

# Sprawozdanie końcowe

Bartosz Młynarski 411947

09.01.2024 r. – godz. 04:34

## 1. Podział pracy.

Projekt został wykonany całkowicie przez 1 osobę, więc nie wpisuję szczegółów procentowych dotyczących podziału pracy, ponieważ każda wartość przypisana do mnie byłaby równa 100%.

## 2. Model zagadnienia

Zagadnienie polegało na problemie związanym z dostarczaniem przesyłek po całej Polsce transportem kolejowym. Mamy zdefiniowane miasta oraz sieć torów kolejowych w całym kraju, miasto w którym znajduje się centrum naszej firmy oraz informacje dotyczące tego ile przesyłek i skąd dokąd mamy je przewieźć. Problem polega na tym, że musimy te przesyłki rozdzielić na wybraną liczbę pociągów, które muszą wybrać optymalną trasę, a by dostarczyć swoje przesyłki. Dodatkowo trzeba w optymalny sposób skonstruować każdy pociąg, jaką będzie miał on lokomotywę oraz jakie i ile wagonów będzie on miał doczepionych.

Ze względu że zadanie opiera się o to, że wszystkie przesyłki musimy dostarczyć, które zatwierdziliśmy, to zadanie polegające na maksymalizacji zysku z transportu przechodzi do minimalizacji zużytego paliwa do przewozu wszystkich przesyłek. Dla uproszczenia założyłem, że tory kolejowe łączące miasta są ustawione w linii prostej, co znacząco skraca obliczenia a wynik działania algorytmu będzie bardzo zbliżony do rzeczywistego.

Żeby rozwiązać problem należy podać miasto, w którym znajduje się centrum firmy, parametry lokomotyw oraz wagonów, które należy wykorzystać i dane dotyczące przesyłek, skąd, dokąd i w jakiej ilości.

Dla danego zagadnienia tak prezentuje się funkcja celu:

$$F = \sum_{i=0}^{i=n} HF_i \rightarrow \min$$

Gdzie n to liczba pociągów, a  $HF_i$  to zużyte paliwo potrzebne na przewóz przesyłek przez i-ty pociąg i liczy się je tak:

$$HF_i = (SL[DL[i]] [0] + \sum_{j=0}^{j=length(SW)-1} SW_{j,0} * DW_{i,j}) * \frac{HL_i}{100}$$

Gdzie  $SL[x,0]$  to struktura zawierająca informacje o lokomotywach, gdzie pod zmienną „x” znajduje się indeks lokomotywy, a pod zmienną „0” spalanie oleju napędowego danej lokomotywy na 100 km,  $DL[i]$  to zmienna decyzyjna przechowująca informacje o typie lokomotywy użytej w i-tym pociągu,  $SW[x,0]$  to struktura zawierająca informacje o wagonach, gdzie pod zmienną „x” znajduje się indeks wagonu, a pod zmienną „0” znajduje się informacja o dodatkowym spalaniu oleju napędowego na godzinę, w wypadku gdy dołączymy ten wagon do pociągu,  $DW[x,y]$  to zmienna decyzyjna przechowująca informacje o wagonach użytych w pociągu, pod zmienną „x” znajduje się informacja o aktualnym indeksie pociągu, a pod zmienną „y” kryje się informacja o typie wagon, sama wartość  $DW[x,y]$  zwraca liczbę wagonów. Zmienna  $HL[i]$  to zmienna pomocnicza przechowująca informacje o pokonanym dystansie przez i-ty pociąg, którą liczy się sumując wszystkie trasy.

Powyższe zagadnienie ma postać rozwiązania w postaci listy, która jako pierwszy parametr przechowuje tensor 3 wymiarowy zmiennej decyzyjnej DT, która przechowuje informacje o trasie przez każdy pociąg, jako drugi parametr przechowuje listę zmiennej decyzyjnej DL, która przechowuje informacje o typie lokomotywy dla każdego pociągu, jako trzeci parametr zwraca tensor 3 wymiarowy zmiennej DW przechowującej informacje o wagonach dołączonych do każdego pociągu, a jako czwarty parametr algorytm zwraca tensor 3 wymiarowy przechowujący informacje o przesyłkach do przeniesienia przez każdy z pociągów.

Zmienne decyzyjne algorytmu:

- DT – przechowuje informacje o trasach każdego z pociągów
- DL – przechowuje informacje o lokomotywach każdego z pociągów
- DW – przechowuje informacje o wagonach każdego z pociągów
- DP – przechowuje informacje o przesyłkach do przewiezienia każdego z pociągów

Zmienne pomocnicze algorytmu:

- HF – przechowuje informacje o zużytym paliwie przez każdy pociąg
- HL – przechowuje informacje o dystansie pokonanym przez każdy pociąg
- HP – przechowuje informacje o miastach, które musi odwiedzić każdy pociąg

Struktury algorytmu:

- SC – przechowuje współrzędne każdego miasta na mapie Polski
- SL – przechowuje informacje o wszystkich lokomotywach
- SW – przechowuje informacje o wszystkich wagonach
- SG – przechowuje informacje o grafie zawierającym informacje o każdym mieście
- SP – przechowuje informacje o wszystkich miastach jakie muszą odwiedzić pociągi

