Wykorzystanie algorytmu mrówkowego w celu znalezienia optymalnego przejazdu komunikacją miejską dla miasta Gdańsk

Mateusz Pilecki

Adam Sygut

1. Model matematyczny

1.1 Opis słowny problemu

Celem projektu było stworzenie programu mającego za zadanie znalezienie optymalnej drogi z punktu A do punktu B za pomocą komunikacji miejskiej oraz poruszania się pieszo na terenie miasta Gdańsk. Problem który rozważaliśmy jest fragmentem dyscypliny optymalizacji matematycznej. Korzystaliśmy z gotowego rozkładu jazdy autobusów i tramwajów ZTM Gdańsk dostępnego na stronie:
https://ckan.multimediagdansk.pl/dataset/tristar

Dane te zawierają między innymi informacje o aktualnym rozkładzie jazdy oraz dane z nimi związane. Zgodnie z regulaminem, dane nie są chronione prawami autorskimi, a co za tym idzie można z nich swobodnie korzystać. W celu wyznaczenia optymalnego rozwiązania zaimplementowaliśmy algorytm mrówkowy.

1.2 Model matematyczny

Nasza funkcja celu skupia się wokół czasu, jaki jest potrzebny użytkownikowi komunikacji miejskiej na dotarcie z punktu startowego do punktu docelowego. Wobec tego, przyjmuje ona poniższą postać:

$$f(x,y,z) = \sum_{i=1} x(i) + y(i) + z(i)$$

Gdzie:

- $x(i) - czas\ potrzebny\ na\ przejazd\ komunikacją\ miejską\ z\ przystanku\ A\ do\ przystanku\ B$
- $y(i) - czas\ potrzebny\ na\ piesze\ przejście\ z\ przystanku\ na\ przystanek$
- $z(i) - czas\ jaki\ trzeba\ odczekać\ na\ przystanku\ na\ następny\ pojazd\ komunikacji\ miejskiej\ po\ pieszym\ przejściu\ na\ inny\ przystanek$

Oczywiście dla danego i składniki powyższej sumy będą się nawzajem wykluczać, tzn. jeśli dla konkretnego i $x(i)$ będzie mniejsze niż $y(i) + z(i)$ do macierzy kosztów wpisana zostanie wartość $x(i)$, natomiast jeśli dla danego i suma $y(i) + z(i)$ dająca sumaryczny czas przejścia na dany przystanek i odczekania na następny pojazd komunikacji miejskiej będzie mniejsza niż czas dojazdu do niego danym środkiem transportu $x(i)$ – wówczas do macierzy kosztów wpisana zostanie wartość $y(i) + z(i)$.

Stanem w naszym modelu matematycznym jest aktualnie rozważany przez algorytm wierzchołek grafu (przystanek). Decyzją w naszym algorytmie jest wybór następnego wierzchołka grafu (przystanku) zgodnie z prawdopodobieństwem obliczonym przez algorytm.

2. Algorytm

2.1 Schemat algorytmu

Algorytm mrówkowy jest probabilistyczną techniką szukania dróg w grafach. Do znalezienia optymalnej trasy wykorzystuje feromony zostawione na ścieżkach.

Natężenie feromonów jest zatem „wskaźnikiem jakości" danej trasy. Wyznaczamy je z następującej zależności:

$$\Delta\tau_{i,j}^{k} = \begin{cases} \frac{1}{L_K}, & jeśli\ mrówka\ przejdzie\ krawędzią\ i,j \\ 0, & w\ przeciwnym\ przypadku \end{cases}$$

Gdzie:

$\Delta\tau - ilość\ feromonu$
$i,j - krawędź\ z\ wierzchołka\ "i"\ do\ wierzchołka\ "j"$
$K - K - ta\ mrówka$
$L_K - długość\ ścieżki$

Z powyższej zależności widać, że im krótsza ścieżka tym większa wartość feromonu.

Uwzględniając liczbę mrówek otrzymujemy:

$$\Delta\tau_{i,j}^{k} = \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Dodatkowo po uwzględnieniu procesu wyparowania feromonów uzyskujemy wzór:

$$\Delta\tau_{i,j}^{k} = (1-\rho)\Delta\tau_{i,j} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Gdzie:

$\rho - stała\ wartość\ z\ zakresu\ (0, 1)$
$\Delta\tau_{i,j} - aktualny\ poziom\ feromonu$
$\sum_{k=1}^{m} \Delta\tau_{ij}^{k} - nowy\ poziom\ feromonu\ dla\ wszystkich\ mrówek$

Wybór ścieżki zależy od prawdopodobieństwa wyliczonego z poniższego wzoru:

$$P_{i,j} = \frac{(\Delta\tau_{i,j})^{\alpha} \cdot (\eta_{i,j})^{\beta}}{\sum((\Delta\tau_{i,j})^{\alpha} \cdot (\eta_{i,j})^{\beta})}$$

Gdzie:

$\eta_{i,j} = \frac{1}{L_{i,j}} -$ „jakość” $krawędzi\ i, j\ w\ grafie$

$\alpha, \beta\ \in constants$

2.2 Pseudokod

Dla każdej iteracji (jednego przejścia mrówki):

Dla każdej mrówki z kolonii:

Dopóki mrówka nie dotarła do pokarmu (przystanek docelowy):

Oblicz prawdopodobieństwo przejścia każdą krawędzią

Losowo wybierz zgodnie z prawdopodobieństwem krawędź

Przejdź daną krawędzią do danego wierzchołka

Przelicz koszt trasy

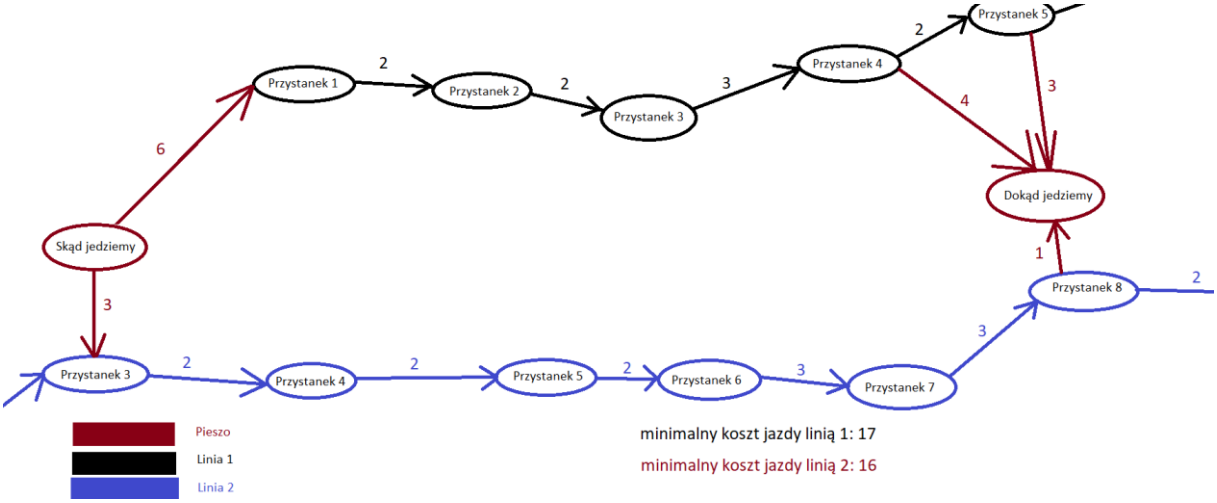
Pomnóż feromony przez „współczynnik parowania”

Dla każdej ścieżki mrówek:

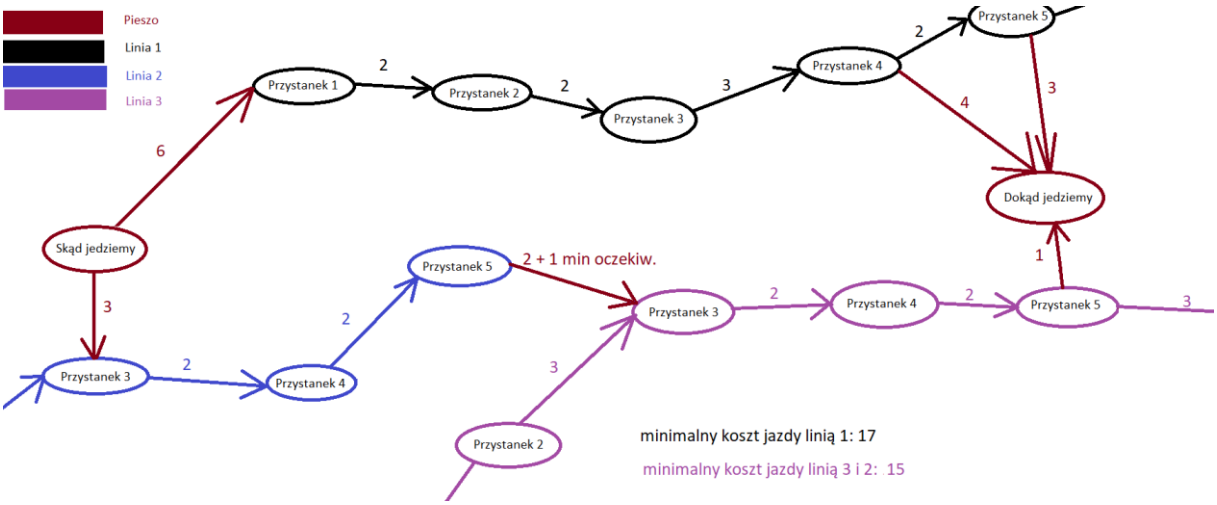
Zaktualizuj feromony zgodnie z kosztem krawędzi, przez które przeszła mrówka

2.3 Graficzne przedstawienie działania programu.

Przykładowe graficzne zobrazowanie działania programu:



Przykładowe graficzne zobrazowanie działania programu dla przykładu z przesiadką:



2.4 Parametry algorytmu

Podstawowymi parametrami algorytmu są: poziom feromonów, współczynnik wyparowania feromonów, współczynniki alfa oraz beta znajdujące się we wzorze na obliczanie prawdopodobieństwa, iteracje, ilość mrówek.

Poziom feromonów – wszystkie przejścia początkowo mają ten sam startowy poziom feromonów. Przejścia, po których przeszły mrówki zwiększają swój poziom feromonów w każdej iteracji, natomiast feromony przejść nieodwiedzonych stopniowo wyparowują.

Współczynnik wyparowania feromonów – określa prędkość wyparowywania feromonów. Im większą ma wartość tym szybciej feromony wyparowują

Współczynnik alfa – jest to współczynnik we wzorze na prawdopodobieństwo wyznaczenia ścieżki ściśle związany z feromonami. Im większa jego wartość, tym większy wpływ ma na wybór następnej ścieżki poziom feromonów danego przejścia

Współczynnik beta – analogicznie, im większa wartość tego współczynnika, tym większy wpływ kosztu przejścia daną ścieżką na wybór ścieżki.

