Matematyczne Metody Wspomagania Decyzji

Sprawozdanie z projektu

**Optymalizacja doboru**

**siatki przebiegu linii autobusowych metodą ewolucyjną**

Dominika Furgała

Dominik Gołębiowski

Jan Kosiński

Automatyka i Robotyka, rok III

Kraków, 01.2019

**Spis treści**

[**Geneza problemu**](#_80jv76bhkic5) **2**

[**Założenia zadania**](#_1e20sakyyjoy) **2**

[**Model matematyczny**](#_es1cqxo8iw1p) **4**

[**Opis algorytmu**](#_dbhu2rhz27f5) **5**

[**Implementacja algorytmu**](#_u8w2frsiroiy) **5**

[1. Tworzenie struktury programu - generowanie problemu](#_wzhwzixpecf) 5

[2. Inicjalizacja początkowej populacji](#_auvuusc27b56) 6

[3. Algorytm ewolucyjny](#_7nm6tydn48vn) 6

[Obliczanie rankingu otrzymanych rozwiązań](#_rrg41ngmh979) 7

[Algorytm wariacji - krzyżowanie](#_xdtxjm5fualt) 7

[Algorytm mutacji](#_dj6qloz2qb08) 9

[**Uproszczony schemat blokowy implementacji**](#_porlh8y4ily0) **10**

[**Oprogramowanie / aplikacja**](#_wcckpmqs2ou2) **11**

[Python](#_ag6tdidr2645) 11

[Biblioteki](#_g6c68mxiwb7f) 11

[GUI](#_srdbujeuwyhu) 12

[Baza Danych](#_ro5xk5ne0ly2) 13

[GitLab](#_6bu4sggfcgn0) 14

[**Metodyka testów**](#_kxymut1bw715) **15**

[**Testy**](#_suzjajofgen3) **16**

[Problem, na którym zostały przeprowadzone testy](#_l00me5h9i14o) 16

[Wynik działania algorytmu przy ustawieniach domyślnych](#_y8tze06hvm33) 17

[Wpływ zmian parametrów funkcji celu](#_wal8ej8ud6po) 18

[Wpływ intensywności operatorów genetycznych](#_b2ljq62u1e3w) 20

[Przykłady rozwiązań dla mniejszych problemów](#_nn58k5jmcl8o) 22

[**Wnioski**](#_sp5dju7cnff8) **23**

# Geneza problemu

Powodem, dla którego zdecydowaliśmy się na wybór problemu zajmującego się optymalizacją tras autobusów, było pewnego rodzaju niezadowolenie z obecnych rozwiązań, jakie są zastosowane dla sieci przystanków w Krakowie. Mieszkając i studiując w tym mieście już ponad dwa lata niejednokrotnie spotkaliśmy się z frustracją użytkowników komunikacji miejskiej, którymi sami także jesteśmy. Często poruszanie się po Krakowie jest utrudnione nie tylko przez zatłoczone ulice, lecz także przez nieoptymalny dobór linii autobusowych, co zmusza wiele osób do przesiadek, tracenia czasu na oczekiwanie kolejnego autobusu lub nawet jeżdżenia okrężną drogą z powodu braku innego, bezpośredniego połączenia.

Postanowiliśmy stworzyć algorytm uniwersalny, który będzie można zastosować do generowania połączeń autobusowych w każdym mieście. Z tego powodu nie ograniczyliśmy się do ściśle określonych parametrów i współrzędnych przystanków.

# Założenia zadania

Dany jest obszar prostokątny o określonej liczbie przystanków rozmieszczonych losowo. Każdy posiada współrzędne swojego położenia. Określona część z nich jest zdefiniowana jako przystanki pełniące funkcję pętli autobusowej - będące początkiem lub końcem trasy autobusu. Są one wybierane jako punkty najbardziej odległe.

Ponadto do każdego z przystanków przyporządkowana jest też określona liczba osób, które chcą jechać w danym kierunku (na konkretny przystanek). W celu urzeczywistnienia problemu uwzględniono zróżnicowaną popularność przystanków.

Stworzona jest początkowa generacja składająca się z ustawianej programowo liczby osobników - wygenerowanych losowo rozwiązań rozpięcia siatki linii autobusowych dla danego problemu.

W konfiguracji początkowej możemy zdefiniować ile przystanków chcemy wygenerować, na jakiej powierzchni, oraz ile zostanie wytypowanych pętli tramwajowych. Dzięki temu otrzymujemy zadeklarowany problem startowy, na którym następnie możemy przeprowadzać testy.

Przyjęte uproszczenia:

Ze względu na skomplikowaną postać zagadnienia musieliśmy przyjąć pewne uproszczenia. W naszym algorytmie staramy się dobrać odpowiednie trasy dla autobusów, bez uwzględnienia docelowego rozkładu jazdy. Stąd też nie zajmujemy się zagadnieniami, które są zmienne w czasie, czyli natężenie ruchu, korki, zmienna ilość osób czekających na przejazd. W naszym algorytmie jako odległość między przystankami rozumiemy średni czas przejazdu między nimi. Dzięki temu optymalizujemy czas przejazdu, a nie długość trasy.

Parametry inicjalizacji problemu:

* liczba przystanków
* wymiary płaszczyzny, na której umieszczone są przystanki (w jednostkach czasowych)
* minimalna odległość między dwoma przystankami
* maksymalna ilość ludzi na danym przystanku kierująca się na inny
* ilość przystanków, które będą pełniły funkcję pętli autobusowej

W celu przeprowadzenia testów i określenia, ile iteracji algorytmu jest niezbędne do uzyskania zoptymalizowanego wyniku, zostały także dodane parametry decydujące o:

* ilości osobników danej populacji
* ilości przeprowadzonych generacji

Celem algorytmu jest rozpięcie siatki linii autobusowych w jak najbardziej optymalny sposób, aby zadowolić jak najwięcej osób, zmniejszyć ich czas dojazdu do danego przystanku oraz zminimalizować ilość przesiadek.

