2021

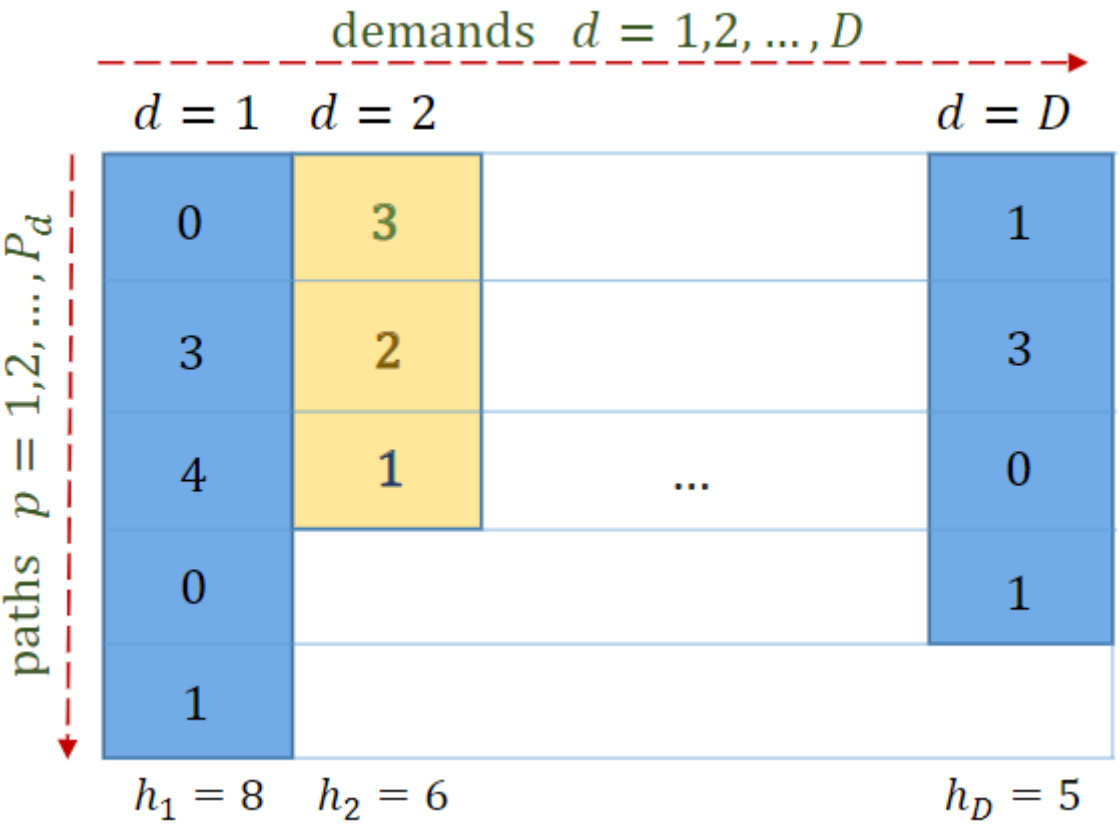
OAST Projekt 1

ALGORYTM EWOLUCYJNY

TOMASZ JASZCZ 293220,
MAJA TURECZEK-ZAKRZEWSKA 261708

1. Opis zaimplementowanego algorytmu

Zaimplementowany algorytm ewolucyjny pozwala w stosunkowo krótkim czasie rozwiązać problemy DAP i DDAP z dosyć dobrym wynikiem. Jego działanie polega na losowym wygenerowaniu określonej liczby chromosomów, które odzwierciedlają rozkład jednostek żądania (demand units) na poszczególnych ścieżkach między węzłami (rys. 1.).



Rysunek 1 Przykładowy Chromosom.

Pierwsza generacja, czyli po prostu pierwszy zestaw N chromosomów, jest generowana losowo, tzn. rozkład jednostek żądania na ścieżkach jest losowy. W kolejnym kroku dla każdego z chromosomów oblicza się wartość funkcji kosztu, która wygląda następująco:

- Dla problemu DAP - $F(x) = max_e\{l(e,x) - C_e\}, (e = 1,2,\dots,E)$ czyli rzeczywiste obciążenie łącza e – pojemność łącza e,
- Dla problemu DDAP - $F(x) = \Sigma_e \xi_e \cdot \left\lceil \frac{l(e,x)}{M} \right\rceil$, czyli funkcja obliczająca łączny koszt wszystkich łączy.

W obu przypadkach dążymy do minimalizacji funkcji kosztu.

W następnym kroku dzieli się populację chromosomów na pary i na podstawie określonego prawdopodobieństwa krzyżuje się je w parach, czyli pobiera z prawdopodobieństwem równym 0,5 gen z pierwszego lub drugiego „rodzica” i dodaje się go do nowego chromosomu pierwszego lub drugiego „dziecka”. Uzyskuje się w ten sposób dwa zupełnie nowe chromosomy utworzone z dwóch poprzednich. Na tym etapie algorytmu populacja jest równa 2N.

W kolejnym kroku mutuje się z zadanym prawdopodobieństwem geny każdego chromosomu, tzn. zmienia się rozkład jednostek żądania o jeden między dwoma, losowo wybranymi, ścieżkami w danym genie.

