Sprawozdanie OK

Ant Colony Optymalization (ACO)

Mikołaj Smolibowski,
nr. 145173
Bioinformatyka III rok,

Spis treści

Spis Figur

1. Opis problemu:

Dany jest spójny graf G=(V, E) z wagami wi przypisanymi do każdej krawędzi vi V. Należy znaleźć ścieżkę łączącą wszystkie wierzchołki (każdy odwiedzony minimum raz) taką, aby zminimalizować jej koszt S.
Koszt wyliczany jest dynamicznie w trakcie konstrukcji ścieżki w taki sposób, że:
a) stanowi sumę wszystkich wag odwiedzonych do tej pory krawędzi,
b) co x odwiedzonych łuków licząc od startu do aktualnej sumy S dodawana jest suma ostatnich x /2 odwiedzonych wag pomnożona przez podwójny stopień wierzchołka, w którym znajduje się algorytm przeszukiwania po przejściu x krawędzi,

Założenia dla instancji problemu: można przyjąć początkowo:
- |V| minimum: 100,
- deg(v) = [1, 6],
- wi = [1, 100],
- x = minimum 5,

2. Implementacja:

a. Python 3.9.0
b. Algorytm mrówkowy (*ang. Ant Colony Optymalization algorithm; ACO),*

3. Opis algorytmu:

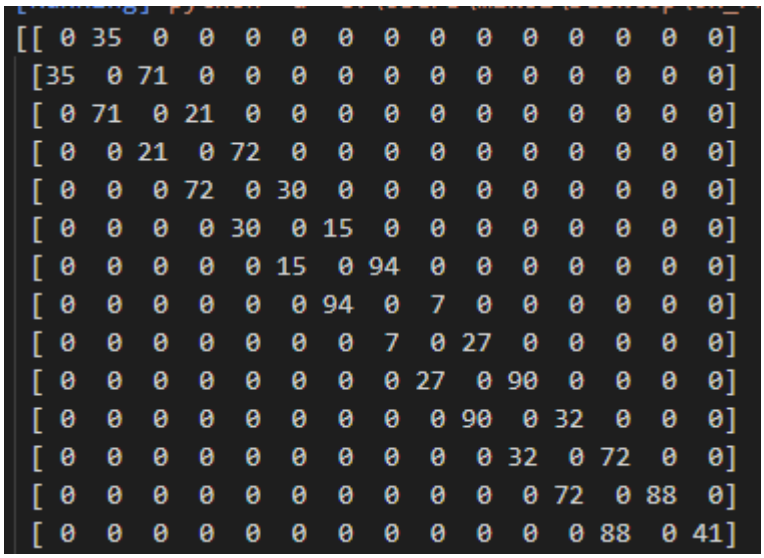
Do implementacji kodu wykorzystano:
- Materiały ze strony: http://www.cs.put.poznan.pl/mradom/teaching/optymalizacja_kombinatoryczna_bio.html#problemy
- Informacje o rozwiązaniu problemu ze strony: https://stackoverflow.com/
  - Np. https://stackoverflow.com/questions/9764298/how-to-sort-two-lists-which-reference-each-other-in-the-exact-same-way

