżej posortowane według kąta jaki tworzy krawędź z osią OX (od największego kąta)

5.2. Struktury danych

5.2.1. Line (proste będące krawędziami grafu)

Atrybuty:

- start - jeden z punktów przez który dana prosta przechodzi, leżący wyżej
- end - drugi z punktów, leżący poniżej
- a - współczynnik kierunkowy prostej
- b - wyraz wolny w równaniu kierunkowym prostej

5.2.2. Section (przedziały znajdujące się pomiędzy prostymi)

Atrybuty:

- helper - pomocnik przedziału (indeks punktu)
- left (Line) - prosta ograniczająca przedział od lewej
- right (Line) - prosta ograniczająca przedział od prawej
- mustConnect - tablica wierzchołków do których musi zostać dodana krawędź po napotkaniu wierzchołka należącego do tego przedziału.

Metody:

- calX - metoda po podaniu wartości Y zwraca wartości X prostych ograniczających dany przedział w postaci krotki
- contains_point - metoda po podaniu punktu zwraca kolejno:
  -1 - jeśli punkt leży po lewo od przedziału
  0 - gdy punkt należy do przedziału
  1 - gdy punkt leży po prawej od przedziału
  W przypadku gdy punk leży na granicy przedziałów, 0 zwracane jest tylko dla przedziału najbardziej po lewo.

5.3. Funkcje pomocnicze

- bin_search - funkcja przyjmuje listę przedziałów (Section) oraz punkt i korzystając z metody *contains_point* wyszukuje binarnie indeks przedziału do którego należy punkt.

5.4. Algorytm zamiatania

Algorytm zajmuje się regularyzacją skierowanego grafu, w którym wierzchołki są uporządkowane według współrzędnej YY od najwyższych do najniższych. Jego celem jest dodanie minimalnego zestawu krawędzi, które umożliwią poprawne połączenie wierzchołków zgodnie z wymaganiami strukturalnymi. Przetwarzanie opiera się na zarządzaniu przedziałami odpowiadającymi wierzchołkom grafu oraz analizie ich krawędzi wchodzących i wychodzących.

Algorytm rozpoczyna swoje działanie od inicjalizacji dwóch struktur danych: listy sections, reprezentującej aktualne przedziały, oraz zbioru *added_edges*, w którym przechowywane są wszystkie dodane krawędzie regularyzacyjne. Następnie wierzchołki grafu są przetwarzane w kolejności od najwyższych do najniższych wartości współrzędnej *Y*. Dla każdego wierzchołka obliczana jest liczba krawędzi wchodzących do tego punktu (*IN*) oraz liczba krawędzi wychodzących (*OUT*).

Jeśli dany wierzchołek nie ma żadnych krawędzi wchodzących, algorytm dodaje krawędź pomocniczą łączącą ten wierzchołek z aktualnym przedziałem. Następnie usuwa *IN+1* przedziałów odpowiadających krawędziom wchodzącym do wierzchołka. W przypadku, gdy jeden z usuwanych przedziałów zawiera wierzchołki oznaczone jako wymagające połączenia (*mustConnect*), algorytm dodaje krawędzie między bieżącym wierzchołkiem a każdym z tych punktów. Po usunięciu odpowiednich przedziałów, algorytm tworzy *OUT+1* nowych przedziałów, które odpowiadają krawędziom wychodzącym z przetwarzanego wierzchołka.

Jeżeli bieżący wierzchołek nie posiada krawędzi wychodzących, jego indeks zostaje dodany do listy *mustConnect* w jedynym nowo utworzonym przedziale. W ten sposób algorytm zapewnia, że wierzchołki bez wyjść zostaną poprawnie połączone w dalszych iteracjach.

Każda krawędź dodana w ramach regularyzacji jest zapisywana w zbiorze *added_edges*, który na końcu stanowi wynik działania algorytmu. Zbiór ten zawiera minimalną liczbę krawędzi, które zostały dodane do grafu, aby spełniał on wymagania strukturalne. Dzięki temu algorytm skutecznie reguluje graf, zachowując jego poprawność i minimalizując liczbę dodanych połączeń.

