# Sprawozdanie środkowo-końcowe

## Zastosowanie algorytmu mrówkowego do rozwiązywania problemu komiwojażera

Zaprojektować algorytm mrówkowy (np. Ant Colony) do rozwiązywania problemu TSP. Zaprojektować reprezentację rozwiązania oraz optymalnie dobrać parametry metody. Przetestować algorytm dla różnych parametrów oraz grafów o różnej wielkości. Zilustrować proces uczenia.

## Opis problemu

Problem komiwojażera (z ang. TSP – *travelling salesman problem*) jest to zagadnienie z teorii grafów, polegające na znalezieniu minimalnego cyklu Hamiltona (ścieżki o początku i końcu w tym samym punkcie, przechodzącej dokładnie raz przez każdy wierzchołek) w pełnym grafie ważonym. Problem należy do klasy NP-zupełnych, a więc nie istnieje dokładny algorytm jego rozwiązania o złożoności wielomianowej. Z tego powodu w celu jego rozwiązania stosuje się różnego rodzaju algorytmy heurystyczne (takie jak sztuczne sieci neuronowe, symulowane wyżarzanie czy algorytmy mrówkowe).

Istnieje kilka odmian problemu TSP, w szczególności symetryczna (gdzie odległość pomiędzy wierzchołkami w grafie jest taka sama w obie strony) i asymetryczna (gdzie odległość od A do B może być inna, niż odległość z B do A). W rozwiązaniu skupimy się na problemie symetrycznym.

## Opis zaimplementowanego algorytmu

Algorytmy mrówkowe są probabilistyczną techniką rozwiązywania problemów poprzez szukanie dobrych dróg w grafach. Inspirowane są zachowaniem mrówek szukających pożywienia, gdzie każdy z osobników porusza się losowo po otoczeniu poszukując pożywienia. Po jego odnalezieniu wraca do kolonii pozostawiając za sobą silny ślad feromonowy. Inne osobniki po natrafieniu na ten ślad przestają poruszać się losowo, a zaczynają nim podążać w kierunku pożywienia. Wraz z upływem czasu ślad ten słabnie, z drugiej strony wzmacniany jest przez podróżujące tą trasą mrówki. Ostatecznie łącząc zachowania losowej eksploracji otoczenia i podążania za śladami feromonowymi mrówki potrafią odnaleźć najkrótszą drogę z kolonii do źródła pożywienia [1].

W przypadku algorytmów mrówkowych możemy wyróżnić trzy podstawowe fazy [2]:

1. pseudolosowe poruszanie się agentów (mrówek) w grafie wykorzystując informacje o sile feromonów na ścieżkach oraz odległościach do sąsiednich miast, aż do momentu odwiedzenia wszystkich miast,
2. proces wyparowywania feromonów z czasem (zmniejszanie siły feromonów),
3. intensyfikacja feromonów na ścieżkach odpowiadających aktualnie najlepszym rozwiązaniom.

W pierwszym procesie prawdopodobieństwo, że agent znajdujący się w punkcie *i* wybierze drogę do punktu *j* można określić wzorem:

Gdzie oznacza siłę feromonu na ścieżce z *i* do *j*, jest dodatkową heurystyką związaną z wyborem ścieżki z *i* do *j* (w tym przypadku jest to odwrotność jej długości), a zbiór *S* zawiera wszystkie możliwe wierzchołki, do których może przejść w danym momencie agent znajdujący się w *j* (a więc takie, których jeszcze nie odwiedził).

W tym miejscu należy także wspomnieć o początkowych wartościach feromonu na ścieżkach – w naszym programie zaimplementowaliśmy możliwość jego ustalenia w jednym z kilku stanów:

1. inicjowanie wszystkich ścieżek jednakową wartością (np. 0 lub 1),
2. inicjowanie ścieżek wartościami losowymi z przedziału [0, 1].