# 

# Model matematyczny

Funkcja celu: ****

Ograniczenia:

Podczas generowania kolejnych populacji są dopuszczone rozwiązania, które pozostawiają niektóre przystanki pominięte. Nakładamy wtedy odpowiednią karę na takie rozwiązanie. Nie jest natomiast dopuszczalne rozwiązanie, które usuwa wszystkie linie.

Oznaczenia:

* **S** - kara za pominięte przystanki przez którąkolwiek linię - liczba pominiętych przystanków
* **T** - kara za nieodwiedzone przystanki będące pętlami - liczba pominiętych pętli
* **So** - kara za przesiadki - suma kwadratów przesiadek, jakie trzeba wykonać poruszając się pomiędzy każdymi dwoma przystankami
* **D** - suma długości wszystkich linii autobusowych
* **Dv** - odchylenie standardowe długości linii
* **P** - kara za niezadowolenie u pasażerów - suma liczb osób chcących się dostać między dwoma danymi przystankami pomnożonych przez liczbę przesiadek, które muszą wykonać
* **w** - współczynniki skalujące parametry funkcji celu
* **n1,n2** - potęgi obciążające dane elementy funkcji celu

Postać rozwiązania:

Rozwiązaniem jest lista wektorów składających się z identyfikatorów przystanków. Każdy wektor symbolizuje inną nitkę i trasę po jakiej przejeżdża autobus tej linii (na jakich przystankach się zatrzymuje). Pierwszym i ostatnim elementem każdego wektora jest identyfikator pętli autobusowej. Rozwiązanie optymalne to takie, dla którego funkcja celu przyjmuje najmniejszą wartość.

# 

# Opis algorytmu

Algorytm ewolucyjny (AE) jaki wykorzystaliśmy do rozwiązania naszego problemu, zgodnie z definicją naukową, polega na próbie wykorzystywania mechanizmów selekcji, reprodukcji i mutacji procesów ewolucyjnych.

Pierwszym wyraźnym podobieństwem AE do ewolucji naturalnej jest istnienie populacji (zbioru różnych możliwych rozwiązań problemu)– w przyrodzie rozumianej jako zbiór współistniejących osobników różnych gatunków. Sama ewolucja to powstawanie gatunków coraz lepiej dopasowanych do środowiska.

Aby móc wzorować się na tej metodzie trzeba ustalić pewną miarę dopasowania otrzymywanych rozwiązań (funkcję oceny). To pozwala na wyewoluowania populacji składającej się z rozwiązań “dobrych”, preferowanych oraz “gorszych”. Rozwiązania traktuje się jak genotypy organizmów, które podlegają krzyżowaniu i mutacji.

Proces ten jest powtarzany wiele razy. Aby miało to sens, a algorytm faktycznie zbiegał się w rozsądnym czasie do optymalnego rozwiązania potrzebny jest jeszcze jeden mechanizm – selekcji. Populację poddaje się ocenie. Następnie w pewien sposób powiela się osobniki z populacji bieżącej do nowej w taki sposób, żeby większe szanse na skopiowanie miały osobniki dobre.

Krzyżowanie, w połączeniu z selekcją preferującą dobre osobniki, powoduje ujednolicanie populacji, czemu z kolei przeciwdziała mutacja. Intensywność tych operacji powinna być zatem wzajemnie wyważona.

# Implementacja algorytmu

W tej części zamieszczony został opis działania naszych algorytmów i stworzonych funkcji. Do implementacji w większości przypadków został wykorzystany język Python.

## 1. Tworzenie struktury programu - generowanie problemu

Tworzona jest tablica o n-wierszach (n - liczba przystanków danego problemu). Zawiera ponumerowaną listę przystanków. Dla każdego z nich losowana jest para współrzędnych na płaszczyźnie kartezjańskiej o podanych wymiarach. Funkcja ponadto sprawdza odległość między każdymi dwoma przystankami, zabezpieczając, aby nie były one od siebie zbyt blisko (określa to kolejny, podany argument).

Następnie tworzona jest macierz nxn zawierająca odległości pomiędzy każdymi dwoma przystankami. W tej fazie tworzy się także oddzielna tablica określająca ile ludzi chce przemieścić się z konkretnego miejsca do innego. Uwzględniono tu możliwość większej popularności niektórych przystanków, w tym wypadku przydzielaną losowo.

## 2. Inicjalizacja początkowej populacji

Zgodnie z zadaną wartością tworzone są osobniki pierwotnej populacji. Proces ten oparty jest w większej części o wybór losowy. Dla każdego osobnika przeprowadzane są następujące działania:

1. Zostaje stworzona lista przystanków, którym nadane jest miano pętli autobusowych. Wielkość tej listy określona jest przez podaną wartość. Wybór następuje według najbardziej oddalonych od siebie par przystanków, dlatego liczba pętli zawsze jest parzysta.
2. Następuje losowe parowanie pętli autobusowych w przypadkowej ilości. Wynik tego zabiegu determinuje ile dany osobnik będzie posiadał linii autobusowych. Wynikiem jest lista par pętli.
3. Do każdej wybranej nitki losowane są przystanki i ustawiane w pewnej kolejności z uwzględnieniem, że dany przystanek nie może powtarzać się w tej samej nitce.

## 3. Algorytm ewolucyjny

Dokonywana jest ocena wszystkich osobników danej populacji ze względu na funkcję celu. Następnie przeprowadzona zostaje selekcja, której celem jest wybór osobników wybranych do grupy rozrodczej (wśród których przeprowadzone zostaną krzyżowania). Jako metodę selekcji (spośród m.in. strategii elitarnej, selekcji turniejowej, rankingu) wybrano **selekcję listą rankingową (liniowo).** Rozwiązanie to posiada wiele zalet w porównaniu do innych metod:

* Podobnie jak selekcja turniejowa, odpowiednia do problemów maksymalizacji i minimalizacji,
* Nie posiada wady metody ruletki dotyczącej konieczności skalowania funkcji przystosowania.

