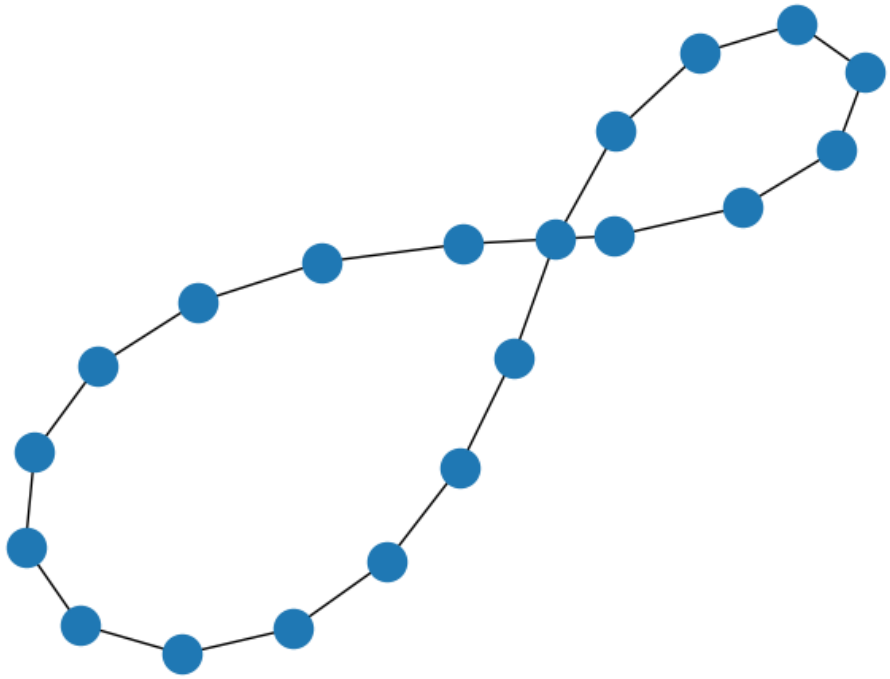
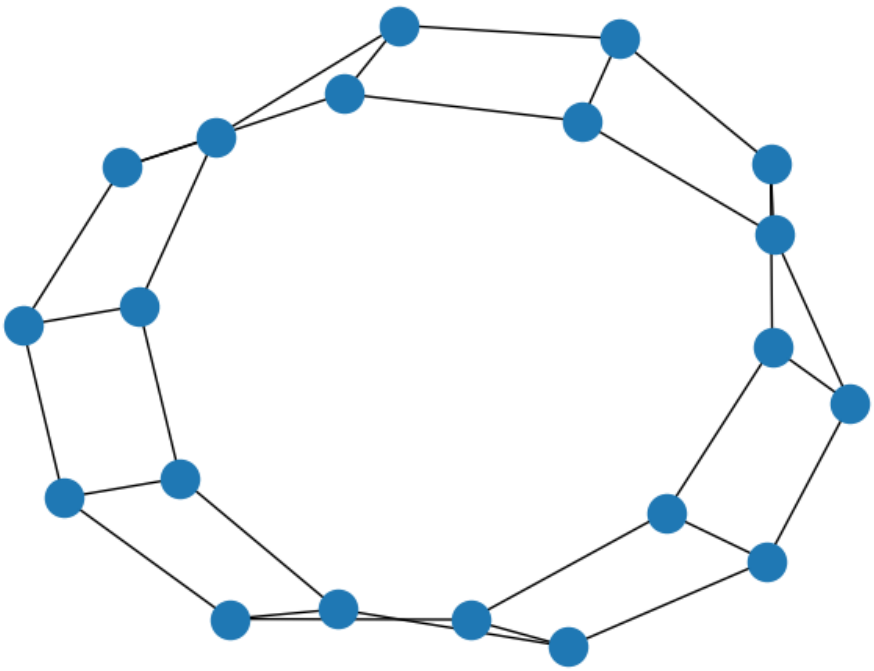
Technologie Sieciowe Lista 2 Sprawozdanie Mateusz Gancarz

Do badań zaproponuję dwa grafy:



Graf cykliczny (20 wierzchołków i 19 połączeń)



Graf o kształcie dwóch połączonych okręgów (20 wierzchołków i 28 połączeń)