### 3. Algorytm

Zaimplementowanym algorytmem, który rozwiązuje dane zagadnienie jest algorytm genetyczny. Pseudokod wykonanego przeze mnie algorytmu prezentuje się następująco:

```
Def AG(size_population, number_of_epochs, mutate_population, mutate_power, start_city):
```

```
    population = generate_first_population(size_population, start_city)
```

```
    mutate_population = mutate_population / 100 * size_population
```

```
    mutate_power = mutate_power / 100 * n * 10
```

```
    list_min_f = []
```

```
    list_max_f = []
```

```
    list_mean_f = []
```

```
    for pech in range(number_of_epochs):
```

```
        population = mutat_sort(population)
```

```
        population = population[:mutate_population]
```

```
        population_to_mutate = copy(population)
```

```
        while length(population) < size_population:
```

```
            population.append(mutate_population, mutate_power)
```

```
        for result in population:
```

```
            result.solve_goal_function()
```

```
        goal_functions = [goal_functions every result in population]
```

```
        list_min_f.append(min(goal_functions))
```

```
        list_max_f.append(max(goal_functions))
```

```
        list_mean_f.append(mean(goal_functions))
```

```
    best_id = goal_functions.index(min(goal_functions))
```

```
    best_result = population(best_id)
```

```
    return best_result
```

Algorytm w pierwszej kolejności generuje startową populację w sposób całkowicie losowy. Początkowo przepisuje wszystkie informacje o przesyłkach do przewiezienia, losowo w miarę równo rozdziela przesyłki do przewiezienia na wszystkie pociągi, po czym ustala on początkowe trasy wszystkich pociągów, łączy losową lokomotywę, liczy przesyłki i losuje wagony w taki sposób, aby każdy pociąg w sytuacji maksimum liczby przesyłek znajdującej się w wagonach pociąg mógł pomieścić wszystkie przesyłki.

Następnie algorytm przelicza na wartości przyjmowane przez algorytm wartości `utates_population` oraz `utate_power`, które kolejno oznaczają procent zostawianej poprzedniej populacji oraz moc mutowania kolejnych pokoleń.

W kolejnym kroku tworzymy kontenery przechowujące informacje o każdej wygenerowanej populacji, tj. o minimalnej wartości funkcji celu osiągniętej w danym pokoleniu, maksymalnej wartości funkcji celu osiągniętej w danym pokoleniu oraz średniej funkcji celu osiągniętej w danym pokoleniu.

W dalszym etapie zaczynamy generować wszystkie epoki, sortujemy nasze pokolenia przez wartość funkcji celu od najmniejszej do największej za pomocą algorytmu Quick Sort, w celu wybrania tylko podanego przez nas procenta najlepszej populacji.

Następnie zaczynamy uzupełniać naszą populację nowo wygenerowaną populacją poprzez mutowanie kolejnych rozwiązań do momentu uzyskania liczby populacji o wielkości populacji startowej.

Mutowanie polega na wybraniu losowo rozwiązania spośród najlepszych pozostałych rozwiązań z poprzedniego pokolenia oraz losowaniu mutacji, tj. wybiera losowe przesyłki z losowego pociągu i przenosi je na inny losowy pociąg, losuje tablicę przechowującą kolejność odwiedzonych miast losowego pociągu, losuje lokomotywę oraz wagony losowych pociągów.

Po wygenerowaniu nowej populacji przechodzimy do liczenia wartości funkcji celu każdego pokolenia oraz łączy statystyk wygenerowanego pokolenia, minimalnej wartości funkcji celu, maksymalnej wartości funkcji celu oraz średniej wygenerowanej funkcji celu uzyskanej w danym pokoleniu.

Po przejściu przez wszystkie epoki algorytm zwraca najlepsze rozwiązanie znalezione podczas wykonywania kodu.

Cała implementacja algorytmu znajduje się w napisanych przeze mnie plikach w języku Python:

- algorithm.py
- first\_population.py
- functions.py
- model.py

## 4. Aplikacja

Aplikacja do programu została napisana w języku python za pomocą biblioteki wxPython, a cała aplikacja znajduje się w pliku o nazwie program.py.

Po odpaleniu programu mamy przed sobą mapę Polski z wyświetlonymi miastami oraz odcinkami reprezentującymi tory między nimi. Dodatkowo po bokach znajdują się paski z przyciskami oraz polami do sterowania funkcją.

Poniżej znajduje się opis funkcjonalności aplikacji:

- Środkowe okno – obszar w aplikacji wyświetlający wybrany przez nas element
- Prawy pasek boczny – pasek roboczy służący do sterowania aplikacją zawierający następujące elementy:
  - Przycisk „Close” – zamyka aplikację
  - Przycisk „Packages” – przycisk przenoszący aplikację w tryb ustawiania parametrów przesyłek. Po wciśnięciu go na środkowym oknie ukazuje się nam tabela z przesyłkami, którą można edytować. Dodatkowo w tym środkowym oknie otrzymujemy dodatkowy pasek roboczy służący do sterowania tabelą:
    - Przycisk „Add row” – dodaje wiersz do tabeli
    - Przycisk „Delete rows” – usuwa z tabeli wiersze z brakującymi wartościami
    - Pole „Number of rows” – parametr opisujący liczbę wierszy do losowego generowania danych
    - Pole „Min packages” – parametr opisujący minimalną liczbę przesyłek wylosowanych do losowego generowania danych
    - Pole „Max packages” – parametr opisujący maksymalną liczbę przesyłek wylosowanych do losowego generowania danych
    - Przycisk „Random Packages” – przycisk generujący losowe parametry przesyłek w celu chęci przetestowania algorytmu
  - Przycisk „Trains” – przycisk przenoszący aplikację w tryb ustawiania parametrów lokomotyw oraz wagonów. Po wciśnięciu go na środkowym oknie ukazuje się nam tabela z lokomotywami oraz wagonami, którą można edytować. Dodatkowo w tym środkowym oknie otrzymujemy dodatkowy pasek roboczy służący do sterowania tabelą:

- Przyciski „Add row” – przyciski dodające wiersze w danych tabelach
- Przyciski „Delete rows” – przyciski usuwające wiersze z brakującymi wartościami
- Pole „Size population” – parametr opisujący wielkość populacji w algorytmie
- Pole „Number of epochs” – parametr opisujący liczbę epok w algorytmie
- Pole „Previous population” – parametr opisujący % pozostawianej poprzedniej populacji
- Pole „Mutate power” – parametr opisujący wielkość mutowania populacji
- Pole „Number of trains” – parametr opisujący liczbę pociągów
- Pole „Start city” – parametr wskazujący siedzibę firmy
- Napis nad przyciskiem start – pokazuje informacje na temat stanu wykonywania się aktualnego algorytmu oraz szacowany czas do końca wykonywania algorytmu
- Przycisk „Start AG” – przycisk rozpoczynający działanie algorytmu
- Przycisk „Stop AG” – przycisk przerywający działanie algorytmu oraz zwracający najlepsze dotychczasowe rozwiązanie (przerywanie dzieje się na początku epoki dlatego algorytm nie zatrzyma się od razu, tylko do przejścia do początku następnej epoki)
- Lewy pasek boczny – pasek roboczy służący do sterowania aplikacji zawierający następujące elementy:
  - Przycisk „Map” – przycisk przełączający tryb środkowego okna w pokazywanie mapy Polski
  - Przycisk „Goal chart” – przycisk przełączający tryb środkowego okna w pokazywanie statystyk funkcji celu każdego wygenerowanego pokolenia w ostatnio wykonanym algorytmie.
  - Przycisk „The best result” – przycisk przełączający tryb środkowego okna w pokazywanie najlepszego rozwiązania w postaci tekstowej
  - Pole „Train numer” – parametr opisujący pociąg w danym rozwiązaniu
  - Przycisk „Show track” – przycisk przełączający tryb środkowego okna w pokazywanie graficzne na mapie Polski trasy wybranego przez nas pociągu

## 5. Testy

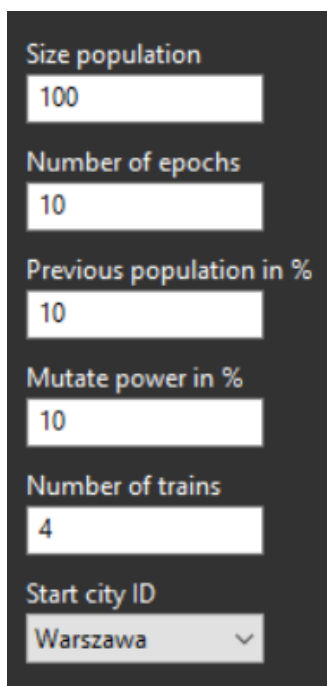
### Test nr 1.

W tym teście wygeneruję małą liczbę wierszy przesyłek (10). Przesyłki wyglądają następująco:

	From ID	From city	To ID	To city	How many
1	45.0	Toruń	25.0	Leszno	331.0
2	44.0	Tarnów	34.0	Rybnik	614.0
3	35.0	Rzeszów	45.0	Toruń	542.0
4	18.0	Katowice	43.0	Słupsk	315.0
5	45.0	Toruń	38.0	Stalowa Wola	730.0
6	7.0	Chojnice	23.0	Kraków	236.0
7	42.0	Szczecinek	39.0	Stargard Szcz	573.0
8	27.0	Olsztyn	9.0	Częstochowa	788.0
9	44.0	Tarnów	10.0	Elbląg	775.0
10	8.0	Ciechanów	10.0	Elbląg	361.0

Rys 1. – tabela z przesyłkami do testu nr 1

W tym wykonaniu przyjmę następujące parametry:



Size population  
100

Number of epochs  
10

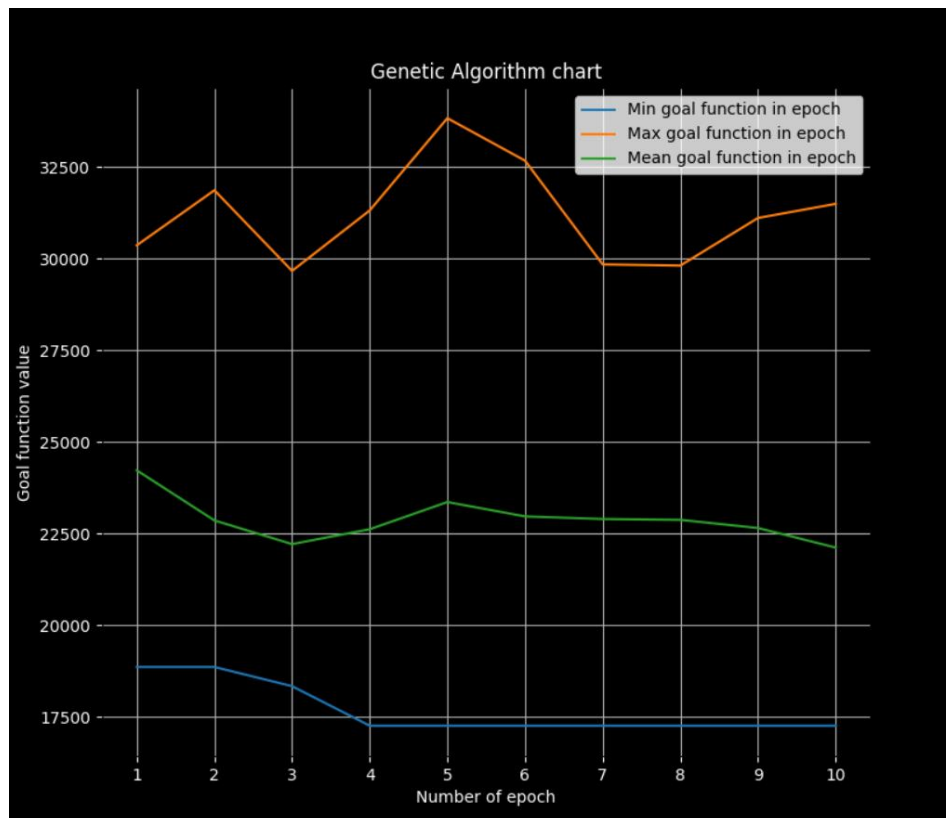
Previous population in %  
10

Mutate power in %  
10

Number of trains  
4

Start city ID  
Warszawa

Rys 2. – tabela z parametrami AG do testu nr 1



Rys 3. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 1



Rys 4. – mapa Polski pokazująca trasę przykładowego pociągu



Jak widać w zamieszczonych rysunkach dla danych parametrów algorytm szybko znalazł potencjalnie dobre rozwiązanie dla danego zagadnienia. Wartość funkcji celu oscyluje w okolicach wartości 17500, a mapa pokazuje w miarę prostą trasę losowo wybranego pociągu.

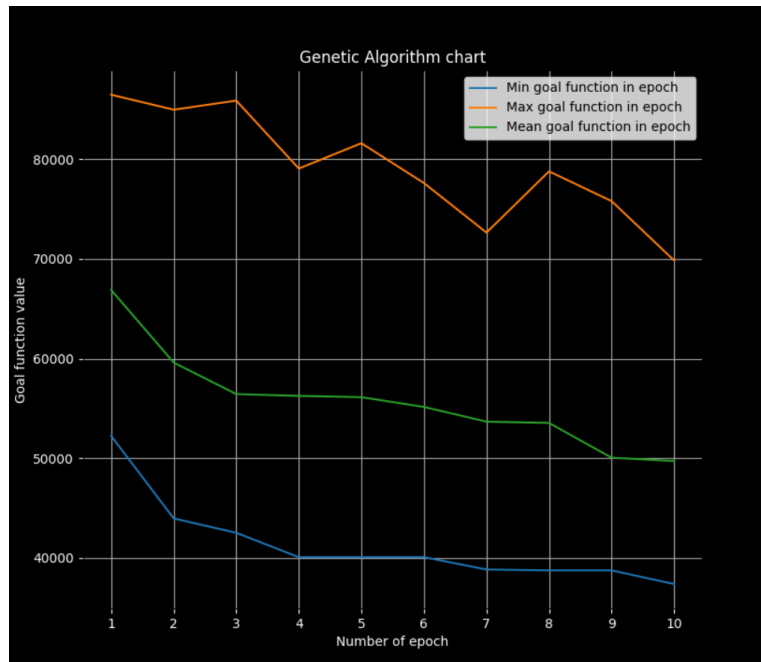
## Test nr 2.

W tym teście wygeneruję trochę większą tabelę z przesyłkami (20 wierszy).

	From ID	From city	To ID	To city	How many
1	20.0	Konin	3.0	Białystok	562.0
2	45.0	Toruń	10.0	Elbląg	724.0
3	15.0	Gorzów Wielki	28.0	Opole	591.0
4	25.0	Leszno	2.0	Biała Podlaska	615.0
5	52.0	Łomża	12.0	Gdańsk	460.0
6	22.0	Kołobrzeg	10.0	Elbląg	785.0
7	38.0	Stalowa Wola	20.0	Konin	738.0
8	27.0	Olsztyn	31.0	Przemyśl	701.0
9	46.0	Warszawa	52.0	Łomża	570.0
10	52.0	Łomża	6.0	Chełm	797.0
11	21.0	Koszalin	18.0	Katowice	498.0
12	36.0	Sanok	32.0	Płock	495.0
13	40.0	Suwałki	26.0	Lublin	220.0
14	24.0	Legnica	27.0	Olsztyn	712.0
15	27.0	Olsztyn	13.0	Gdynia	381.0
16	19.0	Kielce	31.0	Przemyśl	477.0
17	34.0	Rybnik	16.0	Grudziądz	258.0
18	43.0	Słupsk	24.0	Legnica	562.0
19	39.0	Stargard Szczeciński	16.0	Grudziądz	533.0
20	42.0	Szczecinek	11.0	Elk	586.0

Rys 5. – Tabela z przesyłkami do testu nr 2.

W tym teście skorzystam z tych samych danych co w teście nr 1.



