



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W
KRAKOWIE

WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ

Projekt semestralny

*Modelowanie świateł na skrzyżowaniu: „Algorytmy Optymalizacji i ich
wpływ na czas przejazdu oraz płynność ruchu”*

Autorzy pracy: Wojciech Jankowski, Jakub Głód, Mateusz Kuc

Kierunek studiów: Automatyka i Robotyka

Kraków, 2024

Spis treści

Rozdział 1. Wprowadzenie	3
1.1 Wstęp	3
1.2 Cel i zakres projektu	3
1.3 Motywacja do wyboru tematu	3
Rozdział 2. Szczegółowy opis rozpatrywanego zagadnienia	4
2.1 Sformułowanie problemu	4
2.2 Założenia i uproszczenia modelu	4
2.3 Kryteria oceny efektywności	5
2.4 Początkowe ustawienie świateł na skrzyżowaniu	5
2.5 Model matematyczny problemu	5
2.5.1 Zmienne decyzyjne	5
2.5.2 Funkcja celu	5
2.5.3 Funkcja składowa g funkcji celu	6
2.5.4 Funkcja przejścia	6
2.5.5 Funkcja kary	6
Rozdział 3. Opis zaimplementowanych algorytmów	7
3.1 Algorytm genetyczny - zasady działania	7
3.2 Reprezentacja rozwiązania	8
3.3 Tworzenie populacji początkowej	8
3.4 Operatory algorytmu genetycznego	9
3.4.1 Selekcja	9
3.4.2 Mutacja	10
3.4.3 Krzyżowanie	10
3.4.4 Permutacja	11
3.5 Funkcja celu (oceny)	12
3.6 Pseudokod algorytmu genetycznego	12
3.7 Dostosowanie algorytmu do problemu świateł na skrzyżowaniach	13
3.8 Alternatywne podejścia i testowane warianty	13
Rozdział 4. Omówienie zasad działania stworzonej aplikacji	15
4.1 Opis funkcjonalności aplikacji	15
4.2 Instalacja i uruchamianie aplikacji	16
4.3 Wprowadzenie danych wejściowych	16
4.4 Prezentacja wyników i ich interpretacja	17
4.5 Rozwiązywanie potencjalnych problemów w użytkowaniu	17
Rozdział 5. Eksperymenty i testowanie aplikacji	18
5.1 Opis metodyki testów i eksperymentów	18
5.2 Analiza i interpretacja różnych przypadków testowych - cz.1	18
5.2.1 Badanie wpływu wielkości populacji na wartość funkcji celu dla rozwiązań losowych	18
5.2.2 Badanie wpływu wielkości populacji na wartość funkcji celu dla rozwiązań domyślnych (zerowych)	19

5.2.3	Badanie wpływu mutacji i permutacji na przebieg algorytmu - porównanie prawdopodobieństw 0% i 20%	20
5.2.4	Uzasadnienie dodania parametru umożliwiającego dodawanie 5 losowych rozwiązań do krzyżowania	21
5.2.5	Zasada działania funkcji kary oraz zobrazowanie tego na wykresach	22
5.2.6	Próba znalezienia parametrów algorytmu, które pozwolą na powtarzalne uzyskiwanie najlepszego rozwiązania przy sensownym nakładzie czasowym	23
5.3	Analiza i interpretacja różnych przypadków testowych - cz.2	24
5.3.1	Test działania podstawowego algorytmu na przypadku trywialnym	24
5.3.2	Weryfikacja warunku stopu na podstawie braku poprawy	25
5.3.3	Sprawdzenie warunku stopu osiągnięcia wartości progowej	25
5.3.4	Test warunku stopu na podstawie maksymalnej liczby iteracji	26
Rozdział 6. Podsumowanie		27
6.1	Ocena osiągnięcia założonych celów	27
6.2	Problemy napotkane podczas realizacji	27
6.3	Kierunki dalszego rozwoju projektu	27
Dodatek A. Prezentacja pracy w grupie na różnych etapach tworzenia projektu		29
7.1	Omówienie realizowanych zadań wśród członków zespołu	29
Literatura		30

Wprowadzenie

1.1 Wstęp

Optymalizacja sygnalizacji świetlnej jest kluczowym elementem w zarządzaniu ruchem drogowym, szczególnie na skrzyżowaniach o dużym natężeniu ruchu. Dzięki wykorzystaniu nowoczesnych algorytmów optymalizacyjnych możliwe jest zwiększenie efektywności systemu transportowego, co przekłada się na zmniejszenie korków, emisji spalin oraz poprawę bezpieczeństwa. W niniejszym projekcie analizowane jest zastosowanie algorytmu genetycznego do optymalizacji sekwencji świateł na skrzyżowaniu dwóch dróg dwujezdniowych. Zwiększenie płynności ruchu i skrócenie czasu oczekiwania pojazdów stanowi kluczowy cel tego projektu, który ma na celu poprawę efektywności funkcjonowania całego systemu sygnalizacji świetlnej.

1.2 Cel i zakres projektu

Celem projektu jest stworzenie modelu skrzyżowania oraz zaprojektowanie optymalnej sekwencji przełączeń sygnalizacji świetlnej. Model uwzględnia 8 sygnalizatorów (s1 - s8), odpowiadających poszczególnym pasom ruchu, z których każdy stan jest reprezentowany przez ośmioelementowy wektor. Projekt zakłada, że w każdej iteracji algorytmu analizowane są różne konfiguracje ustawień świateł, a spośród nich wybierana jest ta, która w największym stopniu minimalizuje funkcję celu, zapewniając tym samym jak najlepszą płynność ruchu i minimalny czas oczekiwania pojazdów.

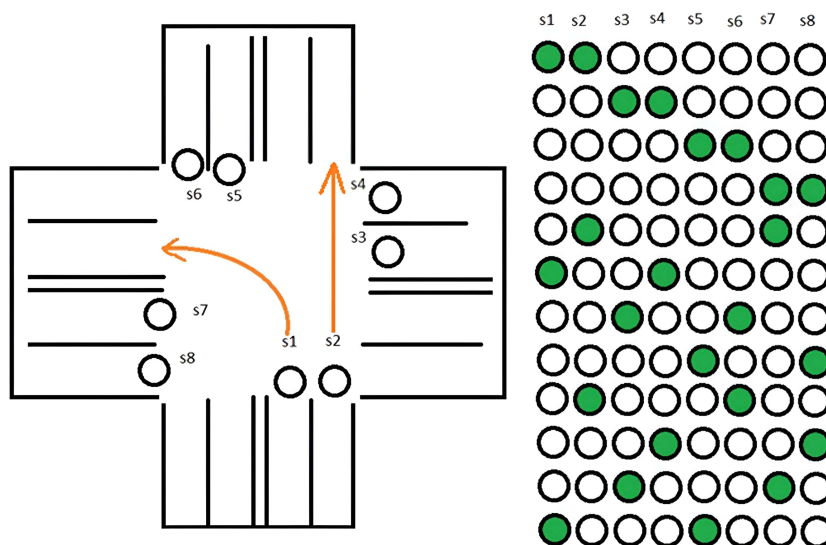
1.3 Motywacja do wyboru tematu

Początkowy pomysł na realizację projektu obejmował stworzenie modelu systemu optymalizującego ruch na skrzyżowaniu ul. Czarnowiejskiej i al. Adama Mickiewicza, które charakteryzuje się dużym natężeniem ruchu oraz skomplikowaną strukturą drogową. Jednak z uwagi na bardzo dużą złożoność tego zagadnienia i trudności związane z danymi rzeczywistymi, zdecydowano się na bardziej ogólny przypadek skrzyżowania dwóch dróg dwujezdniowych, który jest prostszy do modelowania, a jednocześnie pozwala na przeprowadzenie bardziej efektywnej analizy. Taka modyfikacja tematu umożliwia skupienie się na bardziej precyzyjnej optymalizacji sygnalizacji świetlnej, z możliwością łatwiejszego testowania algorytmu na uproszczonym, ale wciąż realistycznym modelu.

Szczegółowy opis rozpatrywanego zagadnienia

2.1 Sformułowanie problemu

Problem polega na optymalnym sterowaniu sygnalizacją świetlną na skrzyżowaniu, w celu minimalizacji czasu przejazdu i uniknięcia nadmiernego gromadzenia się pojazdów na poszczególnych pasach. Istotne jest znalezienie takiej sekwencji świateł, która zminimalizuje sumaryczny czas oczekiwania samochodów na przejazd.



Rysunek 1: Graficzne przedstawienie omawianego problemu

2.2 Założenia i uproszczenia modelu

Poniżej zostały zaprezentowane przyjęte założenia i uproszczenia dla modelu:

- Symulacja dotyczy wyłącznie samochodów.
- Pominęto ruch pieszy, rowerowy, czy tramwajowy na skrzyżowaniu.
- Nie uwzględniana jest zmiana pasów podczas dojazdu pojazdów do skrzyżowania.
- Każdy pojazd znika natychmiast po wykonaniu manewru skrętu, brak jest tu możliwości zatkania się skrzyżowania.
- Nie uwzględnia się skrętów w prawo, ponieważ zakłada się ich bezkolizyjność.
- Wybrany został konkretny typ skrzyżowania: skrzyżowanie się 2 dróg dwujezdniowych.
- Ograniczona liczba samochodów w symulacji: max. 20 pojazdów.

2.3 Kryteria oceny efektywności

Efektywność modelu oceniana jest na podstawie:

- Łącznego czasu oczekiwania pojazdów na czerwonym świetle.
- Liczby kroków potrzebnych do opróżnienia skrzyżowania.
- Funkcji kary, która penalizuje długotrwałe oczekiwanie.

2.4 Początkowe ustawienie świateł na skrzyżowaniu

Na początku symulacji wszystkie światła mogą być dowolnie skonfigurowane, jednak początkowa konfiguracja nie wpływa znacząco na wynik końcowy – algorytm dynamicznie dostosowuje stany świateł w kolejnych iteracjach.

2.5 Model matematyczny problemu

2.5.1 Zmienne decyzyjne

- i – etap decyzyjny (wybór sekwencji świateł),
- j – pas ruchu, $j \in \{1, 2, \dots, 8\}$,
- c_1, c_2, \dots, c_{12} – wektory stanów świateł (każdy ośmioelementowy, np. $[1, 0, 1, 0, 0, 1, 0, 0]$),
- $x^{(i)}$ – zmienna decyzyjna przyjmująca wartość jednego z wektorów c_1, c_2, \dots, c_{12} ,
- $N^{(i)}$ – wektor określający liczbę pojazdów na pasach w i -tym etapie (wartości od 0 do 20),
- $T_j^{(i)}$ – czas oczekiwania samochodów w i -tym etapie na j -tym pasie.

