

POP – Dokumentacja Końcowa

Kacper Siemionek, Michał Pędziwiatr

Opis projektu

Celem projektu jest optymalizacja wykorzystania zasobów w sieci teleinformatycznej poprzez minimalizację liczby systemów teletransmisyjnych niezbędnych do obsłużenia zadanych zapotrzebowaan. Problem został rozwiązyany przy użyciu algorytmu ewolucyjnego. Sieć opisana jest za pomocą grafu:

$$G = (N, E)$$

N – zbiór węzłów (miasta), E – zbiór krawędzi (połączenia między miastami)

Kluczowym aspektem projektu jest zbadanie wpływu modularności systemu oraz agregacji/dezagregacji na koszt sieci. Funkcja kosztu wyraża się następującym wzorem:

$$f_e(o) = \lceil \frac{o_e}{m} \rceil$$

o_e – obciążenie krawędzi e , m – modularność ($m = 1, > 1, \gg 1$)

Pełen opis funkcjonalny

Aplikacja została zrealizowana jako narzędzie konsolowe (CLI), umożliwiające przeprowadzenie symulacji optymalizacyjnych z różnymi parametrami wejściowymi.

Główne funkcjonalności:

- **Wczytywanie danych:** Obsługa plików w formacie SNDlib, wczytująca listy węzłów, łączy, zapotrzebowan i ścieżek dopuszczalnych.

- **Eksport danych:** Automatyczny zapis danych po symulacji (najlepsze chromosomy, historia zbieżności itd.) do plików JSON, co umożliwia dalszą obróbkę i dokładniejszą analizę.
- **Wizualizacja:** Dedykowane moduły pomocnicze służące do generowania graficznej reprezentacji wyników:
 - Generator wykresów zbieżności,
 - Generator wykresu porównania kosztów,
 - Generator wykresu czasu wykonania algorytmu,
 - Generator map obciążenia sieci, wizualizujący rozkład ruchu.
- **Konfigurowalność algorytmu:** Użytkownik ma możliwość sterowania parametrami algorytmu ewolucyjnego poprzez argumenty wywołania:
 - “**--gens**”: Liczba generacji (domyślnie 100)
 - “**--pop**”: Rozmiar populacji dla każdego pokolenia: (domyślnie 300).
 - “**--modularities**”: Wybrane wartości modularności dla których chcemy przeprowadzić symulację (domyślnie: 1, 10, 100, 1000, 10000)
 - “**--mutation_rate**”: Prawdopodobieństwo mutacji osobnika (domyślnie 0,5).
 - “**--sigma**”: Parametr określający początkową wartość siły mutacji. (domyślnie 0,2).
 - “**--mode**”: Wybór trybu przydziału ruchu, dostępne opcje:
 - “agg” - Agregacja.
 - “deagg” - Dezagregacja.
 - “all” - Oba wykonane sekwencyjnie (opcja domyślna).
 - “**--repeats**”: Liczba powtórzeń symulacji - by zwiększyć rzetelność osiąganych wyników oraz zminimalizować losowość podczas analizy, każda z symulacji powinna być przeprowadzona kilkukrotnie (domyślnie 10 razy).
 - “**--heuristic_ratio**”: Parametr “zasiewania populacji”, określający jak duża część początkowej populacji będzie zmutowaną wersją deterministycznie wygenerowanego rozwiązania. Ostatecznie zdecydowaliśmy się potraktować mechanikę tę jako opcjonalną (domyślnie 0).
 - “**--tournament_size**”: Parametr określający rozmiar każdego z turniejów podczas selekcji osobników (domyślnie 8).

