Autor: Kamil Król
Numer indeksu: 244949

Sprawozdanie

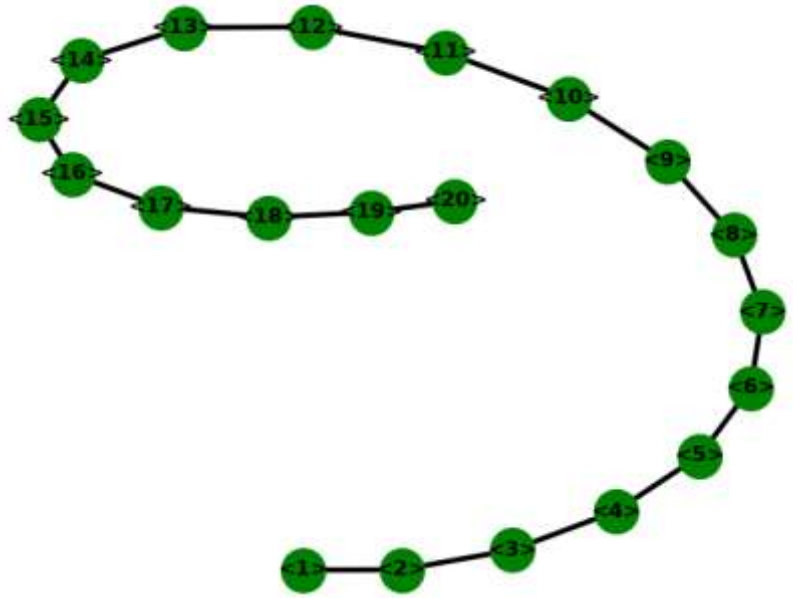
1. Wprowadzenie

Do realizacji zadań wybrałem język Python3.7. Do reprezentacji grafów i działań na nich skorzystałem z biblioteki NetworkX.

