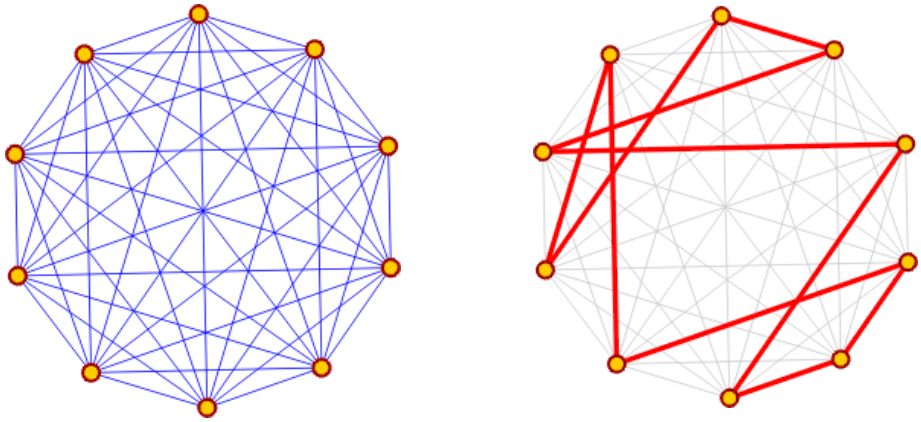
The Traveling Salesman

Problem komiwojażera w kontekście wyboru optymalnej wakacyjnej trasy podróży pośród polskich miast

1. Wstęp

1.1. Przedstawienie problemu komiwojażera

Problem komiwojażera (ang. TSP - Travelling Salesman Problem) uznawany jest za jeden z najtrudniejszych, a zarazem najbardziej popularnych i najszerzej badanych zagadnień w dziedzinie optymalizacji kombinatorycznej. Polega on na znalezieniu zamkniętej ścieżki łączącej zadaną listę miast, przy założeniu, że każde z nich musi zostać odwiedzone dokładnie raz. Trasa zaczyna się i kończy w tym samym miejscu, a jej koszt powinien być możliwie najniższy. W odniesieniu do matematycznej teorii grafów, miasta są wierzchołkami grafu pełnego, a trasy pomiędzy nimi to krawędzie z wagami. Waga krawędzi, w zależności od podejmowanego przedmiotu optymalizacji, może natomiast odpowiadać odległości pomiędzy danymi miastami, czasowi podróży czy kosztom przejazdu. Zakładając, że dla dowolnych miast A i B odległość z A do B jest taka sama jak z B do A, mówimy o symetrycznej odmianie problemu komiwojażera (STSP). Zagadnienie sprowadza się zatem do poszukiwania w grafie cyklu Hamiltona o minimalnej sumie wag krawędzi, przy czym warto zauważyć, że każdy graf pełny, z uwagi na skończoną liczbę wierzchołków, posiada co najmniej jedną taką ścieżkę:



Źródło: Serwis Edukacyjny I LO w Tarnowie, mgr Jerzy Wałaszek

1.2. Kontekst problemu optymalizacyjnego

Problem komiwojażera znajduje szerokie zastosowanie w wielu praktycznych dziedzinach, takich jak logistyka magazynowa, transport, planowanie tras, systemy nawigacyjne czy projektowanie sieci telekomunikacyjnych. Jest on zaliczany do klasy problemów NP-trudnych, co oznacza, że nie jest znany algorytm rozwiązujący je w czasie wielomianowym. Nie ma więc gwarancji znalezienia optymalnego rozwiązania dla wszystkich instancji problemu w czasie, który rośnie w sposób wielomianowy wraz ze wzrostem rozmiaru danych. W grafie pełnym mającym *n* wierzchołków liczba możliwych cykli Hamiltona wynosi aż $O(n-1)!/2$, co prowadzi do wykładniczej klasy złożoności obliczeniowej. W praktyce sprawdzenie wszystkich możliwych permutacji jest zatem wykonalne tylko

dla niewielkiej liczby wierzchołków, co z kolei sprawia, że efektywne metody gwarantujące optymalne rozwiązania istnieją jedynie dla małych zbiorów miast.

Poniżej przedstawiono zestawienie oszacowań złożoności obliczeniowej i czasowej rozwiązań problemu komiwojażera przy użyciu kilku najpopularniejszych algorytmów optymalizacyjnych:

Algorytm	**Złożoność obliczeniowa**	**Czas wykonania**
Przeszukiwanie zupełne	O(n!)	Bardzo długi czas wykonania
Algorytm zachłanny	O(n^2)	Szybki, ale suboptymalne rozwiązanie
Programowanie dynamiczne	O(2^n * n^2)	Przydatne dla niewielkich instancji problemu
K-Nearest Neighbours (K-NN)	O(n^2)	Szybki, ale suboptymalne rozwiązanie
Algorytmy genetyczne	Zależne od parametrów	Zależne od parametrów, zazwyczaj efektywne dla dużych instancji problemu
Algorytm mrówkowy	O(k * n^2), gdzie k-liczba iteracji	Efektywne dla dużych instancji problemu

Warto zauważyć, że algorytmy dokładne, takie jak przeszukiwanie zupełne i programowanie dynamiczne, mogą być bardzo kosztowne obliczeniowo i czasowo dla większych instancji problemu. Heurystyki i metaheurystyki, do których zaliczane są algorytmy genetyczne i algorytm mrówkowy, oferują szybsze rozwiązania, ale nie zawsze zapewniają optymalne wyniki. Ostateczny wybór algorytmu zależy od specyfiki problemu, dostępnych zasobów obliczeniowych oraz czasu, który można poświęcić na znalezienie rozwiązania.

1.3. Motywacja do zastosowania algorytmu mrówkowego

Algorytm mrówkowy jest jedną z metaheurystyk często wykorzystywanych do rozwiązywania problemu komiwojażera. Inspirację do jego powstania zaczerpnięto z natury poprzez obserwację zachowania kolonii mrówek, które w poszukiwaniu pożywienia pozostawiają feromony na ścieżkach, tworząc w ten sposób ślad feromonowy. ACO wykorzystuje zjawisko dodatniego sprzężenia zwrotnego, gdzie inne mrówki korzystają z feromonów jako wskazówek do wyboru tras. Strategię cechuje się zdolność do adaptacji poszukiwań do aktualnych informacji o środowisku. Dzięki temu możliwe staje się dostosowanie zachowania w odpowiedzi na zmieniające się warunki takie jak, chociażby zmienne koszty podróży.

