

# AOD - Laboratorium 4

Michał Waluś 279695

Styczeń 2026

## 1 Zadanie 1

Oznaczmy przez  $c(e)$  pojemność krawędzi, a przez  $f(e)$  przepływ przez nią.

Do rozwiązywania problemu używamy algorytmu Edmondsa-Karpa. Tworzy on dla grafu  $G = (V, E)$  sieć rezydualną  $G_f$ , której wierzchołki są takie same jak w  $G$ , a dla każdej krawędzi  $(u, v)$  z  $G$ , zawiera  $(u, v)$  o pojemności  $c_f((u, v)) = c((u, v)) - f((u, v))$ , a także  $(v, u)$  o pojemności  $f((u, v))$ . Algorytm wykorzystuje przeszukiwanie wszerz (BFS) do znalezienia najkrótszej ścieżki  $p$  ze źródła  $s$  do ujścia  $t$ . Następnie wyznacza maksymalny przepływ  $c_f(p)$  jaki może być na tej ścieżce i dla każdej krawędzi  $(u, v)$  będącej częścią ścieżki zwiększa  $f((u, v))$  o  $c_f(p)$ , jeśli  $(u, v) \in E$  albo zmniejsza  $f((v, u))$ . Pseudokod przedstawiony jako Algorithm 1.

**Function Edmonds-Karp ( $G, s, t$ ):**

```
for each edge  $(u, v) \in G.E$  do
    |  $(u, v).f \leftarrow 0;$ 
end
while there exists a path from  $s$  to  $t$  in residual network  $G_f$  do
    |  $p \leftarrow BFS(G_f, s, t);$ 
    |  $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\};$ 
    for each edge  $(u, v)$  in  $p$  do
        | if  $(u, v) \in E$  then
            | |  $(u, v).f \leftarrow (u, v).f + c_f(p)$ 
        end
        | else
            | |  $(v, u).f \leftarrow (v, u).f + c_f(p)$ 
        end
    end
end
end
```

**Algorithm 1:** Algorytm Edmondsa-Karpa

### 1.1 Złożoność Algorytmu

Niech  $n = 2^k = |V|$ , ponieważ  $G$  to hiper kostka, to  $|E| = nk$ .

Złożoność czasowa przeszukiwania wszerz to  $O(|V| + |E|) = O(nk)$ .

**Twierdzenie** (*Cormen, Leiserson, Rivest, Stein - Introduction to Algorithms*)

Algorytm Edmondsa-Karpa wywołany dla sieci przepływu  $G = (V, E)$  zwiększa przepływ

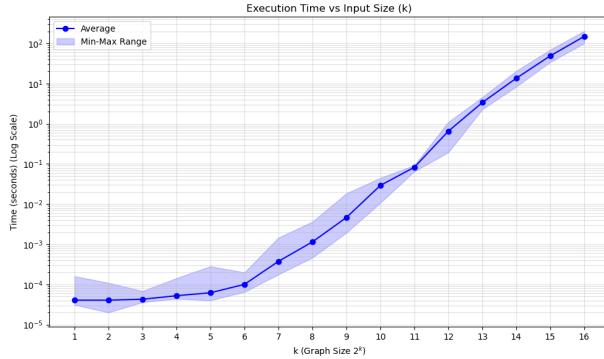
$O(|V||E|)$  razy.

Zatem dla naszego przypadku, przepływ może zostać zwiększyony maksymalnie  $O(n^2k)$  razy, czyli złożoność czasowa całego algorytmu wynosi  $O(n^3k^2) = O(2^{3k}k^2) = O(nm^2)$ .

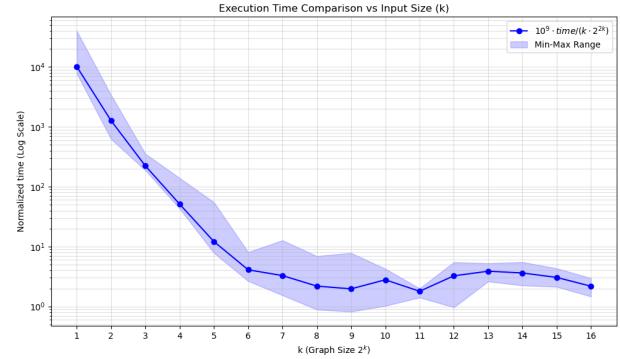
Ponieważ sieć residualna ma co najwyżej dwa razy tyle krawędzi co oryginalna, to złożoność pamięciowa algorytmu wynosi  $O(nk)$ .

