Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI, INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Zastosowanie algorytmu ewolucyjnego w planowaniu trasy przejazdu food trucka

Autorzy:

Dawid Lisek
Bartosz Żak

Kierunek studiów: **Automatyka i Robotyka**

1. Model zagadnienia.

Opis słowny analizowanego problemu:

Rozważamy problem, w którym należy zmaksymalizować zysk uzyskany przez odwiedzanie wydarzeń/festiwali w food trucku w danym przedziale czasowym. Startujemy z ustalonego punktu i mamy zbiór festiwali o danych lokalizacjach, datach trwania, liczbie odwiedzających festiwal w poszególnych dniach jego trwania, koszcie składników w danym mieście oraz opłacie parkingowej za każdy dzień wydarzenia. Koszt odwiedzenia festiwalu jest również zależny od odległości. Pozyskanie przychodu z wydarzenia skutkuje przesunięciem aktualnej daty o jeden dzień. Jeżeli festiwal trwa dłużej niż jeden dzień możliwe jest uzyskanie przychodów więcej niż raz bez dodatkowych kosztów związanych z dojazdem. Po upłynięciu zadanego czasu doliczany jest jeszcze koszt powrotu do punktu startowego.

Przyjęte uproszczenia:

- Cenę paliwa oraz spalanie auta modyfikujemy jednym parametrem algorytmu,
- Czas dojazdu do wydarzenia jest pominięty,

Model matematyczny:

- **Postać rozwiązania:**

Rozwiązanie ma postać macierzy, w której znajdują się informacje o poszczególnych wydarzeniach odwiedzonych przez food trucka. Poszczególne rozwiązania są posortowane według kolejności odwiedzania danych festiwali. Przykładowe rozwiązania składa się z elementów takich jak:

Event_id 1	City 1	Stay_duration	Ingredients_bought
Event_id 2	City 2	Stay_duration	Ingredients_bought
Event_id 3	City 3	Stay_duration	Ingredients_bought

Event_id – indeks festiwalu,

City – miasto, w którym odbywa się festiwal,

Stay_duration – czas przez który pozostajemy na danym festiwalu,

Ingredients_bought – liczba składników zakupiona w mieście odbywania się festiwalu

- **Przyjęte ograniczenia:**

$t_a \leq t_{max}$– data aktualna nie może przekroczyć końcowej daty podanej przez użytkownika,

$t_a \in [t_{start}, t_{end}]$ – aktualna data musi zawierać się w czasie pomiędzy początkiem, a zakończeniem festiwalu,

$0 \leq ing_a \leq capacity$ – ilość aktualnie posiadanych składników nie może być mniejsza od 0 oraz większa od ustalonej pojemności.

- **Funkcja celu:**

$$f(a) = \sum_{i=0}^{s} [\sum_{j=0}^{sd} [v_{coeff} * v_{ij} * p_p] - d_{coeff} * (d_{i-1,\ i} + d_{end,\ 0})] \quad \rightarrow max$$

gdzie:

s – ilość wydarzeń w rozwiązaniu,

sd – stay_duration – ilość dni na wydarzeniu,

v_{coeff} – przewidywany ułamek osób odwiedzających dany festiwal, które skorzystają z naszych usług,

v_{ij} – liczba odwiedzających i-ty festiwal w j-tym dniu jego trwania,

p_p – product_price – cena sprzedaży pojedynczego produktu,

d_{coeff} – współczynnik kosztu przejechanych kilometrów np. 50 zł za 100 km

$d_{i-1,\ i}$ – odległość pomiędzy kolejno odwiedzonymi miejscami,

$d_{end,\ 0}$ – odległość między miejscem końcowym a startowym,

- **Funkcja kary:**

W celu spełnienia ograniczeń zastosowaliśmy dynamiczny model funkcji kary. Wielkość kary zwiększa się wraz ze wzrostem przekroczenia ograniczeń dzięki czemu zapewniamy pogorszenie jakości rozwiązań, które przekraczają przyjęte przez nas ograniczenia. Funkcja kary:

$$f_p(a) = c_{p\_coeff} * ing_{exceeded} + d_{p\_coeff} * (c_{date} - e_{date})$$

gdzie:

c_{p_coeff} – capacity_punishment_coeff – współczynnik funkcji kary za przekroczenie pojemności,

$ing_{exceeded}$ – ilość przekroczonych składników

d_{p_coeff} – date_punish_coeff – współczynnik kary za nietrafienie w zakres dat z wydarzenia,

$c_{date} - e_{date}$ – ilość dni, które przekroczyły zakres odbywanego wydarzenia.

2. Algorytm rozwiązujący problem.

Opis teoretyczny:

Algorytm ewolucyjny przeszukuje przestrzeń rozwiązań problemu w celu wyszukania rozwiązań o najlepszej jakości. Działanie algorytmu przypomina zjawisko ewolucji biologicznej. Zaczynamy od losowego utworzenia populacji startowej. Następnie najlepiej przystosowane osobniki są wybierane do reprodukcji w celu uzyskania potomstwa następnego pokolenia. Kolejną generację tworzymy poddając wybrane osobniki operacjom krzyżowania oraz mutacji. Całość powtarzamy aż do wystąpienia warunku stopu.

Uproszczony schemat algorytmu ewolucyjnego :

Krok 1: Generacja populacji startowej.

Krok 2: Ocena populacji startowej - selekcja.

Krok 3: Operacje krzyżowania.

Krok 4: Operacje mutacji.

Krok 5: Utwórz nową populację.

Krok 6: Powtórz kroki 2-5 aż do osiągnięcia warunku stopu.

Warunkiem stopu w algorytmie może być osiągnięcie maksymalnej ilości iteracji lub zadanej wartości funkcji celu.

Opis elementów opracowanych:

- **Metoda selekcji.**

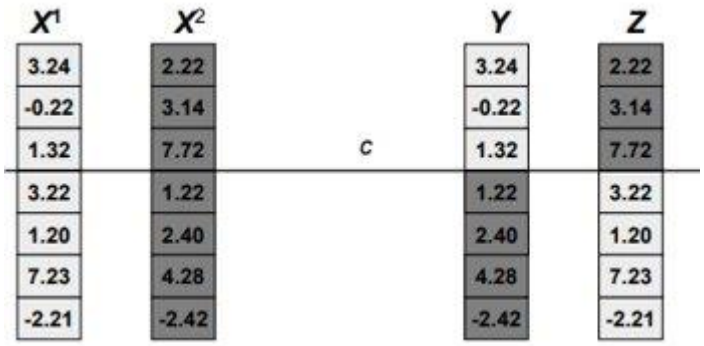
Zadaniem selekcji jest wybór osobników – rodziców, które zostaną poddane operacji krzyżowania w celu utworzenia nowych osobników w populacji – potomstwa. Metoda selekcji na podstawie funkcji celu poddaje ocenie poszczególne rozwiązania. Metoda selekcji ma bezpośredni wpływ na zbieżność algorytmu. W naszym algorytmie wykorzystaliśmy **metodę turniejową**. Polega ona na losowym podziale populacji na podzbiory oraz wybraniu z nich najlepszych osobników. Ilość osobników w poszczególnym turnieju określa **parametr rozmiaru turnieju**. Metoda selekcji jest wykonywana aż do **zapełnienia wymaganej ilości rodziców** podawanej jako parametr algorytmu.