Iteracje – określają ile przejść do pokarmu (przystanku docelowego) mają wykonać mrówki

Ilość mrówek – określa ilość mrówek w kolonii.

Istotne są również parametry, które związane są z algorytmem ze względu na konkretną postać problemu, jakim jest ten projekt. Są to między innymi:

Średnia prędkość poruszania się człowieka – przyjęliśmy wartość 5 km/h.

Zakres czasowy znajdowania pierwszych rozkładów jazdy na podstawie godziny odjazdu podanej przez użytkownika – przyjęliśmy czas +- 10 minut od czasu podanego przez użytkownika

Maksymalny czas dojścia z przystanku na przystanek – przyjęliśmy 15 minut. Czas ten nie ogranicza ewentualnego dojścia do przystanku końcowego.

Maksymalny czas oczekiwania na następny środek transportu po przejściu na inny przystanek – przyjęliśmy 15 minut.

Maksymalny czas dojścia na przystanek docelowy – przyjęliśmy 30 minut.

Maksymalny czas dojścia na przystanek docelowy jest brany pod uwagę jedynie w zoptymalizowanej wersji algorytmu (o dwóch wersjach algorytmu więcej w rozdziale 3.).

Wszystkie powyżej wymienione parametry, istotne z punktu widzenia konkretnego problemu, który implementujemy wpływają na wielkość macierzy kosztów, która jest tworzona dynamicznie – na podstawie danych wejściowych podanych przez użytkownika. Im większe będą maksymalne czasy przejścia/oczekiwania tym większa macierz kosztów. Wynikają z tego dwie rzeczy – algorytm będzie wykonywał się dłużej, jednak zwiększa się prawdopodobieństwo na znalezienie rozwiązania optymalnego.

3. Aplikacja

3.1 Wymagania odnośnie uruchomienia

Aplikacja została napisana w języku python z użyciem frameworka django. Poza modułami domyślnie dostępnymi w interpreterze pythona do uruchomienia programu potrzebne są również poniżej opisane biblioteki:

- django==3.0.3
- matplotlib==3.3.3
- networkx==2.5
- numpy==1.19.2
- pyproj==2.6.1.pos1
- requests==2.25.1
- scipy==1.6.0

Podane są również wersje bibliotek używane w implementacji.

3.2 Postać rozwiązania

Rozwiązanie ma postać listy kolejno odwiedzonych w optymalnej trasie wierzchołków wraz z kosztem tej trasy (czasem dotarcia z miejsca startowego do miejsca docelowego) oraz wybranym środkiem transportu. Każdy element listy powinien zawierać aktualny, rzeczywisty czas odjazdu i przyjazdu oraz rodzaj transportu wraz z czasem poruszania się danym środkiem transportu. W przypadku przejazdu komunikacją miejską powinien być podany numer linii, którą użytkownik ma się poruszać natomiast w przypadku przejścia pieszego z przystanku na przystanek w rozwiązaniu ma być uwzględniony czas przejścia wyliczony na podstawie uśrednionej prędkości poruszania się człowieka oraz odległości między przystankami, a także czas oczekiwania na następny pojazd komunikacji miejskiej zgodny z jego rozkładem jazdy.

Oczywiście wszystkie wymienione powyżej dane wyjściowe są istotne z punktu widzenia użytkownika. Dla nas jednak istotny jest również poziom feromonów, który analizujemy w każdej iteracji przy pomocy wykresów 3D.

3.3 Postać danych wejściowych

Naszym celem było stworzenie aplikacji, która otrzymuje dane wejściowe od użytkownika pobrane ze strony internetowej. Niestety nie udało nam się zaimplementować tej funkcjonalności na naszej stronie internetowej, między innymi ze względu na takie problemy jak taka sama nazwa wielu przystanków. W związku z tym dane wejściowe mamy zapisane na sztywno w pliku *constants.py*. Mają one następującą postać:

- Data
- Czas odjazdu
- Przystanek (na ten moment ID przystanku w bazie danych)
  A. Startowy
  B. Końcowy

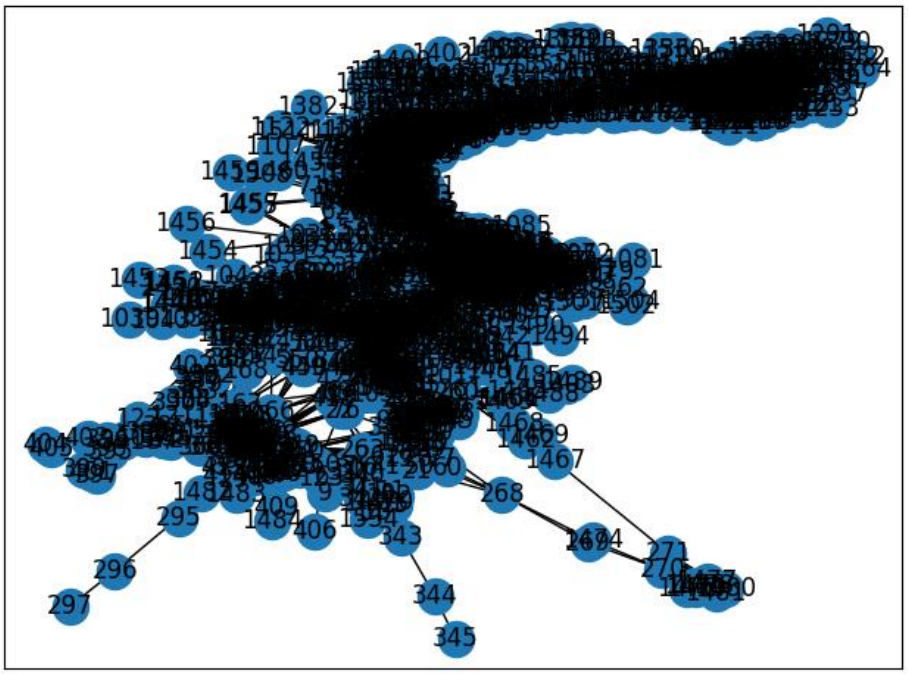
3.4 Funkcjonalność aplikacji.

Funkcjonalność naszej aplikacji opiera się na znalezieniu najlepszego transportu komunikacją miejską z przystanku startowego do przystanku końcowego. Etapy tworzenia rozwiązania końcowego prezentują się następująco:

1. Automatyczne zaktualizowanie bazy danych (do implementacji) średnio raz dziennie.
2. Pobranie od użytkownika danych wejściowych.
3. Stworzenie macierzy kosztów na podstawie danych wejściowych podanych przez użytkownika oraz rozkładów jazdy dostępnych w bazie danych
4. Wyświetlenie grafu obrazującego stworzoną macierz kosztów.
5. Wywołanie algorytmu mrówkowego dla danych wejściowych i stworzonej macierzy kosztów.
6. Wyświetlenie poziomu feromonów na wykresie 3D w dowolnie wybranej iteracji.
7. Wyświetlenie wyników algorytmu.

3.5 Wersje algorytmu.

Mając gotową macierz kosztów musieliśmy się zastanowić, co w przypadku, gdy mrówki dojdą do końca danej gałęzi grafu, jednak nie dotrą do punktu docelowego. Początkowo postanowiliśmy obliczyć czas dojścia z takiego wierzchołka do wierzchołka docelowego i dodać takie połączenie do macierzy kosztów. Jednak przy okazji testowania algorytmu natrafiliśmy na dosyć istotny problem, nie pozwalający algorytmowi na znalezienie rozwiązania optymalnego. Problemem tym była ogromna rozpiętość grafu, co ze względu na probabilistyczny charakter algorytmu nie pozwalało osiągnąć minimum funkcji celu. Przykładowo, dla pesymistycznego grafu macierz kosztów miała rozmiary 1530x1530, a graf przedstawiający ją prezentował się następująco:



W związku z tym zdecydowaliśmy się na wprowadzenie dodatkowego parametru, jakim jest omawiany w rozdziale 2.4 maksymalny czas dojścia na przystanek docelowy równy 30 minut. W tej wersji algorytm w momencie, gdy nie miał już żadnej innej ścieżki do wyboru oraz nie był w punkcie docelowym ponownie obliczał czas przejścia do wierzchołka końcowego. Różnica jednak była taka, że jeśli czas ten przekroczył 30 minut połączenie takie nie było dodawane do macierzy kosztów. Dodatkowo, wszystkie poprzednie połączenia prowadzące do tego wierzchołka, które nie miały żadnych innych rozgałęzień były usuwane z macierzy kosztów. Po tej operacji trasa mrówki zostaje zresetowana i algorytm rusza od nowa. Dzięki tej modyfikacji algorytm jest w stanie znaleźć optymalne rozwiązanie nawet dla bardzo rozgałęzionego grafu.

