Przeszukiwanie i optymalizacja
Dokumentacja projektu
Sieć teleinformatyczna z ograniczeniami

Wojciech Grunwald (311566)

28 stycznia 2024

1 Treść zadania

Przy użyciu Algorytmu Ewolucyjnego zaprojektować sieć teleinformatyczną minimalizującą liczbę użytych systemów teletransmisyjnych o różnej modularności m, dla $(m = 1, m > 1, m >> 1)$*. Sieć opisana za pomocą grafu* $G = (N, E)$*, gdzie N jest zbiorem węzłów, a E jest zbiorem krawędzi. Funkcja pojemności krawędzi opisana jest za pomoca wzoru* $f_e(o) = \lceil \frac{o}{m} \rceil$*. Zbiór zapotrzebowań D, pomiędzy każdą parą węzłów opisuje macierz zapotrzebowań i jest dany. Dla każdego zapotrzebowania istnieją co najmniej 2 predefiniowane scieżki. Sprawdzić, jak wpływa na koszt rozwiązania agregacja zapotrzebowań, tzn. czy zapotrzebowanie realizowane jest na jednej ścieżce (agregacja), czy dowolnie na wszystkich ścieżkach w ramach zapotrzebowania (brak agregacji, pełna dezagregacja). Dobrać optymalne prawdopodobieństwo operatorów genetycznych oraz liczność populacji. Dane pobrać ze strony http://sndlib.zib.de/home.action, dla sieci polskiej.*

2 Opis projektu i jego interpretacja

W projekcie będę miał do czynienia z pewną siecią złożoną z **N** wierzchołków odpowiadających miastom i pewnej liczbie krawędzi (**E=18**) łączących te miasta w jakiś sposób. Zadaniem algorytmu ewolucyjnego będzie **minimalizacja użytych systemów teletransmisyjnych**, gdzie założę konkretną modularność m takich systemów. Modularność systemu teletransmisyjnego m informuje o tym, ile wartości wagi (zamiennie: wagi) o (np. w Gb) zmieści dany system, tzn., przykładowo: jeżeli mamy krawędź o wadze o=51 Gb i m=5, to musimy zapewnić dla tej konkretnej krawędzi $\left\lceil \frac{o=51}{m=5} \right\rceil = 11$ systemów.
Zadanie sprowadzać się będzie więc do następującego zadania optymalizacji:

$$min\ F = min \sum_{e \in E} \left\lceil \frac{o_e}{m} \right\rceil \tag{1}$$

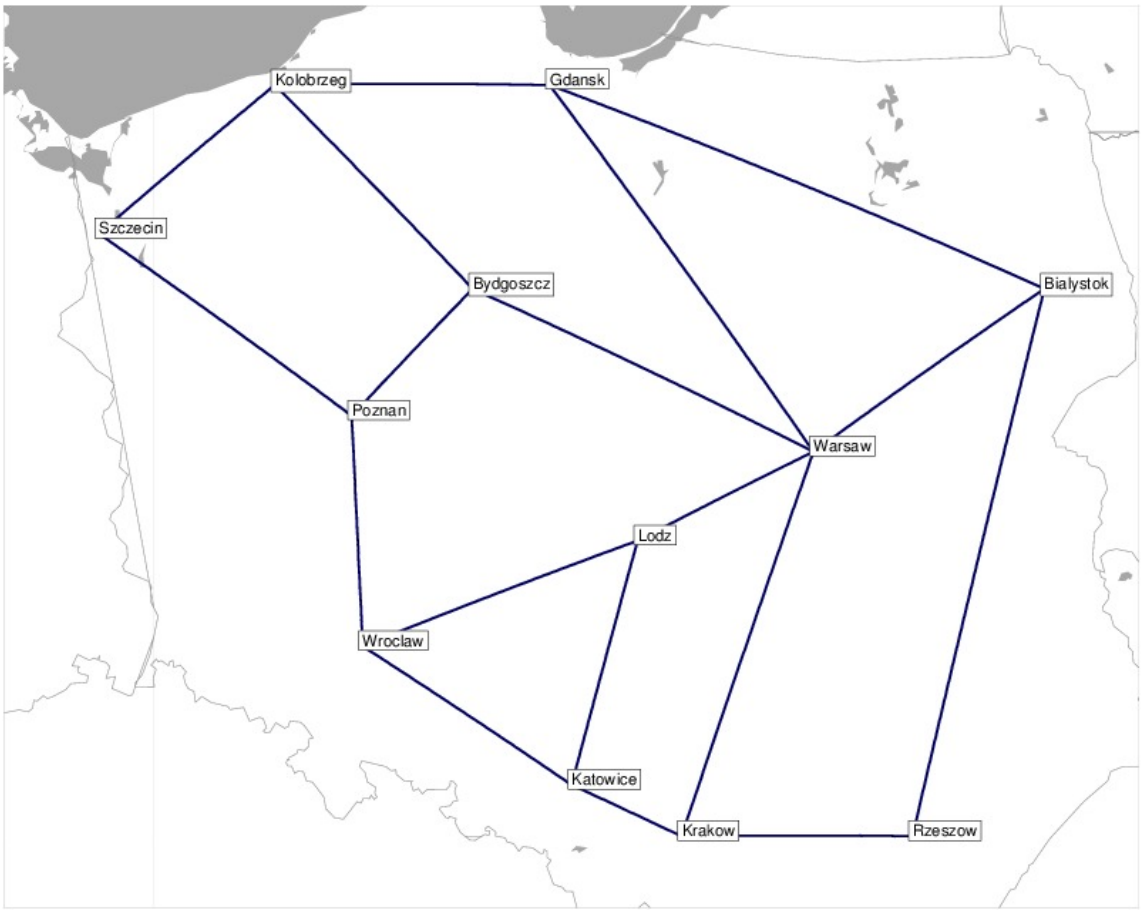
gdzie:

- F - funkcja minimalizowana (funkcja odwrotna do funkcji przystosowania osobnika)
- o_e - to waga danej krawędzi
- m - modularność systemów teletransmisyjnych
- E - zbiór wszystkich krawędzi

Docelowy problem został przekonwertowany na zadanie **maksymalizacji**, biorąc odwrotność funkcji F ze względu na komaptybilność niektórych operacji genetycznych jedynie z typem maksymalizacyjnym algorytmu.

2.1 Zbiór danych

Zbiór danych pochodzić będzie ze strony http://sndlib.zib.de/home.action. W zakładce *Problem instances* z lewej strony witryny. Wybrałem network *polska* i zbiór danych pierwszy z listy: *polska–D-B-M-N-C-A-N-N.*



Rysunek 1: Mapa przedstawiająca sieć miast

```
NODES (
  Gdansk ( 18.60 54.20 )
  Bydgoszcz ( 17.90 53.10 )
  Kolobrzeg ( 16.10 54.20 )
  Katowice ( 18.80 50.30 )
  Krakow ( 19.80 50.00 )
  Bialystok ( 23.10 53.10 )
  Lodz ( 19.40 51.70 )
  Poznan ( 16.80 52.40 )
  Rzeszow ( 21.90 50.00 )
  Szczecin ( 14.50 53.40 )
  Warsaw ( 21.00 52.20 )
  Wroclaw ( 16.90 51.10 )
)
```

Rysunek 2: Węzły sieci (miasta) - ponumerowane od 0 do 11

Nasz zbiór reprezentuje pewną sieć **N** miast, w tym przypadku **N**=12. Dodatkowym narzutem na realizację projektu będzie zbiór zapotrzebowań **D (Demands)**, który będzie określal dla każdej pary (66 par) miast, jakie minimalnie wagi muszą posiadać krawędzie na danej ścieżce (lub ścieżkach, zob. dalej - agregacja/dezagregacja). Takich par miast będzie:

$$\frac{\mathbf{N}(\mathbf{N}+1)}{2} = 66 \tag{2}$$

Wycinek z macierzy zapotrzebowań **D**:

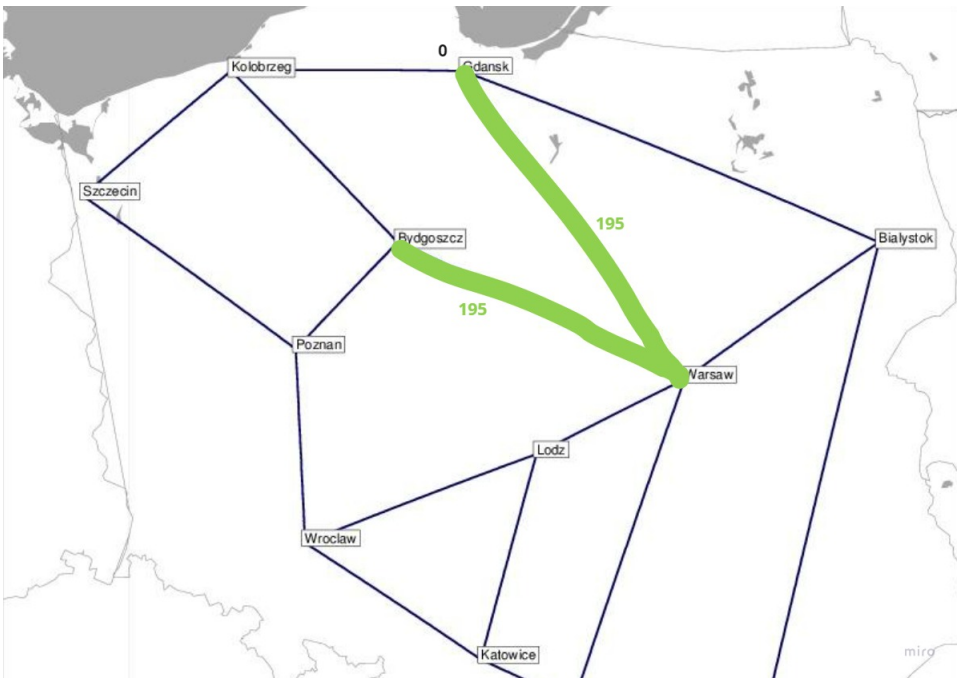
```
DEMANDS (
  Demand_0_1 ( Gdansk Bydgoszcz ) 1 195.00 UNLIMITED
  Demand_0_2 ( Gdansk Kolobrzeg ) 1 158.00 UNLIMITED
  Demand_0_3 ( Gdansk Katowice ) 1 174.00 UNLIMITED
  Demand_0_4 ( Gdansk Krakow ) 1 101.00 UNLIMITED
  Demand_0_5 ( Gdansk Bialystok ) 1 198.00 UNLIMITED
  Demand_0_6 ( Gdansk Lodz ) 1 158.00 UNLIMITED
  Demand_0_7 ( Gdansk Poznan ) 1 182.00 UNLIMITED
  Demand_0_8 ( Gdansk Rzeszow ) 1 154.00 UNLIMITED
  Demand_0_9 ( Gdansk Szczecin ) 1 175.00 UNLIMITED
  Demand_0_10 ( Gdansk Warsaw ) 1 122.00 UNLIMITED
  Demand_0_11 ( Gdansk Wroclaw ) 1 114.00 UNLIMITED
  Demand_1_2 ( Bydgoszcz Kolobrzeg ) 1 179.00 UNLIMITED
  Demand_1_3 ( Bydgoszcz Katowice ) 1 117.00 UNLIMITED
  Demand_1_4 ( Bydgoszcz Krakow ) 1 105.00 UNLIMITED
  Demand_1_5 ( Bydgoszcz Bialystok ) 1 159.00 UNLIMITED
  Demand_1_6 ( Bydgoszcz Lodz ) 1 198.00 UNLIMITED
  Demand_1_7 ( Bydgoszcz Poznan ) 1 189.00 UNLIMITED
  Demand_1_8 ( Bydgoszcz Rzeszow ) 1 166.00 UNLIMITED
```