Pseudokod:

```
Wejście:
słownik posortowanych punktów po Y od najwyższych,
lista sąsiedztwa dla leżących powyżej,
lista sąsiedztwa dla leżących poniżej

sections <- pusta lista
added_edges <- pusty zbiór krawędzi dodanych w ramach regularyzacji
dla każdego punktu w kolejności od najwyższych:
    IN <- liczba krawędzi wchodzących do wierzchołka
    OUT <- liczba krawędzi wychodzących z wierzchołka
    jeżeli IN == 0:
      dodaj krawędź do pomocnika aktualnego przedziału
    usuń IN+1 przedziałów
    jeśli dla usuwanego przedziału .mustConnect != []:
        dodaj krawędzie miedzy wierzchołkiem rozważanym a każdym
elementem listy
    w miejsce usuniętycz dodaj OUT+1 przedziałów
    jeżeli OUT == 0:
       do jedynego dodanego przedziału dodaj indeks wierchołka do
mustConnect

Wyjście: zbiór dodanych krawędzi
```

6. Algorytm wyznaczania lokalizacji punktu na płaszczyźnie

6.1. Wyznaczanie Separatorów

Wyznaczanie separatorów rozpoczynamy od przypisania wag krawędziom grafu według pseudokodu:

```
for każda krawędź e: W(e) := 1
for każdy wierzchołek v w kolejności indeksów rosnących:
  W_IN(v) := SUMA_IN(v)[W(e)]
  d: = pierwsza z lewej krawędź odchodząca z v
  if WIN(v) > WOUT(v)
    then W(d) := WIN(v) – WOUT(v) + W(d);
for każdy wierzchołek v w kolejności indeksów malejących:
  WOUT(v) := SUMA_OUT(v)[W(e)]
  d := pierwsza z lewej krawędź dochodząca do v
  if WOUT(v) > WIN(v)
    then W(d) := WOUT(v) – WIN(v) + W(d)
```

Uzyskane wagi wykorzystujemy w algorytmie generowania separatorów. Separator, zgodnie z definicją, jest monotoniczną łamaną, co oznacza, że musi składać się z kolejno rosnących wierzchołków.

Proces tworzenia separatorów rozpoczynamy od najwyższego wierzchołka w grafie, schodząc stopniowo w dół i wybierając niezerowe krawędzie położone możliwie najbardziej na lewo. Po wybraniu krawędzi zmniejszamy jej wagę o 1 i dodajemy ją do separatora. Czynności te powtarzamy aż do momentu dotarcia do najniższego wierzchołka, tym samym dopełniając separator. Jeśli po zakończeniu tego procesu w grafie pozostaną krawędzie o niezerowych wagach, ponownie zaczynamy od najwyższego wierzchołka i powtarzamy algorytm. Procedura trwa do momentu, aż wszystkie wagi krawędzi osiągną wartość 0.

6.2. Lokalizacja punktu

Po wyznaczeniu separatorów, tworzone jest binarne drzewo poszukiwań. Algorytm lokalizacji opiera się na podwójnie zagnieżdżonych wyszukiwaniach binarnych. Wewnętrzne wyszukiwanie znajduje taką krawędź separatora, żeby rzędna lokalizowanego punktu znajdowała się pomiędzy rzędnymi końców krawędzi. Następnie sprawdzana jest strona, po której leży punkt względem tej krawędzi (i konsekwentnie, separatora) przy użyciu wyznacznika macierzy 2x2 opisanego dokładniej w laboratorium 1. Następnie ta strona jest używana do stwierdzenia, w którym kierunku przejść w zewnętrznym wyszukiwaniu binarnym, do którego stosowane jest drzewo binarne stworzone wcześniej. Pod koniec zwracane są separatory, między którymi znajduje się punkt.

