Warszawa, 30.04.2020

Instytut Telekomunikacji PW

Zakład Sieci i Usług Teleinformatycznych

OAST – projekt

Implementacja algorytmu ewolucyjnego i brute force

1. Opis zadania
   Celem zadania jest zaimplementowanie i przetestowanie algorytmu ewolucyjnego i brute force rozwiązujących problemy DAP i DDAP.

   Wymagania:
   a. Program powinien wczytywać opis topologii sieci i zapotrzebowań z pliku tekstowego (format pliku wejściowego został opisany w Załączniku 1).
   b. Program powinien zapisywać wyniki obliczeń do pliku tekstowego (przykład formatu takiego pliku został przedstawiony w Załączniku 2) - opcjonalnie.
   c. Program powinien umożliwiać przeglądanie pełnej przestrzeni rozwiązań metodą *brute force*,
   d. Program powinien rozwiązywać problemy DAP i DDAP z wykorzystaniem algorytmu ewolucyjnego,
   e. Program powinien umożliwiać:
      - określenie liczności populacji startowej,
      - określenie prawdopodobieństwa wystąpienia krzyżowania i mutacji,
      - wybór kryterium stopu (wymagane są: zadany czas, zadana liczba generacji, zadana liczba mutacji, brak poprawy najlepszego znanego rozwiązania obserwowany w kolejnych N generacjach),
      - zapis trajektorii procesu optymalizacji rozumianej jako sekwencja wartości najlepszych rozwiązań (chromosomów) w kolejnych generacjach,
      - wskazanie ziarna dla generatora liczb losowych.

2. Organizacja
   - zadanie jest realizowane i zaliczane w grupach dwuosobowych,
   - do 7 czerwca należy wysłać prowadzącemu (i.kalesnikau@tele.pw.edu.pl) spakowany kod źródłowy programu i sprawozdanie,
   - terminy konsultacji: IK, czwartki, 15-16;

3. Zawartość sprawozdania
   Powinno zawierać:
   a. opis zaimplementowanych algorytmów (ewolucyjny i brute force),
   b. krótki (co najwyżej jedna strona) opis implementacji,
   c. instrukcję uruchomienia programu,
   d. (dla każdej z sieci net4.txt, net12_1.txt, net12_2.txt, problemy DAP i DDAP) opis najlepszego uzyskanego rozwiązania
      - wartość funkcji kosztu,
      - liczbę wykonanych iteracji AE do znalezienia rozwiązania,
      - czas optymalizacji,

- wartości parametrów algorytmu---liczność populacji, prawdopodobieństwo krzyżowania, prawdopodobieństwo mutacji,
- wynikowe obciążenie łączy, wymiary łączy, rozkład zapotrzebowań na poszczególne ścieżki (przykładowy format zapisu można znaleźć w Załączniku 2).

Uwaga:

Do generacji chromosomów powinien być zastosowany generator liczb pseudolosowych. Generator to funkcja deterministyczna. Do losowania kolejnych liczb wykorzystuje tzw. ziarno (ang. *seed*), całkowicie determinujące wartości kolejnych liczb pseudolosowych.

Dla ustalonego generatora i ziarna generowane będą identyczne liczby losowe bez względu na system operacyjny, nazwę komputera, itp. co pozwala uzyskać powtarzalność otrzymanych wyników.

MM, IK

Warszawa, 10.10.2019

Instytut Telekomunikacji PW

Zakład Sieci i Usług Teleinformatycznych

OAST – projekt, część pierwsza,

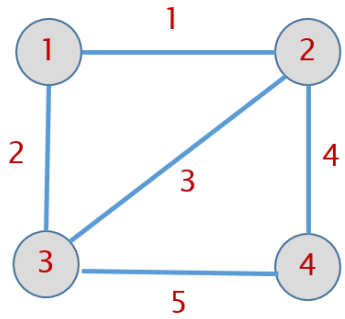
Załącznik 1 – format pliku wejściowego

ćwiczenia - format MP2k (net4.txt)

```
5
1 2 72 1 2
1 3 72 1 2
2 3 72 1 2
2 4 72 1 2
3 4 72 1 2
-1
```

```
6

1 2 3
3
1 1
2 2 3
3 2 5 4

1 3 4
3
1 2
2 1 3
3 1 4 5

1 4 5
3
1 1 4
2 2 5
```

```
2 3 2
3
1 3
2 1 2
3 4 5

2 4 3
3
1 4
2 3 5
3 1 2 5

3 4 4
3
1 5
2 3 4
3 2 1 4
```



h(d): 3, 4, 5, 2, 3, 4
d: (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

routes

P(1,1) = {1}, P(1,2) = {2,3} , P(1,3) = {2,4,5}
P(2,1) = {2}, P(2,2) = {1,3} , P(2,3) = {1,4,5}
P(3,1) = {1,4}, P(3,2) = {2,5}
P(4,1) = {3}, P(4,2) = {1,2} , P(3,3) = {4,5}
P(5,1) = {4}, P(5,2) = {3,5} , P(5,3) = {1,2,5}
P(6,1) = {5}, P(6,2) = {3,4} , P(6,3) = {1,2,4}

ćwiczenia - format MP2k - BNF (Backus-Naur Form)

```
5
1 2 72 1 2
1 3 72 1 2
2 3 72 1 2
2 4 72 1 2
3 4 72 1 2
-1
```

