Podstawy sztucznej inteligencji
Projekt 2
SK.2.12.ant_colony

Tomasz Indeka, 293457

semestr 20L

Spis treści:

1. Wstęp

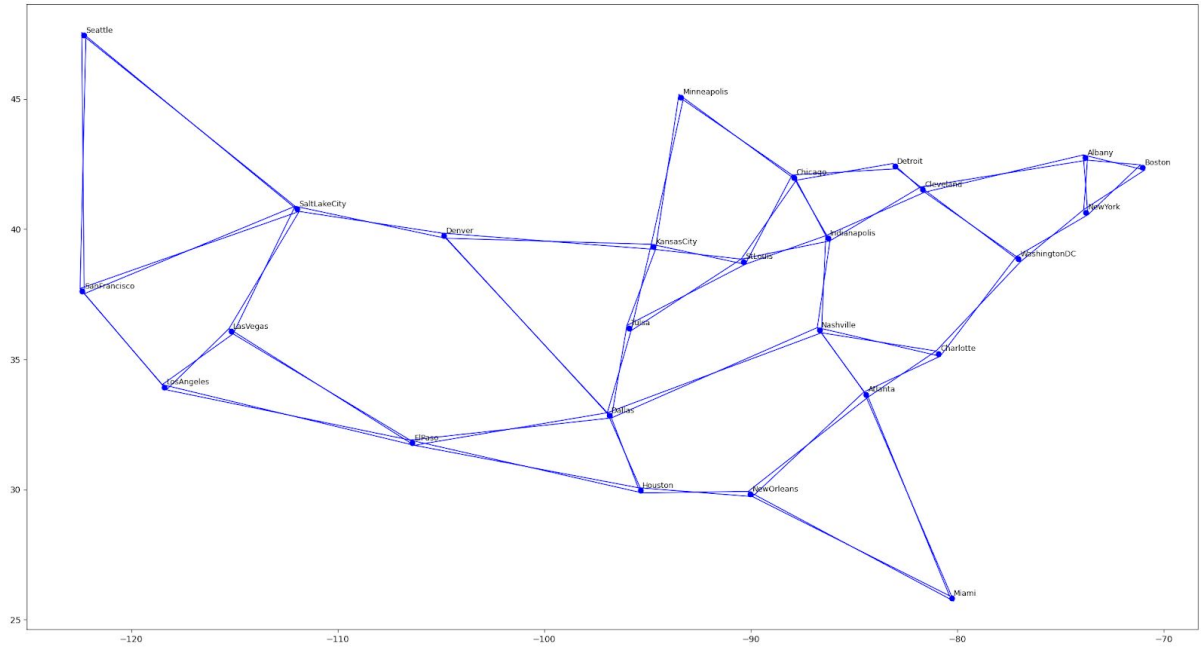
Algorytm mrówkowy jest wzorowany na faktycznym zachowaniu mrówek poszukujących pożywienia. Algorytm dobrze nadaje się do wyszukiwania i optymalizowania trasy w grafach, nawet tych dynamicznie zmieniających się.
Ogólna postać algorytmu prezentuje się następująco:

```
procedure ACO_MetaHeuristic is
    while not_termination do
        generateSolutions()
        daemonActions()
        pheromoneUpdate()
    repeat
end procedure
```

Źródło: https://en.wikipedia.org/wiki/Ant_colony_optimization_algorithms

1.1. Przetwarzane dane

W moim przypadku przeszukiwałem sieć USA, która prezentuje się następująco:



Na grafie są widoczne wszystkie dostępne krawędzie z wyróżnieniem kierunku, co jest przedstawione za pomocą odsunięcia punktów początkowych i końcowych krawędzi nieznacznie od punktu startowego.

1.2. Wykorzystane narzędzia

Projekt został napisany w programie PyCharm Community Edition, w języku programowania Python (v3.7) z wykorzystaniem bibliotek zewnętrznych matplotlib i xml jak również bibliotek standardowych, tj. math, random, statistics, datetime.

2. Implementacja

2.1. Opcje wywołania

Algorytm można skonfigurować korzystając z dostępnych opcji (pełna nazwa (skrót)) podczas wywołania programu:

- generations (i) – liczba generacji (lub iteracji) głównej pętli algorytmu
- ph_min (m) – minimalna wartość feromonu, która musi być na ścieżce
- ph_res (r) – część aktualnej ilości feromonu, która zostaje na ścieżce na kolejną iterację
- demand (d) – numer żądania (predefiniowany początek, koniec i obciążenie trasy), wartość -1 oznacza losowe żądanie
- source (s) – w przypadku definiowania własnego połączenia - miasto początkowe
- target (t) – w przypadku definiowania własnego połączenia - miasto końcowe
- requirement (c) – w przypadku definiowania własnego połączenia - obciążenie trasy
- elimination (e) – wartość 0/1, uruchomienie testowej wersji usuwającej najbardziej wykorzystywane krawędzie w trakcie trwania algorytmu

Daną opcję można zastosować poprzez dodanie po nazwie programu: --<pełna nazwa>=wartość lub -<skrót>=wartość lub -<skrót>wartość.