2.5.2 Funkcja celu

Funkcja celu minimalizuje sumę kroków oraz czas oczekiwania pojazdów:

$$f(x) = \sum_{i=1}^K \left(d + a \cdot g(N^{(i)}, T^{(i)})^b \right) + c \cdot \text{sum}(N^{(-1)})$$

Gdzie:

- a oraz b to odpowiednio współczynniki liniowy i wykładniczy wpływu funkcji g (nieujemne liczby rzeczywiste).
- c to współczynnik zwiększenia funkcji celu w zależności od tego ile pojazdów pozostało na skrzyżowaniu po wykonaniu ostatniego kroku - czyli suma pojazdów na wszystkich pasach po realizacji rozwiązania - moment określony jako N^{-1}
- d to wartość dodawaną w każdym kroku działania algorytmu.

Ustalanie wartości parametrów a , b , c , d również oddajemy w ręce użytkownika, ale sugerujemy wartości odpowiednio: $a=0.1$, $b=1$, $c=5$, $d=1$

2.5.3 Funkcja składowa g funkcji celu

Funkcja g penalizuje długie oczekiwanie na czerwonym świetle. Natomiast wartość $T_j^{(i)}$ tej funkcji określa się w następujący sposób:

$$T_j^{(i)} = \begin{cases} 0 & \text{jeśli przejechał w i-tym etapie} \\ T_j^{(i)} + 1 & \text{jeśli nie przejechał w i-tym etapie} \end{cases}$$

$$g(N^{(i)}, T^{(i)}) = \sum_j N_j^{(i)} \cdot T_j^{(i)}$$

Funkcja g zwiększa swoją wartość wraz z długością wystawiania czerwonego światła na jednym sygnalizatorze

2.5.4 Funkcja przejścia

Funkcja przejścia określająca, jak liczba pojazdów na pasach skrzyżowania zmienia się z etapu na etap, uwzględniając pojazdy, które przejechały skrzyżowanie w poprzednim etapie. W tej interpretacji rozumiana jako iloczyn logiczny wektorów.

$$N^{(i)} = N^{(i-1)} - x^{(i-1)} - (x^{(i-1)} * x^{(i-2)})$$

2.5.5 Funkcja kary

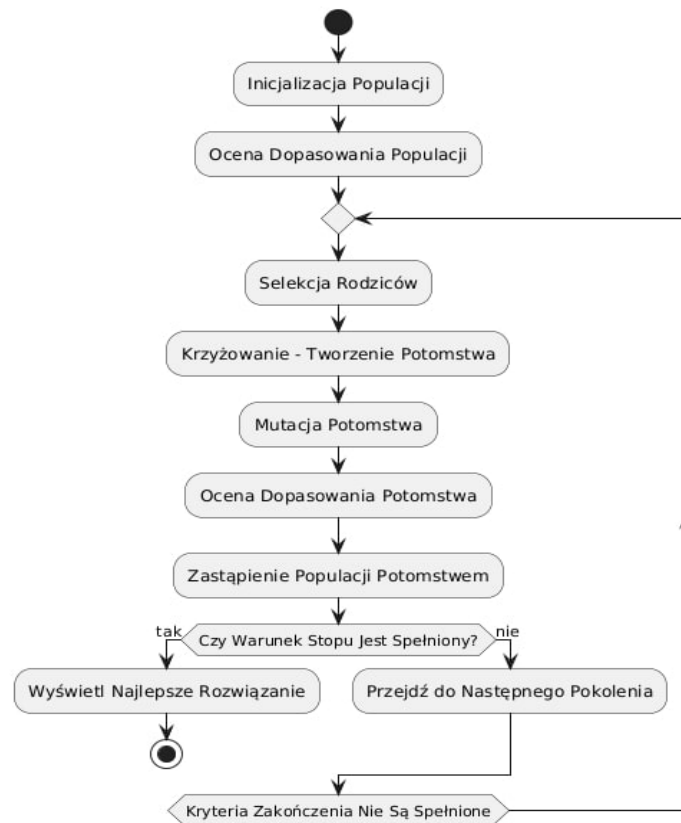
Konieczność dodania funkcji kary została zauważona podczas przeprowadzania pierwszych testów algorytmu, ponieważ w niektórych przypadkach algorytm, zamiast dążyć do wychodzenia z minimów lokalnych i optymalizować rozwiązanie pod kątem zmniejszenia liczby pojazdów pozostających na skrzyżowaniu po zakończeniu symulacji, koncentrował się wyłącznie na minimalizacji wartości funkcji celu wynikającej z funkcji g .

Opis zaimplementowanych algorytmów

3.1 Algorytm genetyczny - zasady działania

Algorytm genetyczny to metoda optymalizacji inspirowana teorią ewolucji biologicznej. Jego działanie opiera się na zasadach selekcji naturalnej, krzyżowania i mutacji. Algorytm rozpoczyna się od generacji początkowej populacji losowych rozwiązań (osobników). Każdy osobnik jest oceniany za pomocą funkcji celu, która odzwierciedla jakość rozwiązania. Następnie najlepiej przystosowane osobniki są wybierane do reprodukcji, tworząc nowe pokolenia. Celem jest znalezienie optymalnego rozwiązania poprzez iteracyjne poprawianie jakości populacji.

W kontekście optymalizacji sygnalizacji na skrzyżowaniach, algorytm genetyczny umożliwia znalezienie optymalnych sekwencji zmiany świateł, co prowadzi do minimalizacji łącznego czasu oczekiwania pojazdów i poprawy płynności ruchu.



Rysunek 2: Przykładowy schemat blokowy działania prostego algorytmu genetycznego

3.2 Reprezentacja rozwiązania

Klasa Solution: W tej klasie znajduje się lista solution, która przechowuje rozwiązanie. Jest to lista długości length, której elementy są liczbami z zakresu od 0 do 11. Każdy element tej listy odpowiada jednej z 12 możliwych kombinacji stanów sygnalizatorów, które są zdefiniowane w słowniku possible_combinations.

```
self.solution = [None for _ in range(length)] # lista do przechowywania rozwiązania
self.quality = 999999999 # duża i charakterystyczna liczba, ale nie nieskończoność
self.length = length
self.kara = 0
self.nadmiar = -1
```

Rysunek 3: Struktura konstruktora klasy Solution

Dodatkowe własności:

- **quality:** Każde rozwiązanie cechuje się swoją jakością obliczoną przez funkcję celu.
- **kara:** W pewnych wariacjach naszej implementacji algorytmu znajduje się tutaj wartość funkcji kary dla danego rozwiązania (związanej ze zbyt częstą zmianą świateł).
- **nadmiar:** Liczba pojazdów, które nie przejadą przez skrzyżowanie po zastosowaniu danego rozwiązania.

3.3 Tworzenie populacji początkowej

Populacja początkowa powinna być zróżnicowana, aby algorytm miał większą szansę na skuteczne znalezienie optymalnych rozwiązań w przestrzeni rozwiązań. W omawianym algorytmie proces tworzenia początkowej populacji realizowany jest przez metodę get_population().

Metoda get_population() jest odpowiedzialna za inicjalizację początkowej populacji rozwiązań. Każde rozwiązanie w populacji jest obiektem klasy Solution, który posiada określoną długość i wartości elementów. Parametry tej metody pozwalają na elastyczne tworzenie populacji, w zależności od tego, czy chcemy zacząć od rozwiązania "domyślne-go" (tzw. "dummy") czy od rozwiązań losowych.

Działanie metody get_population():

```
def get_population(self, quantity, length=12, dummy=0):
    self.population = [Solution(length) for _ in range(quantity)]
    if dummy == 1:
        for el in self.population:
            el.dummy()
    else:
        for el in self.population:
            el.randomize()
```

Rysunek 4: Struktura metody get_population()

- Liczba elementów populacji (*quantity*) – określa liczbę rozwiązań w populacji.

- Długość rozwiązania (*length*) – ustala liczbę elementów, które składają się na każde rozwiązanie. Można narzucić tę długość, lecz w przypadku braku takiego działania, algorytm dobiera ją samodzielnie w sposób umożliwiający opuszczenie skrzyżowania przez niemal wszystkie auta.
- Parametr *dummy* – decyduje, w jaki sposób będą inicjowane poszczególne rozwiązania:
 - Jeśli *dummy* = 1, każde rozwiązanie będzie "dummy", tj. wszystkie elementy w rozwiązaniu będą ustawione na 0. Tego typu rozwiązania są traktowane jako puste lub domyślne.
 - Jeśli *dummy* = 0, każde rozwiązanie jest inicjowane losowo, dzięki czemu w populacji będą znajdować się różnorodne rozwiązania.

W przypadku, gdy parametr *dummy* jest ustawiony na 0, każde rozwiązanie w populacji jest inicjowane losowo. Aby tego dokonać, wykorzystywana jest metoda *randomize()*, która generuje wartości dla elementów rozwiązania z przedziału od 0 do 11. Te liczby reprezentują możliwe stany sygnalizatorów w rozwiązaniu, a sama metoda odpowiada za przypisanie losowych wartości do każdego elementu rozwiązania. Dzięki temu każda inicjalizacja rozwiązania jest unikalna, co sprzyja różnorodności początkowej populacji.

Jeśli zamiast losowego generowania rozwiązań chcemy stworzyć rozwiązanie "puste" lub domyślne, wykorzystywana jest metoda *dummy()*. Ta metoda ustawia wszystkie elementy rozwiązania na 0, co skutkuje stworzeniem rozwiązania, w którym wszystkie sygnalizatory są wyłączone. Metoda ta jest wykorzystywana w przypadku, gdy chcemy rozpocząć proces od neutralnego punktu startowego, który może później być modyfikowany w trakcie działania algorytmu.

3.4 Operatory algorytmu genetycznego

3.4.1 Selekcja

Selekcja odnosi się do wyboru najlepszych osobników w populacji, które będą poddane krzyżowaniu (i mogą przejść przez mutację). W naszym kodzie selekcja odbywa się poprzez posortowanie populacji według jakości rozwiązania, a następnie wybranie określonej ilości najlepszych osobników do dalszych operacji (25%).