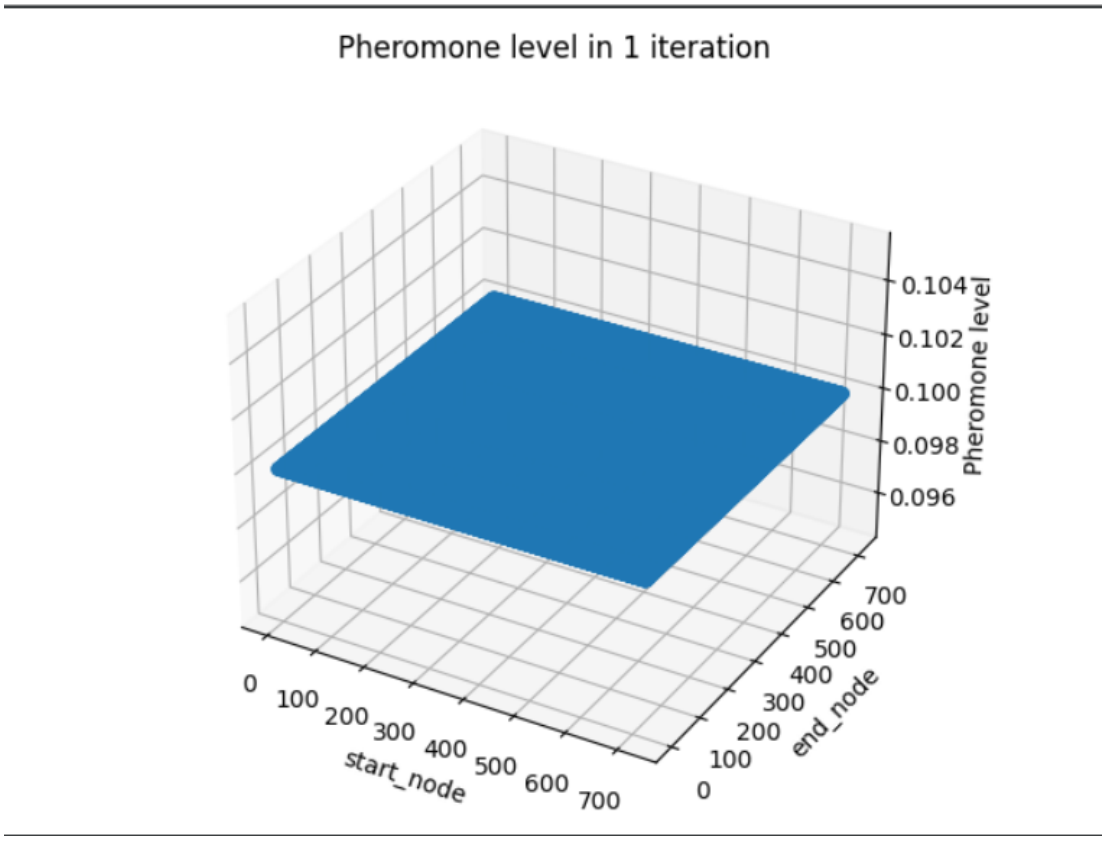
3.6 Sposób tworzenia macierzy kosztów

W związku z tym, że zależało nam na jak najmniejszym czasie wykonywania algorytmu, macierz kosztów jest tworzona dynamicznie, tzn. zaraz po podaniu przez użytkownika danych wejściowych. Zmniejsza to znacząco rozmiar macierzy kosztów, a co za tym idzie, czas wykonywania algorytmu. Za tworzenie macierzy kosztów jest odpowiedzialna funkcja, która przyjmuje jako argument liczbę maksymalnych przesiadek. Funkcja ta w pętli na zmianę dodaje czasy przejazdów komunikacją miejską oraz czasy przejścia „na nogach” do momentu znalezienia przystanka docelowego, bądź do momentu osiągnięcia maksymalnej liczby przesiadek. Dzięki temu macierz kosztów ma ograniczone rozmiary, a algorytm trwa zdecydowanie krócej.

4. Testy

4.1 Rozkład feromonów w zależności od ilości iteracji dla różnych przypadków

Przy każdym wywołaniu algorytmu na początku poziom feromonów jest inicjalizowany taką samą wartością równą 0.1. W związku z tym Wyniki dla pierwszej iteracji są takie same i prezentują się następująco:

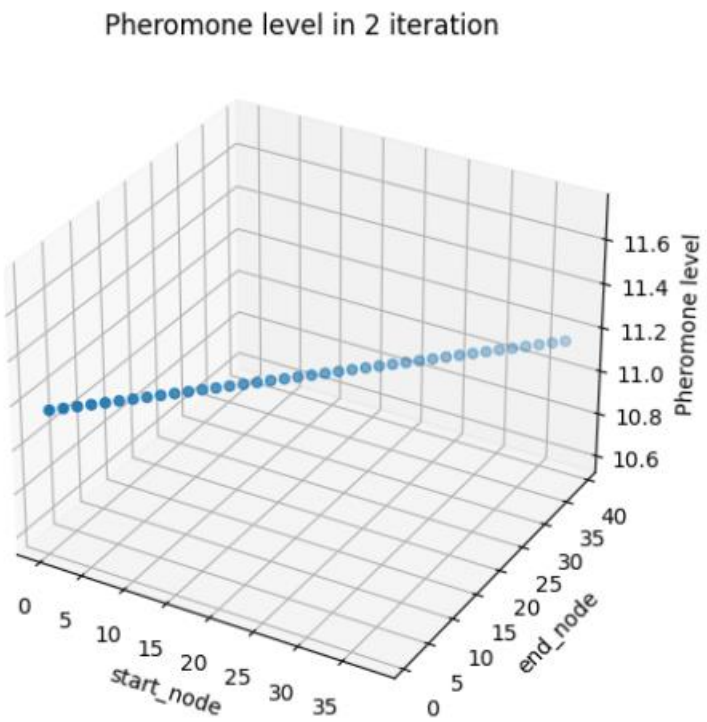


Ponieważ wszystkich feromonów jest bardzo dużo, w każdej następnej iteracji wyświetlamy tylko te feromony, których wartość jest większa bądź równa od 0.1.

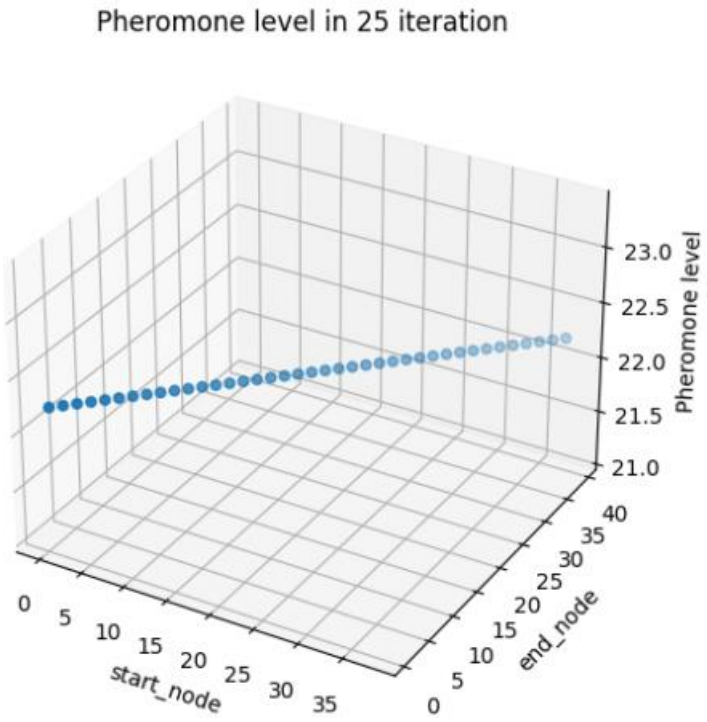
a) Testy pierwszej wersji algorytmu (nieoptymalnej)

1) Przypadek optymistyczny

Jest to taki przypadek kiedy wyszukujemy połączenia z punktu A do punktu B i od razu mamy linię, która prowadzi nas bez przesiadek do punktu docelowego i docieramy do celu jak najszybciej.



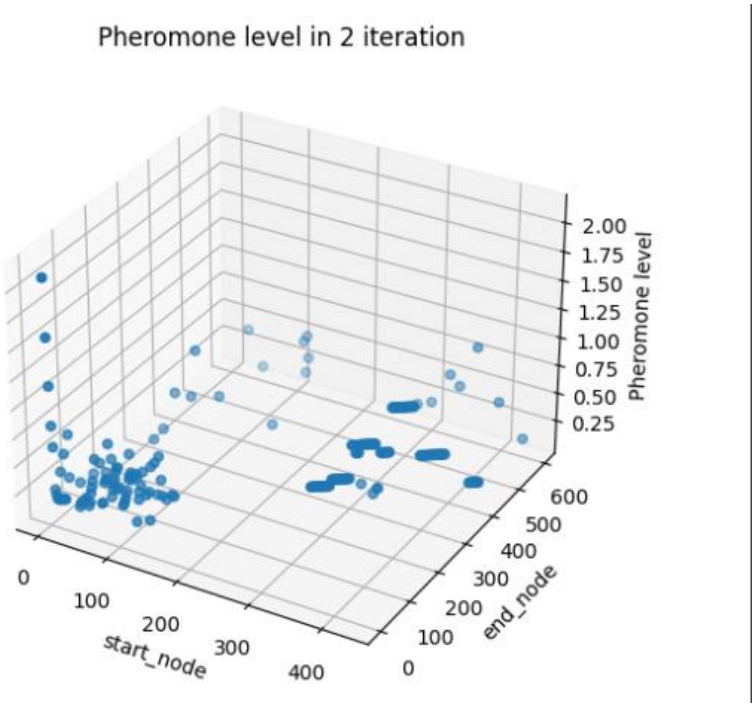
Jak można zauważyć po drugiej iteracji zostały już tylko feromony na jednej ścieżce prowadzącej z punktu A do B, reszta wyparowała.



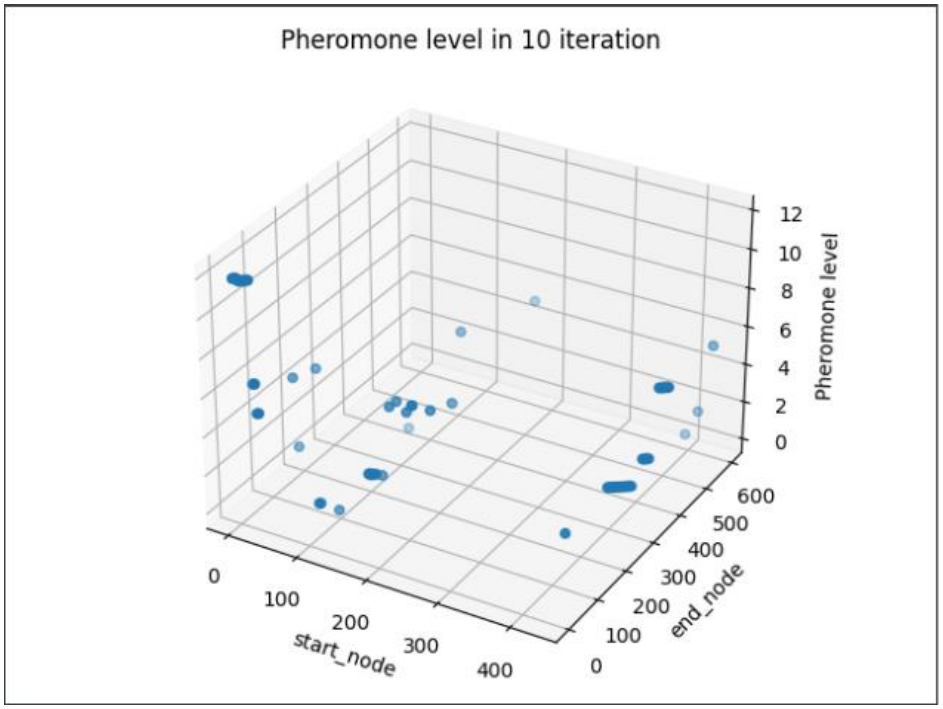
Po 25 iteracjach nie wiele się z zmieniło w stosunku do sytuacji po dwóch iteracjach, jedyną zmianą jest poziom feromonów na znalezionej ścieżce, który rośnie wraz kolejnymi iteracjami.