Rysunek 3: Mapa przedstawiająca sieć

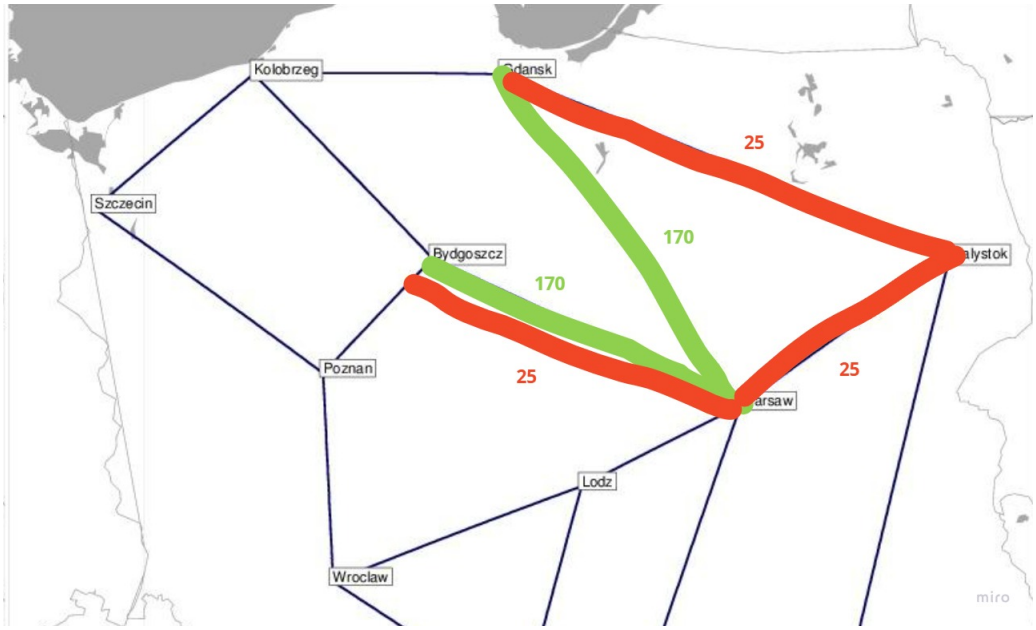
Przykładowo bierzemy *Demand_0_1* i widzimy, że dla ścieżki (scieżek) wiodących z Gdańska do Bydgoszczy na każdej krawędzi musimy zapewnić wartość wag na poziomie 195 (na każdej krawędzi na ścieżce) (**agregacja**) (taką samą wartość wag na każdej krawędzi na poszczególnych ścieżkach, które sumują się do 195 - **dezagregacja**)).

2.2 Agregacja i dezagregacja

Przyjrzymy się bardziej zagadnieniu **agregacji** i **dezagregacji**, posługując się graficznym przedstawieniem:



Rysunek 4: Agregacja dla Demand_0_1



Rysunek 5: Dezagregacja dla Demand_0_1

Algorytm będzie sprawdzał dwa warianty swojego działania:

- w przypadku, gdy dozwolona jest tylko pełna **agregacja** (możliwa tylko jedna ścieżka dla danego zapotrzebowania
- w przypadku, gdy dozwolona jest również **dezagregacja** (możliwe wiele ścieżek dla danego zapotrzebowania)

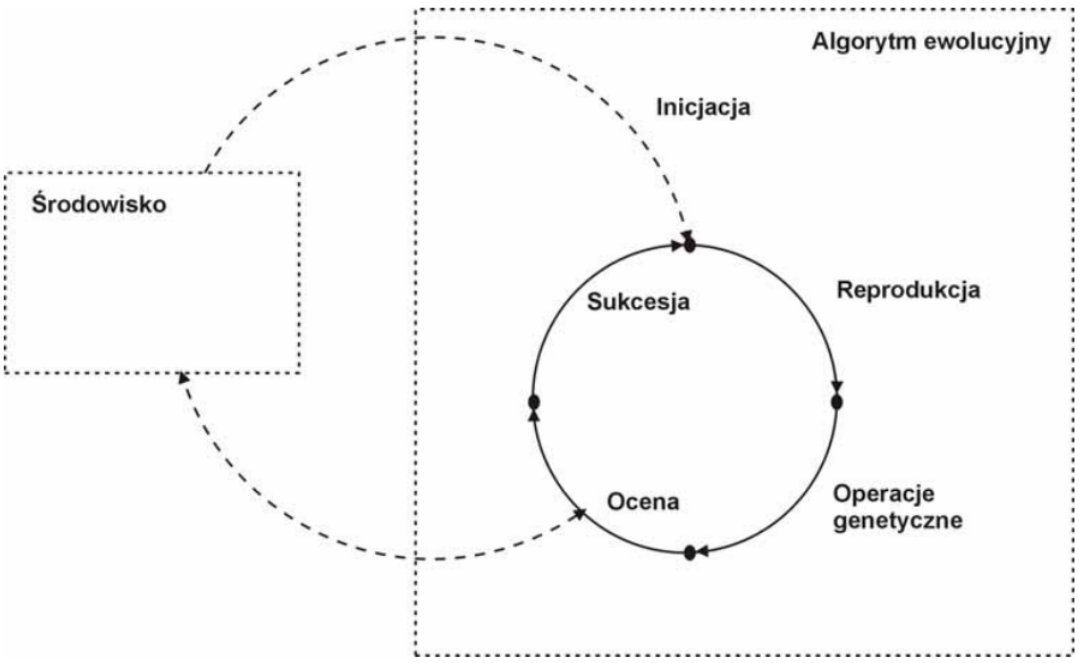
Możliwe ścieżki *Gdańsk-Białystok-Warszawa-Bydgoszcz* i *Gdańsk-Warszawa-Bydgoszcz* są również określone w specyfikacji problemu jako *możliwe ścieżki (**admissible paths**)*, które są zbiorem wszystkich dostępnych ścieżek dla wszystkich macierzy zapotrzebowania:

```
ADMISSIBLE_PATHS (
  Demand_0_1 (
    P_0 ( Link_0_2 Link_1_2 )
    P_1 ( Link_0_10 Link_1_10 )
    P_2 ( Link_0_2 Link_2_9 Link_7_9 Link_1_7 )
    P_3 ( Link_0_5 Link_5_10 Link_1_10 )
    P_4 ( Link_0_10 Link_4_10 Link_3_4 Link_3_11 Link_7_11 Link_1_7 )
    P_5 ( Link_0_2 Link_2_9 Link_7_9 Link_7_11 Link_3_11 Link_3_4 Link_4_10 Link_1_10 )
    P_6 ( Link_0_5 Link_5_8 Link_4_8 Link_4_10 Link_1_10 )
  )
  Demand_0_2 (
    P_0 ( Link_0_2 )
    P_1 ( Link_0_10 Link_1_10 Link_1_2 )
    P_2 ( Link_0_5 Link_5_10 Link_1_10 Link_1_2 )
    P_3 ( Link_0_10 Link_4_10 Link_3_4 Link_3_11 Link_7_11 Link_1_7 Link_1_2 )
    P_4 ( Link_0_10 Link_1_10 Link_1_7 Link_7_9 Link_2_9 )
    P_5 ( Link_0_5 Link_5_8 Link_4_8 Link_4_10 Link_1_10 Link_1_2 )
    P_6 ( Link_0_5 Link_5_10 Link_4_10 Link_3_4 Link_3_11 Link_7_11 Link_1_7 Link_1_2 )
  )
```

Rysunek 6: Wycinek możliwych ścieżek dla zapotrzebowań

Jak widać z zamieszczonego zrzutu ekranu - do wizualizacji **agregacji** użyłem ścieżki ***P_1***, a **dezagregacji** ścieżek ***P_1*** i ***P_3***.