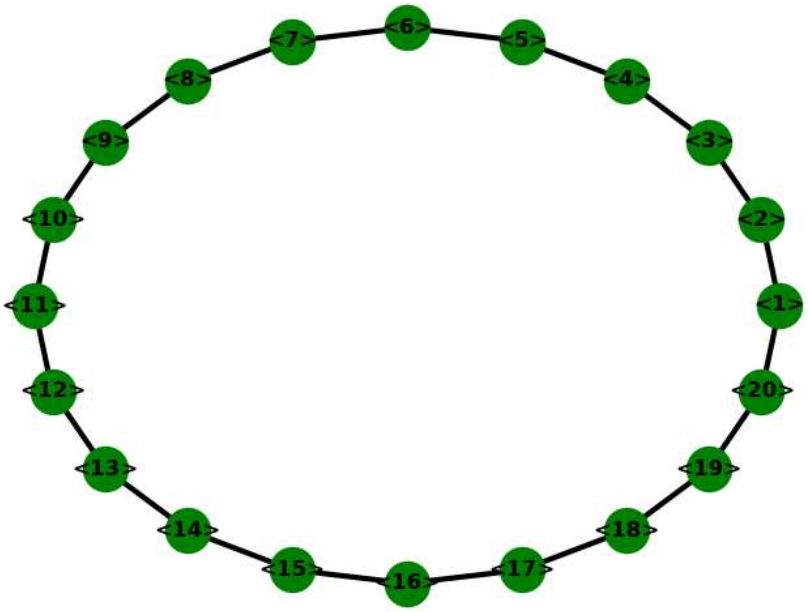
2. Zadanie 1

2.1. Stworzenie grafów do testów

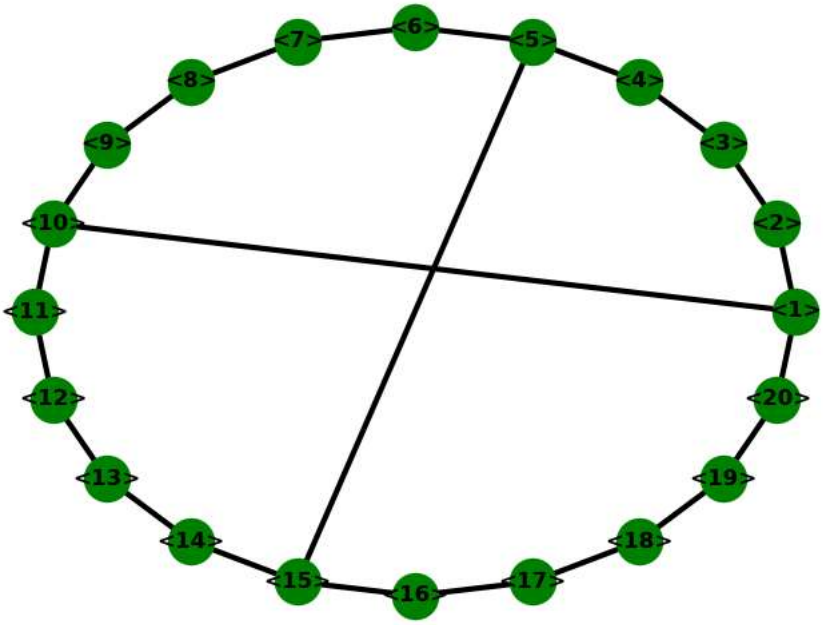
Poniżej znajduje się wizualizacja początkowego grafu zgodnego z treścią zadania. (Graf 1)



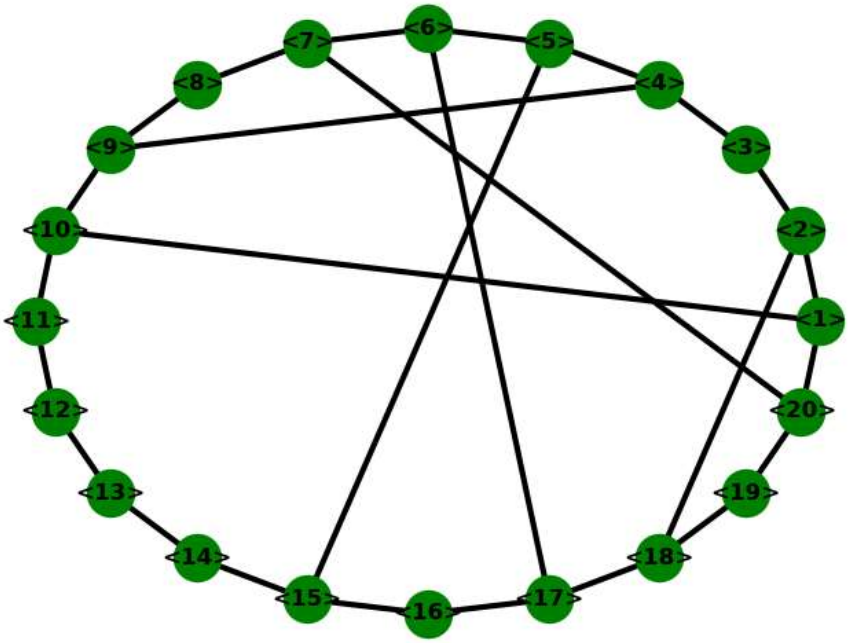
Kolejne grafy były tworzone poprzez modyfikację powyższego. Kolejny graf to graf z dodaną krawędzią e(1,20) przy czym h(e(1,20)) = 0.95. (Graf 2)



Widać, że w skutek dodania krawędzi e(1,20) powstał „okrąg”. Kolejny graf posiada dwie dodatkowe krawędzie: e(1,10) oraz e(5,15) takie, że: h(e(1,10))=0.8, a h(e(5,15))=0.7. (Graf 3)