- **Operatory krzyżowania.**

Operator krzyżowania jest wykorzystywany do tworzenie potomków na podstawie genotypu ich rodziców. Operacji krzyżowania jest poddawana tylko pewna ilość rodziców, która jest określona na podstawie różnicy pomiędzy całkowitą ilością rodziców, a ilością rodziców, którzy są poddawaniu mutacji. W naszym algorytmie wykorzystaliśmy **dwupunktowe oraz jednopunktowe operatory krzyżowania**:

- Krzyżowanie jednopunktowe:

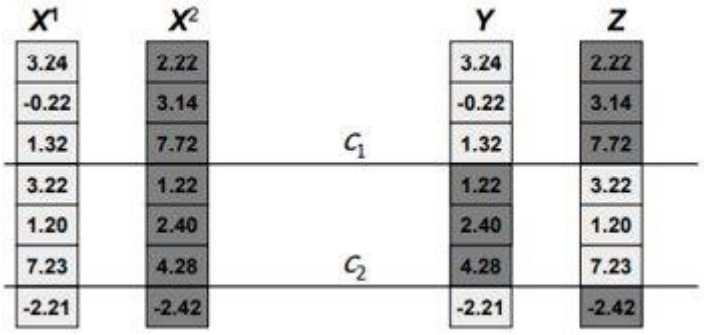
W genotypie rodzica wybieramy losowy punkt cięcia. Następnie tworzymy genotypy potomków zamieniając ze sobą poszczególne części z genotypu rodziców. Powyższą operację wizualizuje poniższy obraz:



Rysunek 1 Krzyżowanie jednopunktowe

- Krzyżowanie dwupunktowe:

Jest to operacja analogiczna do krzyżowania jednopunktowego z tą różnicą że wybierane są losowo dwa punkty cięcia genotypu rodziców. Następnie potomkowie są tworzeni analogicznie do poniższego schematu:



Rysunek 2 Krzyżowanie dwupunktowe

- **Operatory mutacji.**

Operatory mutacji mają za zadanie zapewnić różnorodność osobników w nowej populacji. Dzięki takim operatorom przeszukamy większą część przestrzeni rozwiązań. Ilość rodziców poddanych mutacji określa parametr **mutation_size,** który określa jaki procent populacji rodziców jest poddawany mutacji. W naszym algorytmie zaimplementowaliśmy **mutację równomierną, mutację zamiany kolejności dwóch elementów rozwiązania oraz losową zmianę miejsca odbywania wydarzenia.** Użyta w algorytmie metoda mutacji jest wybierana losowo.

- Mutacja równomierna:

Losowo wybrany gen przyjmuje losową (z rozkładem równomiernym) wartość z zakresu dopuszczalnego

- Mutacja swap:

Z rozwiązania wybieramy losowo dwa elementy, a następnie zamieniamy je kolejnością.

- Mutacja zmiana wydarzenia:

Z listy dostępnych wydarzeń wybieramy jedno i wstawiamy je w losowy punkt rozwiązania zachowując parametry czasu pobytu na wydarzeniu oraz ilości zakupionych produktów.

Parametry algorytmu:

Do poprawnego działania algorytmu użytkownik musi wprowadzić pewne parametry:

- Rozmiar populacji,
- Ilość pokoleń,
- Rodzice [%] – jest to procentowa ilość populacji, która będzie użyta w celach stworzenia kolejnego pokolenia,
- Mutacja [%] - jest to procentowa ilość rodziców, którzy zostaną poddani operacji mutacji w celu utworzenia potomków. Pozostali rodzice zostają poddani operacji krzyżowania,
- Rozmiar turnieju – wielkość podzbioru na jakie ma zostać podzielona cała populacja w metodzie selekcji turniejowej,
- Wybór metody krzyżowania: dostępna jest metoda jednopunktowa oraz dwupunktowa
- Wybór metody mutacji: dostępna jest metoda mutacji równomiernej, mutacji swap oraz mutacji ze zmianą wydarzenia.

3. Aplikacja.

Wykorzystane oprogramowanie - Python:

Do implementacji algorytmu ewolucyjnego wykorzystaliśmy język programowania wysokiego poziomu jakim jest Python. Zaletą tego języka jest wysoka intuicyjność w wykorzystywaniu jego funkcjonalności oraz duża dostępność bibliotek o różnym zastosowaniu wraz z dokumentacją i przykładami użycia funkcji. Do minusów należy zaliczyć szybkość wykonywania operacji w tym języku, gdyż jest to jego największa wada. W naszym zagadnieniu wada ta zaczęła być mocno widoczna przy dużych rozmiarach populacji oraz wysokiej ilości generacji.

Wykorzystane zęwnętrzne biblioteki:

- GeoPy

Biblioteka GeoPy umożliwia nam uzyskanie współrzędnych geograficznych miast, które następnie używamy do pozyskania odległości pomiędzy miastami.

- Requests

Biblioteka wykorzystana do pozyskania odległości z OpenStreetMap. Odległość nie jest wyliczana w linii prostej lecz odbywa się z uwzględnieniem rzeczywistych dróg. Dzięki temu użytkownik nie musi podawać odległości jaką musi przebyć żeby odwiedzić dany festiwal.

- Matplotlib

Biblioteka wykorzystana do generacji wykresów funkcji celu. Jej największą zaletą jest prostota oraz intuicyjność obsługi.

- PySimpleGui

Na podstawie tego modułu stworzyliśmy GUI – Graphical User Interface. Ułatwia ona wprowadzanie oraz zapis parametrów algorytmu oraz zagadnienia. W oknie wyświetlamy również przebieg funkcji celu oraz najlepsze uzyskane rozwiązanie.

- Pandas

Jest to biblioteka służąca do obróbki danych. Za jej pomocą wczytujemy plik programu Microsoft Excel z rozszerzeniem .xlsx. Pakiet ten oferuje wiele funkcji związanych z Data Science jak np. wczytywanie danych zapisanych w różnych formatach oraz odczyt informacji z baz danych w języku SQL.

Wymagania odnośnie uruchomienia

Wszystkie pliki aplikacji można pobrać ze zdalnego repozytorium GitHub znajdującego się w poniższym linku:

Link do zdalnego repozytorium projektu

Aby uruchomić naszą aplikację potrzebny jest zainstalowany język Python w wersji przynajmniej 3.8 oraz opcjonalnie aplikacja IDE taka jak np. PyCharm lub Visual Studio Code. Następnie po otwarciu projektu IDE poprosi nas o instalację pakietów z pliku requirements.txt. Jeśli tak się nie stanie należy wpisać poniższą komendę w terminalu:

```
pip install -r requirements.txt
```

Po instalacji wszystkich pakietów należy uruchomić main.py. Powinno ukazać się poniższe okno GUI, w którym użytkownik może modyfikować wszystkie parametry algorytmu.