Przykładowe wywołanie skryptu może wyglądać następująco: `python main.py -e1 -d=452 -generations=2000`

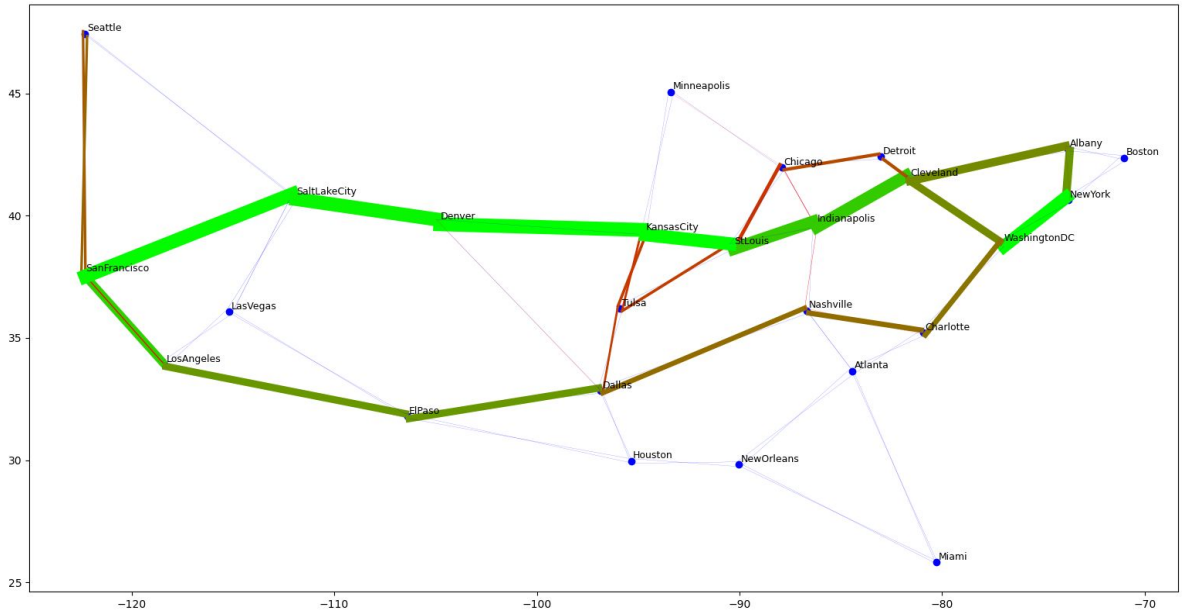
3. Wyniki testów

Głównym wynikiem testów jest aktualizujący się na bieżąco graf połączeń między miastami. Połączenia na grafie są oznaczone liniami. Połączenia wychodzące z danego punktu są zaczepione delikatnie powyżej i na lewo od samego punktu, a połączenia dochodzące poniżej i na prawo od punktu. Szerokość linii oznacza ilość feromonu znajdującego się na danej ścieżce. Kolor linii jest skalowany w zależności od zajętości danej krawędzi - od koloru czerwonego do zielonego wraz z rosnącą zajętością krawędzi. Dodatkowo na niebiesko zostały oznaczone krawędzie, na których nie znajduje się żadna mrówka.

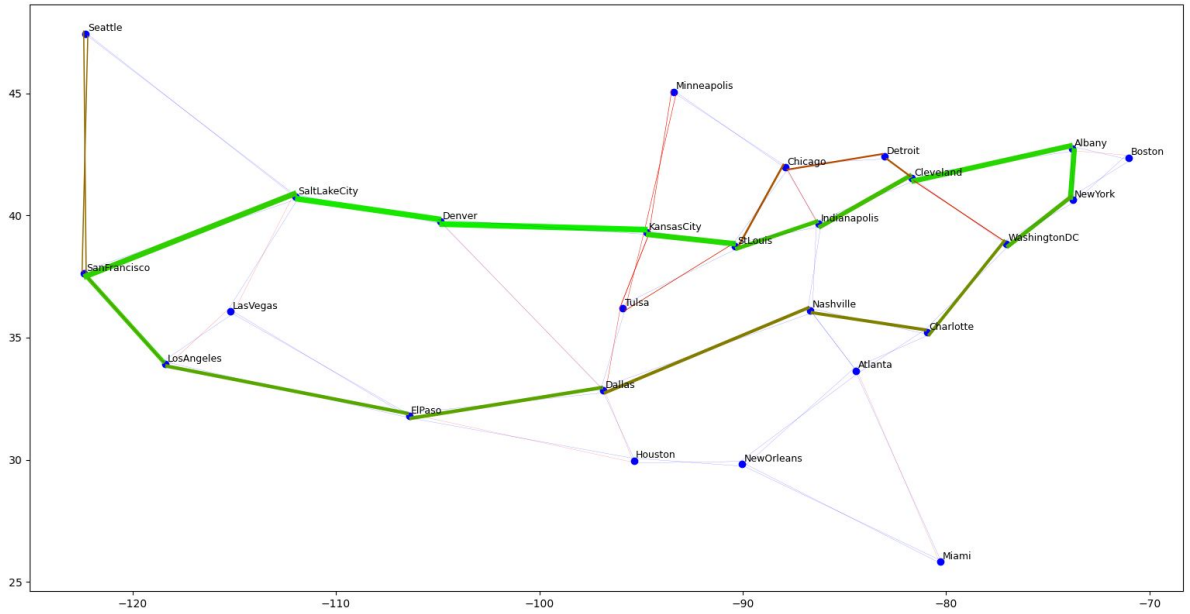
3.1. Testy

Testy zostały wykonane na trasie New York -> San Francisco z obciążeniem 100 i ilością generacji 1000. Podczas testów sprawdzony został wpływ parametrów minimalnej ilości feromonu i ilość pozostającego na trasie feromonu na przebieg wytyczania trasy i finalny wynik.