1. Domyślne parametry:

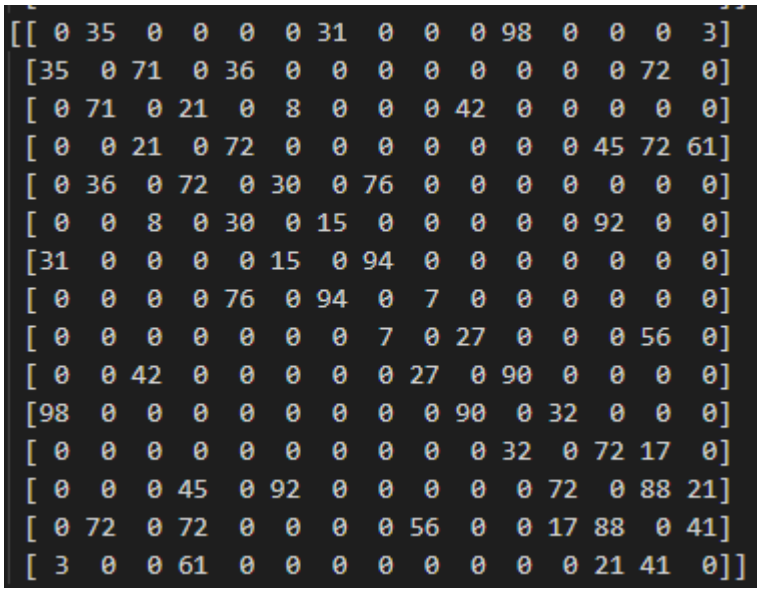
- Iiczba mrówek = 100,
- Czas trwania = 60 sec,
- Ilość wierzchołków = 100,
- Minimalna waga wierzchołka = 1,
- Maxymalna waga wierzchołka = 100,
- Przyrost feromonów = 1,
- Wartość wygładzania feromonów = 20,
- Wartość parowania feromonów = 0.3,

2. Generator instancji:

Funkcja *create()* generuje tablicę dwuwymiarową wielkości równej ilości wierzchołków. Pola tak utworzonej macierzy wypełniane są wartościami 0. Obie przekątne macierzy są następnie losowane i wypełniane losowymi wagami .



Graf zwrócony przez funkcję *create()* jest przekazywany do funkcji *check_and_fill(check_graph)*. Połączenia między wierzchołkami są poprzez losowanie losowych wartości z określonego przedziału 1-100 (domyślnie). Ilość Tworzonych połączeń wynosiła 6 (domyślnie). Wylosowane wartości umieszczane są w określonym wierszu. Wypełniane wierszy przebiega iteracyjnie.



3. Generator rozwiązań losowych:

Funkcja *scoutung_ants*() generuje losowe listę składającą się z kolejno losowo odwiedzonych po sobie wierzchołków będących swoimi sąsiadami w grafie. W momencie wywołania funkcji losowany jest wierzchołek, z którego

mrówka rozpocznie wędrówkę. Wierzchołek startowy przypisywany jest do zmiennej '*add_to scouting_path*'a następnie dodawany jest do rozwiązania. Następnie tworzona jest lista nieodwiedzonych wierzchołków, z której usuwany jest wierzchołek startowy.

Następnie wykonywana jest pętla while, trwająca dopóty dopóki znajdują się elementy w liście wierzchołków nieodwiedzonych. W każdym nowym przelocie pętli wykonywane są następujące operacje:

- stworzenie listy sąsiadów wierzchołka, w którym aktualnie znajduje się mrówka ('*neghbour_vertex'),*

- stworzenie listy wierzchołków nieodwiedzonych przez mrówkę będących sąsiadami wierzchołka, w którym aktualnie znajduje się mrówka ('*missed_neighbours'*),

- wybór: „Jeśli są wierzchołki nieodwiedzone będące sąsiadami wierzchołka, w którym aktualnie znajduje się mrówka; Wykonaj losowanie wierzchołka, do którego przejść z listy '*missed_neighbours'",*

- wybór: „Jeśli nie ma wierzchołków nieodwiedzonych w liście '*missed_neighbours'* wykonaj losowanie z listy '*neghbour_vertex',*

Wierzchołek, do którego przejdzie mrówka losowany jest w momencie spełnienia jednego z powyższych warunków. Wylosowany wierzchołek dodawany jest do rozwiązania, oraz usuwany jest z listy wierzchołków nieodwiedzonych. W następnej iteracji pętli operacje stworzenia list '*neghbour_vertex'* oraz '*missed_neighbours'.*

Pętla kończy się wykonywać w momencie wykorzystania wszystkich wierzchołków do stworzenia ścieżki. Funkcja zwraca utworzoną ścieżkę.

4. Zliczanie kosztu ścieżki:

Funkcja *'count_cost(path)'* jako argument przyjmuje listę zawierającą numery wierzchołków tworzących ścieżkę. Koszt przejścia tej ścieżki obliczany jest w następujący sposów.

