Algorytmy optymalizacji dyskretnej - Lista 4

Wojciech Sęk

30 grudnia 2021

1 Algorytmy

1.1 Rozważane grafy

Oznaczmy przez $e_k(i)$ k-bitową reprezentację liczby i i przez $H(i)$ wagę Hamminga liczby i. Algorytmy szukają największego przepływu na hiperkostkach $H_k = (N_k, A_k)$ o wymiarach $k \in \{1, \ldots, 16\}$, gdzie:

$$N_k = \{0, \ldots, 2^k - 1\}$$
$$A_k = \{(i, j) \in N_k \times N_k : H(e(j)) = H(e(i)) + 1\}$$

1.2 Pojęcia

1.2.1 Przepustowość residualna

Rozważamy sieć $G = (V, E)$ ze źródłem s i ujściem t, niech f to pewien przepływ w sieci G. Przepustowością residualną na krawędzi (u, v) nazywamy:

$$c_f(u, v) = c_{u,v} - f_{u,v}$$

1.2.2 Sieć residualna

Siecią residualną grafu $G = (V, E)$ i przepływu f nazywamy:

$$G_f = (V, \{(u, v) \in V \times V : c_f(u, v) > 0\})$$

czyli jest to sieć, która reprezentuje krawędzie, po których możemy jeszcze przesyłać.

1.2.3 Ścieżka powiększająca

Dla sieci $G = (V, E)$ i przepływu f ścieżką powiększającą p nazywamy każdą ścieżkę ze źródła s do ujścia t w sieci residualnej G_f.

1.2.4 Przepustowość residualna

Przepustowością residualną ścieżki p jest $c_f(p) = \min\{c_f(u, v) : (u, v) \text{ jest na } p\}$

1.2.5 Prawidłowa funkcja odległości

Prawidłową funkcją odległości $d : N \to \mathbb{N}$ sieci residualnej jest funkcja spełniająca warunki:

$$d(t) = 0 \wedge (\forall (i, j) \in G_f)(d(i) \leqslant d(j) + 1)$$

1.2.6 Dopuszczalne łuki i ścieżki

Dopuszczalny łuk to łuk spełniający $f_{i,j} < c_{i,j} \wedge d(i) = d(j) + 1$.
Dopuszczalna ścieżka to ścieżka składająca się z dopuszczalnych łuków.

1.2.7 Liczba wierzchołków i liczba krawędzi

Dla sieci $G = (N, A)$ oznaczamy $|N|$ przez n oraz $|A|$ przez m.
Zauważmy, że dla rozważnych hiperkostek wymiaru k mamy dokładnie $n = 2^k$ oraz $m = 2^{k-1}k$.
Innymi słowy mamy $m = \frac{n}{2} \log n = O(n \log n)$.

1.3 Algorytm Edmondsa-Karpa

1.3.1 Implementacja

Algorytm Edmondsa-Karpa jest implementacją metody Forda-Fulkersona, która w ogólności ma postać:

function FORD-FULKERSON(G, s, t)
  for $(u, v) \in A[G]$ **do**
    $f_{u,v} \leftarrow 0$
    $f_{v,u} \leftarrow 0$
  end for
  while istnieje ścieżka p z s do t w sieci residualnej G_f **do**
    $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \in p\}$
    for $(u, v) \in p$ **do**
      $f_{u,v} \leftarrow f_{u,v} + c_f(p)$
      $f_{v,u} \leftarrow -f_{u,v}$
    end for
  end while
end function

Algorytm Edmondsa-Karpa mówi nam o implementacji kroku, w którym szukamy ścieżki powiększającej. Mianowicie, w sieci residualnej przeszukujemy wszerz od wierzchołka s i jeżeli wierzchołek t jest osiągalny to rozważamy daną ścieżkę. Gdy znajdziemy ścieżkę, szukamy najmniejszej jej przepustowości, a następnie powiększamy przepływ o tę wartość na całej ścieżce. W ten sposób, przepływ zwiększa się maksymalnie i co najmniej jedna krawędź jest wykluczana z sieci residualnej (krawędzie usuwane nazywamy krytycznymi).