<network> ::= <links><EOL><separator><EOL><demands>

<separator> ::= ”-1”

<links> ::= <number of links><EOL><link list>

<number of links> ::= <integer>

<linkList> := <link>[<EOL><link>]*

<link> ::= <start node> <end node> <number of fibre pairs in cable> <fibre pair cost> <number of lambdas in fibre>

<start node> ::= <node id>

<end node> ::= <node id>

<number of fibre pairs in cable> ::= <integer>

<fibre cost> ::= <float>

<number of lambdas in fibre> ::= <integer>

<node id> ::= <integer>

Notacja ta jest powszechnie używana w informatyce do zapisu składni (syntaktyki) języków programowania i protokołów komunikacyjnych. Została wymyślona przez Johna Backusa w latach 50. w czasie prac nad językiem Fortran, a następnie zmodyfikowana przez Petera Naura i użyta do zdefiniowania składni języka Algol (z Wikipedii).

ćwiczenia - format MP2k - BNF (Backus-Naur Form) - 2

```
<demands> ::= <number of demands><EOL><demand list>

<number of demands> ::= <integer>
<demand list> ::= <demand>[<EOL><demand>]*
<demand> ::= <start node> <end node> <demand volume><EOL><demand paths>
<demand volume> ::= <integer>

<demand paths> ::= <number of demand paths><EOL><demand path list>
<number of demand paths> ::= <integer>
<demand path list> ::= <demand path>[<EOL><demand path>]*
<demand path> ::= <demand path id> <link list><EOL>
<demand path id> ::= <integer>
<link list> ::= <link id>[ <link id>]*
<link id> ::= <integer>
```

```
6

1 2 3
3
1 1
2 2 3
3 2 5 4

1 3 4
3
1 2
2 1 3
3 1 4 5

1 4 5
3
1 1 4
2 2 5
```

```
2 3 2
3
1 3
2 1 2
3 4 5

2 4 3
3
1 4
2 3 5
3 1 2 5

3 4 4
3
1 5
2 3 4
3 2 1 4
```

Instytut Telekomunikacji PW

Zakład Sieci i Usług Teleinformatycznych

OAST – projekt, część pierwsza,

Załącznik 2 – przykład format pliku wyjściowego

```
<solution > ::= <link part><EOL><demand part>

<link part> ::= <number of links><EOL><link load list>
<number of links> ::= <integer>
<link load list> ::= <link load>[<EOL><link load>]*
<link load> ::= <link id> <number of signals> <number of fibers>
<link id> ::= <integer>
<number of signals> ::= <integer>
<number of fibers> ::= <integer>

<demand part> ::= <number of demands><EOL><demand flow list>
<number of demands> ::= <integer>
<demand flow list> ::= <demand flow>[<EOL><demand flow>]*
<demand flow> ::= <demand id> <number of demand paths><EOL><demand path flow
list>
<demand id> ::= <integer>
<number of demand paths> ::= <integer>
<demand path flow list> ::= <demand path flow>[<EOL><demand path flow>]*
<demand path flow> ::= <path id> <path signals count>
<path id> ::= <integer>
<path signals count> ::= <integer>
```

OAST – projekt, część pierwsza,

Załącznik 3 – przykładowa postać algorytmu

Schemat proponowanego algorytmu ewolucyjnego:

begin
  $t := 0$
  $\mathbf{P_0} := \emptyset$
  repeat μ razy {inicjalizacja}
  begin
    $\boldsymbol{x}$:= funkcja alokacji (chromosom)
    ewaluacja $f(\boldsymbol{x})$
    $\mathbf{P_0} := \mathbf{P_0} \cup \{\boldsymbol{x}\}$
  end
  while (**not** kryterium stopu) **do**
  begin
    $\mathbf{O_t} := \emptyset$
    repeat λ razy {reprodukcja}
    begin
      $\boldsymbol{x}$:= wybierz losowo z $(\mathbf{P_t})$
      $\boldsymbol{x}'$:= kopiuj $(\boldsymbol{x})$
      $\mathbf{O_t} := \mathbf{O_t} \cup \{\boldsymbol{x}'\}$
    end
    podziel $\mathbf{O_t}$ na $\lambda/2$ rozłącznych dwuelementowych
    podzbiorów K_m $(m = 1, 2, \ldots \lambda/2)$
    for each $(\boldsymbol{x}^{C_1}, \boldsymbol{x}^{C_2} \in K_m)$, $m = 1, \ldots, \lambda/2$ **do**
      if $u_{(0..1)} < p_{cross}$ **then** krzyżowanie $(\boldsymbol{x}^{C_1}, \boldsymbol{x}^{C_2})$
    for each $\boldsymbol{x} \in \mathbf{O_t}$ **do**
    begin
      if $u_{(0..1)} < p_{mut}$ **then** mutacja $(\boldsymbol{x})$
      ewaluacja $f(\boldsymbol{x})$
    end
    $\mathbf{P_{t+1}} := \emptyset$
    repeat μ razy {selekcja}
    begin
      x := wybierz najlepsze z $(\mathbf{P_t} \cup \mathbf{O_t} \setminus \mathbf{P_{t+1}})$
      $\mathbf{P_{t+1}} := \mathbf{P_{t+1}} \cup \{\boldsymbol{x}\}$
    end
    $t := t + 1$
end.