- generacja tablicy bez wartości równej długości ścieżki,

- wpisanie do tablicy wartości jaka została przypisana połączeniu między jednym wierzchołkiem a drugim,

- co x = 5 (domyślnie) wierzchołków wykonywana jest operacja: dodania sumy ostatnich x /2 odwiedzonych wag pomnożona przez podwójny stopień wierzchołka, w którym znajduje się algorytm przeszukiwania po przejściu x krawędzi,
- stopień wierzchołka wyliczany poprzez zliczenie ilości sąsiadów wierzchołka dla którego wykonywana jest operacja,

Funkcja zwraca sumę wartości znajdujących się w tabeli zawierającej wartości połączeń między wierzchołkami,

5. Metaheurystyka:

Przed rozpoczęciem wyszukiwania coraz lepszych rozwiązań program wykonuje sortowanie wyników wygenerowanych przez generator rozwiązań losowych. Następnie wybierane jest 20% najlepszych rozwiązań. Wyselekcjonowane 20% najlepszych ścieżek przekazywanych jest następnie do funkcji zaznaczającej feromony (‘*mark_pheromone(path, power)’)* na macierzy feromonów (‘*pheromone_matrix’*) będącej tablicą dwuwymiarową, w której komórki wypełnione są wartościami 0 natomiast w miejscach połączeń wierzchołków występuje wartość 1.

Funkcja dodaje wartość wzrostu feromonu (‘*pheromone_increase’;* domyślenie 1) do reprezentacji połączenia między wierzchołkami w macierzy feromonów. Tak stworzona macierz feromonów jest następnie wykorzystywana przy wyszukiwaniu coraz lepszych rozwiązań.

Do funkcji *‘meta_ants(chance)’* wyszukującej coraz lepszych rozwiązań przekazywana jest zmienna ‘*chance’* będąca wartością określającą szansę mrówki na wykorzystanie macierzy feromonowej. Wartość argumentu ‘*chance’* rośnie wraz z ilością mrówek, które zostały wykorzystane do znalezienia jak najlepszego rozwiązania. Początkowo szansa wynosi 5%, po przejściu 15% wszystkich mrówek szansa wykorzystania feromonów rośnie do 15%, następnie kolejno po przejściu 30% mrówek szansa rośnie do 35%, po przejściu 50% mrówek szansa rośnie do 45% a po przejściu 70% mrówek szansa wykorzystania macierzy feromonowi rośnie do 55% i dalej już nie wzrasta.

Po wywołaniu funkcji *‘meta_ants(chance)’* zachodzą następujące operacje.

- Tworzona jest lista wierzchołków (‘*following_path’*), do której dodawane będą wierzchołki odwiedzone przez mrówkę.

- Losowany jest wierzchołek startowy mrówki. Wierzchołek startowy dodawany jest do rozwiązania,

- Tworzona jest lista wierzchołków nieodwiedzonych (lista wszystkich wierzchołków w grafie) (‘*following_not_visited’*)

- Wierzchołek początkowy usuwany jest z listy wierzchołków nieodwiedzonych,

Następni w pętli while wykonywane są operacje wyboru następnego wierzchołka oraz podejmowane są decyzje o wykorzystaniu macierzy feromonowej. Pętla while wykonuje się dopóki dopóty istnieją wierzchołki nieodwiedzone przez mrówkę.

Podobnie jak w przypadku generatora rozwiązań losowych w każdym przebiegu pętli wykonywane są następujące operacje:

- utworzenie listy sąsiadów wierzchołka, w którym aktualnie się znajduje mrówka ‘*following_neighbour_vertex’*

- utworzenie listy wierzchołków, które nie zostały wykorzystane w generowaniu ścieżki a są sąsiadem wierzchołka, w którym aktualni znajduje się mrówka *'following_missed_neighbours'*,

- wylosowanie wartości od 1-100 określającej czy mrówka ma wykorzystać macierz feromonów (w przypadku wylosowania wartości mniejszej niż wartość zmiennej *'chance'* mrówka wykorzystuje macierz feromonów, jeśli wylosowana wartość jest większa wtedy mrówka nie korzysta z macierzy feromonów a następny wierzchołek wybierany jest losowo)

W przypadku istnienia nieodwiedzonego wierzchołka będącego sąsiadem wierzchołka, w którym aktualnie się znajduje mrówka losowania oraz operacje na macierzy feromonów dotyczyć będą jedynie wierzchołków nieodwiedzonych.

