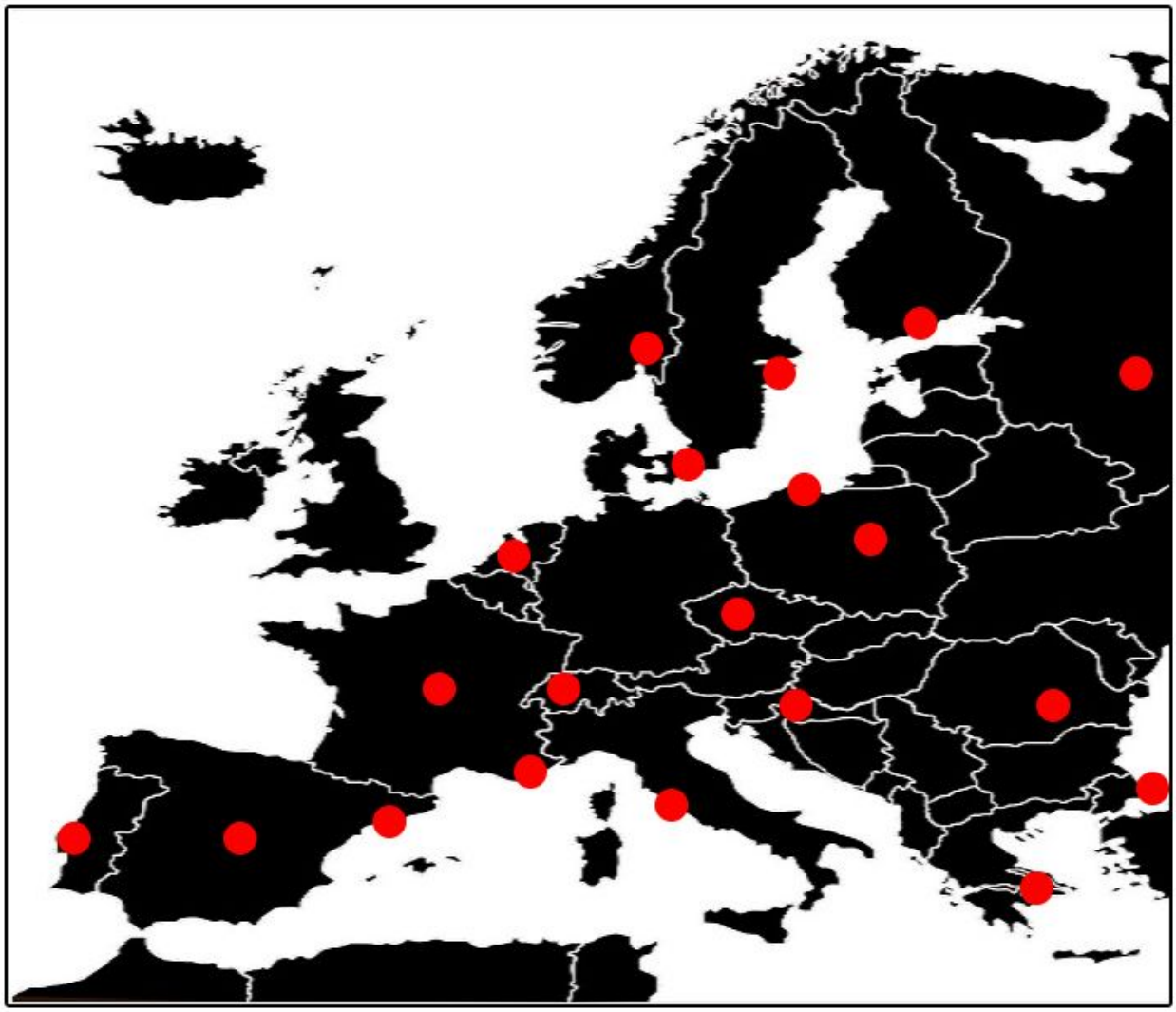
Metody metaheurystyczne

Małgorzata Kindrat 186824

Patrycja Kazek 186799

DEFINICJA PROBLEMU

Celem badania jest opracowanie trasy samochodowej po 20 najciekawszych miejscach w Polsce/Europie. Autorki badania zdecydowały się na wybór europejskich miast, na podstawie których opracują najbardziej optymalną trasę podróży. Do przeprowadzenia testów zostały wytypowane takie miasta jak: Madryt, Lizbona, Barcelona, Paryż, Kopenhaga, Warszawa, Ateny, Stambuł, Amsterdam, Oslo, Sztokholm, Helsinki, Moskwa, Rzym, Bukareszt, Monako, Praga, Gdynia, Zagrzeb, Berno.



Rys. 1 Mapa z miastami, które zostały wytypowane.

Odległości pomiędzy poszczególnymi miastami zostały przedstawione w tabeli poniżej, zostały one określone na podstawie odległości podawanych w km przez serwis internetowy Google Maps.

Tabela 1. Odległości pomiędzy poszczególnymi miastami.

	Madryt	Lizbona	Barcelona	Paryż	Kopenhaga	Warszawa	Ateny	Istambuł	Amsterdam	Oslo	Sztokholm	Helsinki	Moskwa	Rzym	Bukareszt	Monako	Praga	Gdynia	Zagrzeb	Berno
Madryt	0	625	624	1275	2488	2922	3245	3537	1773	2966	3134	3992	4103	1962	3178	1283	2315	2805	2194	1541
Lizbona	629	0	1248	1735	2948	3382	3799	4090	2233	3427	3593	4387	4563	2525	3731	1836	2702	3265	2747	2010
Barcelona	621	1248	0	1038	2143	2354	3081	2940	1530	2582	2749	3420	3592	1361	2581	686	1712	2455	1593	941
Paryż	1270	1736	1037	0	1222	1638	2872	2732	507	1701	1868	2661	3,12,	1421	2315	954	1030	1539	1388	570
Kopenhaga	2482	2948	2143	1215	0	1009	2769	2628	789	603	659	1132	2250	1901	2137	1737	783	537	1470	1234