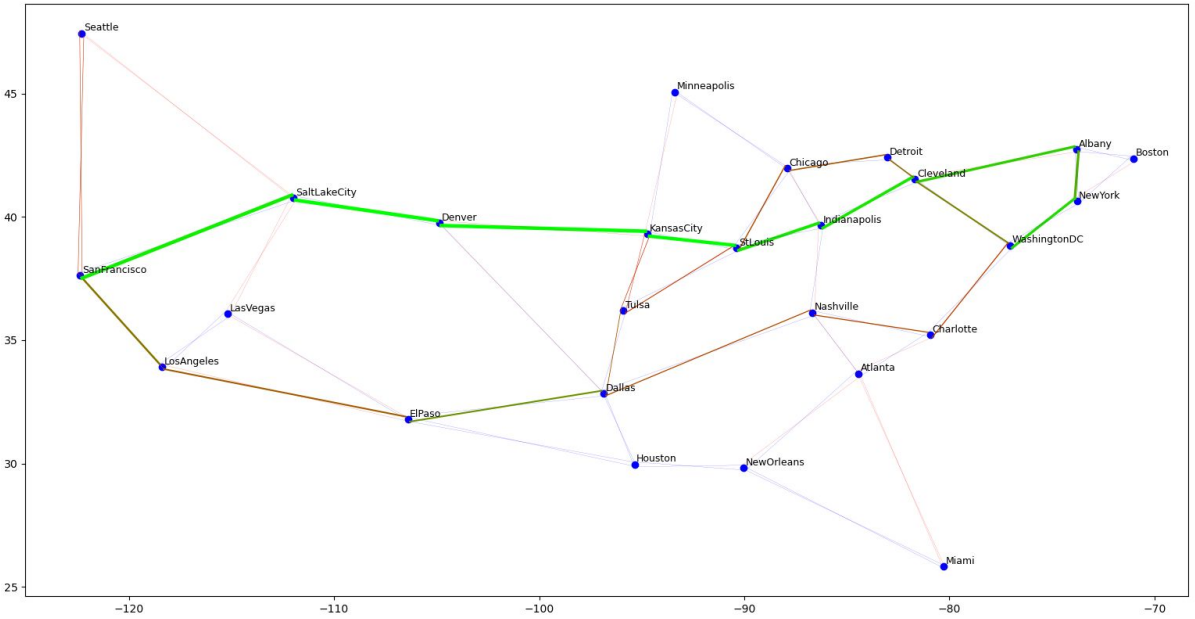
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.9



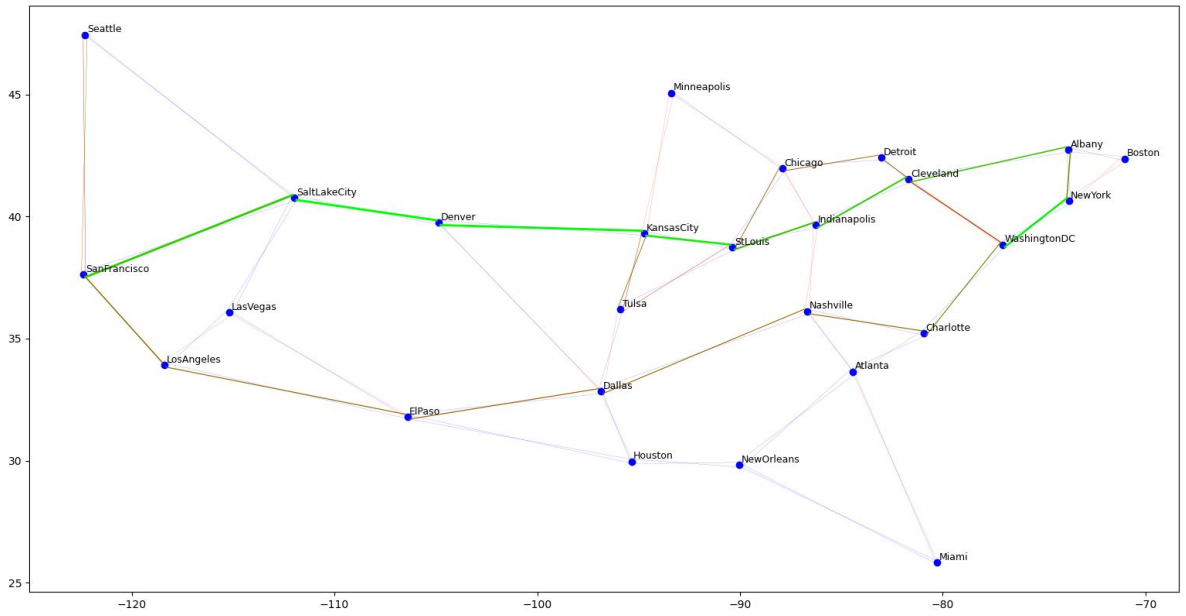
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.8