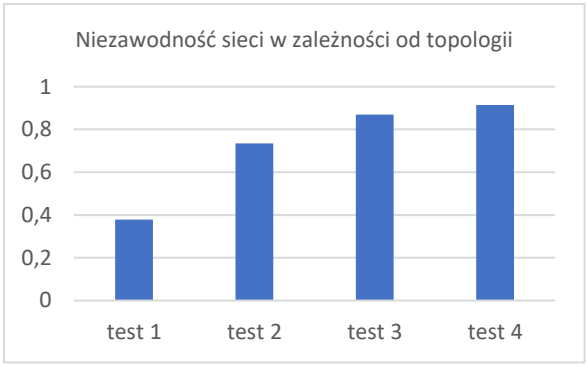
Ostatni graf (Graf 4) różni się od poprzedniego (Grafu 3) dodanymi losowo czterema krawędziami. W szczególności mogło się tak zdarzyć, że dodana losowo krawędź będzie łączyć wierzchołki już połączone, więc między dwoma krawędziami może być więcej niż jedna krawędź.



2.2. Testowanie niezawodności powyższych modelów sieci

Testy przeprowadziłem za pomocą metody Monte Carlo. Ilość pojedynczych prób to 100 000. Wyniki testów znajdują się na zrzucie ekranu poniżej.

```
>>> run_tests()
37797  num of edges:  19
0.37797
73436  num of edges:  20
0.73436
86976  num of edges:  22
0.86976
91416  num of edges:  26
0.91416
```



Prawdopodobieństwo dla pierwszego testu można wyliczyć analitycznie. Jest to $0{,}95^{19} \approx 0.37735360253530761511306362152099609375$, a więc oszacowana wartość jes bardzo blisko wartości rzeczywistej.

3. Zadanie 2

3.1. Sposób reprezentacji sieci

W celu reprezentacji sieci została utworzona klasa. Klasa przechowuje odpowiednie struktury danych i dostarcza odpowiedni interfejs dla użytkownika. Aby stworzyć sieć wymagane są:

-Macierz natężeń (lista list o rozmiarze zgodnym z ilością wierzchołków)

-Zbiór wierzchołków

-Zbiór krawędzi wraz z ich własnościami tzn. przepustowością i prawdopodobieństwem pozostania nieuszkodzoną w dowolnym interwale czasu

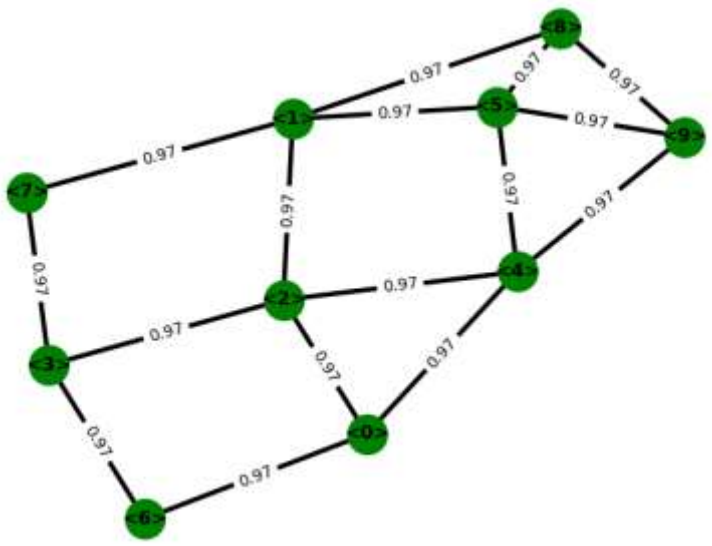
Ponadto można poddać argumenty opcjonalne:

-Średnia wielkość pakietu w bajtach (wartość domyślna to 50)