W przypadku gdy wszyscy sąsiedzi wierzchołka, w którym znajduje się aktualnie mrówka zostali odwiedzeni. Wtedy losowania następnego wierzchołka oraz operacje na macierzy feromonów wykonywane są dla zbioru wszystkich wierzchołków będących sąsiadami wierzchołka, w którym znajduje się aktualnie mrówka

Rozpatrywany przypadek:

- Istnieją nieodwiedzeni sąsiedzi + wykorzystaj macierz feromonów,

W tym przypadku wykopnywane są następujące operacje:

- Stworzenie tablicy wypełnionej wartościami feromonów wierzchołków nieodwiedzonych,
- Wartość wszystkich feromonów jest sumowana a następnie wyliczany jest procentowy udział poszczególnych wartości w sumie wartości feromonów (*'vertex_prob'*),
- Wartości te stanowią wagę prawdopodobieństwa wylosowania danego wierzchołka jako wierzchołka, do którego przejdzie mrówka
- Losowanie wierzchołka wykonywane jest przez wykorzystanie funkcji *'random.choices()'*
- Wylosowany wierzchołek zostaje dodany do rozwiązania,
- Wylosowany wierzchołek usuwany jest z listy wierzchołków nieodwiedzonych,
- Następuje przejście mrówki do wylosowanego wierzchołka
- Ponowna iteracja pętli (jeśli istnieją nieodwiedzone wierzchołki)

- **Istnieją nieodwiedzeni sąsiedzi + nie wykorzystuj macierzy feromonów,**

  W tym przypadku mrówka nie wykorzystuje macierzy feromonowej. Wybór następnego wierzchołka, do którego przejdzie mrówka polega na wylosowaniu wierzchołka z listy wierzchołków nieodwiedzonych będących sąsiadami wierzchołka, w którym znajduje się aktualnie mrówka (*'following_missed_neghbours'*). Do losowania

wykorzystywana jest funkcja *'random.choice()'* . Jak w poprzednim przypadku wylosowany wierzchołek dodawany jest do rozwiązania, jest usuwany z listy nieodwiedzonych wierzchołków i następuje przejście mrówki do wylosowanego wierzchołka.

W przypadkach gdy nie ma nieodwiedzonych sąsiadów operacje przeprowadzane są w analogiczny sposób jak dla powyżej opisanych przypadków, z tą różnicą, że do zliczania wartości feromonów pobierane są wartości wszystkich sąsiadów wierzchołka, w którym znajduje się mrówka oraz losowania przeprowadzane są dla listy wszystkich sąsiadów wierzchołka.

W momencie gdy wszystkie wierzchołki zostały wykorzystane do utworzenia ścieżki funkcja zwraca listę z zapisanymi wierzchołkami do zmiennej.

Przebieg programu:

Na początku pracy programu generowany jest graf o wielkości n wierzchołków (domyślenie 100). Wierzchołki w grafie połączone są wierzchołkami o wagach między 1-100 (domyślnie). Dla zadanej liczby mrówek (domyślnie 100) wywoływana jest funkcja *'scouting_ants()'*. Wyznaczone losowo ścieżki są następnie przekazywane do funkcji *'count_cost()'* wyliczającej koszt przebytej ścieżki. Wyniki są następnie sortowane. Wybieranych jest 20% najlepszych wyników i na ich podstawie nanoszone są wartości feromonów w macierzy feromonów będącej reprezentacją grafu. Wartości są tym wyższe im częściej jakiś wierzchołek pojawiał się w wygenerowanej ścieżce. Po zakończeniu nanoszenia feromonów program rozpoczyna proces poprawy rozwiązań.

