Sprawozdanie z Laboratorium 4

Kajetan Plewa

Styczeń 2025

1 Zadanie 1: Algorytm Edmondsa-Karpa w hiperkostce

1.1 Opis implementacji

Celem zadania było wyznaczenie maksymalnego przepływu w k-wymiarowej skierowanej hiperkostce H_k przy użyciu algorytmu Edmondsa-Karpa. Implementacja została wykonana w języku C++ i składa się z trzech głównych komponentów:

- **Struktura grafu:** Graf reprezentowany jest przy użyciu listy sąsiedztwa. Każda krawędź (`Edge`) przechowuje wierzchołek docelowy, pojemność (c_{uv}), aktualny przepływ (f_{uv}) oraz indeks krawędzi powrotnej (`rev`) w sieci residualnej, co pozwala na aktualizację przepływu w czasie stałym.
- **Budowa hiperkostki:** Wierzchołki są identyfikowane przez liczby od 0 do $2^k - 1$. Krawędź (i, j) tworzona jest wtedy i tylko wtedy, gdy reprezentacje binarne i oraz j różnią się na dokładnie jednej pozycji, a waga Hamminga $H(j) = H(i) + 1$[cite: 10, 12].
- **Pojemności krawędzi:** Zgodnie z treścią zadania, pojemności są losowane z rozkładu jednostajnego ze zbioru $\{1, \ldots, 2^l\}$, gdzie $l = \max(H(i), Z(i), H(j), Z(j))$, a $Z(x)$ to liczba zer w zapisie binarnym[cite: 14, 15].

1.2 Opis działania algorytmu

Kluczowe aspekty działania zaimplementowanego algorytmu to:

1. **Sieć residualna i krawędzie powrotne:** Dla każdej krawędzi (u, v) o pojemności $c(u, v)$, algorytm przechowuje aktualny przepływ $f(u, v)$. W sieci residualnej istnieją dwa rodzaje krawędzi:
   - Krawędź wprost (u, v) z przepustowością residualną $r(u, v) = c(u, v) - f(u, v)$.
   - Krawędź wsteczna (v, u) z przepustowością residualną $r(v, u) = f(u, v)$, która pozwala na "wycofanie"wcześniej przesłanego przepływu.

   W kodzie zostało to zrealizowane poprzez pole `rev`, które wiąże krawędź z jej lustrzanym odbiciem w liście sąsiedztwa drugiego wierzchołka.
2. **Wybór najkrótszej ścieżki (BFS):** W każdej iteracji algorytm wykonuje przeszukiwanie wszerz (BFS), startując z wierzchołka $s = 0$. BFS znajduje ścieżkę z s do $t = 2^k - 1$ o minimalnej liczbie krawędzi w sieci residualnej G_f.
3. **Aktualizacja przepływu:** Po znalezieniu ścieżki powiększającej p, wyznaczana jest minimalna przepustowość residualna na tej ścieżce:

   $$c_f(p) = \min\{r(u, v) : (u, v) \in p\} \tag{1}$$

   Następnie przepływ wzdłuż każdej krawędzi ścieżki p jest zwiększany o $c_f(p)$, a przepływ na odpowiadających im krawędziach wstecznych jest zmniejszany o tę samą wartość.
4. **Warunek stopu:** Algorytm kończy działanie, gdy w sieci residualnej nie istnieje już żadna ścieżka z s do t o dodatniej przepustowości residualnej. Zgodnie z twierdzeniem o maksymalnym przepływie i minimalnym przekroju, wyznaczony przepływ jest wtedy maksymalny.

1.3 Analiza złożoności obliczeniowej

Złożoność algorytmu Edmondsa-Karpa wyraża się wzorem $O(V \cdot E^2)$, gdzie V to liczba wierzchołków, a E to liczba krawędzi. W przypadku k-wymiarowej hiperkostki parametry te wynoszą:

- $V = 2^k$
- $E = k \cdot 2^{k-1}$

Podstawiając te wartości do ogólnego wzoru na złożoność:

$$O(V \cdot E^2) = O(2^k \cdot (k \cdot 2^{k-1})^2) \tag{2}$$

Rozpisując potęgi:

$$O(2^k \cdot k^2 \cdot 2^{2k-2}) = O(k^2 \cdot 2^{3k-2}) = O(k^2 \cdot 2^{3k}) \tag{3}$$

1.3.1 Złożoność pamięciowa

Pamięć zajmowana przez graf zależy od liczby wierzchołków i krawędzi:

$$O(V + E) = O(2^k + k \cdot 2^{k-1}) = O(k \cdot 2^k) \tag{4}$$

2 Zadanie 2: Maksymalne skojarzenie w losowym grafie dwudzielnym

2.1 Opis implementacji i generowania grafu

Zadanie polegało na wyznaczeniu skojarzenia o największym rozmiarze w nieskierowanym dwudzielnym grafie losowym $G = (V_1 \cup V_2, E)$.