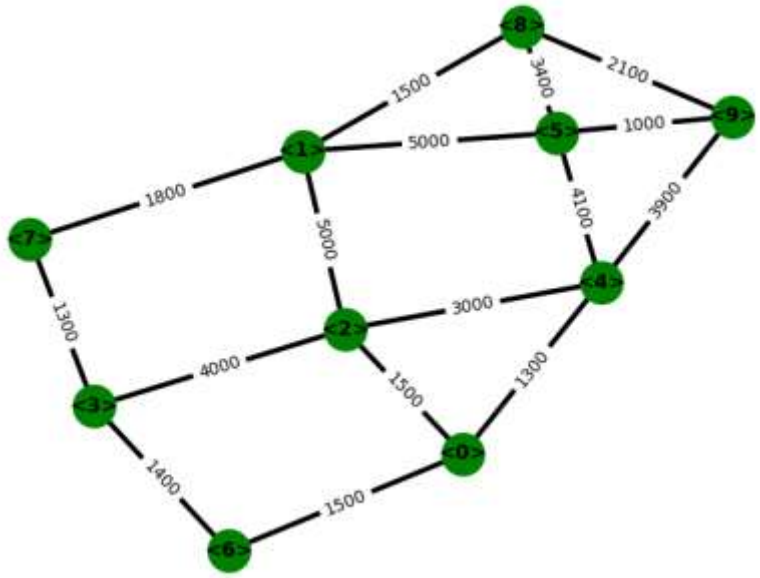
-Maksymalne opóźnienie pakietu (domyślna wartość to 2)

3.2. Proponowany model sieci

Zgodnie z treścią zadania zaproponowałem następującą sieć. Wierzchołki ponumerowałem od 0 do 9. Rozkład krawędzi wraz z ich własnościami znajduje się na poniższych slajdach.



Krawędzie wraz z wartościami prawdopodobieństwami nieuszkodzenia

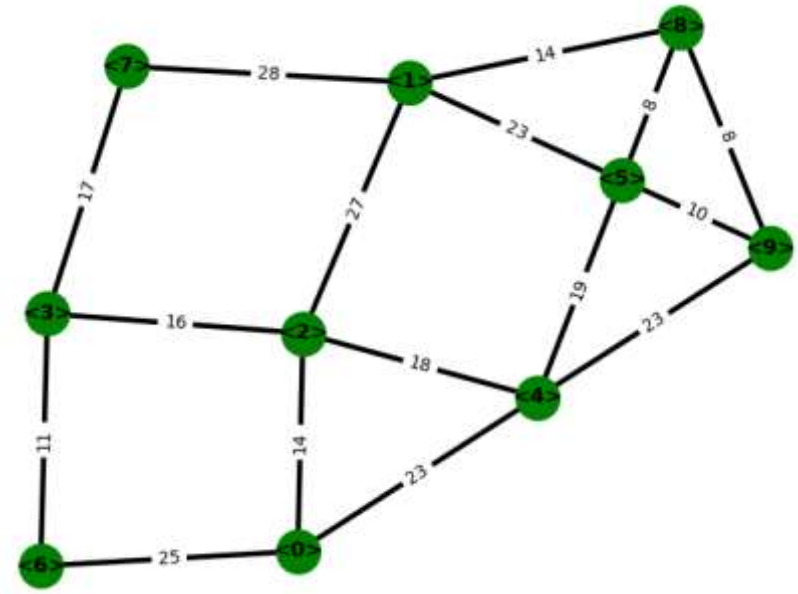


Krawędzie wraz z wartościami ich przepustowości

Zaproponowana prze mnie macierz natężeń wygląda następująco:

```
 0| 1| 2| 3| 4| 5| 6| 7| 8| 9
[0, 2, 2, 2, 2, 2, 2, 2, 2, 2],  # 0
[1, 0, 1, 2, 3, 1, 1, 1, 1, 1],  # 1
[1, 2, 0, 3, 4, 1, 1, 1, 2, 3],  # 2
[1, 1, 1, 0, 1, 1, 1, 1, 3, 3],  # 3
[1, 2, 1, 1, 0, 1, 1, 1, 1, 1],  # 4
[2, 2, 2, 1, 1, 0, 2, 1, 1, 1],  # 5
[4, 3, 1, 1, 1, 1, 0, 2, 3, 1],  # 6
[1, 5, 1, 4, 4, 1, 1, 0, 2, 1],  # 7
[1, 0, 1, 2, 3, 4, 1, 1, 0, 1],  # 8
[2, 3, 1, 1, 2, 3, 1, 1, 1, 0]   # 9
```

