Jakub Kosmydel
Norbert Morawski
Bartłomiej Wiśniewski
Przemysław Węglik

# Badania operacyjne
# Projekt

15 maja 2023

# Spis treści


**1. Wstęp** . . . 2

**2. Opis zagadnienia** . . . 3

2.1. Sformułowanie problemu . . . 3

2.2. Model matematyczny . . . 3

2.2.1. Założenia . . . 3

2.2.2. Dane . . . 3

2.2.3. Szukane . . . 4

2.2.4. Hiperparametry . . . 4

2.2.5. Funkcja kosztu . . . 4

**3. Opis algorytmów** . . . 5

3.1. Reprezentacja środowiska . . . 5

3.1.1. Reprezentacja mapy . . . 5

3.1.2. Reprezentacja genotypu . . . 5

3.1.3. Reprezentacja linii . . . 5

3.2. Rozwiązanie początkowe . . . 5

3.3. Symulacja . . . 6

3.4. Selekcja . . . 6

3.5. Mutacja . . . 7

3.5.1. LineMutator . . . 7

3.5.2. GenotypeMutator . . . 7

3.6. Krzyżowanie . . . 8

3.6.1. GenotypeCrosser . . . 8

**4. Aplikacja** . . . 9

**5. Eksperymenty** . . . 10

5.1. Eksperymenty proste . . . 10

5.2. Przeszukiwanie siatki hiper-parametrów . . . 11

**6. Podsumowanie** . . . 17

# 1. Wstęp

Celem naszego projektu jest znalezienie optymalnych tras linii dla autobusów, aby maksymalizować liczbę pasażerów, przy minimalnej liczbie linii autobusowych. Aby to osiągnąć, wykorzystywane są algorytmy genetyczne - algorytmy przeszukujące przestrzeń rozwiązań, które opierają się na procesie działania mechanizmu dziedziczenia biologicznego.

W systemie założono, że pozycje oraz popularność przystanków są z góry ustalone. Stosowanie algorytmów genetycznych pozwoliło na wygenerowanie zestawu najlepszych połączeń autobusowych, które można skonfigurować dla lepszego wykorzystania zasobów oraz zwiększenie korzyści z transportu publicznego dla pasażerów.

# 2. Opis zagadnienia

## 2.1. Sformułowanie problemu

Naszym celem w projekcie jest zaprojektowanie sieci linii autobusowych pokrywającej dany obszar miejski, który już posiada sieć przystanków autobusowych. Linie te, powinny mieć możliwość obsłużenia jak największej liczby pasażerów, tworząc jak najmniej postojów oraz zatrzymując się na jak najmniejszej liczbie przystanków.

## 2.2. Model matematyczny

### 2.2.1. Założenia

1. Przystankom przypisujemy ilość punktów w zależności od gęstości zaludnienia w pobliżu oraz ciekawych punktów (teatr, park itp.).
   — Dla każdego przystanku obliczamy liczbę ludzi w pobliżu,
   — Głównym punktom w Krakowie (np. D17, teatry, itp.) nadajemy wartość punktową,
   — Dla każdego przystanku sumujemy powyższe wartości.
2. Rozkładamy linie komunikacyjne po mieście tak, by maksymalizować sumę zebranych punktów przez wszystkie linie.
3. Wprowadzamy koszt dla linii: koszt ścieżki w grafie, po której jedzie + koszt utworzenia nowej linii.
4. Linie przebiegające przez jeden przystanek dzielą się punktami,
5. Maksymalizujemy sumę punktów zebranych przez wszystkie linie.

### 2.2.2. Dane

1. $n$ - liczba linii
2. $m$ - liczba przystanków

**Graf**

1. Wierzchołki to istniejące przystanki z przypisanymi punktami,
2. $p(j)$ - wartość punktowa przystanku:
   — W początkowej wersji liczba ta jest określona z góry,
   — $p(j) = \sum_{i=0}^{n-1} \frac{w_{j,i}}{f(d_j,i)}$ gdzie $w_{j,i}$ to wartość obiektu (np. liczba mieszkańców w pobliżu) a $d_{j,i}$ to odległość tego bloku od przystanku, f – funkcja skalująca.
   — Funkcja liczona dla danego przystanku $j$
3. Krawędzie to połączenia między przystankami.
4. Koszt krawędzi to odległości między przystankami.

### 2.2.3. Szukane

$x_{i,j}$ - czy linia $i$ zatrzymuje się na przystanku $j$, gdzie:

1. $i \in [0, n-1]$
2. $j \in [0, m-1]$

### 2.2.4. Hiperparametry

1. $\alpha$ - koszt zatrzymania się na przystanku,
2. $\beta$ - koszt nowej linii,
3. $R$ - hiper parametr zbiegania.

