

Badania operacyjne Projekt

Jakub Kosmydel
Norbert Morawski
Przemysław Węglik
Bartłomiej Wiśniewski

30 maja 2023

Spis treści

1	Wstęp	3
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	Graf	3
2.2.4	Szukane	4
2.2.5	Hiperparametry	4
2.2.6	Funkcja zysku	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	8
5	Eksperymenty	8
5.1	Eksperymenty proste	8
5.2	Przeszukiwanie siatki hiper-parametrów	10
5.2.1	Dodatkowe operatory genetyczne	15
5.3	Eksperymenty z mapą Krakowa	21
6	Podsumowanie	21
7	Bibliografia	22
8	Podział pracy	22

1 Wstęp

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

Pozycje przystanków są ustawione na sztywno w miejscach inspirowanych ich prawdziwą obecnością w Krakowie, a ich popularność jest zależna od gęstości zaludnienia dookoła nich. 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, przy danym rozłożeniu przystanków. 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.
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 uzyskują więcej punktów niż jedna linia, ale stosujemy prawo malejących przychodów.
5. Maksymalizujemy sumę punktów zebranych przez wszystkie linie.

2.2.2 Dane

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

2.2.3 Graf

1. Wierzchołki to istniejące przystanki z przypisanymi punktami, zależącymi od gęstości zaludnienia w pobliżu.
2. $p(j)$ - wartość punktowa przystanku:

- $p(j) = \sum_{i=0}^{n-1} [w_{j,i} \cdot f(d_{j,i})]$ gdzie $w_{j,i}$ to wartość obiektu (np. liczba mieszkańców w pobliżu) a $d_{j,i}$ to odległość tego bloku od przystanku, f – funkcja malejących zysków.
 - Funkcja liczona dla danego przystanku j
3. Krawędzie to połączenia między przystankami.
 4. Koszt krawędzi to odległości między przystankami.

2.2.4 Szukane

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

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

2.2.5 Hiperparametry

1. α - koszt zatrzymania się na przystanku,
2. β - koszt nowej linii,
3. K - funkcja dopasowania dla długości linii,
4. Δ - koszt nieodwiedzenia przystanku,
5. R - hiperparametr zbiegania.

2.2.6 Funkcja zysku

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

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

$$\lim_{l_j \rightarrow \infty} q_j = \frac{e^R}{l_j} \quad q_j \text{ jest ograniczone nawet jeśli liczba lini jest bardzo duża}$$

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

$$S_i \quad \text{długość ścieżki linii } i \text{ w grafie}$$

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

gdzie

- K – funkcja skalująca długość

Względem oryginalnego modelu zmieniło się:

- Dodano funkcje K

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

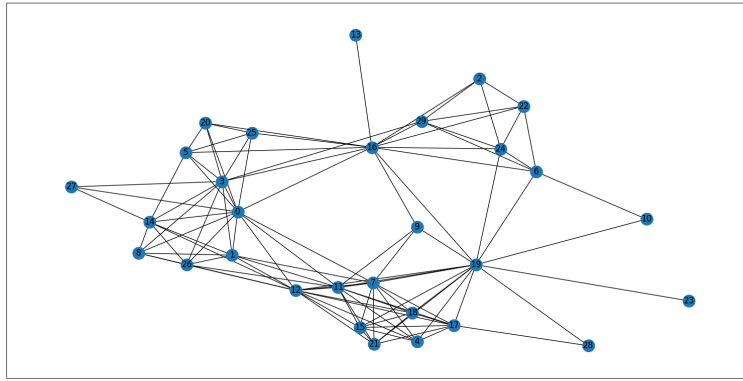
Linia posiada następujące parametry:

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()
        # ordered list of stops
        self.stops = stops
        # list of edges on shortest paths between successive stops
        self.edges = []
```

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 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  $i \leftarrow 0$  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. `exponential_survival(n, lambda)` - pozostawia n osobników w sposób losowy, ale zależny od uzystanej wartości *fitness* i zgodny z rozkładem wykładniczym parametryzowanym przez *lambda*,
5. `exponential_survival_with_protection(best_protected, worst_protected, lambda)` - działa jak *exponential_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 linii i przesuwa je o zadaną (lub losową) liczbę pozycji
2. `cycle_rotation` - losuje pozycje przystanków w linii 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 linii 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 pojedynczych przystanków

5. `cycle_stops_shift(G)` - tworzy nowy genotyp, ustawiając ciągi przystanków z linii obok siebie i wykonując `cycle_rotation` na takim ciągu przystanków

3.6 Krzyżowanie

3.6.1 GenotypeCrossover

1. `merge_genotypes(G1, G2)` - tworzy nowy genotyp poprzez wybranie losowych linii 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 linii z `G1` i `G2` na połowy i jedną z połów każdej linii łączy z połową linii z drugiego genotypu. Z 4 możliwych przypadków połączenia wybiera ten w którym dystans pomiędzy połączonymi przystankami jest minimalny

4 Aplikacja

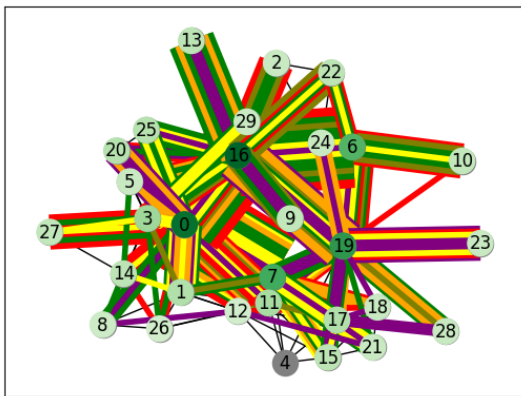
Główny moduł znajduje się w pliku *main.py* (zawiera on główną funkcję *run_simulation*). Szczegółowy opis i instrukcje znajduje się w *README.md*.

Parametry algorytmu można podać zarówno do konstruktora klasy Simulation Engine.

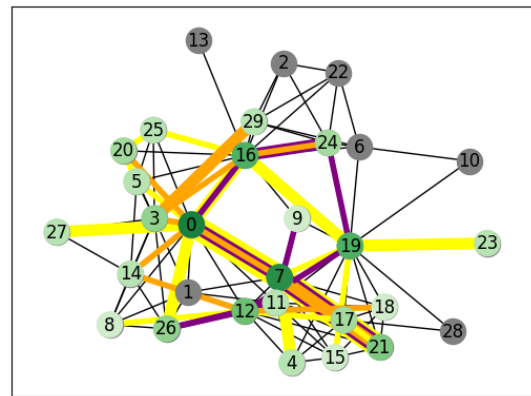
5 Eksperymenty

Na naszym algorytmie przeprowadziliśmy szereg eksperymentów.

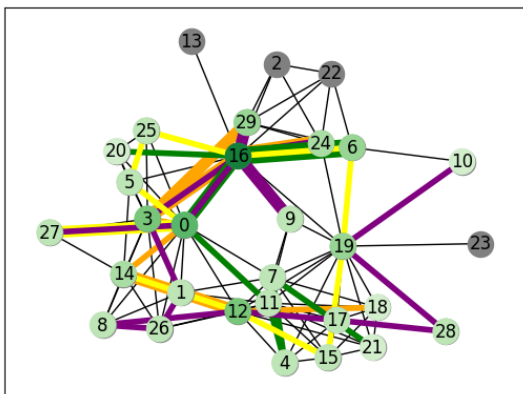
5.1 Eksperymenty proste



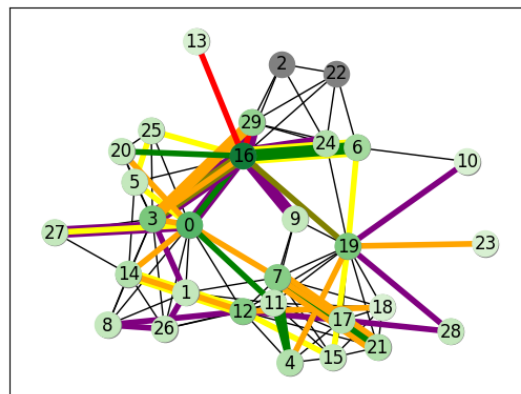
Rysunek 2: Populacja 0
dopasowanie -121.46



Rysunek 3: Populacja 3
dopasowanie 1.71

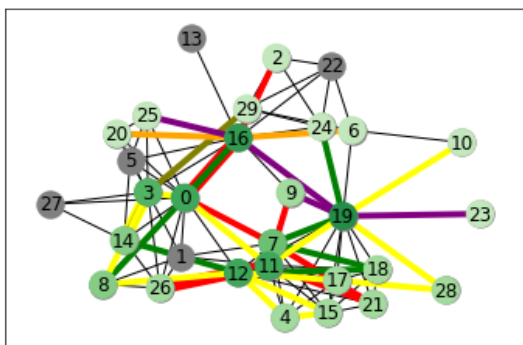


Rysunek 4: Populacja 5
dopasowanie 11.08

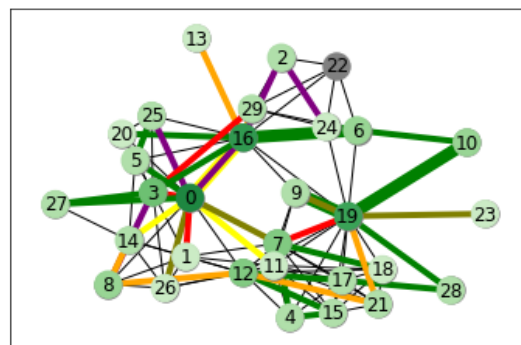


Rysunek 5: Populacja 10
dopasowanie 14.40

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

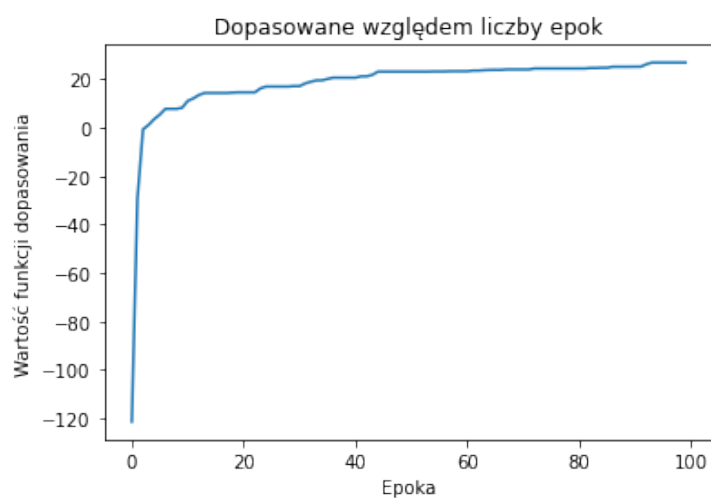


Rysunek 6: Populacja 20
dopasowanie 17.18



Rysunek 7: Populacja 100
dopasowanie 24.64

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



Rysunek 8: 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. Przeszukiwana przestrzeń parametrów:

```
"survival_functions ":      [0 , 1, 2, 3] ,
"chance_create_line ":      [0.1 , 0.8] ,
"chance_cycle ":           [0.1 , 0.8] ,
"chance_erase_line ":       [0.1 , 0.8] ,
"chance_invert ":           [0.1 , 0.8] ,
"chance_merge ":            [0.1 , 0.8] ,
"chance_merge_specimen ":   [0.1 , 0.8] ,
"chance_rot_cycle ":        [0.1 , 0.8] ,
"chance_rot_right ":        [0.1 , 0.8] ,
"chance_split ":            [0.1 , 0.8] ,
```

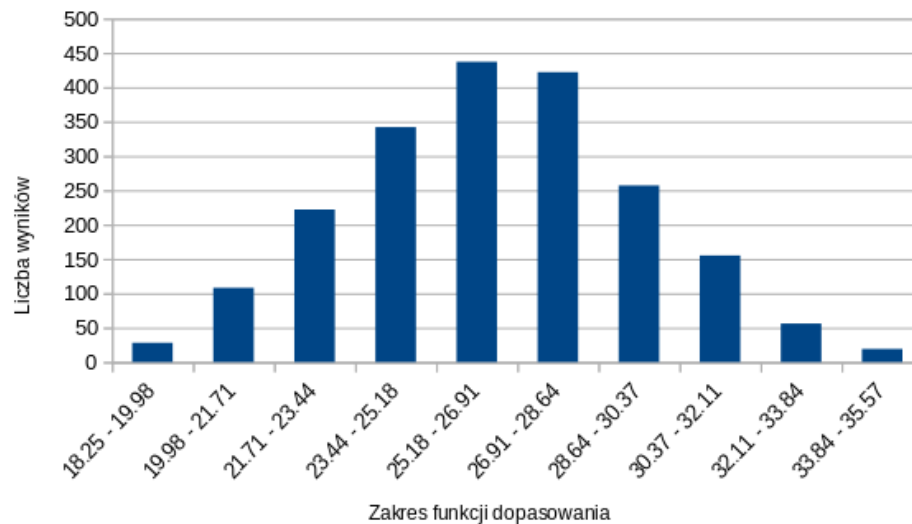
Funkcje przetrwania (opisy funkcji w sekcji 3.4):

0 n_best_survive(N // 4)

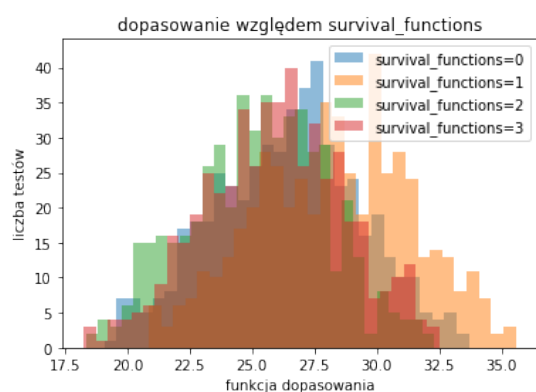
1 n_best_survive(N // 8)

2 n_best_and_m_random_survive(N // 4, N // 10)

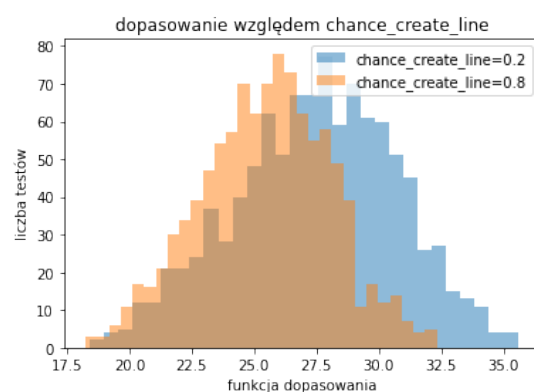
3 n_best_and_m_random_survive(N // 4, N // 20)



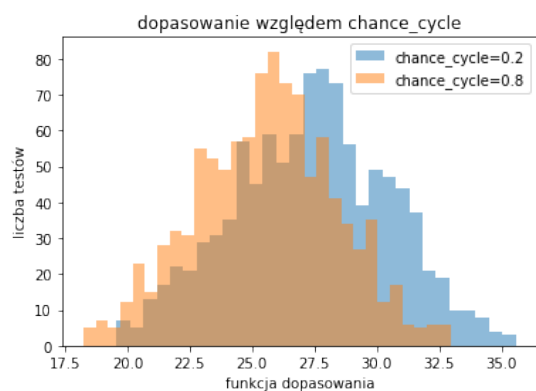
Rysunek 9: Rozkład funkcji dopasowania dla pierwszego przeszukiwania siatki hiper-parametrów



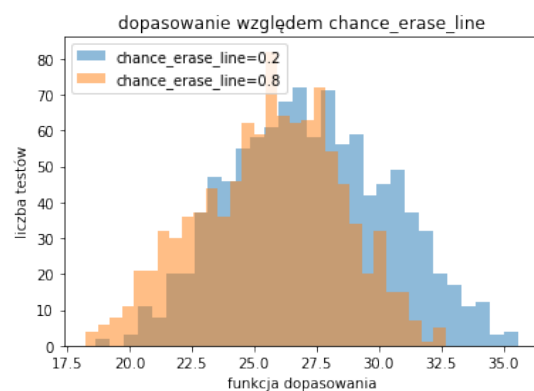
Rysunek 10: Rozkład dopasowania względem hiper-parametru



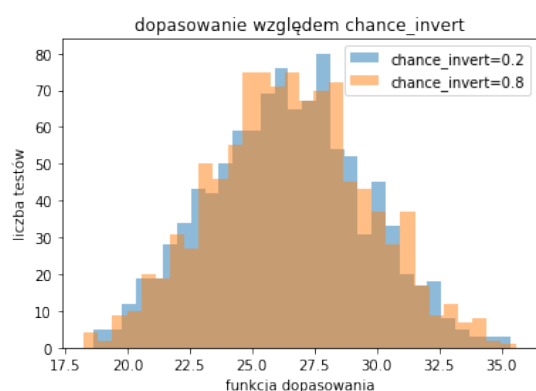
Rysunek 11: Rozkład dopasowania względem hiper-parametru



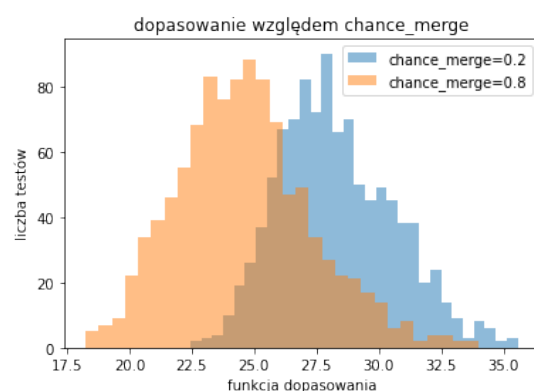
Rysunek 12: Rozkład dopasowania względem hiper-parametru



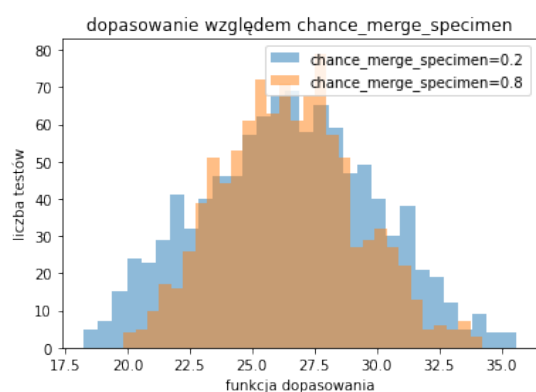
Rysunek 13: Rozkład dopasowania względem hiper-parametru



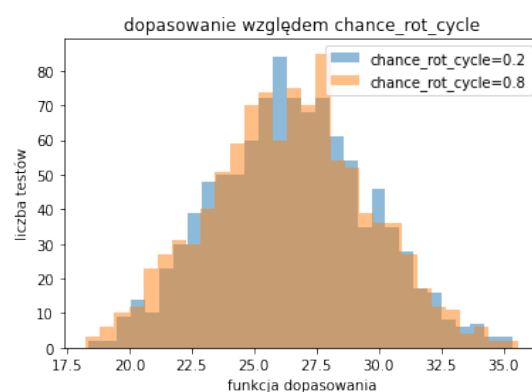
Rysunek 14: Rozkład dopasowania względem hiper-parametru



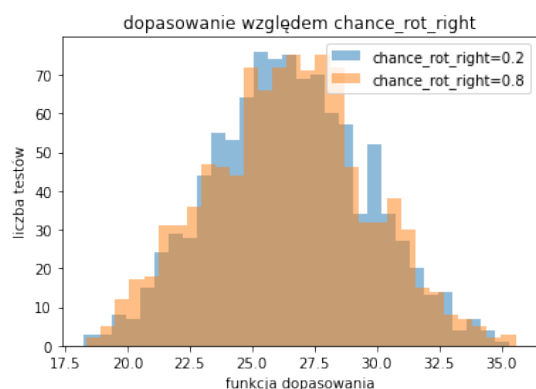
Rysunek 15: Rozkład dopasowania względem hiper-parametru



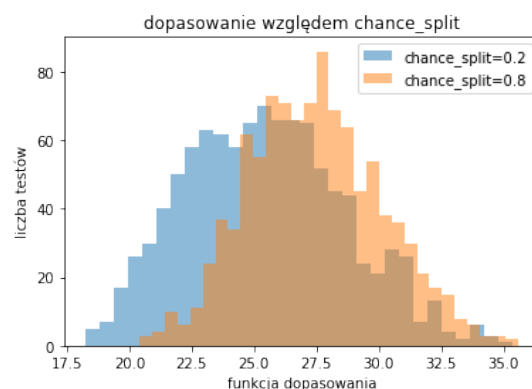
Rysunek 16: Rozkład dopasowania względem hiper-parametru



Rysunek 17: Rozkład dopasowania względem hiper-parametru



Rysunek 18: Rozkład dopasowania względem hiper-parametru



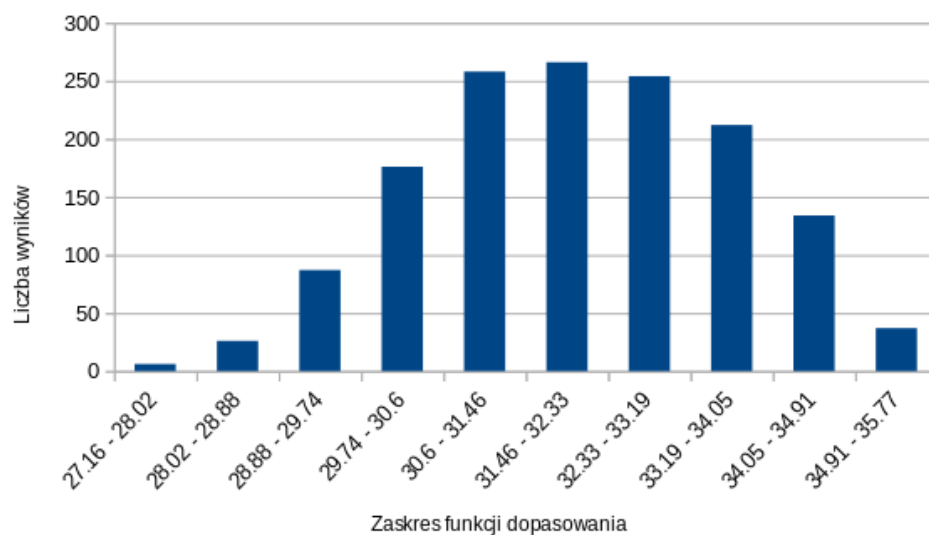
Rysunek 19: Rozkład dopasowania względem hiper-parametru

Jak widać niektóre rozkłady są lewoskośne, więc dalsze eksperymenty zawężiliśmy 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).

Zdecydowanie lepiej radzi sobie funkcja przetrwania 0 od 1 i analogicznie 3 od 2. W poniższym eksperymencie zostały porównane tylko 0 i 1 ale do pozostałych wrócono niżej.

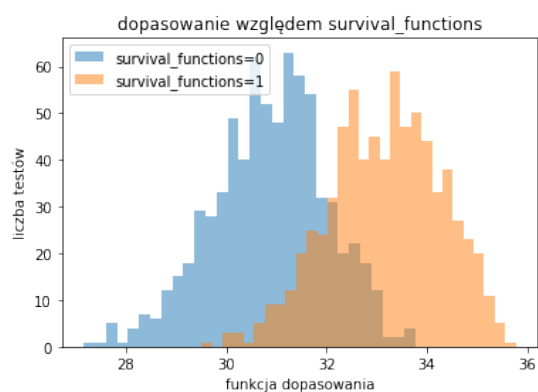
Teraz przestrzeń parametrów wygląda następująco:

```
"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],
```

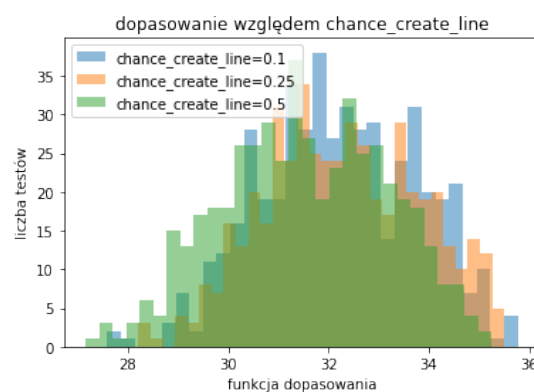


Rysunek 20: Rozkład funkcji dopasowania dla drugiego przeszukiwania siatki hiperparametrów – zawężona przestrzeń parametrów

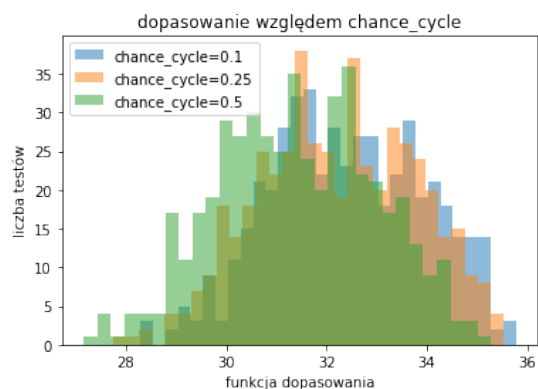
Minimum wzrosło z 18.25 do 27.16, idziemy w dobrą stronę! Ale maksimum wzrosło tylko o 0.2.



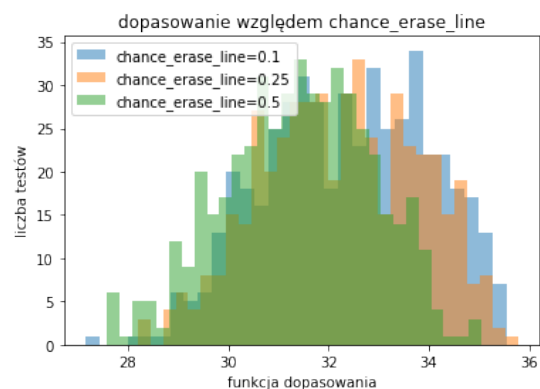
Rysunek 21: Rozkład dopasowania względem hiper-parametru



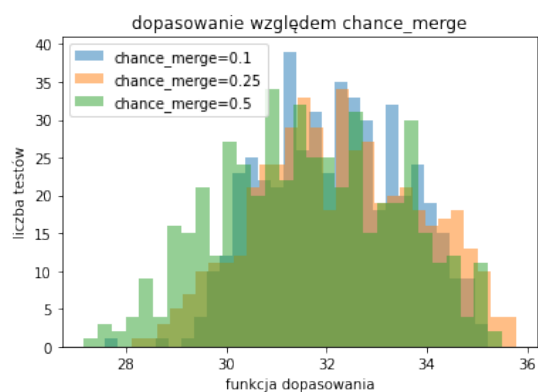
Rysunek 22: Rozkład dopasowania względem hiper-parametru



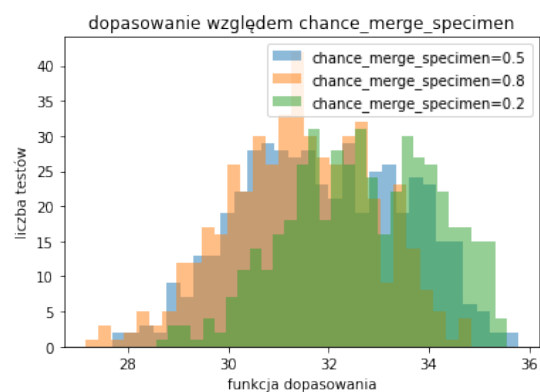
Rysunek 23: Rozkład dopasowania względem hiper-parametru



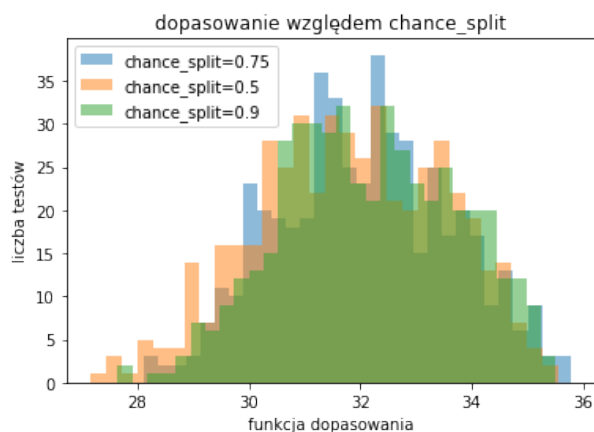
Rysunek 24: Rozkład dopasowania względem hiper-parametru



Rysunek 25: Rozkład dopasowania względem hiper-parametru



Rysunek 26: Rozkład dopasowania względem hiper-parametru



Rysunek 27: Rozkład dopasowania względem hiper-parametru

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

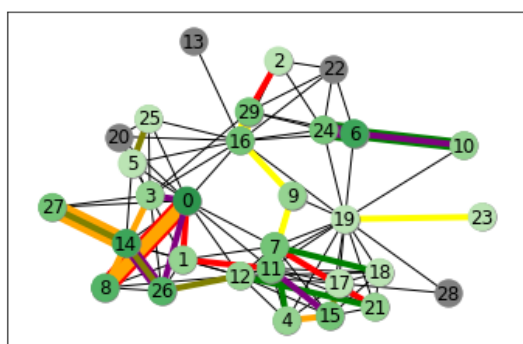
```