Następnie ze zbioru chromosomów, w którym znajdują się pierwotne chromosomy (tj. chromosomy z poprzedniej generacji – nie skrzyżowane i nie zmutowane) i nowe chromosomy (po krzyżowaniu i mutacji), wybiera się N najlepszych chromosomów (tych z najniższymi wartościami funkcji kosztu), które stworzą nową generację.

Następnie cały proces się powtarza aż do spełnienia określonego kryterium.

2. Opis implementacji

Program został napisany w języku ***C#*** na platformie ***.NET 3.1.*** z wykorzystaniem środowiska ***Visual Studio 2019.***

W pierwszym kroku z wybranego przez użytkownika pliku wczytywane są dane o badanej sieci. Służy do tego funkcja *ReadFile()*, zwracająca obiekt *Network*, w którym przechowywane są informacje o ilości łączy i żądań, oraz listy z łączami i żądaniami.

Następnie, na podstawie wczytanej listy z żądaniami generowane są chromosomy do pierwszej populacji w sposób następujący:

- dla każdego żądania sprawdzana jest liczba ścieżek, na które może zostać ono rozłożone,
- następnie przydziela się losowo jednostki żądania między wszystkie ścieżki.
- Tak przydzielone żądanie tworzy jeden gen chromosomu.
- Algorytm powtarza się do momentu aż wszystkie żądania zostaną przydzielone, co skutkuje stworzeniem chromosomu (Rysunek 2.1).

```
public Chromosome GenerateChromosome(List<Demand> _demands)
{
    var genes = new List<Gene>();       // Lista genów chromosomu, którą tworzymy na podstawie żądań

    foreach (var demand in _demands)    // Dla każdego żądania tworzymy losowy podział przepływu
    {
        var demandSize = demand.demandSize;     // Zmienna przechowująca przepływność do rozdysponowania po ścieżkach
        var numberOfPaths = demand.numberOfPaths;       // Zmienna przechowująca liczbę ścieżek

        // lista z allelami, czyli podziałem przepływu na poszczególne ścieżki.
        // Początkowo generowana jest jako lista 0, której długość jest równa ilości ścieżek.
        // Na każdej ścieżce początkowo przydzielona przepływność wynosi 0.
        var listOfAlleles = new List<int>(new int[numberOfPaths]);

        var demandToAssign = demandSize;    // Zmienna pomocnicza która służy do sprawdzenia ile pozostało zasobów do przydzielenia

        while (demandToAssign > 0)
        {
            var alleleToIncrement = random.Next(0, numberOfPaths);      // Wybieramy losowo ścieżkę, do której przydzielimy jednostkę przepływności
            listOfAlleles[alleleToIncrement] += 1;      // i przydzielamy właśnie tę jednostkę

            demandToAssign--;       // przydzieliliśmy 1 jednostkę przepływności, więc zmniejszamy pulę
        }

        genes.Add(new Gene(listOfAlleles, demandSize));     // Po rozdysponowaniu wszystkich przepływności tworzymy nowy gen i go dodajemy.
    }

    var chromosome = new Chromosome(genes, 0, 0);       // Po utworzeniu wszystkich genów tworzymy z nich chromosom

    return chromosome;
}
```

Rysunek 2.1 Funkcja do generowania chromosomu.

W kolejnym kroku generowana jest początkowa populacja (startowa) – w tym celu tworzymy tyle chromosomow ile jest populacji, a następnie do populacji dodawany jest każdy chromosom (Rysunek 2.2)

```
public Population GenerateStartingPopulation(List<Demand> _demands, int _populationSize = DEFAULT_POPULATION_SIZE)
{
    var firstPopulation = new Population()
    {
        generationNumber = 1
    };

    for (int i = 0; i < _populationSize; i++)   // stwórz tyle chromosomów ile jest populacji
    {
        var chromosome = GenerateChromosome(_demands);
        firstPopulation.Chromosomes.Add(chromosome);        // i dodaj do populacji każdy chromosom
    }

    return firstPopulation;
}
```