Motywacja do zastosowania algorytmu mrówkowego w rozwiązaniu problemu komiwojażera wynika z jego zdolności do znalezienia suboptymalnego rozwiązania. Metoda ta pozwala przeszukiwać przestrzeń rozwiązań, balansując między eksploracją nowych ścieżek a eksploatacją już znalezionych rozwiązań. Dzięki temu, mimo że nie gwarantuje znalezienia optymalnego rozwiązania, może

efektywnie znajdować dobre przybliżenia w skomplikowanych instancjach problemu komiwojażera. Jego skalowalność powala na stosowanie strategii do rozwiązywania zarówno małych, jak i dużych instancji problemu. Może również uwzględniać dodatkowe ograniczenia, takie jak czas, priorytety lub inne czynniki wpływające na trasę. Ponadto algorytm ten cechuje łatwość implementacji. Jego idea jest stosunkowo prosta do zrozumienia, a realizacja nie wymaga wyspecjalizowanego sprzętu ani dużych zasobów obliczeniowych, co czyni go łatwo dostępnym i wpływa znacząco na popularność.

W niniejszym raporcie skupimy się na wykorzystaniu algorytmu mrówkowego do rozwiązania problemu komiwojażera na przykładzie poszukiwania optymalnej trasy po największych miastach w Polsce, analizując metodologię działania, główne zalety oraz wpływ poszczególnych hiperparametrów na efektywność poszukiwań i jakość osiąganych rezultatów. Przedstawimy również wyniki eksperymentów numerycznych, które ilustrują działanie algorytmu mrówkowego w praktyce.

2. Algorytm mrówkowy

2.1. Omówienie zasad działania algorytmu mrówkowego

Algorytm mrówkowy jest metaheurystyką inspirowaną zachowaniem kolonii mrówek podczas poszukiwania jedzenia. Inspiracje biologiczne w metaheurystyce są zjawiskiem powszechnym i pozwalają na opracowanie skutecznych technik rozwiązywania różnych problemów optymalizacyjnych.

Algorytm kolonii mrówek jest algorytmem polegającym na inteligencji stadnej. Wykorzystuje on kolektywne zachowanie niezależnych agentów (mrówek), którzy poprzez działanie w tej samej przestrzeni i wymianę informacji tworzą globalne wzorce. W kontekście optymalizacji inteligencja stadna służy rozwiązywaniu skomplikowanych problemów poprzez współdziałanie jednostek bez centralnej kontroli nad nimi.

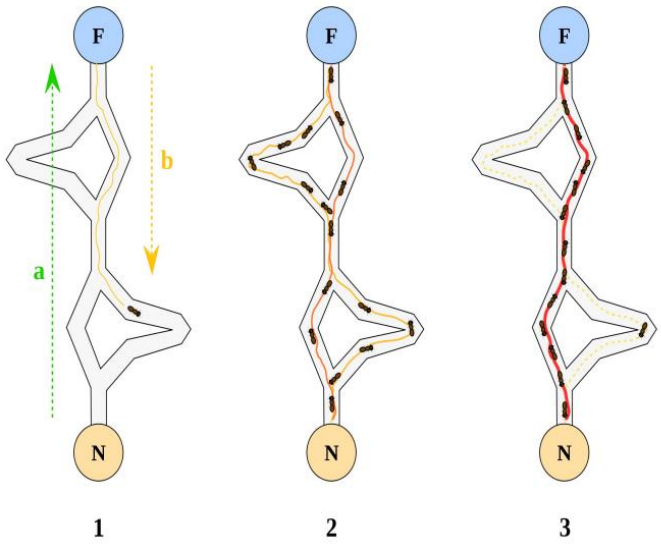
Główne charakterystyki kolonii:

- Elastyczność – kolonia potrafi odpowiedzieć na wewnętrzne perturbacje oraz zewnętrzne wyzwania.
- Odporność – zadania zostają wykonane przez kolonię, nawet jeśli pojedyncze jednostki zawiodą.
- Decentralizacja – nie ma odgórnej kontroli nad kolonią.
- Samoorganizacja – rozwiązania nie są predefiniowane, pojawiają się w miarę eksplorowania problemu.

Ostatnią z charakterystyk (samoorganizację) można również opisać jako zestaw dynamicznych mechanizmów, dzięki którym pojawiają się rozwiązania na szczeblu globalnym, dzięki

nieskoordynowanym interakcjom na poziomie pojedynczego agenta. Od strony teorii optymalizacji samoorganizację można zdefiniować jako składową czterech komponentów:

- Pozytywna informacja zwrotna (amplifikacja) – proces ten polega na wzmocnieniu znaczenia poprawnie wykonanego działania (odnalezienia krótszej drogi do celu) tak, aby w następnej iteracji miało ono większe prawdopodobieństwo zaistnienia.
- Negatywna informacja zwrotna – każda iteracja jest kolejnym elementem szeregu czasowego, każde rozwiązanie (zarówno dobre jak i złe) jest poddane ubytkowi części informacji, które ono zawiera. Ten parametr jest analogiczny do operatora mutacji w algorytmie genetycznym i służy uniknięciu ekstremum lokalnego na wczesnym etapie przeszukiwania przestrzeni rozwiązań.
- Amplifikacja fluktuacji – jest to kolejny mechanizm o charakterystyce globalnej. Jeśli część rozwiązań zaczyna błądzić w nieskoordynowanych z innymi agentami kierunkach, ich działanie jest wzmacniane przez pozytywną informację zwrotną. Zawsze istnieje prawdopodobieństwo, że któryś z błądzących agentów odnajdzie rozwiązanie, które niekoniecznie będzie szybsze, ale lepsze jakościowo. Ten mechanizm upewnia nas również, że nie istnieje potencjalnie dobre rozwiązanie, które nie zostało wypróbowane przez któregoś agenta.
- Interakcja – agenci przekazują między sobą informacje, które posiedli w danej iteracji.



Źródło : Michalina Gryniewicz-Jaworska Politechnika Lubelska, Wydział Elektrotechniki i Informatyki, Instytut Informatyki DOI: 10.5604/20830157.1130178

W jaki sposób do powyższych założeń odnoszą się tytułowe mrówki? Mrówki są owadami, które żyją w zorganizowanych społecznościach, których populacja może wynosić od 30 do milionów jednostek. Pomimo faktu, że mrówki nie potrafią się porozumiewać werbalnie, zauważalnym jest, że wiele czynności mrówki wykonują kolektywnie: zbieractwo, podział pracy, wspieranie słabszych

jednostek, opieka nad potomstwem czy konstrukcja mrowiska. Co więcej, mrówki potrafią reagować na bodźce zewnętrzne, w efekcie pomimo wykonywania prostych czynności indywidualnie istnieje wrażenie, że mrówki wykonują zadania w zorganizowany sposób.