Dla każdego z przypadków przeprowadzono serię testów mających na celu określenie, w jaki sposób początkowe wartości siły feromonu wpływają na jakość otrzymywanych rozwiązań i szybkość ich uzyskiwania.

Wyparowywanie i intensyfikację feromonów można opisać prostym wzorem:

Gdzie *q* jest pewnym, ustalonym parametrem odpowiedzialnym za tempo słabnięcia feromonów, a jest współczynnikiem wzmocnienia (w przypadku krawędzi należących do najkrótszej znanej w danym cyklu ścieżki jest większa od zera, a jest zerowa dla krawędzi nienależących do niej).

## Kryterium stopu

Powtarzając odpowiednio wiele razy cały cykl, ostatecznie można dojść do rozwiązań zbliżonych do optymalnych. Kluczowym elementem jest dobre określenie warunku zakończenia obliczeń. Przewidziane są dwie możliwości:

1. wyczerpanie założonej z góry liczby rund,
2. ustabilizowanie wyniku (przez określoną liczbę kolejnych rund znalezione rozwiązanie się nie zmienia).

Przeprowadzane były też eksperymenty z innym warunkiem stopu – przetwarzanie było kończone dopiero w momencie, kiedy wszystkie mrówki chodziły po tej samej ścieżce (a dokładniej wtedy, kiedy najkrótsza i najdłuższa ścieżka w danej rundzie były tej samej długości). Warunek ten w ogólności nie powodował polepszenia wyników, wydłużał jedynie czas działania całego programu.

## złożoność obliczeń

Na złożoność obliczeniową wpływają dwa parametry:

1. liczba mrówek w algorytmie,
2. liczba miast w zbiorze testowym.

W pierwszym przypadku w algorytmie występuje pętla for wykonująca obliczenia dla każdej mrówki, a ich liczba wewnątrz pętli nie zależy od liczności mrówek. Z tego względu złożoność ze względu na liczbę mrówek jest liniowa. Doświadczenia przeprowadzane na konkretnych danych potwierdzają tę hipotezę. Przeprowadzono serię obliczeń dla tego samego zbioru testowego (zawierającego 16 miast) zwiększając kolejno liczbę mrówek, od 1 do 200. Zebrane wyniki (średni czas wykonania pojedynczej rundy) zebrano w tabeli 1.

Tabela : Zależność czasu wykonania pojedynczej rundy od liczby mrówek (dla ustalonej liczby miast, tutaj 16)

|  |  |
| --- | --- |
| **N - ilość mrówek** | **Czas epoki [s]** |
| 1 | 0,0065 |
| 5 | 0,0213 |
| 10 | 0,0398 |
| 20 | 0,0763 |
| 30 | 0,1132 |
| 50 | 0,1858 |
| 100 | 0,3705 |
| 200 | 0,7322 |

Na rysunku 1 pokazany jest wykres zależności czasu wykonania od liczby mrówek, a także wykres prostej o równaniu , gdzie N jest liczbą mrówek a współczynnikiem skalującym (na maszynie testowej wyposażonej w procesor klasy AMD Turion o taktowaniu 1.9GHz wynosi on .

Wpływ liczby miast na szybkość działania algorytmu jest trudniejszy do oceny. W danej rundzie każda mrówka najpierw sprawdza N-1 miast, następnie N-2 itd., aż do momentu, kiedy zostanie tylko jedno nieodwiedzone miasto. Ostatecznie wykonuje więc sprawdzeń. Można więc ograniczyć złożoność z góry przez .