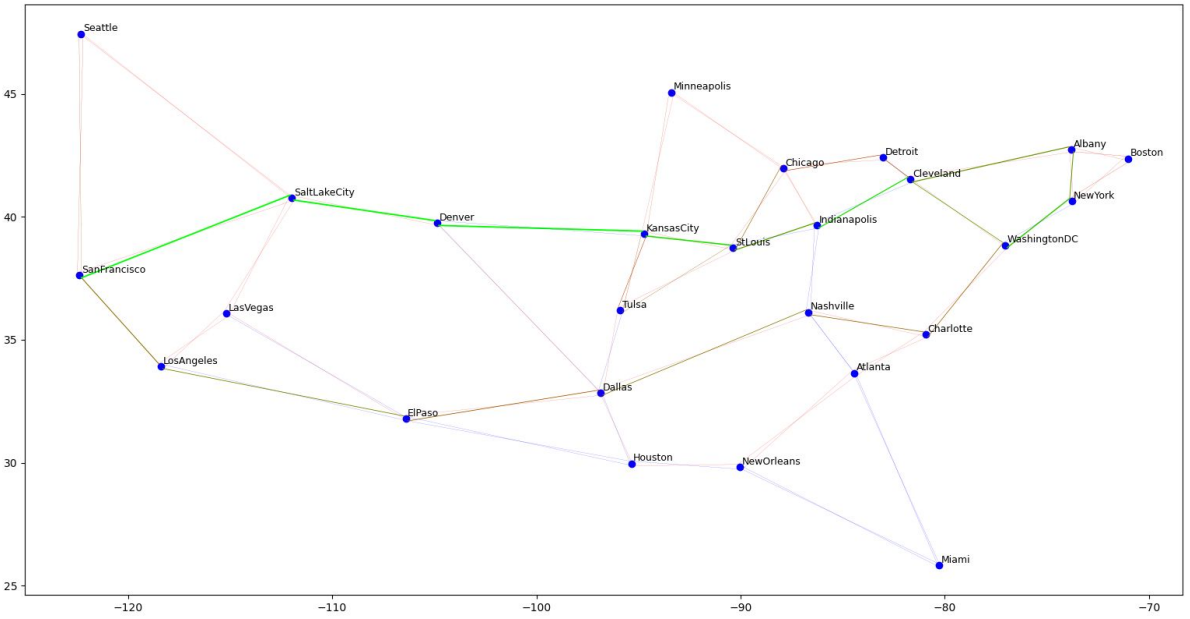
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.7



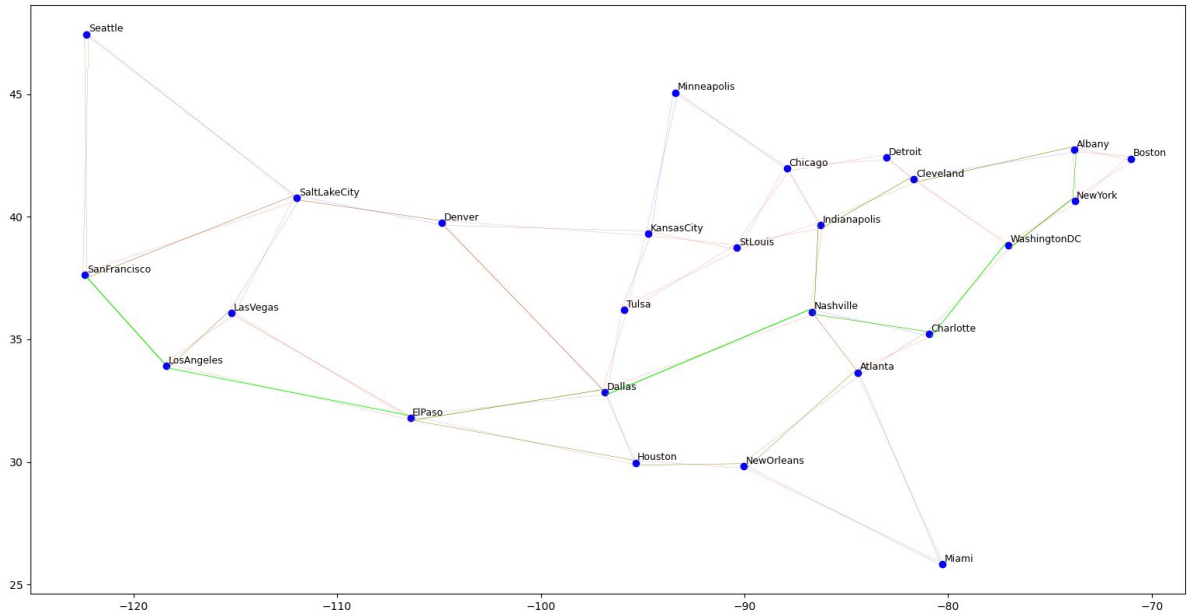
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.6