Dylemat komiwojażera, którym zajmujemy się w poniższym raporcie łatwo zreplikować w świecie mrówek – będąc kolonią mrówek zamierzamy, trafić z punktu A (mrowisko) do punktu B (źródło pożywienia) napotykając po drodze *n* przeszkód (z góry formują potencjalną ilość tras, które kolonia może pokonać, zmierzając z punktu A do punktu B). Celem jest zatem znalezienie najkrótszej drogi (można rozważać, czy minimalizujemy czas, czy przebyty dystans) bez uwzględnienia odgórnego mechanizmu koordynującego działania kolonii. Kluczowy w tym procesie jest znacznie śladu trasy poprzez uwalnianie przez mrówki feromonów - związków chemicznych o charakterystycznym zapachu.

Można przyjąć, że w pierwszym rzucie kolonia pokona trasę między A i B w losowy sposób, każda mrówka wybierze swoją własną ścieżkę, jednocześnie każda z mrówek pozostawi ślad zapachowy na swojej trasie. Jeśli część mrówek wybierze tą samą, najszybszą trasę i w najkrótszym czasie osiągnie punkt końcowy B, w tym miejscu pojawi się najintensywniejszy ślad feromonu u wylotu tej trasy. Te same mrówki powrócą tą trasą, wiedząc, że ich trasa okazała się najszybsza, jednocześnie wzmacniając ślad zapachowy na tej trasie. Jeśli druga grupa mrówek podróżująca do punktu B inną trasą, dotrze do celu w czasie t dłuższym niż grupa pierwsza, spostrzegą one, że część mrówek już do tego miejsca dotarła poprzez wyczucie feromonów. Mrówki z tej grupy podejmą indywidualną decyzję – część z nich wróci trasa, którą przyszła, część wybierze tę trasę, gdzie ślad feromonu jest silniejszy. Iterując powyższy przykład przez ilość mrówek w kolonii, z każdą wyprawą z punktu A do punktu B coraz więcej mrówek będzie się decydowało na wybór trasy z najsilniejszym śladem feromonu, dodatkowo amplifikując ten ślad pokonując tą trasę.

Powyższy przykład jest oczywiście uproszczeniem skomplikowanego procesu decyzyjnego, pozwala on jednak w pewnym stopniu zoperacjonalizować zachowanie owadów w przyrodzie. Powracając do czterech komponentów, z których powinien się składać operator samoorganizacji w algorytmie kolonii mrówkowej, widoczne jest spełnienie założeń:

- Pozytywna informacja zwrotna – mrówki reagują na trasy z intensywniejszym stężeniem feromonu w procesie decyzyjnym.
- Negatywna informacja zwrotna – jeśli trasa jest przez mrówki porzucana, feromony stopniowo ewaporują, przez co spada prawdopodobieństwo wyboru tych tras w przyszłości.

- Amplifikacja fluktuacji – każda mrówka ma indywidualne prawo do samodzielnego poszukiwania trasy i znalezienia lepszego rozwiązania. Po osiągnięciu *k* niepowodzeń ostatecznie wybierze ona rozwiązanie z najwyższym stężeniem feromonu.
- Interakcja – mrówki porozumiewają się między sobą przy użyciu feromonów, sugerując lepsze rozwiązanie problemu.

2.2. Wyjaśnienie etapów algorytmu: inicjalizacja, konstrukcja rozwiązań, aktualizacja feromonów

Algorytm od strony programistycznej opiera się na wykonaniu 3 kroków, a następnie zapętleniu ich do momentu wyczerpania ilości iteracji lub znalezienia najlepszego rozwiązania przez wszystkich agentów:

1. Inicjalizacja – załóżmy sytuację, w której każdy agent zaczyna przeszukiwanie zbioru rozwiązań w tym samym punkcie. Każdy agent ma do wyboru k możliwości pierwszego ruchu. Prawdopodobieństwo jego wykonania jest oparte na tau stężeniu feromonów na krawędzi *d(i, j)* dzielącej punkt startowy od potencjalnego drugiego punktu. Podczas pierwszej iteracji przyznaje się losowe, niskie wartości tau dla każdej krawędzi *d(i, j)*. Wybór pomiędzy *k* ilością pierwszych destynacji dokonywany jest na podstawie wejściowego parametru *r* (również losowego, zadanego indywidualnie dla każdego agenta).
2. Konstrukcja rozwiązań i obliczenie ich kosztu – każdy z agentów pokonuje drogę do punktu końcowego w pierwszej iteracji w sposób losowy (wartości tau oraz r są losowe). Każdy z agentów pokona zatem drogę od punktu startowego do mety w losowej dla agenta ilości kroków. Każdy krok posiada z góry założony koszt (w przypadku problemu komiwojażera jest to odległość między punktami), który po wykonaniu iteracji się sumuje.
3. Dwustopniowa aktualizacja feromonów:
    a. Ewaporacja (negatywna informacja zwrotna) – po wykonaniu pierwszych tras przez agentów dokonuje się fragmentarycznego ubytku feromonów tau z każdej krawędzi w przestrzeni rozwiązań. Ewaporacji dokonuje się w oparciu o parametr ro (który ustala się na tym samym poziomie dla każdej krawędzi). Łatwo zatem zauważyć, że po każdym ruchu w wyniku ewaporacji, z każdej krawędzi ubędzie ten sam ułamek pierwotnego stężenia tau.
    b. Amplifikacja (pozytywna informacja zwrotna) – kluczowe dla poprawiania wyniku algorytmu w każdej iteracji jest wzmacnianie stężenia tau na każdej krawędzi, która została wybrana przez agenta. Jeżeli krawędź *d(i, j)*została wybrana przez 2 z 3 agentów w danej iteracji, to wartość tau (skorygowanego o ewaporację) poprawia się

o parametr *Q* (z góry założony, dla uproszczenia najczęściej jest to 1) podzielony przez kontrybucję agenta (obliczaną jako sumę kosztu poniesioną przez agenta na całej trasie). Widoczne jest zatem, że jeśli agent pierwszy pokonał trasę mniejszym kosztem niż agent drugi używając do tego krawędzi *d(i, j)* to jego kontrybucja do poprawy stężenia tau na tej krawędzi będzie większa (mniejszy mianownik ułamka = większa wartość kontrybucji).