- “**--alpha**”: Współczynnik wagi rodziców, określa w jakim stopniu cechy pierwszego rodzica przekazywane są potomkowi względem drugiego (domyślnie 0,5).
- “**--seed**”: Ziarno losowości, pozwalające na odtwarzalność symulacji. By zapewnić różnorodność pomiędzy powtórzeniami symulacji, jednocześnie zachowując reprodukowalność, parametr ten jest zwiększany o 1 co każdą iterację. (domyślnie losowe).
- “**--no_heuristic**”: Wyłączenieinicjalizacji heurystycznej (start z losowej populacji, bez dodawania wygenerowanego deterministycznie osobnika). Ze względu na to, że owa częśćinicjalizacji jest kluczowa dla założeń naszego projektu, należy traktować tę funkcję jedynie jako narzędzie do testów algorytmu.
- “**--no_elitism**”: Wyłączenie mechaniki elityzmu.

Opis algorytmów

Ze względu na złożoność obliczeniową wyznaczenia tras w czasie rzeczywistym, algorytm korzysta ze statycznego zbioru ścieżek dopuszczalnych. Dla każdego zapotrzebowania wybrane jest rozwiązanie spośród K zdefiniowanych wstępnie tras, co eliminuje ryzyko wyboru ścieżki niepoprawnej lub zapętlonej.

Reprezentacja osobników

Chromosom: macierz C o wymiarach $D \times K$:

- D - liczba zapotrzebowień w sieci
- K - liczba ścieżek kandydujących dla danego zapotrzebowania.

Gen w_{ij} : określa priorytet (wagę) przydziału ruchu dla ścieżki j w ramach zapotrzebowania i .

Sposób interpretacji tych wag zależy od scenariusza:

- **Agregacja** (ruch jest niepodzielny). Dla każdego zapotrzebowania i algorytm wybiera ścieżkę o maksymalnej wadze w danym wierszu macierzy.
- **Dezagregacja** (podział jest dopuszczalny). Wagi w_{ij} traktowane są jako współczynniki proporcji za pomocą których wyznacza się ułamek ruchu kierowany daną ścieżką.

Iinicjalizacja populacji

Aby przyśpieszyć zbieżność algorytmu, jednocześnie nie zmniejszając drastycznie przestrzeni poszukiwań, do generowanych losowo rozwiązań wprowadzany jest jeden osobnik wygenerowany deterministycznie. Dla każdego z zapotrzebowień przypisywana jest maksymalna waga tej ścieżce ze zbioru K , która posiada najmniejszą długość (liczbę węzłów pośrednich).

Operatory genetyczne

- **Selekcja:** turniejowa
- **Mutacja:** gaussowska - do genu w_{ij} dodawana jest wartość losowana według rozkładu normalnego. Wartość jest przycinana do przedziału [0, 1], co zapobiega ujemnym wartościom oraz zwiększa stabilność algorytmu.
- **Krzyżowanie:** arytmetyczne - wagi potomka są liniową kombinacją wag rodziców, kontrolowaną parametrem alpha: $w_{potomka} = \alpha \cdot w_{rodzica\ 1} + (1 - \alpha) \cdot w_{rodzica\ 2}$
- **Elityzm:** klasyczny - najlepszy osobnik z bieżącego pokolenia jest przenoszony do następnego, co gwarantuje monotoniczną zbieżność algorytmu.

Funkcja oceny

Problemem optymalizacyjnym jest minimalizacja całkowitego kosztu sieci, w tym celu zdefiniowano funkcję F_{celu} :

$$F_{celu} = \sum_{e \in E} \lceil \frac{o_e}{m} \rceil \rightarrow \min$$

gdzie o_e to sumaryczne obciążenie krawędzi e wynikające ze zsumowania ruchu wszystkich zapotrzebowień przechodzących przez tę krawędź. Przystosowanie osobników jest interpretowane odwrotnie do wartości tej funkcji.

Opis danych

Projekt bazuje na rzeczywistych danych sieci teleinformatycznej w Polsce. Sieć składa się z 12 węzłów reprezentujących miasta, dla każdego z nich są zdefiniowane bezpośrednie połączenia (łącznie 18 dwukierunkowych połączeń). Każde połączenie posiada swój koszt, jednak on nie jest kluczowy w naszym projekcie. Dostępny jest również zbiór zapotrzebowień, który obejmuje ruch między parami miast, dla każdego zapotrzebowania zdefiniowano listę dozwolonych ścieżek.