3 Algorytm ewolucyjny, Poteralski (2004)



Rysunek 7: Schemat działania algorytmu ewolucyjnego, źródło: Poteralski (2004)

Algorytm ewolucyjny przetwarza pewną populację P_t osobników. **Jest rozwinięciem i uogólnieniem algorytmu genetycznego Hollanda**.
Do danego problemu możemy podejść na wiele sposobów: zarówno w kwestii kodowania, jak i definiowania szeregu operatorów (np. w zadaniu komiwojażera). Należy jednoznacznie określić:

- schemat działania algorytmu ewolucyjnego;
- metodę selekcji;
- sposób kodowania i operatory genetyczne;
- środowisko działania AE

3.1 Wersja pseudokodowa

procedure EvolutionaryAlgorithm():
    begin
    t:=0
    inicjacja $\mathbf{P}^0$
    ocena $\mathbf{P}^0$
    while (not warunek stopu) **do**
        begin
        $\mathbf{T}^t$:= reprodukcja $\mathbf{P}^t$
        $\mathbf{O}^t$:= operacje genetyczne $\mathbf{T}^t$
        Ocena $\mathbf{O}^t$
        $\mathbf{P}^{t+1}$:= sukcesja($\mathbf{P}^t$, $\mathbf{O}^t$)
        t:=t+1
        end
    end

3.2 Wersja tekstowa

EvolutionaryAlgorithm():
    # Wygeneruj populację początkową, losując z dostępnej przestrzeni przeszukiwań
    # Ewauluuj wartości funkcji przystosowania dla każdego osobnika populacji początkowej
    * Aż do warunku zatrzymania wykonuj w pętli:
        1. Wybierz osobników do reprodukcji (**preselekcja**)
        2. Rozmnażaj nowe osobniki operacjami **krzyżowania** i **mutacji** w celu uzyskania potomstwa
        3. Oceń populację potomną
        4. Zastąp najmniej przystosowane osobniki bardziej przystosowanymi osobnikami (sukcesja/postselekcja, uwzględnienie populacji rodziców i populacji potomnej)

3.3 Opis teoretyczny

Po kolei w pętli głównej wykonywane są metody:

- Reprodukcja
- Operatory genetyczne
- Sukcesja

Metody te powtarzane są w pętli do momentu, aż nie zostanie spełniony warunek stopu.
Często operacje **reprodukcji** i **sukcesji** są określane wspólnym terminem **selekcja**. Natomiast trzeba uściślić różnicę występującą pomiędzy tymi metodami:

- **Reprodukcja** określa sposób tworzenia populacji tymczasowej $\mathbf{T}^t$, która jest poddawana działaniu operacji genetycznych, dając populację potomną $\mathbf{O}^t$

- Natomiast **sukcesja** określa, w jaki sposób utworzyć nową populację bazową $\mathbf{P}^{t+1}$ (po zadziałaniu operacji genetycznych) z populacji potomnej $\mathbf{O}^t$ oraz starej populacji bazowej $\mathbf{P}^t$

Często nazywane są odpowiednio: **preselekcją** i **postselekcją**.

3.3.1 Różne kwestie warte uwagi

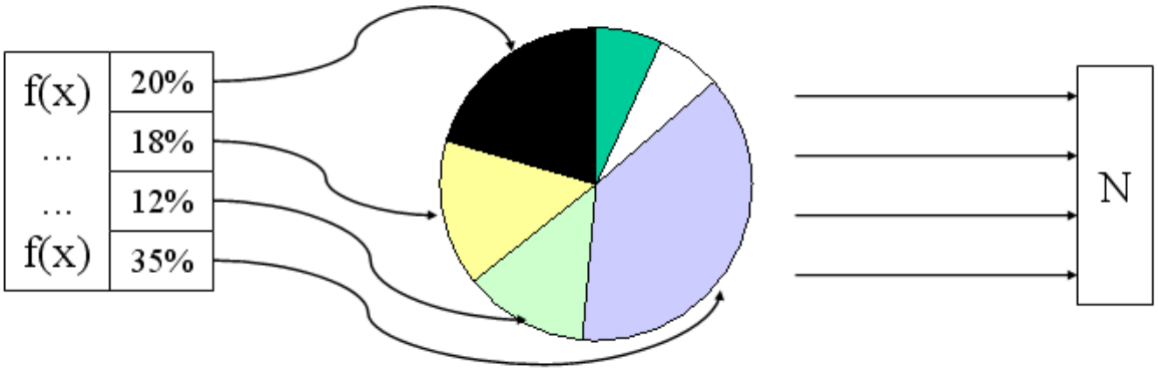
Algorytm, który posiada większą wartość oczekiwaną liczby kopii lepszego osobnika w porównaniu z wartością oczekiwaną liczby kopii gorszego osobnika cechuje się odpowiednio wiekszym **naciskiem selektywnym** (tendencja algorytmu do poprawiania wartości średniej przystosowania).

Warto również wspomnieć o **superosobnikach**, które mogą się pojawiać podczas trwania algorytmu. Są one:

- niepożądane w początkowej fazie działania (przedwczesna zbieżność)
- pozytywne pod koniec pracy algorytmu (zawężenie przestrzeni poszukiwań)

Twarda (brutalna) selekcja – jest to wybór do populacji potomnej i powielanie tylko najlepszego osobnika (metoda stochastycznego wzrostu).

3.3.2 Metody reprodukcji



Rysunek 8: Selekcja ruletkowa

Koło ruletki: Zamieniamy wartości policzonej wcześniej funkcji celu na wartości przystosowania. Następnie spośród N osobników populacji pośredniej losujemy N osobników populacji końcowej (z powtórzeniami). Kroki metody ruletkowej:

- Liczymy sumę wartości funkcji celu: $f_{sum} = f(x_1) + ... + f(x_N)$.
- Liczymy wkład każdego osobnika w sumę: $p(x_i) = \frac{f(x_i)}{f_{sum}}$.
- Traktujemy wartości $p(x_i)$ jako rozkład prawdopodobieństwa $\mathbf{P} = [p_1, p_2, ..., p_N]$ i dokonujemy N-krotnego losowania osobników zgodnie z tym rozkładem. Wylosowane w ten sposób osobniki dokładamy do populacji końcowej.
- Wybieramy osobniki na podstawie dystrybuanty obliczonej z uzyskanego prawdopodobieństwa (odpowiada to określonym polom na kole ruletki). Jest obliczan na podstawie wzoru:

$$F_x(x_k) = \sum_{j=1}^{k} p_x(x_j) \tag{3}$$

- Potem zostaje wylosowana liczba z rozkładu jednostajnego [0, 1] i porównana z zakresami utworzonymi przez dystrybuantę. Jeśli wartość wylosowana wypadnie pomiędzy F_{k-1} a F_k, to osobnik j przechodzi dalej (postępujemy tak aż do zapełnienia całej populacji).

Selekcja turniejowa:

- Wybór k osobników (rozmiar turnieju, zwykle k=2) i selekcja najlepszego z grupy
- Powtarzane pop_size razy

Selekcja rankingowa:

- Szeregowanie osobników według wartości przystosowania i selekcja zgodnie z kolejnością (wg tzw. **linii rangi**
- **zapobiega powstawaniu superosobników**
- **pomija informacje o względnych ocenach osobników**

3.3.3 Operatory genetyczne

Do operatorów ewolucyjnych należą operatory:

- jednoargumentowe: **mutacja** równomierna i brzegowa
- dwuargumentowe: **krzyżowanie** proste i arytmetyczne

MUTACJA RÓWNOMIERNA:

- Losowy wybór genu w osobniku
- Przyjęcie przez gen wartości losowej (z rozkładem równomiernym) z zakresu dopuszczalnego dla danej zmiennej:

$$Y = [X_1, ..., X'_k, ..., X_N] \tag{4}$$

$$X'_k =< left(k), right(k) > \tag{5}$$

- **Szczególnie użyteczna we wczesnej fazie działania AE (gdy pożądane jest szerokie przeszukiwanie obszaru poszukiwań optimum)**

MUTACJA NIERÓWNOMIERNA:

- Należy do grupy, tzw. mutacji ze strojeniem
- Modyfikacja wartości wybranego genu o wartość pewnej funkcji $\Delta(t, y)$

MUTACJA BRZEGOWA:

- Jest odmianą mutacji równomiernej, w której: $\begin{cases} X'_k = left(k) & \text{gdy wylosowano 0} \\ X'_k = right(k) & \text{gdy wylosowano 1} \end{cases}$
- **Szczególnie użyteczna, gdy rozwiązanie optymalne leży na brzegu obszaru dopuszczalnego lub bardzo blisko tego brzegu)**

KRZYŻOWANIE (PARA RODZICÓW-PARA POTOMKÓW): zwykle 2 osobniki rodzicielskie tworzą 2 (sprzężone) osobniki potomne

KRZYŻOWANIE (POJEDYNCZY OSOBNIK POTOMNY):

- wariant dwuosobniczy – para osobników rodzicielskich
- wariant globalny – jeden wiodący i n pomocniczych osobników rodzicielskich (po jednym dla każdego genu)

KRZYŻOWANIE WIELOOSOBNICZE:

- z wieloma osobnikami potomnymi
- z jednym osobnikiem potomnym

Rysunek 9: Wizualizacja krzyżowania jednopunktowego

OPERATORY KRZYŻOWANIA WYMIENIAJĄCEGO

- tworzą osobniki potomne przez składanie ich z wartości genów osobników rodzicielskich
- nie dochodzi do modyfikacji wartości genów zawartych w osobnikach krzyżowanych osobników rodzicielskich (tylko ich przetasowanie)

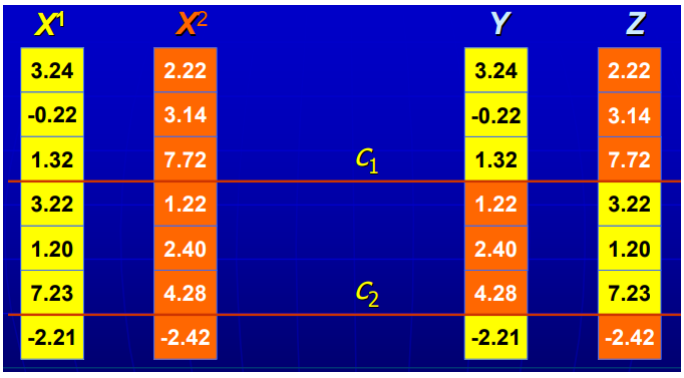
KRZYŻOWANIE JEDNOPUNKTOWE (proste):