Tabela : Zależność czasu wykonania pojedynczej rundy od liczby miast (dla stałej liczby mrówek, w tym przypadku 5)

|  |  |
| --- | --- |
| **N - ilość miast** | **Czas epoki [s]** |
| 16 | 0,0219 |
| 52 | 0,0697 |
| 70 | 0,0947 |
| 127 | 0,1769 |
| 280 | 0,4354 |
| 493 | 0,8986 |
| 574 | 1,1200 |
| 724 | 1,5317 |
| 1060 | 2,6940 |
| 1323 | 3,8446 |
| 1400 | 4,2320 |

Złożoność wielomianowa jest w tym przypadku wynikiem bardzo dobrym. W tabeli 2 zebrane zostały pomiary pokazujące czas przetwarzania pojedynczej epoki dla stałej liczby mrówek przy zwiększającej się liczbie miast w zbiorze testowym. Na rysunku 2 przedstawiony jest wykres pokazujący zarówno rzeczywisty czas przetwarzania, jak i funkcję kwadratową (o pewnym współczynniku dobranym eksperymentalnie). Widać, że od pewnego N (w tym przypadku ok. 700) czas przetwarzania rośnie wolniej niż kwadratowo wraz ze wzrostem liczby miast.

Rysunek : Zależność czasu wykonania pojedynczej rundy od liczby mrówek (dla ustalonej liczby miast, tutaj 16). Dodatkowo pokazana funkcja określająca czas wykonania dla złożoności liniowej.

Rysunek : Zależność czasu wykonania pojedynczej rundy od liczby miast. Dodatkowo pokazana funkcja kwadratowa ograniczająca z góry czas wykonania algorytmu.

# Testowanie

Przygotowany algorytm został przetestowany przy użyciu standardowych zbiorów testowych dla problemu TSP, pozyskanych z bazy danych projektu TSPLIB[[1]](#footnote-1), dzięki czemu możliwe było porównanie jakości otrzymanych rozwiązań z optymalnymi ścieżkami znalezionymi już dla danych zbiorów. Przeprowadzone zostały testy mające za zadanie zbadanie wpływu wszystkich parametrów systemu na jakość i szybkość generowania rozwiązań. Ze względu na konieczność wykonania wielu powtórzeń procedury, do testów wybrany został zbiór **berlin52**, zawierający graf o 52 wierzchołkach. Jest to ilość na tyle duża, że problem nie jest trywialny, jednocześnie przetwarzanie trwa jeszcze na tyle krótko, żeby można było doczekać się wyników przed emeryturą.