### Obliczanie rankingu otrzymanych rozwiązań

Osobnikom nadawana jest ranga (zależna od wartości przystosowania). Polega na wyborze osobników zgodnie z przypisanymi im rangami. Korzystając z takiej „listy rankingowej” definiowana jest funkcja określająca liczbę wybieranych kopii chromosomów w zależności od ich rangi. Na podstawie tej funkcji realizowany jest algorytm selekcji.

)

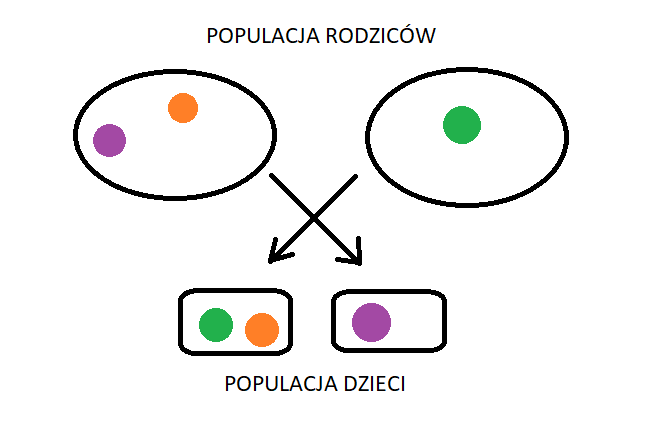
Po odpowiednim przeskalowaniu zabieg ten daje w rezultacie listę rankingową dla każdego osobnika danej populacji. Jako parametr przyjęliśmy 1, natomiast za2. Dzięki temu prawdopodobieństwo wylosowania najlepszego rozwiązania jest 2 razy większe od wylosowania rozwiązania najgorszego. Zmniejsza to zbieżność algorytmu, ponieważ teoretycznie gorsze rozwiązania mają szansę stworzyć nową populację.

### Algorytm wariacji - krzyżowanie

Krzyżowanie polega na przekazaniu potomkowi fragmentów genotypu pochodzących od obojga rodziców i wykonuje się w każdym cyklu algorytmu, ponieważ służy do stworzenia populacji dzieci. Oczekujemy, że pojawi się potomstwo posiadające dobre cechy obojga rodziców.

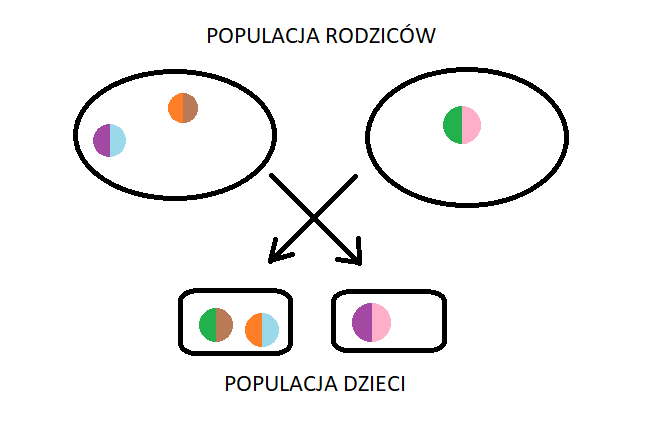
W naszej implementacji wykorzystaliśmy 2 wersje krzyżowania:

* Wariacja, w której wszystkie nitki z obojga rodziców są przenoszone do jednej listy, a następnie losowo rozdzielane do poszczególnych dzieci. Zamieszczony rysunek przedstawia przykład sposobu działania tej wariacji.



*Rys. 1. Zasada działania krzyżowania*

* Wariacja cechująca się podziałem nitek w losowych miejscach, a następnie łączeniem ich części parując je tak jak na zamieszczonym przykładzie. Podobnie jak w poprzednim przypadku, nowo utworzone nitki są losowo przydzielane do poszczególnych dzieci.



*Rys. 2. Inny wariant krzyżowania*

Z powodu tego, że na kolejne generacje w algorytmie ewolucyjnym duży wpływ w ma losowość, może zdarzyć się taki przypadek, że otrzymamy puste dziecko tzn. nieposiadające żadnej nitki. W takim przypadku generujemy nowe rozwiązanie dla danego dziecka. Takie działanie zaburza zbieżność algorytmu, jednak wyklucza rozwiązanie niedopuszczalne.

Wybór rodzaju krzyżowania, a także częstotliwość z jaką się ono wykonuje, definiuje parametr VARIATION\_PROBABILITY, który określa w procentach z jakim prawdopodobieństwem wykonuje się pierwszy rodzaj wariacji.

### Algorytm mutacji

Mutacja polega na losowej zmianie nowego genotypu. Oczekuje się, że pozwoli nam to odkryć nowe, nieznane dotąd cechy, które później mogą okazać się jednym z dobrych rozwiązań. Jednak nieumiejętny dobór częstotliwości lub prawdopodobieństwa jej wywołania może doprowadzić do zbyt dużej rozbieżności algorytmu i utraty dobrych rozwiązań.

W naszej implementacji zostało uwzględnionych 6 rodzajów mutacji:

* Dodawanie losowej liczby nowych przystanków do istniejącej już nitki w przypadkowych miejscach z brakiem możliwości zmiany pętli.

**[ 2 , 4, 5, 6, 10 ] ----> [ 2 , 4, 5, 3, 6, 7, 8, 10 ]**

* Zamiana przypadkowej liczby przystanków w istniejącej już nitce na losowym fragmencie.