Rysunek 2.2 Funkcja do generowania początkowej populacji

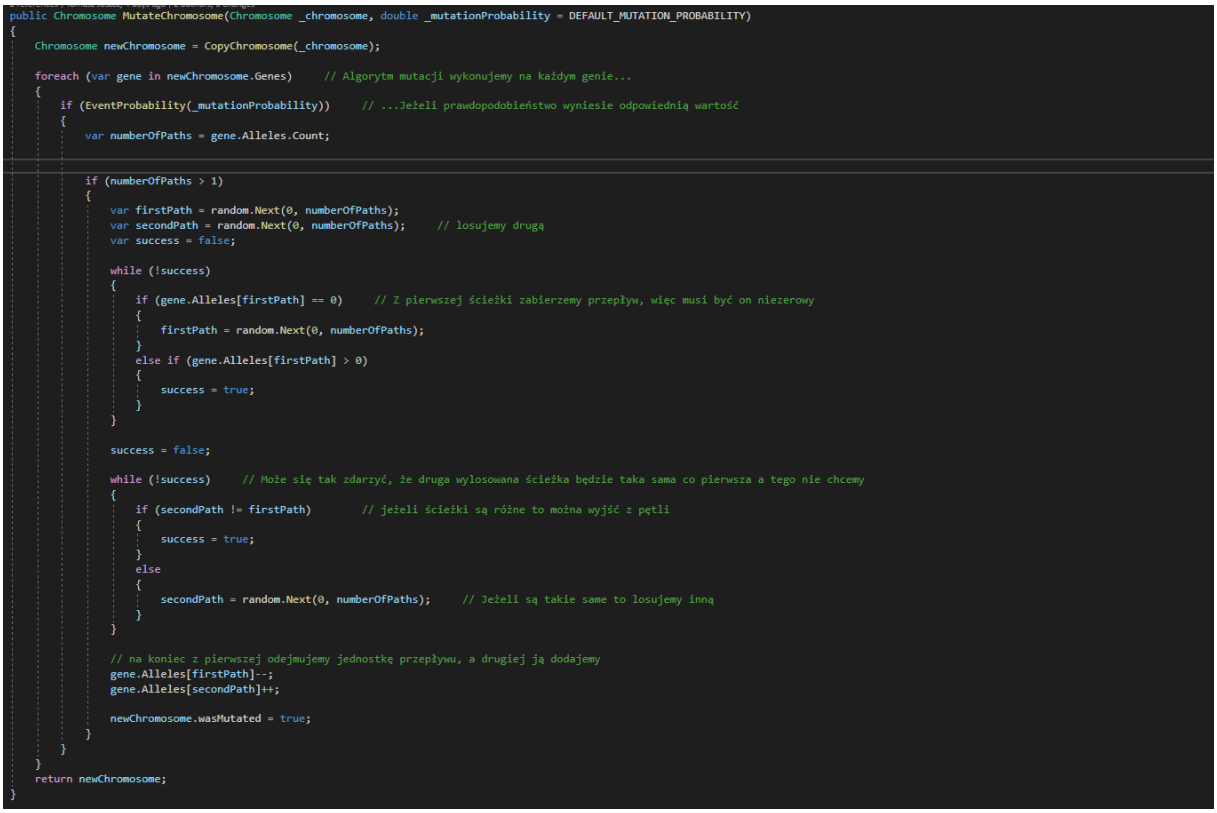
Następnie obliczane jest dopasowanie DAP i DDAP. Wykonywane jest to w następujący sposób:

- Wybierany jest problem DAP lub DDAP
  - Dla problemu DAP rusza funkcja DAPSimulation (DDAP Simulation dla problemu DDAP jest bliźniacza, z tą różnicą, że chromosomy sortowane są na podstawie wartości DDAP), która w każdej kolejnej iteracji krzyżuje i mutuje chromosomy. (z zadanymi prawdopodobieństwami). Chromosomy krzyżowane są zgodnie z następującą zasadą: krzyżowanie jest możliwe, gdy będą dostępne minimum dwa chromosomy. Uprzednio posortowana od najlepszego do najgorszego chromosomu populacja jest kopiowana do listy z rodzicami, na której będzie operował algorytm. W każdej iteracji pobieranych jest dwóch rodziców i w przypadku zajścia krzyżowania, na podstawie ich genów, tworzona jest dwójka dzieci. Dzięki temu mamy pewność, że zawsze krzyżowanie ma szansę zajść dla najlepszego i drugiego najlepszego chromosomu, 3 najlepszego i 4 najlepszego itd. Następnie, do pierwszego dziecka dodawany jest gen jednego rodzica, a do drugiego dziecka gen drugiego z rodziców. Tak utworzone chromosomy dodawane są do listy z dziećmi. Następuje usuwanie z listy rodziców pierwszego i drugiego rodzica i wybierane są kolejne dwa chromosomy o indeksach zero i jeden (ogranicza to możliwość ciągłego powtarzania się). Na końcu do rodziców dodawane są dzieci i tworzona jest lista obu pokoleń (Rysunek 2.3).

```
public List<Chromosome> CrossoverChromosomes(List<Chromosome> _chromosomes, double _crossoverProbability = DEFAULT_CROSSOVER_PROBABILITY)
{
    var parentChromosomes = new List<Chromosome>();  // Lista z rodzicami
    parentChromosomes.AddRange(_chromosomes);
    var childrenChromosomes = new List<Chromosome>();   // Lista z dziećmi

    while (parentChromosomes.Count >= 2)        // Wykonujemy krzyżowanie do momentu aż będą minumum 2 chromosmy do skrzyżowania ze sobą
    {
        var firstChromosome = parentChromosomes[0];     // Pobieranie dwóch rodziców
        var secondChromosome = parentChromosomes[1];

        if (EventProbability(_crossoverProbability))    // Jeżeli zachodzi krzyżowanie to...
        {
            var firstChildrenChromosome = new Chromosome();     // ...tworzone są 2 dzieci...
            var secondChildrenChromosome = new Chromosome();

            for (int i = 0; i < firstChromosome.Genes.Count; i++)         // ...na podstawie genów rodziców
            {
                if (EventProbability(0.5))
                {
                    firstChildrenChromosome.Genes.Add(firstChromosome.Genes[i]);    // Dodaj do pierwszego dziecka gen pierwszego rodzica
                    secondChildrenChromosome.Genes.Add(secondChromosome.Genes[i]);  // a do drugiego gen drugiego rodzica
                }
                else // lub odwrotnie
                {
                    firstChildrenChromosome.Genes.Add(secondChromosome.Genes[i]);    // Dodaj do pierwszego dziecka gen drugiego rodzica
                    secondChildrenChromosome.Genes.Add(firstChromosome.Genes[i]);    // a do drugiego gen pierwszego rodzica
                }
            }

            childrenChromosomes.Add(firstChildrenChromosome);       // utworzone chromosomy dodajemy do listy z dziećmi
            childrenChromosomes.Add(secondChildrenChromosome);

            parentChromosomes.RemoveAt(1);      // usuwamy z pomocniczej listy 1 i 2 rodzica, ponieważ potem na początku pętli
            parentChromosomes.RemoveAt(0);      // wybieramy kolejne 2 chromosomy o indeksach 0 i 1 z tej listy i nie chcę żeby się powtarzały w nieskończoność
        }
    }
    _chromosomes.AddRange(childrenChromosomes);     // Na koniec dodajemy do rodziców ich dzieci, dzięki czemu uzyskujemy listę z oboma pokoleniami

    return _chromosomes;        // i ją zwracamy
}
```