### 2.2.5. Funkcja kosztu

$$l_j = \sum_{i=0}^{n-1} x_{i,j}$$ liczba linii zatrzymujących się na przystanku $j$

$$q_j = \frac{p_j \cdot (1 + \frac{R}{l_j})^{l_j}}{l_j} \to \frac{e^R}{l_j}$$ ile punktów każda linia uzyskuje z przystanku $j$

$S_i$ długość ścieżki linii $i$ w grafie

$$cost_j = \begin{cases} \sum_{i=0}^{n-1} x_{i,j} \cdot (q_j - \alpha) & l_j > 0 \\ -\Delta & l_j = 0 \end{cases}$$ penalizacja nieodwiedzonych przystanków

$$f(x) = \sum_{j=0}^{m-1} cost_j - \sum_{i=0}^{n-1} [S_i - \beta]$$ funkcja kosztu

# 3. Opis algorytmów

Nasz problem rozwiązywaliśmy algorytmami genetycznymi.

## 3.1. Reprezentacja środowiska

Jak już zostało wspomniane, zajmowaliśmy się problemem optymalizacji istniejącej sieci komunikacyjnej, bez tworzenia nowych połączeń.

### 3.1.1. Reprezentacja mapy

Mapa z przystankami jest reprezentowana jako ważony graf z biblioteki NetworkX.

### 3.1.2. Reprezentacja genotypu

Genotyp składa się z listy linii autobusowych:

```
class Genotype:
    def __init__(self, lines: list[Line]):
        self.lines = lines
```

### 3.1.3. Reprezentacja linii

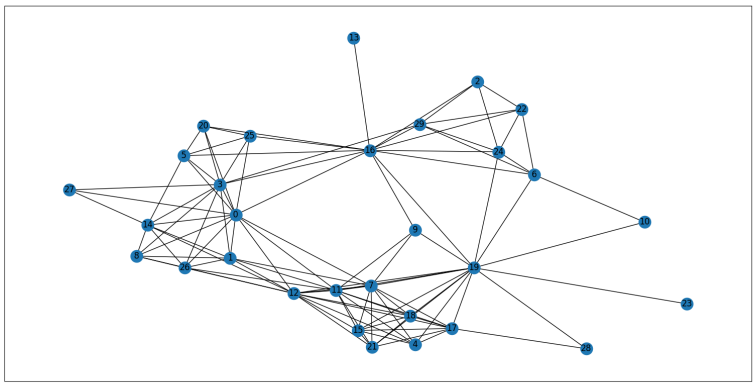
Linia posiada następujące parametry:
1. id - id linii,
2. stops - przystanki, na których się zatrzymuje,
3. edges - wszystkie krawędzie, przez które linia przejeżdża,
4. edge_color - kolor linii; do reprezentacji graficznej,
5. edge_style - styl krawędzi linii; do reprezentacji graficznej,

```
class Line:
    def __init__(self, stops: list[int], best_paths):
        self.id = Line.get_next_id()
        self.stops = stops  # ordered list of stops
        self.edges = []
        self.edge_color = [...]
        self.edge_style = [...]
```

## 3.2. Rozwiązanie początkowe

Na początku, chcąc się skupić na realizacji algorytmu, wygenerowaliśmy losowo sieć połączeń. Powstała ona przez wygenerowanie N punktów na płaszczyźnie, a następnie

połączeniu ich między sobą z pewnym prawdopodobieństwem. Dawało to całkiem dobre rezultaty:



Rysunek 3.1. Przykładowa wygenerowana mapa

## 3.3. Symulacja

**Algorytm 1** Symulacja

```
1: function SYMULUJ(liczba_pokoleń, x)
2:     populacja = POPULACJA_POCZĄTKOWA( )
3:     ZAPISZ_POPULACJĘ( )
4:     for i ← 0 to liczba_pokoleń − 1 do
5:         populacja = USUŃ_PUSTE(populacja)
6:         populacja_dopasowanie = FITNESS(populacja)
7:         populacja = FUNKCJA_PRZETRWANIA(populacja, populacja_dopasowanie)
8:         populacja_nowa = NOWA_POPULACJA(populacja)          ▷ Tutaj zachodzą
   mutacje i krzyżowania
       Co x epok:
9:         ZAPISZ_POPULACJĘ( )
10:    end for
11: end function
```

Powyżej przedstawiony został podstawowy silnik symulacji. W każdej epoce wykonuje on następujące kluczowe czynności:
— Usuwa niedopuszczalne rozwiązania (linie bez przystanków, organizmy bez linii),
— Oblicza funkcję dopasowania,
— Uruchamia funkcję przetrwania, która likwiduje wybrane osobniki,
— Uruchamia funkcję nowej populacji, która dokonuje mutacji i krzyżowań.
Na tym poziomie nie definiujemy co dana funkcja robi. Zostało to zrobione poniżej.