- **Struktura zbiorów:** Oba podzbiory wierzchołków V_1 i V_2 mają równą liczność $|V_1| = |V_2| = 2^k]$.
- **Generowanie krawędzi:** Dla każdego wierzchołka $u \in V_1$ losowanych jest niezależnie i sąsiadów ze zbioru V_2 z rozkładem jednostajnym.
- **Transformacja do problemu przepływu:** Aby wyznaczyć maksymalne skojarzenie, graf nieskierowany transformowany jest w sieć przepływową:
  1. Dodawane jest super-źródło s połączone ze wszystkimi wierzchołkami $u \in V_1$.
  2. Dodawane jest super-ujście t połączone ze wszystkimi wierzchołkami $v \in V_2$.
  3. Wszystkie krawędzie (zarówno te od s, jak i te między V_1 a V_2 oraz do t) otrzymują jednostkową przepustowość $c(e) = 1$.

2.2 Algorytm wyznaczania skojarzenia

Wartość maksymalnego przepływu w tak skonstruowanej sieci jest równa liczności maksymalnego skojarzenia w grafie dwudzielnym.

- **Zastosowana metoda:** Wykorzystano ponownie algorytm Edmondsa-Karpa. W przypadku sieci o jednostkowych pojemnościach, każda ścieżka powiększająca zwiększa przepływ o dokładnie 1 jednostkę.

2.3 Analiza złożoności obliczeniowej

Parametry grafu dla zadania 2 wynoszą:

- Liczba wierzchołków w sieci: $V = 2 \cdot 2^k + 2$ (wierzchołki V_1, V_2 oraz s i t).
- Liczba krawędzi w sieci: $E = 2^k(\text{do } s) + i \cdot 2^k(\text{między } V_1, V_2) + 2^k(\text{do } t) = (i+2)2^k$.

Dla algorytmu Edmondsa-Karpa w ogólnym przypadku złożoność to $O(VE^2)$.
Zatem złożoność w tym konkretnym przypadku wynosi:

$$O(2^k \cdot (V + E)) = O(2^k \cdot (2 \cdot 2^k + (i+2)2^k)) = O(i \cdot 2^{2k}) \tag{5}$$

3 Zadanie 3: Modelowanie problemów za pomocą programowania liniowego (LP)

3.1 Transformacja problemów do postaci modelu LP

Zgodnie z wymaganiami zadania, zaimplementowano generator modeli w języku GNU MathProg.

3.1.1 Model dla maksymalnego przepływu (Zadanie 1)

Problem maksymalnego przepływu w hiperkostce H_k można sformułować jako zadanie programowania liniowego, gdzie zmiennymi decyzyjnymi są wielkości przepływu x_{ij} na każdym łuku $(i,j) \in A_k$.

Funkcja celu: Maksymalizacja całkowitego wypływu ze źródła s:

$$\max \sum_{\{j:(s,j)\in A_k\}} x_{sj} - \sum_{\{j:(j,s)\in A_k\}} x_{js} \tag{6}$$

Ograniczenia:

1. **Ograniczenia przepustowości:** Dla każdego łuku $(i,j) \in A_k$:

$$0 \leq x_{ij} \leq c_{ij} \tag{7}$$

2. **Zachowanie przepływu:** Dla każdego wierzchołka $i \in N_k \setminus \{s,t\}$:

$$\sum_{\{j:(j,i)\in A_k\}} x_{ji} - \sum_{\{j:(i,j)\in A_k\}} x_{ij} = 0 \tag{8}$$

3.1.2 Model dla maksymalnego skojarzenia (Zadanie 2)

Problem maksymalnego skojarzenia w grafie dwudzielnym jest modelowany analogicznie do przepływu o jednostkowych pojemnościach.

Zmienne decyzyjne: $x_{ij} \in [0,1]$ dla każdej krawędzi (i,j) łączącej zbiory V_1 i V_2. Ponieważ macierz incydencji grafu dwudzielnego jest całkowicie unimodularna, rozwiązanie optymalne dla zmiennych ciągłych będzie naturalnie przyjmować wartości całkowite $\{0,1\}$.