Eksperymenty

Eksperiment 1: Test poprawności działania oraz porównanie agregacji z dezagregacją

Cel 1: Weryfikacja stabilności i efektywności algorytmu dla domyślnych parametrów.

Cel 2: Porównanie działania algorytmu dla trybu agregacji i dezagregacji.

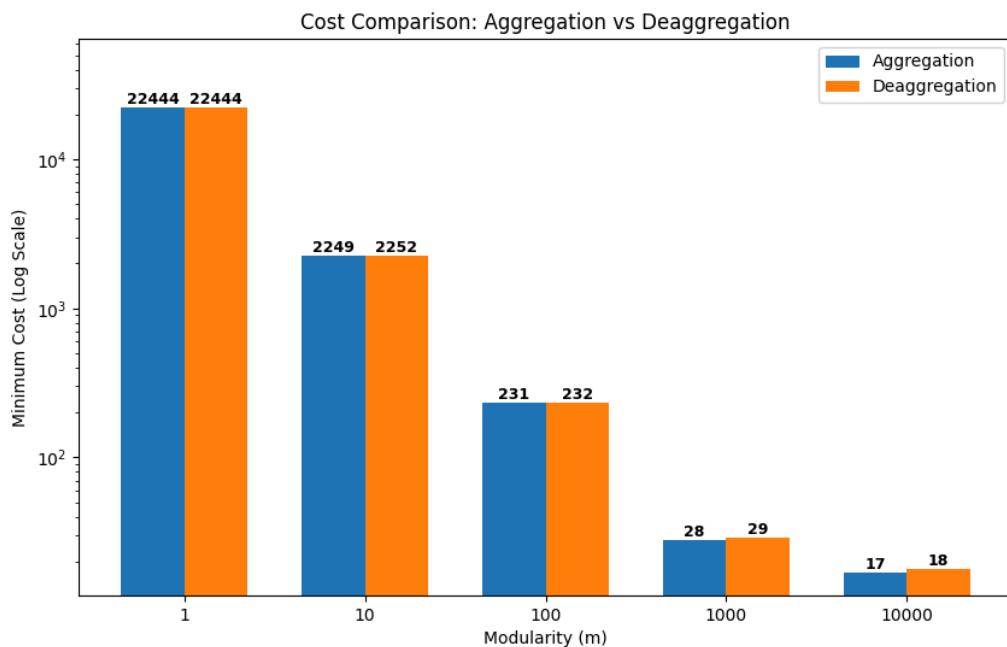
Kluczowe parametry: pop=300, gens=100, repeats=10, seed=67

Wyniki:

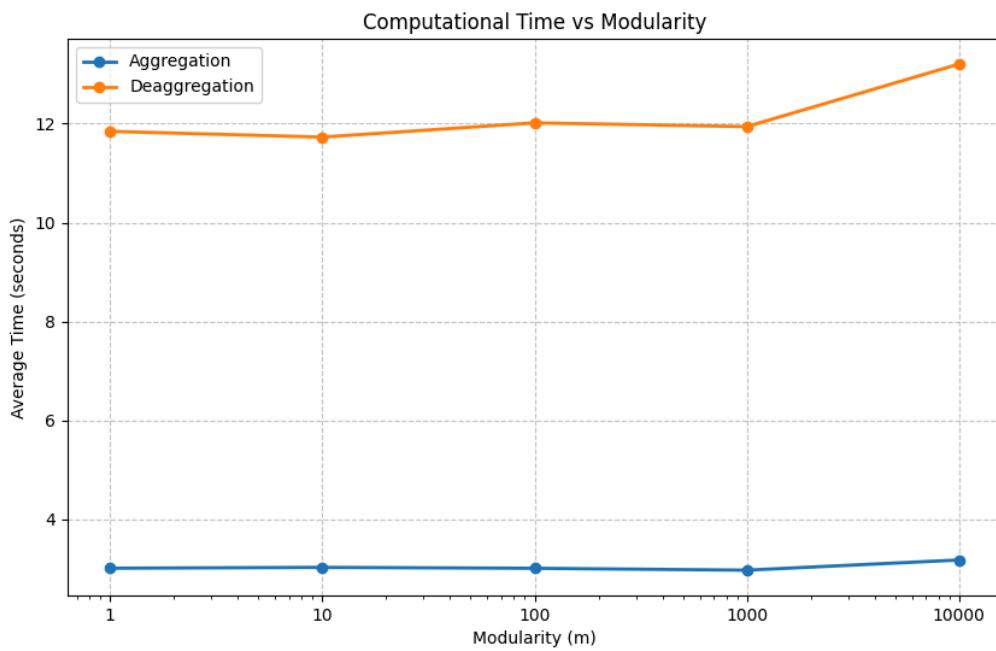
Tryb	Modularność	Koszt	Medianą	STD kosztu	Zbieżność	Czas [s]
Agregacja	1	22444	22444	0	0	3,01
	10	2249	2249,9	0,3	16,4	3,03
	100	231	231	0	9,3	3,01
	1000	28	28,8	0,4	12,9	2,97
	10000	17	17	0	7,4	3,17
Dezagregacja	1	22444	22444	0	0	11,84
	10	2252	2252	0	0	11,72
	100	232	232	0	0	12,01
	1000	29	29	0	0	11,93
	10000	18	18	0	0	13,20

Doprecyzowanie nazw kolumn: "Koszt" - najlepszy (najniższy) znaleziony koszt rozwiązania, "Medianą" - mediana kosztu ze wszystkich powtórzeń symulacji, "STD kosztu" - odchylenie standardowe kosztu, "Zbieżność" - średni numer pokolenia zbieżności, czyli ostatniego pokolenia w którym zaobserwowano poprawę wyniku, "Czas" - czas działania algorytmu).

Wykres kosztów najlepszego znalezionego rozwiązania dla zadanych trybów i modularności:



Wykres średniego czasu działania dla zadanych trybów i modularności:



Analiza i wnioski:

Analiza przeprowadzonych symulacji potwierdziła skuteczność implementacji modelu matematycznego oraz przyjętych mechanizmów ewolucyjnych. Zgodnie z przewidywaniami, wzrost wartości modularności prowadzi do wyraźnej redukcji kosztu sieci (liczby modułów).

Warto jednak zaznaczyć, że - szczególnie dla wyższych modularności - spadek kosztu jest nieproporcjonalny do wzrostu modularności. Wynika to z faktu, że przy dużych pojemnościach modułów występuje zjawisko niepełnego wykorzystania ich przepływności.

Kolejną, wartą wspomnienia obserwacją, jest fakt, że dla modularności $m = 1$ wyniki dla obu trybów przydziału ruchu są identyczne. Jest to rezultat zgodny z oczekiwaniami, przy jednostkowej pojemności każdego modułu, algorytm nie otrzymuje żadnej korzyści z "upakowania ruchu".

W trybie agregacji algorytm wykazał wysoką stabilność oraz zdolność do poprawy rozwiązania startowego. Niskie odchylenie standardowe kosztu świadczy o powtarzalności wyników i odporności na losowy charakter operatorów genetycznych.

Zachowanie algorytmu w trybie dezagregacji jest jednak mniej zadowalające, ponieważ dla żadnej z modularności, algorytm nie był w stanie znaleźć rozwiązania lepszego od startowego. Zjawisko to prawdopodobnie wynika z połączenia charakterystyki przyjętej funkcji kosztu, sposobie mutacji oraz specyfiki badanej sieci. Dezagregacja, promująca rozproszony przepływ ruchu, podczas działania ma tendencje do tworzenia wielu nowych połączeń, które eksploatowane są w bardzo małym stopniu. Zjawisko to wzmacnia także mutacja gaussowska, której szum może zainicjować dodatkowe, nieefektywne wykorzystywane moduły. Taka alokacja pełnego modułu w celu obsłużenia niewielkiego obciążenia jest surowo penalizowana przez funkcję sufitu obecną w funkcji celu, co uniemożliwia algorytmowi ich dalszą eksplorację.

