Jakub Guzek
Carlos Zaldivar Batista
Prowadzący: mgr inż. Paweł Zawistowski
Algorytmy heurysytczne, EiTI 2016

Sieć autostrad

Dokumentacja końcowa

Opis zadania

Celem projektu jest znalezienie optymalnej sieci autostrad łączącej podane miasta tak, aby każde miasto było połączone ze wszystkimi innymi. W zależności od potrzeby (zadanych wartości od użytkownika) mamy minimalizować ilość wylanego asfaltu/betonu oraz minimalizować odległości miast między sobą.

Założenia ogólne

- Miasta są reprezentowane przez punkty przestrzeni dwuwymiarowej o całkowitych współrzędnych.
- Przecięcia autostrad reprezentowane są przez punkty przesztrzeni dwuwymiarowej o rzeczywistych współrzędnych.
- Jeżeli autostrada zostaje przecięta inną autostradą to w tym momencie powstają dwa odcinki autostrad - od starego początku do punktu przecięcia oraz od puntu przecięcia do starego końca.
- Patrząc na punkt wyżej znaczenie odcinka tutaj będzie takie, że nasz odcinek to taki rodzaj odcinka, który nie przecina się z żadnym innym (na końcach może miec wspólne punkty)
- Rozwiązanie zostanie zaimplementowane z wykorzystaniem języka Python.
- Do obliczenia odległości między miastami zostanie użyty algorytm z biblioteki networkx.

Dane wejściowe

Użytkownik ma możliwość określenia następujących parametrów:

- Wagi funkcji celu (parametry *A* i *B*)
- Liczba iteracji

Lista punktów reprezentująca miasta (<$M_1, M_2, \ldots, M_n$>) jest podawana w stdin (jeśli chcemy podać plik jako weście, możemy użyć “cat nazwa_pliku_wej | python3.4 main.py” w systamach Unix lub “type nazwa_pliku_wej | python main.py dla systemu Windows), każde miasto jest reprezentowane przez parę liczb całkowitych oddzielonych przecinkiem.

Przestrzeń przeszukiwania

W naszej przestrzeni przeszukiwań nie występują takie rozwiązania, dla których miasto nie ma połączenia z jakimkolwiek innym miastem.

Opis sąsiedztwa

Stan S jest reprezentowany przez wektor v, którego elementami są pary punktów (miast lub skrzyżowań) reprezentujących odcinki autostrady.

$$v = <K_1, K_2, \ldots, K_m>$$

Sąsiadami stanu S są stany reprezentowane przez wektory:

- krótsze o jeden element, z pozostałymi elementami identycznymi jak w S - czyli zabieramy jedną z krawędzi (odcinków). Trzeba pamiętać tutaj o przestrzeni przeszukiwania - nie możemy zabrać takiego odcinka, który będzie niszczył połączenie między dowolnymi dwoma miastami
- z dodatkową krawędzią łączącą dwa wcześniej nie połączone punkty. W tym przypadku należy pamiętać, aby sprawdzić czy gdzieś nie powstają dodatkowe skrzyżowania i odpowiednio na powstałą sytuację zareagować - podzielić wszystkie przecięte krawędzie na dwa odcinki (również nowo powstałą krawędź)

- o tej samej długości co S, ale różniące się współrzędnymi jednego punktu. Punkt ten może występować w kilku krawędziach, więc wektor sąsiada może różnić się na więcej niż jednej pozycji. Należy pamiętać, że nie powinno się manipulować współrzędnymi punktu stanowiącego miasto.

Funkcja celu

Funkcja celu jest zdefiniowana następująco:

$$f = A\sum_{i=1}^{m} length(s_i) + B\sum_{i=1}^{n}\sum_{j=i+1}^{n} distance(M_i, M_j)$$

Gdzie:

A, B- wagi podane na początku uruchomienia przez użytkownika

length - funkcja licząca długość danego odcinka autostrady

distance - funkcja licząca najmniejszą odległość jaką należy przebyć autostradami aby dotrzeć z jednego miasta do drugiego (patrz założenia)

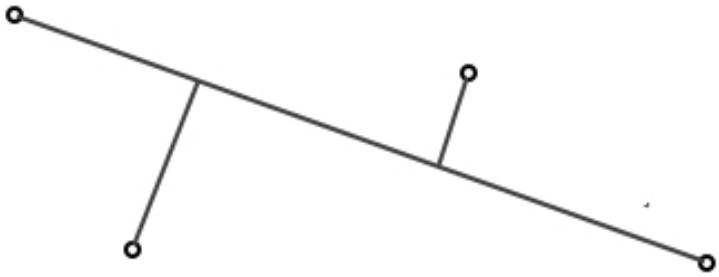
Heurystyka

Wykorzystywanym algorytmem do rozwiązania zadania będzie symulowane wyżarzanie. Algorytm ten wykorzystuje temperaturę T. T początkowa jest z góry ustalona. Będzie ona maleć wraz z kolejnymi iteracjami algorytmu.

Stanem początkowym jest sieć autostrad wyznaczona w następujący sposób:

- Łączymy autostradą dwa najbardziej oddalone od siebie miasta.
- Do głównej (do tej pory jedynej) autostrady przyłączamy prostopadłymi autostradami pozostałe miasta.

Przykład:



Punktem roboczym jest wektor v odcinków, które w sumie łączą każde miasto z każdym.

Kryterium zatrzymania: wyczerpanie liczby iteracji, lub 15 kolejnych iteracji przy, których nie zmienił się punkt roboczy.