2) Przypadek pośredni

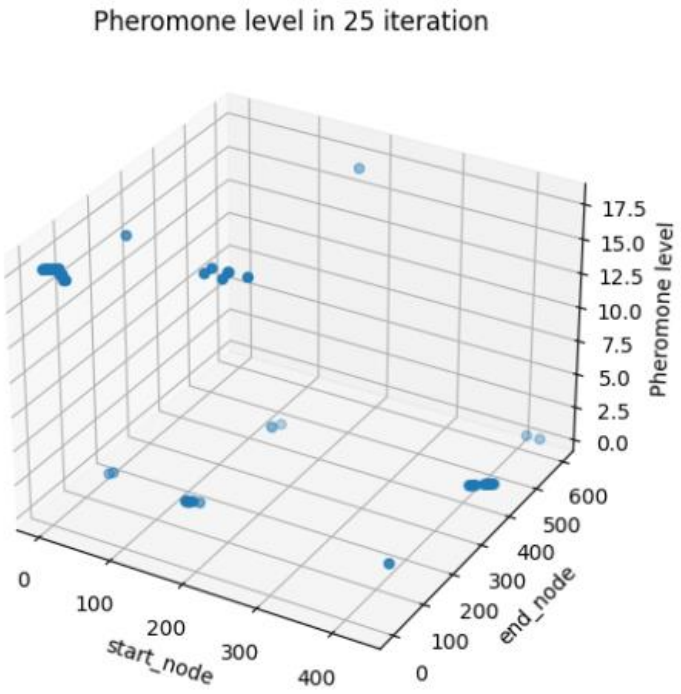
Przypadek pośredni występuje wtedy kiedy nie mamy bezpośredniego połączenia między punktami A i B i musimy skorzystać z 1 przesiadki lub przejść część trasy na piechotę, a punkty A i B leża dość daleko od siebie.



Po 2 iteracjach mamy dużo feromonów które nie wyparowały, ale żaden nie ma dużego poziomu.



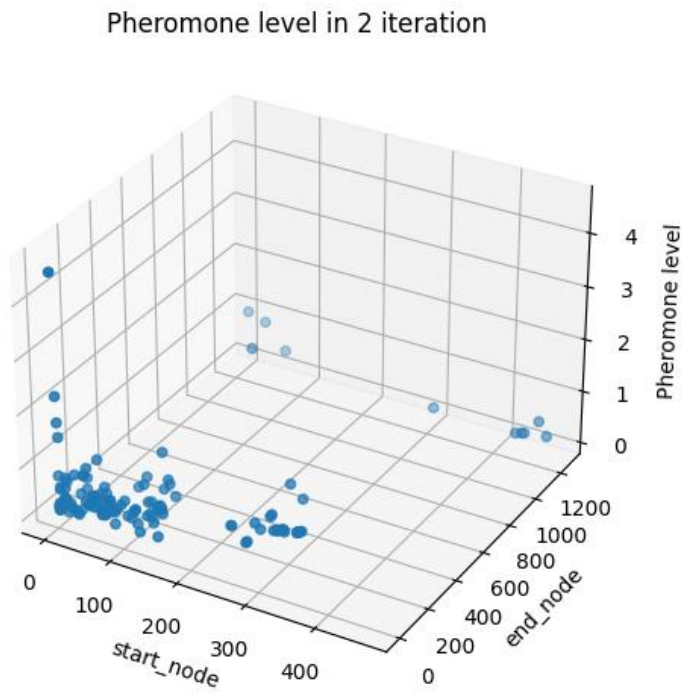
Po 10 iteracjach możemy już zauważyć mniejszą liczbę feromonów oraz wzrost ich poziomu lecz dalej jest sporo feromonów, których poziom jest bliski 0.

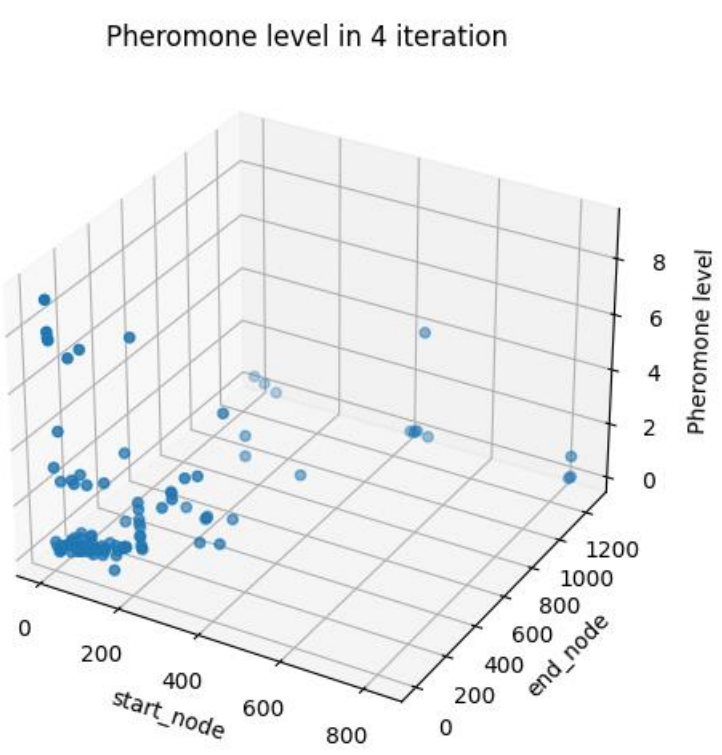


Po 25 iteracjach możemy już coraz lepiej zauważyć powstającą ścieżkę, poziom feromonów niezerowych rośnie.

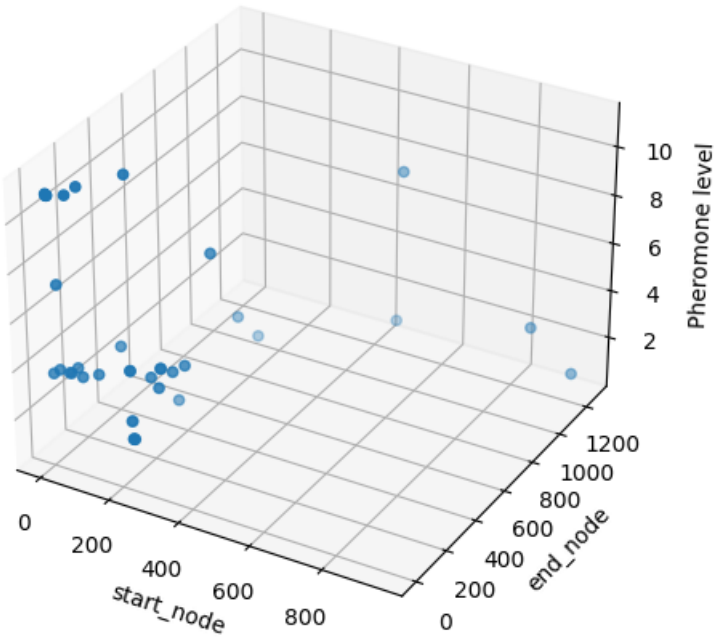
3) Przypadek pesymistyczny

Przypadek pesymistyczny występuje w momencie gdy przystanek początkowy i końcowy znajdują się bardzo daleko od siebie. W takich okolicznościach, aby dotrzeć do przystanku końcowego należy przesiadać się przynajmniej 2 razy.

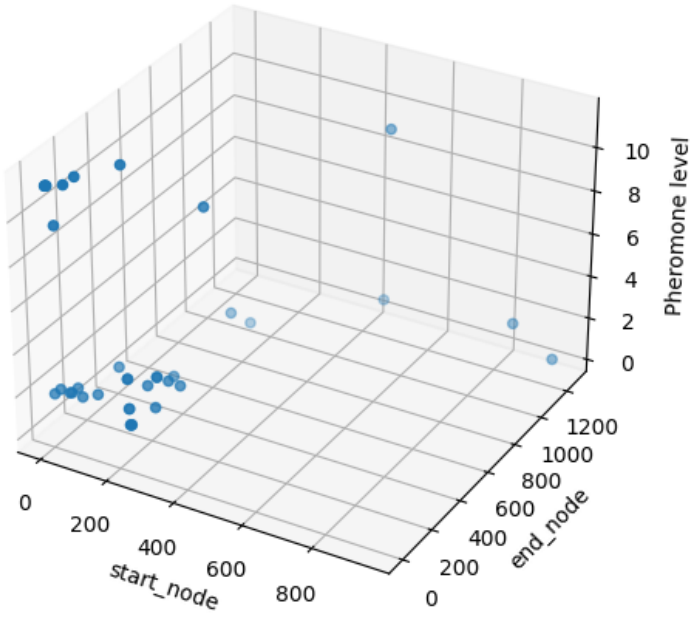

Pheromone level in 2 iteration
Pheromone level
0
1
2
3
4
start_node
0
100
200
300
400
end_node
0
200
400
600
800
1000
1200


Pheromone level in 4 iteration
Pheromone level
0
2
4
6
8
start_node
0
200
400
600
800
end_node
0
200
400
600
800
1000
1200

Pheromone level in 10 iteration

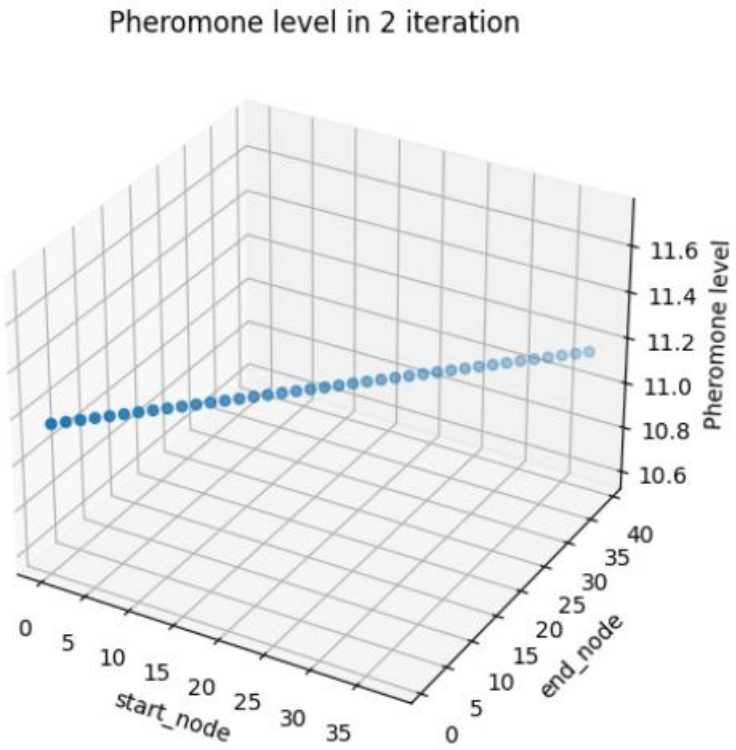


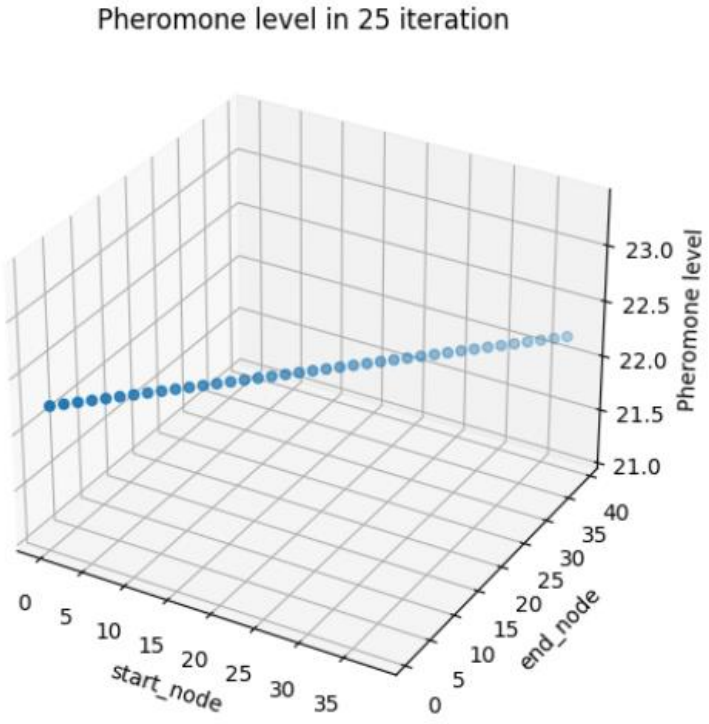
Pheromone level in 24 iteration



b) Testy drugiej wersji algorytmu (optymalnej)