Przez określony czas (60sec domyślnie) program będzie wykonywał następujące operacje:

- w pętli for dla zadanej ilości mrówek (domyślnie 100) wywoływana będzie funkcja poprawiająca rozwiązanie: *'meta_ants(chance)',* Zwracane rozwiązania wpisywane są do listy *'meta_paths'* mającej przechowywać wygenerowane rozwiązania. W momencie zwrócenia rozwiązania dla danej mrówki, ścieżka jest przekazywana do funkcji *'count_cost(path)'*. Wyliczony koszt ścieżki zapisywany jest do kolejnel listy. Po zakończeniu pętli for wartości w listach są sortowane od najmniejszego kosztu do największego kosztu ścieżki (wartości w liście przechowującej wygenerowane ścieżki są sortowane razem z kosztem ich przejścia).

Z posortowanych ścieżek wybierana jest 20% najlepszych ścieżek (o najmniejszym koszcie) i to one zostają następnie przekazana do funkcji *'mark_pheromone(path, power)'*. Funkcja nanosi wartości wzmacniające siłę feromonów na połączeniach między wierzchołkami, które znalazły się w najlepszym rozwiązaniu. Jeśli koszt przejścia jest lepszy niż koszt wcześniej wygenerowanego rozwiązania to wartość ta dodawana jest do listy najlepszych rozwiązań (pomaga zobrazować faktyczne polepszanie wyników przez program w czasie)

Po wzmocnieniu feromonów następuje ich „parowanie”. Parowanie polega na zmniejszeniu wartości feromonów w macierzy feromonów poprzez odjęcie od każdego z nich 30% (domyślnie) wartości feromonu w danym polu w macierzy feromonów.

Po przeprowadzeniu operacji parowania feromonów wywoływana jest funkcja *‘smoothie()’* mająca na celu wygładzić wartości feromonów w przypadku, gdyby wartość jednego z sąsiadów w liście sąsiadów dla wierzchołka, w którym znajduje się mrówka była zbyt dominująca i powodowała znaczny spadek prawdopodobieństwa wyboru innych wierzchołków.

Wygładzenie przebiega w pętli for, pobierane są wartości feromonów wierzchołków sąsiadujących z aktualnie sprawdzanym wierzchołkiem. Wartości są sumowane i liczone jest 10% tej wartości. Jeśli wartość, któregoś z feromonów jest mniejsza niż wyliczone 10% sumy wartości feromonów wtedy wykonywana jest operacja wygładzania. Operacja ta polega na aktualizacji wartości feromonów wszystkich wierzchołków sąsiadujących z aktualnie sprawdzanym wierzchołkiem. Nowa wartość feromonów wyliczana jest z poniższego wzoru:

$$pheromone\_matrix[i][j] = lovest * (1 + \log_{20}(pheromone\_matrix[i][j] / lovest)$$

- *pheromone_matrix[i][j] – aktualnie aktualizowana wartość wierzchołka,*
- *lovest – najniższa wartość feromonów wśród sąsiadów sprawdzanego wierzchołka,*

Po zakończeniu wygładzania wartości w macierzy feromonów są zaokrąglane do dwóch miejsc po przecinku.

Jeślinie osiągnięto limitu czasu wykonywania następuje ponowienie wszystkich operacji.

4. Wyniki Testów

Wszystkie przeprowadzane testy były wykonywane dla parametrów podstawowych. Zmianie ulegał zawsze jeden parametr.

1. Badanie zmienności średniego kosztu znalezionej ścieżki w zależności od ilości wierzchołków

Figura 1. Średni koszt ścieżki w zależności od ilości wierzchołków.

W trakcie prowadzonych testów na początku ilość wierzchołków wynosiła 30. Wraz z usprawnianiem działania algorytmu stopniowo zwiększano tą ilość. Finalnie do wszystkich pozostałych testów wartością domyślną była liczba 100 wierzchołków dla każdego generowanego grafu. Na wykresie powyżej możemy zauważyć stały wzrost koszu ścieżki w zależności od ilości wierzchołków co jest naturalnie zrozumiałe ponieważ związane jest to z większa ilością wierzchołków do odwiedzenia a tym samym większą ilością ścieżek do obilczenia.