Spostrzeżenia, wyniki

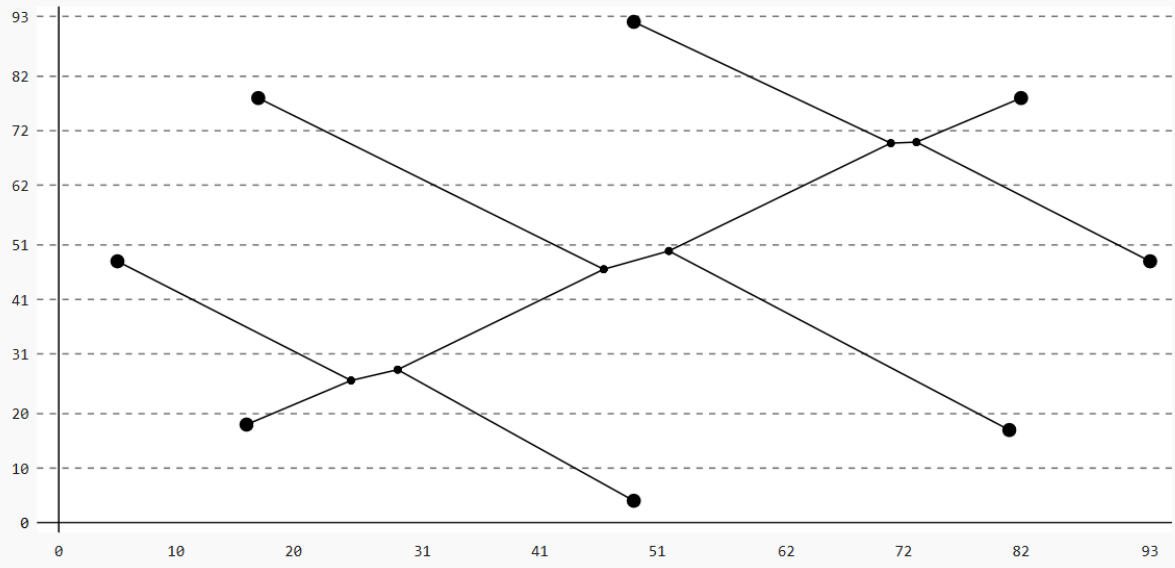
Wynikiem działania jednorazowego uruchomienia naszego programu będzie sieć autostrad. Podczas początkowego testowania dla małej ilości punktów (<8) zauważyliśmy, że sensowne autostrady wychodzą dla parametrów 1 <= a <= 5; 1 <= b <= 5. Zależy, czy zależy nam na tym, aby nie wylać za duzo asfaltu, czy aby drogi między miastami były nie za długie.
Dla większej ilości miast (punktów początkowych) lepiej jest minimalizować ilość wylanego betonu, gdyż, jeśli parametr b będzie większy to ilość miast i odcinków szybko zaczyna rosnąć. Potem jeśli chcemy dodać kolejny odcinek, a raczej taka akcja nastąpi, gdyż minimalizujemy odległość między miastami - b>>a - będziemy musieli sprawdzać przecinanie się bardzo dużej ilości odcinków, punktów, a w konsekwencji, jeśli takie przecięcie nastąpi, będziemy musięli nie tylko nowy odcinek, tylko nowy pocięty odcinek oraz wszystkie linie które przeciął na nowo.

Z drugiej strony, jeśli ustawimy zbyt duże a, to nie powstaną nowe odcinki i jedyne akcje, jakie nastąpią to przesuwanie punktów - skrzyżowań.
Wszystkie trzy nastawy parametrów mają swoje plusy i minusy - zależy jaki efekt chcemy uzyskać.
Wyniki naszych badań są zapisywane jako results+nr. Jest ich całkiem sporo i wszystkich nie będziemy zamieszczać . Są to pliki typu svg i najlepiej oglądać jest autostrady w przeglądarce. Można interaktywnie najechać na każdy punkt oraz odcinek. Są tam także zapisane (w początkowych nie ma) nastawy, dla jakich algorytm został uruchomiony. Wszystkie zostaną spakwane i wysłane razem z projektem

Przykład 1: Dla puntków: 5,48, 93,48, 49,4, 49,92, 16,18, 81,17, 82,78, 17,78

Dla dużego A widzimy, że od stanu początkowego dużo się nie zmienił. Jedyne co, to punkty się poruszyły **result1465134898261**. Dla innego uruchomienia z tymi samymi parametrami uzyskaliśmy podobne wyniki - **result1465134853193**