- wybór (z rozkładem jednostajnym) liczby c (punkt rozcięcia) ze zbioru [1, 2, ..., N-1] (N – długość osobnika)
- podział chromosomów X^1 i X^2 poddawanych krzyżowaniu na dwie części i ich sklejanie:

$$Y = [X_1^1, ..., X_c^1, X_{c+1}^2, ..., X_N^2] \tag{6}$$

- W wersji z 2 osobnikami potomnymi drugi potomek:

$$Z = [X_1^2, ..., X_c^2, X_{c+1}^2, ..., X_N^1] \tag{7}$$



Rysunek 10: Wizualizacja krzyżowania dwupunktowego

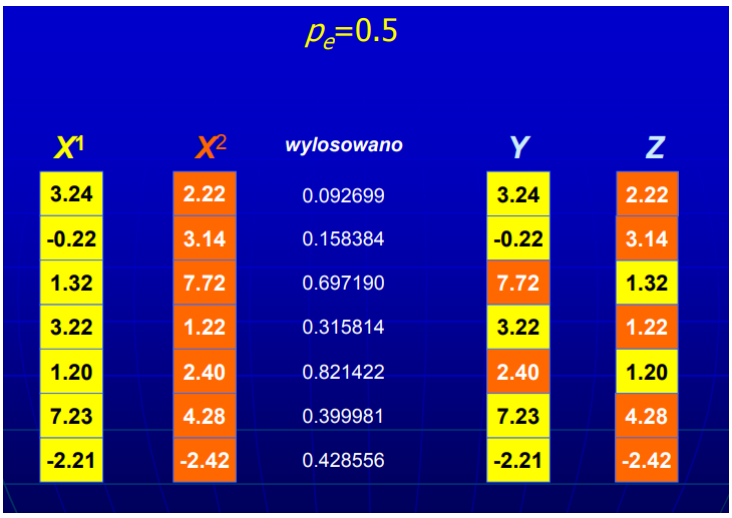
KRZYŻOWANIE DWUPUNKTOWE (proste):

- wybór 2 punktów rozcięcia c_1 i c_2
- podział chromosomów X^1 i X^2 poddawanych krzyżowaniu na trzy części i wymiana środkowej części:

$$Y = [X_1^1, ..., X_{c_1}^1, X_{c_1+1}^2, ..., X_{c_2}^2, X_{c_2+1}^1, ..., X_N^1] \tag{8}$$

- W wersji z 2 osobnikami potomnymi drugi potomek:

$$Z = [X_1^2, ..., X_{c_1}^2, X_{c_1+1}^1, ..., X_{c_2}^1, X_{c_2+1}^2, ..., X_N^2] \tag{9}$$



Rysunek 11: Wizualizacja krzyżowania równomiernego

KRZYŻOWANIE RÓWNOMIERNE:

- Osobnik potomny:
$$Y_i = \begin{cases} X_i^1 & \text{jeśli wylosowano liczbę} < p_c \\ X_i^2 & \text{w przeciwnym razie} \end{cases}$$
- W wersji z 2 osobnikami potomnymi drugi potomek:
$$Z_i = \begin{cases} X^2 & \text{jeśli } Y_i = X_i^1 \\ X^1 & \text{w przeciwnym razie} \end{cases}$$
- Wartość parametru krzyżowania typowo ustawia się na $p_e = 0.5$.

KRZYŻOWANIE DIAGONALNE:

- Jest krzyżowaniem wieloosobniczym
- Tworzy r potomków z r rodziców przy c=r-1 punktach krzyżowania
- Osobniki potomne powstają w wyniku składania fragmentów kodu po przekątnej

OPERATORY KRZYŻOWANIA UŚREDNIAJĄCEGO:

- Są specyficzne dla kodowania rzeczywistoliczbowego
- Oddziałują na wartości genów osobników poddawanych krzyżowaniu
- Wartości każdego genu osobników potomnych są liczbami zawierającymi się między największą i najmniejszą wartością genu osobników rodzicielskich

KRZYŻOWANIE ARYTMETYCZNE:

- generowanie liczby losowej k z zakresu (0, 1) lub jej arbitralny wybór
- uśrednianie arytmetyczne wartości genów osobników rodzicielskich:
$$Y = X^1 + k(X^2 - X^1) \tag{10}$$
- w wersji z 2 osobnikami:
$$Z = X^2 + X^1 - Y^1 \tag{11}$$

3.3.4 Metody sukcesji

Sukcesja trywialna (z całkowitym zastępowaniem): Nową populacją bazową staje się populacja potomna:

$$P(t+1) = O(t) \tag{12}$$

- **Najbardziej odporna na przedwczesną zbieżność**
- **Najwolniej prowadzi do rozwiązania optymalnego**
- **Może prowadzić do sytuacji, w któej nie zawsze najlepsze rozwiązania z populacji P(t) znajdą się w populacji P(t+1)**

Sukcesja z częściowym zastępowaniem: W nowej populacji bazowej są osobniki z populacji potomnej i ze starej populacji bazowej:

$$P(t+1) = O(t) + P(t) \tag{13}$$

- **Prowadzi zwykle do stabilniejszej pracy AE**
- **Może spowodować tendencję do osiągania maksimów lokalnych**

Warianty mechanizmów usuwania:

- usuwanie najgorzej przystosowanych osobników
- usuwanie osobników podobnych do potomnych
- usuwanie losowo wybranych osobników

Sukcesja elitarna Gwarantuje przeżycie co najmniej najlepszego osobnika poprzez odpowiedni wybór osobników z P(t) do P(t+1)

- **Wzrost wielkości elity powoduje przyspieszenie zbieżności algorytmu**
- **Wzrost wielkości elity powoduje większe prawdopodobieństwo osiągania ekstremów lokalnych**

Wartość wielkości elity δ decyduje o naporze selekcyjnym (δ=0 – sukcesja trywialna). **Najkorzystniej: jeden, ew. kilka osobników.**

3.4 Wynik działania algorytmu ewolucyjnego

Często do zbadania zachowania danego algorytmu optymalizacyjnego stosuje się krzywe zbieżności, będące wykresami zmian rozwiązania roboczego w zależności od czasu.
Kiedy mamy do czynienia z algorytmem ewolucyjnym w każdym pokoleniu zmiany odnoszą się do całej populacji chromosomów (osobników), co posuwa nas do decyzji, czy chcemy wizualizować tę zależność dla średniej, maksymalnej lub innego sposobu estymacji tendencji zmian rozwiązania roboczego.
Wyróżniona jest również pewna krzywa – zmian w kolejnych pokoleniach wartości przystosowania najlepszego osobnika znalezionego począwszy od generacji początkowej. **Po zakończeniu działania algorytmu osobnik ten jest jest rozwiązaniem wyznaczonym przez pojedyncze jego uruchomienie**.

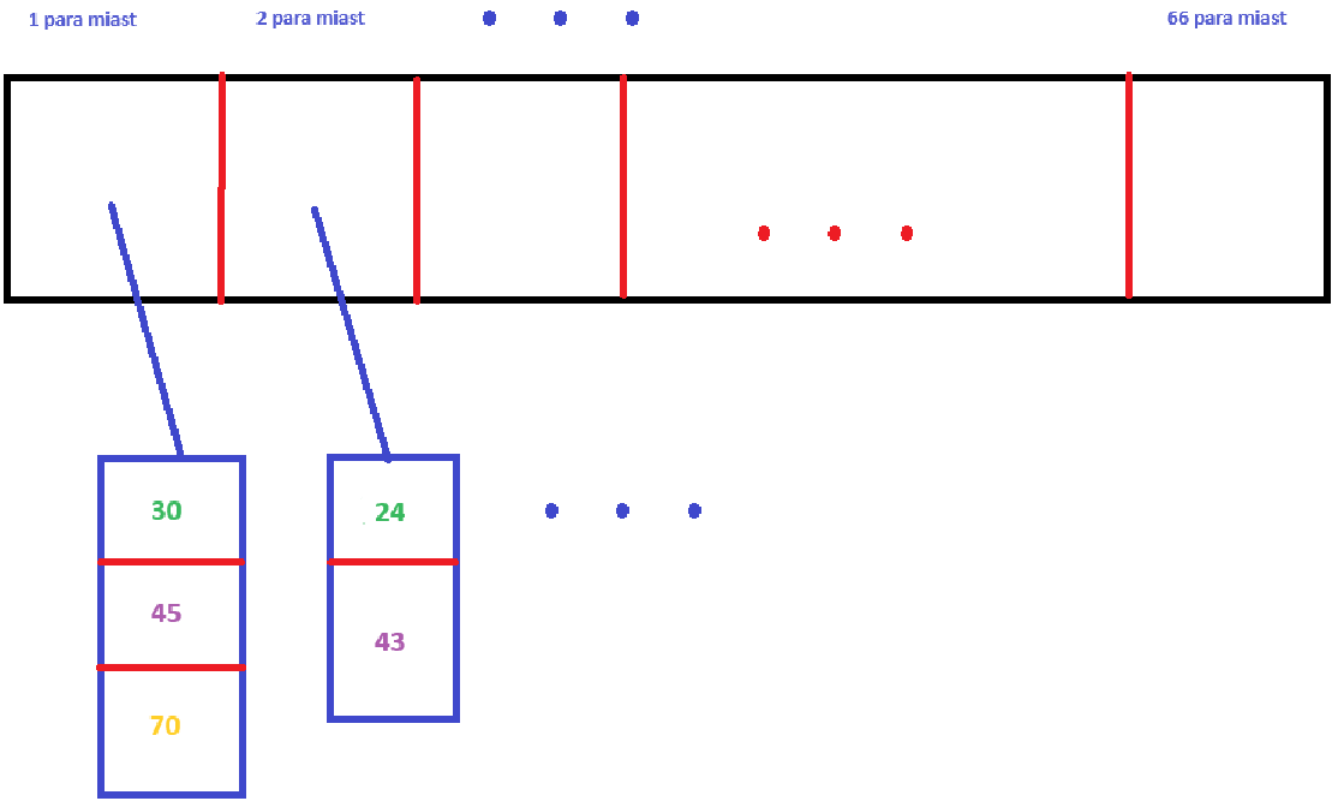
3.5 Struktura chromosomu i całej populacji

Populacja będzie się składać z K (liczba osobników populacji) 66-elementowych (66 genów) osobników (chromosomów), reprezentujących rozmieszczenie ścieżek wśród każdej pary miast. Każdy z osobników będzie posiadał strukturę informującą o tym, jakie ścieżki będą wchodzić w skład ich połączenia (w przypadku wariantu **pełnej agregacji** będzie możliwa tylko jedna ścieżka, a z kolei w wariancie **dezagregacyjnym** tych ścieżek może być więcej i ich ilości mogą się różnić między poszczególnymi parami miast). Będzie więc to macierz wyglądająca w ten sposób:

3.6 Możliwe parametry do optymalizowania

W algorytmie ewolucyjnym mamy do czynienia z mnogością różnych hiperparametróww, które możemy optymalizoować i dobierać tak, by maksymalizować lub minimalizować funkcję przystosowania wedle naszych potrzeb:

- wszelkie parametry prawdopodobieństw:
  - operatorów genetycznych
  - selekcji
  - liczność populacji
  - liczba pokoleń



Rysunek 12: Przykładowy chromosom w przypadku dezagregacji (w przypadku pełnej agregacji każda para miasta miałaby przypisaną jedną ścieżkę)

4 Technologia

Algorytm ewolucyjny, eksperymenty i wizualizacja będą zaimplementowane w języku *Python*, korzystając z bibliotek *matplotlib*, *numpy*, *collections*, *itertools*, *xml* i innych.