## 1.2 Wyniki

Uzyskane wyniki (średnia ze 100 powtórzeń) zostały przedstawione na Rysunkach 1-3.

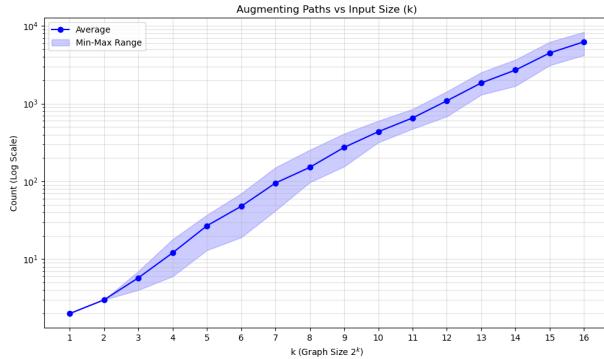


(a) Skala logarytmiczna

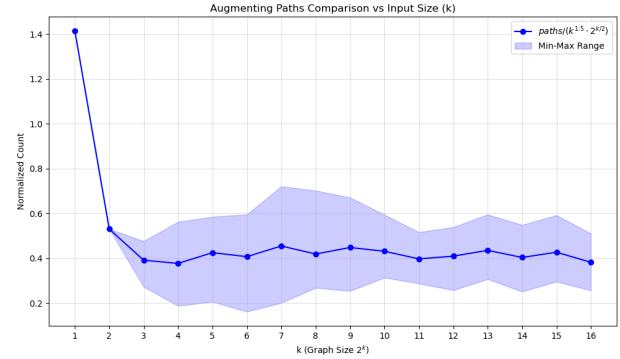


(b)  $10^9 \cdot \text{Czas} / (k \cdot 2^k)$

Rysunek 1: Wyniki dla czasu



(a) Skala logarytmiczna



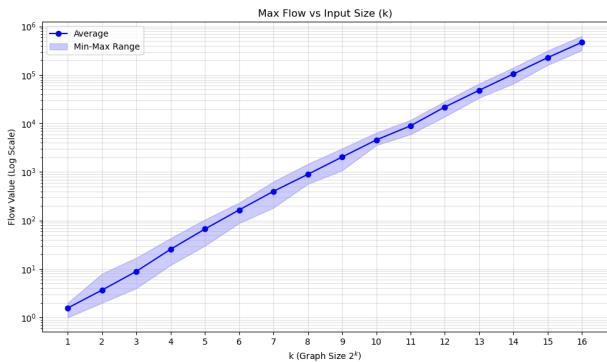
(b) Ścieżki / ( $k^{1.5} \cdot 2^{k/2}$ )

Rysunek 2: Wyniki dla ilości ścieżek

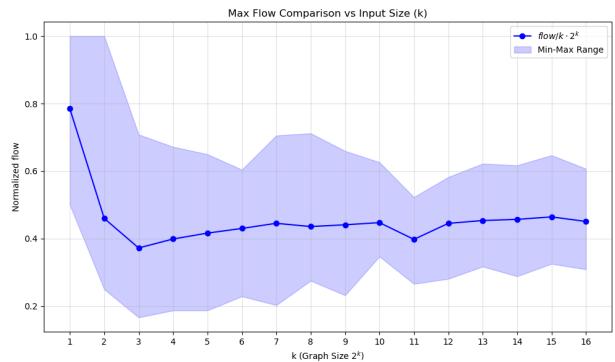
Na wykresach przedstawiono również znormalizowane wyniki, podzielone przez eksperymentalnie wyznaczoną funkcję. Widzimy, że czas wykonania rośnie wolniej niż  $k \cdot 2^{2k}$ , co jest znacznie wolniejszym tempem wzrostu niż to pesymistycznie obliczone podczas analizy algorytmu. Liczba ścieżek zwiększających zdaje się być zbliżona do pierwiastka z liczby krawędzi, a wartość maksymalnego przepływu zdaje się mieć takie samo tempo wzrostu jak liczba krawędzi.