**[ 2 , 4, 5, 6, 10 ] ----> [ 2 , 4, 3, 7, 8, 10 ]**

* Usuwanie losowej liczby przystanków z istniejącej już nitki w przypadkowych miejscach z brakiem możliwości usunięcia pętli

**[ 2 , 4, 5, 6, 10 ] ----> [ 2 , 4, 10 ]**

* Całkowita zamiana jednej nitki. Jeśli jest to możliwe dobierane są nowe pętle.

**[ 2 , 4, 5, 6, 10 ] ----> [ 9 , 3, 6, 7, 4, 1 ]**

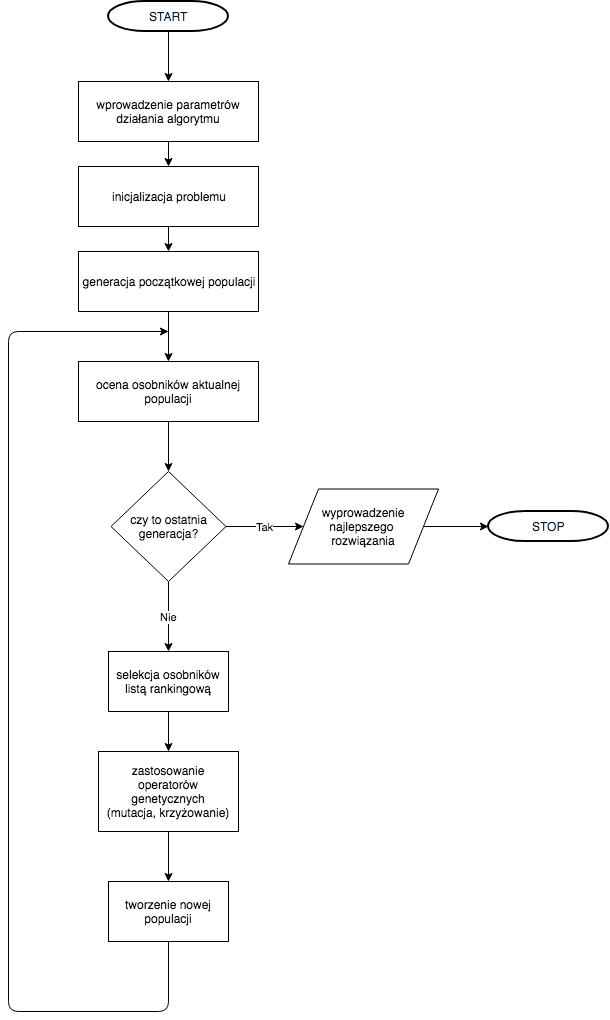
* Całkowite usunięcie jednej z nitek danego rozwiązania. W momencie gdy rozwiązanie ma tylko jedną nitkę, mutacja nie jest wykonywana.

**[ 2 , 4, 5, 6, 10 ] ----> XXXXXXXXXXXXX**

Mutacje przeprowadzane są zgodnie z określonym prawdopodobieństwem oraz częstotliwością wykonywania podaną w procentach.

# 

# Uproszczony schemat blokowy implementacji

[](https://www.draw.io/#G1VVTtz9IoKwn1xuUwXhGgUfgyVKuhCz6E)

# Oprogramowanie / aplikacja

## Python

Do implementacji algorytmu optymalizującego zdecydowaliśmy wykorzystać język programowania wysokiego poziomu Python3.7. Ma on bardzo ogólne przeznaczenie i współpracuje z wieloma programami. Posiada rozbudowany pakiet bibliotek, jest czytelny i intuicyjny. Ponadto wspiera paradygmaty programowania takie jak obiektowość czy funkcyjność. Posiada w pełni dynamiczny system typów i automatyczne zarządzanie pamięcią. Te udogodnienia pozwoliły nam w największym stopniu skupić się na poprawność i skuteczności algorytmu, a nie na samej kompilacji kodu.

### Biblioteki

* **Matplotlib**

Pakiet umożliwiający generowanie wykresów 2D. W pracy został wykorzystany do zobrazowania konkretnych osobników populacji - rozmieszczenia punktów problemu oraz sposobu połączenia przystanków między nimi.

* **Numpy**

Moduł Numpy jest podstawowym zestawem narzędzi dla języka Python umożliwiającym zaawansowane obliczenia matematyczne, w szczególności do zastosowań naukowych. W projekcie została wykorzystana jedna z funkcji tej biblioteki do losowego wyboru uwzględniając prawdopodobieństwo.

* **Math**

Biblioteka stricte matematyczna. Wspomagaliśmy się nią w niektórych, bardziej naukowych obliczeniach (np. wyznaczając wartość bezwzględną lub pierwiastkując).

* **Random**

Moduł implementuje generatory liczb pseudolosowych o różnych rozkładach. Umożliwia wybór losowej liczby całkowitej z zadanego zakresu jak również losowego elementu z podanej sekwencji przy założeniu równomiernego rozkładu prawdopodobieństwa.

* **Time**

Biblioteka służąca do wykorzystania czasu systemowego. W naszej pracy została wykorzystana głównie w testach do obliczenia czasu działania.

* **Os**

W module os znajduje się szereg funkcji umożliwiających pozyskiwanie informacji na temat plików i folderów znajdujących się na dysku twardym komputera a także tworzenie i usuwanie tychże plików. Wykorzystanie go pomogło nam w uporządkowaniu rezultatów testów i solucji.

* **Mutliprocessing**

Biblioteka umożliwiająca dzielenie wykonywanego programu na wiele procesów. Wykorzystaliśmy ją do generowania wykresów w celu umożliwienia algorytmowi wykonywania się w trakcie generacji wykresów.