1.3.2 Złożoność

Początkowo zerujemy przepływ w sieci ze złożonością $O(m)$. Następnie szukamy ścieżki w sieci residualnej przez przeszukiwanie wszerz, czyli ze złożonością $O(n+m)$. W ogólności możemy też pokazać, że łączna liczba powiększeń w tym algorytmie wynosi $O(nm)$, ponieważ każda krawędź może być krytyczna co najwyżej $O(n)$ razy, bo od momentu, w którym jest krytyczna, musi zwiększyć się jej odległość od źródła o co najmniej 2 do momentu, gdy znowu będzie krytyczna. Wszystkich krawędzi w grafie residualnym może być m, więc wszystkich powiększeń może być $O(nm)$. Złożoność przypisań nowych wartości dla przepływu f wynosi $O(m)$ z racji liczby krawędzi na ścieżce, zatem ostatecznie złożoność algorytmu wynosi:

$$O(nm \cdot (n + m + m)) = O(n^2 m + nm^2) = O(nm^2)$$

1.4 Model LP

1.4.1 Opis

Model został zaimplementowany w języku Julia z użyciem biblioteki GLPK i JuMP. Dla kostki k wymiarowej $H_k = (N_k, A_k)$ z przepustowościami $c_{i,j}$ na łukach $(i, j) \in A_k$ mamy następujący model:

$$\begin{aligned}
\max \quad & \sum_{(1,j)\in A_k} x_{1,j} \\
\text{Subject to} \quad & (\forall (i,j) \in A_k)(x_{i,j} \in \mathbb{N}) \\
& (\forall (i,j) \in A_k)(x_{i,j} \leqslant c_{i,j}) \\
& (\forall (i,j) \in (N_k \times N_k) \setminus A_k)(x_{i,j} = 0) \\
& \sum_{(i,2^k-1)\in A_k} x_{i,2^k-1} = \sum_{(0,j)\in A_k} x_{0,j} \\
& (\forall l \in [1, \dots, 2^k - 2]) \left(\sum_{(i,l)\in A_k} x_{i,l} = \sum_{(l,j)\in A_k} x_{l,j} \right)
\end{aligned}$$

1.5 Shortest Augmenting Path

1.5.1 Implementacja

Shortest augmenting path ma podobną budowę do algorytmu Edmondsa-Karpa, ale wykorzystuje fakt, że długość najkrótszej ścieżki od dowolnego węzła do ujścia t jest niemalejąca względem powiększania o przepustowość ścieżki powiększającej.

```
function SHORTEST-AUGMENTING-PATH(G, s, t)
    for (u, v) ∈ A[G] do
        f_{u,v} ← 0
        f_{v,u} ← 0
    end for
    dla i ∈ N wylicz odległości d(i) od ujścia t
    i ← s
    while d(s) < n do
        if i ma dopuszczalny łuk then
            (i, j) ← pewien dopuszczalny łuk
            pre(j) ← i
            i ← j
            if i = t then
                stwórz ścieżkę p na podstawie tablicy pre
                δ ← min{c_{i,j} − f_{i,j} : (i, j) ∈ p}
                powiększ przepływ na ścieżce o δ
                i ← s
            end if
        else
            d(i) ← min{d(j) + 1 : (i, j) ∈ A ∧ f_{i,j} < c_{i,j}}
            if i ≠ s then
                i ← pre(i)
            end if
        end if
    end while
end function
```

Można pokazać, że gdy $d(s) \geqslant n$ to sieć residualna nie posiada ścieżki skierowanej od s do t, każda dopuszczalna ścieżka jest najkrótszą ścieżką powiększającą oraz, że algorytm Shortest Augmenting Path zachowuje poprawność funkcji odległości w każdym kroku. Z tych trzech własności wynika, że po zakończeniu działania, czyli gdy $d(s) \geqslant n$ algorytm policzy maksymalny przepływ.

1.5.2 Złożoność

Skorzystamy z następujących faktów:

1) Jeżeli algorytm zmienia wartość d dowolnemu węzłowi co najwyżej k razy, to czas spędzony na szukaniu dopuszczalnego łuku i zmiany wartości d wynosi $O(km)$.