Rys 6. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 2



Rys 7. – mapa Polski pokazująca trasę przykładowego pociągu

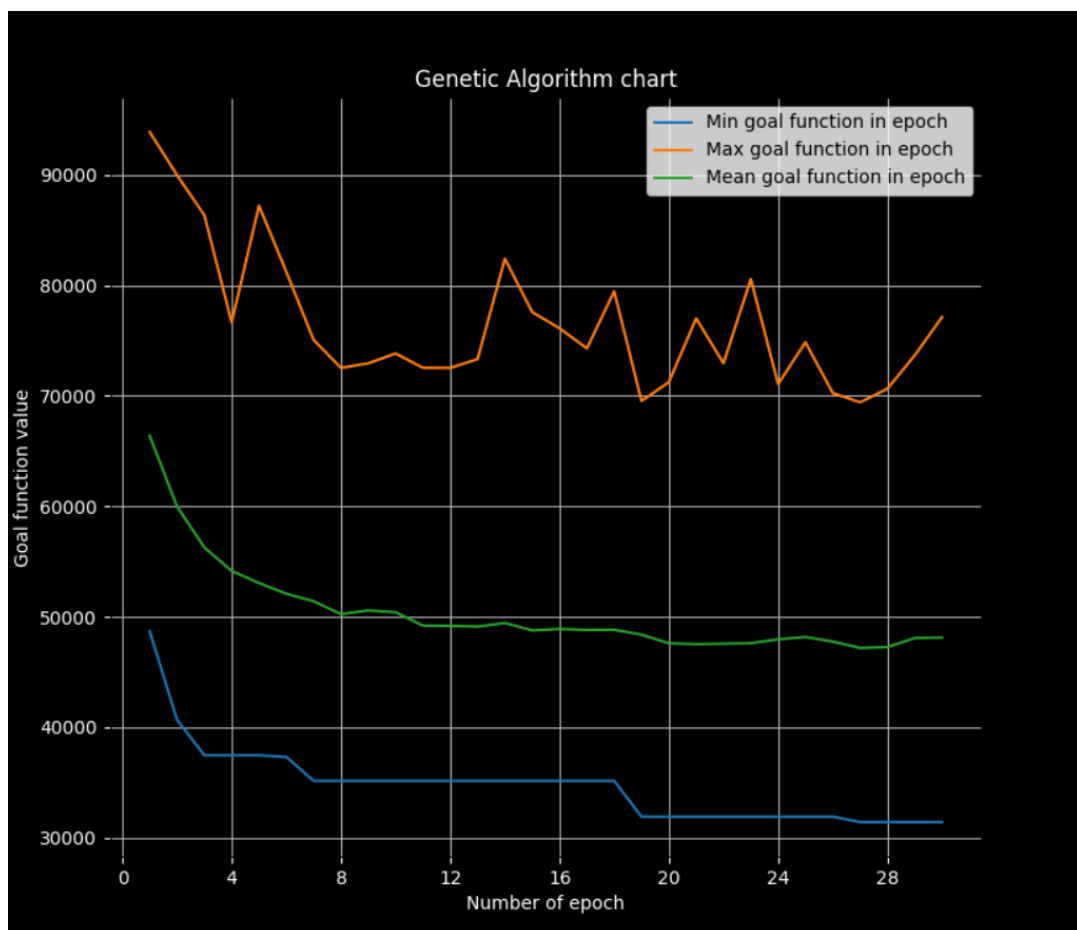
W tym teście po zwiększeniu liczby wierszy z przesyłkami algorytm nie znalazł najlepszego rozwiązania, funkcja celu jest bardzo wysoka i oscyluje w wartościach 38000, a trasa losowo wybranego pociągu jest bardzo chaotyczna i prezentuje skomplikowaną trasę.

### Test nr 3.

W tym teście skorzystam z tej samej tabeli przesyłek co w teście nr 2, jedynie dodaje parametry pozwalające na dokładniejsze obliczenia:

Size population	<input type="text" value="500"/>
Number of epochs	<input type="text" value="30"/>
Previous population in %	<input type="text" value="10"/>
Mutate power in %	<input type="text" value="10"/>
Number of trains	<input type="text" value="4"/>
Start city ID	<input type="text" value="Warszawa"/>

Rys 8. – tabela z parametrami AG do testu nr 3



Rys 9. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 3



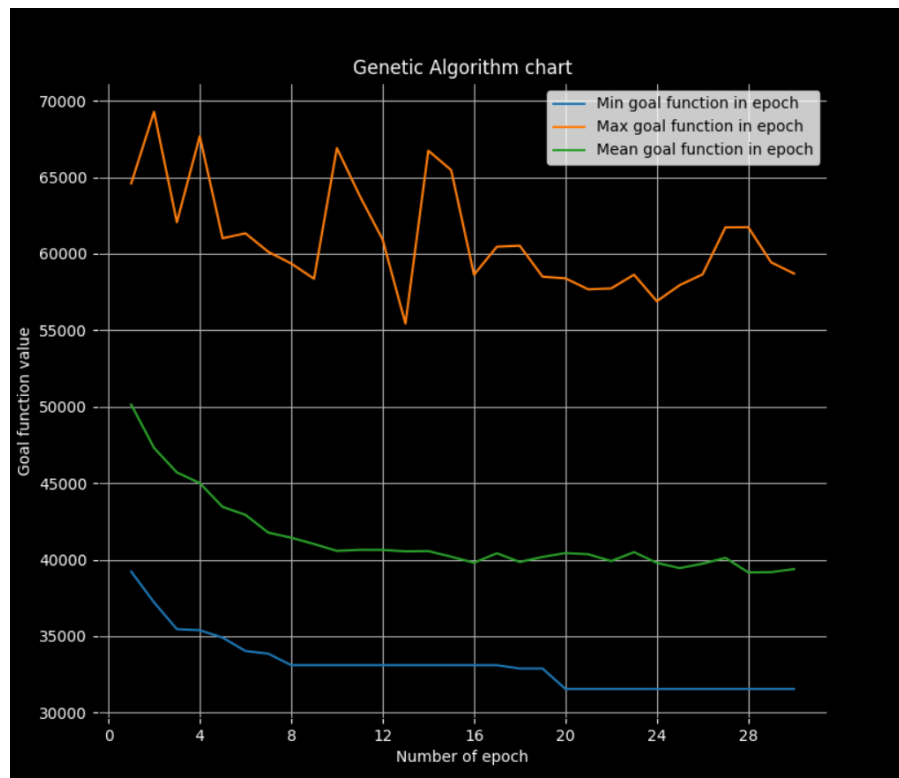
Rys 10. – mapa Polski pokazująca trasę przykładowego pociągu