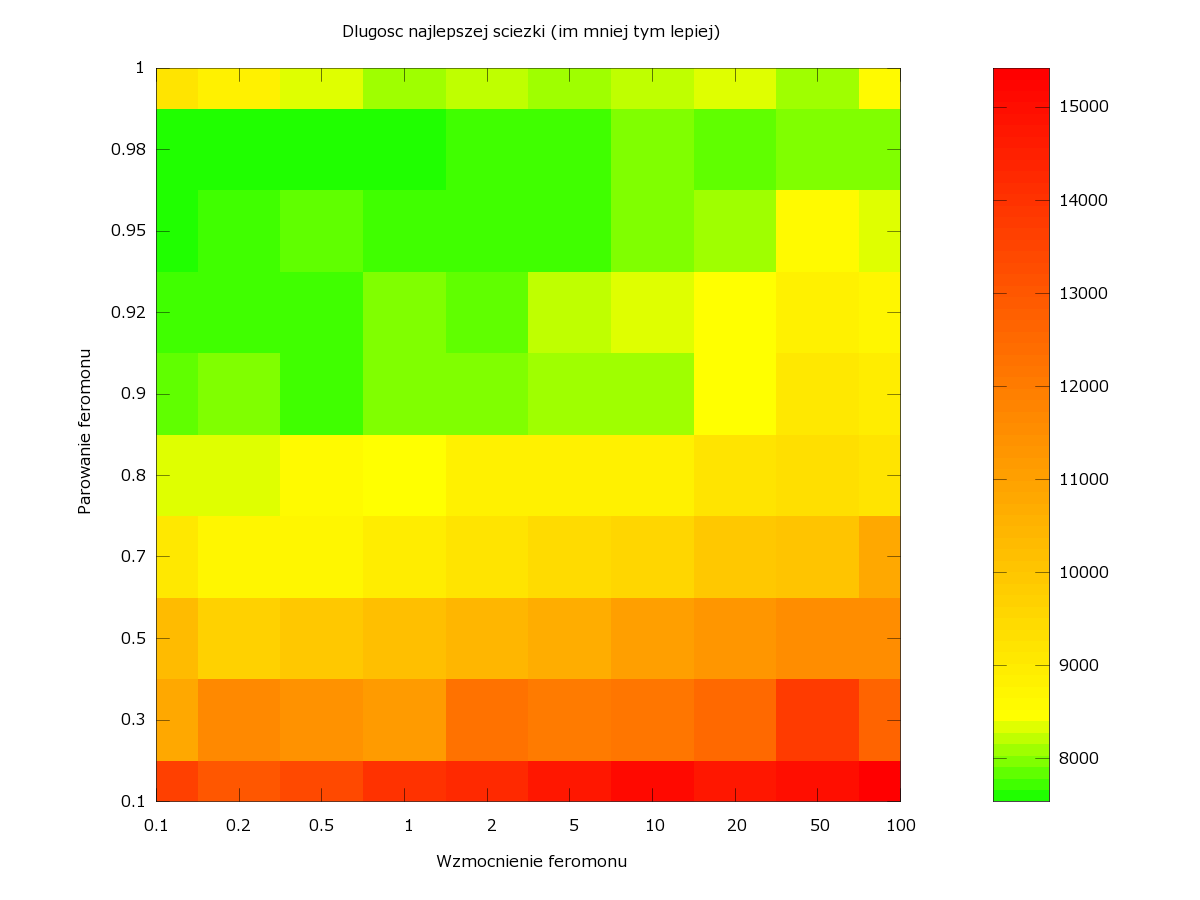
## ilość mrówek

W przypadku tego parametru można od razu stwierdzić jego wpływ na działanie algorytmu – im więcej mrówek wpuścimy do „labiryntu”, tym szybciej znajdą one rozwiązanie.