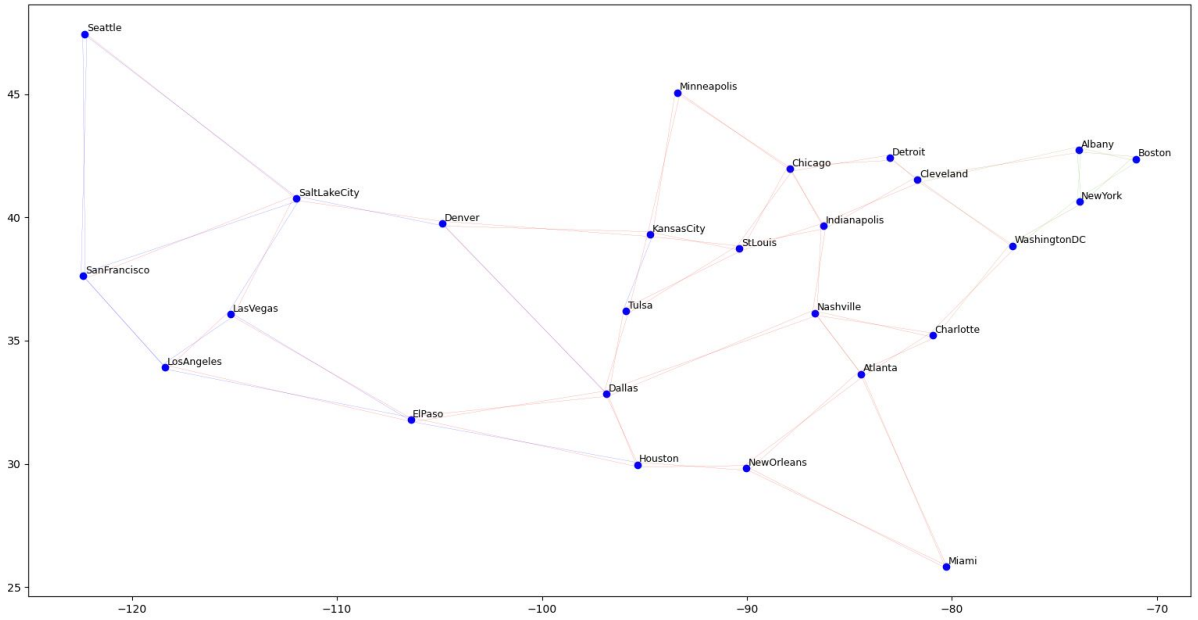
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.5



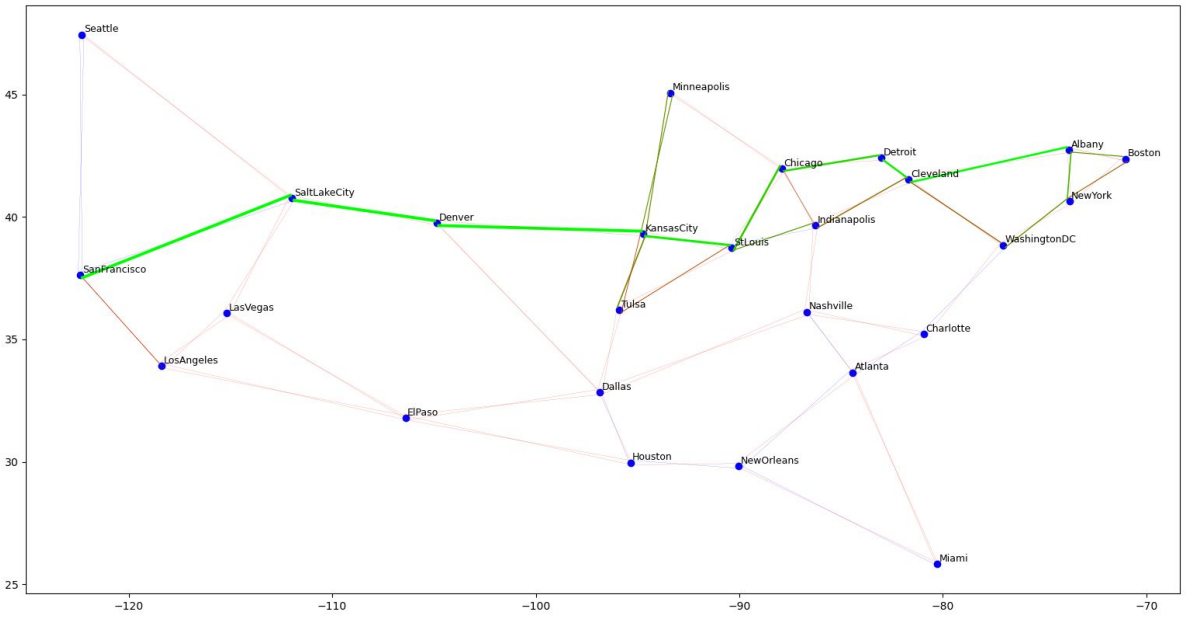
- Minimalna ilość feromonu: 0.1, pozostający feromon: 0.4