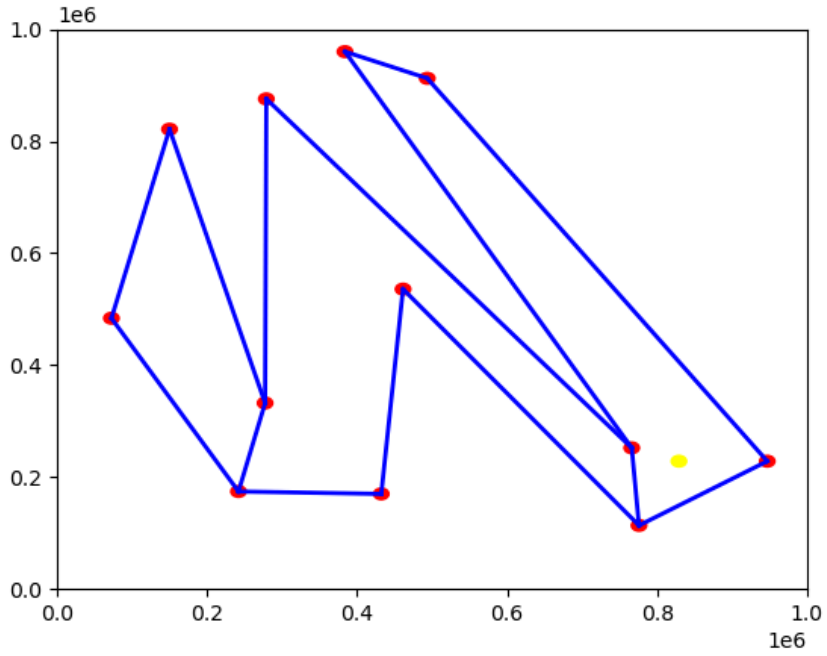
6.3. Doprecyzowanie podziału

Obszary podane na wejściu nie są jednak jednoznacznie wyznaczone przez separatory. Aby znaleźć obszar w którym znajduje się punkt, użyty jest algorytm dwóch wskaźników. Idea jest taka, żeby przejść po każdym separatorze, w podobnych tempach, tj. gdy wierzchołek, na który wskazuje wskaźnik separatora pierwszego jest dalej od drugiego, należy zwiększyć wskaźnik drugiego, aby mu dorównać. Jeżeli oba wskaźniki wskażą na ten sam wierzchołek, oznacza to, że domknięty został obszar. Jeżeli poszukiwany punkt znajduje się wyżej, to zwracamy ten obszar, jeśli nie, kontynuujemy algorytm.

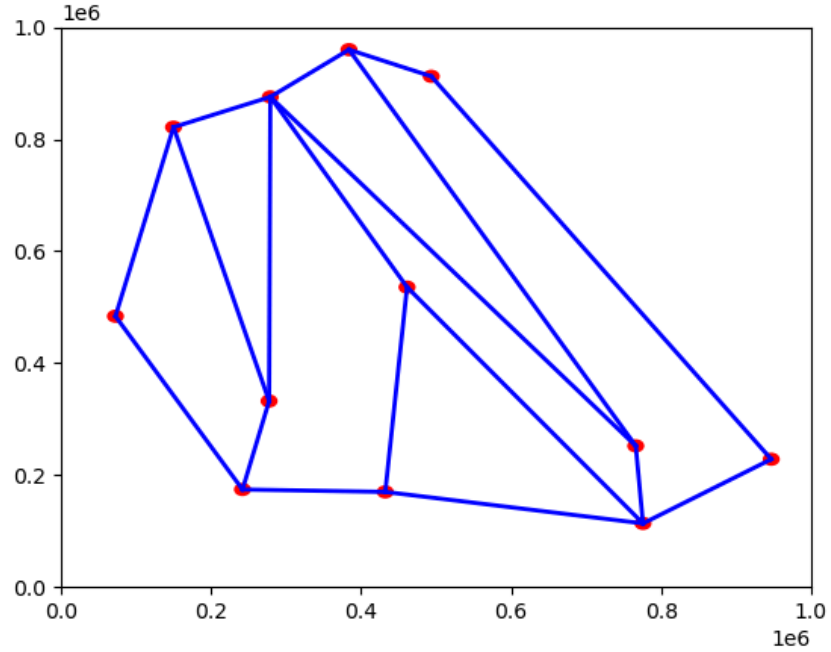
Regularyzacja podziału dodała sztuczne krawędzie, które nie zostały podane przez użytkownika. Zatem jest możliwe, że poprzedni algorytm zwrócił obszar ograniczony poprzez jedną ze sztucznych krawędzi. W takim wypadku, odczytywana jest krawędź znalezionego obszaru niedodana przez regularyzację, a następnie, idąc w kierunku wskazówek zegara, przechodzimy po **oryginalnym** grafie zawsze biorąc krawędź najbliższą prawej strony względem poprzedniej krawędzi. W ten sposób jesteśmy w stanie odzyskać oryginalny obszar.

