Wrocław, 04.04.2017

Laboratorium

Technologii Sieciowych

Temat:

Badanie topologii grafów sieciowych.

Autor: Bartosz Banasik
Informatyka, semestr: 4
Prowadzący: dr Łukasz Krzywiecki

1. Wstęp

1.1 Cele pracy

Celem pracy jest zbadanie topologii grafu przedstawiającego sieć internetową.

2. Program testujący niezawodność[1]

2.1 Wstępne wyjaśnienia

Poprzez niezawodność sieci będziemy rozumieć prawdopodobieństwo nierozspójnienia takiej sieci w dowolnym interwale. Każdej krawędzi ze zbioru E przypisujemy wartość niezawodności 0.95, daną pewną funkcją h(e).

Graf G = <V, E> gdzie V – zbiór wierzchołków, E – zbiór krawędzi.

2.2 Pierwszy test

```
G = ([v1, v2, v3, v4, v5, v6, v7, v8, v9, v10, v11, v12, v13, v14, v15, v16, v17,
v18, v19, v20], [{v1,v2}, {v2,v3}, {v3,v4}, {v4,v5}, {v5,v6}, {v6,v7}, {v7,v8},
{v8,v9}, {v9,v10}, {v10,v11}, {v11,v12}, {v12,v13}, {v13,v14}, {v14,v15},
{v15,v16}, {v16,v17}, {v17,v18}, {v18,v19}, {v19,v20}])

Output: Testowana niezawodnosc wynosi = 37.515 %
```

2.3 Drugi test

Dodajemy 1 krawędź o wartości niezawodności 0.95

```
G1 = G + {v1,v20}
Output: Testowana niezawodnosc wynosi = 73.653 %
```

2.4 Trzeci test

Dodajemy jeszcze 2 krawędzie e(1,10) o niezawodności 0.8 i e(5,15) o niezawodności 0.7

```
G2 = G1 + {{v1,v10}, {v5,v15}} : h({v1,v10}) = 0.8 , h({v5,v15}) = 0.7
Output: Testowana niezawodnosc wynosi = 87.081 %
```

1 Przykładowa implementacja na listingu nr 1.

2.5 Czwarty test

Dodajemy 4 nieistniejące wcześniej losowe krawędzie o niezawodności 0.4

```
Output: Testowana niezawodnosc wynosi = 86.516 %
```

Wnioski: Jak widzimy, wraz ze wzrostem ilości krawędzi niezawodność naszej sieci rośnie, ale do pewnego momentu. Podczas 4 testu widzimy, że po dodaniu losowych 4 krawędzi, o bardzo małej niezawodności, testowana niezawodność całej sieci nieznacznie spada.

3. Użyta metoda testowania

Do testowania posłużyliśmy się metodą Monte Carlo.

Dla każdej krawędzi losujemy wartość z zakresu [0,1), jeśli jest ona większa od danej niezawodności krawędzi, to usuwamy ją. Po przejściu po wszystkich krawędziach sprawdzamy spójność grafu. Jeśli jest spójny, to osiągnęliśmy sukces. Powtarzamy wszystko 100000 razy. Na podstawie tych danych jesteśmy w stanie określić niezawodność sieci. Przykładowy kod wykorzystujący tę metodę napisany w języku Java 8.0 znajduje się na listingu nr 1.

```
private static void test1() {

   int zle = 0;

   for(int i = 0; i < liczbaTestow; i++) {
      if(!testNetwork(generate(N))) zle++;
   }
   double p = (liczbaTestow - zle)/(liczbaTestow * 1.0) * 100.0;
   System.out.println("Testowalismy Siec = "+generate(N).toString());
   System.out.println("Testowana niezawodnosc wynosi = "+ p + " %");
}
```

listing 1

Funkcja generete(N) jako argument przyjmuje liczbę wierzchołków i zwraca nam dany graf, który jest poddawany testom. Przykładowy pojedynczy test znajduje się na listingu nr 2.

```
private static boolean testNetwork(SimpleWeightedGraph<String, DefaultWeightedEdge>
graph){
   for(int i=1; i<=N; i++)
       {
          for(int j=1; j<=N; j++)
          {
             if(i < j)
             {
                String vertex1 = "v" + Integer.toString(i);
                 String vertex2 = "v" + Integer.toString(j);

                 if(graph.containsEdge(vertex1, vertex2))
                 {
                    Random generator = new Random();
                    double p = generator.nextInt(10000+1)/10000.0;
                    if(p > graph.getEdgeWeight(graph.getEdge(vertex1, vertex2)))
                    {
                       graph.removeEdge(vertex1, vertex2);
                    }
                }
             }
           }

          ConnectivityInspector inspector = new ConnectivityInspector(graph);
           if(!inspector.isGraphConnected())
          {
             return false;
          }
       }
       return true;
}
```

listing 2

Dla każdej pary wierzchołków, tzn. dla krawędzi, losujemy prawdopodobieństwo rozspójnienia i porównujemy je z wartością dla danej krawędzi. Jeśli jest większa, usuwamy ją. W międzyczasie sprawdzamy spójność grafu. Jeśli graf przejdzie test, zwracamy ‘true’. W przypadku rozspójnienia zwracamy ‘false’.