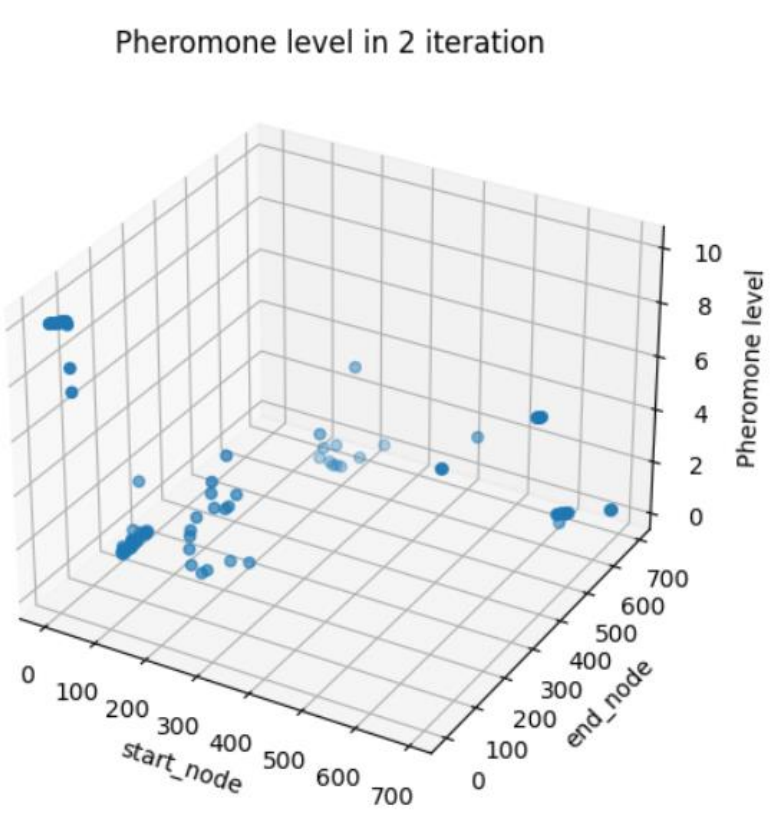
1) Przypadek optymistyczny

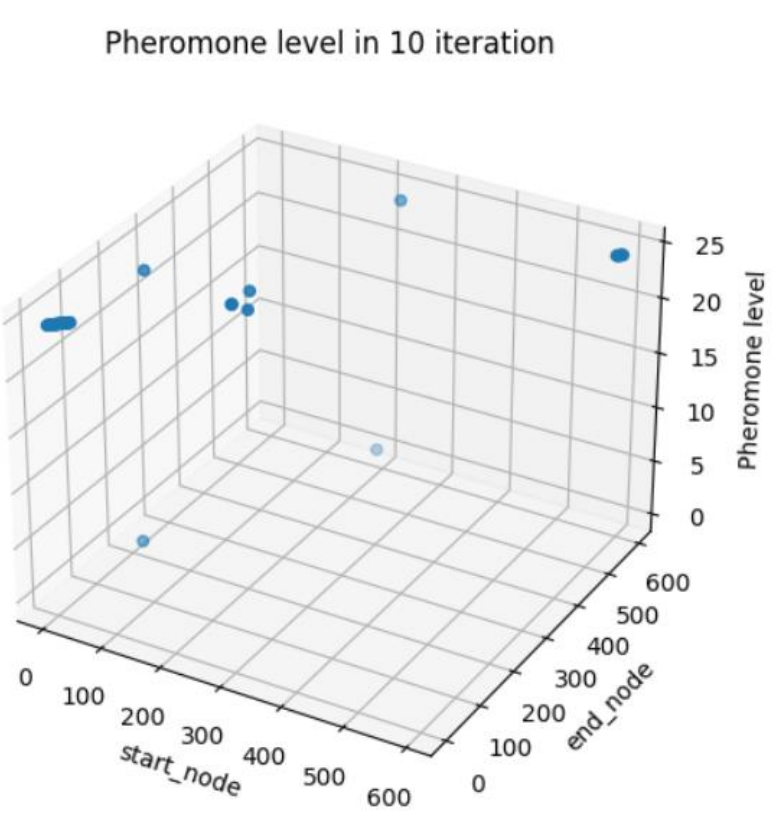


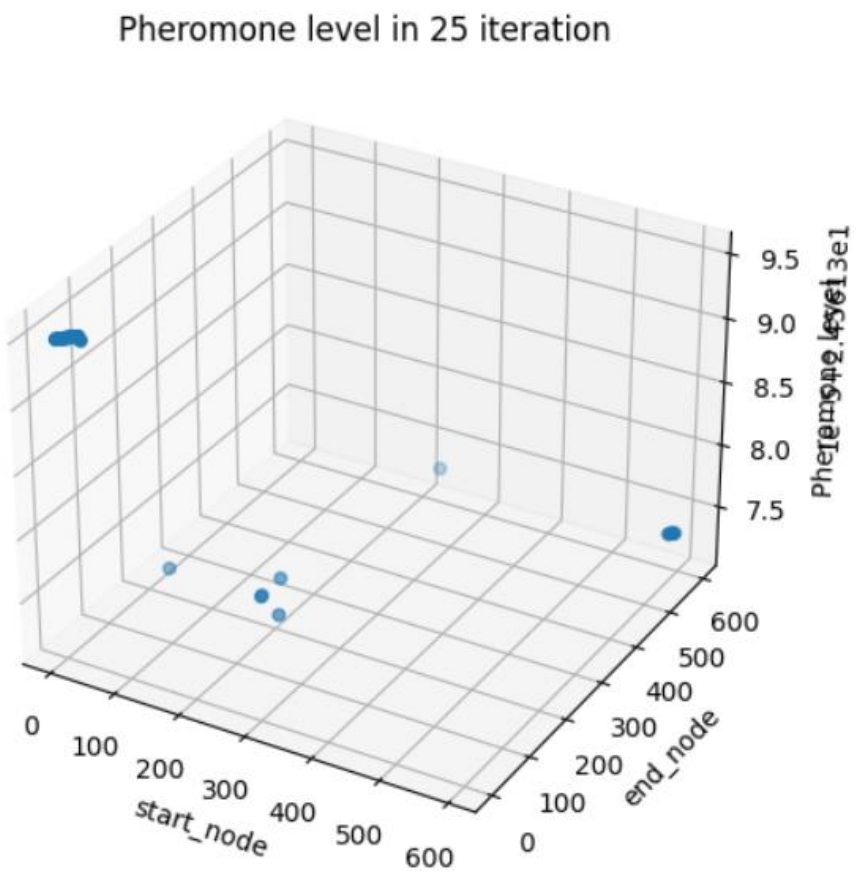


Możemy zauważyć że w tym przypadku funkcja optymalizacyjna nie miała żadnego wpływu na rozkład feromonów, gdyż te rozkłady prezentują się identycznie zarówno dla 2 iteracji jak i dla 25 algorytmu niezoptymalizowanego.