2.3. Wpływ parametrów algorytmu na jego skuteczność

W powyższym opisie parametrycznym algorytmu widocznych jest kilka zmiennych, od których może zależeć jakość przeszukiwania przestrzeni przez algorytm. Wybór tych wartości na początku może znacznie wpłynąć na tempo jego pracy oraz jakość rozwiązań przez niego generowanych:

- Parametr tau – początkowe stężenie feromonów na danej krawędzi powinno być liczbą małą – wysoka wartość tau może ograniczyć wpływ ewaporacji na unikanie ekstremum lokalnego, z drugiej strony zbyt małe tau może doprowadzić do randomizacji przeszukiwania przestrzeni rozwiązań, w efekcie do rozwiązania problemów może być konieczna duża liczba iteracji.
- Liczba agentów – im więcej agentów tym potencjalnie szybciej algorytm odnajdzie optymalne rozwiązanie, z drugiej strony duża liczba agentów może spowolnić jego działanie.
- Parametr ro – od którego zależna jest cykliczna ewaporacja tau – jeżeli będzie zbyt mały, algorytm zacznie zbiegać do ekstremum lokalnego, jeżeli będzie zbyt duży, istnieje ryzyko utracenia znacznej części pozytywnej informacji zwrotnej generowanej przez każdą iterację, w efekcie może doprowadzić do opóźnienia znalezienia rozwiązania.

2.4. Zalety i ograniczenia algorytmu mrówkowego w kontekście problemu komiwojażera

Zaletą użycia algorytmu mrówkowego jest jego skuteczność w rozwiązywaniu problemów NP-trudnych, takich jak problem komiwojażera. Ponadto algorytm ten charakteryzuje się możliwością znalezienia dobrej jakości rozwiązania przy ograniczonej ilości czasu. Efektywność algorytmu mrówkowego zależy od odpowiedniego dobrania parametrów, takich jak współczynnik parowania feromonu, współczynniki wyboru ścieżki przez mrówki czy ilość feromonu pozostawiana na trasie. Algorytm jest wrażliwy na parametry, muszą być one dobrze dostrojone do danego problemu, nieodpowiednie ustawienie tych parametrów może prowadzić do otrzymania niesatysfakcjonujących wyników.

3. Opis i cel analizy

W związku z rozpoczynającym się okresem wakacyjnym chcemy zaplanować optymalną trasę po Polsce tak, aby odwiedzić łącznie 18 miast. Naszą podróż zaczynamy w Warszawie, a kończymy w Gdańsku. Podróż będziemy odbywać samochodem, dlatego wybór jak najlepszej trasy jest niezwykle istotny, ponieważ pozwoli nam skrócić czas spędzony w podróży, a także zmniejszyć koszty poniesione na paliwo. Dzięki odpowiedniej strategii i zaplanowanej wcześniej trasie będziemy w stanie z wyprzedzeniem zdecydować o pozostałych istotnych czynnikach jak noclegi czy znalezienie odpowiednich atrakcji co również może przyczynić się do zmniejszenia ogólnych kosztów podróży.

3.1. Rozwiązanie wybrane w celu przeprowadzenia analizy

Do implementacji algorytmu mrówkowego w projekcie korzystamy z pakietu AntColony w języku Julia. Główną funkcją w pakiecie służącą do uruchomienia algorytmu mrówkowego jest funkcja **aco,** która jako argumenty przyjmuje m.in. macierz odległości - w naszym przypadku macierz odległości pomiędzy poszczególnymi miastami odpowiednio zdefiniowana w kroku wcześniejszym oraz szereg parametrów, które w celu wybrania optymalnej trasy zostały początkowo ustawione w następujący sposób:

start_node - początkowy wierzchołek z którego rozpoczyna się podróż. W naszym przypadku jest to Warszawa, której indeks w macierzy odległości wynosi 16.

end_node - argument określający końcowy wierzchołek - punkt, w którym kończymy podróż - indeks miasta Gdańsk wynosi 3.

is_tour = true/false - jest to flaga określająca czy wracamy do początkowego miejsca podróży. W naszym przypadku kończymy podróż w Gdańsku, zatem wynikiem będzie jedynie optymalna ścieżka z miasta początkowego do miasta docelowego.

beta - współczynnik wpływu feromonów. Określa, jak bardzo feromony przyciągają mrówki do wyboru danej krawędzi w grafie. Im wyższa wartość, tym większe znaczenie feromonów.

rho - parametr określający szybkość odparowywania feromonów na krawędziach. Wyższa jego wartość oznacza szybsze odparowywanie feromonów.

q - wpływa na ilość feromonu pozostawianego przez każdą mrówkę na swojej trasie. W tym przypadku wartość 0.1 oznacza, że każda mrówka zostawia 10% feromonu na swojej trasie.

Q - ilość feromonu, jaką mrówka pozostawia na najlepszej trasie po każdej iteracji.

tau_min - minimalna wartość feromonu na krawędzi.

tau_max - maksymalna wartość feromonu na krawędzi.

max_iter - maksymalna liczba iteracji.

reset_iter - określa co o ile iteracji następuje reset feromonów na krawędziach.

top_perc_ants - procent najlepszych mrówek.

verbose = true/false - flaga określająca czy podczas działania algorytmu mają być wyświetlane dodatkowe informacje.

3.2. Opis działania wybranej metody

Po definiowaniu macierzy odległości następuje uruchomienie algorytmu ACO za pomocą funkcji aco, następnie obliczana jest całkowita długość trasy poprzez sumowanie odległości między miastami na znalezionej zoptymalizowanej ścieżce podróży. Algorytm wykonuje określoną liczbę iteracji, w której dla każdej mrówki, są podejmowane decyzje dotyczące wyboru kolejnych punktów na trasie. Następnie każda z mrówek rozpoczyna wędrówkę od punktu startowego (Warszawa), kierując się do kolejnych punktów na podstawie prawdopodobieństwa, które jest modyfikowane na podstawie feromonów i odległości między punktami.

Mrówki wybierają punkty zgodnie z regułą wyboru, która stara się równoważyć eksplorację (przez wybieranie mniej odwiedzanych tras) i eksploatację (przez wybieranie tras z większą ilością feromonów). Każda mrówka posiada pamięć o odwiedzonych wierzchołkach oraz sumę feromonów na każdej krawędzi. Następnie następuję aktualizacja ścieżki mrówki o nowo odwiedzane miasto, a także aktualizacja ilości feromonów na odwiedzonym odcinku drogi.

Na ścieżkach które zostały odwiedzone przez lepsze rozwiązania ilość feromonów się zwiększa a na pozostałych ścieżkach liczba ta stopniowo maleje. Po zakończeniu wszystkich iteracji obliczana jest całkowita odległość trasy, a zsumowany dystans zwracany w wyniku.