## 3.4. Selekcja

Wypróbowaliśmy wielu różnych metod selekcji nowych osobników:

1. n_best_survive(n) - pozostawia daną liczbę $n$ najlepszych osobników,
2. n_best_and_m_random_survive(n, m) - pozostawia $n$ najlepszych osobników, oraz $m$ losowych spośród pozostałych,
3. n_best_and_m_worst_survive(n, m) - pozostawia $n$ najlepszych i $m$ najgorszych osobników,
4. exponentional_survival(n, lambda) - pozostawia $n$ osobników w sposób losowy, ale zależny od uzystanej wartości $fitness$ i zgodny z rozkładem wykładniczym parametryzowanym przez $lambda$,
5. exponentional_survival_with_protection(best_protected, worst_protected, lambda) - działa jak $exponentional\_survival(lambda, n)$, ale gwarantuje przeżycie $best\_protected$ najlepszym i $worst\_protected$ najgorszym osobnikom,

## 3.5. Mutacja

### 3.5.1. LineMutator

Tworzy nowe mutacje dla danej linii.
Możliwe mutacje:

1. rotation_to_right - losuje spójny ciąg przystanków w lini i przesuwa je o zadaną (lub losową) liczbę pozycji
2. cycle_rotation - losuje pozycje przystanków w lini i przesuwa obecne na tych pozycjach przystanki o jedną pozycję w ramach wylosowanych pozycji
3. invert - odwraca kolejność przystanków, pomiędzy losowymi indeksami start oraz end,
4. erase_stops - losowo usuwa zadaną liczbę przystanków z linii,
5. add_stops - losowo dodaje zadaną liczbę przystanków, spośród tych, które w linii nie występują
6. replace_stops - losowo zmienia zadaną liczbę przystanków z lini na inne. Nowe przystanki są wybierane z rozkładu jednostajnego lub wykładniczego gdzie przystanki bliższe do obecnego są bardziej prawdopodobne

### 3.5.2. GenotypeMutator

Możliwe mutacje:

1. erase_line(G) - tworzy nowy genotyp, usuwając losową linię,
2. create_line(G) - tworzy nowy genotyp, dodając losowo wygenerowaną linię,
3. split_line (G) - tworzy nowy genotyp, rozdzielając losową, losową linię dwie różne.
4. merge_lines(G) - tworzy nowy genotyp, łącząc zadaną liczbę losowych linii. W zależności od wartości parametru line mogą być łączone całościowo lub na poziomie pojedyńczych przystanków
5. cycle_stops_shift(G) - tworzy nowy genotyp, ustawiając ciągi przystanków z lini obok siebie i wykonując cycle_rotation na takim ciągu przystanków

## 3.6. Krzyżowanie

### 3.6.1. GenotypeCrosser

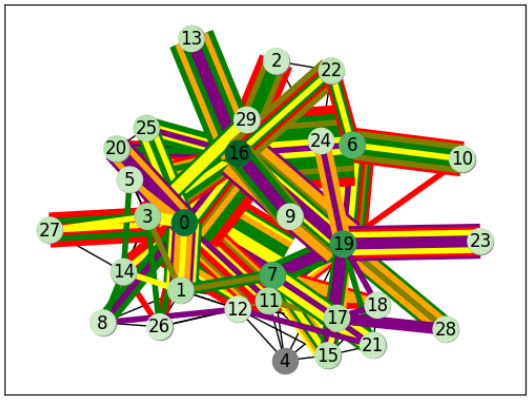
1. merge_genotypes(G1, G2) - tworzy nowy genotyp poprzez wybranie losowych lini z genotypów G1 i G2
2. cycle_stops_shift(G1, G2) - najpierw wykonuje merge_genotypes(G1, G2), a następnie GenotypeMutator.cycle_stops_shift(G)
3. line_based_merge(G1, G2) - dzieli każdą z lini z G1 i G2 na połowy i jedną z połów każdej lini łączy z połową lini z drugiego genotypu. Z 4 możliwych przypadków połączenia wybiera ten w którym dystans pomiędzy połączonymi przystankami jest minimalny

# 4. Aplikacja

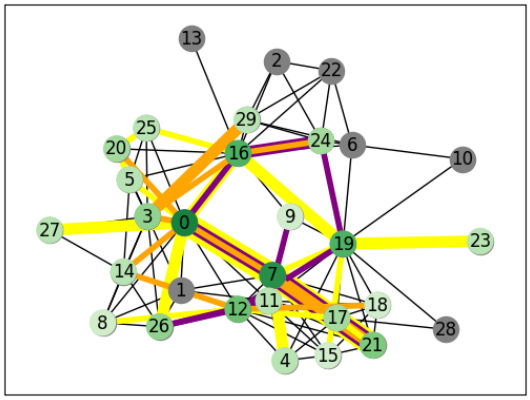
# 5. Eksperymenty

Na naszym algorytmie przeprowadziliśmy szereg eksperymentów.