Format danych / wyników

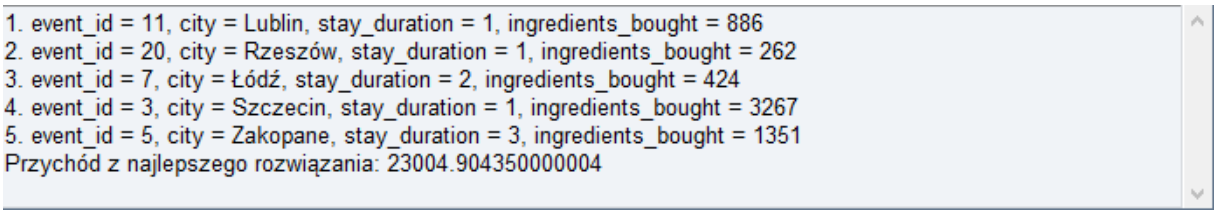
Przykładowe dane dotyczące festiwali są przedstawione w pliku example_data w folderze …\Badania_Operacyjne_2\data. Użytkownik może go edytować lub stworzyć własny plik z rozszerzeniem .xslx. Powinien on składać się z następujących kolumn:

- event_id – indeks wydarzenia,
- city – miasto, w którym odbywa się dane wydarzenie,
- start_date – data początkowa festiwalu,
- end_date – data zakończenia festiwalu,
- visitors – przewidywana liczba odwiedzających festiwal w poszczególnym dniu,
- ingredient_cost – cena składników jaką ponosimy kupując je w danym mieście,
- parking_cost – opłata parkingowa w poszczególnych dniach trwania festiwalu

event_id	city	start_date	end_date	visitors	ingredient_cost	parking_cost
0	Kraków	04.07.2022	06.07.2022	10000,11000,9000	5,34	300,200,100
1	Warszawa	05.07.2022	07.07.2022	12000,30000,25000	4,94	300,200,100
2	Gdynia	06.07.2022	08.07.2022	8000,10000,13000	5,50	300,200,100
3	Szczecin	07.07.2022	09.07.2022	9000,8000,7000	4,89	300,200,100

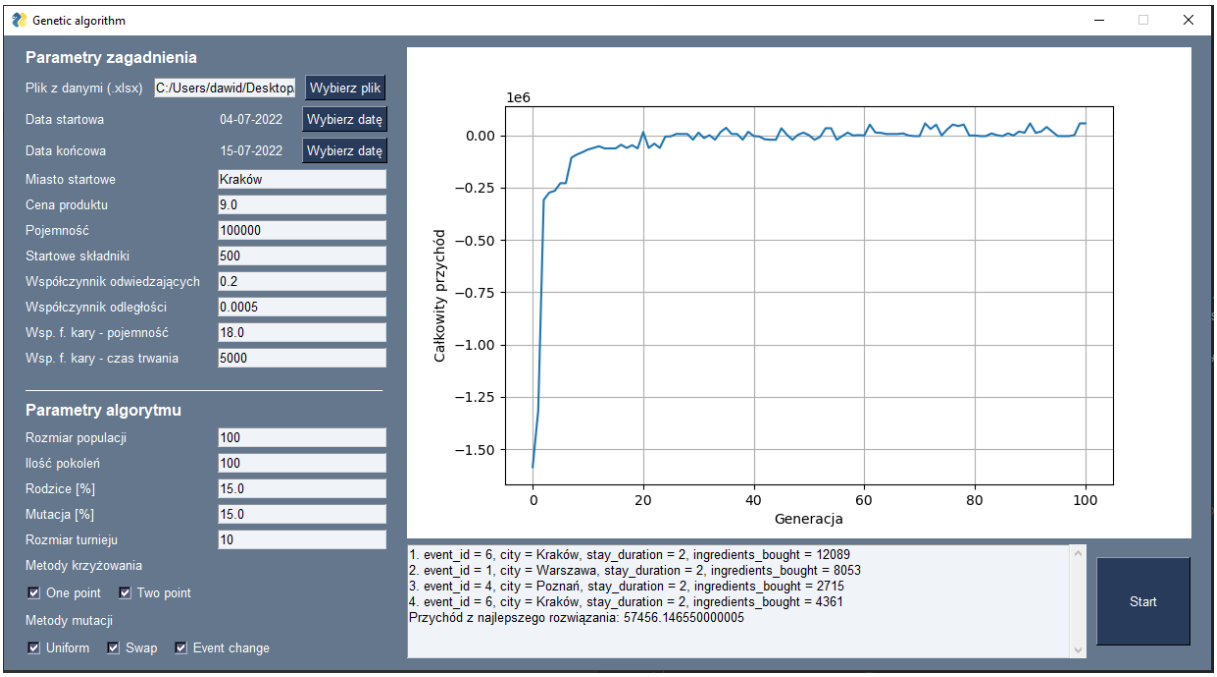
Rysunek 3 Przykładowa tabela z danymi:

Wyniki są zapisywane w postaci macierzy tak jak to zostało wspomniane w rozdziale o modelu matematycznym zagadnienia. Użytkownik dostaje wynik w postaci ciągu wydarzeń uporządkowanych w kolejności odwiedzania. Każde wydarzenie posiada również informację o swoim ID, ponieważ w jednym mieście może odbywać się kilka wydarzeń w różnym lub tym samym czasie. Wydarzenie posiada również ilość składników jaką musimy zakupić w danym mieście oraz czas pobytu na danym wydarzeniu. Pod rozwiązaniem kosztowym jest wyświetlana również wartość funkcji celu – zarobek jaki uzyskaliśmy na naszym Food Trucku.



Rysunek 4 Przykładowe rozwiązanie

Wygląd aplikacji oraz obsługa i opisy poszczególnych pól:



Rysunek 5 Wygląd aplikacji oraz obsługa i opisy poszczególnych pól.

Parametry zagadnienia:

- Plik z danymi (.xlsx) – należy tutaj dodać ścieżkę do pliku z danymi dotyczącymi festiwali,
- Data startowa – wybieramy datę startową od której zaczynamy optymalizacje,
- Data końcowa – wybieramy datę końcową,
- Miastowe startowe – podajemy miasto od którego zaczynamy przejazd,
- Cena produktu – cena sprzedawanego produktu, musi być większa od 0,
- Pojemność – maksymalna ilość składników, które możemy przechowywać, musi być większa od 0,
- Startowe składniki – ilość składników z jakimi zaczynamy przejazd, musi być większy od 0 oraz mniejsze od maksymalnej pojemności,
- Współczynnik odwiedzających – przewidywany ułamek osób odwiedzających dany festiwal, które skorzystają z naszych usług, musi być w zakresie [0; 1],
- Współczynnik odległości - współczynnik kosztu przejechanych kilometrów np. 50 zł za 100 km, musi być większe od 0,
- Wsp.f.kary – pojemność – współczynnik funkcji kary za przekroczenie pojemności, musi być wartość większy od 0,
- Wsp.f.kary – czas trwania – współczynnik kary za nietrafienie w zakres dat z wydarzenia, musi być większy od 0,

Parametry algorytmu:

- Rozmiar populacji – musi być większy od 0,
- Ilość pokoleń – ilość iteracji algorytmu, musi być większe od 0,
- Rodzice [%] – procentowa ilość rodziców, która jest tworzona na podstawie obecnej populacji, musi być za zakresu (0; 100),
- Mutacja [%] – procentowa ilość potomków, która jest poddawana mutacji, musi być z zakresu (0; 100),
- Rozmiar turnieju - wielkość podzbioru na jakie ma zostać podzielona cała populacja w metodzie selekcji turniejowej,
- Checkboxy do metody krzyżowania oraz mutacji – zaznaczenie przynajmniej jednej metody mutacji oraz krzyżowania, która zostanie użyta w algorytmie.