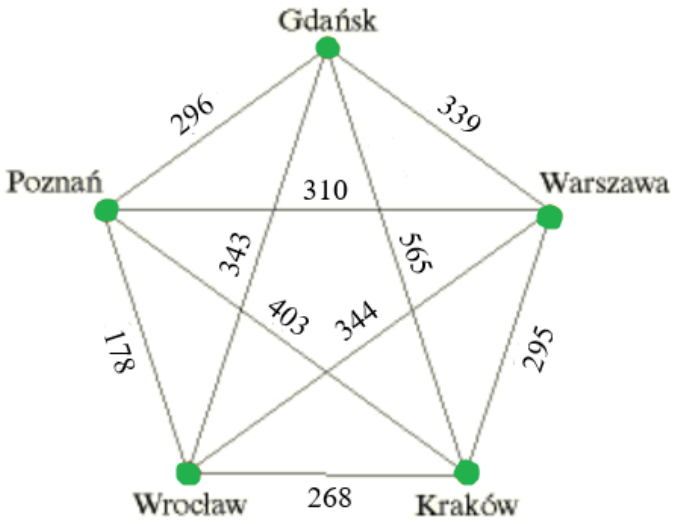
4. Przykład numeryczny

4.1. Dane wejściowe

Wyznaczenie optymalnej kolejności zwiedzania poszczególnych miast nie byłoby możliwe bez przygotowania odpowiednich danych wejściowych. Z tego powodu opracowany został wykaz rzeczywistych odległości drogowych między 18 największymi miastami w Polsce, uwzględniającym przy tym wszystkie miasta wojewódzkie. Aktualna siatka połączeń drogowych pozwala przyjąć założenie, że między dowolnie wybranymi miastami z listy istnieje droga dojazdowa. Przygotowana na podstawie internetowego kalkulatora macierz odległości jest ponadto symetryczna i obrazuje dystans między miastami na zasadzie „każdy z każdym”:

Dystans (km)	Białystok	Bydgoszcz	Gdańsk	Gorzów Wlkp.	Katowice	Kielce	Kraków	Lublin	Łódź	Olsztyn	Opole	Poznań	Rzeszów	Szczecin	Toruń	Warszawa	Wrocław	Zelona Góra
Białystok	0	389	378	603	485	363	477	260	322	222	507	491	430	656	347	188	532	601
Bydgoszcz	389	0	167	214	391	348	430	421	205	217	318	129	516	267	46	255	265	259
Gdańsk	378	167	0	315	545	483	565	500	340	156	485	296	642	348	181	339	432	411
Gorzów Wlkp.	603	214	315	0	454	480	529	594	341	431	354	129	643	105	260	439	266	109
Katowice	485	391	545	454	0	156	75	323	196	479	113	335	244	561	364	297	199	365
Kielce	363	348	483	480	156	0	114	167	143	394	220	354	163	585	307	181	300	422
Kraków	477	430	565	529	75	114	0	269	220	500	182	403	165	634	384	295	268	427
Lublin	260	421	500	594	323	167	269	0	242	370	382	465	170	683	375	161	428	542
Łódź	322	205	340	341	196	143	220	242	0	281	181	212	306	446	159	134	204	303
Olsztyn	222	217	156	431	479	394	500	370	281	0	452	323	516	484	177	213	442	453
Opole	507	318	485	354	113	220	182	382	181	452	0	261	347	459	312	319	86	245
Poznań	491	129	296	129	335	354	403	465	212	323	261	0	517	234	151	310	178	130
Rzeszów	430	516	642	643	244	163	165	170	306	516	347	517	0	751	470	303	433	585
Szczecin	656	267	348	105	561	585	634	683	446	484	459	234	751	0	313	524	371	214
Toruń	347	46	181	260	364	307	384	375	159	177	312	151	470	313	0	209	279	281
Warszawa	188	255	339	439	297	181	295	161	134	213	319	310	303	524	209	0	344	413
Wrocław	532	265	432	266	199	300	268	428	204	442	86	178	433	371	279	344	0	157
Zelona Góra	601	259	411	109	365	422	427	542	303	453	245	130	585	214	281	413	157	0

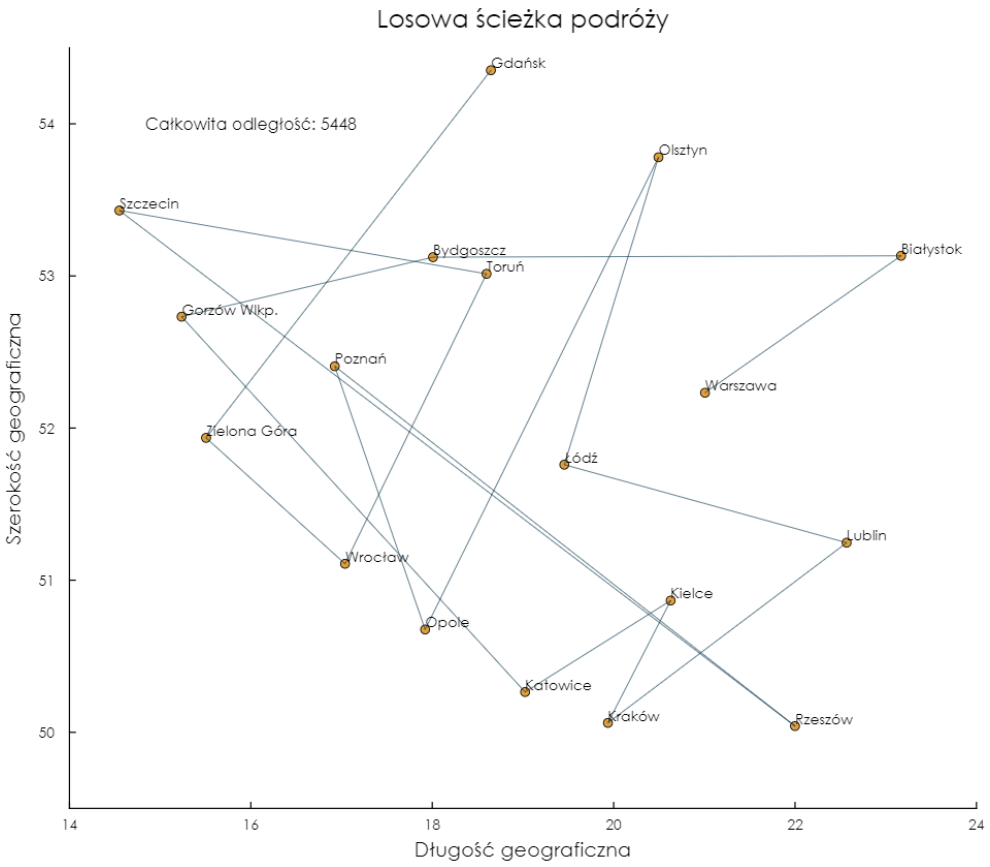
W odniesieniu do definicji problemu komiwojażera miasta te stanowią węzły grafu poddawanego procesowi przeszukiwania, w celu wyznaczenia najkrótszej ścieżki. Każda para miast połączona jest krawędzią z nadaną wagą, co odpowiada długości drogi wyrażonej w kilometrach. Otrzymujemy w ten sposób graf pełny, który ma tyle wierzchołków, ile miast musi odwiedzić komiwojażer, wliczając w to także miasto, z którego wyrusza. Sytuację obrazuje poniższy schemat, uproszczony do pięciu miast:



Źródło: Opracowanie własne

4.2. Implementacja algorytmu mrówkowego

W celu wyznaczenia punktu odniesienia do dalszych etapów implementacji algorytmu mrówkowego w pakiecie AntColony wyznaczyliśmy za jego pomocą trasę ustawiając parametry na minimum. Dla takich ustawień otrzymaliśmy losową sieć połączeń pomiędzy miastami o długości 5448 kilometrów.



Źródło: Opracowanie własne

4.3. Wpływ parametrów algorytmu na generowane rozwiązanie

Rozpoczynając optymalizację ścieżki, początkowo zwiększyliśmy podstawowe parametry. Modyfikacji podlegały:

- **tau:** został zwiększony do wartości maksymalnej 5.0 i minimalnej 1.0,
- wartość **rho** ustalono na 0.1 co powodowało wolne odparowywanie feromonów,
- **q:** przyjął wartość 0.1, co powodowało pozostawianie 10% feromonu na trasie mrówki,
- **Q:** również został ustawiony na 0.2, co pozostawiało większą liczbę feromonów na najlepszych trasach,
- maksymalna liczba iteracji wzrosła do 20.

Kod po modyfikacjach prezentował się następująco:

sciezka_mrowki = aco(macierz_odleglosci_miast, start_node=16, end_node=3, is_tour=false, rho=0.1, q=0.1, Q=0.2, tau_min=1.0, tau_max=5.0, max_iter=20, reset_iter=10, top_perc_ants=0.05, verbose=true)

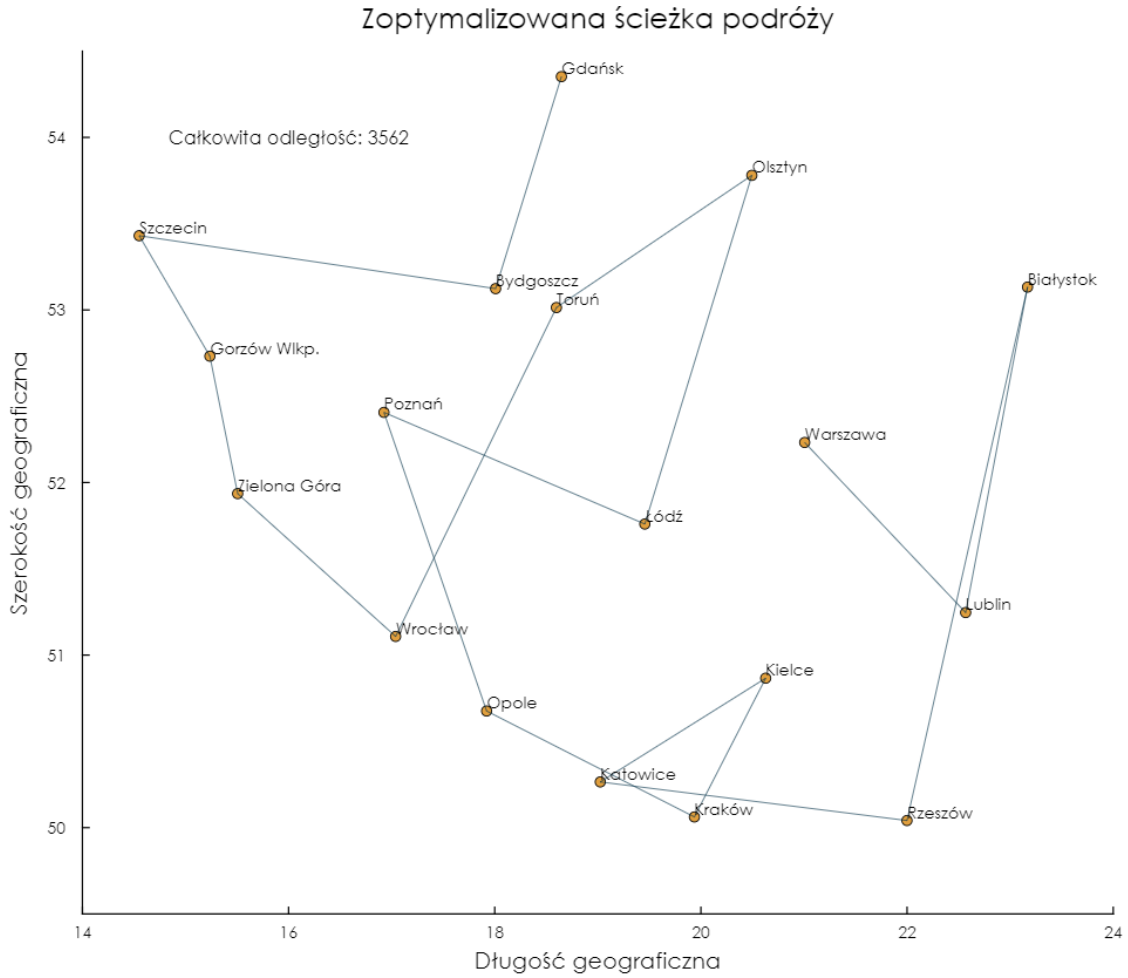
Dla powyższych ustawień wykonano 10 prób wyznaczenia najkrótszej trasy dla których otrzymaliśmy poniższe statystyki.