Grafy stworzyłem za pomocą następujących funkcji:

```
def create_graph(p, a, c):
    g = nx.cycle_graph(20)
    nx.set_edge_attributes(g, p, 'p')
    nx.set_edge_attributes(g, a, 'a')
    nx.set_edge_attributes(g, c, 'c')
    nx.draw(g)
    plt.show()
    return g
```

```
def create_custom_graph(p, a, c):
    g = nx.cycle_graph(20)
    g.add_edge(0, 10)
    g.add_edge(1, 11)
    g.add_edge(2, 12)
    g.add_edge(3, 13)
    g.add_edge(4, 14)
    g.add_edge(5, 15)
    g.add_edge(6, 16)
    g.add_edge(7, 17)
    g.add_edge(8, 18)
    g.add_edge(9, 19)
    nx.set_edge_attributes(g, p, 'p')
    nx.set_edge_attributes(g, a, 'a')
    nx.set_edge_attributes(g, c, 'c')
    nx.draw(g)
    plt.show()
    return g
```

Do tworzenia grafu wykorzystałem bibliotekę networkx. Tworząc grafy wybrałem wielkość pakietu m = 1, maksymalną ilość przekazywanych danych przez krawędź c = 3000 i szansę na zepsucie się krawędzi p = 10%.

```
g = create_graph(0.9, 1, 3000)
```

Macierz natężeń wypełniłem następującą funkcją, w której wypełniam macierz losowymi wartościami z podanego w argumencie funkcji zakresu:

```python
def create_and_fill_matrix(n, border):
    matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            if (i != j):
                matrix[i][j] = random.randint(1, border)

    return matrix

matr = create_and_fill_matrix(20, 50)
```

Funkcja przepływu, w której iteruję przez najkrótszą ścieżkę między dwoma wierzchołkami znalezioną przez bibliotekę networkx oraz wyliczamy jej koszt:

```python
def a_func(g, n):
    nx.set_edge_attributes(g, 0, 'a')
    for i, row in enumerate(n):
        for j, val in enumerate(row):
            path = nx.shortest_path(g, i, j)
            for k in range(len(path) - 1):
                g[path[k]][path[k + 1]]['a'] += val
```

Do badania niezawodności sieci wykorzystamy następującą funkcję obliczająca opóźnienie:

```python
def count_delay(g, matr, m):
    my_sum = 0
    suma = matr.sum()
    for i in g.edges():
        if (g.get_edge_data(*i).get('a') < g.get_edge_data(*i).get('c')):
            my_sum += (g.get_edge_data(*i).get('a')) / (
                    g.get_edge_data(*i).get('c') / m - g.get_edge_data(*i).get('a'));
    return my_sum / suma
```

Wykonam trzy badania na podanych wcześniej grafach: będę **zwiększać wartości w macierzy natężeń**, **maksymalny przepływ między wierzchołkami** oraz będę **dodawać do grafu kolejne krawędzie**. Badania wykonamy za pomocą pętli, w której z każdą iteracją będziemy zwiększać dane wartości oraz wykonywać pomiary 600 razy, po czym uśrednimy ten wynik. Badania wykonamy za pomocą następujących funkcji:

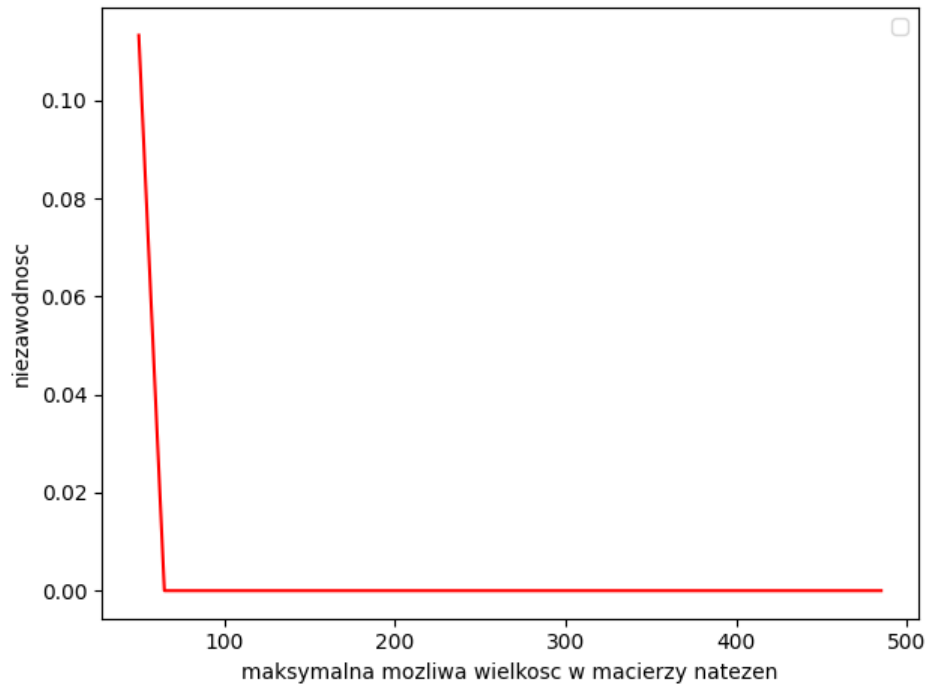
```python
def test1(graph, T_max, ppb, m, n):
    matr = create_and_fill_matrix(20, n)
    nx.set_edge_attributes(graph, ppb, 'p')
    g = nx.Graph(graph)
    for j in g.edges():
        rnd = random.random()
        if rnd > g.get_edge_data(*j).get('p'):
            g.remove_edge(*j)

        if not nx.is_connected(g):
            return 0

    a_func(g, matr)

    for i in g.edges():
        if (g.get_edge_data(*i).get('a') >= g.get_edge_data(*i).get('c') / m):
            return 0

    if (count_delay(g, matr, m) < T_max):
        return 1
    else:
        return 0
```

```python
def test2(graph, T_max, ppb, c, m, matr):
    nx.set_edge_attributes(graph, ppb, 'p')
    nx.set_edge_attributes(graph, c, 'c')
    g = nx.Graph(graph)
    for j in g.edges():
        rnd = random.random()
        if rnd > g.get_edge_data(*j).get('p'):
            g.remove_edge(*j)

        if not nx.is_connected(g):
            return 0

    a_func(g, matr)

    for i in g.edges():
        if (g.get_edge_data(*i).get('a') >= g.get_edge_data(*i).get('c') / m):
            return 0

    if (count_delay(g, matr, m) < T_max):
        return 1
    else:
        return 0
```