Na podstawie tej macierzy natężeń wyliczona zostaje funkcja a (liczba pakietów wprowadzana do danego kanału komunikacyjnego w ciągu sekundy)



Krawędzie wraz z odpowiadającymi im wartościami funkcji a

Wartość funkcji a(e) jest liczona w następujący sposób. Na początek dla wszystkich krawędzi wartość a jest ustawiana na 0. Następnie brane są wszystkie minimalne ścieżki pomiędzy wierzchołkami. Później wartości funkcji a dla odpowiednich krawędzi są inkrementowane wraz z iterowaniem po tych najmniejszych ścieżkach. Poniżej dane krawędzi w jednej grafice.

```
>>> n.show_edges_data()
[(<0>, <4>, {'a': 23, 'capacity': 1300, 'probability': 0.97}),
 (<0>, <6>, {'a': 25, 'capacity': 1500, 'probability': 0.97}),
 (<0>, <2>, {'a': 14, 'capacity': 1500, 'probability': 0.97}),
 (<1>, <5>, {'a': 23, 'capacity': 5000, 'probability': 0.97}),
 (<1>, <7>, {'a': 20, 'capacity': 1800, 'probability': 0.97}),
 (<1>, <8>, {'a': 14, 'capacity': 1500, 'probability': 0.97}),
 (<1>, <2>, {'a': 27, 'capacity': 5000, 'probability': 0.97}),
 (<2>, <4>, {'a': 18, 'capacity': 3000, 'probability': 0.97}),
 (<2>, <3>, {'a': 16, 'capacity': 4000, 'probability': 0.97}),
 (<3>, <7>, {'a': 17, 'capacity': 1300, 'probability': 0.97}),
 (<3>, <6>, {'a': 11, 'capacity': 1400, 'probability': 0.97}),
 (<4>, <9>, {'a': 23, 'capacity': 3900, 'probability': 0.97}),
 (<4>, <5>, {'a': 19, 'capacity': 4100, 'probability': 0.97}),
 (<5>, <9>, {'a': 10, 'capacity': 1000, 'probability': 0.97}),
 (<5>, <8>, {'a': 0, 'capacity': 3400, 'probability': 0.97}),
 (<8>, <9>, {'a': 0, 'capacity': 2100, 'probability': 0.97})]
```

3.3. Sprawdzenie sensowności zaproponowanej sieci

Teraz zgodnie z pierwszym podpunktem zadania 2 należy sprawdzić czy wielkość faktycznej liczby pakietów przechodzących przez daną krawędź jest mniejsza od jej przepustowości. Inaczej czy dla każdego e

a(e) * (średni rozmiar pakietu w bajtach) < c(e)

Poniżej znajduje się wywołanie funkcji sprawdzającej ten warunek dla każdej krawędzi. Wartość zwrócona przez funkcję to prawda, zatem sieć spełnia powyższy warunek.

```
>>> n.check_flows_correctness()
1300  <  1150
1500  <  1250
1500  <  700
5000  <  1150
1800  <  1400
1500  <  700
5000  <  1350
3000  <  900
4000  <  800
1300  <  850
1400  <  550
3900  <  1150
4100  <  950
1000  <  500
3400  <  400
2100  <  400
True
```

Zatem pierwszy podpunkt zadania drugiego został wykonany.