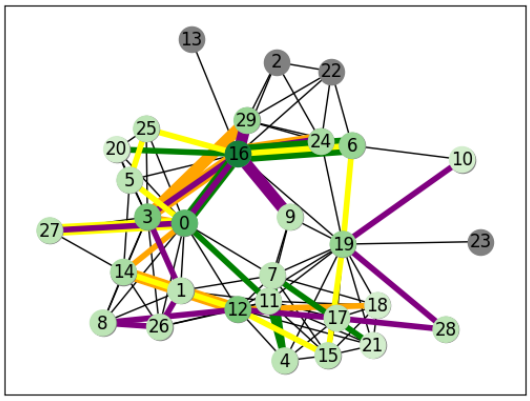
## 5.1. Eksperymenty proste



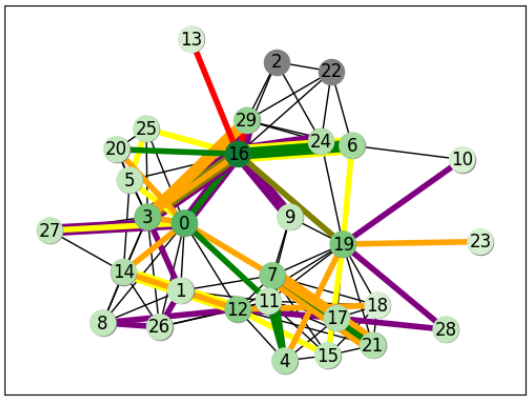

Rysunek 5.1. Populacja 0
dopasowanie −121.46




Rysunek 5.2. Populacja 3
dopasowanie 1.71

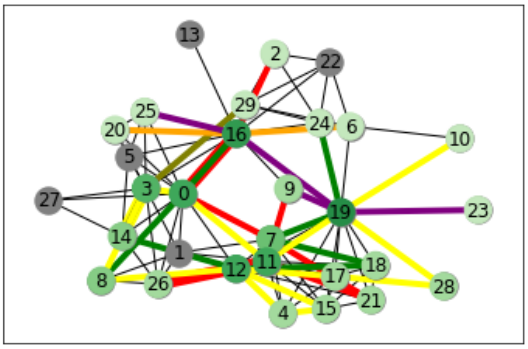



Rysunek 5.3. Populacja 5
dopasowanie 11.08

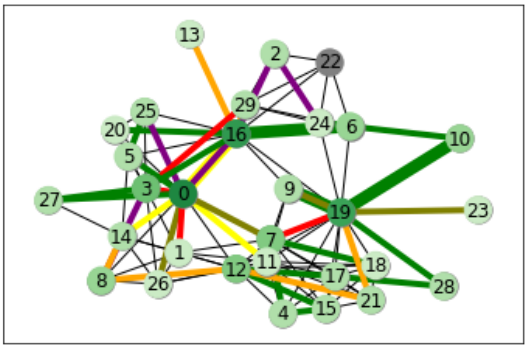



Rysunek 5.4. Populacja 10
dopasowanie 14.40

Jak widzimy, już po 10 epokach sieć połączeń znacznie się wyklarowała. Funkcja dopasowania wzrosła znacząco od generacji 0 do 10.

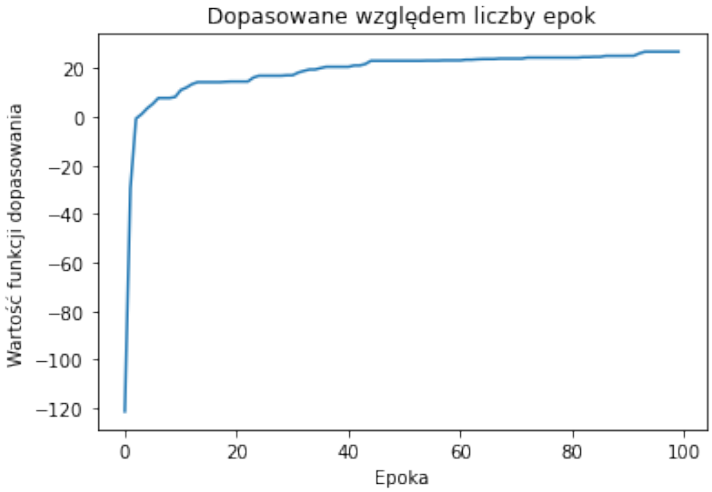



Rysunek 5.5. Populacja 20
dopasowanie 17.18




Rysunek 5.6. Populacja 100
dopasowanie 24.64

Sieć pokryła jeszcze więcej przystanków. Tempo wzrostu funkcji dopasowania zmalało.