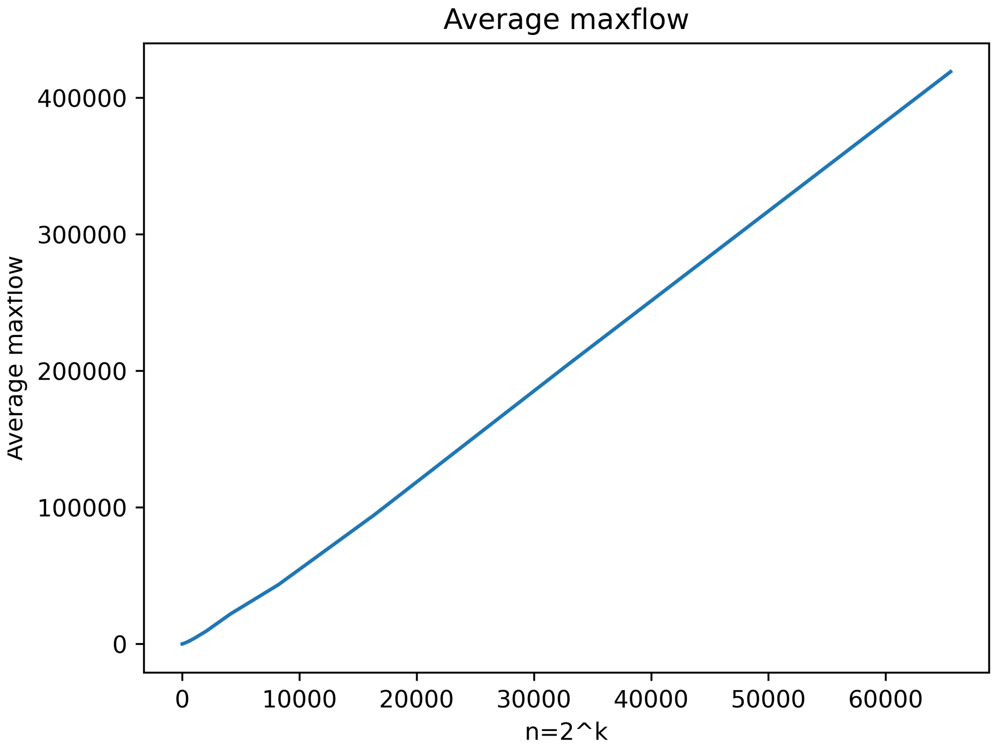
2) Jeżeli algorytm zmienia wartość d dowolnemu węzłowi co najwyżej k razy, to algorytm usuwa krawędzie krytyczne z sieci residualnej co najwyżej $\frac{km}{2}$.

3) W algorytmie Shortest Augmenting Path każda wartość $d(i)$ jest zmieniana co najwyżej n razy, więc liczba operacji zmiany d wynosi co najwyżej n^2.

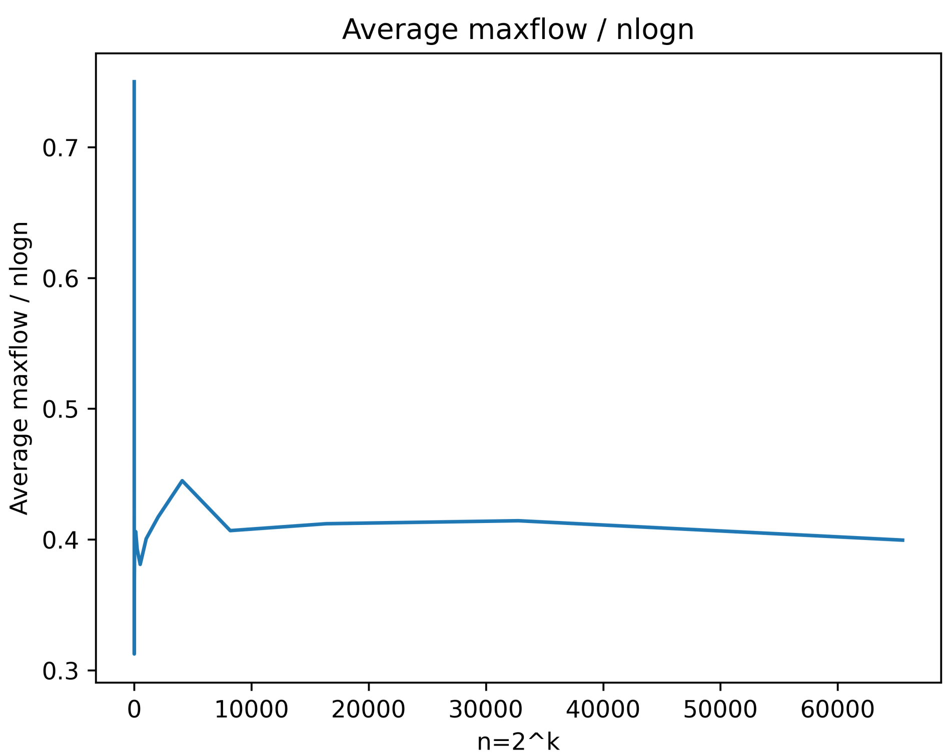
4) Operacji poszerzania przepływu jest co najwyżej $\frac{nm}{2}$.