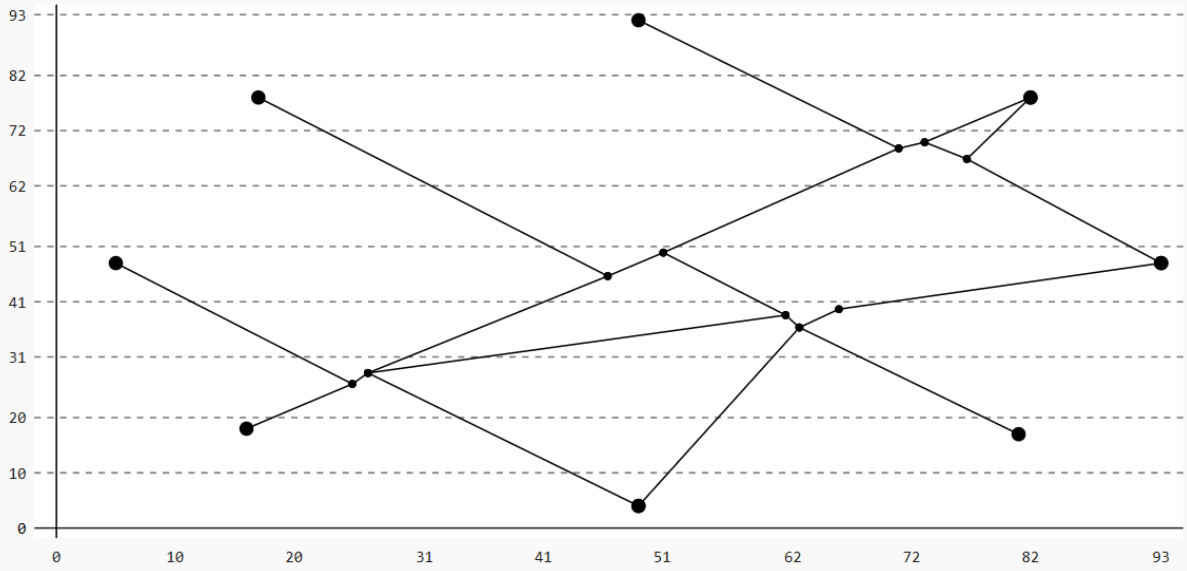
```
4100.64133289
a = 5
b = 1
iterations = 50
ended after = 50
same_state = 15
[(5, 48), (93, 48), (49, 4), (49, 92), (16, 18), (81, 17), (82, 78), (17,
78)]
```



Jak zmnijszyliśmy różnicę między a i b, zaczęło to wyglądać dużo sensowniej

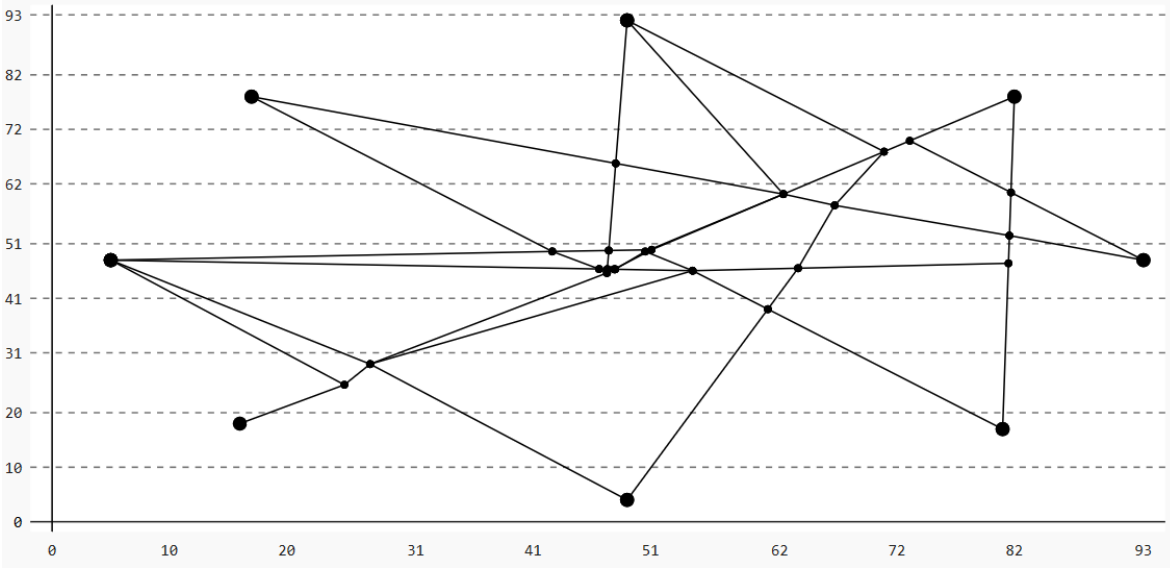
result1465135072639:

```
3211.12938711
a = 2
b = 1
iterations = 50
ended after = 50
same_state = 15
[(5, 48), (93, 48), (49, 4), (49, 92), (16, 18), (81, 17), (82, 78), (17,
78)]
```



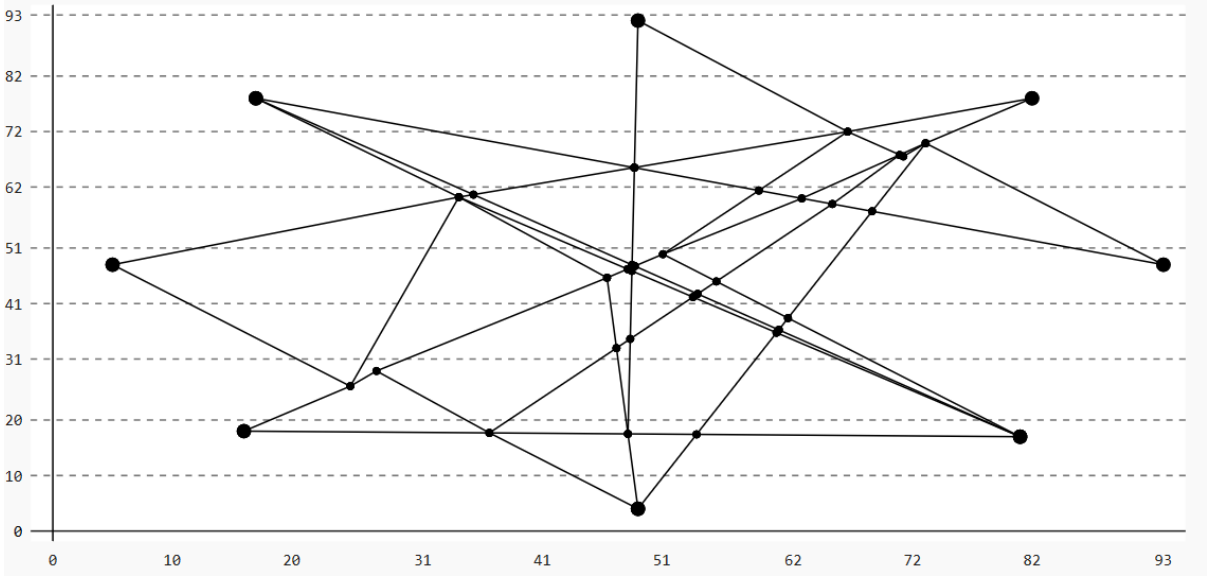
Dla nastawów a=b=1 zaczynają powstawać tuż obok siebie autostrady. Wyniki zaczynają być mniej sensowne, ale jeszcze czytelne **result1465135502199**:

```
2905.24193196
a = 1
b = 1
iterations = 50
ended after = 50
same_state = 15
[(5, 48), (93, 48), (49, 4), (49, 92), (16, 18), (81, 17), (82, 78), (17,
78)]
```