2. Badanie zmienności średniego kosztu znalezionej ścieżki w zależności od długości działania algorytmu

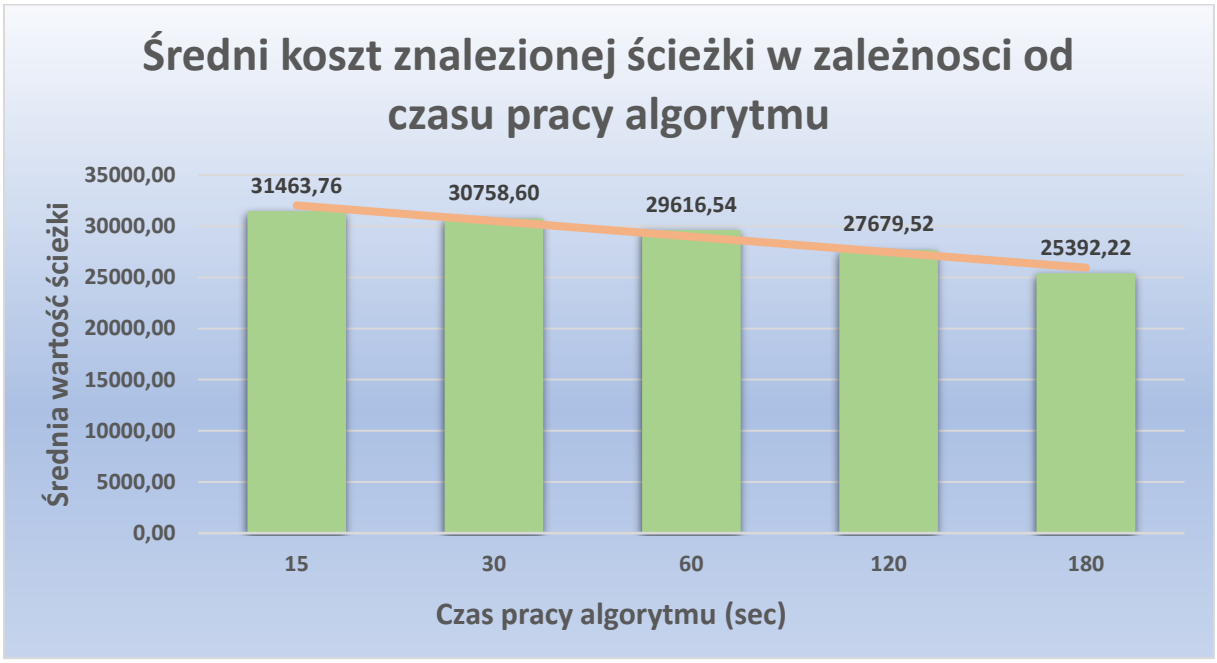


Figura 2. Średni koszt znalezionej ścieżki w zależności od czasu pracy algorytmu.

W prowadzonych testach wykorzystano przedział czasowy od 15 sekund do 180.Jak można zauważyć na Figurze 2 koszt obliczonej ścieżki malał wraz z długością Trwania algorytmu. Najpewniej jest to związane z możliwością wykonania większej ilości operacji na badanych 100 wierzchołkach i użyciu 100 mrówek w każdym poszukiwaniu najlepszej ścieżki. Z całą stanowczością można stwierdzić, ze długość czasu trwania algorytmu jest bardzo istotna w przypadku poszukiwania jak najlepszego rozwiązani o czym świadczy malejący koszt znalezionych ścieżek wraz ze wzrostem długości czasu pracy programu.

3. Badanie zmienności średniego kosztu znalezionej ścieżki w zależności od ilości wykorzystywanych mrówek

Figura 3. Średni koszt znalezionej ścieżki w zależności od ilości wykorzystywanych mrówek.