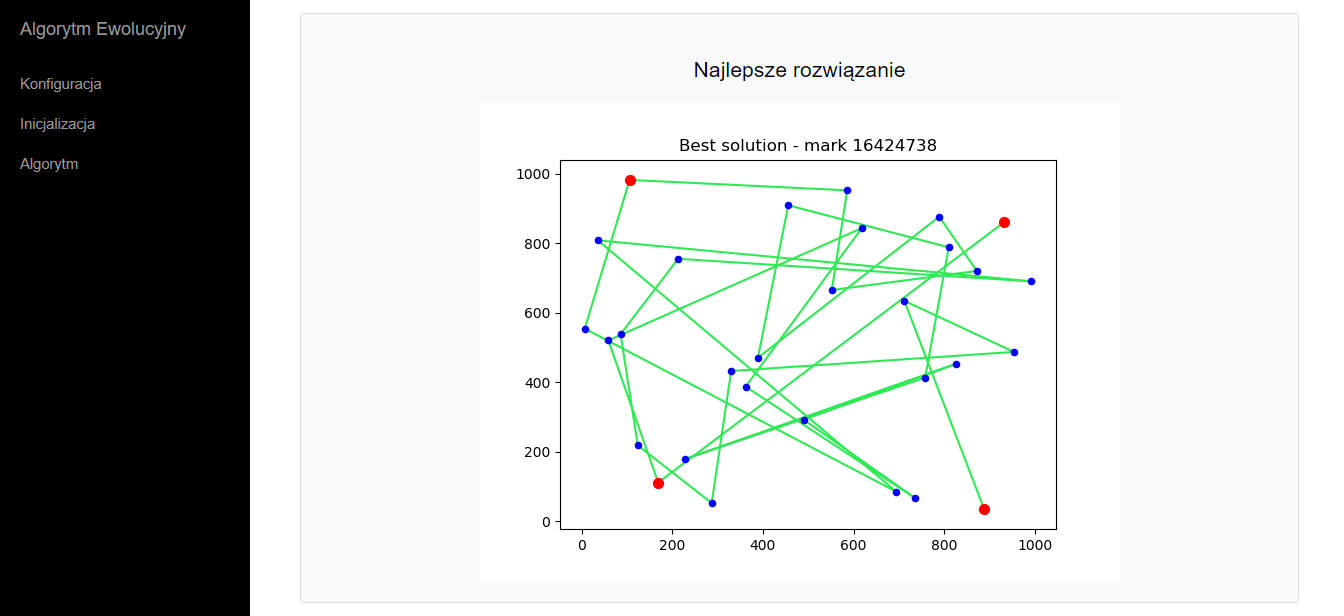
## GUI

Opracowaliśmy również dla naszego algorytmu aplikację graficzną, umożliwiającą konfigurowanie parametrów algorytmu, inicjalizację problemu oraz generowanie kolejnych rozwiązań. Na końcu możemy dokonać graficznego przeglądu wyniku - narysowana jest funkcja celu, najlepsze rozwiązanie, rozwiązania początkowe oraz końcowe.

Do stworzenia aplikacji wykorzystaliśmy technologię **webową**. Za elementy logiki odpowiada serwer webowy zaimplementowany w technologii **Flask**. Za graficzny aspekt odpowiedzialne są połączone siły HTML’a, CSS’a oraz JavaScript’u. Bardzo pomocne były również gotowe elementy dostarczone przez Bootstrap’a.