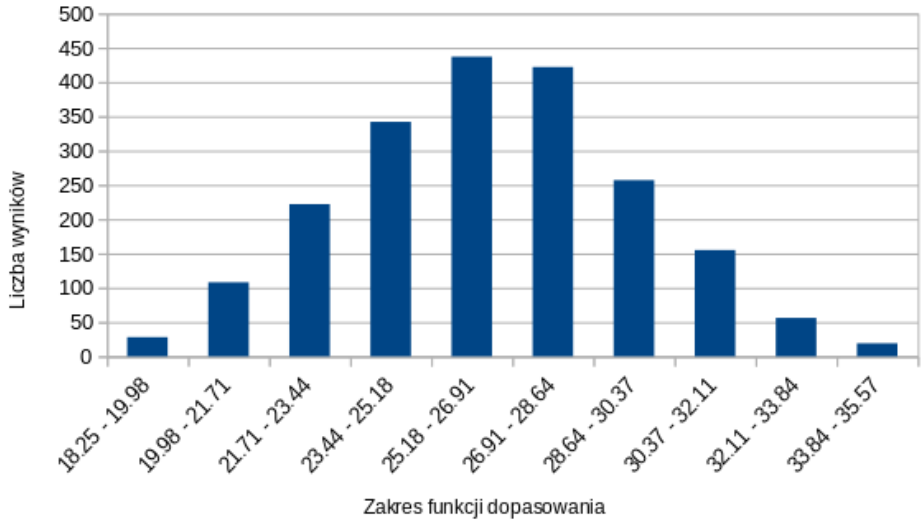



Rysunek 5.7. Wykres funkcji dopasowania

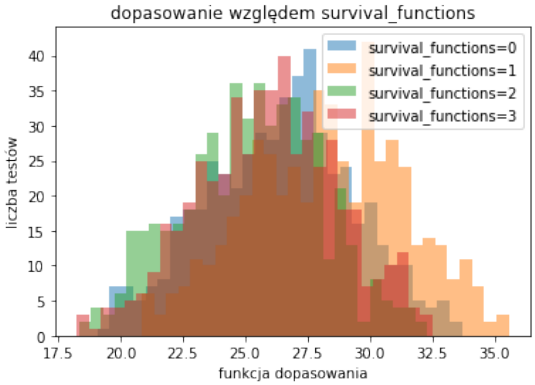
Jak widać, rzeczywiście tempo dopasowywania się modelu znacznie spada w późniejszych etapach symulacji.

## 5.2. Przeszukiwanie siatki hiper-parametrów

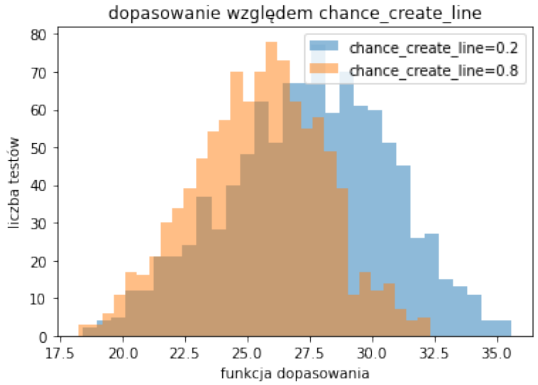
Pierwszy eksperyment obejmował wszystkie funkcje przetrwania. Pozostałe parametry były dobierane jako 0.2 lub 0.8.



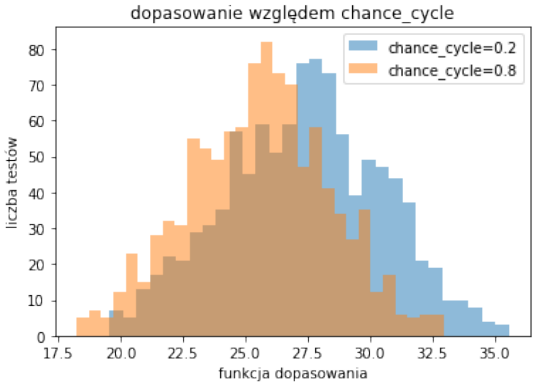

Rysunek 5.8. Rozkład funkcji dopasowania dla pierwszego przeszukiwania siatki hiper-parametrów