4. Testy.

- **Przygotowany zbiór danych:**

Pierwszy zestaw testów zostanie przeprowadzony dla zestawu danych składającego się z 21 wydarzeń. Kilka wydarzeń odbywa się w tym samym mieście, ale w różnym czasie.

event_id	city	start_date	end_date	visitors	ingredient_cost	parking_cost
0	Kraków	04.07.2022	06.07.2022	10000,11000,9000	5,34	300,200,100
1	Warszawa	05.07.2022	07.07.2022	12000,30000,25000	4,94	300,200,100
2	Gdynia	06.07.2022	08.07.2022	8000,10000,13000	5,50	300,200,100
3	Szczecin	07.07.2022	09.07.2022	9000,8000,7000	4,89	300,200,100
4	Poznań	08.07.2022	10.07.2022	13000,12500,10000	5,40	300,200,100
5	Zakopane	09.07.2022	11.07.2022	7000,9000,10000	5,12	250,200,100
6	Kraków	10.07.2022	11.07.2022	12000,30000	5,05	200,100
7	Łódź	11.07.2022	12.07.2022	15000,20000	4,93	500,300
8	Bydgoszcz	15.07.2022	17.07.2022	10000,8000,12000	5,37	300,250,200
9	Toruń	13.07.2022	14.07.2022	3000,7000	4,72	300,200
10	Gdańsk	10.07.2022	12.07.2022	7000,4000,3000	5,03	400,350,250
11	Lublin	12.07.2022	12.07.2022	8000	5,02	500
12	Katowice	08.07.2022	12.07.2022	12000,13000,15000,200	5,16	500,300,250,2
13	Kielce	04.07.2022	07.07.2022	1000,3000,8000,7000	5,02	500,300,250,2
14	Opole	05.07.2022	06.07.2022	10000,15000	5,5	600,300
15	Zielona Gó	04.07.2022	05.07.2022	6000,7000	5,01	250,200
16	Lubin	05.07.2022	08.07.2022	6000,7000,6500,6900	5,13	300,250,200,2
17	Suwałki	08.07.2022	10.07.2022	13000,12000,8000	5,06	420,385,200
18	Piła	10.07.2022	12.07.2022	5000,4000,6000	5,12	370,300,250
19	Słupsk	12.07.2022	12.07.2022	1200	5,11	350
20	Rzeszów	13.07.2022	13.07.2022	10000	5,15	500
21	Jawidz	13.07.2022	13.07.2022	10001	5,38	501

Rysunek 6 Zbiór danych

- **Przyjęta metodyka testów.**

Ze względu na dużą losowość algorytmu testy były wykonywane wielokrotnie dla tego samego zestawu parametrów. Następnie z uzyskanych wyników była wyliczana średnia, która jest przedstawiona w tabeli porównawczej dla każdego testu. Dla zobrazowania wpływu parametru na działanie algorytmu zostały dołączone również przebiegi funkcji celu, które najczęściej pojawiały się w danym zbiorze testu dla tego parametru.

- **Test nr 1: Test ze względu na rozmiar populacji:**

Parametry algorytmu:

Wykorzystane metody krzyżowania: **One point, Two point**

Wykorzystane metody mutacji: **Uniform, Swap, Event change**

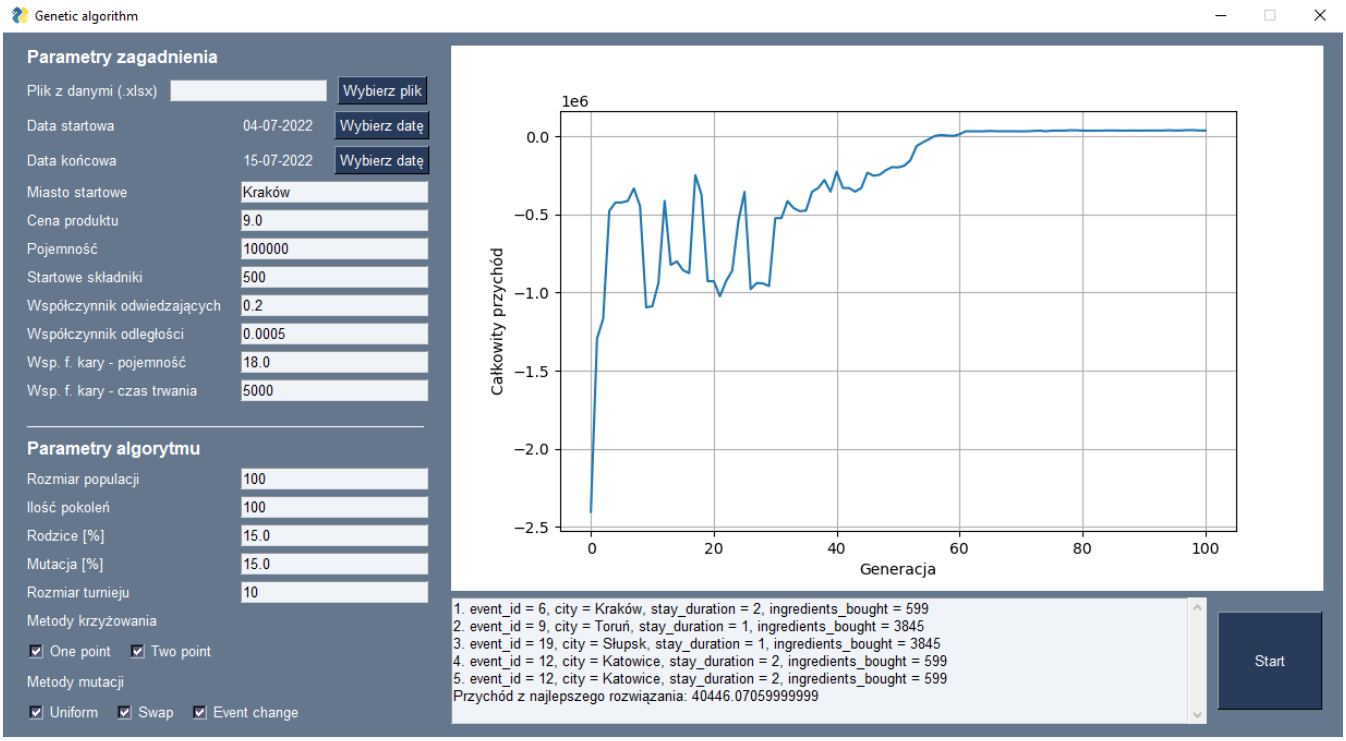
Ilość pokoleń = 100, Rodzice = 15%, Mutacja = 15%