Warszawa	2918	3385	2354	1639	1011	0	2188	2191	1193	1418	1473	1080	1251	1794	1265	1826	635	374	1044	1446
Ateny	3241	3798	2646	2949	2771	2336	0	1094	2858	3362	3417	3237	3286	1273	1180	1932	1989	2585	1487	2461
Istambuł	3530	4086	2934	2803	2625	2190	1094	0	2713	3216	3272	2732	2150	2226	638	2258	1844	2439	1342	2315
Amsterdam	1769	2235	1562	508	799	8,5	2056	2716	0	1269	1436	1910	2434	1653	2225	1410	877	1140	1324	836
Oslo	2958	3425	2631	1692	606	1427	3363	3222	1266	0	592	992	1923	2472	2731	2225	1377	1105	2064	1722
Sztokholm	3126	3592	2787	1860	657	1477	3414	3273	1434	522	0	479	1439	2545	2724	2381	1428	1156	2114	1879
Helsinki	3905	4438	3408	2693	1136	1065	3410	3270	2247	984	484	0	1101	2873	2325	2880	1731	1140	2123	2500
Moskwa	4163	4629	12,2	2883	2256	1255	3281	2157	2438	1919	1439	1107	0	3060	1781	3070	1922	1471	2313	2690
Rzym	1957	2513	1361	1420	1901	1795	1300	2229	1651	2475	2548	2889	3065	0	1870	685	1299	2044	885	876
Bukareszt	3342	3899	2575	2313	2135	1262	1181	638	2223	2726	2782	2338	1771	1867	0	1899	1354	1650	983	1941
Monako	1281	1837	685	954	1736	1823	1970	2261	1414	2225	2382	2893	3068	686	1902	0	1167	1925	918	558
Praga	2310	2777	1716	1031	783	635	1987	1846	877	1374	1430	1751	1923	1299	1355	1170	0	786	641	804
Gdynia	2979	3368	2464	1635	574	372	2580	2440	1210	1104	1160	1141	1468	2043	1653	1925	859	0	1293	1545
Zagrzeb	2191	2748	1596	1384	1475	1050	1484	1343	1326	2066	2121	2144	2191	888	984	920	699	1299	0	976
Berno	1538	2004	943	569	1234	1445	2458	2317	833	1724	1881	2514	2686	931	1942	562	804	1546	974	0