Selekcja jakościowa:

- Po obliczeniu jakości rozwiązania dla każdego osobnika w populacji, selekcja odbywa się poprzez posortowanie populacji według jakości rozwiązania oraz kary (jeśli taka opcja jest włączona): (`self.population.sort(key=lambda x: (x.quality + x.kara), reverse=False)`). Najlepsze rozwiązania (o najniższej sumie jakości i kary) trafiają na początek listy.
- Najlepsze 25% populacji (*crossing_population*) jest wybierane do krzyżowania.

Dodawanie losowych rozwiązań:

- W przypadku, gdy parametr *add_random_5_inside* jest ustawiony na *True*, do wybranej grupy najlepszych osobników dodawanych jest 5 losowych rozwiązań – jeszcze przed przeprowadzeniem krzyżowania (aby wprowadzić więcej różnorodności).

```
# wybierz jakościowo najlepsze 25% populacji do krzyżowania
crossing_population = deepcopy(self.population[:quantity//4])

# dodaj pięć losowych rozwiązań
if add_random_5_inside:
    for _ in range(5):
        random_solution = Solution(length)
        random_solution.randomize()
        crossing_population.append(random_solution)
```

Rysunek 5: Struktura kodu selekcji

3.4.2 Mutacja

Mutacja jest mechanizmem w algorytmie genetycznym, który wprowadza losowe zmiany do rozwiązania, aby zwiększyć różnorodność w populacji i uniknąć zbieżności do lokalnych minimów. W kodzie zostały zaimplementowane dwa typy mutacji :

- **Typ 0:**

Zmiana jednej losowej wartości w rozwiązaniu na nową losową wartość (w zakresie od 0 do 11, ponieważ tyle jest możliwych kombinacji sygnalizatorów).

- **Typ 1:**

Zmiana 1/4 losowych elementów w rozwiązaniu, które zostają przypisane do nowych losowych wartości.

```
def mutation(self, typ=0):
    """
    Metoda mutująca
    0) jedna podmianka
    1) podmiana 1/4 losowych indeksów na losowe wartości
    :return: podmienia w rozwiązaniu wartości na określonych indeksach na inne losowe
    """
    length = len(self.solution)
    if typ == 0:
        idx = random.randint(a=0, length-1)
        self.solution[idx] = random.randint(a=0, b=11) # bo 11 jest kombinacji świateł
    if typ == 1:
        number_of_changes = length // 4
        for _ in range(number_of_changes):
            idx = random.randint(a=0, length-1)
            self.solution[idx] = random.randint(a=0, b=11)
```

Rysunek 6: Struktura kodu mutacji typu "0" i "1"

3.4.3 Krzyżowanie

Krzyżowanie (ang. crossover) łączy dwa rozwiązania (osobników) w celu stworzenia nowych, potencjalnie lepszych rozwiązań. W kodzie dostępne są dwa typy krzyżowania:

- **Typ 0:**

Krzyżowanie jednopunktowe — rozwiązania są dzielone w losowym punkcie i wymieniane częściami.

- **Typ 1:**

Krzyżowanie ”w kratkę” — w którym na przemian wybierane są elementy z dwóch rozwiązań.

Krzyżowanie odbywa się z losowo wybraną opcją, jeśli `typ=0` lub `typ=1` (w zależności od wartości parametru `cros_opt`).

```
if typ == 0:
    bound = random.randint(a:1, self.length - 1)
    new_sol1.solution = self.solution[:bound] + other.solution[bound:]
    new_sol2.solution = other.solution[:bound] + self.solution[bound:]
```

Rysunek 7: Struktura kodu krzyżowania typu ”0”

```
elif typ == 1:
    # Krzyżowanie ”w kratkę”
    new_sol1.solution = [self.solution[i] if i % 2 == 0 else other.solution[i] for i in range(self.length)]
    new_sol2.solution = [other.solution[i] if i % 2 == 0 else self.solution[i] for i in range(self.length)]
```

Rysunek 8: Struktura kodu krzyżowania typu ”1”

3.4.4 Permutacja

Permutacja jest operatorem zbliżonym do mutacji, jednak nie wprowadza losowych wartości do rozwiązania, a jedynie przetasowuje je w określony sposób. Działanie implementacji permutacji zostało opisane w komentarzu do funkcji w poniższym listingu.

```
def permutation(self, typ=0):
    """
    Metoda dokonująca permutacji w obrębie jednego rozwiązania
    :param typ: w zależności od typu:
    0) losuje dwie liczby a, b i podmienia wartości na tych indeksach
    1) losuje dwie liczby a, b i w tym zakresie odwraca kolejność
    ... można chyba do woli tworzyć możliwe permutacje
    :return: nic nie zwraca, modyfikuje to rozwiązanie
    """

    if typ == 0:
        a = random.randint(a:0, self.length - 1)
        b = random.randint(a:0, self.length - 1)
        x = self.solution[a]
        self.solution[a] = self.solution[b]
        self.solution[b] = x

    if typ == 1:
        a = random.randint(a:0, self.length)
        b = random.randint(a:0, self.length)
        if a > b:
            a, b = b, a

        self.solution[a:b] = self.solution[a:b][::-1]
```

Rysunek 9: Struktura kodu permutacji typu ”0” i ”1”

3.5 Funkcja celu (oceny)

Funkcja celu została zrealizowana zgodnie z założeniami – implementacja okazała się dosyć skomplikowana, a jej realizacja wymagała dodania wielu list pomocniczych. Z tego względu kod funkcji nie jest zamieszczony tutaj, lecz udostępniony do wglądu w skrypcie. Realizacja funkcji odbywa się w kodzie za pomocą metody `calculate_solution_quality(solution)` w klasie `Simulation`.

3.6 Pseudokod algorytmu genetycznego

Pseudokod algorytmu genetycznego dla problemu optymalizacji sekwencji świateł na skrzyżowaniu:

- **Inicjalizacja:**
 - Ustawienie parametrów algorytmu (m.in. rozmiar populacji, liczba iteracji, prawdopodobieństwo mutacji i permutacji).
 - Utworzenie początkowej populacji rozwiązań (ustawienia świateł).
 - Każdemu rozwiązaniu przypisanie losowy lub domyślny (zerowy) zestaw ustawień świateł.
- **Obliczanie jakości rozwiązania:**
 - Dla każdego rozwiązania w populacji obliczenie funkcji celu.
 - Przechowanie najlepszego rozwiązania globalnego
- **Selekcja:**
 - Posortowanie populacji według jakości (im mniejsza wartość funkcji celu, tym lepsze rozwiązanie).
 - Wybieranie najlepszego rozwiązania do dalszego krzyżowania (np. 25% najlepszych rozwiązań).
- **Krzyżowanie:**
 - Wykonanie krzyżowania między wybranymi rozwiązaniami.
 - Zastosowanie różnych metod krzyżowania (np. punktowe, krzyżowanie w kratkę).
- **Mutacja i permutacja:**
 - Zastosowanie mutacji i permutacji na rozwiązaniach wraz z odpowiednim prawdopodobieństwem z nowo utworzonej populacji.
- **Sprawdzanie warunków stopu:**
 - Jeśli liczba iteracji przekroczy określoną wartość lub jeśli poprawa jakości rozwiązania przestanie być zauważalna, zakończ algorytm.
- **Zakończenie:**
 - Wydrukuj najlepsze rozwiązanie znalezione przez algorytm.
 - Przeanalizuj historię funkcji celu.

3.7 Dostosowanie algorytmu do problemu świateł na skrzyżowaniach

- **Reprezentacja rozwiązania:**

Każde rozwiązanie jest reprezentowane jako ciąg kombinacji świateł dla poszczególnych etapów cyklu świetlnego. Kombinacje te są zapisane w postaci wektora, gdzie każda wartość odpowiada jednej z możliwych konfiguracji świateł (np. [1, 1, 0, 0, 0, 0, 0, 0] dla dwóch działających świateł).

- **Funkcja celu:**

Funkcja celu ocenia rozwiązanie w kontekście minimalizacji nadmiaru pojazdów oraz czasu oczekiwania na sygnalizatorach. Jest to główny element, który kieruje procesem optymalizacji. Używa współczynników kary, które penalizują rozwiązania, w których pojazdy oczekują zbyt długo na sygnalizatorach lub w których nie przejeżdżają wszystkie pojazdy.

- **Krzyżowanie:**

Krzyżowanie jest dostosowane do struktury rozwiązania. Dla każdej pary rodziców algorytm tworzy dwóch potomków poprzez wymianę części rozwiązania (np. punktowe krzyżowanie, gdzie podzielony jest ciąg ustawień świateł). Istnieje także możliwość krzyżowania "w kratkę", co pozwala na lepsze łączenie cech dwóch rozwiązań.

- **Mutacja i permutacja:**

Mutacja zmienia ustawienia świateł w wybranym miejscu rozwiązania, a permutacja modyfikuje kolejność ustawień w obrębie rozwiązania. Zastosowanie tych operacji zapewnia różnorodność w populacji i umożliwia algorytmowi unikanie utknięcia w lokalnych minimach.

- **Selekcja i dobór populacji:**

Algorytm selekcjonuje rozwiązania w oparciu o ich jakość. Najlepsze 25% populacji zostaje wybrane do dalszego krzyżowania, co pozwala na koncentrację na najbardziej obiecujących rozwiązaniach. Dodatkowo, do populacji dodawanych jest pięć losowych rozwiązań, co pozwala na zwiększenie różnorodności w algorytmie i może pomóc w poszukiwaniach globalnych optymalnych rozwiązań.

- **Punkty karne:**

Zastosowanie kary w przypadku nieoptymalnych ustawień (np. ciągłej zmiany świateł, co nie pozwala na uzyskanie 'zielonej fali') pozwala na dodatkowe ukierunkowanie algorytmu w stronę rozwiązań bardziej optymalnych pod kątem ilości pojazdów pozostałych na skrzyżowaniu.

3.8 Alternatywne podejścia i testowane warianty

W implementacji wprowadzono kilka modyfikacji bazowego algorytmu genetycznego, w tym dodanie funkcji kary i uwzględnienie jej podczas selekcji. Motywacja wprowadzenia funkcji kary została uzasadniona na etapie jej definicji w Rozdziale 2.