Rozmiar populacji	Średnia wartość funkcji celu
50	-53542
100	44503
200	59995
500	68066
1000	106019
5000	127348
10000	132378

Tabela 1 Wyniki testu nr 1

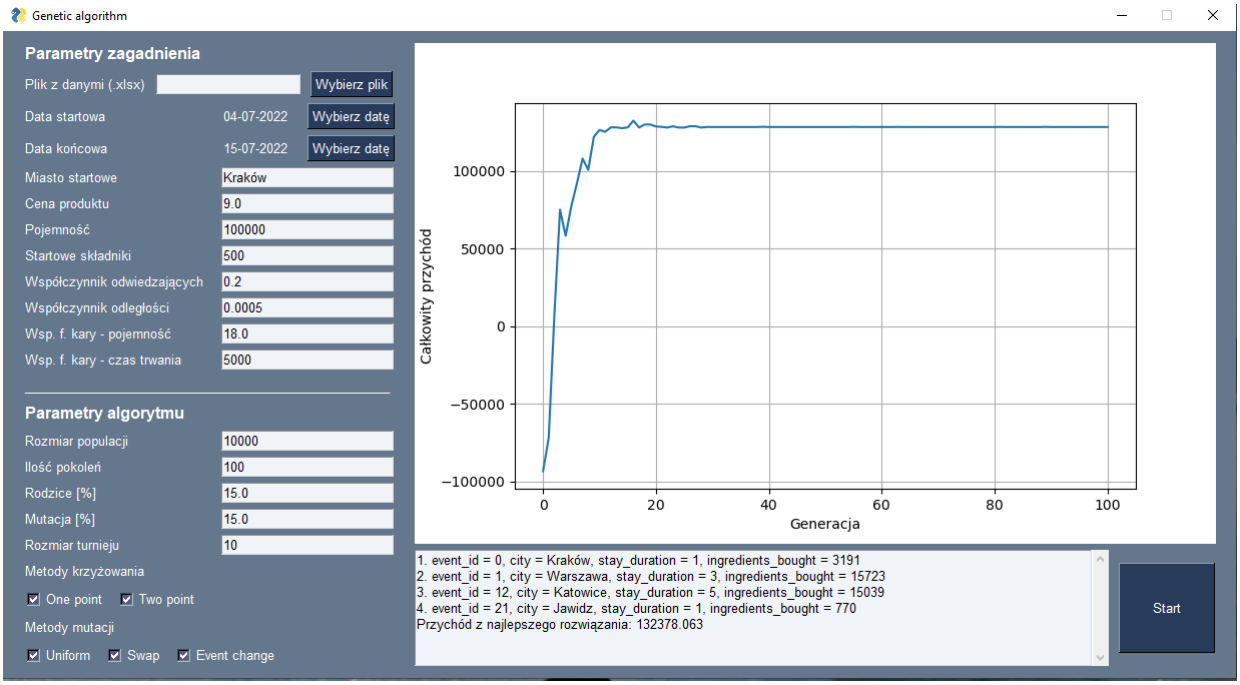
Porównanie przebiegów funkcji celu dla dużej oraz małej populacji

1) Populacja = 100:



Rysunek 7 Przebieg algorytmu dla rozmiaru populacji równej 100

2) Populacja = 10000:



Rysunek 8 Przebieg algorytmu dla rozmiaru populacji równej 10000

Wnioski: Rozmiar populacji ma bezpośredni wpływ na przebieg funkcji celu oraz przeszukiwanie przestrzeni rozwiązań. Dla dużego rozmiaru populacji możemy zauważyć praktycznie stały przebieg funkcji celu po małej liczbie iteracji. Duży rozmiar populacji zapewnia przeszukanie większej części przestrzeni rozwiązań. Dla małej populacji możemy zauważyć oscylacje na przebiegu funkcji celu. Przy małym rozmiarze populacji algorytm działał w sposób bardzo losowy co często kończyło się ujemną wartością funkcji celu.

- **Test nr 2: Test ze względu na ilość pokoleń:**

Parametry algorytmu:

Wykorzystane metody krzyżowania: **One point, Two point**

Wykorzystane metody mutacji: **Uniform, Swap, Event change**

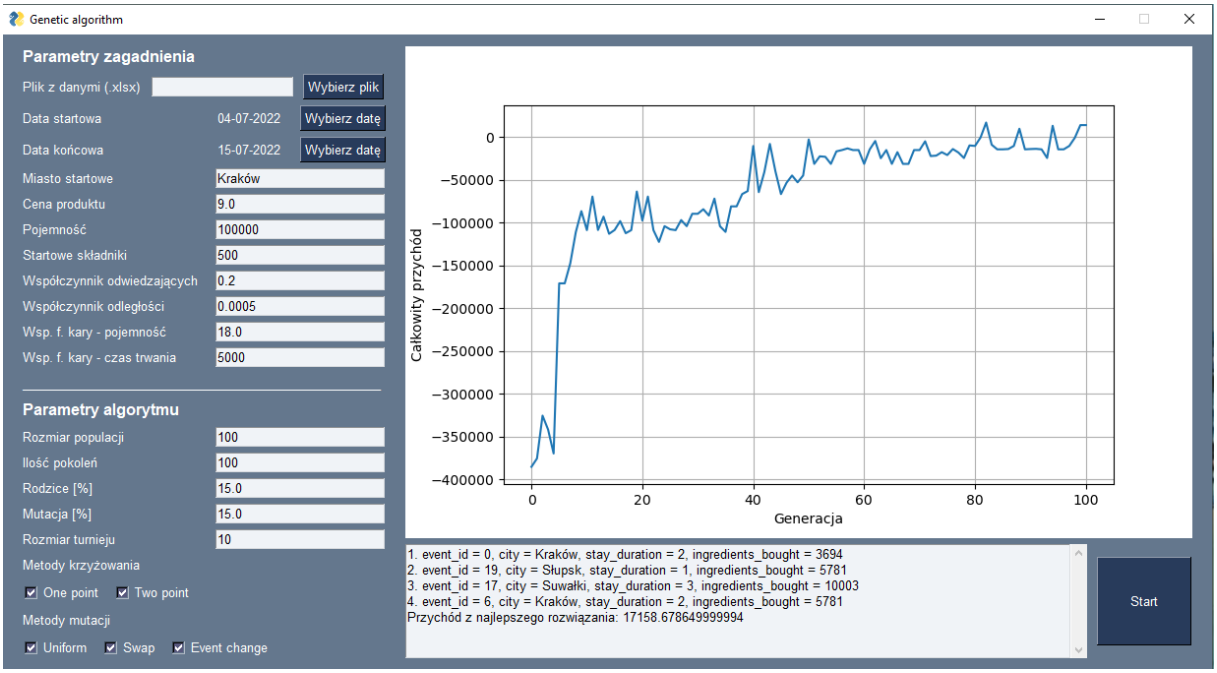
Rozmiar populacji = 1000, Rodzice = 15%, Mutacja = 15%