5 Wczytanie danych

Dane zostały załadowane ze strony (ww.) sndlib za pomocą modułu *xml.etree.ElementTree* do czterech słowników:

- ***Nodes*** - przypisujący współrzędne do poszczególnych miast
- ***Links*** - przypisujący miasta źródłowe i docelowe dla danej istniejącej ścieżki w grafie miast
- ***Demands_paths*** - przypisujący możliwe (dostępne), predefiniowane ścieżki dla wszystkich połączeń (*Links*), które docelowo mają spełniać konkretne zapotrzebowania
- ***Demands_values*** - przyporządkowane wartości zapotrzebowań dla konkretnych połączeń (*Links*)

6 Plan eksperymentów

6.1 Inicjalizacja populacji początkowej

Po wczytaniu wszystkich słowników opisujących przestrzeń przeszukiwań algorytmu została zaprojektowana funkcja losująca populację początkową, używając modułu *random* i funkcji takich jak *sample* czy *randint*. W przypadku:

- pełnej agregacji - algorytm miał możliwość przypisania tylko jednej ściezki dla danej pary miast
- dezagregacji - algorytm miał możliwość przypisania wielu ścieżek (było to losowane) i musiał (zgodnie z rys. 5) rozłożyć wartość konkretnego zapotrzebowania (dla danego połączenia, czyli pary miast); zostało to zaimplementowane poprzez użycie **rozkładu wielomianowego** - funkcji *multinomial* z modułu *random*, który rozdziela liczbę całkowitą N (zbieżność oznaczeń z liczbą określająca wielkość osobnika) na p (liczba ścieżek) liczb całkowitych sumujących się do N

W przypadku n niezależnych prób, z których każda prowadzi do sukcesu dokładnie w jednej z k kategorii, przy czym każda kategoria ma określone prawdopodobieństwo sukcesu, rozkład wielomianowy podaje prawdopodobieństwo dowolnej szczególnej kombinacji liczb sukcesów dla różnych kategorii (np. modelowanie prawdopodobieństwa zliczeń dla każdej strony kości o boku k rzuconej n razy). Innymi słowy rozkłada tak zadaną liczbę całkowitą N, jakbyśmy tego w tym przypadku oczekiwali (Grygiel):

$$p_{k_1 k_2,\ldots,k_N} = \frac{N!}{k_1!k_2!\ldots k_N!} p_1^{k_1} p_2^{k_2} \ldots p_N^{k_N} \tag{14}$$

gdzie:

$$k_1 + k_2 + \ldots + k_n = N \tag{15}$$

6.2 Przedmioty analiz

Będzie sprawdzany/e:

- dla kilku wartości modularności: małej, niewielkiej i dużej, np. (1, 5, 20)
- wpływ liczebności populacji

- wpływ liczebności iteracji algorytmu, współczynnika mutacji i prawdopodobieństw krzyżowania na jego działanie
- wykorzystanie różnych algorytmów selekcji (reprodukcji i sukcesji), mutacji i krzyżowania (o tym dalej)
- wpływ dezagregacji na działanie algorytmu (sprawdzenie dwóch wariantów - również dla różnych modularności)

Należałoby także przeprowadizć wiele niezależnych uruchomień (L-launches) dla losowej próby różnych populacji bazowych $\mathbf{P(0)}$. W przypadku wielu uruchomień dla tej samej populacji $\mathbf{P(0)}$ można mówić o właściwościach danego algorytmu dla konkretnej populacji początkowej.
Do implementacji faktycznego algorytmu użyte będą dwa warianty algorytmu:

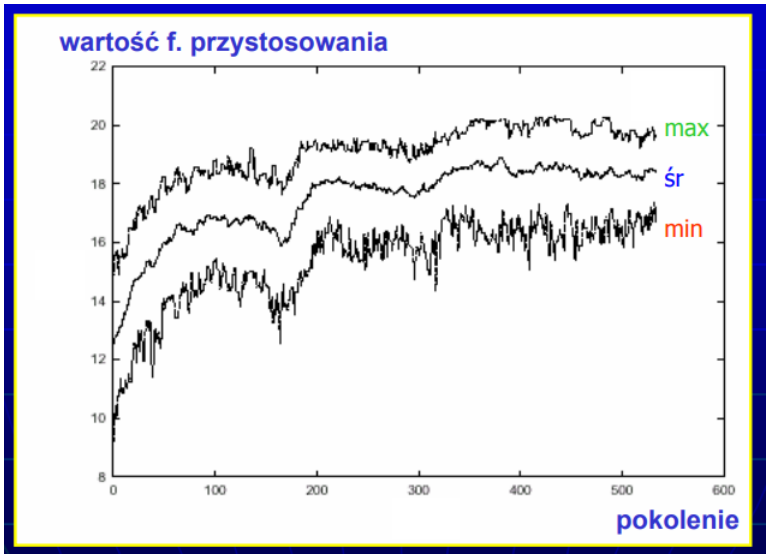
- (1) **z preselekcją ruletkową, mutacją równomierną, krzyżowaniem dwupunktowym (w wersji z jednym osobnikime potomnym) i sukcesją z częściowym zastępowaniem (z wariantem usuwania najgorzej przystosowanych osobników)**
- (2) **z preselekcją turniejową, mutacją równomierną, krzyżowaniem równomiernym (w wersji z jednym osobnikiem potomnym) i sukcesją elitarną (nadmiaru osobników pozbywamy się, odrzucając najgorsze osobniki z populacji potomnej)**

Dla każdej losowej populacji początkowej uruchamia się dwa porównywane algorytmy.

6.3 Analiza statystyczna

- Analiza wykresów zbieżności, map i histogramów
- Uwzględnienie informacji o minimalnej i maksymalnej osiąganej wartości
- Prezentacja histogramu dla wartości f. przystosowania (pozwala ocenić rozkład)

6.3.1 Krzywe zbieżności

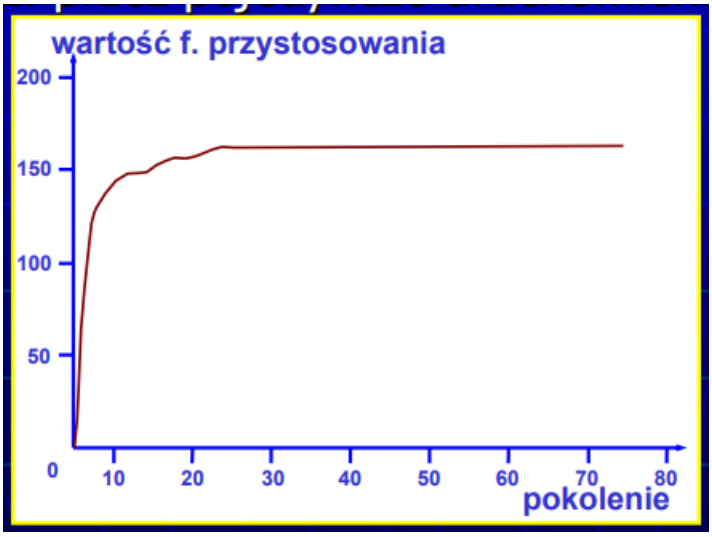


Rysunek 13: Przykładowa krzywa zbieżności, źródło: Beluch

Tak jak wcześniej zostało to wspomnianie krzywe zbieżności stanowią ważny element analizy działania gotowego algorytmu:

- są wykresem zmian wartości rozwiązania roboczego (średniego, najlepszego, najgorszego) w czasie
- kreśli się je:
  - dla pojedynczego uruchomienia algorytmu (tego wariantu użyto)
  - dla wielu niezależnych uruchomień (uśrednione - bardziej reprezentatywne, ale gubi niektóre informacje o zachowaniu AE w pojedynczych uruchomieniach)

Istnieje również szczególna krzywa zbieżności: wykres zmian w kolejnnych pokoleniach wartości przystosowania najlepszego osobnika znalezionego od początku działania AE – jest to rozwiązanie wyznaczone przez pojedyncze uruchomienie AE.



Rysunek 14: Szczególna krzywa zbieżności osobnika najlepszego, źródło: Beluch

Kryterium stopu będzie hiperparametrem, okreslającym ilość iteracji algorytmu (pokoleń).

7 Struktura projektu

Program implementujący algorytm ewolucyjny został podzielony na 5 modułów:

- *EvolutionAlgorithm.py* - zawiera główny moduł algorytmu
- *ParseXML.py* - odpowiada za wczytanie danych z bazy *sndlib*
- *visualization.py* - funkcje wizualizujące (wykresy, mapy, histogramy)
- *utils.py* - funkcje pomocnicze i zapisywania danych symulacji do pliku
- *tests.ipynb* - notatnik *Jupyter Notebook* zawierający analizę statystyczną uzyskanych wyników

8 Testy

8.1 Wykresy zbieżności i mapy rozwiązań

Dla kilku przykładowych zbiorów wartości parametrów wywoływano algorytm i obserwowano, jak zmieniają sie zależności:

- średniej funkcji przystosowania
- minimalnej funkcji przystosowania
- maksymalne funkcji przystosowania

w funkcji numeru iteracji (pokolenia).