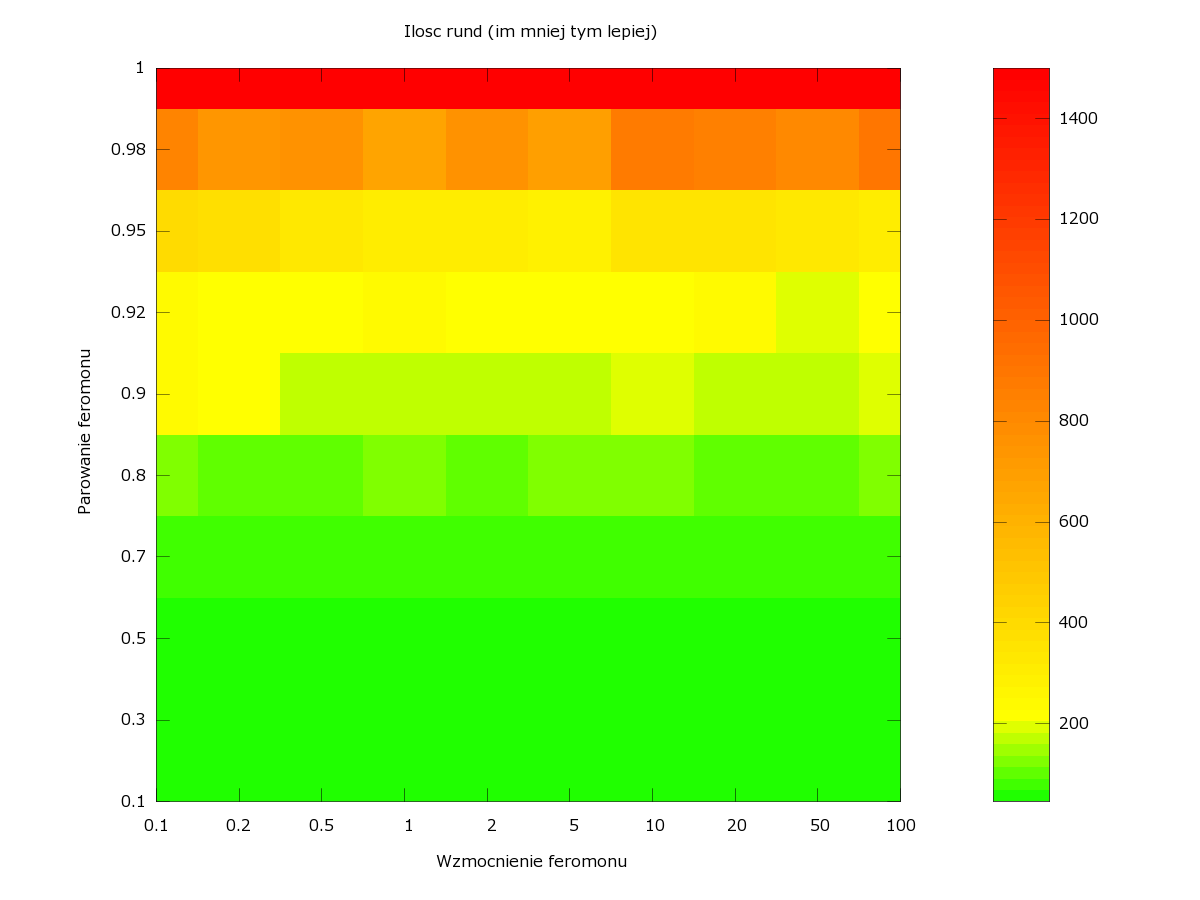
## współczynnik wyparowywania feromonu

## współczynnik wzmacniania dobrych ścieżek

Im większe wzmacnianie tym słabsze wyniki osiągane (dłuższe ścieżki), czas przetwarzania się nie zmienia.



Rysunek 3. Zależność długości znajdowanej ścieżki od wartości parametrów metody.



Rysunek 4. Zależność ilości rund, po której zostanie odnaleziona ścieżka, od wartości parametrów metody.

## sposób generowania początkowej siły feromonu

Ostatnim parametrem, który można zmieniać w naszym programie, jest początkowa siła feromonów na ścieżkach. Może być ona losowana (wtedy dla każdej ścieżki losowana jest wartość z przedziału [0, 1]) bądź ustalona na tym samym poziomie dla wszystkich ścieżek. Testy tego parametru mają dać odpowiedź, czy ma on jakikolwiek wpływ, a jeśli tak, to jaki, na szybkość znajdowania rozwiązań i ich jakość.

Przewidywane znaczenie parametru: dla stałej wartości feromonu, początkowo mrówki będą się poruszały biorąc pod uwagę jedynie odległości między miastami (prawdopodobieństwo jest ważone wartością feromonu, więc dla jednakowych wartości wpływ ma tylko odległość). W przypadku, kiedy wylosujemy te wartości, możliwe stanie się wybranie przez mrówki od razu innej ścieżki niż wskazywana przez początkowo najkrótsze połączenie, a ostatecznie doprowadzić może do szybszego znalezienia rozwiązania. Może też się okazać, że zadziała to wręcz odwrotnie, i mrówki wpadną w ślepy zaułek na początku (wszystko zależy od wylosowanych wartości). Z drugiej strony – po znalezieniu pierwszego, dowolnego rozwiązania, feromon zostawiony przez mrówki na najlepszej ścieżce wzmocni ją, a cała reszta ścieżek wygaśnie zgodnie z parametrem wyparowywania, a więc bardzo szybko początkowe wartości feromonów przestaną mieć jakiekolwiek znaczenie.

Niestety, ten zestaw testów nie został jeszcze przeprowadzony.