Rysunek 2.3 Funkcja krzyżowania chromosomów

- Następnie chromosomy są mutowane (również z zadanym prawdopodobieństwem mutacji) Algorytm mutacji wykonywany jest na każdym genie, jeżeli prawdopodobieństwo mutacji wyniesie zadaną wartość, oraz jeżeli liczba ścieżek w genie jest większa niż 1 (inaczej nie dałoby się wybrać 2 ścieżek do wymiany). Początkowo losowana jest pierwsza, a następnie druga ścieżka. Z pierwszej ścieżki pobieramy przepływ, a do drugiej go dodajemy. W przypadku, gdy pierwsza i druga ścieżka są takie same, następuje ponowne losowanie drugiej ścieżki. Na koniec od pierwszej ścieżki odejmowana jest jednostka przepływu, a do drugiej zaś dodawana.(Rysunek 2.4).



```
public Chromosome MutateChromosome(Chromosome _chromosome, double _mutationProbability = DEFAULT_MUTATION_PROBABILITY)
{
    Chromosome newChromosome = CopyChromosome(_chromosome);

    foreach (var gene in newChromosome.Genes)     // Algorytm mutacji wykonujemy na każdym genie...
    {
        if (EventProbability(_mutationProbability))     // ...Jeżeli prawdopodobieństwo wyniesie odpowiednią wartość
        {
            var numberOfPaths = gene.Alleles.Count;


            if (numberOfPaths > 1)
            {
                var firstPath = random.Next(0, numberOfPaths);
                var secondPath = random.Next(0, numberOfPaths);     // losujemy drugą
                var success = false;

                while (!success)
                {
                    if (gene.Alleles[firstPath] == 0)     // Z pierwszej ścieżki zabierzemy przepływ, więc musi być on niezerowy
                    {
                        firstPath = random.Next(0, numberOfPaths);
                    }
                    else if (gene.Alleles[firstPath] > 0)
                    {
                        success = true;
                    }
                }

                success = false;

                while (!success)     // Może się tak zdarzyć, że druga wylosowana ścieżka będzie taka sama co pierwsza a tego nie chcemy
                {
                    if (secondPath != firstPath)        // jeżeli ścieżki są różne to można wyjść z pętli
                    {
                        success = true;
                    }
                    else
                    {
                        secondPath = random.Next(0, numberOfPaths);     // Jeżeli są takie same to losujemy inną
                    }
                }

                // na koniec z pierwszej odejmujemy jednostkę przepływu, a drugiej ją dodajemy
                gene.Alleles[firstPath]--;
                gene.Alleles[secondPath]++;

                newChromosome.wasMutated = true;
            }
        }
    }
    return newChromosome;
}
```

Rysunek 2.4 Funkcja mutująca chromosomy

- Po krzyżowaniu i mutacji obliczane jest DAP dla każdego chromosomu. Następnie nowe chromosomy są dodawane do starej populacji i w efekcie uzyskujemy populację o liczebności 3N (N – liczebność populacji początkowej). Chromosomy sortowane są ze względu na wartość funkcji F(x) (DAP) i wybierane jest N najlepszych, które utworzą następną generację, oraz sprawdzane jest czy i w jaki sposób zmieniła się wartość funkcji kosztu. Jeżeli nie uległa zmianie to zwiększana jest ilość iteracji w kryterium „ilości iteracji bez poprawy”, w przypadku, gdy jednak się zmieniła, obliczana jest nowa najlepsza wartość DAP oraz iteracja zerowana jest dla kryterium „iteracji bez poprawy”.
- Kolejnym krokiem jest dobranie najepszego chromosomu. W tym celu dodawany jest najlepszy chromosom z danej populacji do drzewa genealogicznego w celu utworzenia historii najlepszych chromosomów.
- W ten sposób otrzymujemy wynik końcowy zwracajacy najlepszy chromosom dla DAP

- W przypadku wyboru DDAP, w analogiczny sposób obliczany jest najlepszy chromosom dla DDAP (funkcja DDAPSimulation)
    - W każdej iteracji tworzona jest kolejna generacja, następuje krzyżowanie i mutowanie chromosomów.
    - Obliczane jest DDAP dla każdego chromosomu
    - Ze starej i nowej generacji chromosomy są sortowane po DDAP i wybierane z nich są najlepsze, które utworzą nową generację