## 2 Zadanie 2

Niech  $G = (V_1 \cup V_2, E)$ ,  $|V_1| = |V_2| = n = 2^k$ ,  $V_1 = \{v_1, \dots, v_n\}$ ,  $V_2 = \{v_{n+1}, \dots, v_{2n}\}$ ,  $(\forall v \in V_1)(\exists w_1, \dots, w_i \in V_2)(v, w_1), \dots, (v, w_i) \in E$ . Dla każdej krawędzi  $e$  ustalamy jej pojemność



(a) Skala logarytmiczna



(b) Przepływ  $/(k \cdot 2^k)$

Rysunek 3: Wyniki dla maksymalnego przepływu

$c(e) = 1$ . Dodatkowo dodajemy do grafu 2 wierzchołki  $v_0, v_{2n+1}$ , takie że

$$(\forall v \in V_1) ((v_0, v) \in E)$$

$$(\forall v \in V_2) ((v, v_{2n+1}) \in E)$$

Wszystkie o pojemności 1. W takim grafie maksymalny przepływ z  $v_0$  do  $v_{2n+1}$  jest równoważny skojarzeniu o największym rozmiarze. Do wyznaczenia rozwiązania możemy znowu użyć algorytmu Edmonda-Karpa.

## 2.1 Złożoność Algorytmu

Złożoności pozostają niezmienione, czyli złożoność pamięciowa algorytmu wynosi  $O(|E|) = (i \cdot 2^k)$ , a czasowa  $O(nm^2) = O(i^2 \cdot 2^{3k})$ .

## 2.2 Wyniki

Czasy działania dla różnych  $i$  w zależności od  $k$  zostały przedstawione na rysunku 4, a na rysunku 5 wielkość maksymalnego skojarzenia w zależności od  $i$ .

Czas działania zwiększa się wraz ze wzrostem wartości  $i$  oraz  $k$ , wykres sugeruje wzrost wykładniczy względem  $k$ , czyli wielomianowy względem rozmiaru grafu.

Wraz ze wzrostem  $i$ , zwiększa się rozmiar skojarzenia o największym rozmiarze (przypomina on logarytmiczny), przy czym od  $i = 6$  lub  $7$ , skojarzenia zawierają wszystkie wierzchołki w grafie i są maksymalne.

## 3 Zadanie 3

Modele programowania liniowego dla zadania pierwszego są w następującej formie:

- Parametry:

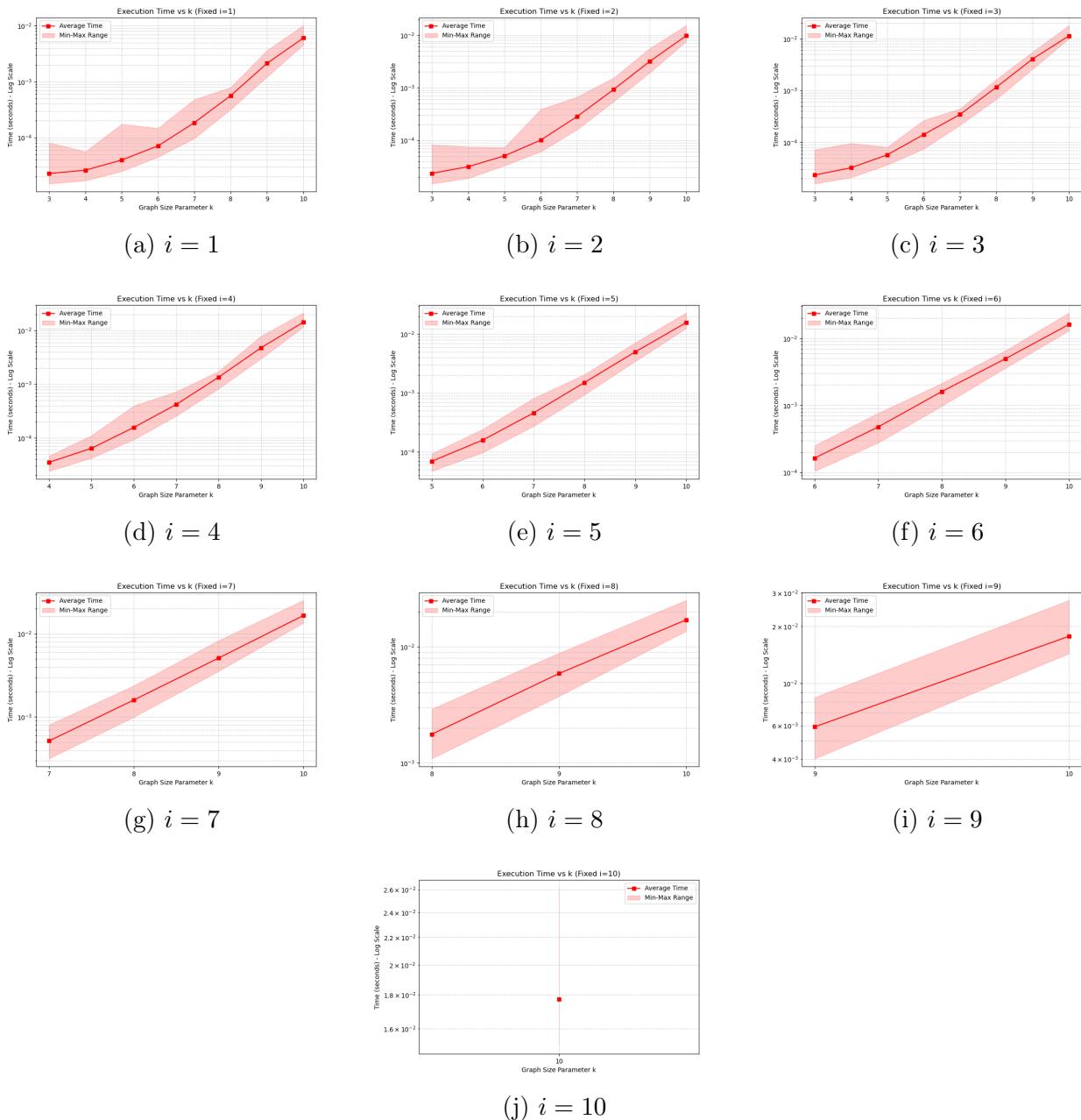
$$V, E \subseteq V \times V, (\forall e \in E) C(e) \geq 0, s, t \in V$$

- Zmienne decyzyjne:

$$(\forall e \in E) 0 \leq flow(e) \leq C(e)$$

- Ograniczenia:

$$(\forall u \in V \setminus \{s, t\}) \sum_{(u,v) \in E} flow((u, v)) = \sum_{(v,u) \in E} flow((v, u))$$



Rysunek 4: Czasy działań w zależności od  $k$

- Funkcja celu:

$$\max \sum_{(s,v) \in E} \text{flow}((s,v))$$

Modele programowania liniowego dla zadania drugiego są w następującej formie:

- Parametry:

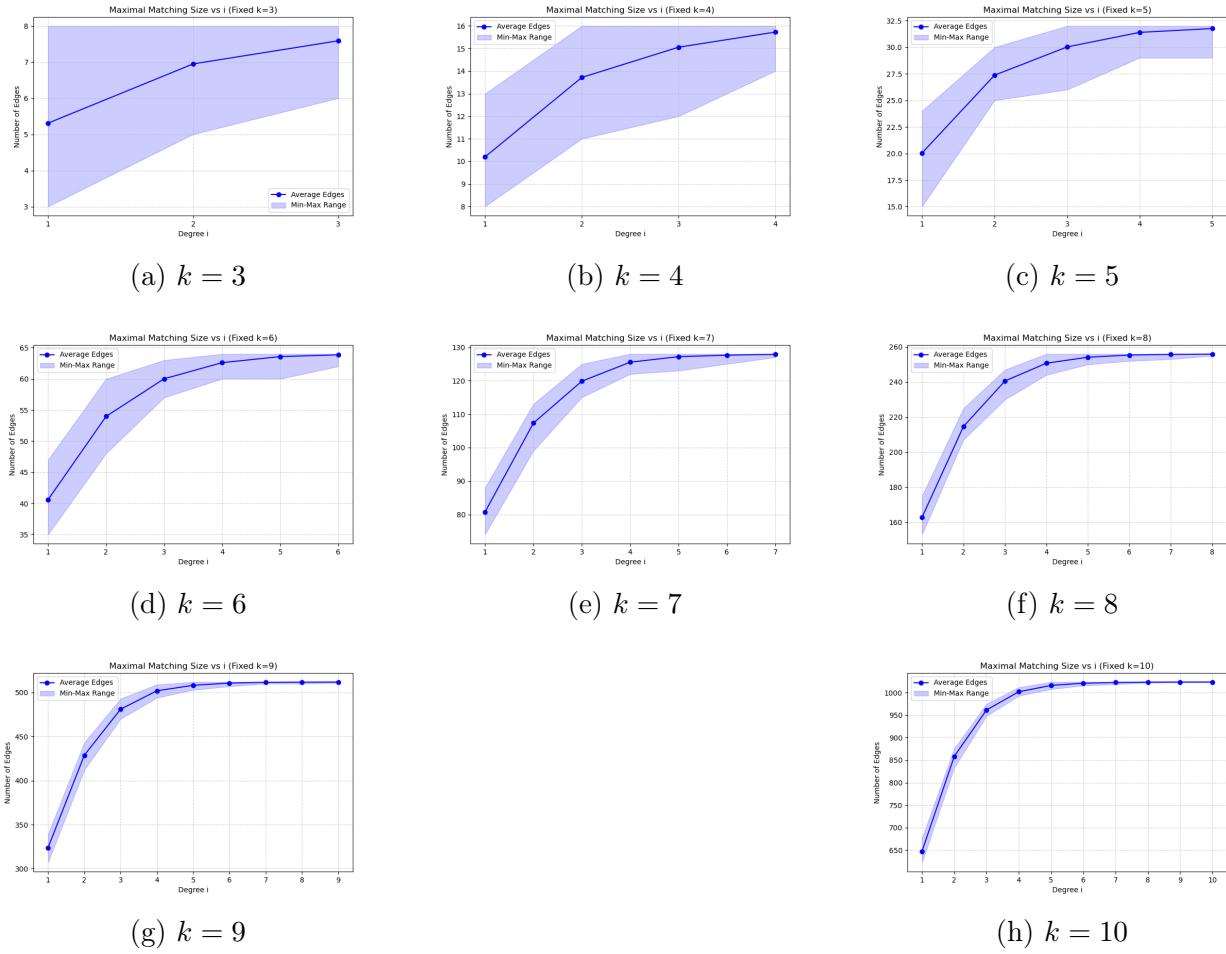
$$V, E \subseteq V \times V, s, t \in V$$

- Zmienne decyzyjne:

$$(\forall e \in E) 0 \leq \text{flow}(e) \leq 1$$

- Ograniczenia:

$$(\forall u \in V \setminus \{s, t\}) \sum_{(u,v) \in E} \text{flow}((u,v)) = \sum_{(v,u) \in E} \text{flow}((v,u))$$



Rysunek 5: Wielkość maksymalnego skojarzenia w zależności od  $i$

- Funkcja celu:

$$\max \sum_{(s,v) \in E} \text{flow}((s,v))$$

Programy z zadań 1 i 2 obliczają szybciej niż solver GLPK.

## 4 Zadanie 4

Algorytm Dinica jest usprawnieniem metody Edmondsa-Karpa i również utrzymuje on sieć rezydualną  $G_f$ . Różnicą jest że algorytm działa w fazach i zamiast wyszukiwać pojedynczą ścieżkę, znajduje wszystkie ścieżki tej samej długości. W każdej fazie najpierw za pomocą przeszukiwania wszerz (BFS) konstruowana jest sieć warstwowa  $G_L$ . W sieci tej wierzchołki mają przypisany poziom  $\text{level}(v)$  równy odległości od źródła  $s$  w sieci rezydualnej, a krawędzie to tylko te łuki  $(u, v)$  z  $G_f$ , dla których  $\text{level}(v) = \text{level}(u) + 1$ .