8.2 Histogramy

Dla przykładowych zbiorów wartości i wybranych numerów pokoleń ilustrowano rozkład wartości funkcji przystosowania w danej populacji:

8.3 Przeszukiwanie hipersiatki

Przeszukiwanie siatki hiperparametrów posłużyło do analizy ich wzajemnych, często dość skomplikowanych zależności, których nie dałoby się uzyskać poprzez analizę samych wykresów zbieżności średniej, minimalnej i maksymalnej funkcji przystosowania, jak również ich histogramów dla danego pokolenia czy map utworzonych z osobnika wyjściowego algorytmu. Postępowano w następujący sposób:

1. Tworzono listy wartości danych hiperparametrów
2. Konstruowano produkt kartezjański wartości z tych list przy pomocy pakietu *itertools* i funkcji *product*
3. Następnie iterowano po tak utworzonym produkcie kartezjańskim i dla zadanych wartości uruchamiano algorytm
4. Zbierano wartości funkcji przystosowania osobników końcowych populacji, aby porównać w jakiś sposób działania algorytmów i korelacje pomiędzy nimi

W przypadku algorytmu z wariantem były to (1):

- współczynnik mutacji

- współczynnik krzyżowania
- liczba iteracji (pokoleń)
- ilość populacji
- modularność
- flaga dezagregacji

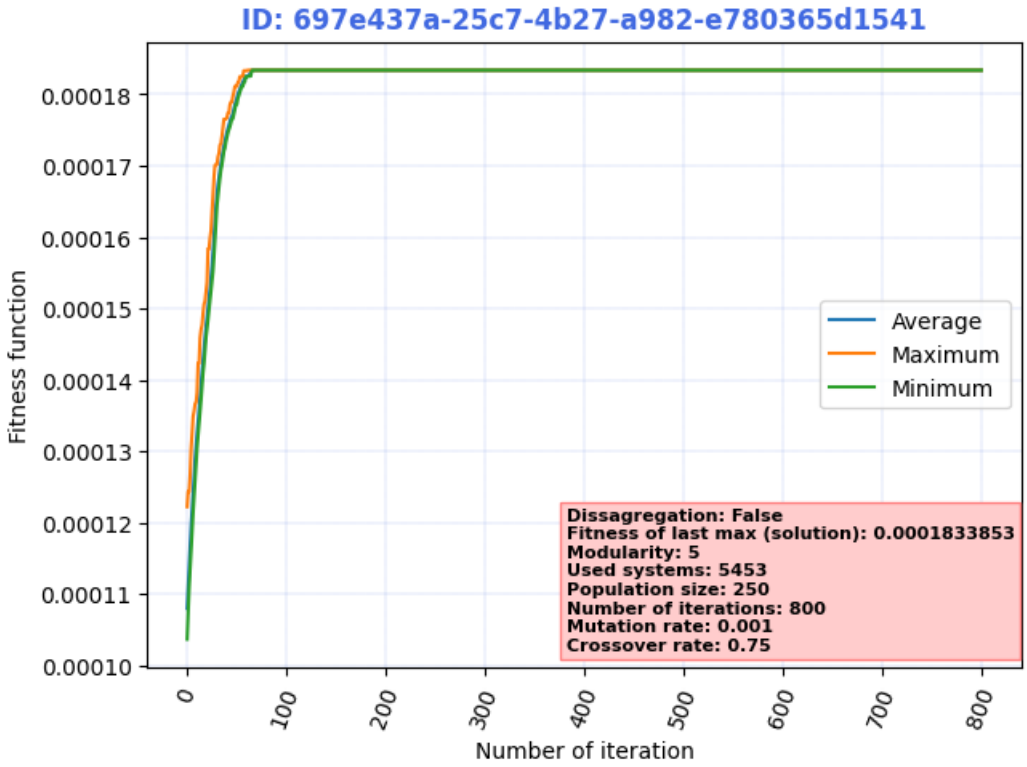
W przypadku algorytmu z wariantem były to (2):

- współczynnik mutacji
- współczynnik krzyżowania
- liczba iteracji (pokoleń)
- wielkość elity
- wielkość turnieju
- ilość populacji
- modularność
- flaga dezagregacji

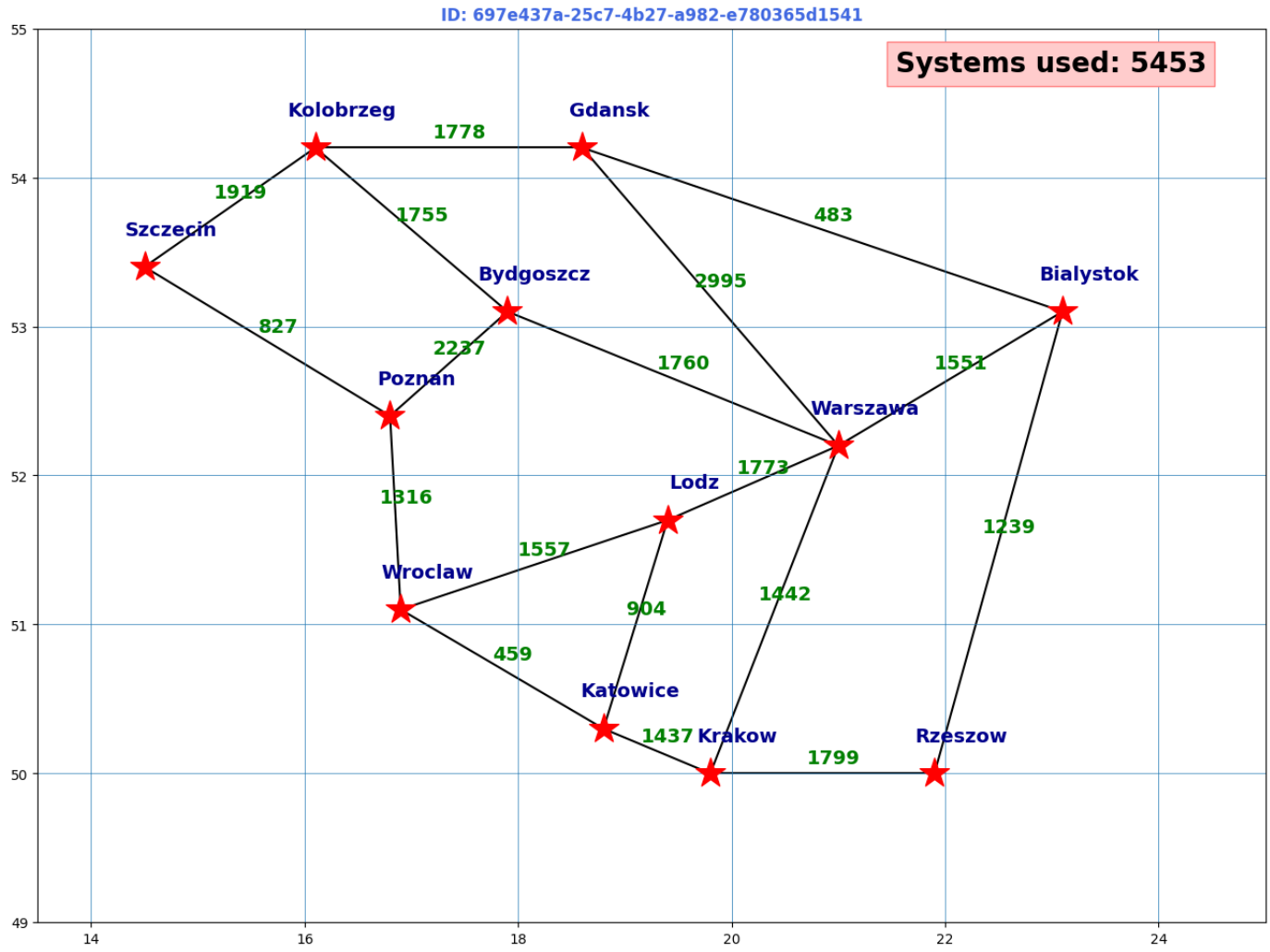
Testy zostały przeprowadzone w środowisku *Jupyter Notebook tests.ipynb*.

9 Uzyskane wyniki

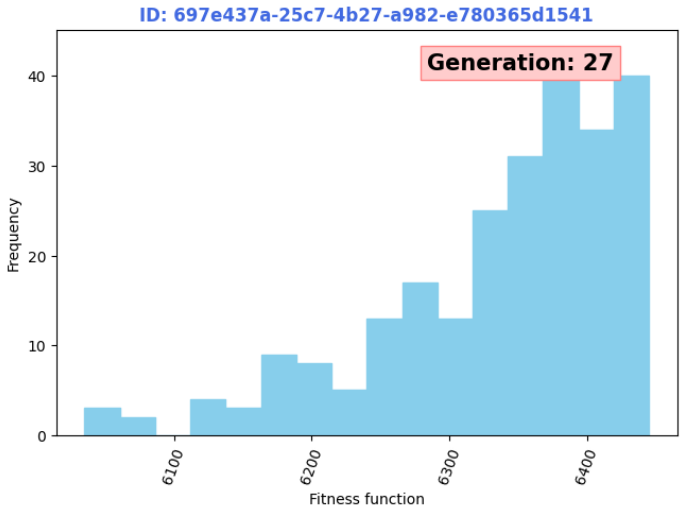
9.1 Wykresy zbieżności, mapy rozwiązań i histogramy