Do pełnego sprawdzenia sensowności sieci brakuje jeszcze sprawdzenia czy średnie opóźnienie pakietu nie jest większe niż maksymalne (kolejny podpunkt zadania). W tym celu wywołamy funkcję, która sprawdza średnie opóźnienie pakietu liczone według wzoru w treści zadania. Nasza maksymalna wartość opóźnienia to 2s.

$$\sum_{e \in E} \frac{a(e)}{\frac{c(e)}{m} - a(e)}$$

```
>>> n.average_delay()
0.15925431127203893
```

Wartość 0.15925431127203893 s jest zdecydowanie mniejsza od 2 s, zatem sieć spełnia wymagania odnośnie szybkości działania.

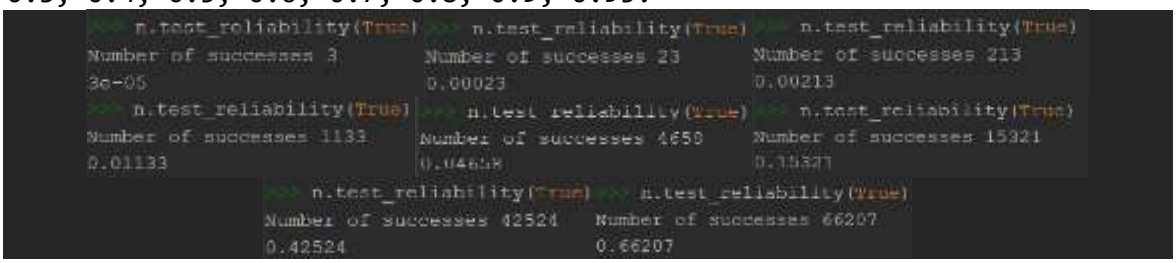
3.4. Obliczenie niezawodności sieci

Niezawodność sieci uwzględnia również prawdopodobieństwa nierozerwania kanałów komunikacyjnych. Niezawodność szacowałem metodą Monte Carlo dla 100 000 pojedynczych prób.

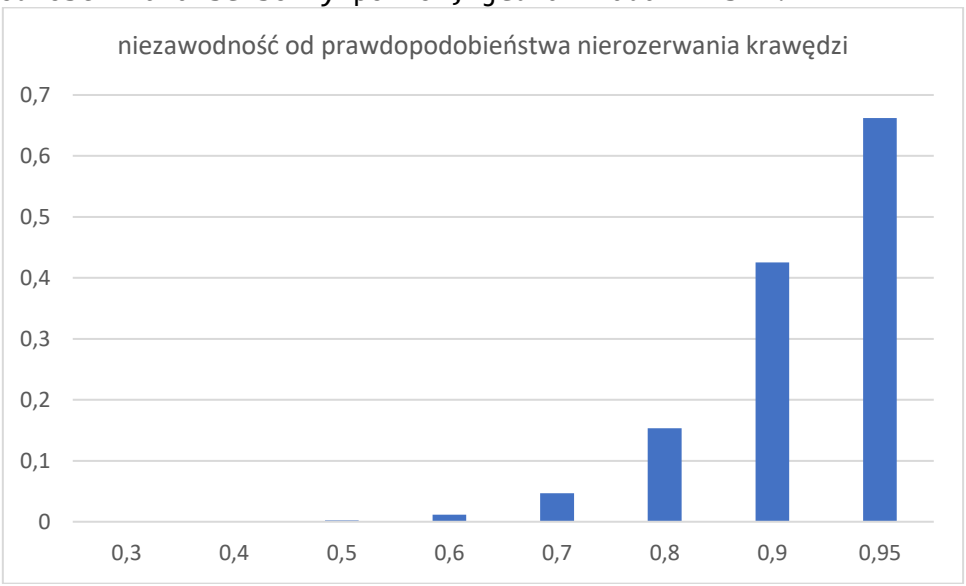
```
>>> n.test_reliability(True)
Number of successes 78035
0.78035
```

3.5. Zależność niezawodności od prawdopodobieństw nierozerwania sieci

Postanowiłem sprawdzić jak będzie się zmieniać niezawodność sieci jeśli zmienię prawdopodobieństwa nierozerwania sieci. Zaproponowane w poprzedniej części sprawozdania wartości to 0.97 dla każdej krawędzi. Rozważymy prawdopodobieństwa kolejno:
0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95.