Zagadnienie przedstawione w badaniu reprezentuje klasyczny problem komiwojażera (ang. travelling salesman problem, TSP), którego celem jest znalezienie, w badanym przypadku, najkrótszej trasy pomiędzy wszystkimi miastami. Na jego podstawie porównane zostanie działanie dwóch wybranych algorytmów metaheurystycznych .

OPIS ZASTOSOWANYCH METOD

Do rozwiązania wyżej wymienionego problemu wykorzystane zostały algorytmy symulowanego wyżarzania oraz genetyczny zaimplementowane w języku JavaScript.

Algorytm symulowanego wyżarzania został pierwotnie zainspirowany procesem wyżarzania w obróbce metalu. Polega ona na stopniowym podgrzewaniu i schładzaniu materiału w celu maksymalizacji rozmiaru kształtów i uniknięcia powstawania defektów. Zasada działania zaimplementowanego algorytmu:

1) Ustalenie początkowej temperatury i losowe wygenerowanie początkowego przykładowego rozwiązania,
2) Wewnątrz pętli wykonywane są obliczenia, aż zostanie spełniony warunek stopu, zazwyczaj albo system dostatecznie się ochłodził, albo znaleziono wystarczające rozwiązanie.

a) Dwóch sąsiadów zostaje wybranych losowo,
b) Zostają oni zamienieni miejscami,
c) Algorytm podejmuje decyzję czy nowe rozwiązanie jest lepsze od poprzedniego.
d) Temperatura jest zmniejszana i system przechodzi do następnej iteracji.

Funkcja akceptująca nowe rozwiązania

Najpierw sprawdzane jest czy nowe rozwiązanie jest lepsze od poprzedniego, jeżeli tak, to jest akceptowane. Jeżeli nie, określane jest, przy pomocy liczby eulera podniesionej do określonej potęgi, jak bardzo "gorsze" jest nowe rozwiązanie i jak wysoka jest "temperatura" systemu. Przy wyższych temperaturach system jest skłonny zaakceptować gorsze rozwiązania.

```
function acceptanceProbability(energy, newEnergy, temperature){
    if(newEnergy < energy){
        return 1.0;
    }
    return Math.exp((energy - newEnergy) / temperature);
}
```

Rys.2 Funkcja akceptująca nowe rozwiązania.

Im mniejsza zmiana energii (jakości rozwiązania) i wyższa temperatura, tym bardziej prawdopodobne jest, że algorytm zaakceptuje nowe rozwiązanie.

Algorytm genetyczny oparty jest na procesie naturalnej selekcji. Jego sposób działania przebiega następująco:

1) inicjalizacja losowej populacji bazowej,
2) każdy członek populacji jest oceniany (selekcja) i określana jest wartość jego funkcji przystosowania, najlepiej przystosowane osobniki biorą udział w procesie reprodukcji,
3) genotypy wybranych osobników są poddawane krzyżowaniu i mutacji
    a) krzyżowanie polega na złączeniu genotypów rodziców
    b) mutacja wprowadza do genotypu drobne losowe zmiany
4) Rodzi się kolejne pokolenie, najsilniejsze osobniki są powielane, a najsłabsze usuwane. Jeżeli nie znaleziono dostatecznie dobrego rozwiązania, algorytm

powraca do kroku drugiego. W przeciwnym wypadku wybierany jest najlepszy osobnik z populacji.