# Wizualizacja

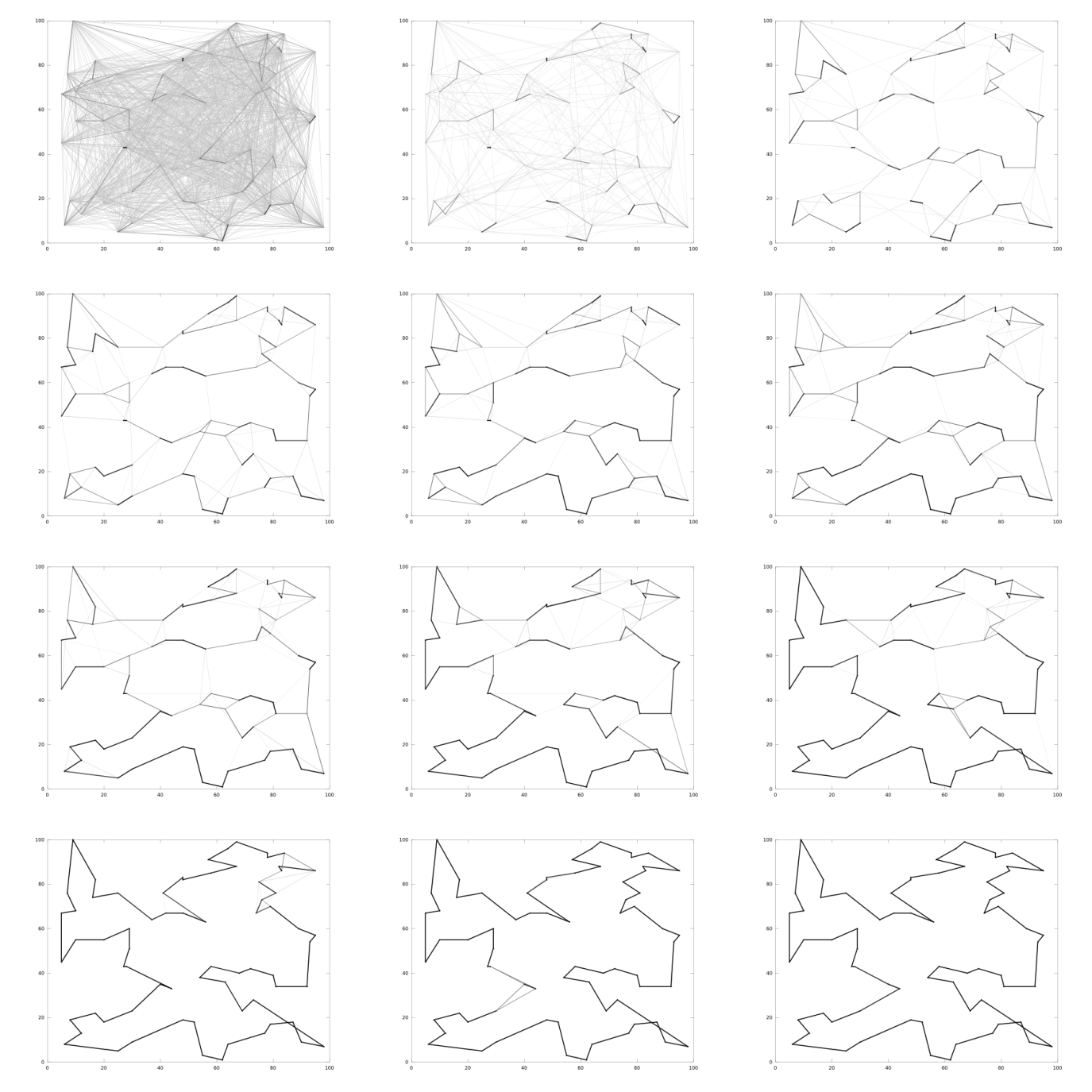
Podczas działania algorytmu możliwe jest generowanie obrazu reprezentującego graf z krawędziami pokolorowanymi zgodnie z bieżącą siła feromonu. Poniżej przedstawione jest kilka kroków z działającego algorytmu. Im linia grubsza i ciemniejsza, tym większa siła feromonu. Kolory są skalowane względem najmocniejszej krawędzi, a wszystkie śmierdzące słabiej niż 10% najsilniejszej są pomijane przy rysowaniu (dlatego niektóre z wierzchołków wyglądają na niepołączone). Na pokazanej serii zrzutów widać pośrednie kroki podczas rozwiązywania problemu st70 (siedemdziesiąt wierzchołków w grafie), przetwarzanie zakończyło się po 530 rundach z najkrótszą ścieżką o długości 689 (przy optymalnej ścieżce w tym zbiorze długości 675).