W tak skonstruowanej sieci warstwowej szukamy przepływu blokującego przy użyciu przeszukiwania w głąb (DFS). Przepływ blokujący to taki przepływ, po którego nasyceniu nie istnieje już żadna ścieżka powiększająca w  $G_L$ , czyli wszystkie pozostałe ścieżki z  $s$  do  $t$  muszą być dłuższe. Algorytm powtarza te dwie fazy (budowa  $G_L$  i nasycanie), dopóki  $t$  jest osiągalne z  $s$  w sieci rezydualnej. Ponieważ DFS przechodzi po sąsiadach danego wierzchołka  $u$  w tej samej kolejności za każdym razem gdy go napotka, to zapamiętujemy, z których wierzchołków nie ma ścieżki do  $t$  (w grafie  $G_L$ ). Pseudokod przedstawiony jako Algorithm 3.

```

Function BFS(s, t):
    level // tablica długości n zawierająca same -1
    level[s]  $\leftarrow 0$ ;
    Q  $\leftarrow s$  // Kolejka
    while Q niepusta do
        u  $\leftarrow Q.pop()$ ;
        for v  $\in N(u)$  do
            if level[v] = -1 i  $C(u, v) - flow(u, v) > 0$  then
                level[v]  $\leftarrow level[u] + 1$ ;
                Q.push(v);
            end
        end
    end
    Return level[t]  $\neq -1$ 
end

Function Dinic (G, s, t):
    for each edge (u, v)  $\in G.E$  do
        | (u, v).f  $\leftarrow 0$ ;
    end
    while BFS(Gf, s, t) znajduje ścieżkę do
        // Faza: Budowa sieci warstwowej i szukanie potoku blokującego
        while (p, flow)  $\leftarrow DFS(G_f, s, t, \infty)$  do
            for each edge (u, v) in p do
                if (u, v)  $\in E$  then
                    | (u, v).f  $\leftarrow (u, v).f + flow$ ;
                end
                else
                    | (v, u).f  $\leftarrow (v, u).f - flow$ ;
                end
            end
        end
    end
end

```

**Algorithm 2:** Algorytm Dinica

## 4.1 Złożoność Algorytmu

Podobnie jak wcześniej niech  $n = 2^k = |V|$  oraz  $|E| = nk$ .

Pojedyncza faza algorytmu Dinica składa się z wyznaczenia sieci warstwowej (BFS) w czasie  $O(|E|)$  oraz znalezienia potoku blokującego (wielokrotny DFS). Dzięki technice usuwania naświetnych krawędzi (lub pamiętania wskaźnika na ostatnio odwiedzoną krawędź), znalezienie potoku blokującego zajmuje  $O(|V||E|)$ .

*Lemat 3 z portalu Ważniak MIMUW*

W algorytmie Dinica liczba faz wynosi co najwyżej  $|V| - 1$ , ponieważ w każdej kolejnej fazie odległość w sieci rezydualnej od źródła do ujścia ściśle rośnie.

Zatem ogólna złożoność algorytmu dla dowolnych pojemności wynosi  $O(|V|^2|E|)$ . Dla naszego przypadku hiperkostki, gdzie  $|V| = n$  i  $|E| = nk$ , złożoność wynosi:

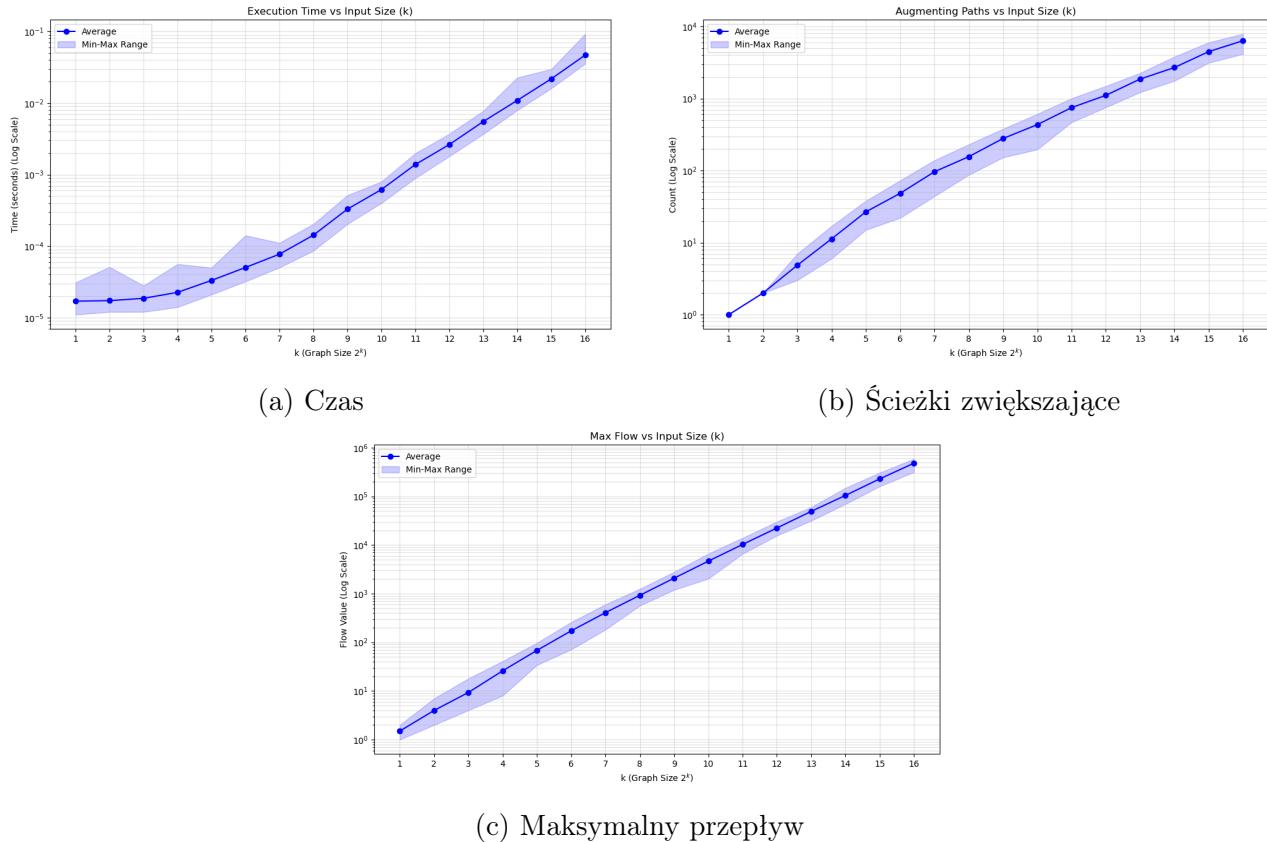
$$O(n^2 \cdot nk) = O(n^3k) = O(2^{3k}k)$$

Jest to wynik lepszy od algorytmu Edmondsa-Karpa o czynnik  $k$  (lub  $O(n/m)$  w ogólnym przypadku grafów gęstych).

Złożoność pamięciowa pozostaje  $O(nk)$ , gdyż przechowujemy jedynie strukturę grafu, poziomy wierzchołków oraz wskaźniki dla procedury DFS.

## 4.2 Wyniki

Otrzymane wyniki są przedstawione na rysunku 6. Ścieżki zwiększające oraz maksymalny przepływ pozostają bez większych zmian w stosunku do algorytmu Edmondsa-Karpa, ale czas wykonania jest zdecydowanie szybszy, dla  $k = 16$  algorytm wykonał się ponad 1000 razy szybciej.



Rysunek 6: Wyniki dla Algorytmu Dinica

## 5 Wnioski

Mimo tego, że problem maksymalnego przepływu jest bardzo ogólny (jedno zastosowanie znajdywania skojarzeń w Zadaniu 2), to istnieją dość proste i efektywne algorytmy rozwiązujące go w czasie wielomianowym. Przy dobrej implementacji, algorytm dla grafu jak w zadaniu 4, z ponad pół miliona wierzchołków wykonuje się w czasie poniżej 0.1s, czyli możliwe jest rozwiązanie bardzo dużych problemów nawet na zwykłych komputerach. Pokazuje to że nie wszystkie ogólne problemy muszą być skompilkowane i mieć rozwiązania w czasie wykładniczym.