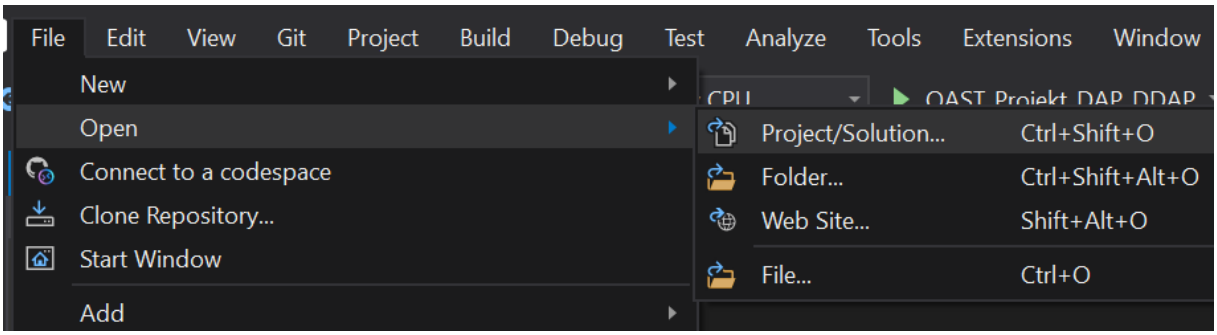
- Jeżeli wartość DDAP się nie poprawiła następuje zwiększanie ilości iteracji do zadanego kryterium
- Jeżeli natomiast wartość DDAP uległa zmianie, obliczana jest nowa jej najlepsza wartość, a wartość iteracji zerowana jest do kryterium iteracji bez poprawy
- Następuje dodane najlepszego chromosomu z danej generacji do drzewa genealogicznego.

3. Instrukcja uruchomienia programu

Do poprawnego uruchomienia programu niezbędne jest posiadanie platformy .NET na komputerze (w systemach Windows jest ona zainstalowana domyślnie) lub zapewnienie odpowiedniego środowiska (najlepiej użytego do implementacji -Visual Studio 2019).

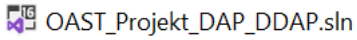
Program można uruchomić poprzez dwukrotne kliknięcie lewym przyciskiem myszy na plik OAST_Projekt_DAP_DDAP.exe znajdujący się w folderze netcoreapp3.1 lub poprzez środowisko Visual Studio (w tym przypadku przy kompilacji zostanie utworzony plik .exe, który następnie będzie można uruchomić poza środowiskiem). W obu przypadkach należy uprzednio załadować pliki zawierające topografię sieci do folderu data, który należy umieścić w wymienionym wyżej folderze netcoreapp3.1.

Nawigując w środowisku do ***File->Open->Project/Solution*** (Rysunek 3.1)



Rysunek 3.1 Nawigacja do otworzenia projektu

należy otworzyć solucję (plik z rozszerzeniem .sln) OAST_Projekt_DAP_DDAP.sln (Rysunek 3.2),



Rysunek 3.2 Wybór solucji

a następnie nacisnąć przycisk run OAST_Projekt_DAP_DDAP (Rysunek 3.3).

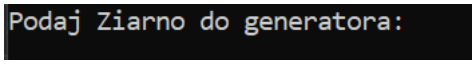
Rysunek 3.3 Przycisk uruuchamiający program

Pojawi się okno konsoli, w którym użytkownik zostanie poproszony o wybór pliku z danymi- ta oraz kolejne wartości wprowadzane są z klawiatury (Rysunek 3.4)

Rysunek 3.4 Wybór pliku z danymi

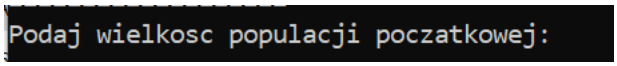
Użytkownik wybiera interesującą go opcję (1,2 lub 3) . W przypadku podania niepoprawnej wartości, program ponownie prosi o podanie jednej z trzech dopuszczalnych opcji.

Następnie, użytkownik wprowadza ziarno generatora (Rysunek 3.5).



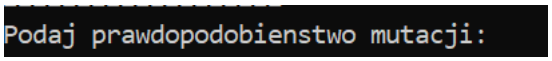
Rysunek 3.5 Podanie ziarna generatora

W kolejnym kroku użytkownik proszony jest o podanie wielkości populacji początkowej (Rysunek 3.6),



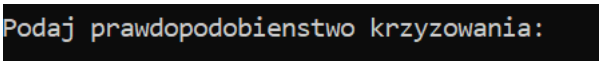
Rysunek 3.5 Podanie wielkości populacji początkowej

Oraz prawdopodobieństwa wystąpienia mutacji (uwaga: format wpisywania liczb dziesiętnych zależy od języka zainstalowanego systemu, tu opisany jest przypadek, gdzie ułamki zapisywane są jako np 0.5)(Rysunek 3.7)



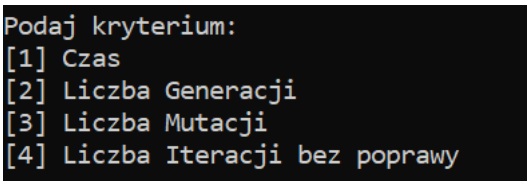
Rysunek 3.7 Wprowadzenie prawdopodobieństwa mutacji