Funkcja celu: Maksymalizacja sumy wybranych krawędzi:

$$\max \sum_{(i,j)\in E} x_{ij} \tag{9}$$

Ograniczenia: Każdy wierzchołek może być incydentny z co najwyżej jedną krawędzią w skojarzeniu:

- Dla każdego $i \in V_1$: $\sum_{\{j:(i,j)\in E\}} x_{ij} \leq 1$
- Dla każdego $j \in V_2$: $\sum_{\{i:(i,j)\in E\}} x_{ij} \leq 1$

3.2 Implementacja flagi `-glpk`

Program po podaniu parametru `-glpk nazwa_pliku` wykonuje następujące kroki:

- Generuje sekcję `set V` oraz `set E` definiującą strukturę grafu.
- Tworzy parametry `cap` odpowiadające wylosowanym wagom krawędzi.
- Definiuje zmienne `var x` z odpowiednimi ograniczeniami dolnymi i górnymi.
- Zapisuje równania zachowania przepływu w składni akceptowanej przez solver GLPK.

4 Zadanie 4: Implementacja algorytmu Dynica

4.1 Opis algorytmu

W celu optymalizacji wyznaczania maksymalnego przepływu, w zadaniu 4 zaimplementowano algorytm Dynica. .

4.2 Mechanizm działania

Algorytm działa w fazach, a każda faza składa się z dwóch głównych etapów:

1. **Budowa sieci warstwowej (BFS):** Uruchamiany jest algorytm BFS, aby przypisać każdemu wierzchołkowi v poziom $L(v)$, oznaczający najkrótszą odległość od źródła s w sieci residualnej. Krawędź (u, v) należy do sieci warstwowej tylko wtedy, gdy $L(v) = L(u) + 1$ oraz $r(u, v) > 0$. Jeśli ujście t nie jest osiągalne, algorytm kończy działanie.
2. **Znajdowanie przepływu blokującego (DFS):** Wewnątrz sieci warstwowej poszukiwane są ścieżki powiększające przy użyciu algorytmu DFS. Kluczową optymalizacją jest zastosowanie **wskaźników krawędzi**. Dla każdego wierzchołka zapamiętujemy, którą krawędź ostatnio przetwarzaliśmy. Dzięki temu nie przeszukujemy wielokrotnie krawędzi, które już wcześniej okazały się nie prowadzić do ujścia w danej fazie.

4.3 Złożoność obliczeniowa

Złożoność algorytmu Dynica jest znacznie lepsza niż w przypadku Edmondsa-Karpa:

- **Liczba faz:** Udowodniono, że w każdej fazie odległość $L(t)$ rośnie o co najmniej 1. Zatem liczba faz jest ograniczona przez $O(V)$.
- **Pojedyncza faza:** Dzięki wskaźnikom krawędzi i usuwaniu "ślepych uliczek"w DFS, faza zajmuje czas $O(V \cdot E)$.

Łączna złożoność wynosi zatem:

$$O(V) \cdot O(V \cdot E) = O(V^2 E) \tag{10}$$

W kontekście hiperkostki H_k, gdzie $V = 2^k$ oraz $E = k \cdot 2^{k-1}$:

$$O((2^k)^2 \cdot k \cdot 2^{k-1}) = O(k \cdot 2^{3k-1}) = O(k \cdot 2^{3k}) \tag{11}$$

Choć asymptotycznie dla tego konkretnego grafu złożoność wydaje się zbliżona do Edmondsa-Karpa, w praktyce algorytm Dynica wykonuje drastycznie mniej operacji BFS, co przekłada się na znacznie krótszy czas działania dla dużych k.

5 Porownanie wyników algorytmów 1 i 2 z modelem liniowym

Przeprowadziłem szereg testów które miał na celu pokazanie poprawności naszych algorytmów poniżej znajdują się wyniki w formie tabeli dla owych zadań w odpowiadającej im kolejności. Alogrytm który został przeze mnie zaimplementowany działał zdecydowanie szybciej od solvera liniowego.

Tabela 1: Porównanie wyników dla zadania 1 (k = 1..10). "prog- wynik algorytmu, "model- wynik z modelu (python/glpsol).