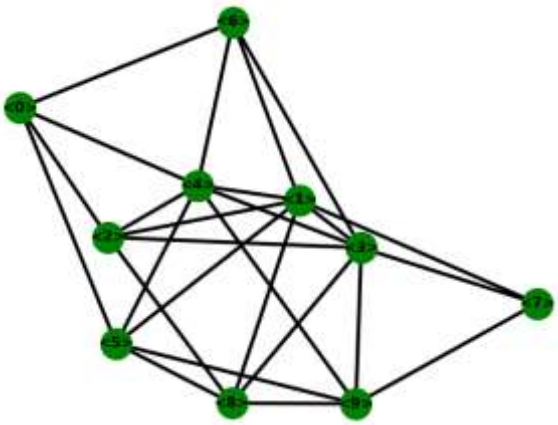
Wywołania znajdują się powyżej, dla każdego wywołania przeprowadzono 100 000 pojedynczych prób. Jak widać prawdopodobieństwa takie jak 0.3 czy 0.4 dały fatalne wyniki. Dopiero od prawdopodobieństwa 0.9 niezawodność miała sensowny poziom, jednak nadal niski.

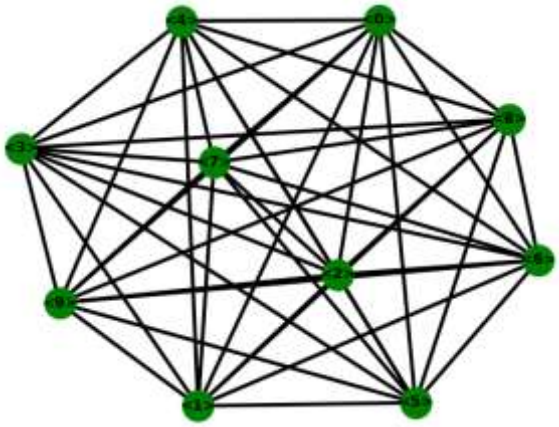


3.6. Próba zmniejszenia wrażliwości sieci na rozerwania

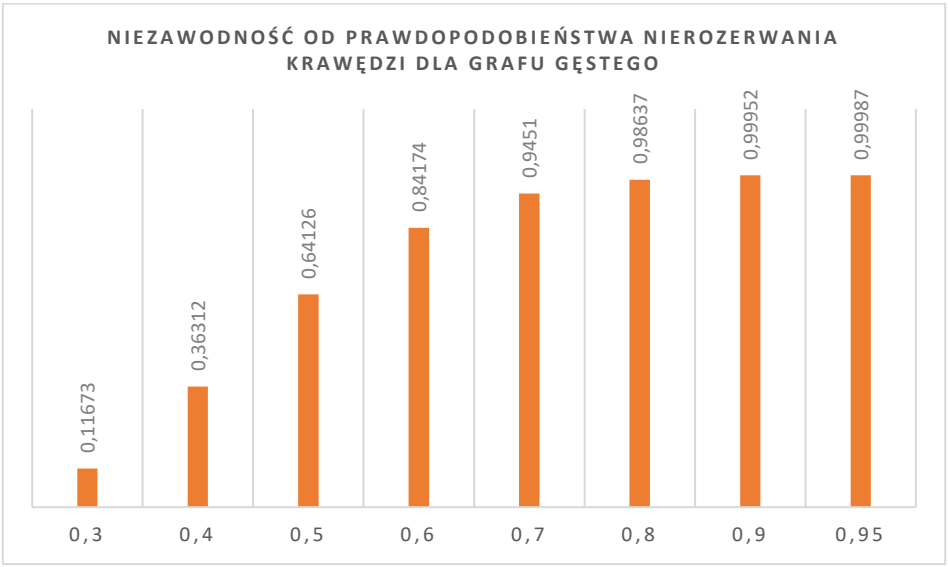
Poprzednia badana przeze mnie sieć okazała się wrażliwa na uszkodzenia krawędzi. Teraz zbadam jak wpływa to dla sieci gęstych, ponieważ intuicyjnie sieci gęste powinne być na to mniej wrażliwe. Na koniec sprawdzę sieć reprezentowana przez graf pełny. Poniżej wizualizacje sieci gęstej (zaproponowana przeze mnie) i „pełnej”. Wartości pojemności dla wszystkich krawędzi ustawiłem na 5000. Macierz natężeń pozostała taka jak poprzednio.