I prawdopodobieństwa wystąpienia krzyżowania (Rysunek 3.8)



Rysunek 3.8 Wprowadzenie prawdopodobieństwa krzyżowania

Następnie, użytkownik wybiera kryterium stopu (czas/liczba generacji/lizba mutacji lub liczba iteracji bez poprawy -Rysunek 3.9)

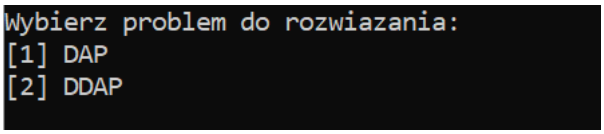


Rysunek 3.9 Wybór kryterium

W zależności od wybranej opcji użytkownik zostanie poproszony o podanie

1. Czasu (wyrażonego w sekundach)
2. Liczby generacji
3. Liczby mutacji
4. Liczby iteracji

a następnie o podanie problemu do rozwiązania – DAP lub DDAP (Rysunek 3.10)



Rysunek 3.10 Wybór problemu do rozwiązania

Po wprowadzeniu wszystkich wartości program rozwiązuje problemy DAP i DDAP z wykorzystaniem algorytmu ewolucyjnego – w oknie konsoli wyświetlana jest trajektoria procesu optymalizacji (sekwencji wartości najlepszych rozwiązań w kolejnych generacjach). Wszystkie wyniki zapisane zostają w pliku tekstowym znajdującym w folderze ***Wyniki*** w katalogu ***bin***. Nazwa pliku z wynikami jest generowana na podstawie parametrów wprowadzonych przez użytkownika, rozwiązywanego problemu, oraz na podstawie pliku z danymi. Przykładowa nazwa pliku to:

net4_Population_10_mutation_0,3_Crossover_0,3_Wyniki_DAP.txt

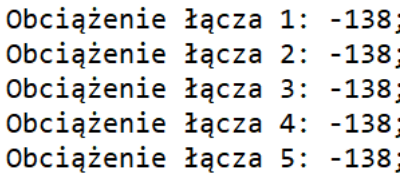
4.Opis najlepszego uzyskanego rozwiązania

Przetestowano algorytm na każdej z sieci: net4.txt, net12_1.txt, net12_2.txt. Dla każdej sieci sprawdzono wyniki dla ziarna równego 100 przy populacji równej 100 dla prawdopodobieństw wystąpienia mutacji i krzyżowania równych 0.3. Pod tym kątem porównano wyniki końcowe. Każde z zadanych kryteriów (czas/liczba generacji/liczba mutacji/liczba iteracji bez poprawy) wynosiło 10.

Najlepsze uzyskane w ten sposób wyniki:

- DAP
  - Net4.txt

    W tym przypadku najniższy wynik DAP (-138) został uzyskany po 22 iteracjach algorytmu i 4083 mutacjach. , przy zadanym kryterium ‘liczba iteracji bez poprawy’Czas optymalizacji wyniósł 0.01s. Uzyskane obciążenie łącza (Rysunek 4.1)



Rysunek 4.1 obciążenie łącza dla sieci Net4 przy zadanym kryterium licba iteracji bez poprawy(DAP)

- Net12_1.txt

W tym przypadku najlepszy wynik DAP (-18) uzyskano przy zadanym kryterium iteracje bez poprawy, po 58 iteracjach algorytmu i 11600 mutacjach. Czas optymalizacji wyniósł 3s, a onciążenie łączy przedstawione jest na zdjęciu poniżej (Rysunek 4.2).

```
Obciążenie łącza 1: -25;
Obciążenie łącza 2: -20;
Obciążenie łącza 3: -21;
Obciążenie łącza 4: -32;
Obciążenie łącza 5: -19;
Obciążenie łącza 6: -57;
Obciążenie łącza 7: -24;
Obciążenie łącza 8: -19;
Obciążenie łącza 9: -68;
Obciążenie łącza 10: -42;
Obciążenie łącza 11: -28;
Obciążenie łącza 12: -25;
Obciążenie łącza 13: -28;
Obciążenie łącza 14: -19;
Obciążenie łącza 15: -48;
Obciążenie łącza 16: -69;
Obciążenie łącza 17: -18;
Obciążenie łącza 18: -19;
```

Rysunek 4.2 uzyskane obciążenia łącza dla sieci Net12_1 przy zadanym kryterium ‘ liczba iteracji bez poprawy’(DAP)

- Net12_2.txt

Dla tej sieci najlepszy uzyskany wynik nastąpił przy pierwszym kryterium (czas). Uzyskano wartość funkcji kosztu równą 19 po 10s czasu optymalizacji, przy 31200 mutacjach, po 156 iteracjach. Uzyskane obciążenie łącza (Rysunek 4.3).

```
Obciążenie łącza 1: 6;
Obciążenie łącza 2: 17;
Obciążenie łącza 3: 6;
Obciążenie łącza 4: 19;
Obciążenie łącza 5: 13;
Obciążenie łącza 6: 10;
Obciążenie łącza 7: 19;
Obciążenie łącza 8: 11;
Obciążenie łącza 9: -6;
Obciążenie łącza 10: 16;
Obciążenie łącza 11: 14;
Obciążenie łącza 12: 12;
Obciążenie łącza 13: 13;
Obciążenie łącza 14: 19;
Obciążenie łącza 15: -38;
Obciążenie łącza 16: -18;
Obciążenie łącza 17: 18;
Obciążenie łącza 18: 12;
```