W przypadku rozpatrywanego problemu optymalizacji funkcję oceny będzie stanowić suma odległości pomiędzy miastami. Ze względu na specyfikę rozpatrywanego zagadnienia, operatory ewolucyjne musiały zostać odpowiednio dostosowane, tak aby nie generować niepoprawnych rozwiązań.

Mutacja

```
this.mutate = function(tour){
    // Loop through tour cities
    for (var i = 0; i < tour.tourSize(); i++) {
        // Apply mutation rate
        if(Math.random() < this.mutationRate){
            var tourSize = tour.tourSize() - 1;
            // Get a second random position in the tour
            var tourPos = parseInt( tourSize * Math.random() );
            // Get the cities at target position in tour
            var city1 = tour.getCity(i);
            var city2 = tour.getCity(tourPos);
            // Swap them around
            tour.setCity(tourPos, city1);
            tour.setCity(i, city2);
        }
    }
```

Rys. 2 Metoda odpowiadająca za mutację.

Ta metoda ma za zadanie jedynie wprowadzić losowe zmiany, nie może dodawać żadnych elementów ani ich usuwać. Jej realizacja polega na tym, że dla każdego miasta z podanej trasy losowane jest drugie miasto. Następnie te miasta są zamieniane miejscami.

Krzyżowanie

```
this.crossover = function(parent1, parent2){
    var child = new Tour(initialiseTour(amount));
    var parent1TourSize = parent1.tourSize() - 1;
     // Get start and end sub tour positions for parent1's tour
    var startPos = parseInt(Math.random() * parent1TourSize);
    var endPos = parseInt(Math.random() * parent1TourSize);
    // Loop and add the sub tour from parent1 to our child
    for (var i = 0; i < child.tourSize(); i++) {
         // If our start position is less than the end position
         if( (startPos < endPos)  && (i > startPos) && (i < endPos) ){
            child.setCity(i, parent1.getCity(i));
         }// If our start position is larger
         else if(startPos > endPos){
            if(!(i < startPos && i > endPos)){
                child.setCity(i,parent1.getCity(i));
            }
         }
    }
    // Loop through parent2's city tour
     for (var i = 0; i < parent2.tourSize(); i++) {
        // If child doesn't have the city add it
        if( !child.containsCity(parent2.getCity(i)) ){
            // Loop to find a spare position in the child's tour
            for (var j = 0; j < child.tourSize(); j++) {
                // Spare position found, add city
                if(child.getCity(j) === null){
                    child.setCity(j, parent2.getCity(i))
                    break;
                }
            }

        }
    }

    return child;
```

Rys. 3 Metoda odpowiadająca za krzyżowanie.

Do przeprowadzenia tej operacji zostało wykorzystane **krzyżowanie z porządkowaniem** (ang. order crossover - OX). Wybierany jest podzbiór miast od pierwszego rodzica i dodawany do trasy dziecka, następnie brakujące wartości, które jeszcze nie występują na trasie, są uzupełniane od drugiego rodzica.

Kryterium zatrzymania

Jako kryterium zatrzymania algorytmu wykorzystane zostało **kryterium maksymalnego kosztu**. Wartością tego parametru jest maksymalna liczba generacji

algorytmu, a więc w momencie gdy maksymalny koszt zostanie przekroczony, algorytm zakończy swoje działanie.

Wybrane metody selekcji

Wybraną metodą reprodukcji jest **reprodukcja turniejowa**. Polega ona na dwustopniowej selekcji kandydatów do reprodukcji, wybierana jest określona liczba osobników z populacji bazowej Pt - tworząca populacje turniejową. Następnie odbywa się turniej między osobnikami z nowo utworzonej populacji. Zwycięzcy są kopiowani do populacji tymczasowej. Spośród nich wybierany jest najlepszy osobnik do reprodukcji. W przedstawionym problemie została ona zaimplementowana w następujący sposób:

1. ustalenie liczebności turnieju np. k,
2. wylosowanie k osobników,
3. spośród wylosowanych zostanie reprodukowany najlepszy osobnik.