2) Przypadek pośredni

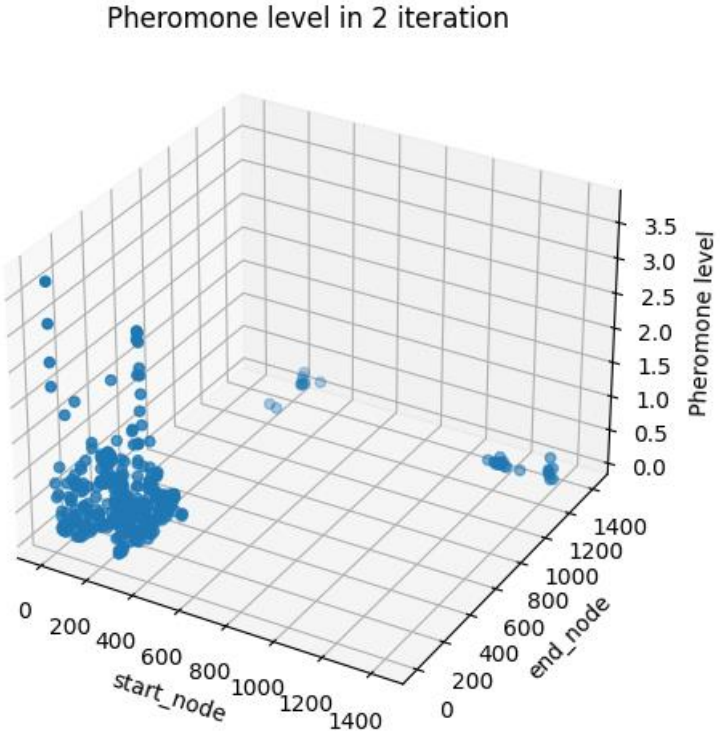


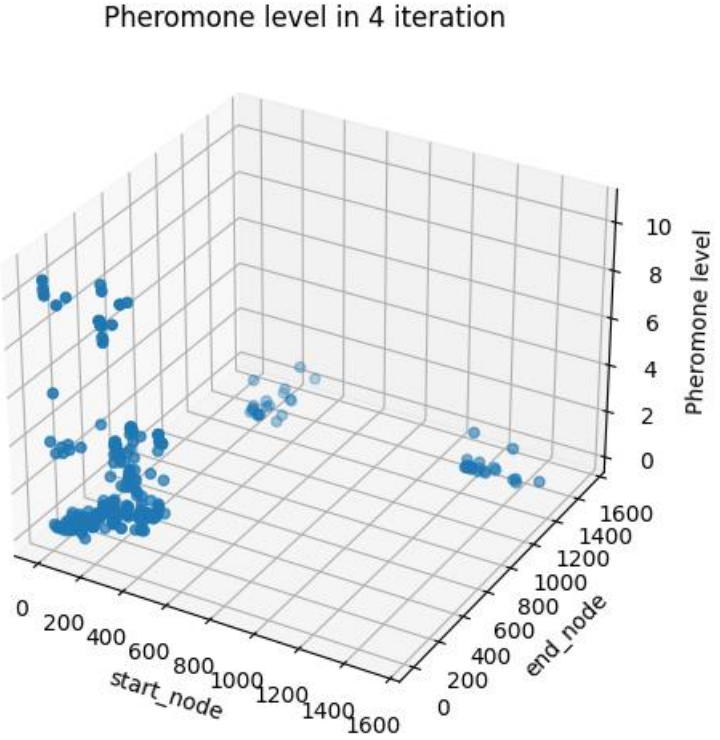




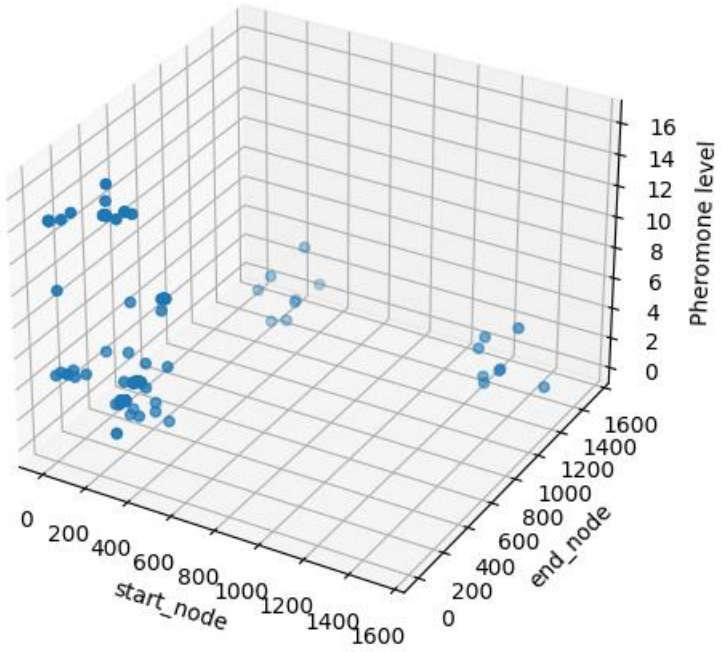
W tym przypadku możemy zauważyć że szybciej znajdowana jest właściwa ścieżka gdyż przy tej samej iteracji ilość feromonów jest mniejsza dla przypadku z funkcją optymalizacyjną.