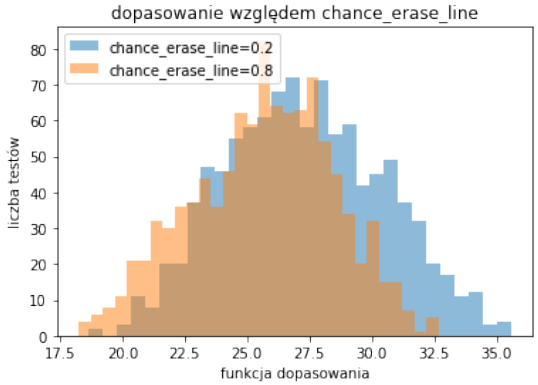

Rysunek 5.9. Rozkład dopasowania względem hiper-parametru



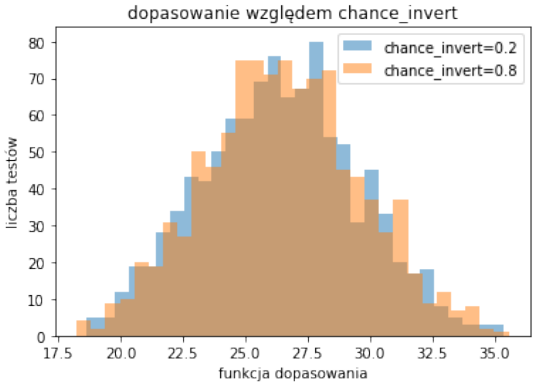

Rysunek 5.10. Rozkład dopasowania względem hiper-parametru



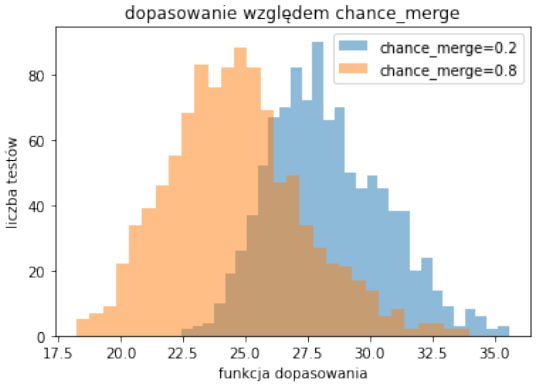

Rysunek 5.11. Rozkład dopasowania względem hiper-parametru



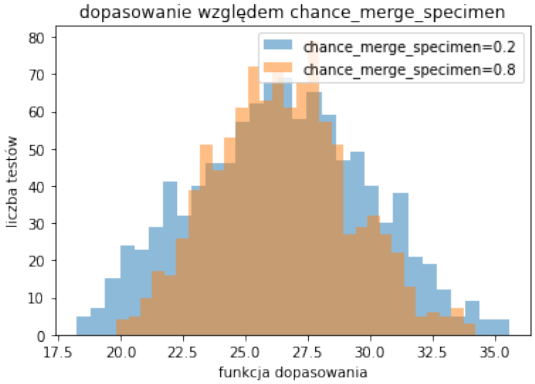

Rysunek 5.12. Rozkład dopasowania względem hiper-parametru



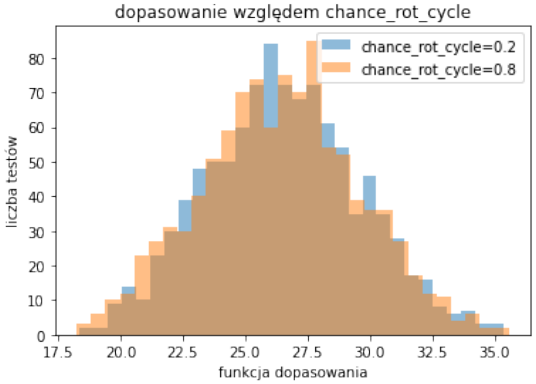

Rysunek 5.13. Rozkład dopasowania względem hiper-parametru



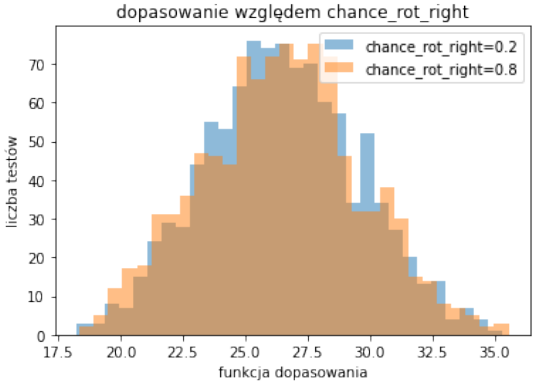

Rysunek 5.14. Rozkład dopasowania względem hiper-parametru



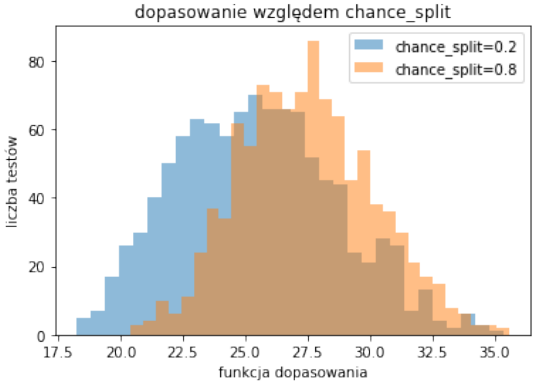

Rysunek 5.15. Rozkład dopasowania względem hiper-parametru




Rysunek 5.16. Rozkład dopasowania względem hiper-parametru




Rysunek 5.17. Rozkład dopasowania względem hiper-parametru

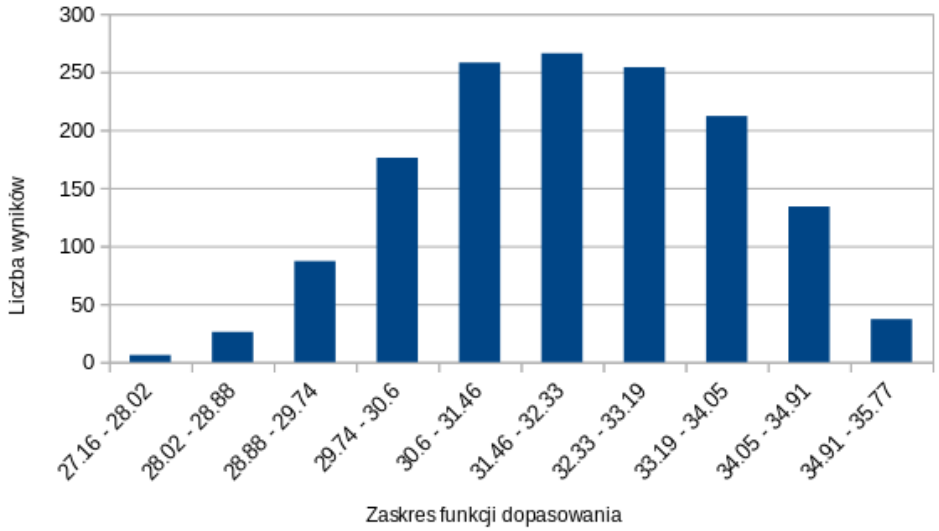



Rysunek 5.18. Rozkład dopasowania względem hiper-parametru

Jak widać niektóre rozkłady są lewoskośne, więc dalsze eksperymenty zawęziliśmy do wartości dla tych rozkładów. Jeżeli był to rozkład z parametrem 0.8 to wartości w dalszych eksperymentach to 0.5, 0.75, 0.9; dla parametru o wartości 0.2: 0.1, 0.25, 0.5. Dla rozkładów symetrycznych przyjęliśmy stałą wartość 0.5 (oprócz chance_merge_specimen – tutaj zostawiono duży rozrzut). Dalej zostały wykorzystane funkcje przetrwania 0 i 1.