Należy również odnotować różnice w złożoności obliczeniowej: tryb dezagregacji wymagał średnio aż czterokrotnie dłuższego czasu działania, co dodatkowo przemawia za tym, że w naszym przypadku tryb ten jest nieefektywny.

Wizualizacja obciążenia sieci dla jednego z rozwiązań w trybie agregacji (m=10.0):



Eksperyment 2: Wpływ heurystycznej inicjalizacji na efektywność algorytmu

Cel 1: Zbadanie zdolności algorytmu do optymalizacji struktury sieci bez wspomagania się deterministycznie wygenerowanym osobnikiem.

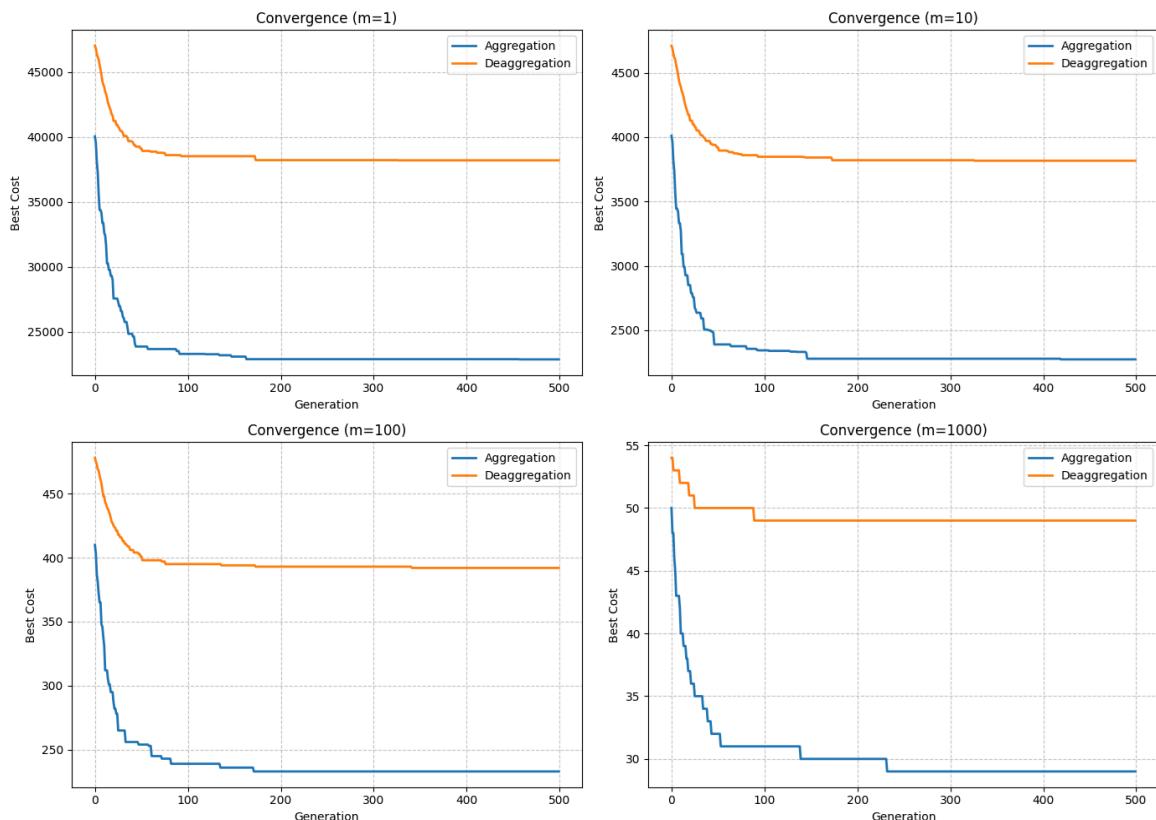
Cel 2: Test działania algorytmu dla trybu dezagregacji.