7. Przykład działania algorytmu

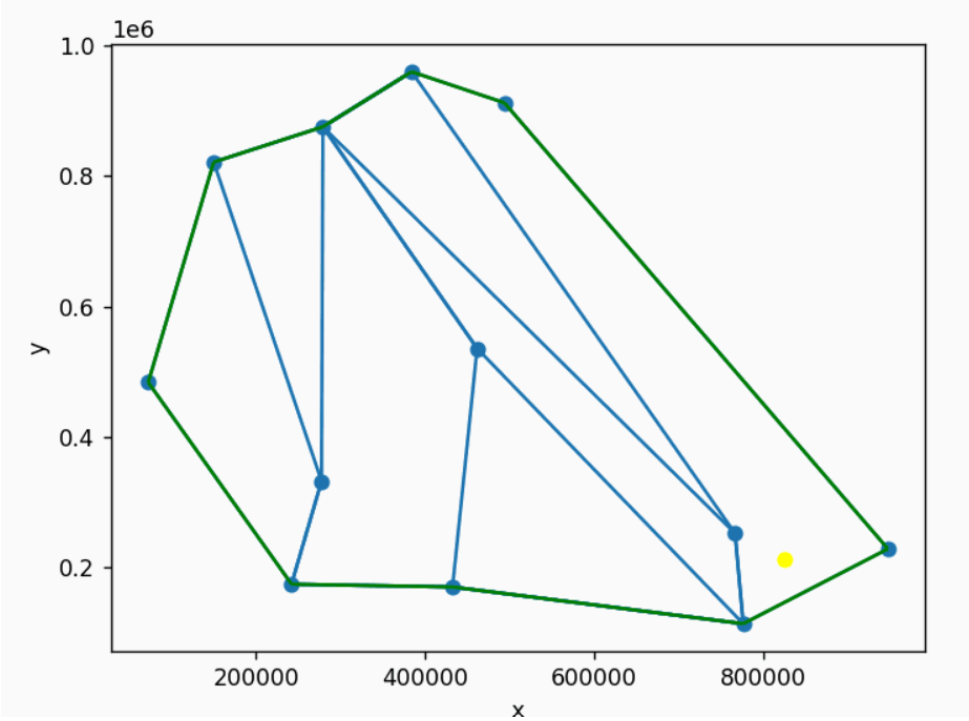
7.1. Przykład A



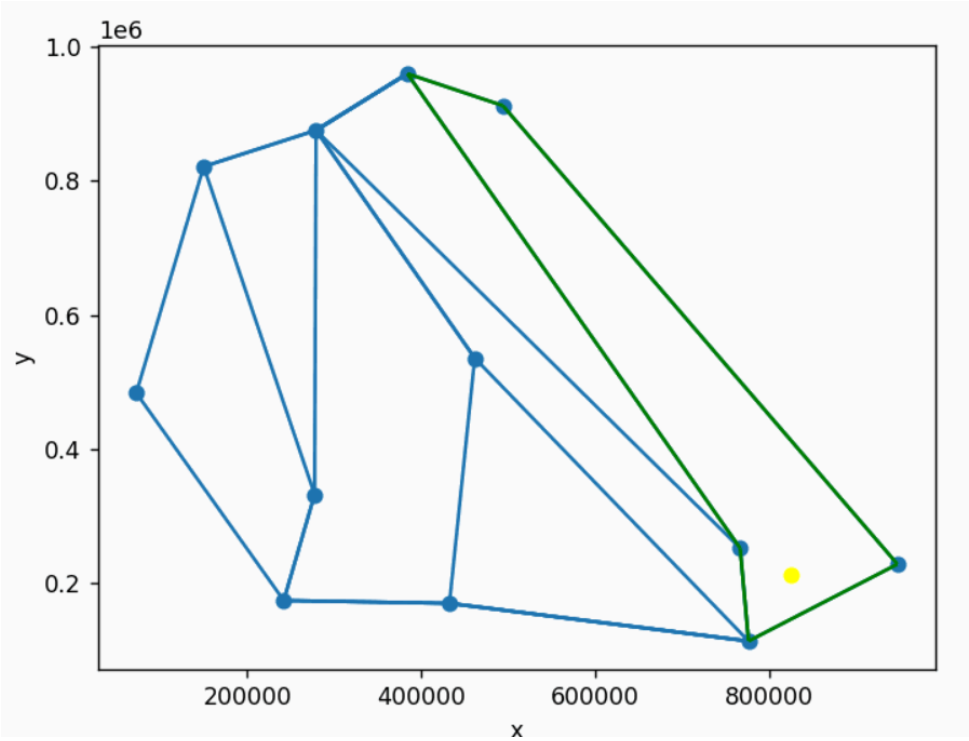
Rysunek 1: Dany graf oraz punkt



Rysunek 2: Graf po regularyzacji

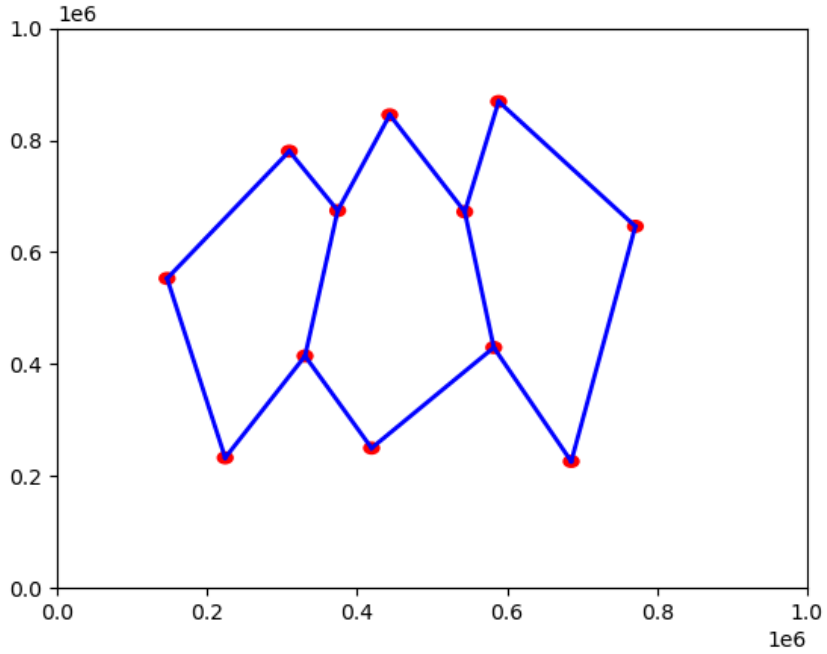


Rysunek 3: Początkowa faza wyszukiwania binarnego

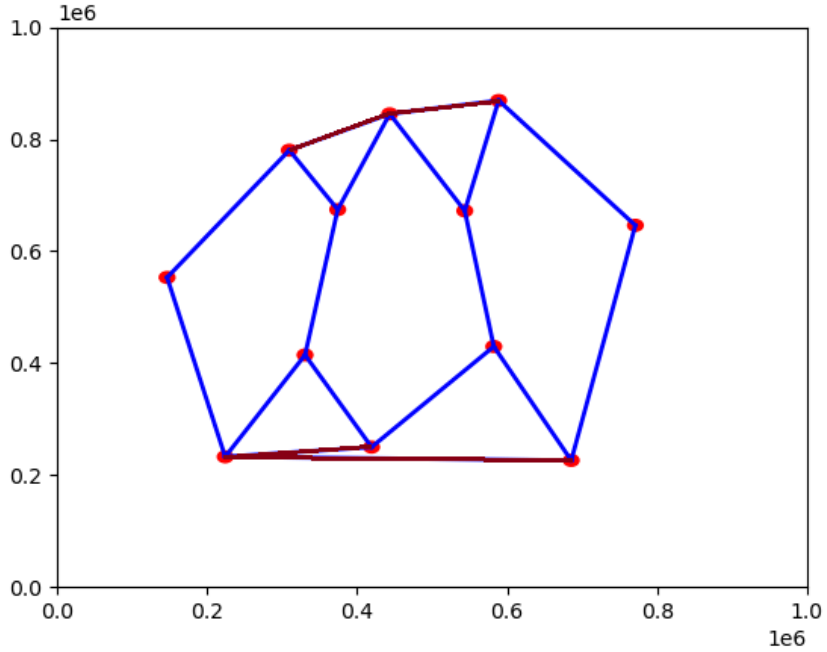


Rysunek 4: Końcowa faza wyszukiwania binarnego

7.2. Przykład B

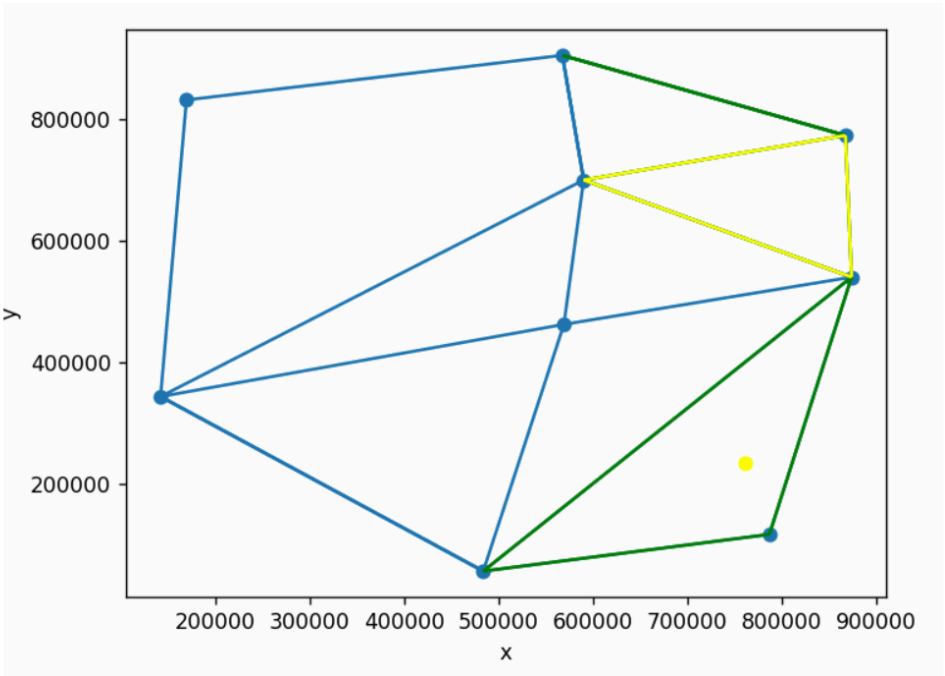


Rysunek 5: Wielokąt przed regularyzacją

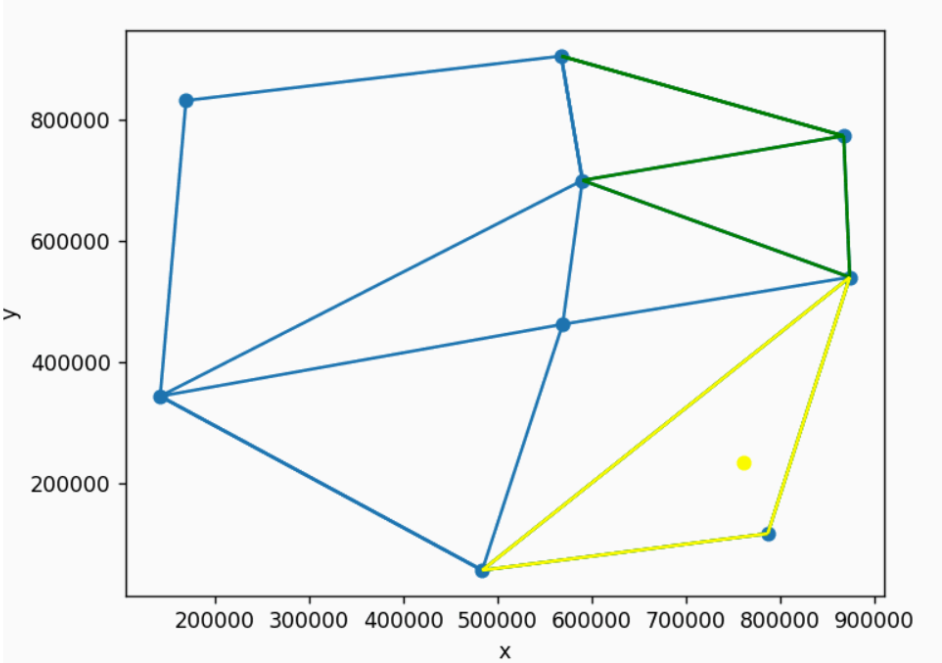


Rysunek 6: Wielokąt po regularyzacji

7.3. Przykład C