Rysunek 4.3 obciążenie łącza dla sieci Net12_2 przy zadanym kryterium czasu (DAP)

- DDAP
  - Net4.txt

    Dla tej sieci wyniki przy zadaniu kryterium czasu jak i kryteriumiteracji bez poprawy były zbliżone. Najniższą uzyskaną wartością funkcji kosztu było 13. Przy kryterium iteracji bez poprawy, wynik ten został uzyskany znacznie szybciej, gdyż już po 12 iteracjach w czasie 0.01s, pezy 4074 mutacjach.Uzyskane obciążenie łącza (Rysunek 4.4).

```
Obciążenie łącza 1: -138;
Obciążenie łącza 2: -138;
Obciążenie łącza 3: -142;
Obciążenie łącza 4: -138;
Obciążenie łącza 5: -138;
```

Rysunek 4.4 uzyskane wynikowe obciążenie łącza przy kryterium liczba ieracji bez poprawy dla sieci NET4 (DDAP)

  - Net12_1.txt

    Dla tej sieci najniższy uzyskany wynik funkcji kosztu DDAP wyniósł 36. Nastąpiło przy kryterium czasu. Czas optymalizacji wyniósł 10s, a obciążenie łącza zostało przedstawione na zdjęciu poniżej (Rysunek 4.5).

```
Obciążenie łącza 1: -1;
Obciążenie łącza 2: -11;
Obciążenie łącza 3: -5;
Obciążenie łącza 4: -6;
Obciążenie łącza 5: 93;
Obciążenie łącza 6: -78;
Obciążenie łącza 7: -58;
Obciążenie łącza 8: -30;
Obciążenie łącza 9: -107;
Obciążenie łącza 10: -36;
Obciążenie łącza 11: -8;
Obciążenie łącza 12: -7;
Obciążenie łącza 13: -9;
Obciążenie łącza 14: 88;
Obciążenie łącza 15: -36;
Obciążenie łącza 16: -100;
Obciążenie łącza 17: -59;
Obciążenie łącza 18: -2;
```

Rysunek 4.5 Obciążenie łącza dla sieci Net12_1, przy zadanym kryterium: czas(DDAP)

  - Net12_2.txt

    W tym przypadku najlepszy wynik uzyskano przy zadanym kryterium liczby generacji. Najniższa uzyskana funkcja kosztu DDAP: 46. Wynik ten został uzyskany po 6 iteracjach, przy liczbie mutacji równej 2000, w czasie optymalizacji 0.01s. Uzyskane obciążenia łącza (Rysunek 4.6).

```
Obciążenie łącza 1: 95;
Obciążenie łącza 2: 76;
Obciążenie łącza 3: -11;
Obciążenie łącza 4: 69;
Obciążenie łącza 5: 158;
Obciążenie łącza 6: -22;
Obciążenie łącza 7: -8;
Obciążenie łącza 8: -12;
Obciążenie łącza 9: -52;
Obciążenie łącza 10: -13;
Obciążenie łącza 11: 76;
Obciążenie łącza 12: -1;
Obciążenie łącza 13: 52;
Obciążenie łącza 14: 160;
Obciążenie łącza 15: -12;
Obciążenie łącza 16: -74;
Obciążenie łącza 17: -18;
Obciążenie łącza 18: 54;
```

Rysunek 4.6 Obciążenia łącza dla sieci Net12_2 przy zadanym kryterium: liczba generacji (DDAP)

Po analizie otrzymanych wyników, przeprowadzono dodatkowe symulacje. Okazało się, że kosztem nieznacznie dłuższego czasu optymalizacji można uzyskać nawet niższy wynik DAP dla sieci Net12_1 i Net12_2. Wprowadzając kryterium braku poprawy po 100 iteracjach otrzymano DAP równe -29 (czas optymalizacji wyniósł 10s) dla Net12_1. W przypadku sieci Net12_2 udało się uzyskać najniższy wynik DAP równe 7 przy czasie optymalizacji równym 17s. W zależności od woli użytkownika i jego priorytetów (bezwzględnie niższa wartość funkcji kosztów niezależnie od czasu optymalizacji, czy też zadowolenie się niską, lecz nie najniższą wartością osiągając ją w krótszym czasie).

```
Obciążenie łącza 1: -43;
Obciążenie łącza 2: -29;
Obciążenie łącza 3: -36;
Obciążenie łącza 4: -39;
Obciążenie łącza 5: -29;
Obciążenie łącza 6: -29;
Obciążenie łącza 7: -35;
Obciążenie łącza 8: -42;
Obciążenie łącza 9: -54;
Obciążenie łącza 10: -49;
Obciążenie łącza 11: -38;
Obciążenie łącza 12: -32;
Obciążenie łącza 13: -30;
Obciążenie łącza 14: -29;
Obciążenie łącza 15: -51;
Obciążenie łącza 16: -71;
Obciążenie łącza 17: -29;
Obciążenie łącza 18: -44;
```