Opis statystyczny po pierwszej modyfikacji	
Średnia	3905.2
Błąd standardowy	68.28385
Mediana	3964.5

Dominanta	4014
Odchylenie standardowe	215.9325
Wariancja próbki	46626.84
Zakres	599
Minimum	3562
Maksimum	4161
Liczba prób	10

Źródło: Opracowanie własne

Jedna z prób zoptymalizowała długość trasy do 3562 kilometrów i stworzyła bardziej przejrzystą siatkę połączeń pomiędzy miastami, a najkrótsza trasa dla tej próby została wyznaczona w 12 iteracjach.



Źródło: Opracowanie własne

Podczas kolejnej modyfikacji dodaliśmy parametr beta, który przyjął wartość 0.1 oraz pozostawiając pozostałe współczynniki bez zmian. Dla tej konfiguracji kod prezentował się następująco:

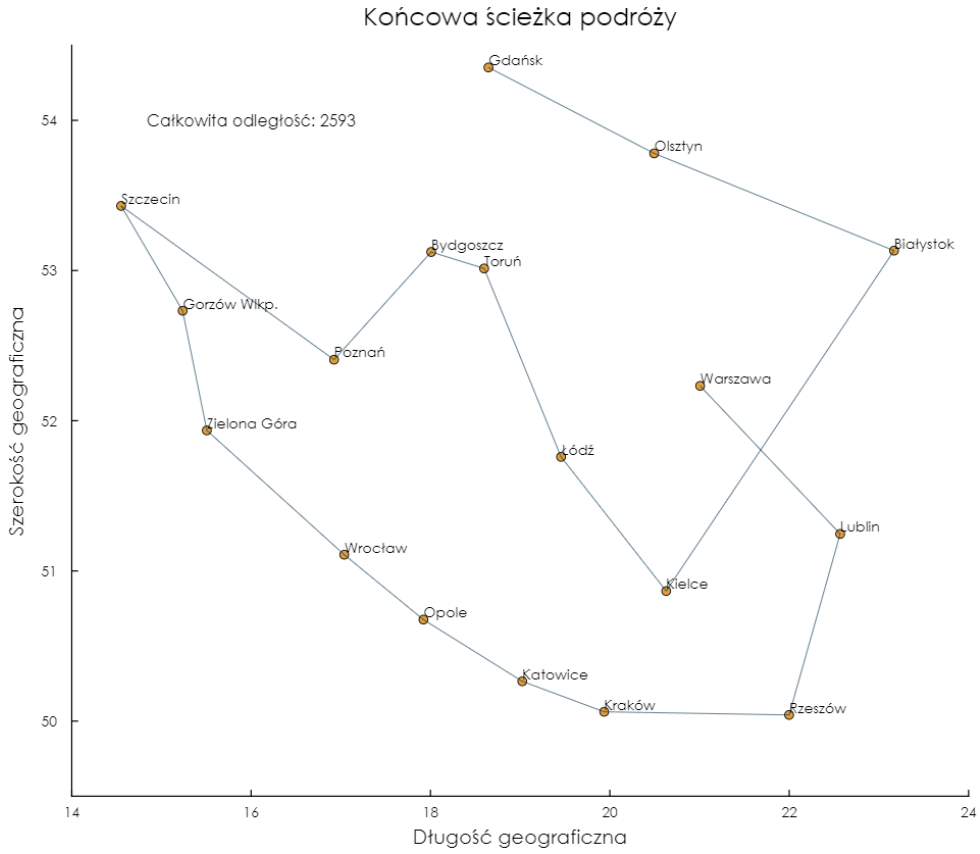
sciezka_mrowki = aco(macierz_odleglosci_miast, start_node=16, end_node=3, is_tour=false, rho=0.1, q=0.1, Q=0.2, tau_min=1.0, tau_max=5.0, max_iter=20, reset_iter=10, top_perc_ants=0.05, beta=0.1, verbose=true)

Dla 20 prób na powyższych ustawieniach otrzymaliśmy bardzo dobry i stabilny rezultat. Minimalna długość trasy którą udało się uzyskać to 2510 kilometrów. Natomiast z racji tego, że trasa o długości 2593 kilometrów występowała najczęściej, postanowiliśmy uznać ją za najbardziej optymalną trasę.

Opis statystyczny po drugiej modyfikacji	
Średnia	2590.5
Błąd standardowy	4.545964086
Mediana	2593
Dominanta	2593
Odchylenie standardowe	20.33016944
Zakres	116
Minimum	2510
Maksimum	2626
Liczba prób	20

Źródło: Opracowanie własne

Wygenerowane połączenie diametralnie się różni się od pierwszej losowej trasy i zamykają się w spójnej ścieżce. Różnica pomiędzy losową siatką połączeń, a aktualną trasą to 2855 kilometrów co jest skróceniem odległości o ponad połowę.



Źródło: Opracowanie własne

4.4. Stabilność wyników w powtórzeniach eksperymentu

Podczas kolejnych podejść do modyfikacji nie udało się skrócić długości trasy poniżej uzyskanych powyżej rezultatów. Żadna zmiana parametrów nie wpływała pozytywnie na wyniki. Aktualne ustawienia algorytmu zapewniają stabilny wynik, powtarzający się w czasie i osiągający taką samą wartość przy kolejnych próbach. Dodatkowo przy niewielkiej liczbie punktów końcowych algorytm nie potrzebował dużej liczby iteracji i generował najlepszy wynik dla średniej równej 7.

5. Podsumowanie

W celu zbadania algorytmu oraz znalezienia jak najlepszego rozwiązania naszego problemu przeprowadziliśmy serię eksperymentów, modyfikując różne parametry. Pierwszą modyfikacją było zwiększenie wartości kilku kluczowych wartości, takich jak tau, rho, q i Q, oraz zwiększenie maksymalnej liczby iteracji. Następnie dokonano kolejnej modyfikacji, dodając parametr beta o wartości 0.1. Po przeprowadzeniu 20 prób uzyskano stabilny rezultat. Nasza końcowa trasa wynosi 2593 km. Możemy zauważyć, że po odpowiednich zmianach uzyskano stabilne powtarzające się w czasie i osiągające podobną wartość przy kolejnych próbach rezultaty. Osiągnięto znaczną poprawę w porównaniu z początkową losową siecią połączeń między miastami, skracając długość trasy o ponad połowę. Kolejne próby modyfikacji parametrów nie przyniosły poprawy rezultatu, co sugeruje, że aktualne ustawienia algorytmu są optymalne dla analizowanych danych. Dalsze modyfikacje mogą wpływać na wyniki, jednak nie zawsze przynoszą pozytywne rezultaty.