Rysunek 7: Pierwszy obszar między separatorami - brak szukanego punktu



Rysunek 8: Drugi obszar między separatorami - znaleziono szukany punkt

8. Analiza złożoności obliczeniowej

1. Regularyzacja:
   obejmuje to liniowe przejście po posortowanej liście wierzchołków, co wymaga złożoności $O(n \log n)$
2. Znalezienie separatorów:
   wymaga to przejścia po każdej krawędzi zatem $O(n)$
3. Tworzenie drzewa binarnego:
   jest to stworzenie drzewa z posortowanej listy $O(n)$
4. Lokalizacja punktu:
   są to dwa zagnieżdżone wyszukiwanie binarne zatem $O(\log^2 n)$

9. Narzędzia do wizualizacji

- Narzędzie wizualizacji BIT
- Biblioteka matplotlib
- Biblioteka Manim

10. Wnioski

Algorytm lokalizacji punktu oparty na metodzie separatorów wykazuje wysoką efektywność, szczególnie w przypadku dużych grafów i wielokątów podzielonych na mniejsze obszary. Dzięki zastosowaniu regularyzacji, binarnego drzewa poszukiwań oraz podziału na separatory, możliwe jest znaczące ograniczenie liczby operacji wymaganych do znalezienia poszukiwanego obszaru.

Metoda jest prosta w implementacji, efektywna pod względem czasu i pamięci oraz uniwersalna.

Jednakże, w przypadku dużej liczby separatorów lub grafów gęstych, etap znajdywania separatorów może wpłynąć na czas działania. Ponadto algorytm wymaga wstępnej regularyzacji grafu, co wiąże się z dodatkowymi operacjami.

11. Bibliografia

- Dr. Inż. Barbara Głut, prezentacja Lokalizacja punktu, stworzona na potrzeby programu Algorytmy Geometryczne.
- Herbert Edelsbrunner, Leo J. Guibas, Jorge Stolfi, Optimal Point Location in Monotone - Subdivision https://graphics.stanford.edu/courses/cs268-07--winter/manuals/monotone-pointloc.pdf
- Jack Snoeink, Point location https://www.csun.edu/~ctoth/Handbook/chap38.pdf
- Der-Tsai Lee, Franco P. Preparata, Location of a Point in a Planar Subdivision and Its Application https://www.ideals.illinois.edu/items/100389