Na początku testów ilość mrówek w wykonywanych testach wynosiła 50 a następnie była stopniowo zwiększana. Na powyższym wykresie zauważyć można, że wraz ze wzrostem ilości mrówek wykorzystanych w trakcie jednej pętli for malał również średni koszt znalezionej ścieżki. Różnice średniego kosztu ścieżki pomiędzy wartościami 50-100 czy 200-300 nie są szczególnie wysokie, lecz porównując wynik średniego kosztu ścieżki dla 100 mrówek ze średnim kosztem ścieżki znalezionym gdy mrówek było 300 zaobserwujemy znaczącą różnice. Stąd może przychodzić wniosek, że aby wygenerować jak najlepsze koszty przejścia ścieżki należałoby wykorzystać więcej mrówek.

4. Badanie zmienności średniego kosztu znalezionej ścieżki w zależności od wag krawędzi w grafie

Figura 4. Średni koszt znalezionej ścieżki w zależności od wag krawędzi.

Przetestowano w jaki sposób zmiana przedziału wartości, z którego losowane są wagi dla połączeń w grafie wpłynie na wzrost kosztów generowanych ścieżek. Zgodnie z założeniem wzrost kosztów połączeń między wierzchołkami spowodował wzrost kosztów generowanych rozwiązań.

5. Badanie zmienności średniego kosztu znalezionej ścieżki w zależności od wartości ‘X’

Figura 5. Średni koszt znalezionej ścieżki w zależnosci od wartości 'X'.

Zmienna ‘X’ jest wykorzystywana przy dodawaniu dodatkowego kosztu do tworzonego kosztu ścieżki. Parametr ten wyznacz po odwiedzeniu ilu wierzchołków mamy wykonać operację dodającą wartości X/2 odwiedzonych wierzchołków pomnożone. Ta wartość dodatkowo jest mnożona przez stopien wierzchołka, w którym znajduje się mrówka. Początkowo zakładałem, że wraz ze wzrostem wartości zmiennej ‘X’ koszt ścieżek będzie rosnąć ze względu na sumowanie kosztów większej ilości wierzchołków. Jednak częstość tych operacji spadała wraz ze wzrostem ‘X’ co najpewniej przyczyniło się do mniejszych średnich wartości wygenerowanych ścieżek.

6. Wnioski

Ważnym spostrzeżeniem dla mnie w trakcie tworzenia tego projektu jest fakt, ze napisanie algorytmu i sprawdzenie, że generuje poprawiające się wyniki nie oznacza jeszcze, że przy większej ilości obliczeń i instancji do przetestowania wyniki te będą spójne. W trakcie wykonywania testów zauważyłem, że wyniki są poprawiane, lecz nie zmieniają się one specjalnie w przypadku zwiększania/zmniejszania ilości mrówek czy czasu trwania algorytmu. Pozwoliło mi to na zauważenie, że może nie wszystko w kodzie było zapisane poprawnie i tak też było. Po dokonaniu zmian w obrębie wykorzystania feromonów oraz oznaczania najlepszych ścieżek algorytm zaczął generować wyniki bliższe temu czego spodziewałbym się zobaczyć w przypadku zmian poszczególnych parametrów. Co więcej uświadomiłem sobie, że wykonanie operacji metaheurystycznych najlepiej prowadzić dla dużej liczby powtórzeń dla jednego grafu. Przy niskiej liczbie powtarzanych obliczeń różnice były bardzo losowe. Dopiero po zaimplementowaniu większej liczby powtórzeń dla tych samych zmiennych pozwoliło mi na wygenerowanie wyników, które miały jakiś sens i mogłem je wykorzystać.

Do ważnych wniosków dotyczących algorytmu ważnym zauważenia jest fakt, że spadek kosztu znajdowanej ścieżki maleje wraz z wydłużeniem czasu trwania algorytmu oraz wraz ze wzrostem liczby wykorzystywanych mrówek. Wzrost kosztu znalezionej ścieżki wraz ze wzrostem ilości wierzchołków w grafie oraz wraz ze wzrostem kosztów połączeń między wierzchołkami jest dość wydaje mi się oczywisty. Zaskoczeniem dla mnie natomiast był wpływ wzrostu zmiennej ‘X’ na koszt znajdowanej ścieżki.