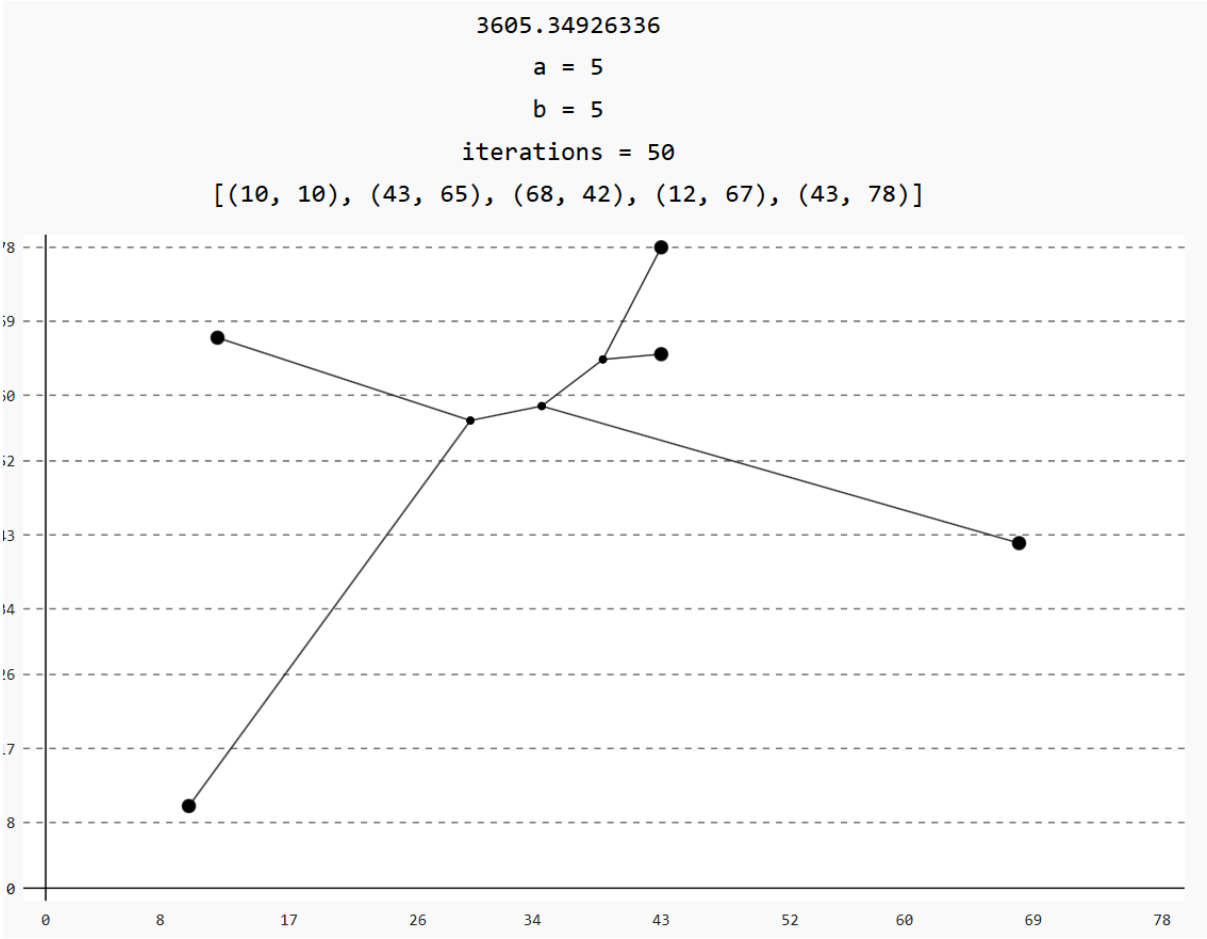
Nie wiem, czy wynik dla nastawu b=5, a=1 jest sensowny w rozumieniu autostrad. To są wyniki **result1465135697792**:

```
11129.9910245
a = 1
b = 5
iterations = 40
ended after = 40
same_state = 15
[(5, 48), (93, 48), (49, 4), (49, 92), (16, 18), (81, 17), (82, 78), (17,
78)]
```



Przykład 2:

Kolejnym całkiem ładnym oraz sensownym przykładem jest **result1464919346761**:



Przykład 3:

Kolejny przykład, który warto omówić to ten, który był konsultowany - pary punktów (1, 1), (1, 25), (25, 1), (25, 25). Myślę, że rysunki same się tłumaczą:

result1464950120394:

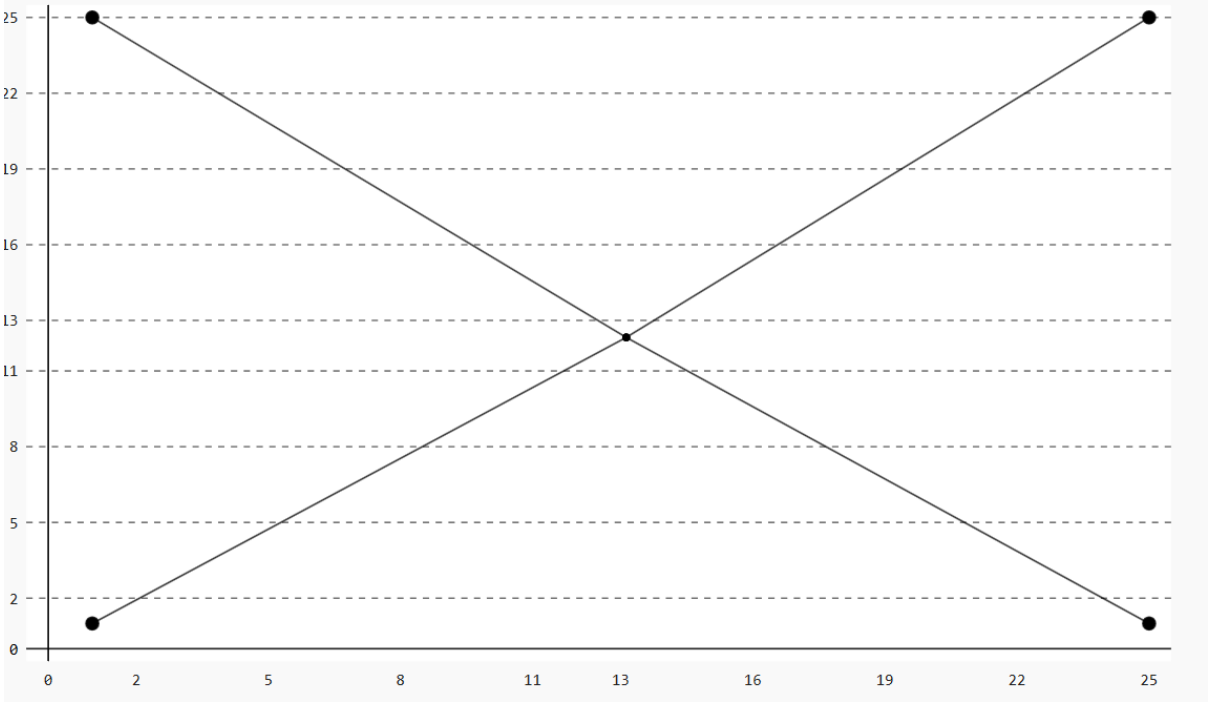
271.638968065

a = 1

b = 1

iterations = 50

[(1, 1), (1, 25), (25, 1), (25, 25)]



result1464950160986:

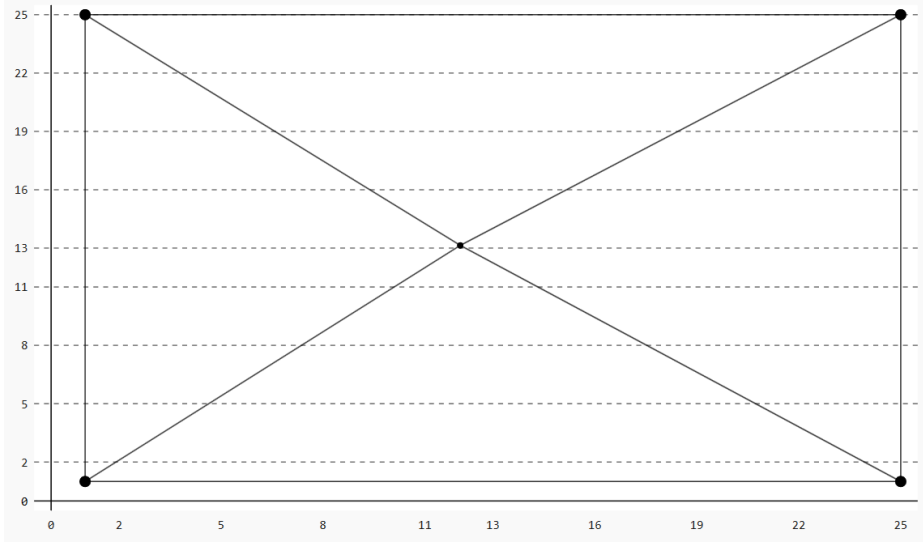
655.751120399

a = 1

b = 3

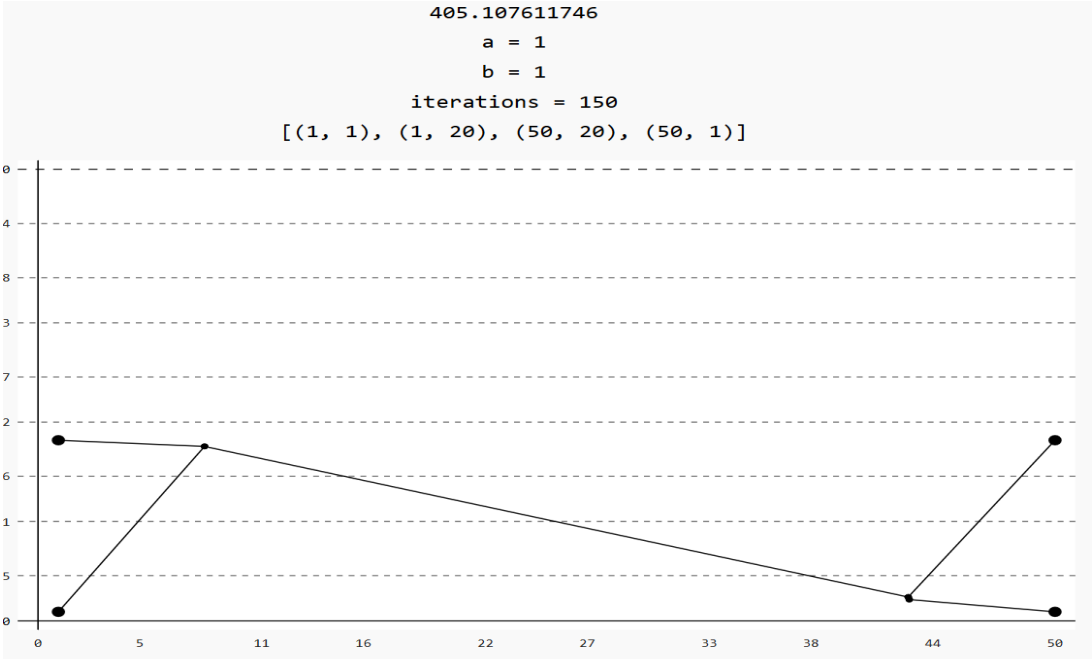
iterations = 50

[(1, 1), (1, 25), (25, 1), (25, 25)]

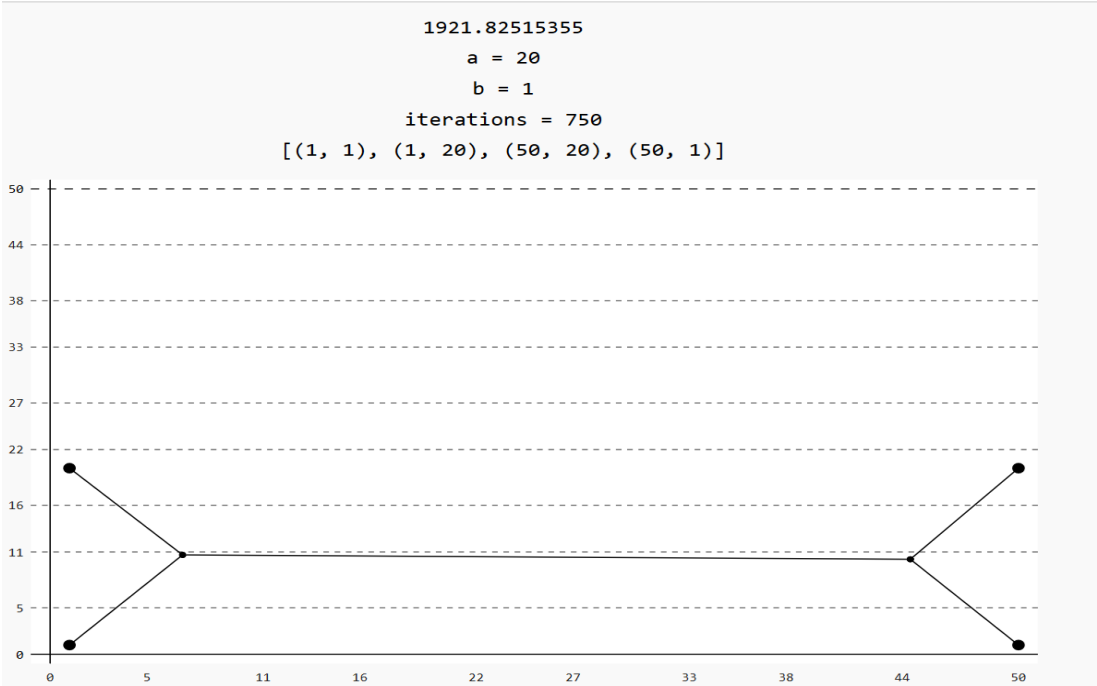


Bardzo pokrewnym i bardziej realistycznym jest przypadek, kiedy skrajne punkty nie leżą na tej samej przekątnej: (1, 1), (1, 20), (50, 20), (50, 1)

result1464953214787:

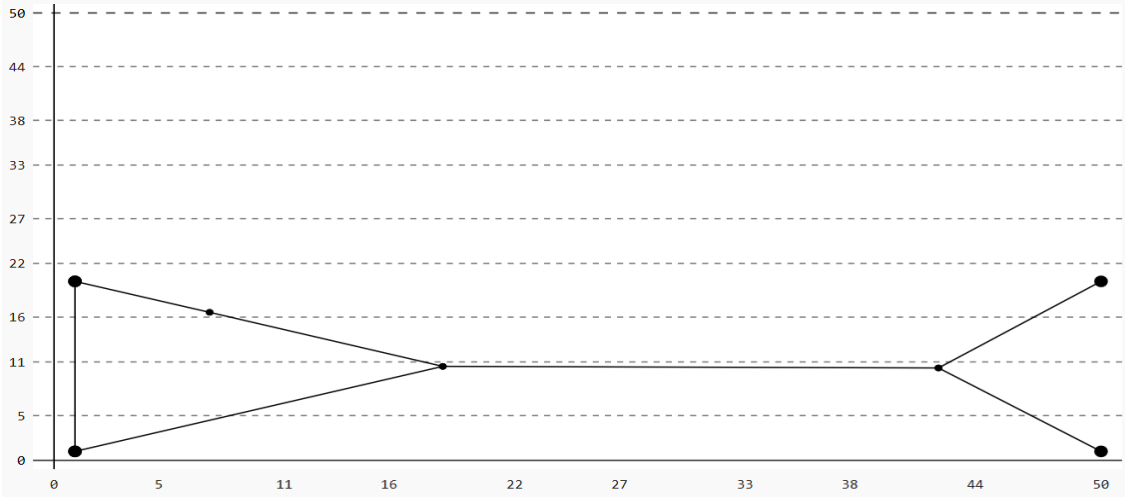


result1464953386220:



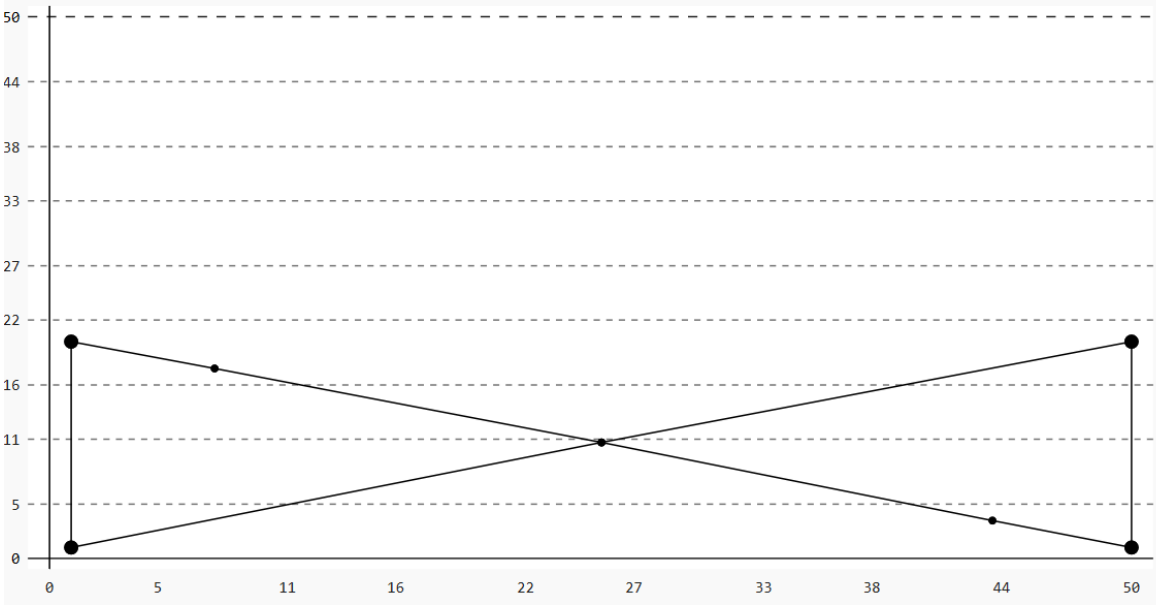
result1464953933017:

```
374.331832684
a = 1
b = 1
iterations = 750
ended after = 564
same_state = 50
[(1, 1), (1, 20), (50, 20), (50, 1)]
```



result1464954037774

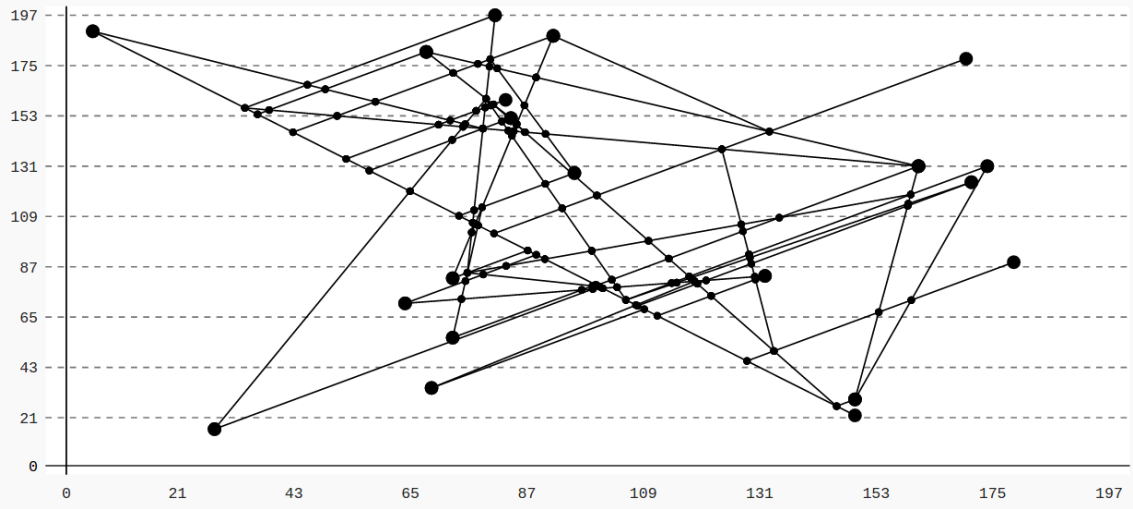
```
515.450299651
a = 1
b = 1.5
iterations = 750
ended after = 156
same_state = 50
[(1, 1), (1, 20), (50, 20), (50, 1)]
```



Luźne przykłady:

result1465051064679:

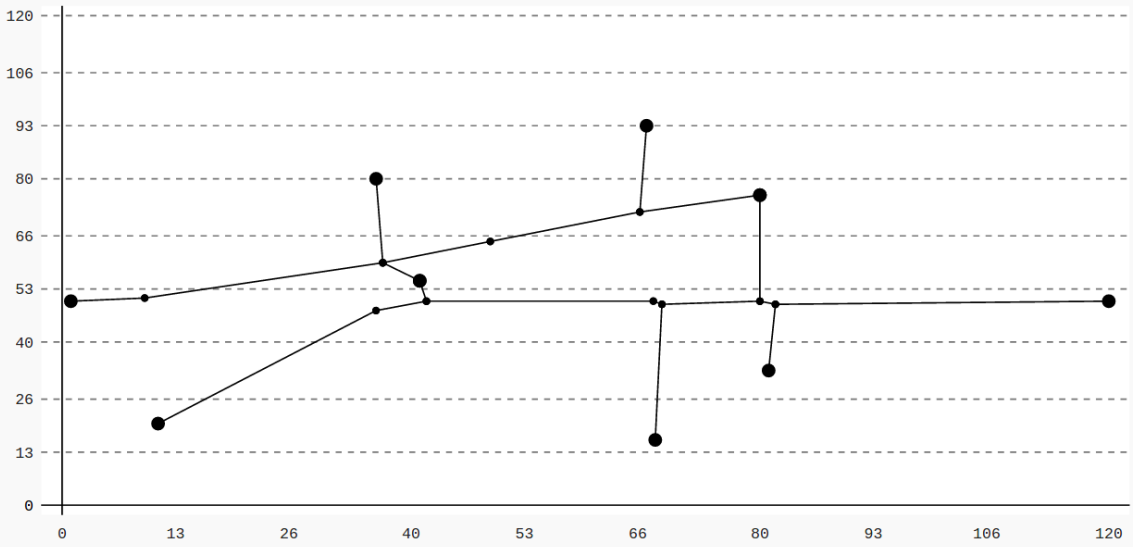
```
71120.8719712
a = 1
b = 3
iterations = 50
ended after = 50
same_state = 50
[(5, 190), (81, 197), (96, 128), (161, 131), (132, 83), (28, 16), (149,
 22), (84, 152), (92, 188), (171, 124), (149, 29), (64, 71), (170, 178),
(179, 89), (68, 181), (83, 160), (73, 56), (73, 82), (69, 34), (174, 131)]
```



Widać, że przy dużej liczbie miast i wysokim współczynniku b, tworzone są znaczne ilości nowych odcinków. W wyniku tego wykonanie czas pracy programu znacznie wzrasta.

result1465093518404:

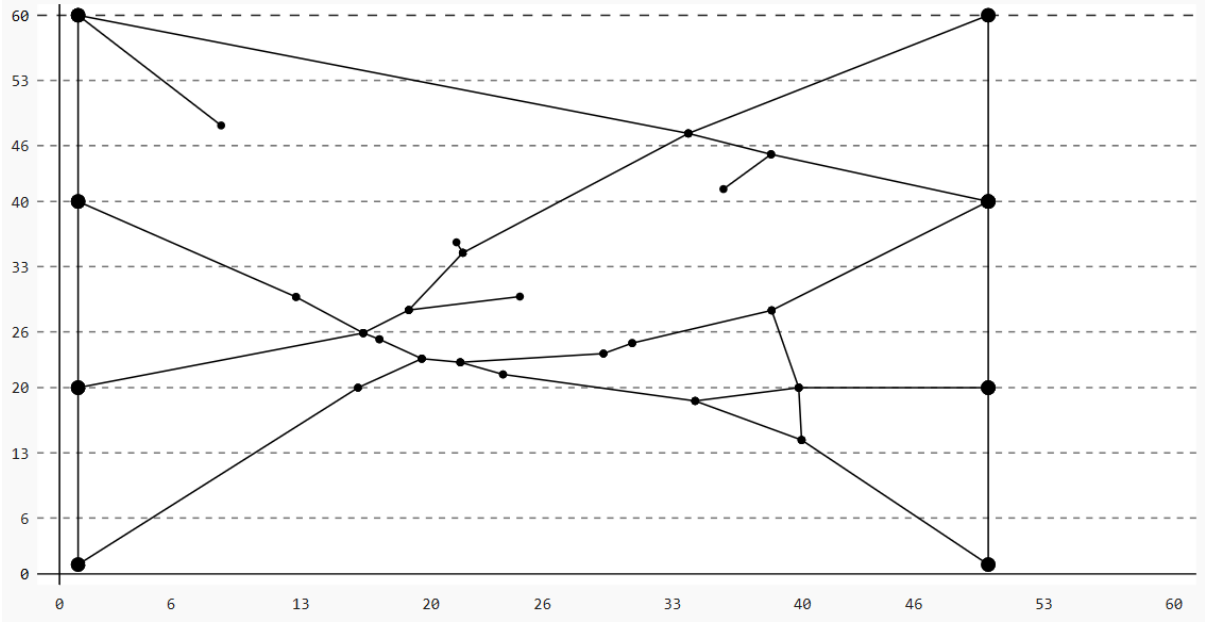
```
3856.38055918
a = 3
b = 1
iterations = 300
ended after = 254
same_state = 25
[(1, 50), (11, 20), (36, 80), (41, 55), (67, 93), (68, 16), (80, 76), (81,
33), (120, 50)]
```



W tym przypadku nie wykonały się wszystkie iteracje, ponieważ w ostatnich 25 nie nastąpiła poprawa.

result1464964905097:

```
1841.28529211
a = 1
b = 1
iterations = 750
ended after = 750
same_state = 50
[(1, 1), (50, 1), (1, 20), (50, 20), (1, 40), (50, 40), (1, 60), (50, 60)]
```



Powyższy przykład pokazuje, że czasami nawet dla dużej ilości iteracji mogą zostać autostrady - odcinki, które prowadzą do nikąd. Tutaj mamy aż 4 takie odcinki.