Dzięki zastosowaniu parametrów pozwalających na dokładniejsze obliczenia uzyskaliśmy wynik lepszy od poprzedniego. Wartość funkcji celu oscyluje w wartościach 31000, za to mapa losowo wybranego pociągu dalej jest chaotyczna i wydaje w miarę skomplikowaną trasę, ale zdecydowanie mniej niż w poprzednim rozwiązaniu. Pomimo tego przystąpiłem do testu nr 4, w którym zmieniłem jeden parametr i spróbowałem pobić wynik.

#### Test nr 4.

W tym teście skorzystam z tych samych danych i parametrów co w teście nr 3, jedynie zwiększę liczbę pociągów z 4 do 6.





Rys 11. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 4



Rys 12. – mapa Polski pokazująca trasę przykładowego pociągu

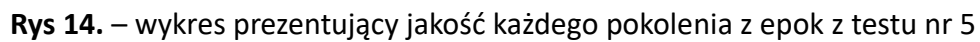
W tym teście uzyskałem wynik funkcji celu minimalnie większy niż w poprzednim teście. Wynika to z faktu iż po dodaniu 2 pociągów automatycznie mamy 2 lokomotywy więcej, które same z siebie spalają dodatkowe paliwo. Mimo to wartość funkcji celu oscyluje w wartościach 33000, więc dalej jest to porównywalna wartość. Mapa trasy pojedynczego pociągu jest prosta i pokazuje umiarkowanie skomplikowaną trasę. Stało się tak dzięki mniejszym obciążeniu przesyłek na każdy pociąg, co również przekłada się na szybsze wykonanie całego zlecenia.

## Test nr 5.

W tym teście użyłem nowej tabeli z przesyłkami tym razem z 30 wierszami, ale tych samych parametrów do algorytmu co w teście nr 4.

	From ID	From city	To ID	To city	How many
1	13.0	Gdynia	22.0	Kołobrzeg	700.0
2	38.0	Stalowa Wola	31.0	Przemyśl	678.0
3	33.0	Radom	11.0	Elk	367.0
4	9.0	Częstochowa	33.0	Radom	334.0
5	15.0	Gorzów Wielki	31.0	Przemyśl	252.0
6	53.0	Łódź	15.0	Gorzów Wielki	227.0
7	3.0	Białystok	2.0	Biała Podlaska	424.0
8	48.0	Wrocław	25.0	Leszno	789.0
9	30.0	Poznań	39.0	Stargard Szczeciński	384.0
10	22.0	Kołobrzeg	24.0	Legnica	226.0
11	25.0	Leszno	22.0	Kołobrzeg	535.0
12	50.0	Zamość	53.0	Łódź	369.0
13	46.0	Warszawa	42.0	Szczecinek	478.0
14	31.0	Przemyśl	30.0	Poznań	492.0
15	38.0	Stalowa Wola	2.0	Biała Podlaska	213.0
16	54.0	Żyrardów	43.0	Słupsk	218.0
17	1.0	Bełchatów	14.0	Gliwice	506.0
18	43.0	Słupsk	51.0	Zielona Góra	526.0
19	33.0	Radom	49.0	Włocławek	786.0
20	30.0	Poznań	42.0	Szczecinek	391.0
21	7.0	Chojnice	32.0	Płock	259.0
22	41.0	Szczecin	36.0	Sanok	279.0
23	22.0	Kołobrzeg	37.0	Siedlce	709.0
24	2.0	Biała Podlaska	35.0	Rzeszów	632.0
25	18.0	Katowice	25.0	Leszno	303.0
26	53.0	Łódź	37.0	Siedlce	350.0
27	44.0	Tarnów	18.0	Katowice	618.0
28	1.0	Bełchatów	9.0	Częstochowa	442.0
29	48.0	Wrocław	27.0	Olsztyn	258.0
30	34.0	Rybnik	18.0	Katowice	346.0

Rys 13. – Tabela z przesyłkami do testu nr 5.





W tym teście pomimo zwiększenia liczby przesyłek z 20 wierszy do 30 wierszy i pozostawieniu tych samych parametrów, to algorytm poradził sobie bardzo dobrze z poszukianiem rozwiązania. Pomimo 50% liczby miast potrzebnych do odwiedzenia i 6 pociągów na torach, funkcja celu oscyluje w wartościach 35500, co jest wynikiem bardzo dobrym dla tak wielu przesyłek. Sama mapa trasy losowego pociągu pokazuje dosyć prostą trasę, mało skomplikowaną, co tylko potwierdza, że algorytm wykonał tu dobrą robotę. Niestety algorytm liczył to rozwiązanie około 7 min, co jak na małe zaawansowanie tego zagadnienia, gdzie w profesjonalnych firmach liczba wierszy w tabeli przesyłek może wynosić kilkaset, jest łagodnie mówiąc średnim jeśli chodzi o optymalizację.