Z powyższych własności mamy, że czas na poszukiwanie dopuszczalnych łuków i zmiany d wynosi $O(nm)$. Każda operacja poszerzania przepływu wymaga $O(n)$ czasu, więc wszystkie te operacje wymagają $O(n^2m)$ czasu. Każda dopuszczalna ścieżka jest długości co najwyżej n, więc algorytm wymaga $O(n^2 + n^2m)$ operacji przedłużania aktualnie znalezionej ścieżki, bo możemy co najwyżej n razy zmieniać wartości $d(i)$ i co najwyżej $\frac{nm}{2}$ razy dokonywać poszerzania przepływu. Ostatecznie złożoność czasowa wynosi $O(n^2m)$.

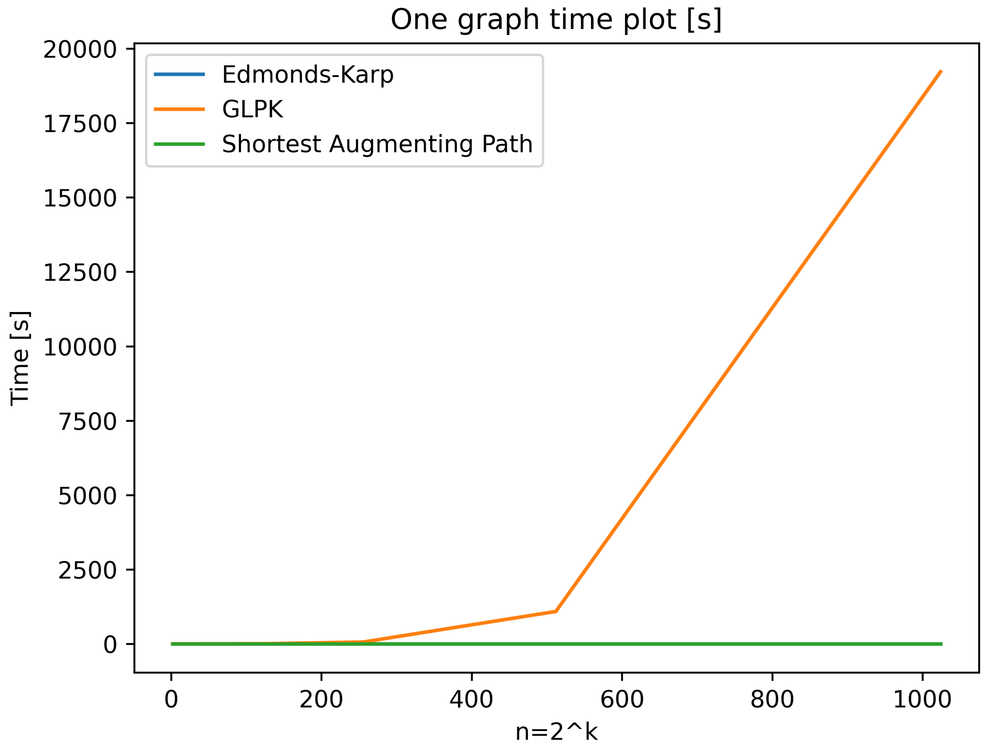
2 Wyniki eksperymentów



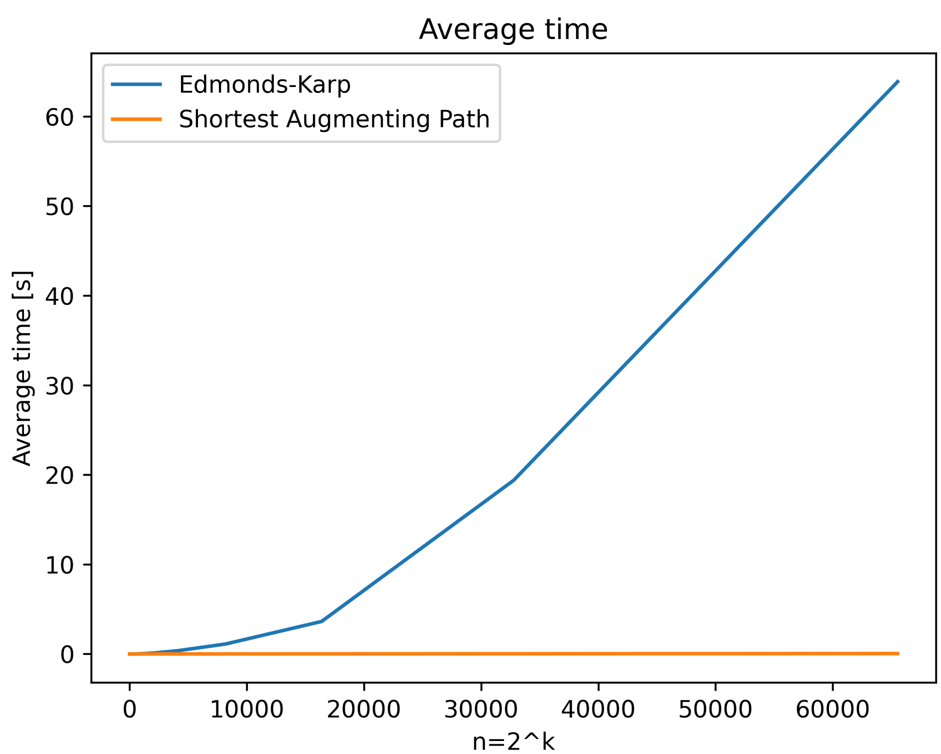
Rysunek 1: Średni maksymalny przepływ w zależności od n