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

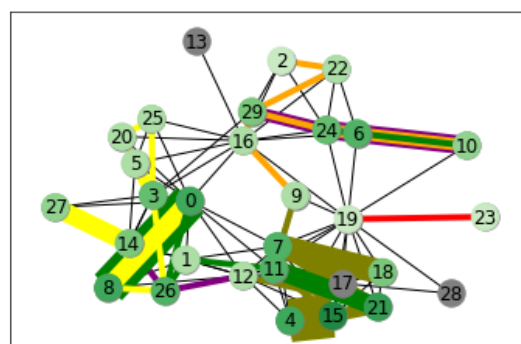
```

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 28: Epoka 100, dopasowanie 34.90



Rysunek 29: Epoka 1000, dopasowanie 41.96

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

5.2.1 Dodatkowe operatory genetyczne

Przetestowaliśmy dodatkowo (poprzednie parametry takie same jak powyżej):

```

"survival_functions ": [1, 4] ,
"chance_erase_stop ":  [0.2, 0.8] ,
"chance_add_stop ":    [0.2, 0.8] ,
"chance_add_stop_mix ": [0.2, 0.8] ,
"chance_replace_stops ": [0.2, 0.8] ,
"chance_replace_stops_proximity ": [0.2, 0.8] ,

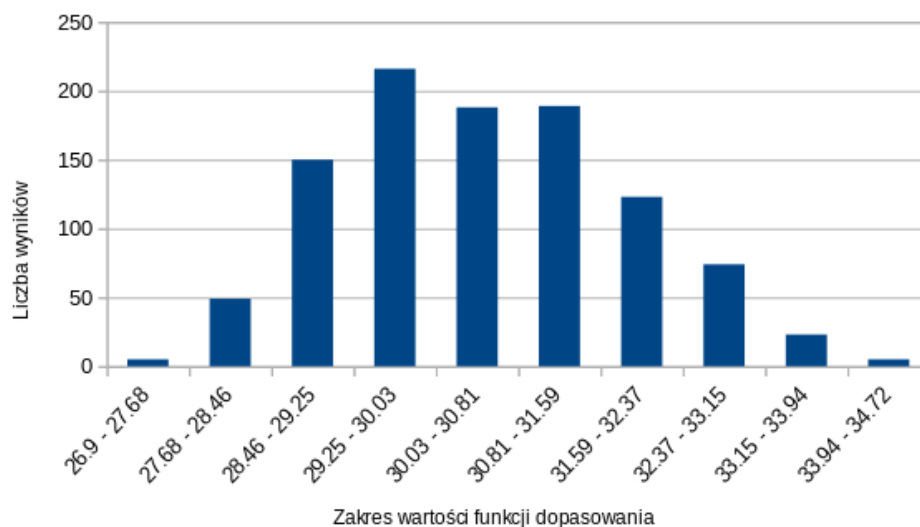
```

"chance_merge_mix": [0.2, 0.8],
 "cycle_stops_shift": [0.2, 0.8],
 "chance_cycle_stops_shift": [0.2, 0.8],
 "chance_line_based_merge": [0.2, 0.8],

Funkcje przetrwania:

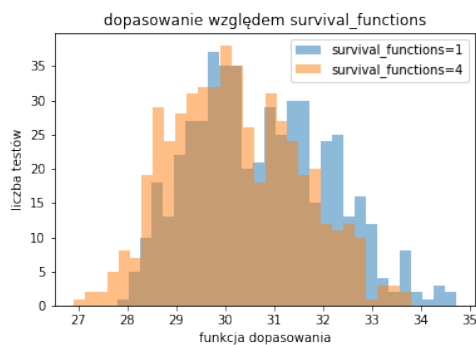
1 n_best_survive(N // 8)

4 n_best_and_m_random_survive(N // 8, N // 20) – połączenie 1 i 3 z poprzednich eksperymentów (najlepsze wyniki)

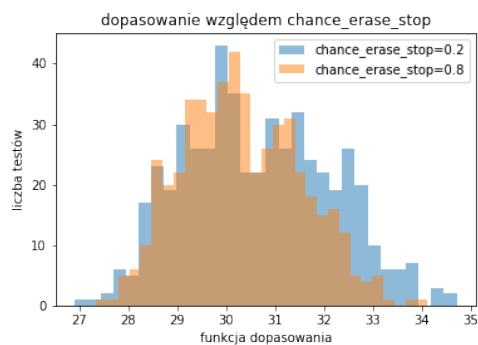


Rysunek 30: Rozkład funkcji dopasowania dla trzeciego przeszukiwania siatki hiperparametrów – nowe operatory

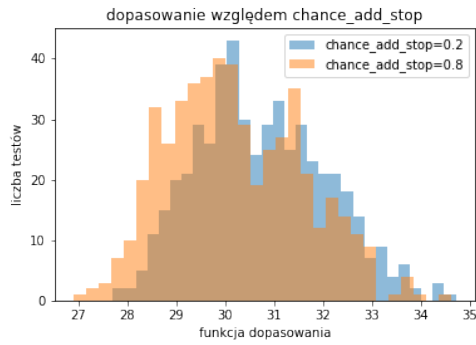
Minimum spadło z 27.16 do 26.90, a maksimum z 35.77 do 34.72.



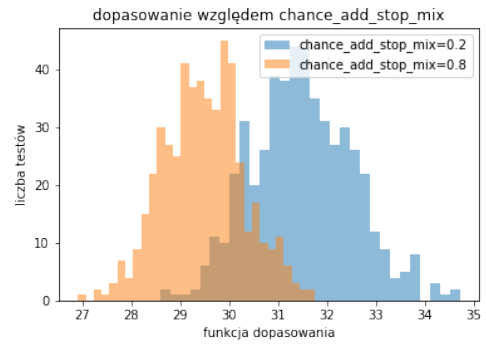
Rysunek 31: Rozkład dopasowania względem hiperparametru



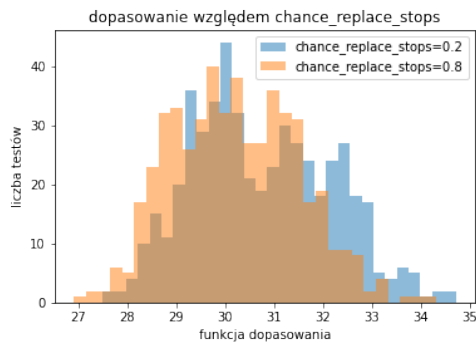
Rysunek 32: Rozkład dopasowania względem hiperparametru



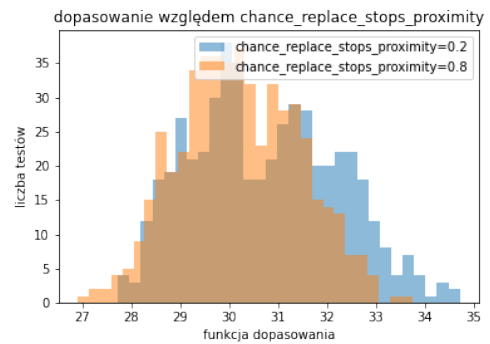
Rysunek 33: Rozkład dopasowania względem hiper-parametru



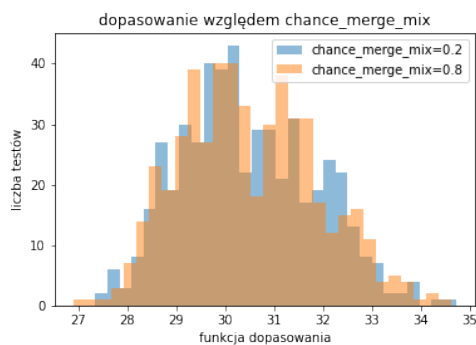
Rysunek 34: Rozkład dopasowania względem hiper-parametru



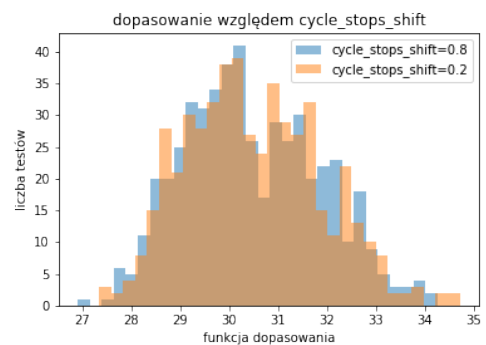
Rysunek 35: Rozkład dopasowania względem hiper-parametru