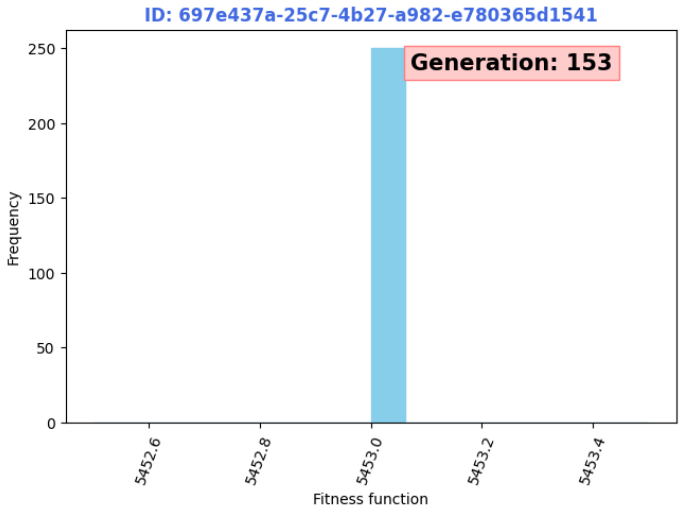
Rysunek 15: Wykres zbieżności AE



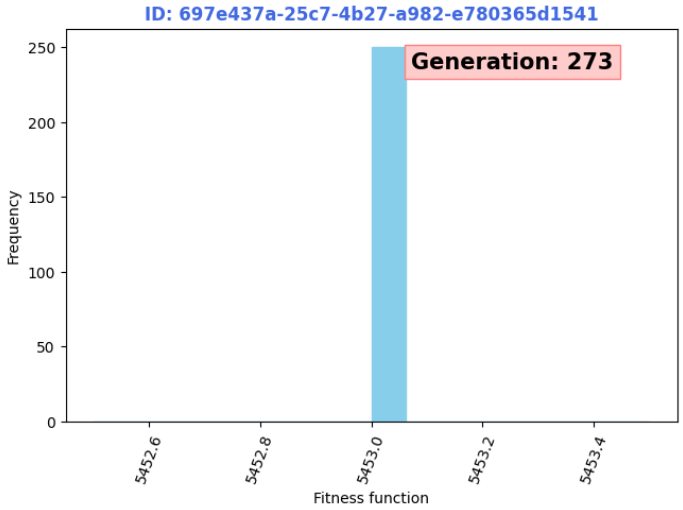
Rysunek 16: Mapa rozwiązania



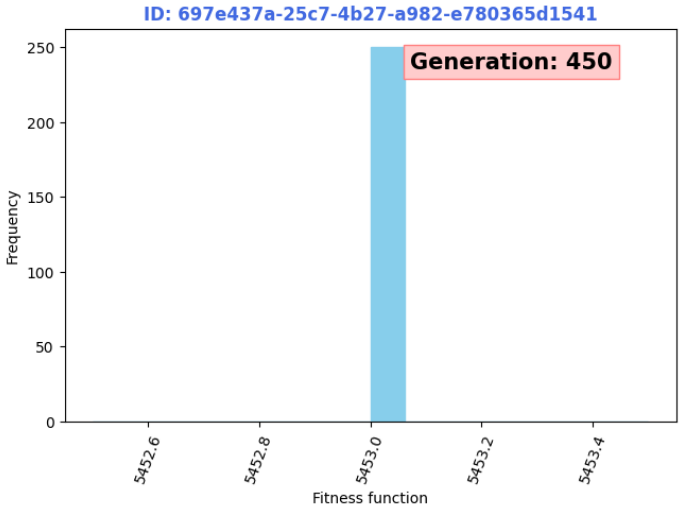
(a) Rozkład wartości funkcji przystosowania dla pokolenia 27