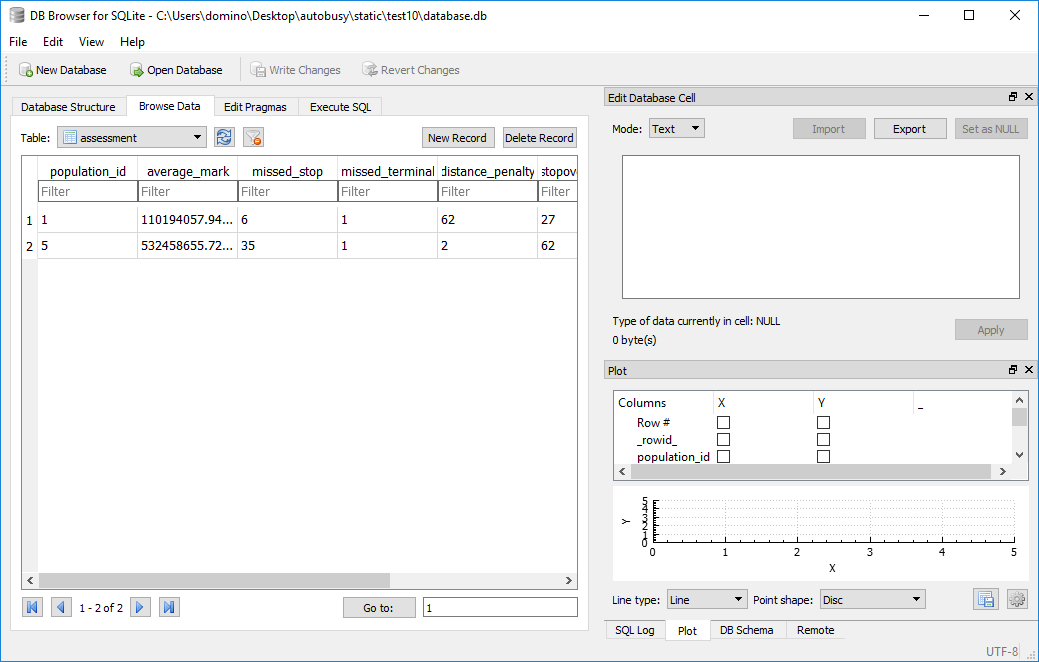
*Rys. 3. Ekran konfiguracji algorytmu*

**

*Rys. 4. Ekran przeglądu algorytmu*

## Baza Danych

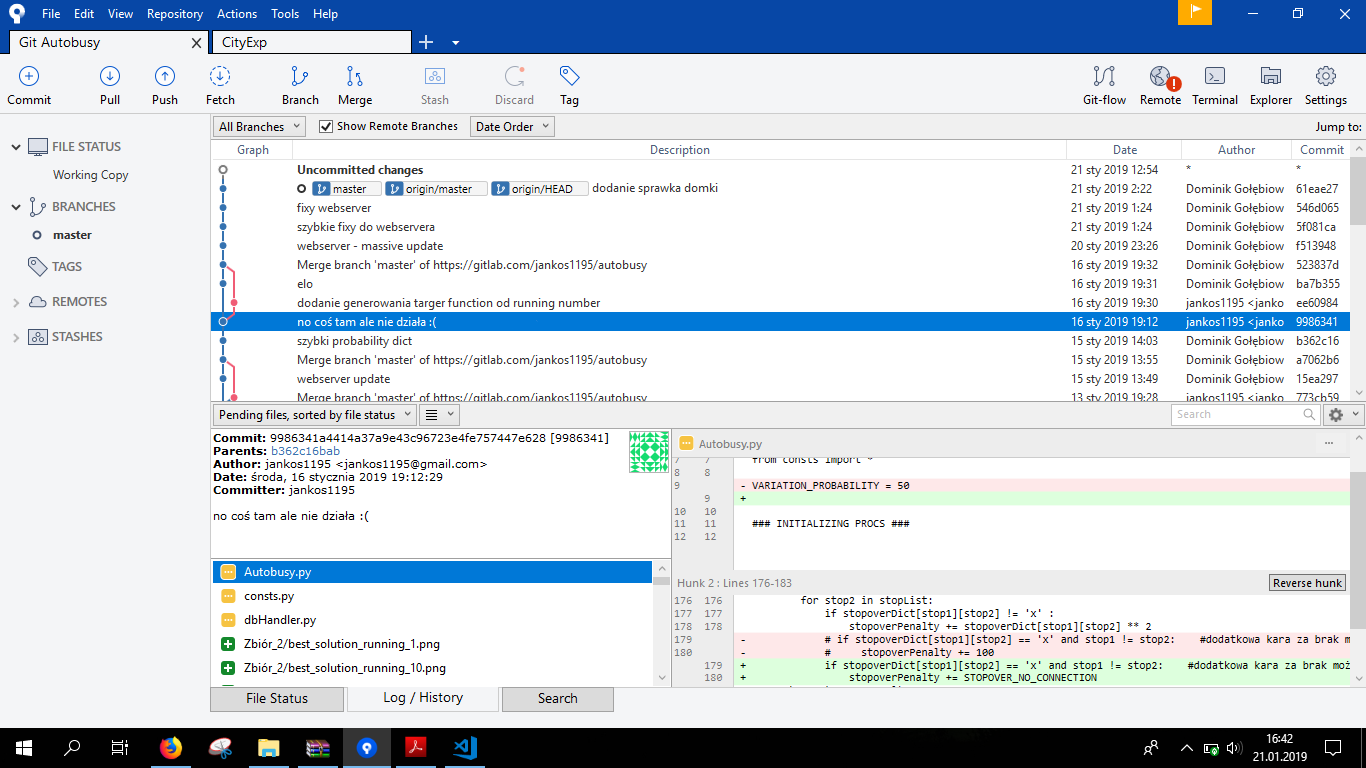
Do przechowywania rozwiązań, parametrów algorytmu oraz współczynników oceniających wykorzystaliśmy bazę danych **sqlite**. Pozwala ona w prosty sposób przechowywać dane w jednym pliku. Do komunikacji algorytmu z tą bazą wykorzystaliśmy pythonową bibliotekę sqlite3 oraz klasę DBHandler z metodami, w których znajdują się odpowiednie kwerendy. Do podglądu danych wykorzystaliśmy program DB Browser.

  
*Rys. 5. Przegląd bazy danych*

## GitLab

Obecnie często wykorzystywany przez programistów jest menadżer repozytoriów oparty o architekturę Gita. Służy do kontroli wersji i umożliwia pracę nad danym projektem wielu osobom jednocześnie. GitLab jest bardzo pomocny zarówno dla deweloperów, jak i również dla innych członków zespołu, zapewniając wszystkim pracownikom pojedynczą, unikatową platformę roboczą.

Wykorzystywaliśmy bezpłatną aplikację **Sourcetree,** która upraszcza obsługę repozytoriów Git, co pozwala skupić się na programowaniu. Prosty graficzny interfejs użytkownika Git umożliwia wizualizację i zarządzanie repozytoriami.



*Rys. 6. Fragment drzewa obrazującego pracę nad algorytmem z wykorzystaniem Sourcetree*

# 

# 

# Metodyka testów

Po zakończeniu implementacji algorytmu przystąpiliśmy do szeregu testów w celu wysterowania algorytmu parametrami wejściowymi. Postanowiliśmy najpierw oddzielnie zbadać wpływ konkretnych parametrów na wynik funkcji celu przy stałych pozostałych właściwościach. Zbadaliśmy funkcję celu w zależności od:

* współczynnika kary za pominięte przystanki
* współczynnika kary za ilość przesiadek
* współczynnika kary za brak wspólnego przystanku dwóch linii
* współczynnika kary za niezadowolenie u pasażerów
* współczynnika kary za zbyt długą trasę oraz odchylenie standardowe długości linii

Zbadaliśmy także wpływ intensywności poszczególnych współczynników wariacji i mutacji na przebieg funkcji celu, jej zbieżność oraz wartość jej najlepszego rozwiązania:

* wpływ procentowego współczynnika wariacji
* wpływ wagi działania mutacji wstawiającej losowe przystanki
* wpływ wagi działania mutacji zmieniającej losowe przystanki
* wpływ wagi działania mutacji usuwającej losowe przystanki
* wpływ wagi działania mutacji zamieniającej losową nitkę
* wpływ wagi działania mutacji usuwającej losową nitkę
* wpływ nie działania żadnej mutacji

Każdy z powyższych testów został przeprowadzony na tym samym zbiorze (problemie zainicjowanym):