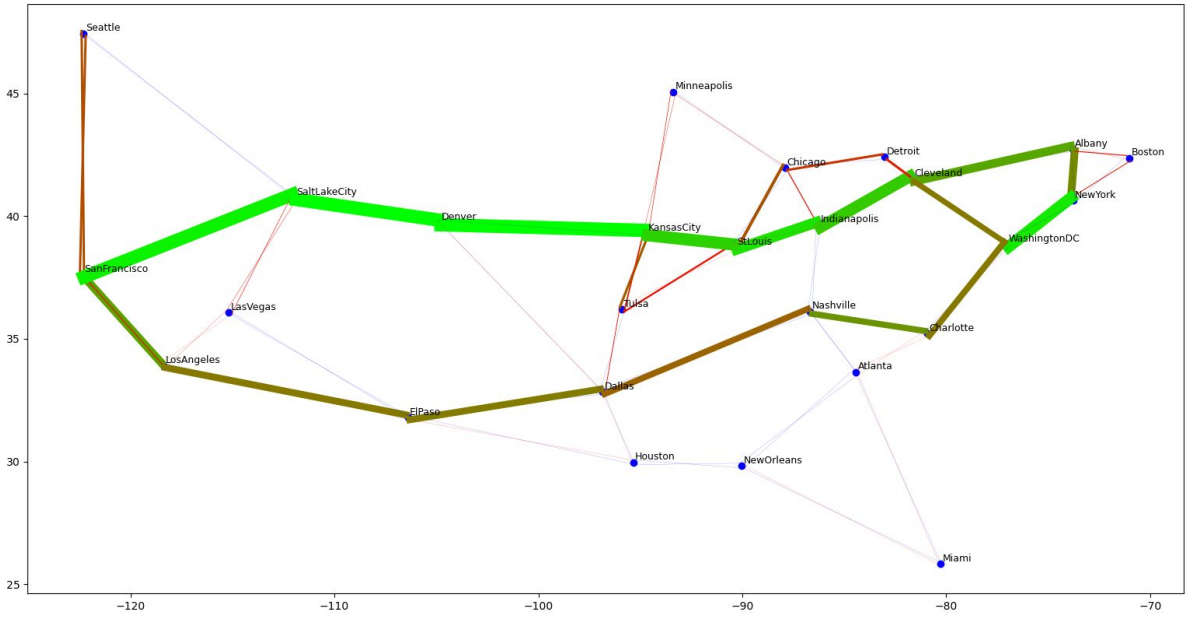
- Minimalna ilość feromonu: 0.5, pozostający feromon: 0.5