Kluczowe parametry: pop=500, gens=500, repeats=5, seed=67

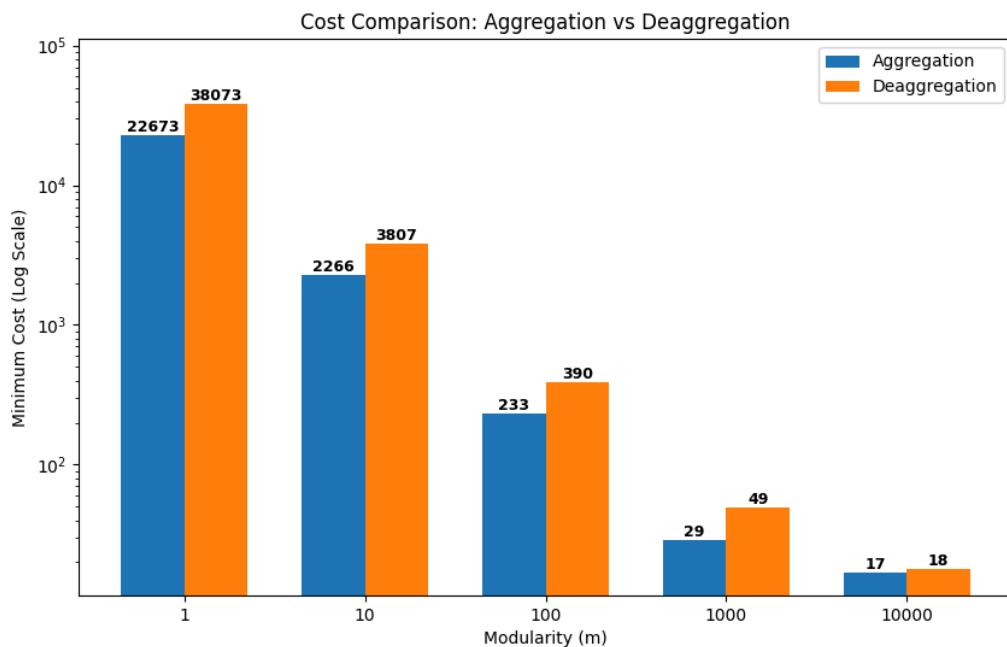
Wyniki:

Tryb	Modularność	Koszt	Mediania	STD kosztu	Zbieżność	Czas [s]
Agregacja	1	22673	22740,2	70,50	360	26,20
	10	2266	2282,6	13,80	231,2	26,05
	100	233	234,6	1,35	226,6	25,70
	1000	29	29,8	0,4	212	26,28
	10000	17	17	0	6,8	27,32
Dezagregacja	1	38073	38246,8	178,17	281,6	103,33
	10	3807	3822,2	12,74	281	110,04
	100	390	391,6	1,49	239,2	113,87
	1000	49	49	0	79,6	110,70
	10000	18	18	0	0	111,74