Ilość pokoleń	Wartość funkcji celu
50	57296
100	30177
200	66762
500	85142
1000	85884
2000	74070
5000	65095
10000	85018

Tabela 2 Wyniki testu nr 2

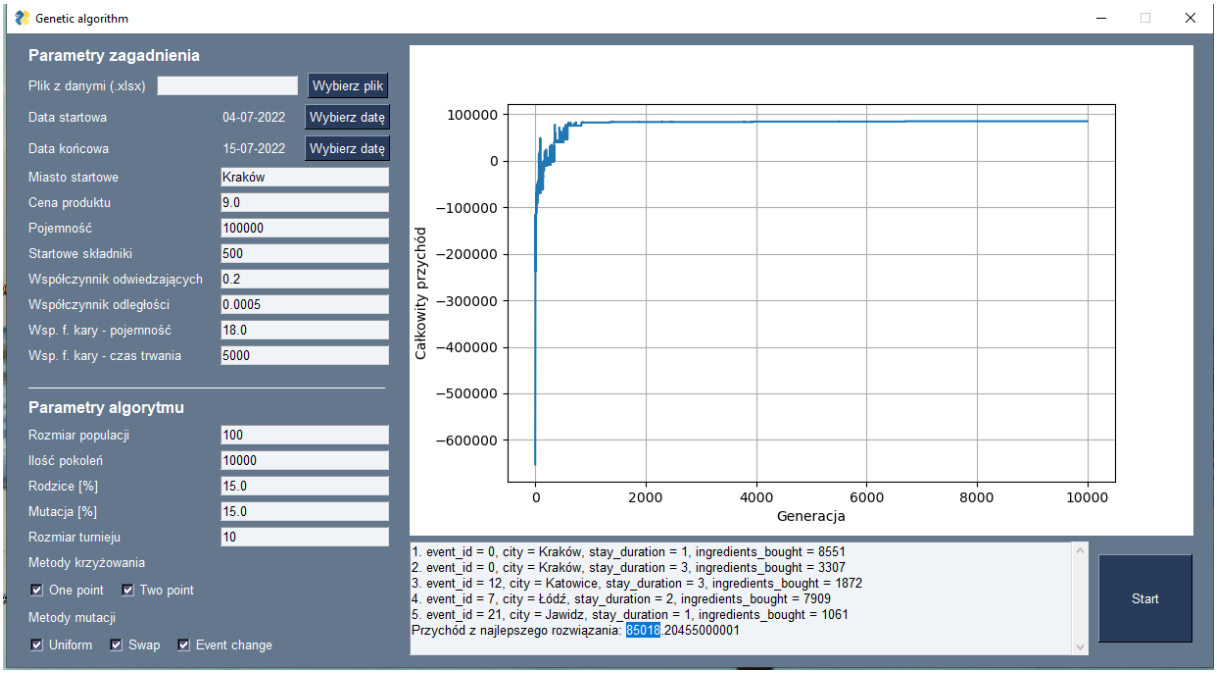
Porównanie przebiegów funkcji celu dla dużej oraz małej ilości pokoleń:

1) Ilość pokoleń = 100:



Rysunek 9 Przebieg algorytmu dla ilości pokoleń równej 100

2) Ilość pokoleń = 10000



Rysunek 10 Przebieg algorytmu dla ilości pokoleń równej 10000

Wnioski: Ilość pokoleń ma również duży wpływ na przebieg funkcji celu. Dla mniejszej ilości pokoleń algorytm zachowywał się losowo co widać na przebiegu funkcji celu. Przy dużej ilości pokoleń algorytm utyka w pewnym miejscu i widać że kolejne iteracje nie wpływają na najlepsze znalezione rozwiązanie.

- **Test nr 3: Test ze względu na zastosowane operatory mutacji oraz krzyżowania:**

Parametry algorytmu:

Rozmiar populacji = 200, Ilość pokoleń = 200 Rodzice = 15%, Mutacja = 15%

Przeprowadzone testy ze względu na użyte operatory mutacji oraz krzyżowania:

Użyty operator mutacji	Użyte operatory krzyżowania	Średnia wartość funkcji celu
Uniform	One point	38580
Uniform	Wszystkie	39118
Swap	One point	-50880
Swap	Wszystkie	-39838
Event_change	One point	-78525
Event_change	Wszystkie	-156363
Uniform&Swap	One point	16264
Uniform&Swap	Wszystkie	31195
Uniform&Event change	One point	96457
Uniform&Event change	Wszystkie	70699
Swap&Event change	One point	34537
Swap&Event change	Wszystkie	25204
Wszystkie	One point	93910
Wszystkie	Wszystkie	84391

Tabela 3 Wyniki testu nr 3

Wnioski: Po przeprowadzonych testach możemy zauważyć że największe znaczenie dla uzyskania dodatniej wartości funkcji celu ma operator mutacji równomiernej – Uniform. Można to uzasadnić tym że ta mutacja zmienia losowo wybraną wartość genu z zakresu dopuszczalnego. Dzięki temu na nasze rozwiązania nie zostaje nałożona funkcja kary. Operatory swap oraz event change zmieniają kolejność rozwiązania przez co duży wpływ ma funkcja kary. Zapewniają one jednak przeszukanie większej przestrzeni rozwiązań.

- **Test nr 4: Test ze względu na ilość rodziców poddawanej mutacji i krzyżowaniu:**

Parametry algorytmu:

Wykorzystane metody krzyżowania: **One point, Two point**

Wykorzystane metody mutacji: **Uniform, Swap, Event change**

Rozmiar populacji = 200, Ilość pokoleń = 200, Rodzice[%] = 15

Mutacja [%]	Średnia wartość funkcji celu
5	37673
10	83675
15	98201
20	109275
25	78045
30	106607
35	96520
45	97312
50	116135
70	123069
99	133320

Tabela 4 Wyniki testu nr 4

Wnioski: Zwiększając ilość rodziców poddanych mutacji zmniejszamy tym samym ilość rodziców poddaną krzyżowaniu. Dla dużych wartości parametru mutacji wartość funkcji celu oscylowała na wysokim poziomie oraz była szybko znajdowana. Niska szansa wystąpienia mutacji powoduje zapewnienie większej różnorodności jakości rozwiązań.

- **Test nr 5: Test ze względu na ilość rodziców poddawanej:**

Parametry algorytmu:

Wykorzystane metody krzyżowania: **One point, Two point**

Wykorzystane metody mutacji: **Uniform, Swap, Event change**

Rozmiar populacji = 200, Ilość pokoleń = 200, Mutacja[%] = 15

Rodzice [%]	Średnia wartość funkcji celu
5	63003
10	59660
15	65928
20	55494
25	51103
30	51946
35	65745
45	-9828
50	-38351
70	-105167

Tabela 5 Wyniki testu nr 5

Wnioski: Duża ilość rodziców powoduje pogorszenie jakości rozwiązań. Zwiększając ilość rodziców powodujemy małą zmianę w elementach rozwiązania początkowego przez co spada jakość rozwiązania. Optymalną ilość rodziców możemy przyjąć na poziomie 10-20 %.