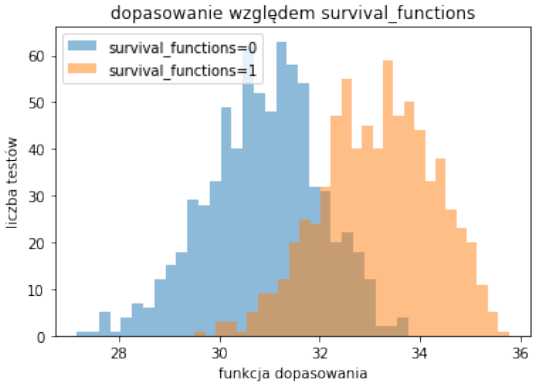
Teraz przestrzeń parametrów wygląda następująco:

```
"survival_functions":      [0, 1],
"chance_create_line":      [0.1, 0.25, 0.5],
"chance_cycle":            [0.1, 0.25, 0.5],
"chance_erase_line":       [0.1, 0.25, 0.5],
"chance_invert":           [0.5],
"chance_merge":            [0.1, 0.25, 0.5],
"chance_merge_specimen":   [0.2, 0.5, 0.8],
"chance_rot_cycle":        [0.5],
"chance_rot_right":        [0.5],
"chance_split":            [0.5, 0.75, 0.9],
```

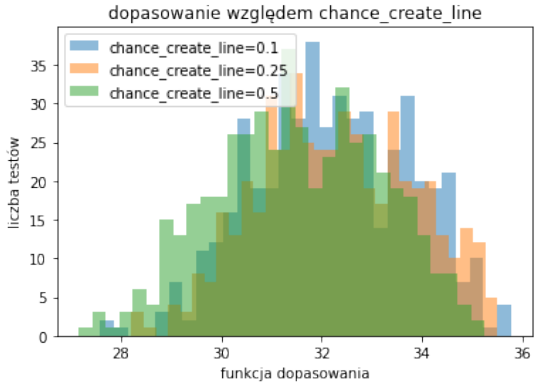



Rysunek 5.19. Rozkład funkcji dopasowania dla pierwszego przeszukiwania siatki hiper-parametrów – zawężona przestrzeń parametrów

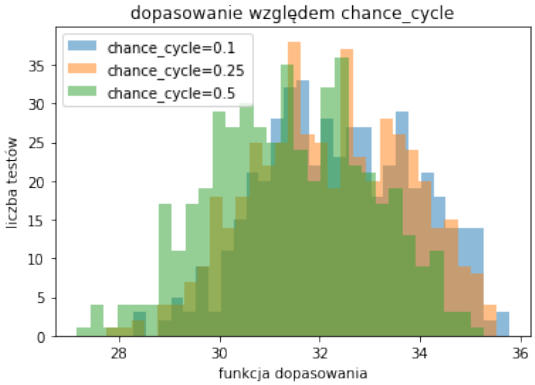
Minimum wzrosło z 18.25 do 27.16, idziemy w dobrą stronę! Ale maksimum wzrosło tylko o 0.2.