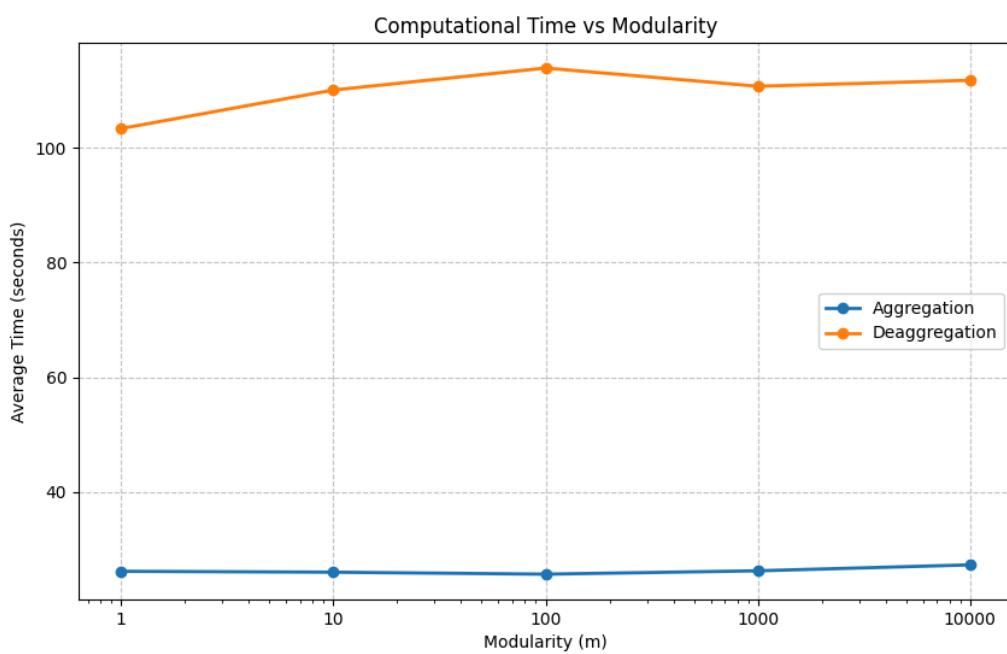
Wykres pokolenia zbieżności obu trybów dla modularności $\in \{1, 10, 100, 1000\}$:



Wykres kosztów najlepszego znalezionego rozwiązania dla zadanych trybów i modularności:



Wykres średniego czasu działania dla zadanych trybów i modularności:



Analiza i wnioski:

Porównując wyniki eksperymentu 2 z wynikami eksperymentu 1, widzimy potwierdzenie tego jak kluczowy jest dla naszego algorytmu osobnik wygenerowany deterministycznie. Mimo tego, że eksperiment ten potwierdził prawidłowe działanie dezagregacji (faktyczne dążenie do optymalnego rozwiązania, czego nie dało się wywnioskować z pierwszego eksperymentu), nie sposób nie zauważyc, że ponownie ujawnił jej krytyczną słabość.

Możemy przypuszczać, że znaczco gorsze wyniki dla tego trybu w porównaniu do agregacji, wynikają z losowo przydzielonych wag podczas inicjalizacji osobników, to natomiast drastycznie rozprasza ruch na wszystkie dostępne trasy. Tak jak zostało to wspomniane podczas analizy przeprowadzonej przy eksperymencie pierwszym, jest to niezwykle karane przez naszą funkcję kosztu (przez funkcję sufitu). Nawet minimalny przepływ na krawędzi wymusza alokację pełnego modułu, a co za tym idzie także wysokiego kosztu. Rozpraszająca wartości natura mutacji gaussowskiej, po raz kolejny nie wydaje się pomagać.

Dużo bardziej satysfakcjonujące wyniki algorytm uzyskał dla trybu agregującego. Mimo losowego generowania populacji startowej, algorytm potrafił samodzielnie zredukować koszt do wartości bardzo zbliżonych do tych z eksperymentu 1. Przykładowo, dla modularności $m = 1$, zaczynając od najniższego kosztu na poziomie powyżej 40 tysięcy (co widzimy na wykresie zbieżności), po 360 generacjach udało mu się zredukować tę wartość do 22673. Porównując wartość tą z 22444 uzyskaną w poprzednim eksperymencie widzimy, że algorytm jest w stanie osiągnąć zadowalające wyniki i bez deterministycznego osobnika w początkowej populacji, choć nie zmienia to faktu, że wciąż jest to bardzo pomocne i wymagane do uzyskania jak najlepszego wyniku.

Opis wykorzystanych narzędzi

- Python 3.10: Wykorzystany język programowania.
- NumPy: Obliczenia macierzowe oraz operacje na chromosomach.
- NetworkX: Wizualizacja struktury grafowej sieci.
- Matplotlib: Generowanie wykresów i wizualizacja.
- JSON: Obsługa danych wejściowych/wyjściowych.
- Argparse: Interfejs CLI.