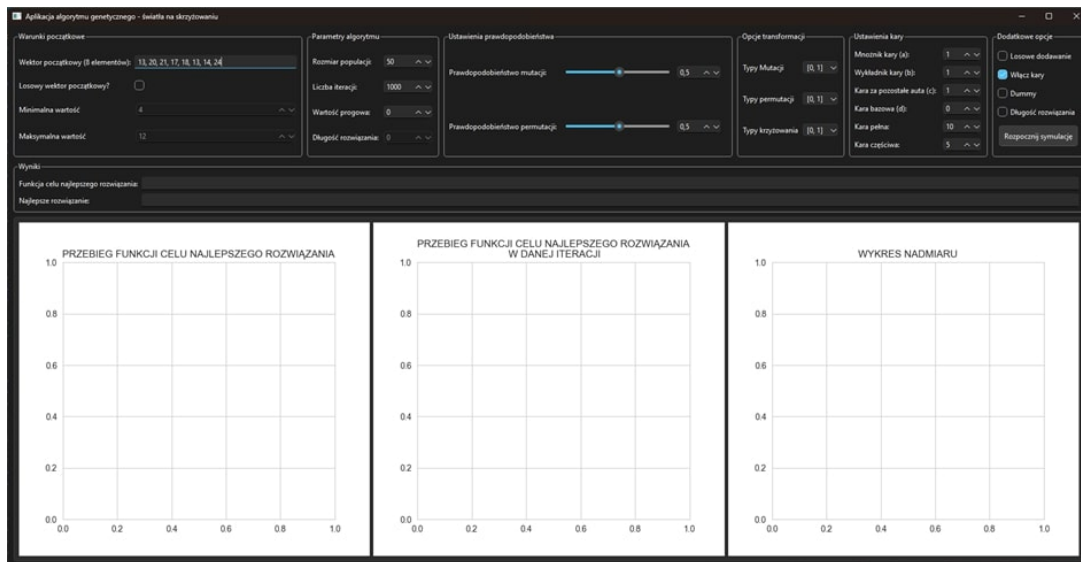
Kolejną z modyfikacji jest możliwość dodania losowych pięciu rozwiązań do krzyżowania – w pewnych przypadkach złośliwych, zwłaszcza podczas rozpoczynania od „zero-

wych” rozwiązań domyślnych z jednoczesnym zagęszczeniem samochodów tylko na niektórych miejscach okazała się ona bardzo pomocna.

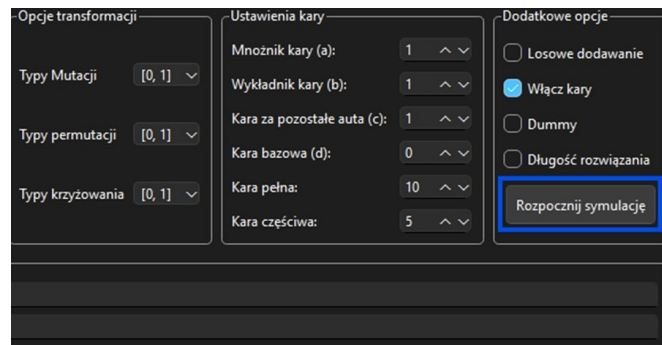
Omówienie zasad działania stworzonej aplikacji

4.1 Opis funkcjonalności aplikacji

Aplikacja pozwala na znalezienie rozwiązania na podstawie ustawionych parametrów z wykorzystaniem zaimplementowanego algorytmu genetycznego. Przykładowe wartości, ustawione domyślnie po uruchomieniu, pozwalają na szybkie sprawdzenie działania programu.



Rysunek 10: Okno aplikacji po uruchomieniu

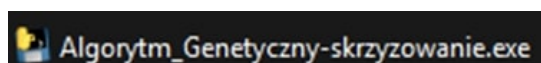


Rysunek 11: Przycisk uruchamiający działanie algorytmu dla wprowadzonych danych

Po najechaniu kursorem na wybrane pole pojawi się dymek z dodatkowymi informacjami na jego temat, natomiast po zakończeniu obliczeń zostanie wyświetlona wartość funkcji celu, najlepsze rozwiązanie oraz trzy wykresy ilustrujące przebieg działania algorytmu.

4.2 Instalacja i uruchamianie aplikacji

Aby uruchomić aplikację, należy odnaleźć i uruchomić plik wykonywalny o nazwie Algorytm_Genetyczny-skrzysowanie.exe. Plik ten zawiera wszystkie niezbędne komponenty umożliwiające poprawne działanie programu. Po jego uruchomieniu aplikacja automatycznie załaduje interfejs użytkownika, w którym można przeprowadzić konfigurację parametrów algorytmu oraz rozpocząć obliczenia.



Rysunek 12: Plik .exe aplikacji do uruchomienia

4.3 Wprowadzenie danych wejściowych

Aplikacja jest podzielona na następujące segmenty:

- Warunki początkowe
- Parametry algorytmu
- Ustawienia kary
- Wyniki
- Ustawienia prawdopodobieństwa
- Opcje transformacji
- Dodatkowe opcje

W każdej z tych sekcji, z wyjątkiem „Wyniki”, użytkownik ma możliwość modyfikacji parametrów, które bezpośrednio wpływają na działanie algorytmu. Znajdują się tam wszystkie szczegółowo opisane i zdefiniowane ustawienia omówione w poprzednich rozdziałach.

Warunki początkowe można ustawić na dwa sposoby – losowo, poprzez zaznaczenie odpowiedniej opcji, lub ręcznie, wprowadzając wektor początkowy składający się z ośmiu liczb oddzielonych przecinkami.

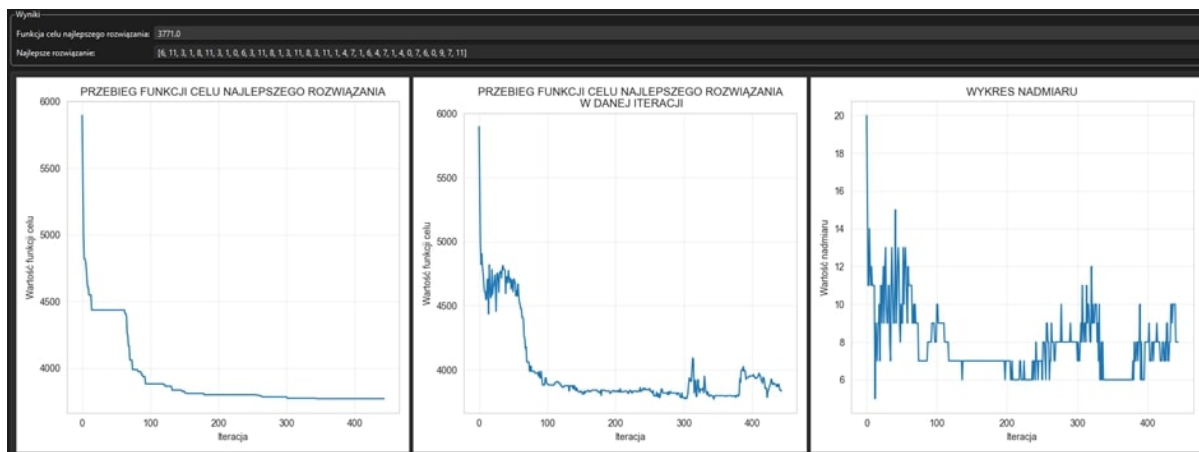
Zrzutek ekranu przedstawiający okno dialogowe o tytule "Warunki początkowe". W oknie znajdują się następujące elementy: pole tekstowe "Wektor początkowy (8 elementów):" z wartością "13, 20, 21, 17, 18, 13, 14, 24" (całe pole jest zaznaczone niebieskim prostokątem), przycisk "Losowy wektor początkowy?" z nieaktywnym polem wyboru, oraz dwa suwaki: "Minimalna wartość" ustawiony na 4 i "Maksymalna wartość" ustawiony na 12. Suwaki mają przyciski "↑" i "↓" do zmiany wartości.

Rysunek 13: Pole do wprowadzenia wektora początkowego

4.4 Prezentacja wyników i ich interpretacja

Przykładowy wynik działania programu:

- Funkcja celu najlepszego rozwiązania: 3771.0
- Najlepsze rozwiązanie (optymalna sekwencja świateł): [6, 11, 3, 1, 8, 11, 3, 1, 0, 6, 3, 11, 8, 1, 3, 11, 8, 3, 11, 1, 4, 7, 1, 6, 4, 7, 1, 4, 0, 7, 6, 0, 9, 7, 11]



Rysunek 14: Przykładowy wynik działania algorytmu

Na podstawie przebiegu funkcji celu najlepszego rozwiązania można stwierdzić, że ostatnia poprawa wartości funkcji celu nastąpiła w około 350 iteracji. Od tego momentu algorytm znajdował tylko gorsze rozwiązania, które nie były już zapamiętywane.

4.5 Rozwiązywanie potencjalnych problemów w użytkowaniu

W przypadku wprowadzenia danych znacząco zwiększających złożoność problemu (np. bardzo duże liczby w wektorze początkowym) algorytm może działać przez znaczący okres czasu, lub w ogóle nie skończyć obliczeń (aplikacja będzie wyglądać na zawieszoną).

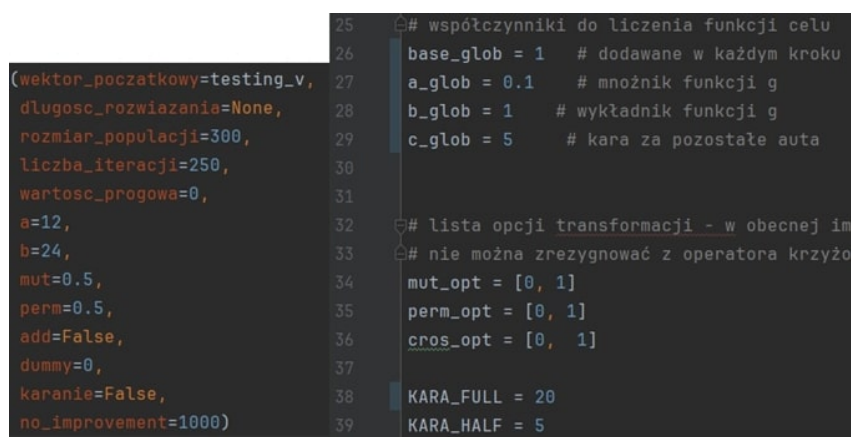
Eksperymenty i testowanie aplikacji

5.1 Opis metodyki testów i eksperymentów

Celem przeprowadzonych testów i eksperymentów było zbadanie efektywności algorytmu optymalizacyjnego, który ma na celu poprawę funkcji celu poprzez iteracyjne dostosowywanie parametrów algorytmu, takich jak wielkość populacji, zastosowanie operatorów mutacji, permutacji, a także wprowadzenie funkcji kary. Badania przeprowadzono w celu określenia, jak różne zmienne wpływają na wyniki uzyskane przez algorytm, a także na jego czasową złożoność obliczeniową.