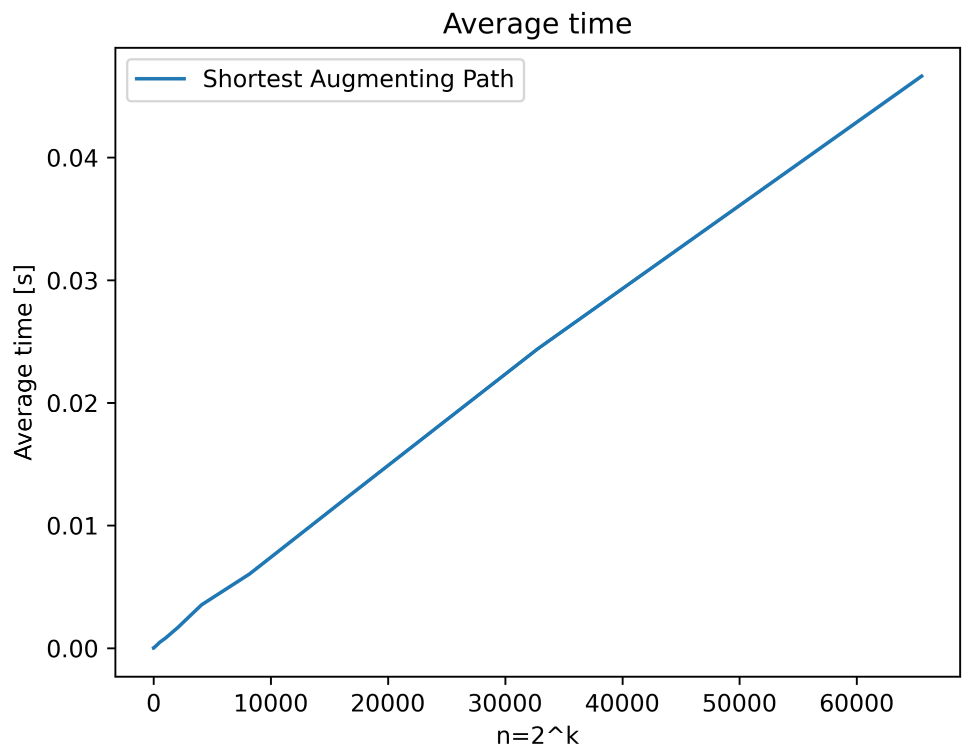
Rysunek 2: Średni maksymalny przepływ podzielony przez $n \log n$ w zależności od n