Rysunek 4.7 Obciążenia łącza dla sieci Net12_1 przy zadanym kryterium: liczba generacji = 100 (DAP)

```
Obciążenie łącza 1: 3;
Obciążenie łącza 2: 5;
Obciążenie łącza 3: -3;
Obciążenie łącza 4: 4;
Obciążenie łącza 5: 7;
Obciążenie łącza 6: 5;
Obciążenie łącza 7: 5;
Obciążenie łącza 8: 4;
Obciążenie łącza 9: -5;
Obciążenie łącza 10: 3;
Obciążenie łącza 11: 7;
Obciążenie łącza 12: 5;
Obciążenie łącza 13: 1;
Obciążenie łącza 14: 7;
Obciążenie łącza 15: -37
Obciążenie łącza 16: -13
Obciążenie łącza 17: 6;
Obciążenie łącza 18: -9;
```

Rysunek 4.8 Obciążenia łącza dla sieci Net12_2 przy zadanym kryterium: liczba generacji = 100 (DAP)

W przypadku sieci net4 zwiększenie iteracji nie wpływa tak bardzo na wynik, ponieważ jest to mała sieć i posiada ona dużo mniej kombinacji genów niż pozostałe dwie sieci. W przypadku DDAP nie uzyskaliśmy lepszych wyników dla większej ilości iteracji.

W teorii im większa populacja startowa, tym wyższe prawdopodobieństwo uzyskania najlepszego chromosomu. Dzieje się tak, ponieważ chromosomy są generowane losowo w pierwszej populacji i przy odpowiednio dużej ilości chromosomów istnieje duża szansa, że zostaną wygenerowane wszystkie możliwe kombinacje rozkładu przepływności po ścieżkach i już po 1 iteracji uzyskany zostanie najlepszy wynik. By sprawdzić jak najniżej jesteśmy w stanie zejść z wartościami DAP, uruchomiliśmy symulację dla populacji równej 300 i kryterium 1000 iteracji bez poprawy. Dla takich parametrów udało się uzyskać DAP równy -29 dla sieci Net12_1, czyli taki sam jak dla mniejszej populacji i DAP równy 0 dla sieci Net12_2, czyli lepszy o 7. Czas symulacji jednak bardzo wydłużył się dla tych przypadków i wyniósł aż 127 sekund dla sieci Net12_2.

By jednak mieć stuprocentową pewność tego, że jest to najniższy możliwy wynik to uruchomiliśmy symulację dla populacji równej 10 000 i kryterium 10 000 iteracji bez poprawy. Algorytm wykonywał się około 5 godzin dla obu sieci (Net12_1 i Net12_2) i zwrócił dokładnie takie same wyniki jak te opisane w akapicie powyżej.

Ostateczne, najlepsze wyniki:

```
Obciążenie łącza 1: -34;
Obciążenie łącza 2: -29;
Obciążenie łącza 3: -33;
Obciążenie łącza 4: -34;
Obciążenie łącza 5: -29;
Obciążenie łącza 6: -70;
Obciążenie łącza 7: -32;
Obciążenie łącza 8: -41;
Obciążenie łącza 9: -46;
Obciążenie łącza 10: -34;
Obciążenie łącza 11: -36;
Obciążenie łącza 12: -39;
Obciążenie łącza 13: -42;
Obciążenie łącza 14: -34;
Obciążenie łącza 15: -51;
Obciążenie łącza 16: -56;
Obciążenie łącza 17: -29;
Obciążenie łącza 18: -39;
```

Rysunek 4.9 Obciążenia łącza dla sieci Net12_1 przy zadanym kryterium: liczba generacji = 300 (DAP = -29)

```
Obciążenie łącza 1: -9;
Obciążenie łącza 2: -2;
Obciążenie łącza 3: -5;
Obciążenie łącza 4: -1;
Obciążenie łącza 5: 0;
Obciążenie łącza 6: -4;
Obciążenie łącza 7: 0;
Obciążenie łącza 8: -5;
Obciążenie łącza 9: -31;
Obciążenie łącza 10: -7;
Obciążenie łącza 11: -2;
Obciążenie łącza 12: -1;
Obciążenie łącza 13: -16
Obciążenie łącza 14: 0;
Obciążenie łącza 15: -13
Obciążenie łącza 16: -20
Obciążenie łącza 17: 0;
Obciążenie łącza 18: -10
```

Rysunek 4.9 Obciążenia łącza dla sieci Net12_2 przy zadanym kryterium: liczba generacji = 300 (DAP = 0)

Jednak sugerowanie się czasem symulacji nie jest najlepszym wyznacznikiem opłacalności. Wynika to z tego, że na różnych komputerach ten czas będzie oczywiście różny, ponieważ różnią się one podzespołami. Z ciekawości uruchomiliśmy program na tym samym komputerze, ale na dwóch różnych dyskach SSD, z których jeden jest zauważalnie szybszy od drugiego, co poskutkowało skróceniem czasu optymalizacji aż o 7 sekund dla tych samych parametrów. Kolejnym czynnikiem, nawet ważniejszym, który wpływa na czas symulacji jest szybkość procesora.

Podsumowując, dla odpowiednio dużych populacji i odpowiednio dużego kryterium można uzyskać dużo lepsze wyniki, ale trzeba liczyć się z tym, że na słabszych komputerach wykonywanie algorytmu w takiej sytuacji może potrwać bardzo długo.