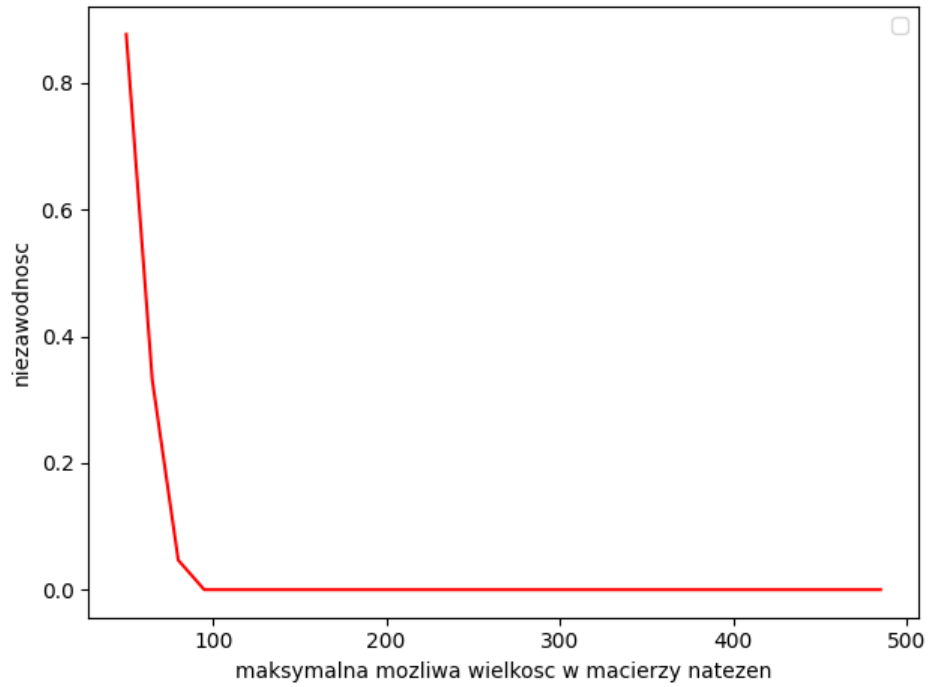
```python
def test3(graph, T_max, ppb, m, matr):
    nx.set_edge_attributes(graph, ppb, 'p')
    g = nx.Graph(graph)
    for j in g.edges():
        rnd = random.random()
        if rnd > g.get_edge_data(*j).get('p'):
            g.remove_edge(*j)

        if not nx.is_connected(g):
            return 0

    a_func(g, matr)

    for i in g.edges():
        if (g.get_edge_data(*i).get('a') >= g.get_edge_data(*i).get('c') / m):
            return 0

    if (count_delay(g, matr, m) < T_max):
        return 1
    else:
        return 0
```

Badanie nr 1 – wpływ maksymalnej wartości w macierzy natężeń na niezawodność grafu