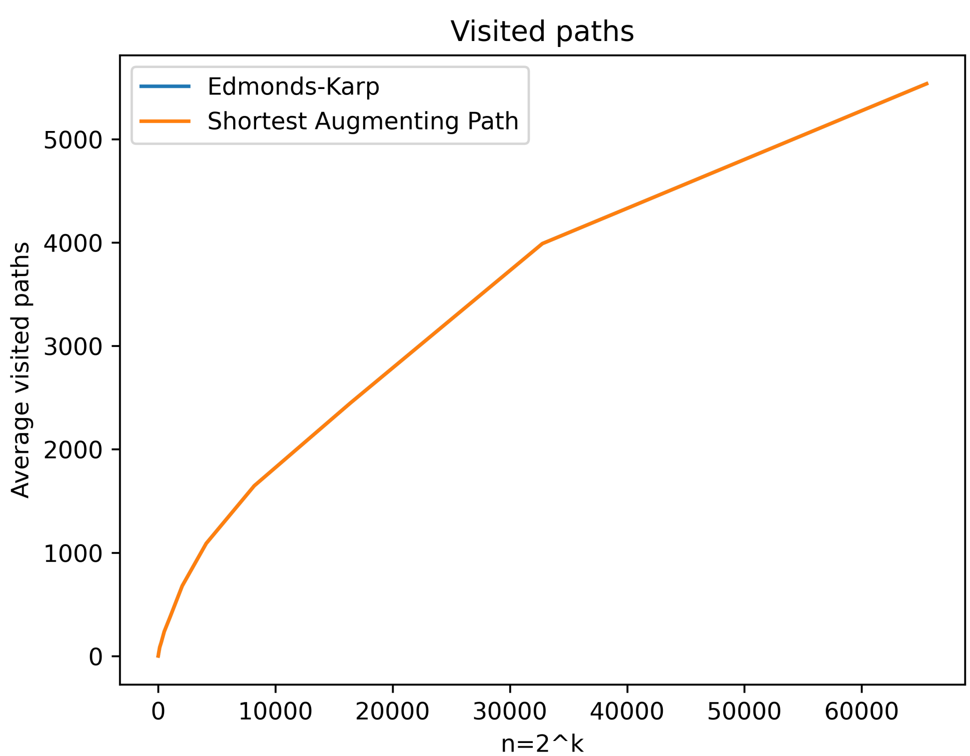
Rysunek 3: Czas działania dla losowej hiperkostki wymiaru od 1 do 10 dla algorytmów i modelu LP



Rysunek 4: Średni czas działania dla algorytmów



Rysunek 5: Średni czas działania dla algorytmu Shortest Augmenting Path



Rysunek 6: Średnia liczba wyznaczanych ścieżek

3 Intepretacja i wnioski

3.1 Średni maksymalny przepływ

Wynik eksperymentu jest konsekwencją budowy grafu. Rozważmy łuk (u, v). Wartością oczekiwaną przepustowości tego łuku jest 2^{l-1}, gdzie l to maximum z liczby jedynek i liczby zer w binarnej reprezentacji u i v. Jeżeli narysujemy graf warstwami, tak że w danej warstwie są węzły z kolejnymi wartościami wagi Hamminga, to na brzegowych warstwach mamy największe średnie przepustowości, natomiast w środkowych warstwach małe, ale za to jest więcej węzłów o takich wartościach wagi Hamminga. Zatem średnio przesyłamy tyle ile można wysłać od źródła 0, czyli $2^{k-1}k$, co daje dokładnie $\frac{1}{2}n \log n$. Eksperyment potwierdził, że średnio ten współczynnik jest bliski $\frac{1}{2}$, ale delikatnie mniejszy.

3.2 Średni czas

Czas rozwiązywania modelu przez solver $GLPK$ był absurdalnie wielki, przez co eksperymenty były przeprowadzone tylko dla hiperkostek o wymiarze co najwyżej 10.
Natomiast algorytm Edmondsa-Karpa był znacznie większy od Shortest Augmenting Path i osiągnął jest zdecydowanie złożony bardziej niż $O(n)$. Mimo teoretycznej złożoności $O(n^2m)$ w praktyce Shortest Augmenting Path osiąga średnio liniową złożoność.
Złożoność algorytmu Edmondsa-Karpa wynika z tego, że ścieżki są znajdowane przez przeszukiwanie wszerz, co w zadanej strukturze grafu daje, że musimy przejrzeć wszystkie osiągalne wierzchołki od 0 do warstwy i wszystkie ich krawędzie, a to przeglądanie praktycznie całego grafu (dla grafu residualnego ze wszystkimi krawędziami przejrzymy optymistycznie aż $m - k + 1$ krawędzi i n wierzchołków).
Shortest Augmenting Path natomiast przeszukuje wgłąb i optymistycznie może przejrzeć zaledwie $k + 1$ wierzchołków i k krawędzi.

3.3 Liczba wyznaczanych ścieżek

Liczba wyznaczanych ścieżek ma jest $o(n)$ i co więcej pokrywa się dla algorytmu Edmondsa-Karpa oraz dla Shortest Augmenting Path. Oba algorytmy przeglądają w tej samej kolejności krawędzie wychodzące od danego węzła (zamieniają zera na jedynki w zapisie bitowym zaczynając od prawej strony), więc znajdują te same przepływy.