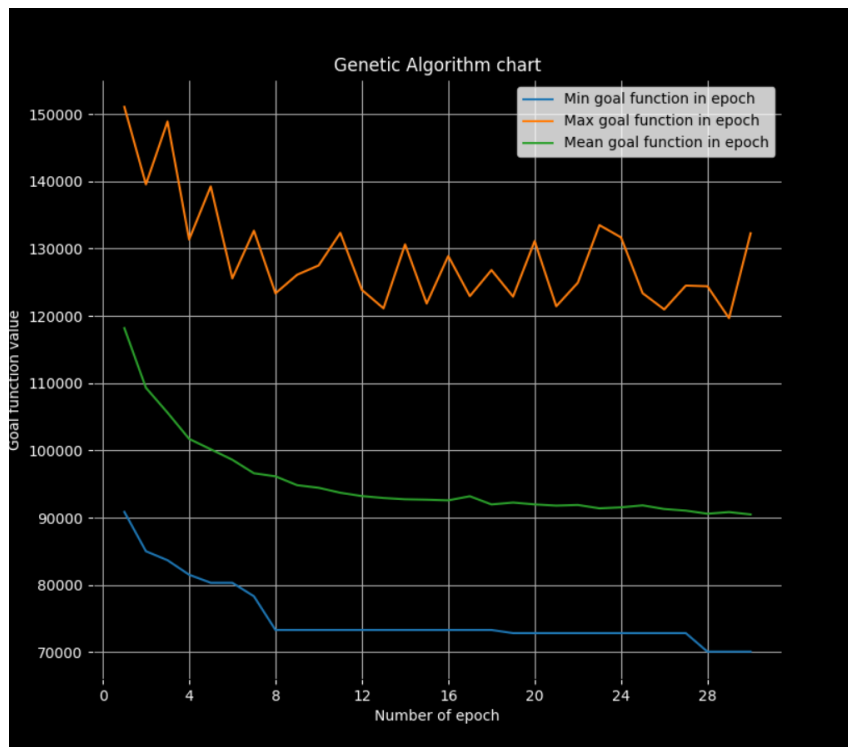
## Test nr 6.

W tym teście wygenerowałem nową tabelę z przesyłkami o długości 40 wierszy, a dane zachowałem te same co w poprzednim teście, poza parametrem opisującym liczbę pociągów (z 6 pociągów na 8 pociągów).

	From ID	From city	To ID	To city	How many
1	50.0	Zamość	35.0	Rzeszów	770.0
2	51.0	Zielona Góra	32.0	Płock	311.0
3	25.0	Leszno	4.0	Bielsko-Biała	533.0
4	36.0	Sanok	48.0	Wrocław	314.0
5	37.0	Siedlce	25.0	Leszno	731.0
6	38.0	Stalowa Wola	36.0	Sanok	374.0
7	3.0	Białystok	8.0	Ciechanów	773.0
8	26.0	Lublin	37.0	Siedlce	649.0
9	47.0	Wałbrzych	43.0	Słupsk	337.0
10	11.0	Elk	52.0	Łomża	455.0
11	29.0	Piła	50.0	Zamość	663.0
12	26.0	Lublin	47.0	Wałbrzych	434.0
13	20.0	Konin	3.0	Białystok	763.0
14	40.0	Suwałki	51.0	Zielona Góra	741.0
15	44.0	Tarnów	49.0	Wrocław	609.0
16	41.0	Szczecin	25.0	Leszno	234.0
17	16.0	Grudziądz	2.0	Biała Podlaska	250.0
18	51.0	Zielona Góra	53.0	Łódź	361.0
19	50.0	Zamość	4.0	Bielsko-Biała	690.0
20	17.0	Kalisz	36.0	Sanok	614.0
21	18.0	Katowice	10.0	Elbląg	671.0
22	11.0	Elk	22.0	Kołobrzeg	749.0
23	34.0	Rybnik	18.0	Katowice	235.0
24	53.0	Łódź	25.0	Leszno	601.0
25	42.0	Szczecinek	6.0	Chelm	292.0
26	48.0	Wrocław	12.0	Gdańsk	309.0
27	37.0	Siedlce	24.0	Legnica	794.0
28	8.0	Ciechanów	47.0	Wałbrzych	386.0
29	8.0	Ciechanów	31.0	Przemyśl	631.0
30	22.0	Kołobrzeg	6.0	Chelm	357.0
31	34.0	Rybnik	41.0	Szczecin	625.0
32	53.0	Łódź	49.0	Wrocław	616.0
33	40.0	Suwałki	34.0	Rybnik	236.0
34	8.0	Ciechanów	16.0	Grudziądz	642.0
35	46.0	Warszawa	6.0	Chelm	493.0
36	40.0	Suwałki	42.0	Szczecinek	343.0

Rys 16. – Tabela z przesyłkami do testu nr 6.