3) Przypadek pesymistyczny

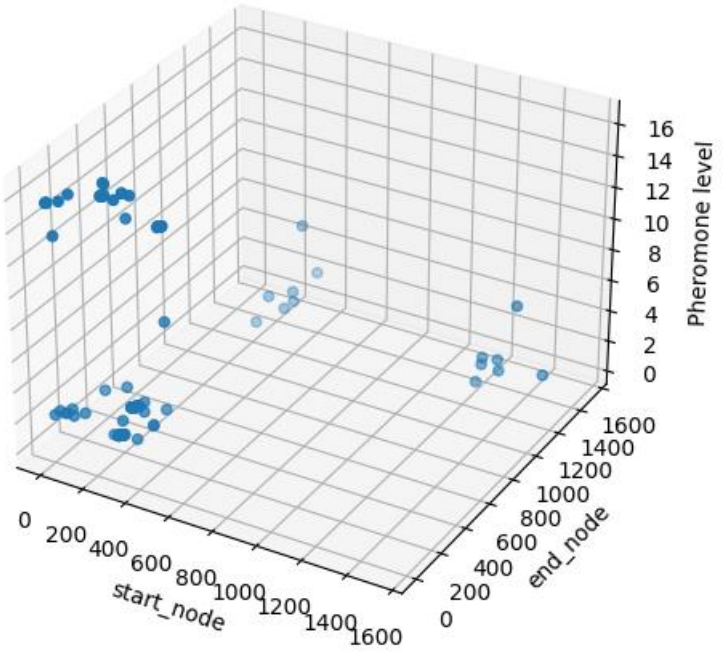




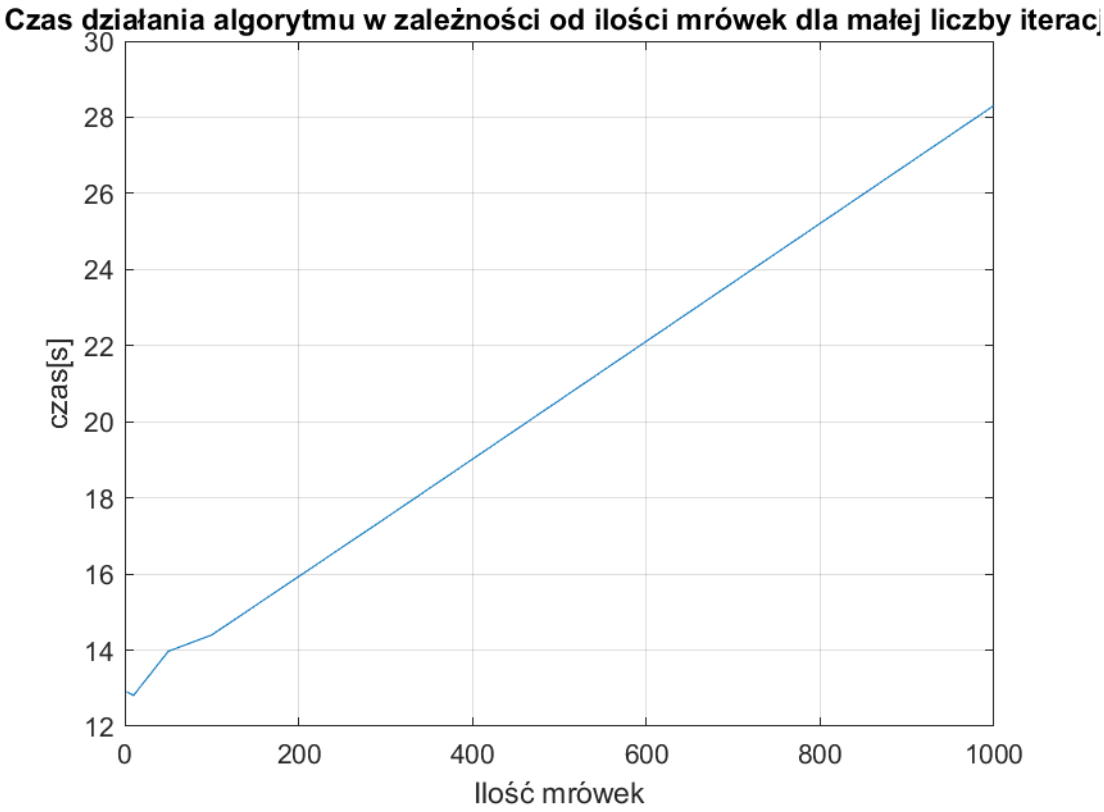
Pheromone level in 10 iteration

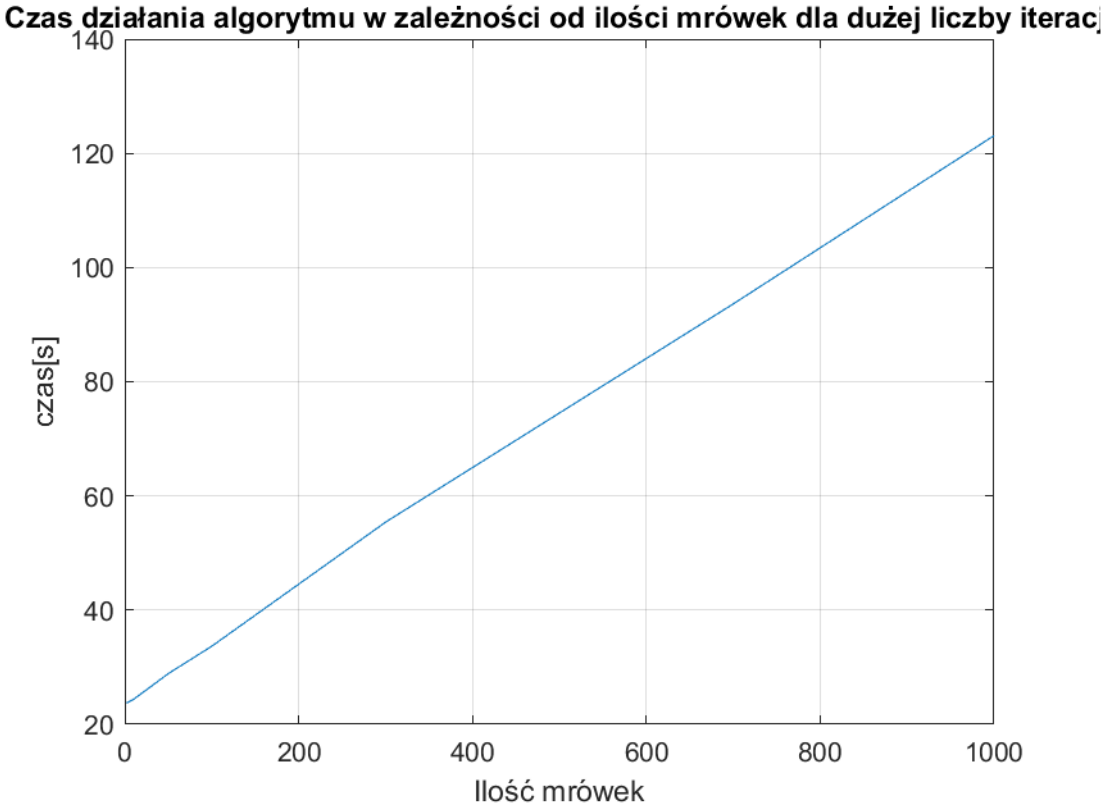


Pheromone level in 24 iteration



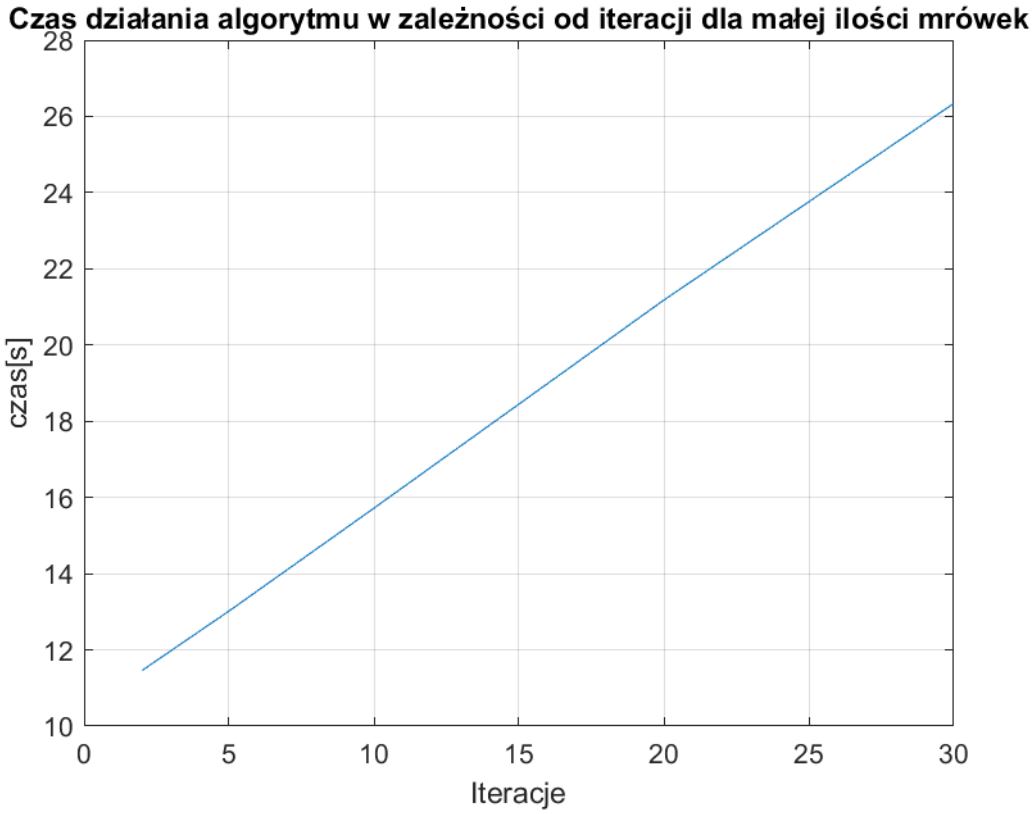
c) **Czas działania algorytmu w zależności od ilości mrówek dla małej oraz dużej ilości iteracji**

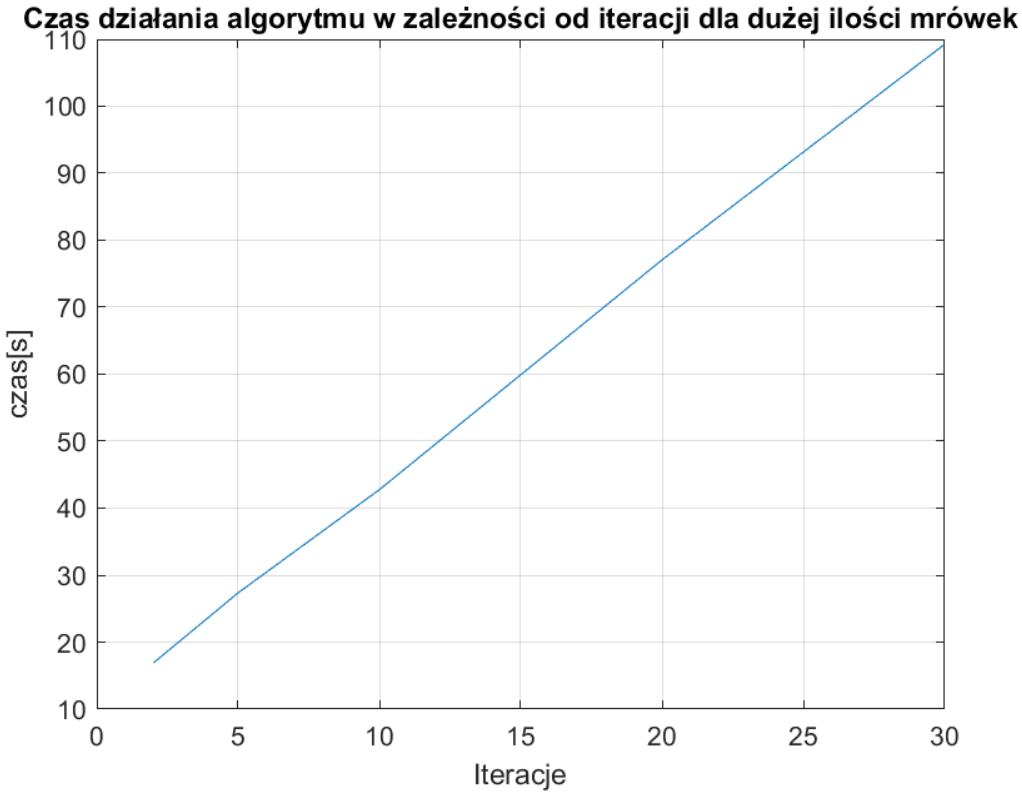




W obu przypadkach funkcję można aproksymować prostą, dlatego możemy stwierdzić że czas działania algorytmu rośnie liniowo wraz ze wzrostem ilości mrówek.

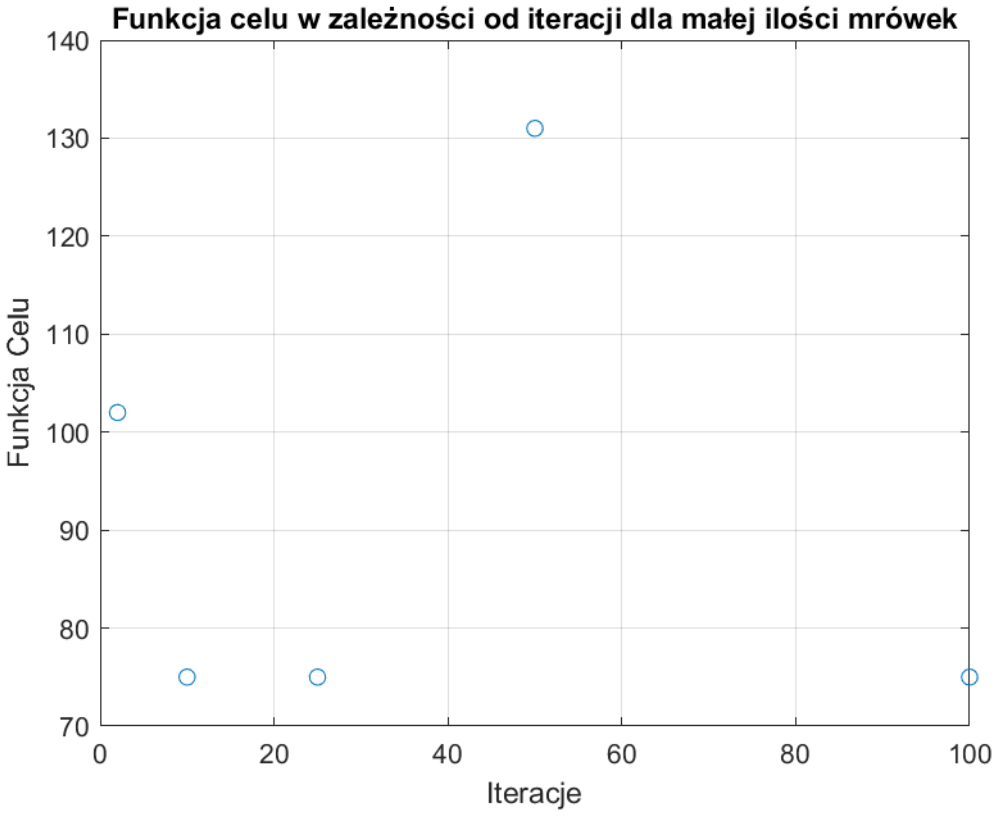
d) Czas działania algorytmu w zależności od iteracji dla małej oraz dużej ilości mrówek

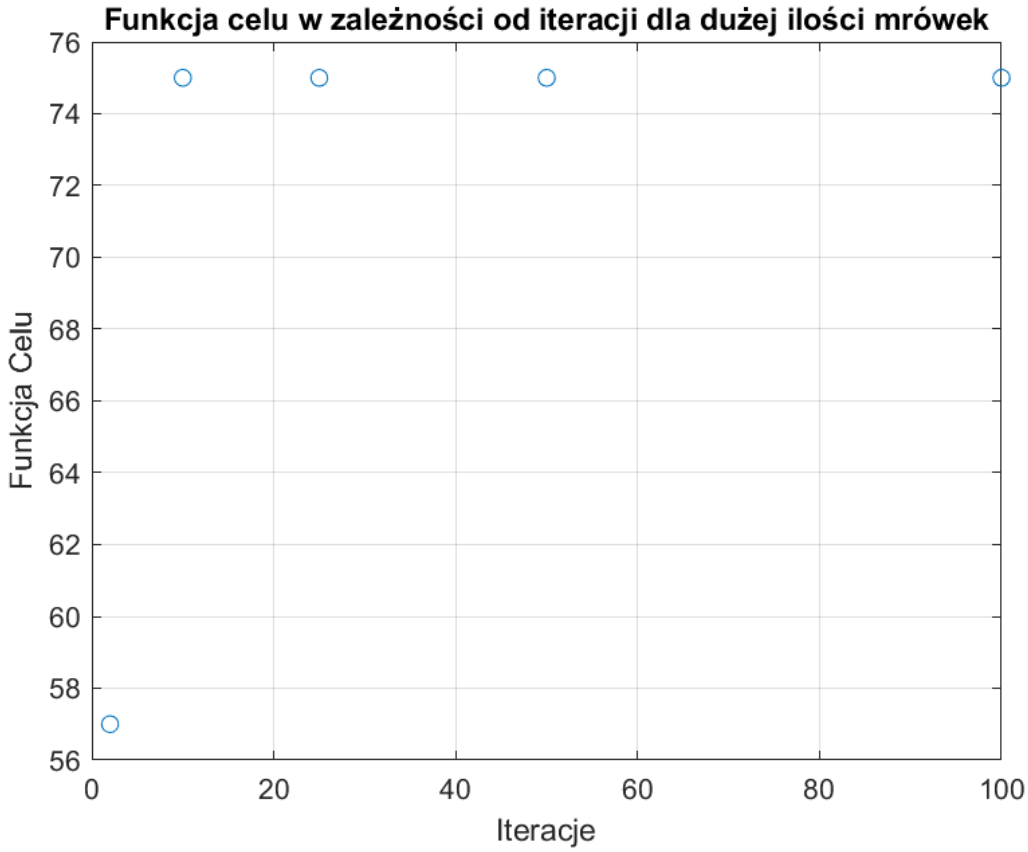




W obu przypadkach można aproksymować funkcję prostą, dlatego możemy stwierdzić, że czas działania algorytmu rośnie liniowo wraz z iteracjami.