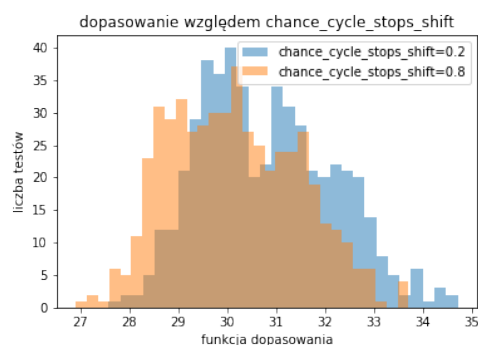
Rysunek 36: Rozkład dopasowania względem hiper-parametru



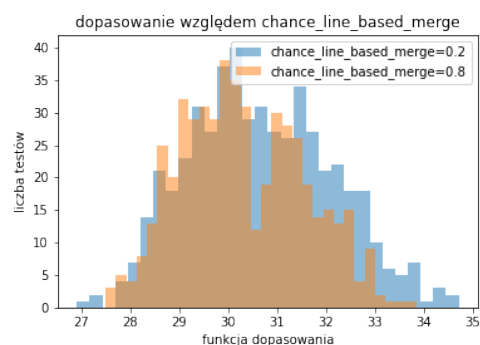
Rysunek 37: Rozkład dopasowania względem hiper-parametru



Rysunek 38: Rozkład dopasowania względem hiper-parametru



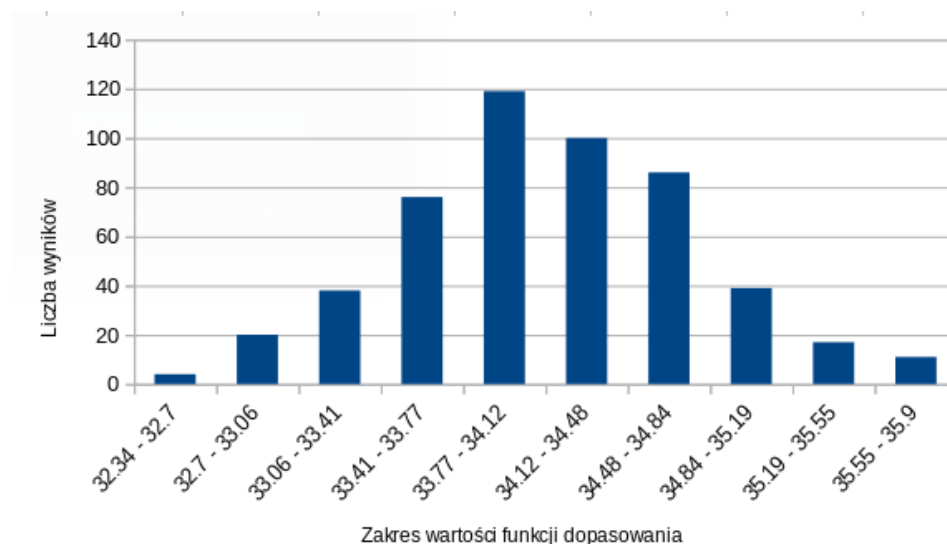
Rysunek 39: Rozkład dopasowania względem hiper-parametru



Rysunek 40: Rozkład dopasowania względem hiper-parametru

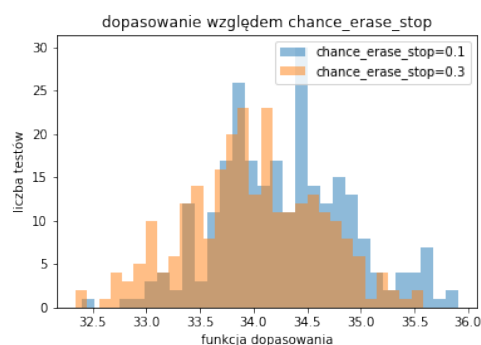
Ponownie zawężony została przestrzeń przeszukiwań parametrów (zgodnie z zasadami z wcześniejszych eksperymentów):

"survival_functions"	: [1] ,
"chance_erase_stop"	: [0.1 , 0.3] ,
"chance_add_stop"	: [0.1 , 0.3] ,
"chance_add_stop_mix"	: [0.1 , 0.3] ,
"chance_replace_stops"	: [0.1 , 0.3] ,
"chance_replace_stops_proximity"	: [0.1 , 0.3] ,
"chance_merge_mix"	: [0.2 , 0.8] ,
"cycle_stops_shift"	: [0.2 , 0.8] ,
"chance_cycle_stops_shift"	: [0.1 , 0.3] ,
"chance_line_based_merge"	: [0.1 , 0.3] ,

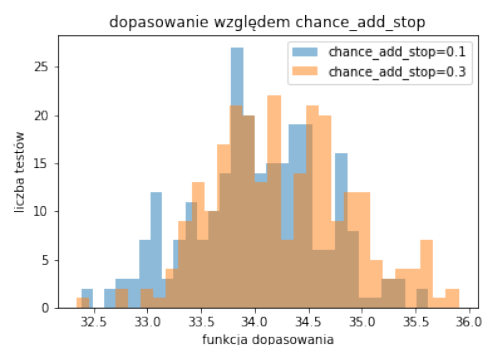


Rysunek 41: Rozkład funkcji dopasowania dla czwartego przeszukiwania siatki hiper-parametrów – nowe operatory

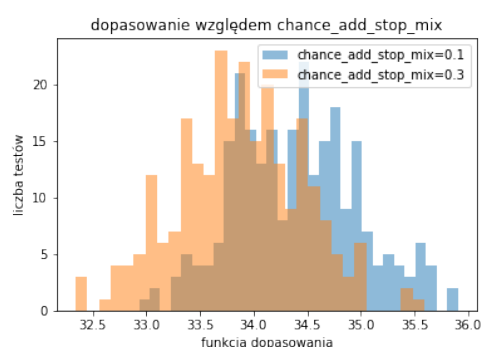
Minimum wzrosło z 32.34 do 26.90, a maksimum z 34.72 do 35.90 (przed implementacją nowych operatorów było to 35.77). Mamy nowy globalnie lepszy wynik.



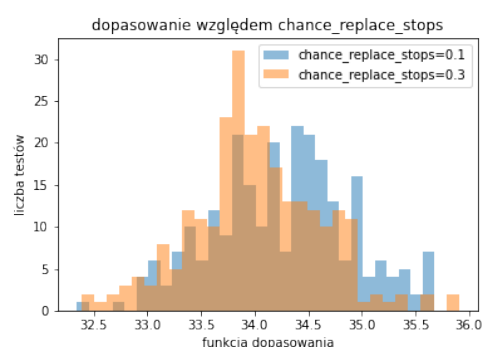
Rysunek 42: Rozkład dopasowania względem hiper-parametru



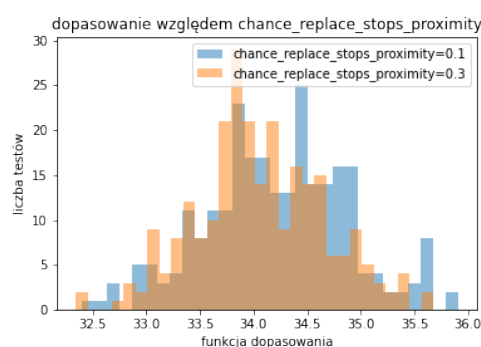
Rysunek 43: Rozkład dopasowania względem hiper-parametru



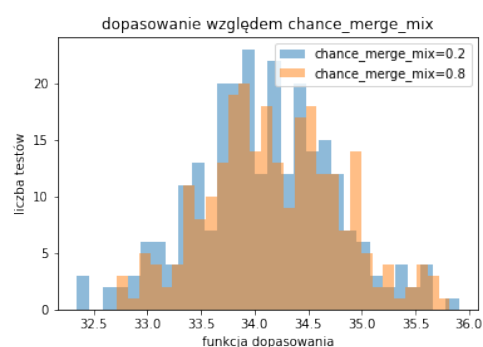
Rysunek 44: Rozkład dopasowania względem hiper-parametru



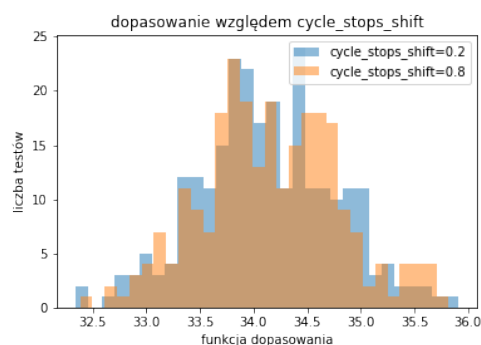
Rysunek 45: Rozkład dopasowania względem hiper-parametru



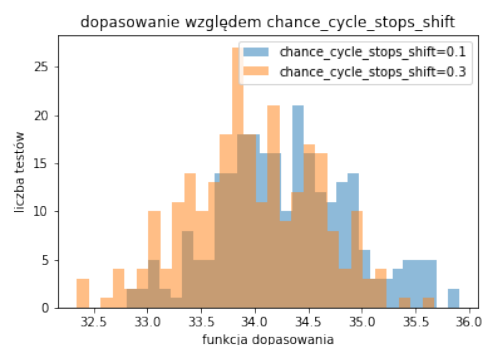