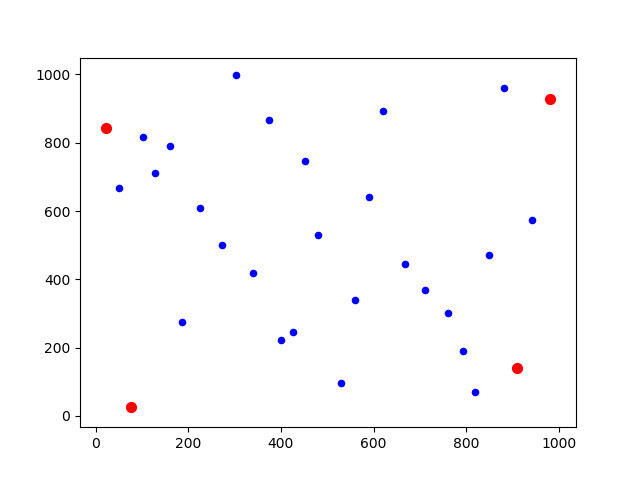
* liczba przystanków - 30
* rozmiar mapy - 1000x1000 jednostek czasowych
* minimalny dystans między każdymi dwoma przystankami - 25 jednostki czasowe
* maksymalna ilość ludzi na danym przystanku kierująca się na inny - 4
* ilość przystanków, które będą pełniły funkcję pętli autobusowej - 4
* ilość osobników w danej populacji - 10
* ilość generacji dla danego testu - 100

Domyślne współczynniki funkcji celu (cechy nietestowane były ustawiane na stałą wartość):

* waga kary za pominięte przystanki - 1000, oraz jej potęga - 2
* waga kary za pominięte pętle autobusowe - 1500, oraz jej potęga - 2
* waga kary za wiele przesiadek - 100
* waga kary za brak połączeń między danymi nitkami - 50
* waga kary za niezadowolenie pasażerów - 10
* waga współczynnika sumy długości nitek - 25
* waga współczynnika odchylenia standardowego długości nitek - 50
* procentowy udział krzyżowań wariacji - 70
* waga współczynników mutacji - 0,025
* waga współczynnika braku mutacji - 0,875

# Testy

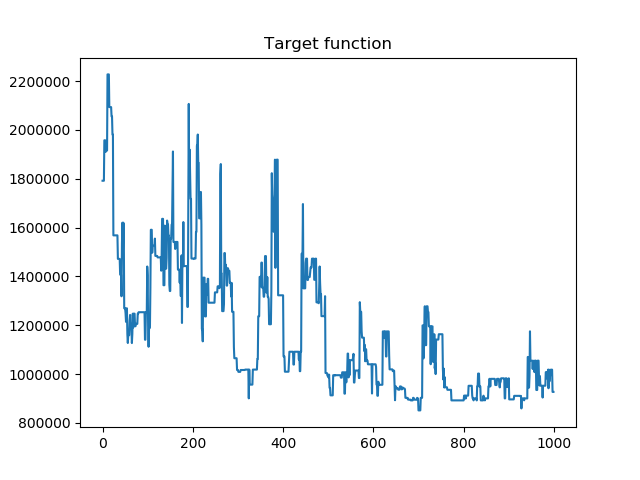
## Problem, na którym zostały przeprowadzone testy



*Rys. 7. Wygenerowany problem do testowania algorytmu*

## Wynik działania algorytmu przy ustawieniach domyślnych

*Rys. 8. Najlepsze rozwiązanie dla domyślnych ustawień*



*Rys. 9. Przebieg funkcji celu dla domyślnych ustawień*

## Wpływ zmian parametrów funkcji celu

**Zależność funkcji celu od współczynnika kary za pominięte przystanki**

Ilość przystanków jest określana przez dwa parametry:

* kara za pominięte pętle
* kara za pominięte przystanki niebędące pętlami

Nie rozdzielamy tego na dwa przypadki, ponieważ w naszych testach nie wykazywało to praktycznie żadnych zmian - zawsze wszystkie pętle były uwzględniane w ostatecznym rozwiązaniu.

*Tabela 1. Zależność funkcji celu od współczynnika kary za pominięte przystanki*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynników  **MISSED\_STOP\_WEIGHT, MISSED\_STOP\_POWER** | wartość współczynnika  **MISSED\_TREMINAL\_WEIGHT, MISSED\_TREMINAL\_POWER** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązanie [%] | czas wykonywania algorytmu [s] |
| 100 , 1 | 150, 1 | **1215827** | **694,26882** | **164** |
| 100 , 2 | 150, 2 | **1087570** | **610,48179** | **126** |
| 1000, 3 | 1500, 3 | **1206067** | **687,89286** | **231** |

*Tabela 2. Zależność funkcji celu od współczynnika kary za ilość przesiadek*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **STOPOVER\_PENALTY\_WEIGHT** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| 10 | **1320290** | **762,51184** | **239** |
| 1000 | **1087992** | **610,75747** | **232** |
| 10000 | **1015298** | **563,26833** | **235** |

*Tabela 3. Zależność funkcji celu od współczynnika kary za brak wspólnego przystanku*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **STOPOVER\_NO\_CONNECTION** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| 5 | **1028762** | **572,06402** | **236** |
| 500 | **1017512** | **564,71468** | **230** |
| 5000 | **1284065** | **738,84697** | **228** |

*Tabela 4. Zależność funkcji celu od współczynnika kary za niezadowolenie u pasażerów*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **UNSATISFIED\_PPL\_PENALTY\_WEIGHT** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu |
| 1 | **1658626** | **983,53813** | **241** |
| 100 | **1321599** | **763,36698** | **231** |
| 1000 | **1364887** | **791,64593** | **248** |

*Tabela 5. Zależność funkcji celu od współczynnika kary za zbyt długą trasę*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **DISTANCE\_PENALTY\_WEIGHT** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **2** | **153075** | **0** | **220** |
| **20** | **640639** | **318,51315** | **192** |
| **200** | **5040420** | **3192,778** | **175** |

Tabela 6. *Zależność funkcji celu od współczynnika kary za odchylenie standardowe długości linii*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **DEVIATION\_WEIGHT** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **2** | **643234** | **320,20839** | **178** |
| **20** | **559128** | **265,26409** | **248** |
| **200** | **1597637** | **943,69557** | **244** |

## Wpływ intensywności operatorów genetycznych

Podczas testowania poszczególnych współczynników decydujących o prawdopodobieństwie wystąpienia danej konkretnej mutacji, zostały zmieniane dwa parametry - testowany oraz współczynnik odpowiadający za brak jakiejkolwiek mutacji. Był to jedyny sposób, aby zachować sumowanie się wszystkich współczynników do jedynki.