Graf nr 1



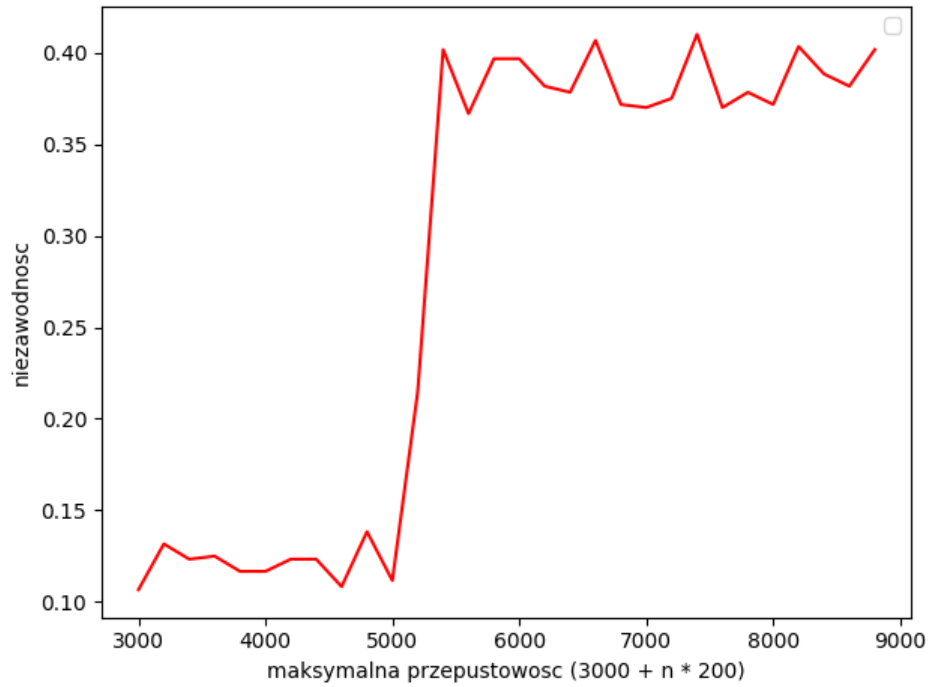
Graf nr 2



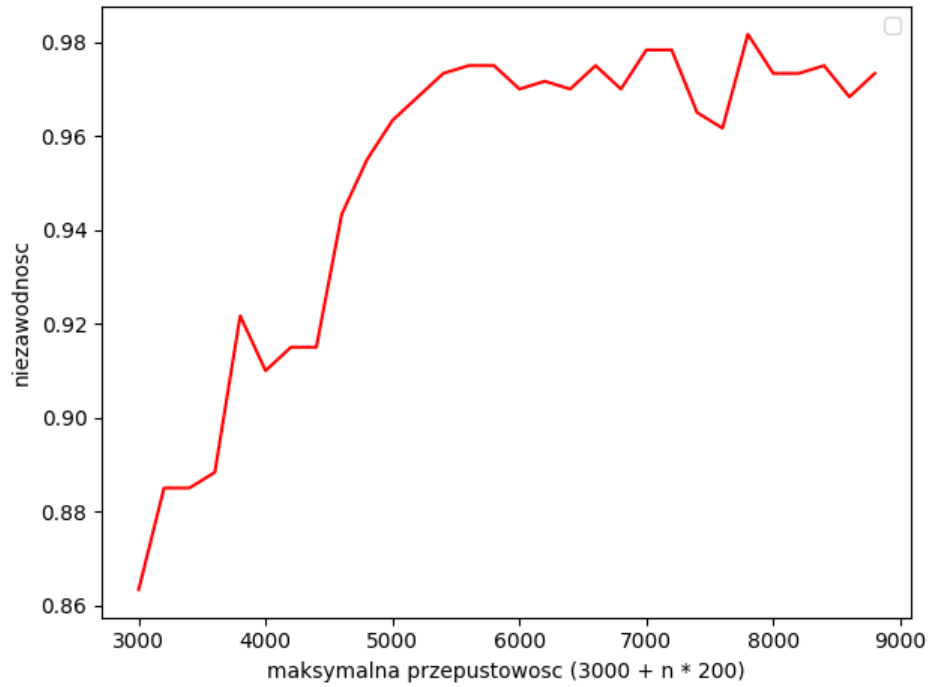
Wnioski: im większa możliwa maksymalna wartość w macierzy natężeń, tym większa szansa, że nasz graf przekroczy wartość maksymalnego przepływu **c**.