- Minimalna ilość feromonu: 0.5, pozostający feromon: 0.7

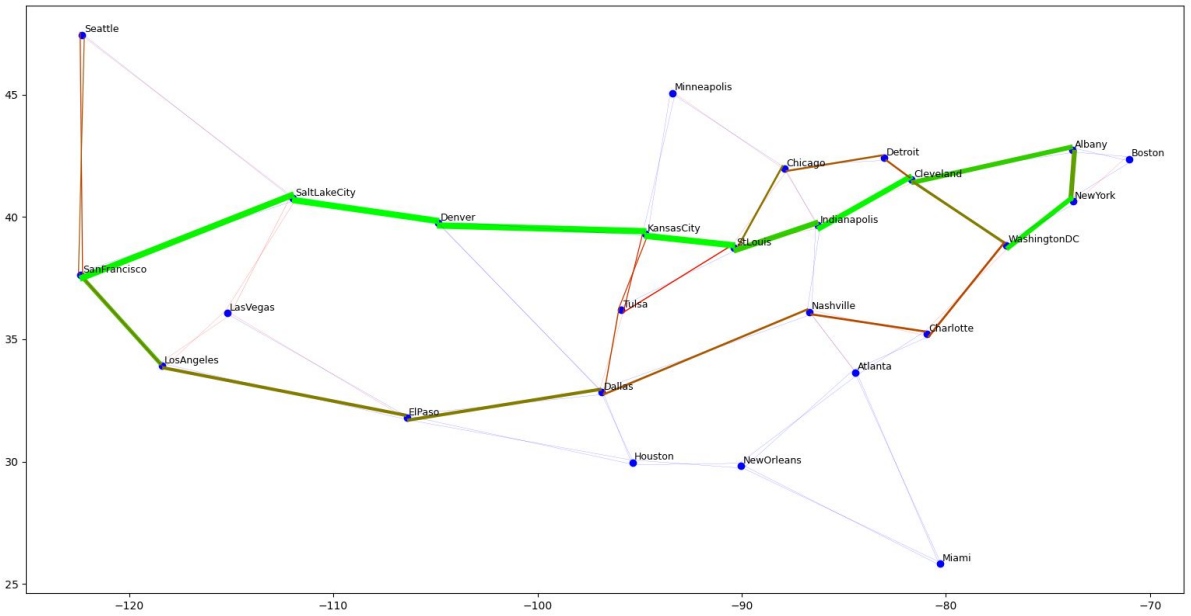


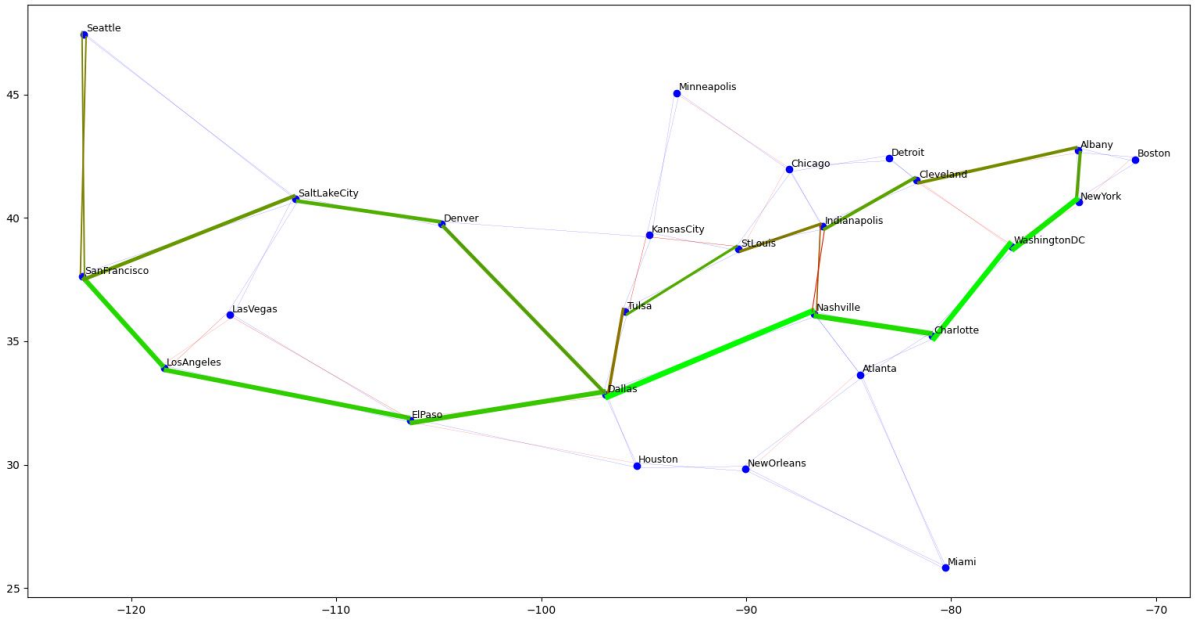
- Minimalna ilość feromonu: 0.5, pozostający feromon: 0.9