Dodatkowo można także obejrzeć dwie animacje pokazujące „na żywo” działanie algorytmu dla dwóch zbiorów testowych (umieszczone w serwisie YouTube):

1. <http://www.youtube.com/watch?v=lHv135uKeHo> – zestaw danych berlin52
2. <http://www.youtube.com/watch?v=aPNb97EBSuE> – zestaw danych ulysses16

Zostały one stworzone poprzez zrzucanie wyniku działania każdej rundy do pliku.

Dzięki wizualizacjom można zauważyć pewne cechy stworzonego algorytmu. Jedną z najbardziej charakterystycznych jest bardzo szybkie ustalanie dużych fragmentów trasy przez mrówki (głównie na „obrzeżach” grafu), a długie i żmudne prace nad dopracowywaniem szczegółów w problematycznych miejscach. Stąd można wyciągnąć wniosek, że warto pomyśleć nad dodatkowym warunkiem stopu, który kończyłby działanie algorytmu po ustabilizowaniu się wahań pomiędzy kolejnymi rozwiązaniami na pewnym poziomie, np. +/- 10 jednostek, dzięki czemu bez szczególnej utraty jakości rozwiązania można bardzo przyspieszyć jego odnajdowanie.



Rysunek . Wizualizacja siły feromonu na ścieżkach. Zrzut po rundzie (w kolejności od góry, rzędami) 20, 40, 100, 150, 200, 250, 300, 350, 400, 450, 500, 530.

# Technologia wykonania

Projekt został stworzony przy użyciu środowiska Matlab. Wybór podyktowany był głównie chęcią uczestników zespołu do pogłębienia umiejętności posługiwania się tym językiem, a w mniejszym stopniu dobrym dostosowaniem tego środowiska do specyfiki zadania. Czasami okazywało się jednak, że niektóre metody trzeba zaimplementować od zera, w szczególności te dotyczące wizualizacji otrzymywanych wyników.

## opis najważniejszych struktur danych

Podczas działania algorytmu wszystkie dane trzymane są w różnego rodzaju macierzach. Podczas szukania ścieżki, każda mrówka reprezentowana jest przez wektor zawierający dotychczas odwiedzone przez nią miasta, dodatkowo zapamiętywany jest drugi wektor zawierający miasta jeszcze nieodwiedzone. Dodatkowo przechowywana jest globalna dla wszystkich mrówek macierz siły feromonu, gdzie wartość oznacza siłę feromonu na drodze z miasta *i* do miasta *j*.

Poza tym w celach testowych pamiętane są długości najkrótszej i najdłuższej ścieżki w każdej epoce.

## Opis najważniejszych funkcji

Cały projekt podzielony jest na kilka funkcji, ich krótki opis przedstawiony jest poniżej:

**Plik: main.m**

[ret\_distance, ret\_route, best\_round, shortest\_tab] = **main**(filename, type=*'euc'*, total\_ants = *50*, stink\_fade = *0.95*, stink\_power = *0.1*, total\_rounds = *1000*, initial\_stink = *1*, headless = *0*)

Główna funkcja w projekcie, wykonuje pełny algorytm dla podanego zbioru testowego oraz parametrów.