(b) Rozkład wartości funkcji przystosowania dla pokolenia 153

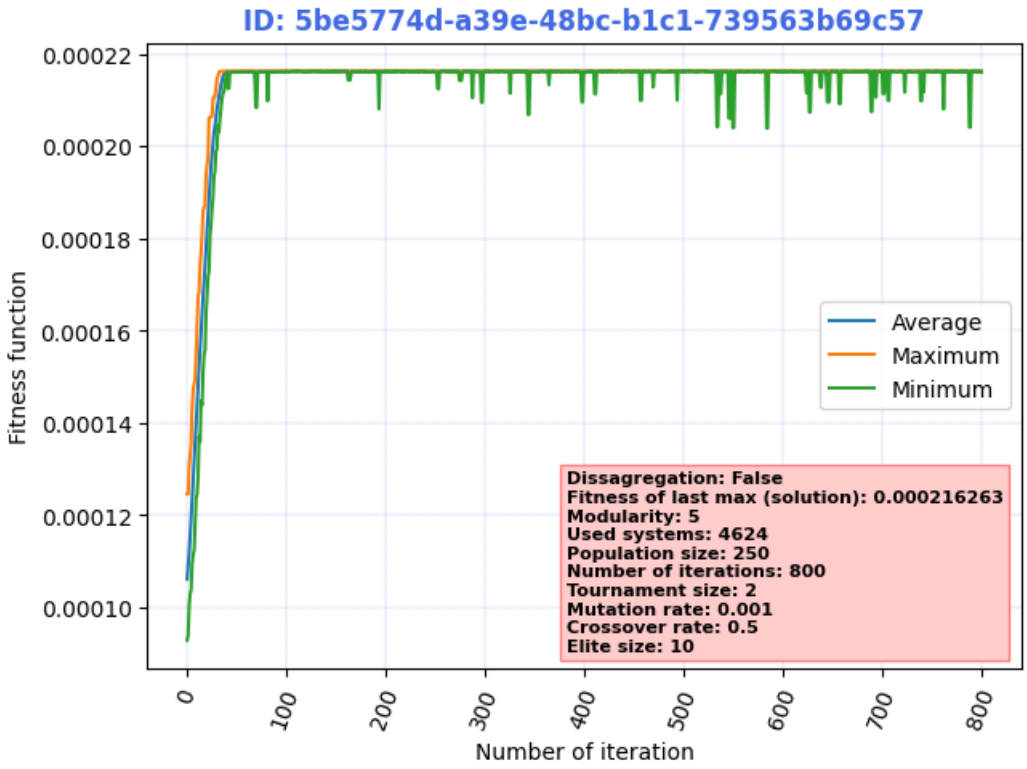


(c) Rozkład wartości funkcji przystosowania dla pokolenia 273

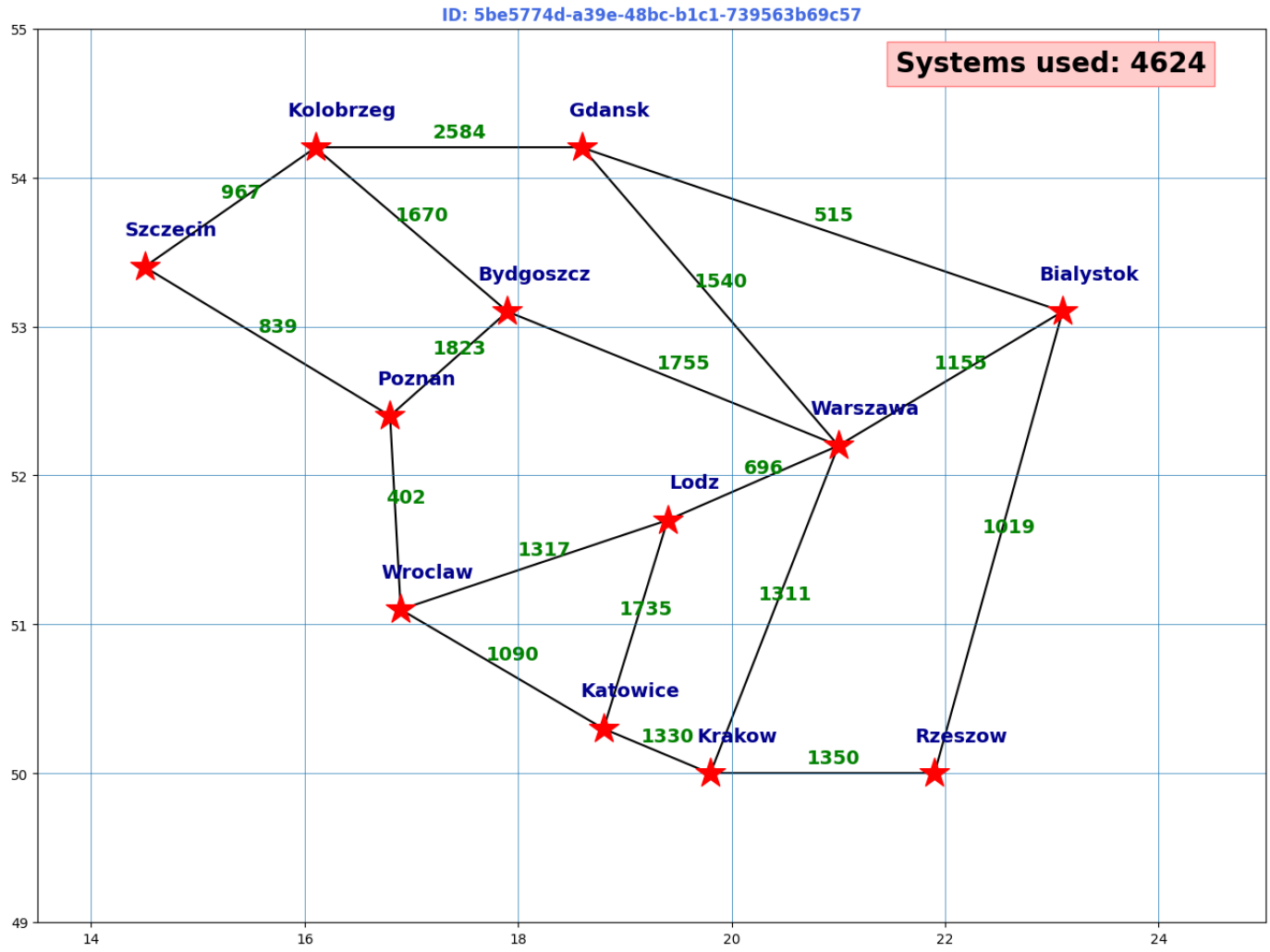


(d) Rozkład wartości funkcji przystosowania dla pokolenia 450

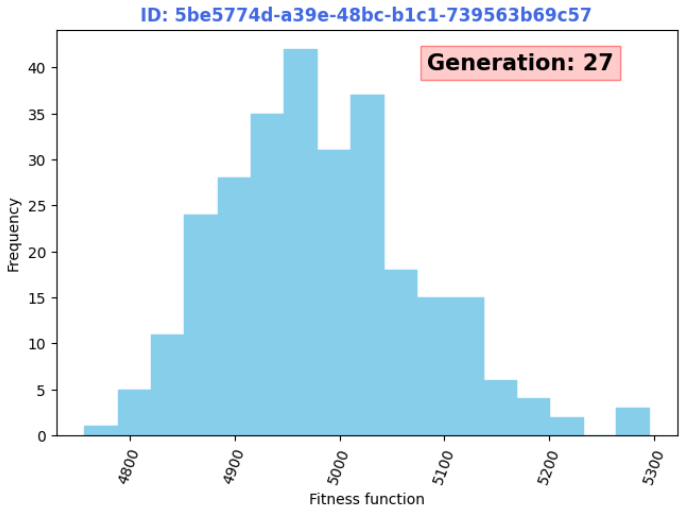
Rysunek 17: Histogramy



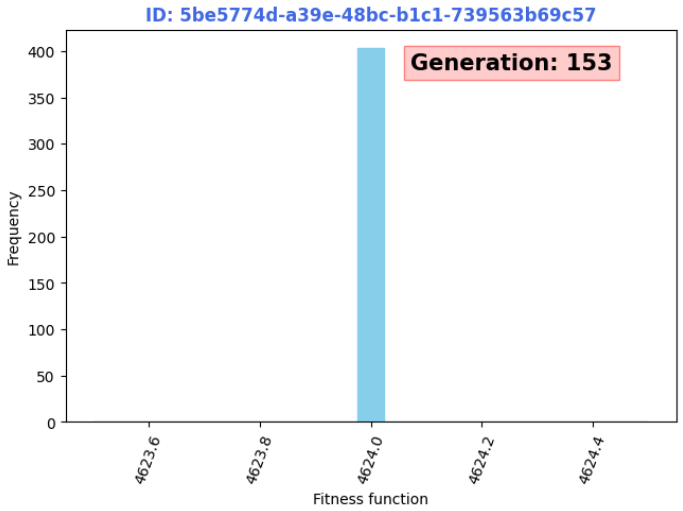
Rysunek 18: Wykres zbieżności AE



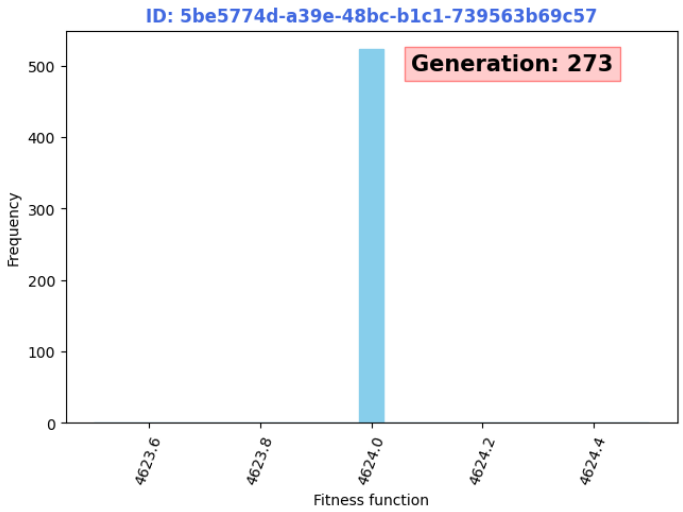
Rysunek 19: Mapa rozwiązania



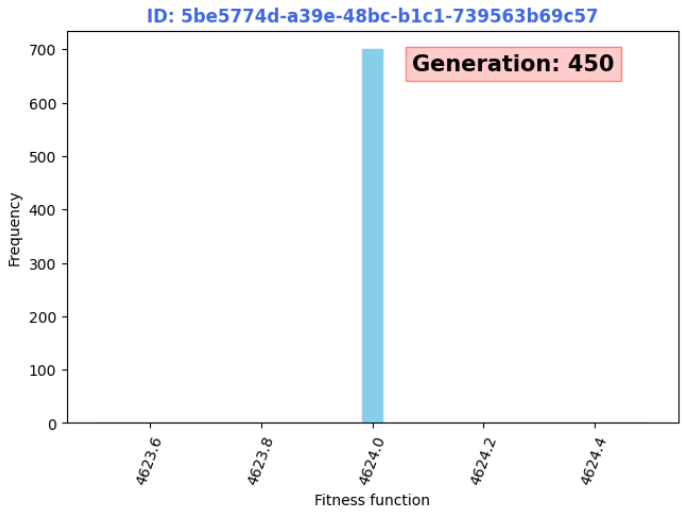
(a) Rozkład wartości funkcji przystosowania dla pokolenia 27



(b) Rozkład wartości funkcji przystosowania dla pokolenia 153



(c) Rozkład wartości funkcji przystosowania dla pokolenia 273



(d) Rozkład wartości funkcji przystosowania dla pokolenia 450

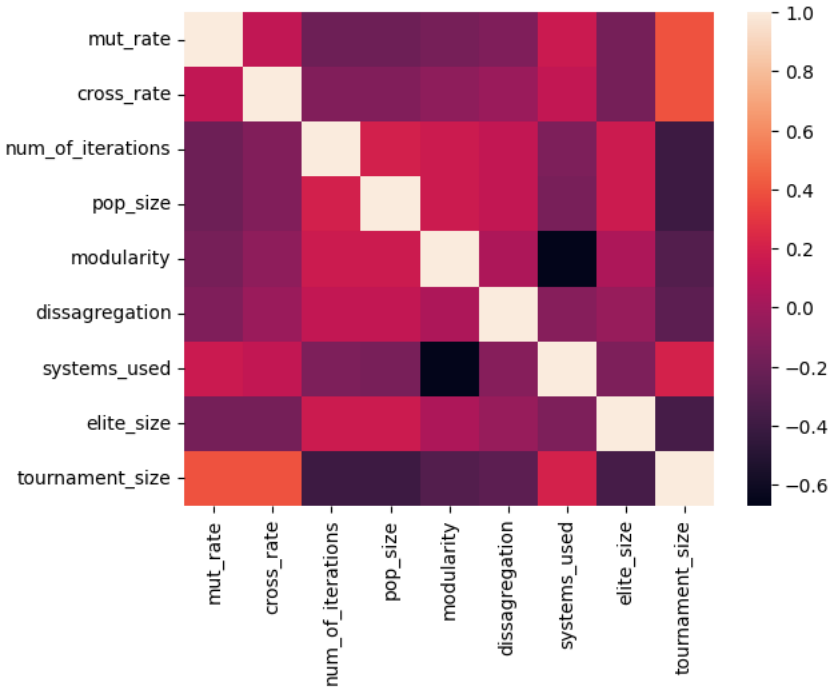
Rysunek 20: Histogramy

9.2 Przesukiwanie hipersiatki

9.2.1 Tabele wyników

Tabele umieszczone w notatniku *tests.ipynb* wraz z wykresem wizualizującym korelację pomiędzy poszczególnymi parametrami.

9.2.2 Macierz korelacji



Rysunek 21: Macierz korelacji dla wszystkich połączonych wyników

10 Wnioski i perspektywy na przyszłość

10.0.1 Analiza wyników

- Na wykresie zbieżności 15 dla algorytmu (1) można zauważyć, że po pewnym ogonie stanowiącym początek działania algorytmu rozwiązania stabilizowały się na pewnym poziomie, natomiast w przypadku algorytmu (2) 18 wartość minimum fluktuowała, co można przypisać działaniu przechodzenia elity z pokolenia na pokolenia (która w pewnym momencie przestaje już być korzystna dla algorytmu – stąd gdzieniegdzie pojawiające się spore piki)

- Jeżeli chodzi o histogramy - 17 i 20 zauważa się przewidywane zachowanie: na początku rozkład wartości dla psozczególnych osobników jest dość różnorodny – wraz z postępem algorytmu rozkład ten znacznie się zawęża (w celu dokładniejszej analizy można by zwiększyć ilość binów)

- Z macierzy korelacji 21 wyczytać można różnorakie zależności pomiędzy zastosowanymi hiperparametrami – możemy wnioskować, że największy wpływ na uzyskaną liczbę systemów teleinformatycznych dla sieci miały następujące parametry:

  – zwiększenie liczby iteracji (pokoleń), liczebności populacji i wielkość elity (modularność nie brana pod uwagę, ponieważ zmienia ona explicite ilość użytych systemów) pozytywnie wpływa na zmniejszenie liczby użytych systemów teleinformatycznych

  – zwiększenie współczynnika mutacji, krzyżowania i wielkości turnieju powoduje zwiększenie liczby użytych systemów (niepożądana sytuacja)

- Ciekawym wnioskiem z tablicy wyników (w notatniku *tests.ipynb*), o którym warto by było wspomnieć jest to, że dezagregacja **negatywnie wpływała na minimalizację liczby użytych systemów teleinformatycznych**, więc można postawić hipotezę, że **rozkładanie zapotrzebowań na jednej ścieżce (pełna agregacja)** jest **najkorzystniejsze**; oczywiście trzeba by było przeprowadzić znacznie większą ilość pomiarów, aby móc z większym przekonaniem mówić o jej potencjalnej prawdziwości

10.0.2 Perspektywy

Algorytm jest w początkowej fazie rozwoju i w przyszłości możliwe są różne modyfikacje, ulepszenia i rozszerzenia, a także przeznaczenie większej ilości zasobów sprzętowych:

- Można rozszerzać hipersiatkę przeszukiwań, co wiązać się będzie niestety z bardzo dużym wzrostem złożoności obliczeniowej (ze względu na to – póki co – ograniczono się tylko do kilku wartości w każdej z list opisujących wartości poszczególnych hiperparametrów)
- Rozważenie użycia pakietu *Numba* do przyspieszenia obliczeń – jest to kompilator JIT typu open source, który tłumaczy podzbiór *Pythona* i *NumPy* na szybki kod maszynowy przy użyciu LLVM za pośrednictwem pakietu Python *llvmlite*. Oferuje szereg opcji zrównoleglania kodu Pythona dla procesorów i procesorów graficznych, często z niewielkimi zmianami w kodzie
- Implementacja innych rodzajów operatorów genetycznych – np. krzyżowania arytmetycznego, mutacji brzeegowej itd. i sprawdzanie ich działania na wydajność algorytmu

Literatura

Witold dr hab. inż. Beluch. imio.polsl.pl. http://www.imio.polsl.pl/Dopobrania/AE2.pdf. [Accessed 07-01-2024].

Kazimierz dr Grygiel. Wstęp do obliczeń ewolucyjnych i neuronowych. https://www.mimuw.edu.pl/~grygiel/woen/woen6.pdf. [Accessed 05-01-2024].

Arkadiusz mgr inż. Poteralski. *Optymalizacja strukturalna przestrzennych układów mechanicznych z wykorzystaniem algorytmów ewolucyjnych*. PhD thesis, Politechnika Śląska, Wydział Mechaniczny Technologiczny, 2004.