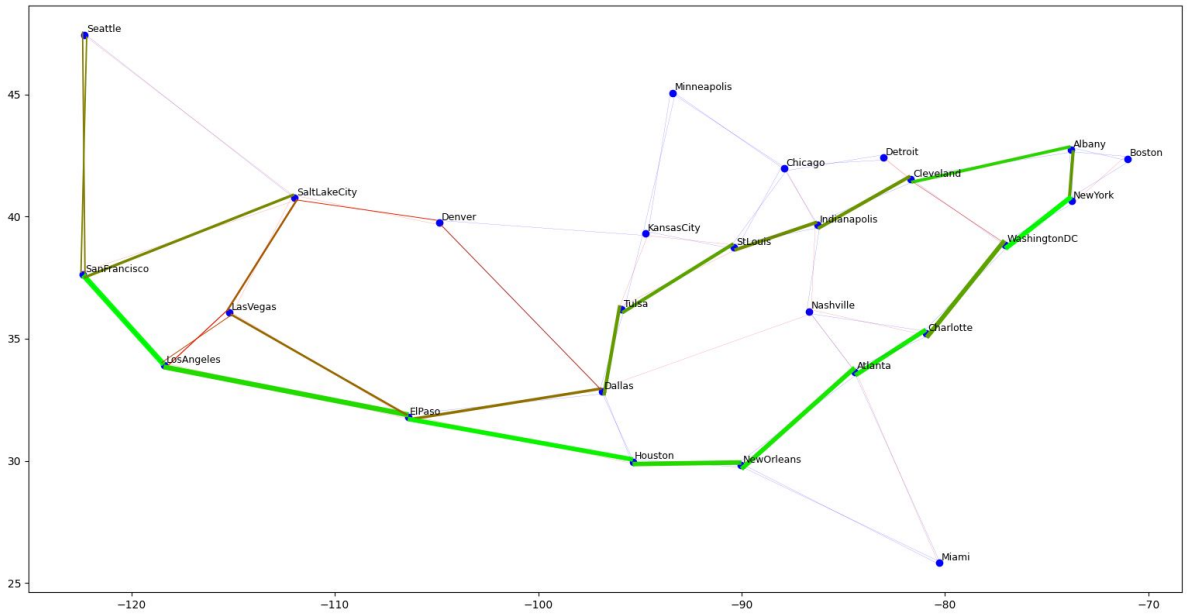


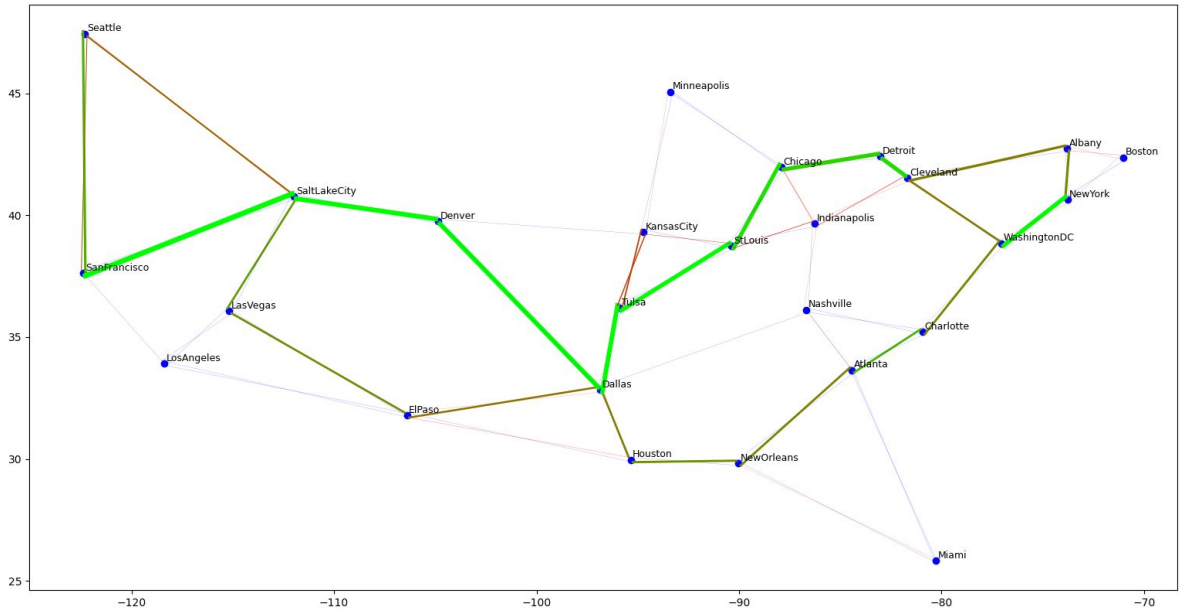
3.2. Odporność na zakłócenia trasy

Testy również zostały wykonane na trasie New York -> San Francisco z obciążeniem 100 i ilością generacji 4000. Podczas testów sprawdzona została odporność najlepszego wybranego algorytmu mrówkowego (minimalna ilość feromonu: 0.1, pozostający feromon: 0.8) na usuwanie najlepszych (z największą ilością feromonu) krawędzi i zdolność adaptacji do nowego środowiska. Kolejne etapy wytyczania trasy są przedstawione poniżej.


Seattle
Minneapolis
Albany
Boston
Detroit
Chicago
Cleveland
SaltLakeCity
NewYork
Denver
KansasCity
Indianapolis
StLouis
WashingtonDC
SanFrancisco
LasVegas
Tulsa
Nashville
Charlotte
LosAngeles
Atlanta
Dallas
ElPaso
Houston
NewOrleans
Miami
45
40
35
30
25
−120
−110
−100
−90
−80
−70


Seattle
Minneapolis
Albany
Boston
Detroit
Chicago
Cleveland
SaltLakeCity
NewYork
Denver
KansasCity
Indianapolis
StLouis
WashingtonDC
SanFrancisco
LasVegas
Tulsa
Nashville
Charlotte
LosAngeles
Atlanta
Dallas
ElPaso
Houston
NewOrleans
Miami
45
40
35
30
25
−120
−110
−100
−90
−80
−70


Seattle
Minneapolis
Albany
Boston
Detroit
Chicago
Cleveland
SaltLakeCity
NewYork
Denver
KansasCity
Indianapolis
StLouis
WashingtonDC
SanFrancisco
Tulsa
Nashville
LasVegas
Charlotte
LosAngeles
Atlanta
Dallas
ElPaso
Houston
NewOrleans
Miami
45
40
35
30
25
−120
−110
−100
−90
−80
−70


Seattle
Minneapolis
Albany
Boston
Detroit
Chicago
Cleveland
SaltLakeCity
NewYork
Denver
KansasCity
Indianapolis
StLouis
WashingtonDC
SanFrancisco
Tulsa
Nashville
LasVegas
Charlotte
LosAngeles
Atlanta
Dallas
ElPaso
Houston
NewOrleans
Miami
45
40
35
30
25
−120
−110
−100
−90
−80
−70

4. Podsumowanie

Wykresy końcowe są do siebie bardzo podobne i nie ma między nimi większej różnicy, poza ilością feromonu na ścieżce (szerokością na wykresie). Podczas obserwacji w czasie rzeczywistym zauważyłem, że zwiększona pozostawalność feromonu przyspiesza szukanie optimum trasy. Wydaje się również, że zwiększona opzostawalność feromonu zwiększa ilość mrówek, które dotarły do celu, ponieważ linie wiodące są bardziej zielone niż w przypadku zmniejszonej trwałości feromonu. Jest to także logicznie wytłumaczalne, ponieważ przy mniejszej ilości feromonu częściej wybierają losowe krawędzie co zwiększa prawdopodobieństwo, że mrówka kilkukrotnie wybierze tą samą krawędź. W takim przypadku moja implementacja algorytmu mrówkowego zabija taką mrówkę, ponieważ wykonała pętlę i informacja wnoszona przez nią jest potencjalnie bezużyteczna.
Zwiększanie minimalnej ilości feromonu na ścieżce jest przewrotnie parametrem, który pogarsza znajdowanie prawidłowej trasy. W niektórych kombinacjach, szczególnie z niską trwałością feromonu algorytm wcale nie potrafi wytyczyć prawidłowej drogi do celu. Wraz ze zwiększaniem się pozostawionego feromonu sytuacja się poprawia, ale sam algorytm wciąż jest mniej stabilny i może nie znajdować optymalnych rozwiązań.
Odporność na zakłócenia algorytmu mrówkowego z dobranymi parametrami jest bardzo dobra. Algorytm bazując na poprzedniej trasie stara się znaleźć nową okrążającą wyłączoną z ruchu krawędź. Jeśli nie potrafi znaleźć nowej trasy w pobliżu starej to słabnąca ilość feromonu na krawędziach powoduje, że algorytm zaczyna działać w sposób bardziej przeszukujący, podobnie do tego kiedy jeszcze nie znalazł żadnej trasy. Po znalezieniu trasy algorytm wciąż poprawia znalezione rozwiązanie, aż do znalezienia minimum.
Napisany przeze mnie algorytm posiada także małe niedociągnięcia - niektóre mrówki po dotarciu do celu wykonują kolejny krok mimo, iż nie powinny. Na ten moment nie udało mi się tego błędu poprawić.