- **Test nr 6: Test ze względu na wielkość turnieju:**

Parametry algorytmu:

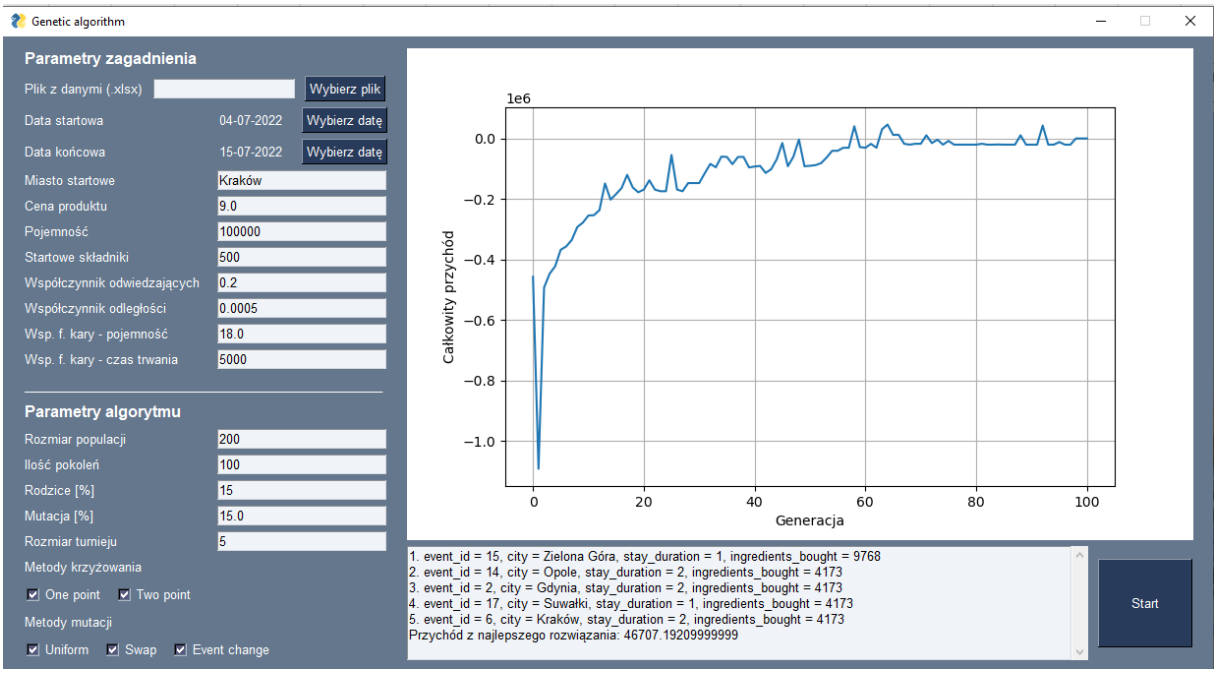
Wykorzystane metody krzyżowania: **One point, Two point**

Wykorzystane metody mutacji: **Uniform, Swap, Event change**

Rozmiar populacji = 200, Ilość pokoleń = 100, Rodzice[%] = 15, Mutacja[%] = 15

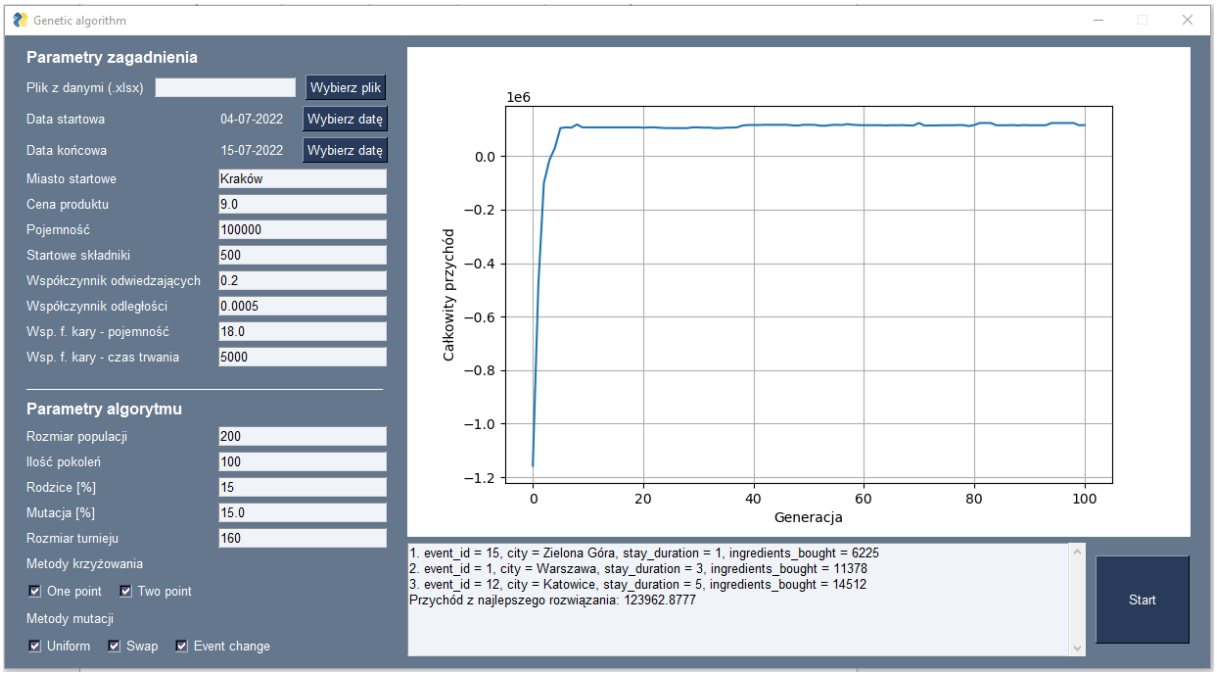
Rozmiar turnieju	Średnia wartość funkcji celu
5	60016
20	72419
40	82013
60	79223
80	84728
100	79021
120	109268
160	110505

Przebieg algorytmu dla małego rozmiaru turnieju = 5:



Rysunek 11 Przebieg algorytmu dla rozmiaru turnieju równego 5

Przebieg algorytmu dla dużego rozmiaru turnieju = 160:



Rysunek 12 Przebieg algorytmu dla rozmiaru turnieju równego 160

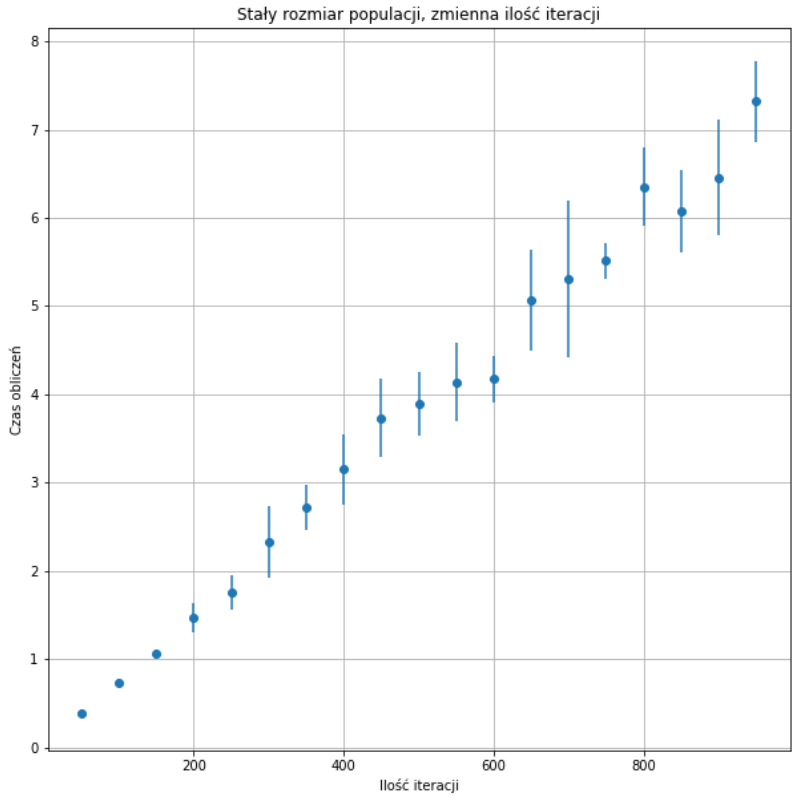
Wnioski: Rozmiar turnieju wpływa bezpośrednio na zbieżność algorytmu. Im jest większy, tym algorytm jest szybciej zbieżny.

Test nr 7: testy czasowe działania algorytmu:

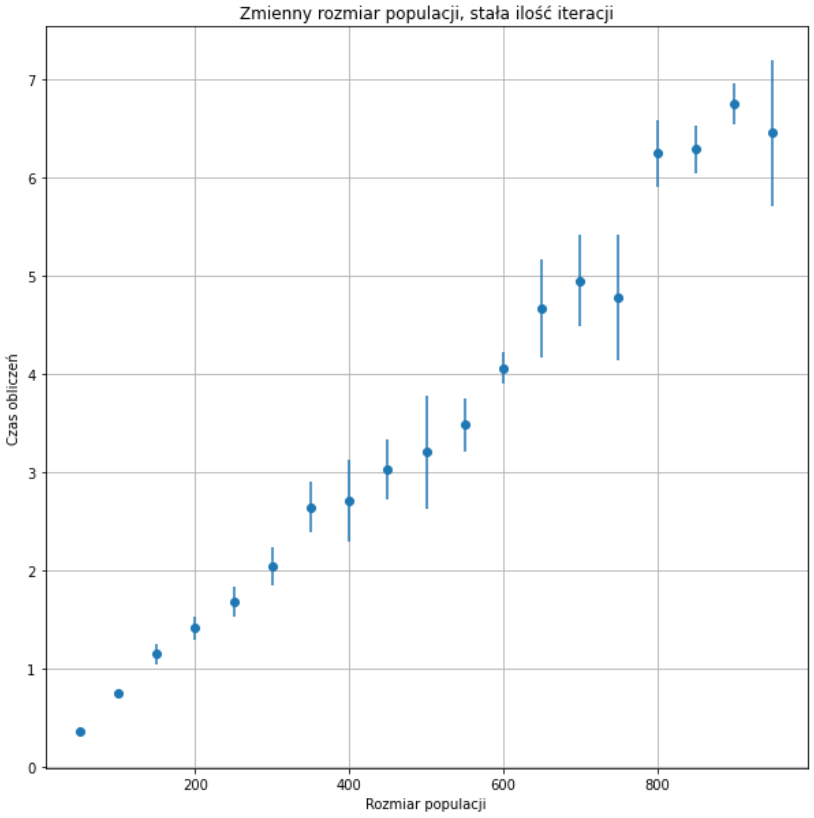
Na złożoność obliczeniową algorytmu największy wpływ ma rozmiar populacji oraz ilość pokoleń. Złożoność obliczeniową samego algorytmu genetycznego można przedstawić w przybliżeniu jako:

$$ilość\ iteracji * rozmiar\ populacji$$

W naszym przypadku na złożoność obliczeniową duży wpływ ma rodzaj wykonywanych operacji w języku Python. Początkowo po utworzeniu kodu algorytmu oraz jego testach nasz algorytm dla rozmiaru populacji np. 1000 oraz ilości iteracji=500 potrafił wykonywać się **7 minut**. Po zastosowaniu biblioteki Pickle oraz pewnych metod alokacji pamięci udało się skrócić ten czasu do ok. **40 sekund.** Duży wpływ na złożoność działania programu na również szybkość generatora liczb losowych. Jednak w naszym przypadku wykorzystanie biblioteki Random nie wpłynęło. Poniżej przedstawiono wykresy obecnej złożoności obliczeniowe algorytmu:



Rysunek 13 Czas wykonywania algorytmu [s] w zależności od liczby iteracji



Rysunek 14 Czas wykonywania algorytmu [s] w zależności od rozmiaru populacji

5. Podział pracy.

Etap	Bartosz Żak	Dawid Lisek
Model zagadnienia	40%, Struktury danych, ograniczenia	60%, Funkcja kary, funkcja celu
Algorytm opracowanie	60%, Metoda selekcji, operatory krzyżowania	40%, Operatory mutacji
Implementacja aplikacji	536/790, GUI, klasa SolutionElement, klasa Solution i metody profit(), ingredients_cost(), parking_cost(), overall_distance_cost(), capacity_punishment(), duration_punishment(), overall_profit()	254/790, geocode(), load_event_list(), driving_distances(), genetic_algorithm(), create_initial_population(), selection(), crossover(), mutation()
Testy	40%, Test 3, 4, 7	60%, Test 1, 2, 5, 6
Dokumentacja	40%, Rozdział 2, 3, 5	60%, Rozdział 1, 4, 6, 5

Tabela 6 Podział pracy członków zespołu

6. Wnioski.

Zaletą algorytmu ewolucyjnego jest niewątpliwie jego wszechstronność. Możemy go zastosować to rozwiązania problemu z dużą ilością zmiennych decyzyjnych. Posiada on jednak dużo parametrów, które wpływają na przebieg oraz zbieżność funkcji celu. Złe ich ustawienie może spowodować utknięcie algorytmu w maksimum lokalnym co często uniemożliwi eksplorację całej przestrzeni rozwiązań.

Przeprowadzone testy pokazały, że duży wpływ na zbieżność miał dobór operatorów mutacji. Operatory Swap oraz Event change znacznie zmniejszały zbieżność. Wybór operatorów krzyżowania miał mniejszy wpływ i jest to najprawdopodobniej spowodowane podobną zasadą działania obu z nich. Innym parametrem mającym znaczący wpływ na zbieżność był rozmiar turnieju.

Najbardziej dotkliwym problemem w działaniu aplikacji jest czas obliczeń. Dla dużego zestawu danych potrzeba większego rozmiaru populacji oraz ilości pokoleń, a nasza aplikacja słabo radzi sobie w takich przypadkach.

W przyszłości można na przykład napisać aplikację w szybszym języku programowania lub częściowo odejść od obiektowego podejścia na rzecz tabel i macierzy z biblioteki Numpy.