**Argumenty:**

1. filename – nazwa pliku zawierającego zbiór testowy
2. type – sposób wyliczania odległości w danym zbiorze, możliwe wartości:
   1. *euc* – standardowa metryka euklidesowa
   2. *geo* – współrzędne wierzchołków traktowane są jako geograficzne, a odległość wyliczana jest jako odległość po powierzchni kuli
3. total\_ants – liczba mrówek poruszająca się po świecie
4. stink\_fade – parametr określający szybkość wyparowywania feromonu
5. stink\_power – parametr określający siłę feromonu dodawanego do najlepszych ścieżek
6. total\_rounds – maksymalna liczba epok po której przetwarzanie zostanie przerwane
7. initial\_stink – sposób generowania początkowych wartości feromonu:
   1. jeśli mniejszy od zera, to wartości te są losowe
   2. w przeciwnym wypadku wartości te są ustawiane na podany parametr
8. showevery – jeśli mniejsze od zera, to podczas działania nie będą generowane żadne wykresy ani zapisywane żadne obrazy (przydatne w przypadku przeprowadzania długiej serii testów na maszynach zdalnych), jeśli większe od zera, to obrazy generowane są co określoną liczbę rund.

**Zwracane wartości:**

1. ret\_distance – długość najlepszej znalezionej ścieżki
2. ret\_route – lista kolejnych miast w najkrótszej ścieżce
3. best\_round – numer epoki, w której znaleziono najlepsze rozwiązanie (w szczególności może być mniejszy niż liczba wykonanych rund)
4. shortest\_tab – wektor zawierający długości najkrótszych ścieżek w kolejnych epokach.

**Plik: load\_cities.m**

[distances, stink, nr\_cities, cities] = **load\_cities**(filename, type=*'euc'*, initial\_stink = *1*)

Funkcja ładuje z podanego pliku listę współrzędnych miast i dokonuje obliczenia początkowych wartości feromonu a także długości poszczególnych połączeń. Argumenty mają takie samo znaczenie, jak w przypadku funkcji głównej.

**Plik: plot\_stink.m**

**plot\_stink**(stink, coords)

Funkcja jest odpowiedzialna za wizualizację bieżącego stanu systemu, a więc za rysowanie grafu połączeń i kolorowanie ścieżek w zależności od siły feromonu na nich.

**Argumenty:**

1. stink – macierz zawierająca aktualny poziom feromonu na ścieżkach
2. coords – współrzędne miast

## Format pliku wejściowego

Do działania niezbędne jest podanie pliku z definicją współrzędnych kolejnych miast. Format pliku jest bardzo prosty, w każdej linii należy podać trzy wartości liczbowe oddzielone od siebie spacją, w kolejności:

1. nr miasta,
2. współrzędna x,
3. współrzędna y.

Ilość linii w pliku określa wielkość całego problemu, a więc ilość wierzchołków w grafie. Odległości pomiędzy miastami wyliczane są zgodnie z podanym wymaganiem (euklidesowa bądź geograficzna). Przykładowy plik wejściowy wygląda następująco (zestaw danych ulysses16):

1 38.24 20.42

2 39.57 26.15

3 40.56 25.32

4 36.26 23.12

5 33.48 10.54

6 37.56 12.19

7 38.42 13.11

8 37.52 20.44

9 41.23 9.10

10 41.17 13.05

11 36.08 -5.21

12 38.47 15.13

13 38.15 15.35

14 37.51 15.17

15 35.49 14.32

16 39.36 19.56

# bibliografia

1. M. Dorigo i L. M. Gambardella, „Ant colonies for the traveling salesman problem,” *BioSystems,* tom 43, nr 2, s. 73-82, 1997.
2. D. Merkle i M. Middendorf, „Swarm Intelligence and Signal Processing,” *IEEE Signal Processing Magazine,* s. 152-158, 11 2008.

1. http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/ [↑](#footnote-ref-1)