*Tabela 7. Wpływ procentowego współczynnika wariacji*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **VARIATION\_PROBABILITY** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **20** | **557437** | **264,1594** | **235** |
| **50** | **989352** | **546,31847** | **235** |
| **80** | **1271433** | **730,59481** | **219** |

*Tabela 8. Wpływ wagi działania mutacji wstawiającej losowe przystanki*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynnika  **MUT\_INSERT\_STATIONS** | wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **0.01** | 0.89 | **1074682** | **602,06239** | **244** |
| **0.2** | 0.7 | **1191892** | **678,6327** | **306** |
| **0.5** | 0.4 | **11322234** | **7296,5272** | **325** |

*Tabela 9. Wpływ wagi działania mutacji zmieniającej losowe przystanki*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynnika  **MUT\_EXCHANGE\_STATIONS** | wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **0.01** | 0.89 | **1073362** | **601,20007** | **287** |
| **0.2** | 0.7 | **1525122** | **896,32337** | **401** |
| **0.5** | 0.4 | **1183314** | **673,02891** | **354** |

*Tabela 10. Wpływ wagi działania mutacji usuwającej losowe przystanki*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynnika  **MUT\_REMOVE\_STATIONS** | wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **0.01** | 0.89 | **1089986** | **612,0601** | **217** |
| **0.2** | 0.7 | **1005700** | **556,9982** | **235** |
| **0.5** | 0.4 | **1271029** | **730,33088** | **239** |

*Tabela 11. Wpływ wagi działania mutacji zamieniającej losową nitkę*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynnika  **MUT\_EXCHANHE\_THREAD** | wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **0.01** | 0.89 | **1669377** | **990,56149** | **248** |
| **0.2** | 0.7 | **966141** | **531,15532** | **224** |
| **0.5** | 0.4 | **1017071** | **564,42659** | **200** |

*Tabela 12. Wpływ wagi działania mutacji usuwającej losową nitkę*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| wartość współczynnika  **MUT\_REMOVE\_THREAD** | wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| **0.01** | 0.89 | **1075084** | **602,325** | **242** |
| **0.2** | 0.7 | **1090783** | **612,58076** | **235** |
| **0.5** | 0.4 | **946184** | **518,11792** | **228** |

*Tabela 13. Wpływ nie działania żadnej mutacji*

|  |  |  |  |
| --- | --- | --- | --- |
| wartość współczynnika  **MUT\_NOTHING** | najlepsza uzyskana wartość funkcji celu | porównanie z najlepszym rozwiązaniem [%] | czas wykonywania algorytmu [s] |
| 1 | **878074** | **473,62339** | **322** |

## Przykłady rozwiązań dla mniejszych problemów:

# 

# Wnioski

Podczas testowania współczynników algorytmu, najlepsze rezultaty otrzymaliśmy przy zmianie współczynnika odpowiadającego za odchylenie standardowe. Dzięki zwiększeniu jego wartości otrzymaliśmy rozwiązania o podobnej długości, które znacznie lepiej dopasowywały się do problemu. W tabelach umieściliśmy również informacje o ile procent jest gorsze dane rozwiązanie od tego najlepszego.  
 Algorytm bardzo dobrze radzi sobie z instancjami o niewielkiej złożoności, przy liczbie przystanków około 15 jest w stanie dosyć szybko znaleźć rozwiązanie zbliżone do optymalnego. Przy większych problemach - około 50 przystanków, podczas testowania funkcja celu minimalizowała się do pewnego momentu, zbiegając do minimum lokalnego. Podczas kolejnych prób zwiększaliśmy współczynniki mutacji, tak aby zmniejszyć zbieżność, co nieco poprawiło rezultaty.

Bardzo duża ilość współczynników wpływających zarówno na funkcję celu jak i na przebieg mutacji i wariacji, spowodowała, że dosyć ciężko było nam skalibrować algorytm. Dopiero po przeprowadzeniu kilkudziesięciu testów udało się doprowadzić do minimalizacji i zaczęliśmy otrzymywać dobre rozwiązania.

Algorytm ewolucyjny jest bardzo dobrym narzędziem do znajdowania optymalnych rozwiązań w realnych zastosowaniach. Definicja książkowa mówi, że charakteryzuje się on zwykle bardzo dużą liczbą zmiennych dyskretnych lub ciągłych, dużą złożonością przestrzeni poszukiwań, posiada wiele ograniczeń i celów, które czasami mogą być nawet ze sobą sprzeczne. Wszystkie te cechy idealnie wpasowują się w problem, który optymalizowaliśmy, a rezultat działania zaimplementowanego przez nas programu potwierdza słuszność wyboru poprawnej ścieżki optymalizacji, czyli tym samym odpowiedniego algorytmu.

Zdecydowanie był to jeden z najlepszych projektów na tych studiach. Jesteśmy zachwyceni i niezwykle dumni z rezultatów jakie osiągnęliśmy oraz, że mogliśmy w takowym projekcie uczestniczyć. Nie spodziewaliśmy się, że posiadamy tak proste a zarazem tak potężne narzędzie optymalizacyjne, które pozwala dobrać najlepsze parametry bardzo skomplikowanego problemu.