5.2 Analiza i interpretacja różnych przypadków testowych - cz.1

Przypadek testowy [13, 20, 21, 17, 18, 13, 14, 24] – typowy wektor początkowy z losowymi wartościami z przedziału 12-24. Parametry bazowe algorytmu przedstawiono na poniższych zrzutach ekranu:

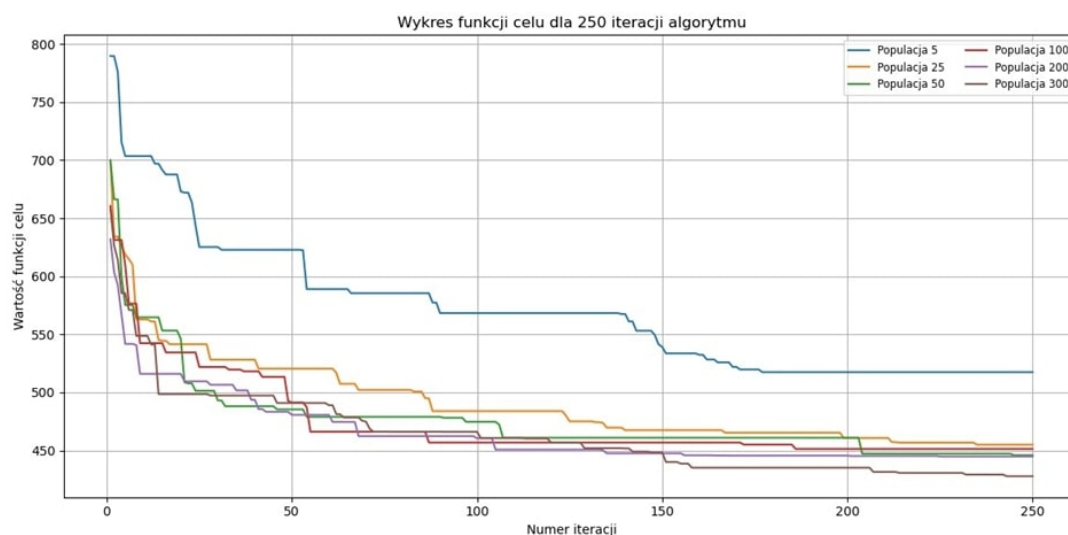


```
(wektor_początkowy=testing_v,  
  dlugosc_rozwiazania=None,  
  rozmiar_populacji=300,  
  liczba_iteracji=250,  
  wartosc_progowa=0,  
  a=12,  
  b=24,  
  mut=0.5,  
  perm=0.5,  
  add=False,  
  dummy=0,  
  karanie=False,  
  no_improvement=1000)  
  
25  # współczynniki do liczenia funkcji celu  
26  base_glob = 1  # dodawane w każdym kroku  
27  a_glob = 0.1  # mnożnik funkcji g  
28  b_glob = 1  # wykładnik funkcji g  
29  c_glob = 5  # kara za pozostałe auta  
30  
31  
32  # lista opcji transformacji - w obecnej im  
33  # nie można zrezygnować z operatora krzyżo  
34  mut_opt = [0, 1]  
35  perm_opt = [0, 1]  
36  cros_opt = [0, 1]  
37  
38  KARA_FULL = 20  
39  KARA_HALF = 5
```

Rysunek 15: Wartości bazowe parametrów algorytmu genetycznego

5.2.1 Badanie wpływu wielkości populacji na wartość funkcji celu dla rozwiązań losowych

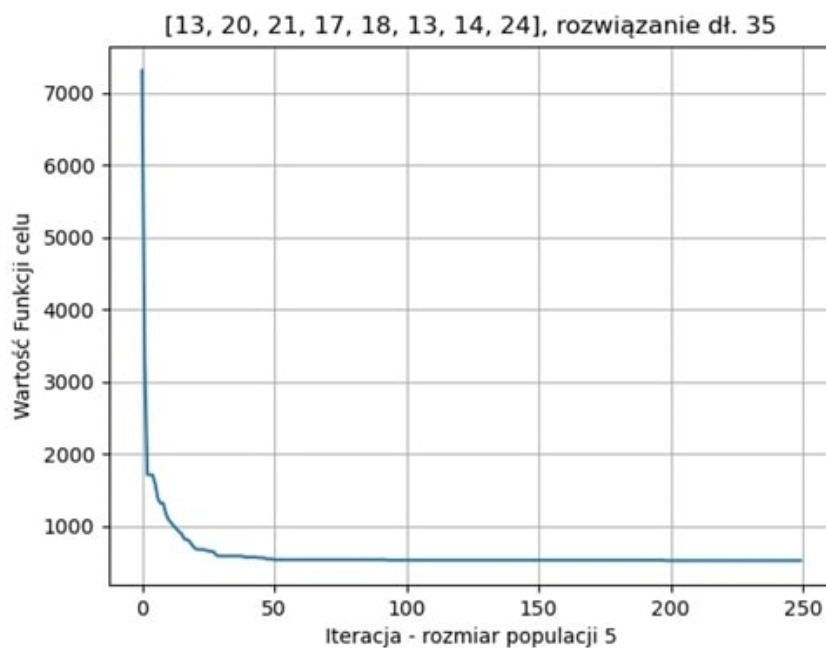
Badanie to pokazuje, że na ogół zwiększanie parametru odpowiedzialnego za liczbę osobników w populacji poprawia wartość funkcji celu najlepszego uzyskanego rozwiązania, natomiast z racji istnienia czynnika losowego podczas wykonywania algorytmu nie jest to regułą. Według przeprowadzonych obserwacji populacja o wielkości w zakresie 50 – 100 jest optymalna pod względem czasowej złożoności obliczeniowej oraz uzyskanego rozwiązania.



Rysunek 16: Wykres funkcji celu dla 250 iteracji algorytmu

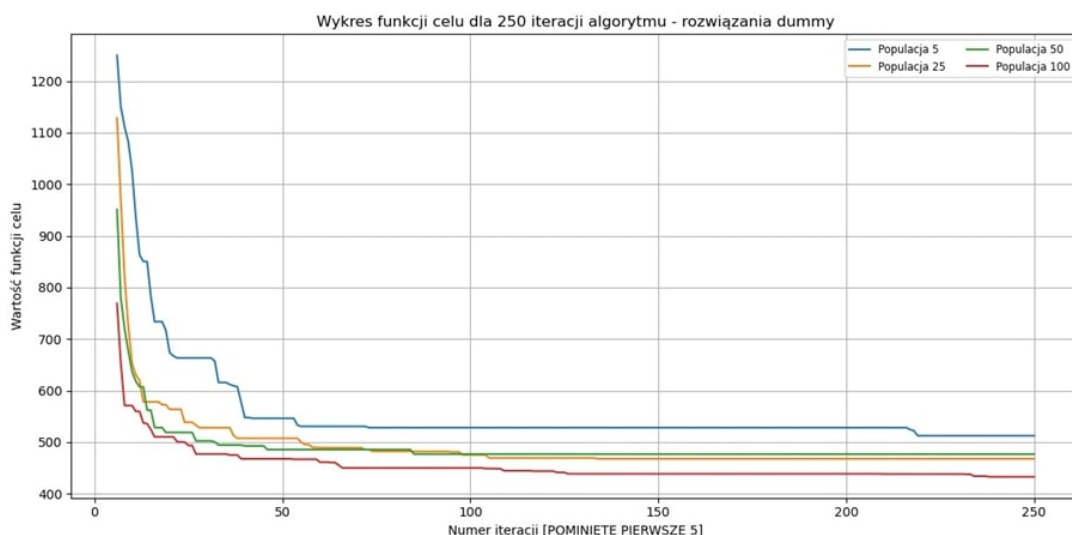
5.2.2 Badanie wpływu wielkości populacji na wartość funkcji celu dla rozwiązań domyślnych (zerowych)

Każdy wykres charakteryzował się gwałtownym spadkiem funkcji celu na przestrzeni pierwszych kilku iteracji – rozwiązania zerowe są dalekie od optymalnych, więc właściwie każda mutacja poprawiała wartość funkcji celu. Opisane zjawisko zostało przedstawione na poniższym wykresie:



Rysunek 17: Wykres funkcji celu - [13,20,21,17,18,13,14,24], rozwiązanie dł. 35

Z tego też powodu, dla zachowania lepszej skali na osi Y w wykresie porównawczym pominięto dane dla pierwszych pięciu iteracji:



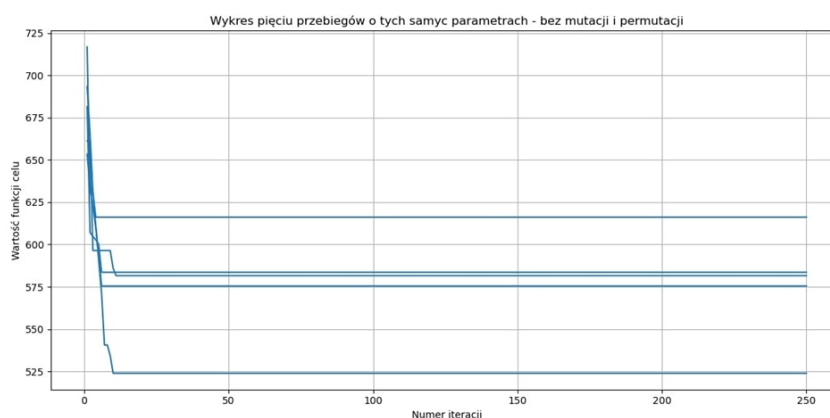
Rysunek 18: Wykres funkcji celu dla 250 iteracji algorytmu - rozwiązanie dummy

Wykres ma na celu zobrazowanie działania mutacji w algorytmie – mimo startu od domyślnych rozwiązań, dalekich optymalnych, algorytm zbiega do wartości funkcji celu podobnej dla symulacji rozpoczynającej od rozwiązań losowych – bliższych optymalnemu.