Wybraną metodą sukcesji jest **sukcesja elitarna**, która gwarantuje przeżycie najlepszego osobnika z populacji poprzez odpowiedni wybór osobników z Pt do P(t+1)

WYNIKI

Algorytm symulowanego wyżarzania został uruchomiony z następującymi parametrami :

Cities	20
Initial Temperature	1000
Absolute Zero	.0001
Cooling Rate	0.003
Maximum iterations	100

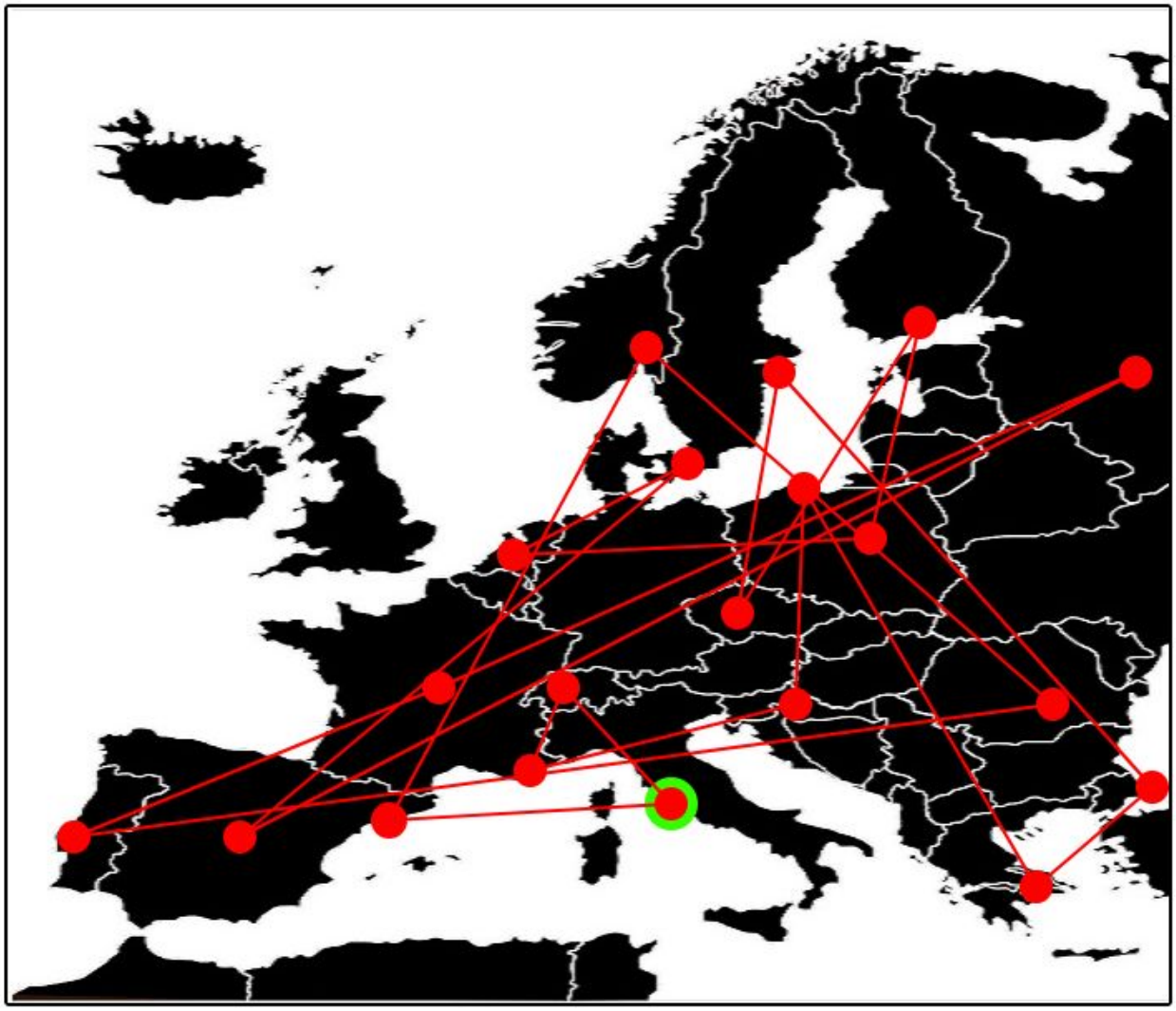
Rys. 4. Parametry algorytmu

Po wykonaniu niezbędnych obliczeń wyniki prezentują się następująco:

Tabela 2. Wynik działania algorytmu.

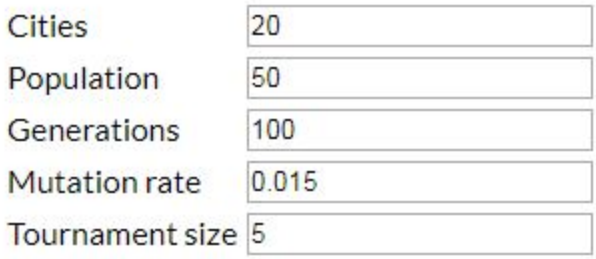
Optymalna długość trasy: 33734 km.
Czas wykonywania obliczeń: ok. 4.9 milisekund.
Trasa: Rzym --> Berno --> Monako --> Zagrzeb --> Gdynia --> Ateny --> Stambuł --> Sztokholm --> Praga --> Helsinki --> Warszawa --> Amsterdam --> Kopenhaga --> Madryt --> Moskwa --> Paryż --> Lizbona --> Bukareszt --> Oslo --> Barcelona --> Rzym

Poniżej została przedstawiona graficzna reprezentacja optymalnej trasy, zielona obramówka wokół jednego z miast oznacza, że jest to miasto, od którego trasa się zaczyna.



Rys. 5. Reprezentacja graficzna znalezionej trasy.

Algorytm genetyczny został uruchomiony z następującymi parametrami:



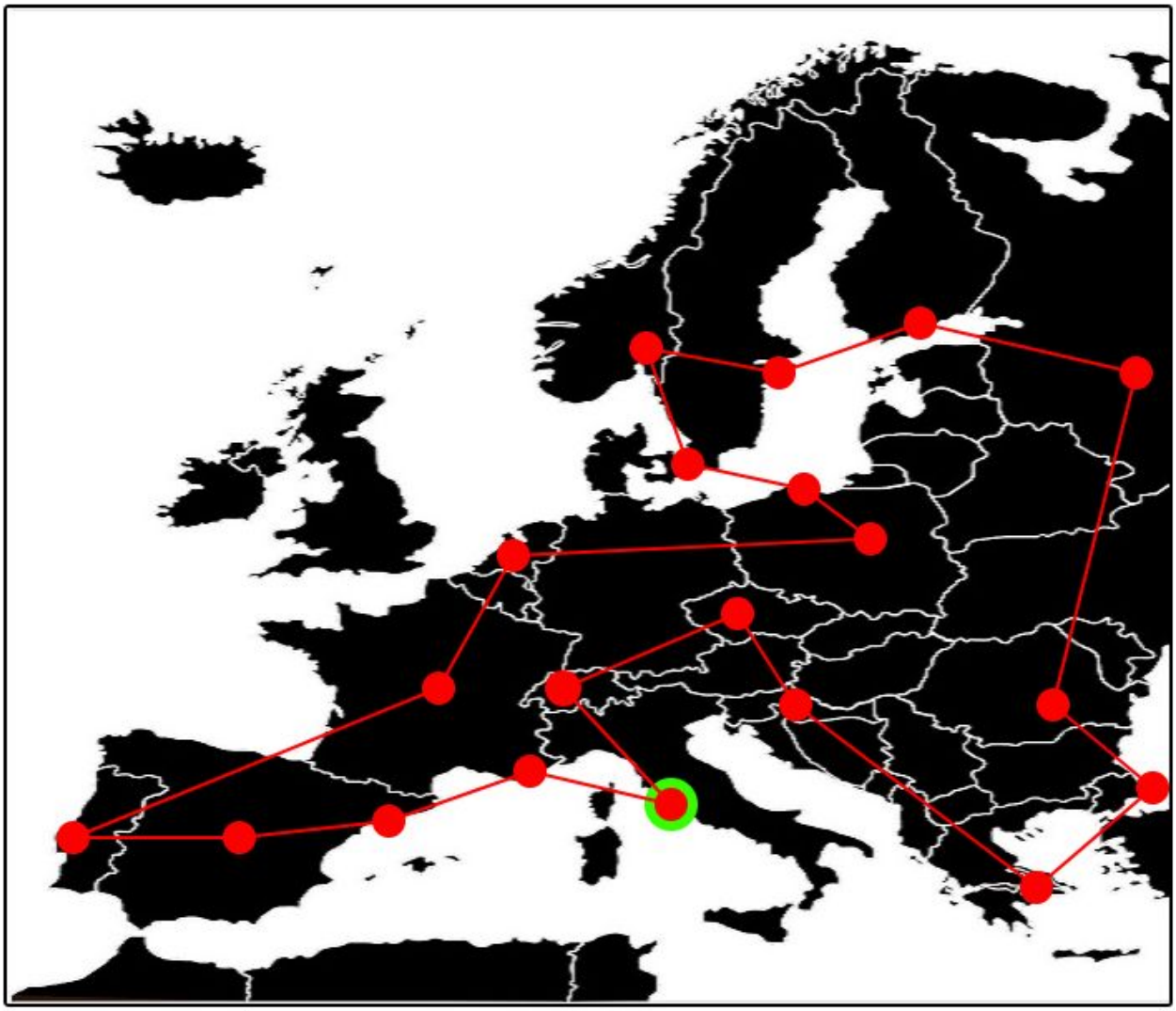
Rys. 6. Parametry algorytmu

Po wykonaniu niezbędnych obliczeń wyniki prezentują się następująco:

Tabela 3. Wynik działania algorytmu.

Optymalna długość trasy: 16998 km.
Czas wykonywania obliczeń: ok. 55.7 milisekund.
Trasa: Rzym --> Monako --> Barcelona --> Madryt --> Lizbona --> Paryż --> Amsterdam --> Warszawa --> Gdynia --> Kopenhaga --> Oslo --> Sztokholm --> Helsinki --> Moskwa --> Bukareszt --> Stambuł --> Ateny --> Zagrzeb --> Praga --> Berno --> Rzym

Poniżej została przedstawiona graficzna reprezentacja optymalnej trasy, zielona obramówka wokół jednego z miast oznacza, że jest to miasto, od którego trasa się zaczyna.



Rys. 7 Reprezentacja graficzna znalezionej trasy.

WNIOSKI

Z przeprowadzonych badań wynika, że lepszy wynik dało zastosowanie algorytmu genetycznego. Wygenerowana przez niego trasa jest niemal o połowę krótsza i wynosi 16998 km, mimo iż znalezienie rozwiązania zajęło więcej czasu - ok. 55.7 milisekund. Algorytm symulowanego wyżarzania wygenerował trasę nieefektywną do odbycia w rzeczywistości, mimo iż rozpoczyna się w tym samym mieście (Rzym). Jej długość wyniosła 33734 km, a wylosowanie tej trasy zajęło ok. 4.9 milisekund.

Reasumując, algorytm genetyczny daje efektywniejsze rozwiązania problemu badawczego jakim jest problem komiwojażera.