6. Literatura

- Traveling Salesman Problem, Federico Greco, InTech, 2008
- Metaheuristics for Hard Optimization, Johann Dréo, Alain Pétrowski, Patrick Siarry, Eric Taillard
- Ant Colony Optimization, M. Dorigo, T. Stutzle
- Algorytm mrówkowy w problemie komiwojażera, Katarzyna Rutczyńska-Wdowiak, Norbert Jabłoński
- The traveling salesman problem, Corinne Brucato, University of Pittsburgh, 2013
- AntColony · Julia Packages https://www.juliapackages.com/p/antcolony

7. Kod źródłowy

```
# Instalacja pakietów
using Pkg
#Pkg.add("AntColony")
#Pkg.add("Plots")
```

```
# Importowanie pakietów

using AntColony
using Plots

# Definicja macierzy odległości
macierz_odleglosci_miast = [
  0   389 378 603 485 363 477 260 322 222 507 491 430 656 347 188 532 601; # odległość Białystok
  389 0   167 214 391 348 430 421 205 217 318 129 516 267 46  255 265 259; # odległość Bydgoszcz
  378 167 0   315 545 483 565 500 340 156 485 296 642 348 181 339 432 411; # odległość Gdańsk
  603 214 315 0   454 480 529 594 341 431 354 129 643 105 260 439 266 109; # odległość Gorzów Wlkp.
  485 391 545 454 0   156 75  323 196 479 113 335 244 561 364 297 199 365; # odległość Katowice
  363 348 483 480 156 0   114 167 143 394 220 354 163 585 307 181 300 422; # odległość Kielce
  477 430 565 529 75  114 0   269 220 500 182 403 165 634 384 295 268 427; # odległość Kraków
  260 421 500 594 323 167 269 0   242 370 382 465 170 683 375 161 428 542; # odległość Lublin
  322 205 340 341 196 143 220 242 0   281 181 212 306 446 159 134 204 303; # odległość Łódź
  222 217 156 431 479 394 500 370 281 0   452 323 516 484 177 213 442 453; # odległość Olsztyn
  507 318 485 354 113 220 182 382 181 452 0   261 347 459 312 319 86  245; # odległość Opole
  491 129 296 129 335 354 403 465 212 323 261 0   517 234 151 310 178 130; # odległość Poznań
  430 516 642 643 244 163 165 170 306 516 347 517 0   751 470 303 433 585; # odległość Rzeszów
  656 267 348 105 561 585 634 683 446 484 459 234 751 0   313 524 371 214; # odległość Szczecin
  347 46  181 260 364 307 384 375 159 177 312 151 470 313 0   209 279 281; # odległość Toruń
  188 255 339 439 297 181 295 161 134 213 319 310 303 524 209 0   344 413; # odległość Warszawa
  532 265 432 266 199 300 268 428 204 442 86  178 433 371 279 344 0   157; # odległość Wrocław
  601 259 411 109 365 422 427 542 303 453 245 130 585 214 281 413 157 0   # odległość Zielona Góra
]

# Uruchamia algorytm ACO
sciezka_mrowki = aco(
        macierz_odleglosci_miast,
        start_node=16,
        end_node=3,
        is_tour=false,
        beta=1,
        rho=0.1,
        q=0.1,
        Q=1,
        tau_min=1.0,
        tau_max=5.0,
        max_iter=20,
        reset_iter=10,
        top_perc_ants=0.05,
```

```
            verbose=true)

# Oblicza całkowitą odległość dla ścieżki
calkowita_odleglosc = sum(macierz_odleglosci_miast[sciezka_mrowki[i], sciezka_mrowki[i+1]] for i
in 1:length(sciezka_mrowki)-1)

# Liczba miast do odwiedzenia
miasta = [
        "Białystok", "Bydgoszcz", "Gdańsk",
        "Gorzów Wlkp.", "Katowice", "Kielce",
        "Kraków", "Lublin", "Łódź", "Olsztyn",
        "Opole", "Poznań", "Rzeszów", "Szczecin",
        "Toruń", "Warszawa", "Wrocław", "Zielona Góra"
    ]

# Położenie geograficzne miast
polozenie_miast = [
  (53.132488, 23.168840),  # Białystok
  (53.123480, 18.008438),  # Bydgoszcz
  (54.352025, 18.646638),  # Gdańsk
  (52.732097, 15.236571),  # Gorzów Wlkp.
  (50.264892, 19.023782),  # Katowice
  (50.866077, 20.628567),  # Kielce
  (50.061947, 19.936856),  # Kraków
  (51.246453, 22.568446),  # Lublin
  (51.759248, 19.455983),  # Łódź
  (53.779959, 20.494184),  # Olsztyn
  (50.675845, 17.921290),  # Opole
  (52.406374, 16.925168),  # Poznań
  (50.041187, 21.999121),  # Rzeszów
  (53.430181, 14.550962),  # Szczecin
  (53.013790, 18.598444),  # Toruń
  (52.231923, 21.006726),  # Warszawa
  (51.107883, 17.038538),  # Wrocław
  (51.935621, 15.506208)   # Zielona Góra
]

# Tworzy wykres punktowy miast
scatter(
      [koordynaty[2] for koordynaty in polozenie_miast],
      [koordynaty[1] for koordynaty in polozenie_miast],
      legend = false,
```

```
        color = "#e09f3e",
        markersize = 4,
        markerstrokewidth = 0.5,
        xlim = (14.0, 24.0),
        ylim = (49.5, 54.5))

# Tytuł wykresu
title!("Końcowa ścieżka podróży")

# Tytuły osi
xlabel!("Długość geograficzna")
ylabel!("Szerokość geograficzna")

# Dodanie etykiet do miast
for i in 1:length(miasta)
    annotate!(
            polozenie_miast[i][2],
            polozenie_miast[i][1],
            text(miasta[i],
            :black,
            :bottom,
            :left,
            :auto,
            0
            )
            )
end

# Dodanie linii łączących miasta
for i in 1:(length(sciezka_mrowki) - 1)
    plot!(
        [polozenie_miast[sciezka_mrowki[i]][2], polozenie_miast[sciezka_mrowki[i+1]][2]],
        [polozenie_miast[sciezka_mrowki[i]][1], polozenie_miast[sciezka_mrowki[i+1]][1]],
        color = "#023047",
        line = :solid,
        linewidth = 0.5)
end

# Dodanie etykiety z całkowitą odległością
annotate!(
        16.0,
        54.0,
```

```
        text(
            "Całkowita odległość: $calkowita_odleglosc",
            font(9, "Calibri"))
            )

# Ustawienie rozmiaru wykresu i czcionki
plot!(size=(800, 700), grid=false, fontfamily= "Calibri", 11)

# Wyświetlenie wykresu
plot!()
```