Warto zaobserwować również, że od samego początku wykresu, do ok. 50 iteracji, wartości funkcji celu różniły się znacząco między sobą na korzyść tych z większą populacją – było to spowodowane szybszym zróżnicowaniem populacji.

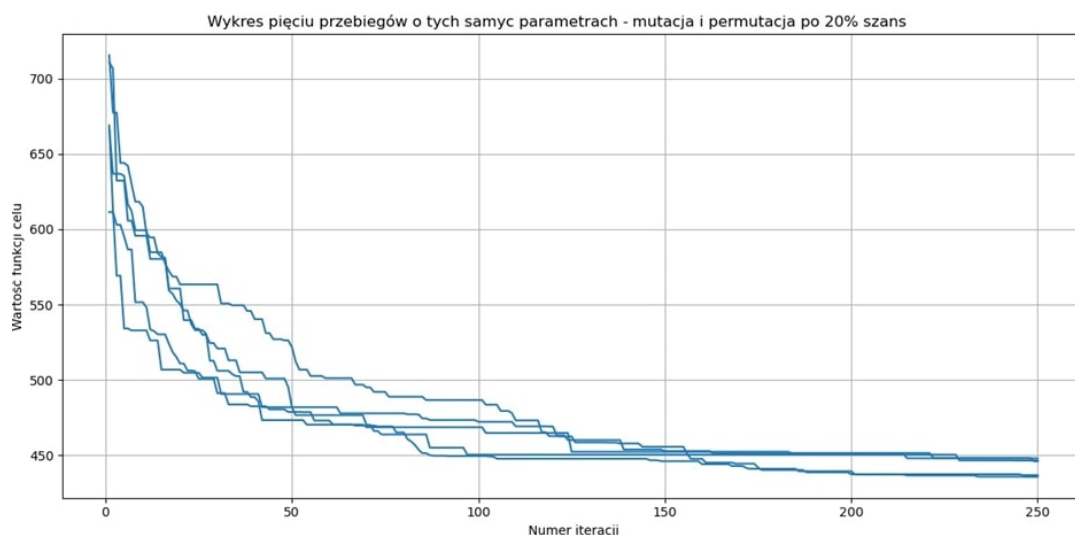
5.2.3 Badanie wpływu mutacji i permutacji na przebieg algorytmu - porównanie prawdopodobieństw 0% i 20%

Porównanie oparte zostało na pięciu realizacjach algorytmu o tych samych parametrach, rozmiar populacji 25, losowe rozwiązania początkowe. Na pierwszym wykresie prezentujemy przebieg wartości funkcji celu najlepszego rozwiązania podczas działania algorytmu bez operatorów mutacji i permutacji:



Rysunek 19: Wykres 5 przebiegów dla tych samych paramterów - bez mutacji i permutacji

Kolejny wykres analogicznie, ale szansa na permutację i mutację wynosi już 20%:

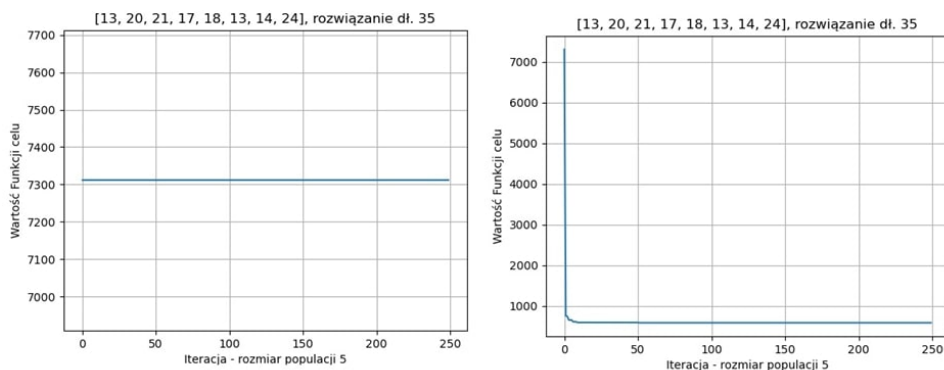


Rysunek 20: Wykres 5 przebiegów dla tych samych paramterów - mutacja i permutacja po 20% szans

Porównując te dwa wykresy można zaobserwować, że dzięki zastosowaniu operatorów mutacji oraz permutacji w każdej z pięciu realizowanych prób uzyskujemy lepszy wynik działania algorytmu. Dzięki nim możliwe jest również stopniowe polepszanie funkcji celu – zastosowanie samych operatorów krzyżowania po mniejszej ilości iteracji niż 20 tak bardzo ujednoliciła populację, że dalsze spadki funkcji celu są niemożliwe.

5.2.4 Uzasadnienie dodania parametru umożliwiającego dodawanie 5 losowych rozwiązań do krzyżowania

Algorytm, po zainicjalizowaniu rozwiązań populacji początkowej, dysponował jedynie jedną możliwością pozyskiwania nowych numerów kombinacji świateł spoza dostępnej puli – operatorem mutacji. W celu bardziej efektywnego różnicowania populacji podczas działania algorytmu, wprowadzono rozwiązanie umożliwiające dodawanie pięciu losowych rozwiązań w każdej iteracji. Na wykresach przedstawiono działanie algorytmu przy prawdopodobieństwie mutacji równym zero. Populacja została zainicjalizowana rozwiązaniami domyślnymi – zerowymi. Po prawej stronie widoczny jest parametr add, odpowiedzialny za dodawanie 5 losowych rozwiązań, ustawiony na True.

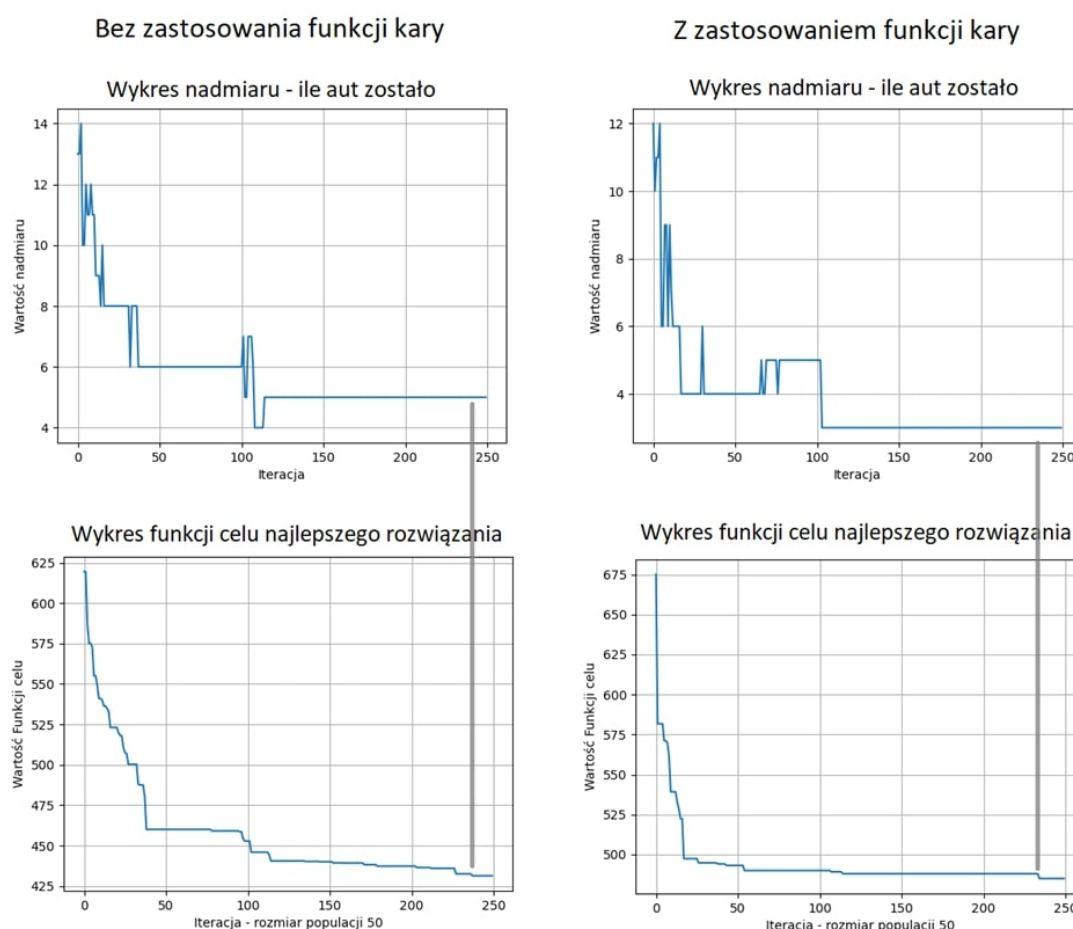


Rysunek 21: Wykres funkcji celu - [13,20,21,17,18,13,14,24], rozwiązanie dł. 35
Dodawanie losowych rozwiązań pozwoliło na działanie algorytmu z powodzeniem.

5.2.5 Zasada działania funkcji kary oraz zobrazowanie tego na wykresach

Podczas przeprowadzania testów zauważono, że algorytm niekoniecznie przeszukuje rozwiązania w sposób zamierzony – światła są często zmieniane, przez co przez skrzyżowanie przejeżdża mniej samochodów, niż by mogło. Funkcja przejścia uwzględnia sytuację, w której na danym sygnalizatorze światło zielone było zapalone dwa lub więcej razy pod rząd, co skutkuje zmniejszeniem liczby samochodów na danym pasie o 2, zamiast o 1. Wprowadzono zatem (opcjonalną) funkcję kary, która dla każdego rozwiązania sprawdza częstotliwość zmiany światel – porównuje kolejne dwie kombinacje i w zależności od tego, czy są one identyczne, pokrywają się jednym zielonym światłem, lub żadnym, nalicza odpowiednią wartość. Następnie rozwiązania są sortowane pod względem sumy funkcji celu i funkcji kary.

Poniżej przedstawiono wykresy ilustrujące przebieg algorytmu z zastosowaniem oraz bez zastosowania funkcji kary.