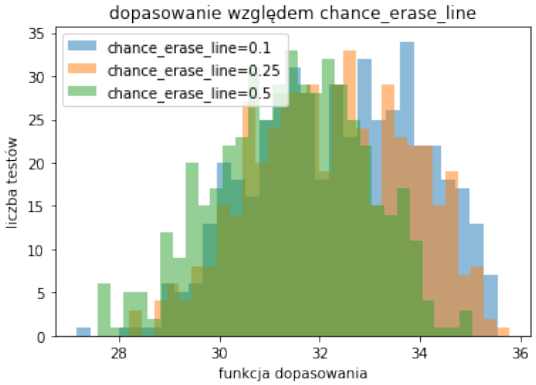

Rysunek 5.20. Rozkład dopasowania względem hiper-parametru



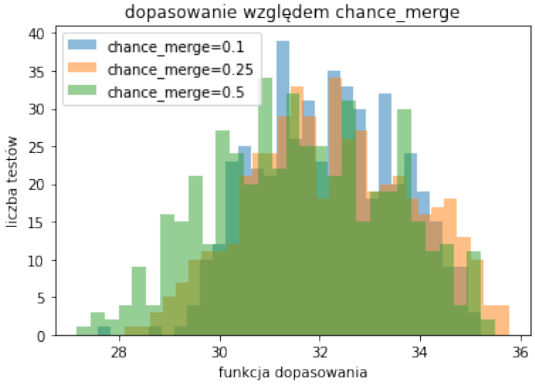

Rysunek 5.21. Rozkład dopasowania względem hiper-parametru



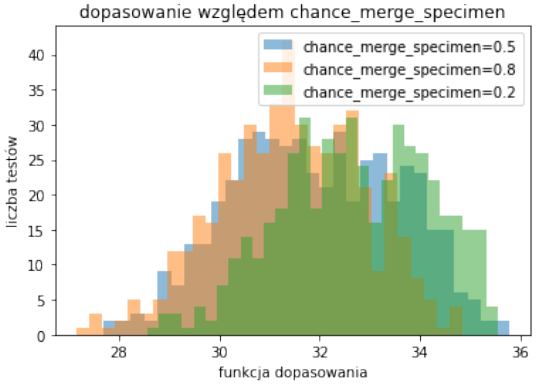

Rysunek 5.22. Rozkład dopasowania względem hiper-parametru



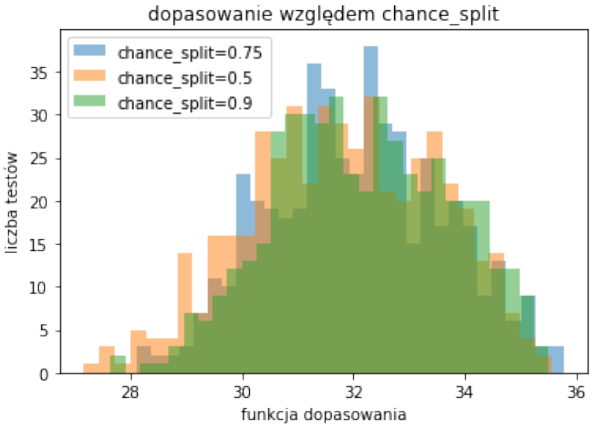

Rysunek 5.23. Rozkład dopasowania względem hiper-parametru




Rysunek 5.24. Rozkład dopasowania względem hiper-parametru




Rysunek 5.25. Rozkład dopasowania względem hiper-parametru




Rysunek 5.26. Rozkład dopasowania względem hiper-parametru

Największy rozrzut powoduje funkcja dopasowania. Zdecydowanie lepiej radzi sobie 1 (przeżywa tylko 1/8 najlepszych osobników, w opcji 0 przeżywa aż 1/4). Najlepsze parametry z wykresów odczytano jako (najbardziej lewoskośny/najwięcej przypadków po prawej/najmniej po lewej):

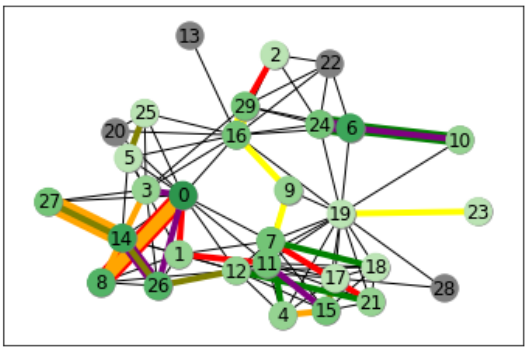
```
"survival_functions":      [1],
"chance_create_line":      [0.1],
"chance_cycle":            [0.1],
"chance_erase_line":       [0.1],
"chance_invert":           [0.5],
"chance_merge":            [0.25],
"chance_merge_specimen":   [0.5],
"chance_rot_cycle":        [0.5],
"chance_rot_right":        [0.5],
"chance_split":            [0.75],
```

Maksymalne dopasowanie (35.77) osiągnięto dla

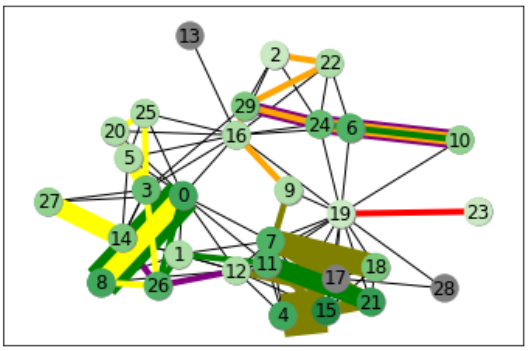
| | |
|---|---|
| chance_create_line | 0.1 |
| chance_cycle | 0.1 |
| chance_erase_line | 0.25 |
| chance_invert | 0.5 |
| chance_merge | 0.25 |
| chance_merge_specimen | 0.5 |
| chance_rot_cycle | 0.5 |
| chance_rot_right | 0.5 |
| chance_split | 0.75 |

Jedyna różnica w chance_erase_line. Na wykresie niebieski(=0.1) i pomarańczowy(=0.25) prawie się pokrywają.

Dla najlepszych parametrów graf miasta prezentuje się następująco:




Rysunek 5.27. Epoka 100, dopasowanie 34.90




Rysunek 5.28. Epoka 1000, dopasowanie 41.96

Nadal widoczne są patologiczne sytuacje. Np wierzchołki 19 i 23 są połączone tylko między sobą.

# 6. Podsumowanie

Problem generowania linii autobusowych jest bardzo skomplikowany. W celu jego rozwiązania, przydatne są algorytmy genetyczne. Z odpowiednią liczbą nowych generacji jesteśmy w stanie osiągnąć ciekawe wyniki. Nie są one jednak w pełni satysfakcjonujące.