Badanie nr 2 – wpływ wartości przepływu na niezawodność grafu

Graf nr 1



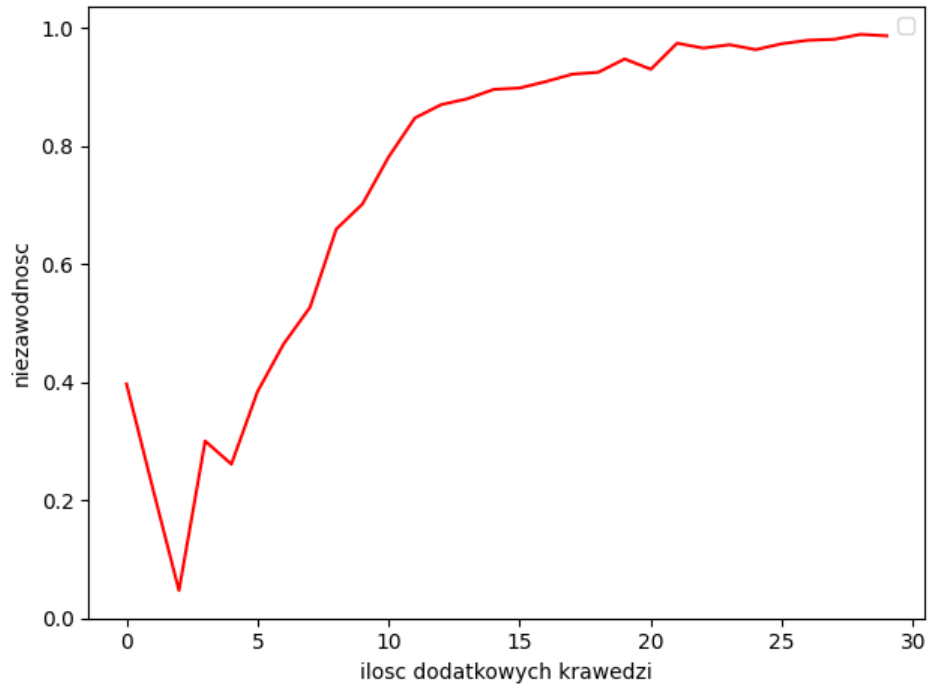
Graf nr 2



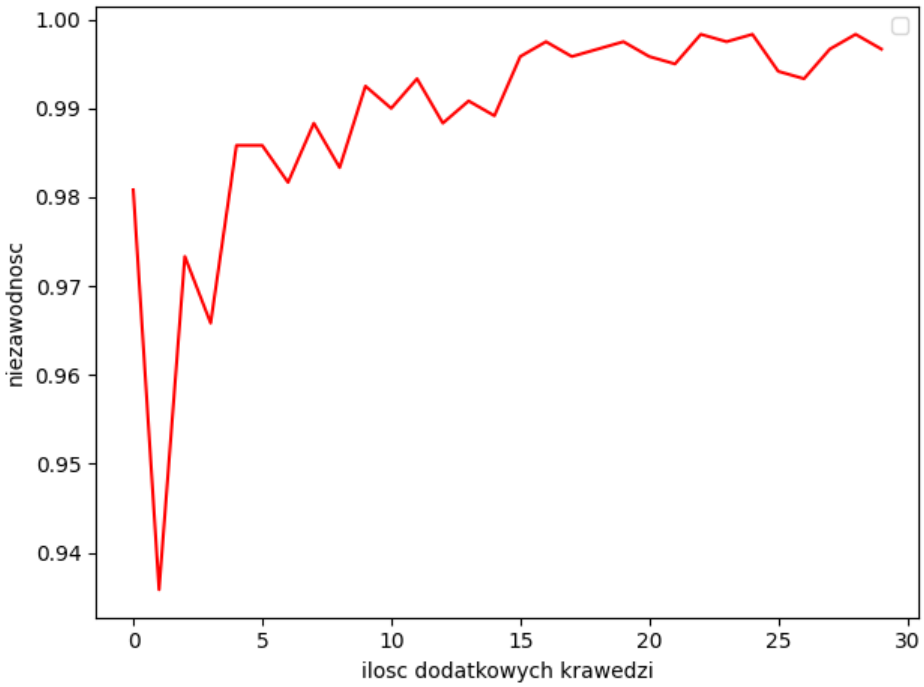
Wnioski: im większa przepustowość krawędzi, tym większa niezawodność naszego grafu.

Badanie nr 3 – wpływ ilości krawędzi na niezawodność grafu

Graf nr 1



Graf nr 2



Wnoski: im większa ilość krawędzi, tym większa niezawodność naszego grafu.

Podsumowanie

Z badań wynika, że na niezawodność grafu ma wpływ przepustowość krawędzi, ilość krawędzi oraz maksymalna możliwa wartość w macierzy natężeń. Możemy te fakty wykorzystać przy tworzeniu sieci.