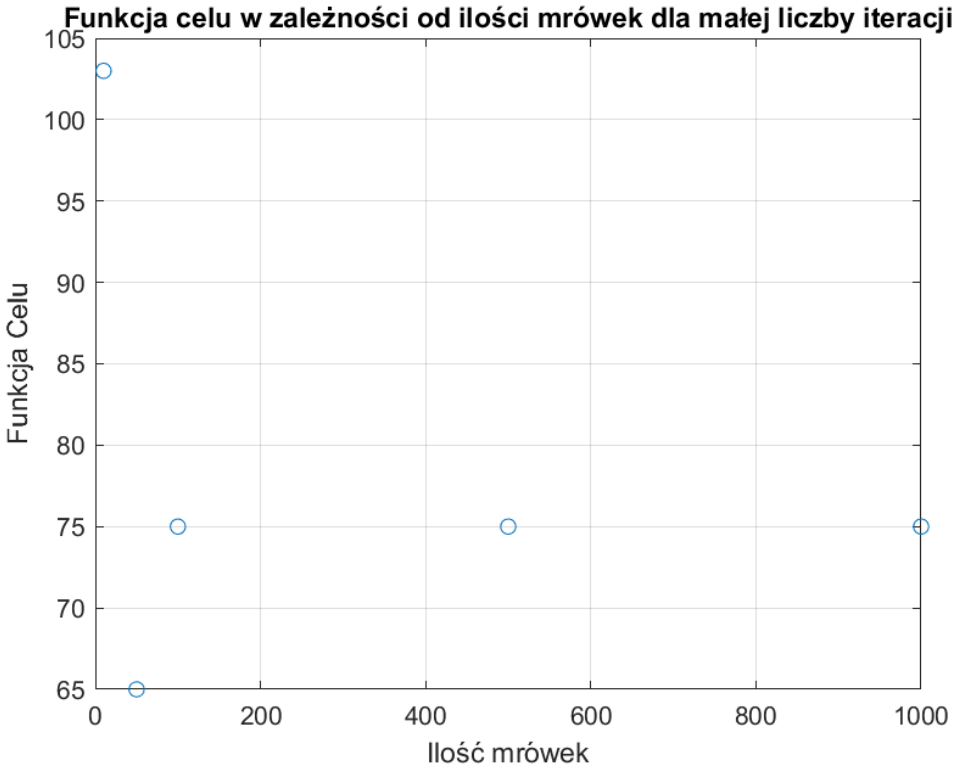
e) Wartość funkcji celu w zależności od iteracji dla małej oraz dużej ilości mrówek

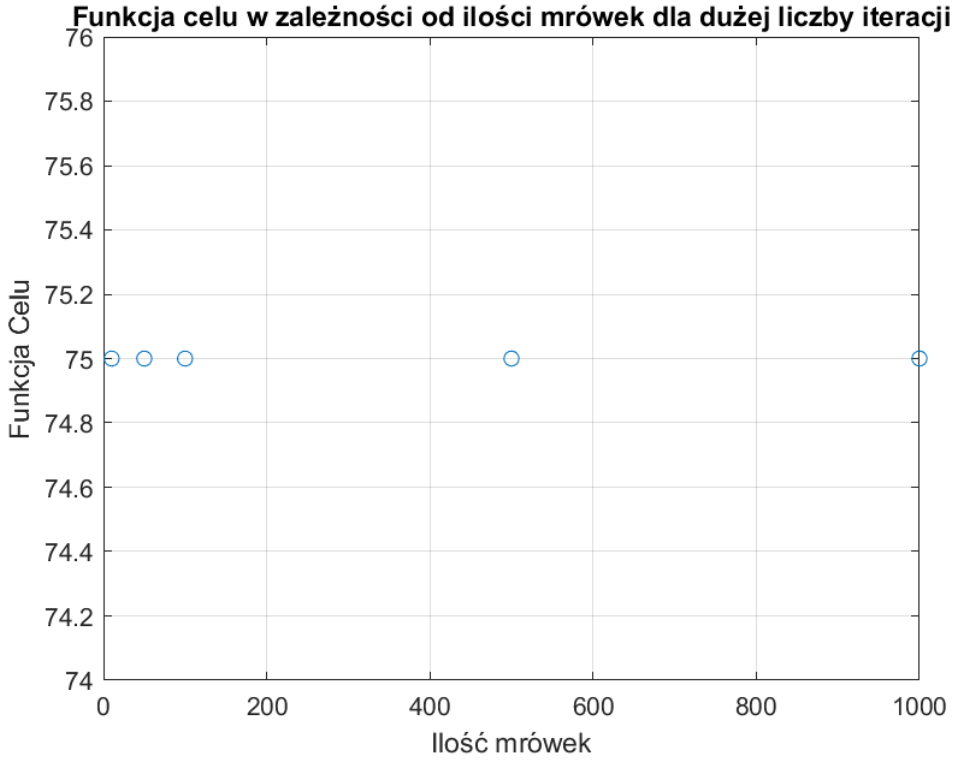




Ze względu na charakter rzeczywistych danych testowych (niewiele ścieżek z stosunkowo niewielkim kosztem docierających do przystanku końcowego) oraz fakt, że algorytm mrówkowy jest algorytmem probabilistycznym ilość iteracji nie wpływa istotnie na wartość funkcji celu.

f) Wartość funkcji celu w zależności od ilości mrówek dla małej oraz dużej liczby iteracji





W tym przypadku mamy do czynienia z podobną sytuacją jak w wartości funkcji celu w zależności od iteracji. Algorytm znajduje przede wszystkim rozwiązanie o funkcji celu = 75

4.2 charakterystyka zadań testowych

Nasze dane potrzebne do zadań testowych przechowujemy w 3 bazach danych. Ze względu na zgodność danych testowych wszystkie dane są z jednego dnia – 06.12.2020 (niedziela).

1. Baza danych przystanków

Baza danych przystanków zawiera łącznie 2308 rekordów. Każdy rekord reprezentuje jeden przystanek i posiada takie wartości jak:

- StopId – unikalny identyfikator przystanku
- StopDesc – Nazwa przystanku
- StopLat – szerokość geograficzna
- StopLon – długość geograficzna
- OnDemand – wartość Boolean (Prawda/Fałsz) określająca czy przystanek należy do przystanków na żądanie
- Date – data dla której dane są aktualne

2. Baza danych tras

Baza danych tras zawiera łącznie 133 różne linie komunikacyjne. Każdy rekord posiada poniższe wartości:

- RouteId – unikalny identyfikator linii komunikacyjnej
- RouteShortName – numer linii komunikacyjnej zgodny z tym wyświetlanym na pojeździe komunikacji miejskiej
- RouteLongName – pełna nazwa linii w formacie <przystanek początkowy>-<przystanek końcowy>
- Date - data dla której dane są aktualne

3. Baza danych rozkładów jazdy

Baza danych rozkładów jazdy zawiera łącznie 144 644 rozkłady jazdy. Każdy rekord zawiera takie wartości:

- Stop – przystanek z bazy danych przystanków
- Route – Linia komunikacyjna z bazy danych tras
- StopSequence – numer porządkowy w ramach danego kursu
- ArrivalTime – godzina odjazdu z przystanku
- Date – data dla której dane są aktualne
- BusServiceName – zadanie pojazdu. Kursy pogrupowane w zadanie tworzą rozkład jazdy dla pojedynczego pojazdu
- Order – numer porządkowy kursu w ramach zadania pojazdu

5. Podsumowanie

5.1 Wnioski

Stworzenie projektu dla danych rzeczywistych i wykorzystanie algorytmu mrówkowego w celu znalezienia najlepszej trasy dojazdu komunikacją miejską okazało się bardziej skomplikowane niż początkowo przypuszczaliśmy. Naszym pierwotnym założeniem było wyszukiwanie optymalnej trasy w jak najkrótszym czasie, aby użytkownik nie musiał długo czekać na wynik algorytmu. Niestety czasy, jakie otrzymaliśmy są bardzo dalekie od oczekiwań. Pomimo iż wyniki testowe dla naszych danych nie były do końca takie, jakich oczekiwaliśmy, jesteśmy zadowoleni, że udało nam się zintegrować algorytm z danymi rzeczywistymi i otrzymać całkiem dobre wyniki.

5.2 Stwierdzone problemy

Największym naszym problemem było scalenie danych rzeczywistych i dopasowanie algorytmu mrówkowego, początkowo stworzonego do działania z losowymi danymi liczbowymi do tych danych. Kolejnym problemem okazała się postać macierzy kosztów tzn. wielokrotność rozgałęzień grafy, które w sporej ilości przypadków były skierowane daleko od punktu docelowego poszukiwań. Mimo to udało nam się w pewnym stopni poradzić z tymi problemami i przetestować działanie algorytmu.

5.3 Kierunki dalszego rozwoju

- Stworzenie strony internetowej, na której użytkownik będzie mógł wyszukać optymalne połączenie komunikacją miejską. Zaimplementowanie na stronie funkcjonalności API Google Maps w celu zwizualizowania wyników działania algorytmu na interaktywnej mapie.
- Zoptymalizowanie działania algorytmu pod względem czasowym w celu uzyskania szybszej odpowiedzi o najlepszej trasie – konieczne w przypadku tworzenia strony internetowej.

6. Podział pracy

Imię i nazwisko	Mateusz Pilecki	Adam Sygut
Model matematyczny	Model matematyczny 50%	Opis słowny 50%
Algorytm Plik: Funkcje: Inne: Wkład:	 Ant_colony_optimization.py aco_algorithm() schemat algorytmy 50 %	 Ant_colony_optimization.py aco_algorithm() parametry algorytmu 50 %
Aplikacja Plik: Funkcje: Wkład: Inne: Wkład:	 Create_cost_matrix.py timedelta_to_minutes() return_cost_matrix_cost() return_cost_matrix_cost_for_row() convert_user_input_to_stop_id() convert_cost_dict_to_cost_matrix() find_label_dict_key_with_cost_matrix_label() create_cost_matrix() add_transport_costs() add_transport_costs_for_first_time() add_transport_costs_to_cost_dict() add_walking_costs() 280/550 linijek models.py – bazy danych 40 linijek	 Create_cost_matrix.py calculate_date_range() get_today_date(): load_json_from_url() load_stops_to_database() load_routes_to_database() load_timetables_to_database() load_databases() addEdge() visualize() add_stop_label() 270/550 linijek constants.py – dane testowe i stałe 75 linijek
Testy Plik: Funkcje:	 Tests_.py display_pheromone()	 Tests_.py create_plot()

Wkład:	50 %	50 %
Dokumentacja Rozdziały:	1.2 Rozdział 2 Rozdział 3	1.1 Rozdział 4 Rozdział 5