Rysunek 22: Wykresy przedstawiające przebieg algorytmu AG z funkcją kary oraz bez jej zastosowania

Szarymi liniami zaznaczono moment (iterację), w której znalezione zostało rozwiązanie quasi-optymalne. W pierwszym przypadku wartość funkcji celu jest znacznie lepsza i wynosi około 430, jednak górny wykres pokazuje, że w tej iteracji aż pięć samochodów nie opuściło skrzyżowania. Zastosowanie funkcji kary pozwoliło uzyskać lepsze rozwiązanie, w którym więcej samochodów opuściło skrzyżowanie. Wartość funkcji celu w tym przypadku jest jednak mniej korzystna – wynosi około 485. Taka różnica wynika z losowo-

ści algorytmu, nieidealnego dobrania współczynników funkcji kary oraz prawdopodobnie nienajlepszej implementacji funkcji kary. Podsumowując, funkcja kary spełnia swoje założenia, lecz w niektórych przypadkach pogarsza funkcję celu.

Dla pełnej przejrzystości, zamieszczono parametry globalne funkcji kary (kara pełna za brak zgodności świateł, kara połowiczna za zgodność tylko jednego sygnalizatora, brak kary za pełną zgodność):

```
KARA_FULL = 8
KARA_HALF = 3
```

Rysunek 23: Parametry globalne funkcji kary

5.2.6 Próba znalezienia parametrów algorytmu, które pozwolą na powtarzalne uzyskiwanie najlepszego rozwiązania przy sensownym nakładzie czasowym

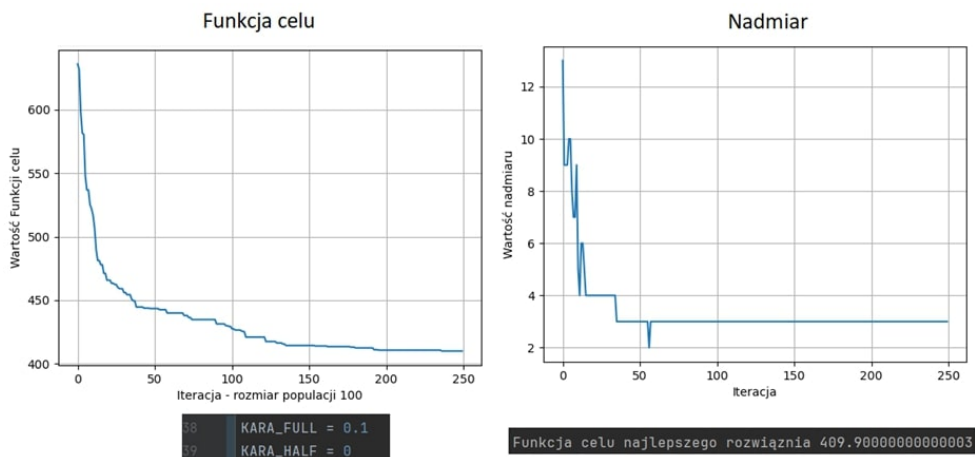
Podczas dostrajania parametrów algorytmu zdecydowano się pozostać przy rozmiarze populacji wynoszącym 100 i liczbie iteracji 250, co dawało dosyć długi, ale akceptowalny czas wykonywania algorytmu. Stosując algorytm sparametryzowany w sposób opisany poniżej, najlepsze rozwiązanie charakteryzowało się funkcją celu o wartości 412.3. Rzadko zdarzały się wartości przekraczające 430, co uznano za wystarczającą powtarzalność. Z obserwacji wynika, że aby zapewnić zadowalające i powtarzalne rozwiązanie, wartości parametrów mutacji i permutacji nie mogą być ani za duże, ani za małe, a ich zmiana ma istotny wpływ na działanie algorytmu. Warto zauważyć, że wyniki te uzyskano przy najprostszych operatorach mutacji i krzyżowania, oraz bardziej wyrafinowanym operatorem permutacji, co sugeruje, że zwiększanie możliwości algorytmu i wpływu losowości na rozwiązywanie nie zawsze jest dobrym kierunkiem rozwoju algorytmu.

```
przeprowadzenie_symulacji(wektor_poczkowy=testing_v,
    dlugosc_rozwiazania=None,
    rozmiar_populacji=100,
    liczba_iteracji=250,
    wartosc_progowa=0,
    a=12,
    b=24,
    mut=0.25,
    perm=0.3,
    add=False,
    dummy=0,
    karanie=False,
    no_improvement=1000)

25 # współczynniki do liczenia funkcji celu
26 base_glob = 1 # dodawane w każdym kroku
27 a_glob = 0.1 # mnożnik funkcji g
28 b_glob = 1 # wykładnik funkcji g
29 c_glob = 5 # kara za pozostałe auta
30
31
32 # lista opcji transformacji - w obecnej in
33 # nie można zrezygnować z operatora krzyżo
34 mut_opt = [0]
35 perm_opt = [1]
36 cros_opt = [0]
```

Rysunek 24: Parametry przeprowadzania symulacji oraz współczynniki do obliczania funkcji celu

Jedno z lepszych rozwiązań (411.8) udało się również uzyskać wzbogacając symulację o funkcję kary – jednak z o wiele mniejszymi współczynnikami niż w podrozdziale poświęconym jej w całości. Na poniższym zestawieniu wykresów widać jak nawet tak mała wartość kary wpłynęła na faworyzację rozwiązań, w których mniej pojazdów zostaje na skrzyżowaniu – osiągnięte zostało nawet jedno rozwiązanie z dwoma takimi pojazdami, lecz niestety nie ewoluowało dalej.



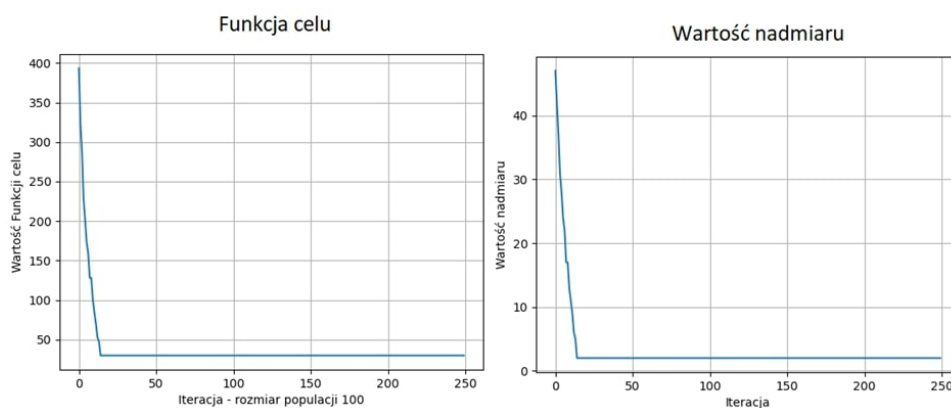
Rysunek 25: Wykresy funkcji celu i nadmiaru z zastosowaną karą oraz wynik funkcji celu.

5.3 Analiza i interpretacja różnych przypadków testowych - cz.2

Przypadek testowy $[40, 40, 0, 0, 0, 0, 0, 0]$ stanowi trywialny przykład, w którym oczekiwane jest rozwiązanie ze stałą kombinacją zielonych świateł na dwóch sygnalizatorach. Parametry algorytmu pozostają takie same jak najlepsze wartości określone w rozdziale 5.2.6.

5.3.1 Test działania podstawowego algorytmu na przypadku trywialnym

Uruchomienie programu w celu przetestowania algorytmu (przypadek trywialnym) i zobrazowanie rezultatów zwróconych przez program:



Rysunek 26: Wykresy funkcji celu i funkcji obrazującej zachowanie się nadmiaru

```

Funkcja celu najlepszego rozwiązania 30.0
Najlepsze rozwiązanie: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

```

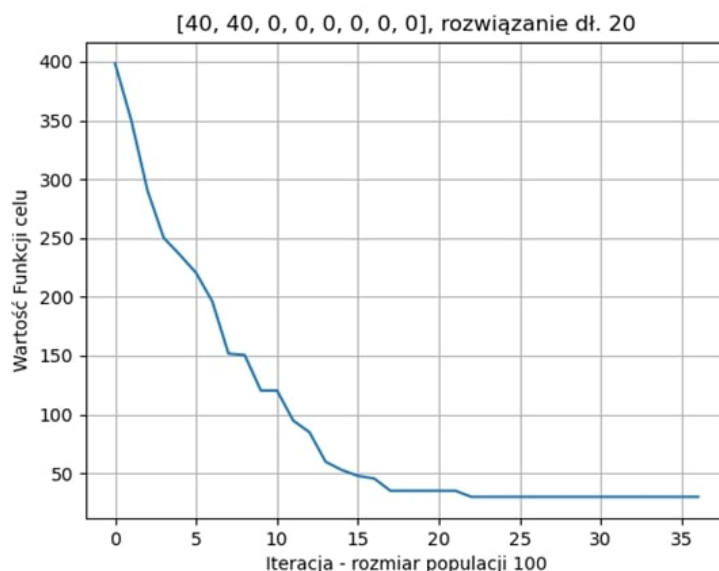
Rysunek 27: Wynik funkcji celu i najlepsze znalezione rozwiązanie

Algorytm stosunkowo szybko znajduje rozwiązanie optymalne. Wartość nadmiaru równa 2 może budzić wątpliwości, co wynika z faktu, że bonus za wyświetlanie zielonego światła „pod rząd” nie jest naliczany w pierwszej iteracji – w rezultacie na końcu 20-elementowego rozwiązania pozostaje po jednym samochodzie na pas.

Przypadek ten ilustruje, że opracowany skrypt realizujący algorytm genetyczny nie opiera się wyłącznie na błędzeniu losowym. Pomimo pozornego wrażenia, że problem mógłby zostać rozwiązany przez losowe dobieranie kolejnych rozwiązań, algorytm bazuje na klasycznej wersji książkowej. Osiągnięcie optymalnego rozwiązania (wektor dwudziestu zer) w tym przypadku nie byłoby możliwe przy użyciu jedynie losowości.

5.3.2 Weryfikacja warunku stopu na podstawie braku poprawy

Na podstawie tego trywialnego przypadku przeprowadzane są testy warunków stopu, zaczynając od warunku zatrzymania w przypadku braku poprawy przez określoną liczbę iteracji. W tym przypadku parametr `no_improvement` został ustawiony na 15.