Rys 17. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 6

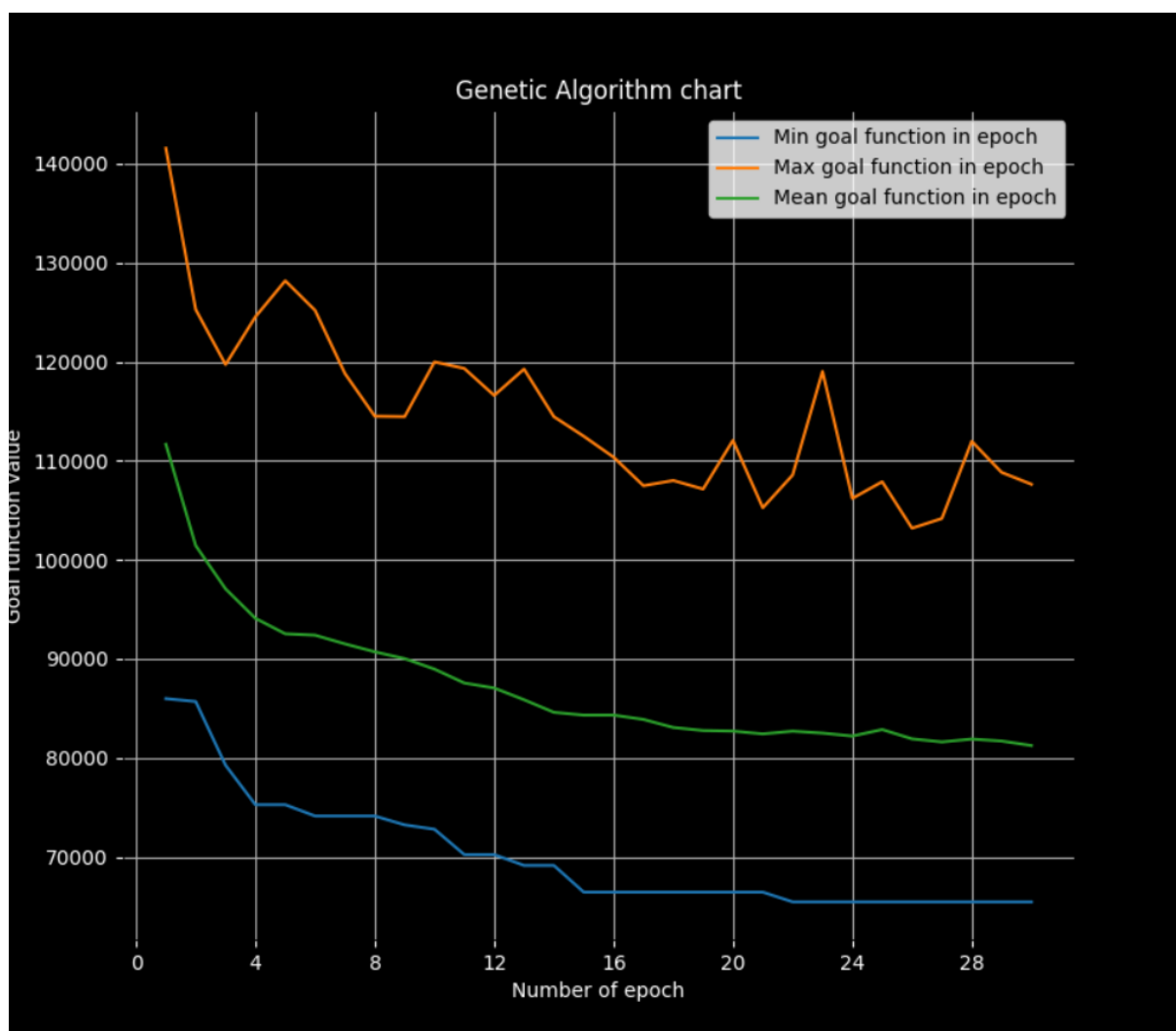


Rys 18. – mapa Polski pokazująca trasę przykładowego pociągu

W tym teście wyniki końcowe okazały się akceptowalne, jednak w stosunku do poprzedniego testu dołożyłem tylko 33% dodatkowych wierszy z przesyłkami zwiększając przy tym liczbę pociągów z 6 do 8, a mimo to wartość funkcji celu była aż 2-krotnie większa niż w teście nr 5, bowiem oscylowała ona w wartościach 70000. Mapa za to ukazuje umiarkowane skomplikowanie trasą co daje nam informacje o nienajgorszym rozplanowaniu jej, lecz można by było na pewno lepiej tą trasę rozplanować. Dla aktualnych parametrów algorytm liczył to zadanie już 12 min, więc stosunkowo długo, ale mimo wszystko zszedł on z rozwiązania losowego do pewnej niższej wartości.

## Test nr 7.

W tym teście zachowałem te same dane przesyłek i te same parametry, zmieniając tylko liczbę pociągów z 8 do 10, sprawdzając czy dzięki temu algorytm poradzi sobie lepiej z problemem.



Rys 19. – wykres prezentujący jakość każdego pokolenia z epok z testu nr 7



Rys 20. – mapa Polski pokazująca trasę przykładowego pociągu

W tym teście po zwiększeniu liczby pociągów z 8 do 10, algorytm znalazł lepsze rozwiązanie niż w teście nr 7. Nie dość że ilość zużytego paliwa łączna jest mniejsza, to dzięki temu że wszystkie przesyłki rozdzielają się na większą liczbę pociągów, to też szybciej zdecydowanie te pociągi obsługują te przesyłki, co przełoży się na szybsze wykonanie zlecenia. Wartość funkcji celu w tym teście oscyluje w wartościach 67000, a mapka pokazuje prostsze trasy niż w teście poprzednim.

## 6. Podsumowanie

Aplikacja i zaimplementowany algorytm tworzą bazę do bardzo dobrego projektu, jednak jest on zbyt niedopracowany w kontekście użytku dla realistycznych problemów. Jest to związane z małymi niedociągnięciami w interfejsie, nieodpowiednią optymalizacją, ale jest to również połączone z dużym złożeniem problemu, który ma dużo zmiennych decyzyjnych i wiele wariantów. Optymalizacji algorytmu mogłoby na pewno pomóc programowanie wielowątkowe, co by mogło kilkukrotnie przyspieszyć algorytm. Aby zaradzić tym wszystkim zmianom potrzeba by było większej liczby osób i większej liczby czasu, ale są to problemy do wyregulowania.