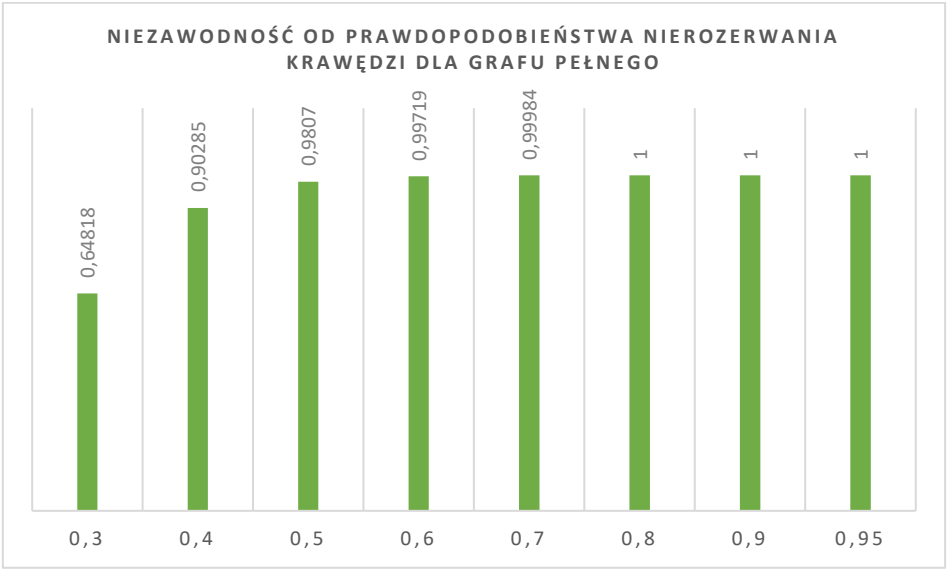
Ilość krawędzi w grafie pełnym wynosi 45, więc zdecydowałem, że graf gęsty będzie miał ich 26 (poprzednio badany miał 16). Poniżej kolejno graf gęsty zaproponowany przeze mnie i graf pełny. Tworząc graf gęsty zadbałem o to by w każdego wierzchołka wychodziły co najmniej 3 krawędzie.





Poniżej wykresy prezentujące jak powyższe grafy są wrażliwe na rozerwania kanałów komunikacyjnych.





Widać wyraźną zależność ze nadmiarowe krawędzi zwiększają niezawodność sieci, nawet w sytuacjach gdy krawędzi są nietrwałe.

3.7. Wpływ maksymalnej wartości opóźnienia na niezawodność

Postanowiłem sprawdzić zależność niezawodności sieci od maksymalnej wartości opóźnienia na sieci z pierwszego podpunktu (16 krawędzi). Średnie opóźnienie dla niej wynosiło ~0.16s. Wybrałem 3 wartości maksymalnego opóźnienia: 0.16s (bardzo bliską średniej) 0.32s (dwukrotnej średniej) i 0.64s.

T_max	0.16s	0.32s	0.64.s
Reliability	0.63516	0.78173	0.78355

Wnioskiem jest to, że jeśli wartość maksymalna wartość opóźnienia pakietu jest bliska średniej (lub na odwrót) to niezawodność sieci spada.