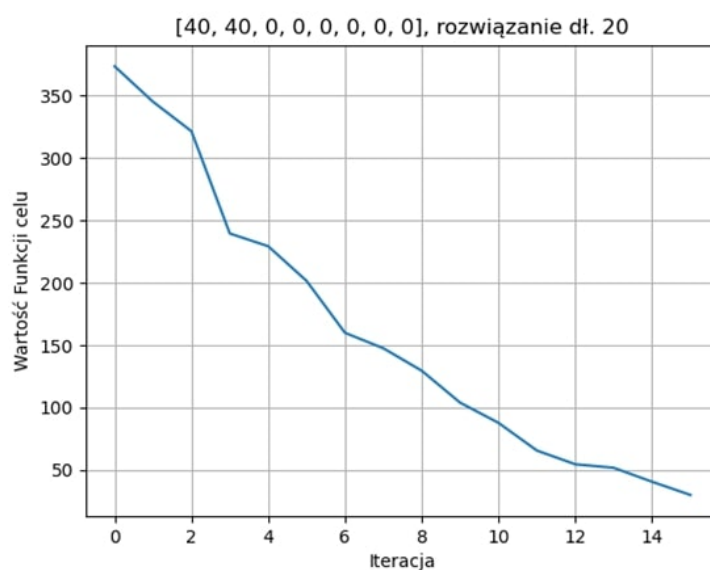


Rysunek 28: Wykres funkcji celu - $[40, 40, 0, 0, 0, 0, 0, 0]$, rozwiązanie dł. 20

Działanie warunku jest prawidłowe – algorytm zatrzymuje się po dokładnie 15 iteracjach, gdy wartość funkcji celu wynosi 30.

5.3.3 Sprawdzenie warunku stopu osiągnięcia wartości progowej

Znając najlepszą wartość funkcji celu osiąganą przez algorytm (30), przeprowadzane są testy warunku stopu polegającego na osiągnięciu tej wartości. W tym celu parametr `wartosc_progowa` został ustawiony na 30.1.

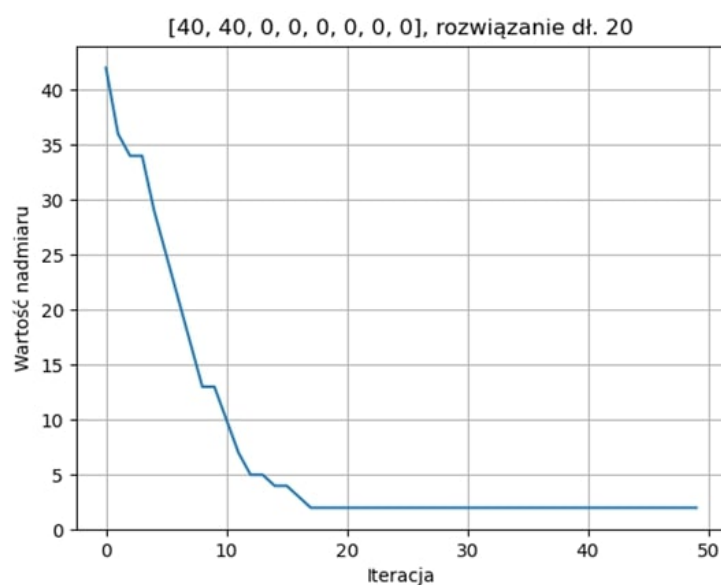


Rysunek 29: Wykres funkcji celu - $[40, 40, 0, 0, 0, 0, 0, 0]$, rozwiązanie dł. 20

Algorytm zatrzymuje się natychmiast po uzyskaniu rozwiązania, w którym wartość funkcji celu jest mniejsza niż ustalona wartość.

5.3.4 Test warunku stopu na podstawie maksymalnej liczby iteracji

Algorytm zawiera również warunek stopu oparty na maksymalnej liczbie iteracji, który przez większość czasu był ustawiony na 250 i działał prawidłowo. Dla formalności zaprezentowane zostaje ograniczenie do 50 iteracji. W tym celu parametr liczba iteracji został ustawiony na 50 (ze względu na numerację od zera, wykres kończy się na iteracji numer 49).



Rysunek 30: Wykres funkcji nadmiaru - $[40, 40, 0, 0, 0, 0, 0, 0]$, rozwiązanie dł. 20

Podsumowanie

6.1 Ocena osiągnięcia założonych celów

Gdy po raz pierwszy analizowano problem optymalizacji sterowania światłami w ruchu drogowym, zauważono wzajemną relację skrzyżowań w ciągu, np. obwodnicy miasta, oraz fakt skończonej długości pasa pomiędzy nimi. Ruch pieszy i tramwajowy, znacznie komplikujący ruch samochodowy, a jednocześnie występujący w rzeczywistości, także był brany pod uwagę, jak i zmiany natężenia ruchu w czasie rzeczywistym, wymagające adaptacji systemu do tych zmian. Podczas tworzenia modelu szybko zrozumiano, że problem, który początkowo wydawał się prosty, wymagał opracowania matematycznego modelu o dużym stopniu komplikacji. Z tego powodu zdecydowano się na rezygnację z rozważania kolejnych zmiennych i uproszczenie realizacji modelu do wersji przedstawionej w sprawozdaniu. Mimo tego uznano, że zaprezentowana symulacja ma dużą wartość merytoryczną w kontekście rozważania omawianego problemu, ukazując skalę trudności występujących w problemach optymalizacji. Ponadto, realizacja projektu umożliwiła uzyskanie praktycznego spojrzenia na rozwiązanie problemu oraz dokładne zapoznanie się z działaniem algorytmu genetycznego. Podsumowując, cel, jakim była próba zamodelowania i rozwiązania rzeczywistego problemu za pomocą algorytmu genetycznego, został w pełni osiągnięty.

6.2 Problemy napotkane podczas realizacji

Podczas realizacji projektu nie napotkano żadnych istotnych problemów, ani przy formułowaniu problemu, ani przy późniejszej implementacji jego rozwiązania. Na etapie definiowania celów udało się precyzyjnie określić wszystkie kluczowe zmienne oraz wymagania dotyczące systemu, co pozwoliło na płynne przejście do fazy implementacji. Algorytm genetyczny został zaimplementowany zgodnie z założeniami, a testowanie poszczególnych modułów odbywało się bez większych trudności. Dzięki odpowiedniemu doborowi narzędzi oraz metod optymalizacji, cały system zadziałał zgodnie z przewidywaniami.

6.3 Kierunki dalszego rozwoju projektu

Zwiększenie liczby zmiennych: W przyszłości projekt może uwzględniać dodatkowe zmienne, takie jak:

- Liczba pasów na skrzyżowaniu,
- Zmienność liczby pojazdów w czasie (np. zmiany natężenia ruchu),

- Wpływ warunków pogodowych na ruch (np. deszcz, śnieg),
- Losowe opóźnienia wynikające z przechodzenia pieszych przez przejścia tuż przed końcem cyklu sygnalizacyjnego, co może wydłużać czas oczekiwania dla pojazdów.

Zaawansowane techniki optymalizacji: Zastosowanie innych algorytmów optymalizacyjnych, takich jak algorytmy uczenia maszynowego (np. *Reinforcement Learning*), które mogą dynamicznie dostosowywać sygnalizację w odpowiedzi na zmiany w natężeniu ruchu.

Modelowanie w czasie rzeczywistym: Implementacja modelu, który będzie działał w czasie rzeczywistym i sterował sygnalizacją na żywo w zależności od zmieniającego się ruchu.

Zastosowanie do rzeczywistych skrzyżowań: Rozwój systemu do zastosowań w rzeczywistych miastach, gdzie algorytmy będą sterować fizycznymi sygnalizatorami w oparciu o dane w czasie rzeczywistym (np. z kamer monitorujących ruch, czujników wbudowanych w jezdnię).

Interakcja z systemami transportu publicznego: Zintegrowanie algorytmów z systemami zarządzania transportem publicznym, aby poprawić płynność ruchu nie tylko dla samochodów prywatnych, ale również dla autobusów i tramwajów.

Prezentacja pracy w grupie na różnych etapach tworzenia projektu

7.1 Omówienie realizowanych zadań wśród członków zespołu

Prace nad projektem prowadzono systematycznie od października do grudnia, a kluczową rolę w jego realizacji odegrały cotygodniowe, dwugodzinne spotkania zespołu na platformie *Microsoft Teams*. W ich trakcie omawiano nie tylko postępy każdego członka zespołu, czy bieżące problemy, ale również przydzielano każdemu nowe zadania, co pozwalało na bardziej efektywne zarządzanie całym projektem. Postępy dokumentowano na bieżąco zarówno w formie prezentacji, dokumentów tekstowych, jak i kodu udostępnianego w serwisie *GitHub*.

Projekt wymagał od każdego ścisłej współpracy na wszystkich etapach jego realizacji, począwszy od analizy zagadnienia, poprzez opracowanie algorytmów, aż po implementację i testowanie aplikacji. Każdy z członków zespołu wniósł także do niego cenne umiejętności oraz pomysły, co miało istotny wpływ na końcowy rezultat. Poniższa tabela przedstawia podział zadań i procentowy udział poszczególnych osób w różnych etapach realizacji projektu:

Etapy projektu	Wojciech Jankowski	Jakub Głód	Mateusz Kuc
Model zagadnienia	30%	35%	35%
Opracowanie Algorytmu	30%	40%	30%
Implementacja Aplikacji	35%	35%	30%
Interfejs GUI	20%	30%	50%
Testowanie Aplikacji	35%	35%	30%
Dokumentacja	50%	25%	25%

Tabela 1: Tabela prezentująca procentowy udział członków zespołu w realizowanym projekcie

Literatura

- [1] Joanna Kwiecień, Wojciech Chmiel, *Wykłady z badań operacyjnych 2*, 2024/2025.
- [2] David E. Goldberg, *Algorytmy genetyczne i ich zastosowania*, Wydawnictwo WNT, 1995.
- [3] Bogusław Filipowicz, *Badania operacyjne. Wybrane metody obliczeniowe i algorytmy. Cz. 2.*, Wydawnictwo ABART, 2007.
- [4] Niklaus Wirth, *Algorytmy + struktury danych*, Wydawnictwo WNT, 2004.
- [5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, *Wprowadzenie do algorytmów*, Wydawnictwo PWN, 2017.