Rysunek 46: Rozkład dopasowania względem hiper-parametru



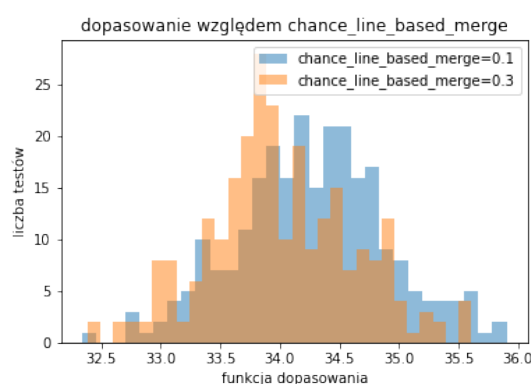
Rysunek 47: Rozkład dopasowania względem hiper-parametru



Rysunek 48: Rozkład dopasowania względem hiper-parametru



Rysunek 49: Rozkład dopasowania względem hiper-parametru



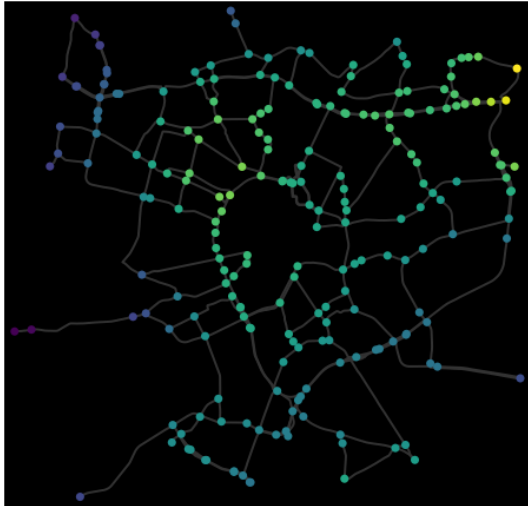
Rysunek 50: Rozkład dopasowania względem hiper-parametru

Teraz całość najlepszych hiper-parametrów wygląda następująco:

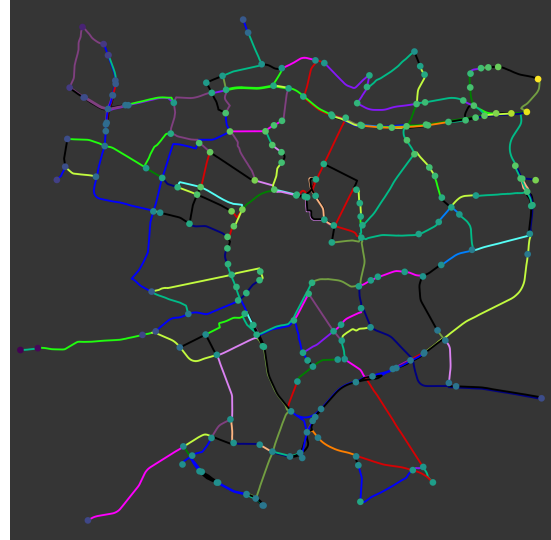
```
"survival_functions"           : [1] ,
"chance_create_line"           : [0.1] ,
"chance_cycle"                  : [0.1] ,
"chance_erase_line"            : [0.1] ,
"chance_invert"                 : [0.5] ,
"chance_merge"                  : [0.25] ,
"chance_merge_specimen"         : [0.5] ,
"chance_rot_cycle"              : [0.5] ,
"chance_rot_right"              : [0.5] ,
"chance_split"                  : [0.75] ,
"chance_erase_stop"             : [0.1] ,
"chance_add_stop"               : [0.3] ,
"chance_add_stop_mix"          : [0.1] ,
"chance_replace_stops"          : [0.1] ,
"chance_replace_stops_proximity": [0.1] ,
"chance_merge_mix"              : [0.5] , // brak wpływu
"cycle_stops_shift"             : [0.5] , // brak wpływu
"chance_cycle_stops_shift"      : [0.1] ,
"chance_line_based_merge"       : [0.1] ,
```

5.3 Eksperymenty z mapą Krakowa

Udało nam się pobrać mapę Krakowa dzięki bibliotece OSMNX (OpenStreetMap NetworkX). Dane o ludności pobrano z msip.krakow.pl. Mapa z nałożoną punktacją wierzchołków (wg modelu) wygląda następująco:



Rysunek 51: Mapa Krakowa z danymi o ludności



Rysunek 52: Wynik algorytmu

Jaśniejszy kolor wierzchołka oznacza wyższą wartość punktową. Po uruchomieniu naszego algorytmu, uzyskaliśmy: Ile przystanków ma ile linii:

```
2: 10 (10 linii po 2 przystanki , itd.)
3: 9
5: 8
4: 8,
8: 6,
11: 4,
6: 4,
7: 3,
9: 3,
13: 2,
14: 2,
20: 1,
26: 1,
19: 1,
15: 1
```

Liczba linii: 63

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.

7 Bibliografia

1. Wikipedia, prawo malejących przychodów

8 Podział pracy

- Jakub Kosmydel - 25%
 - Implementacja algorytmu Grid Search do poszukiwania optymalnych hiperparamterów
 - Wizualizacje eksperymentów
 - Dokumentacja
- Norbert Morawski - 25%
 - Przeszukiwanie przestrzeni hiperparametrów gridsearchem
 - Zaimportowanie rzeczywistych danych przy użyciu OpenStreetMap
 - Eksperymenty z optymalnymi paramterami
 - Dokumentacja
- Przemysław Węglik - 25%
 - Generacja grafów losowych
 - Prace nad architekturą projektu (Simulation Engine)
 - Pierwsze eksperymenty
- Bartłomiej Wiśniewski - 25%
 - Stworzenie większości algorytmów mutujących i krzyżujących
 - Prace nad architekturą projektu (klasy Line i Genotype)
 - Prace nad utrzymaniem kodu (typing, rewizje i śledzenie tasków)