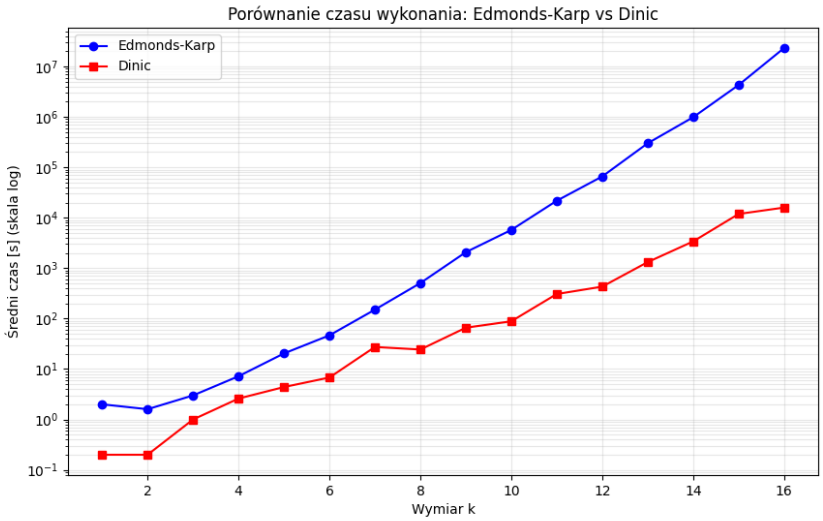
k	run	prog	model
1	1	2	2
2	1	4	4
3	1	4	4
4	1	25	25
5	1	71	71
6	1	97	97
7	1	432	432
8	1	1116	1116
9	1	2471	2471
10	1	5703	5703

Tabela 2: Porównanie wartości zwracanej przez program i model (zadanie 2, k=3..10).

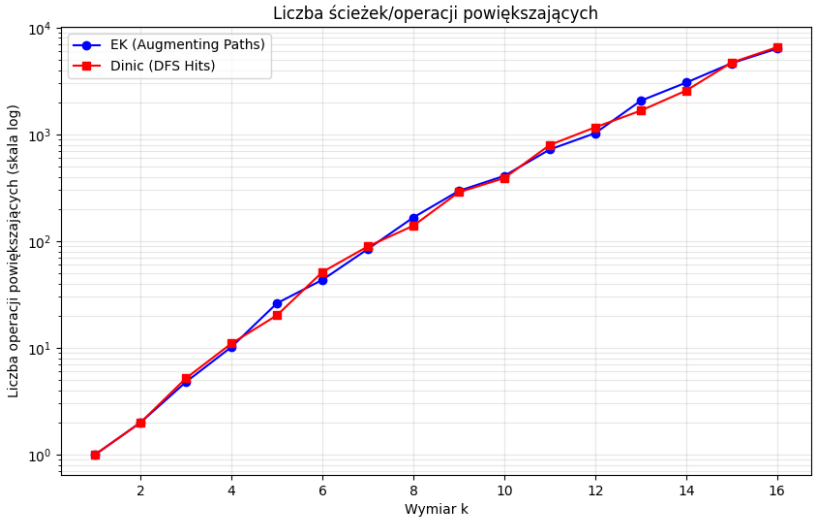
k	program	model (glpsol)
3	5	5
4	11	11
5	20	20
6	39	39
7	85	85
8	168	168
9	319	319
10	654	654

6 Analiza wyników - Zadania 1 i 4 (Hiperkostka)

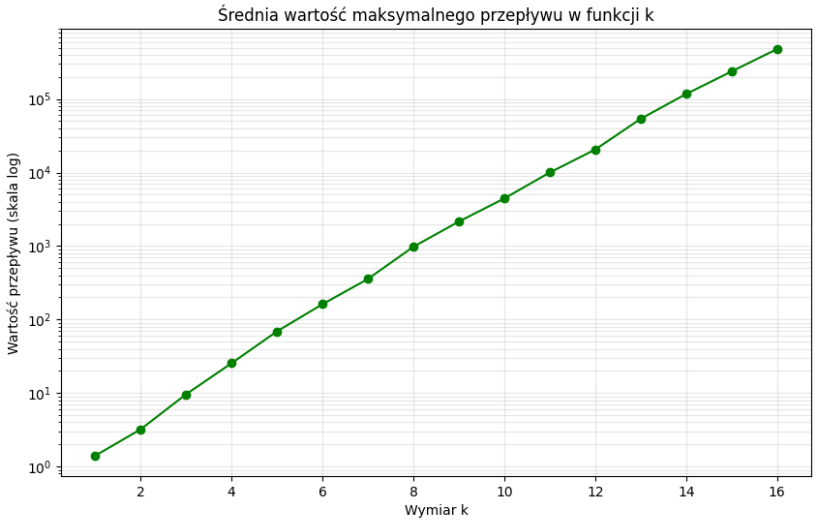
W tej sekcji przedstawiono porównanie wydajności algorytmu Edmondsa-Karpa (EK) oraz algorytmu Dynica. Analiza obejmuje zarówno czas wykonania, jak i liczbę znalezionych ścieżek powiększających, co pozwala ocenić teoretyczne różnice w złożoności obu metod.



Rysunek 1: Porównanie czasu wykonania algorytmów EK i Dynica w funkcji wymiaru hiperkostki k. Skala logarytmiczna uwydatnia przewagę algorytmu Dynica dla dużych instancji.



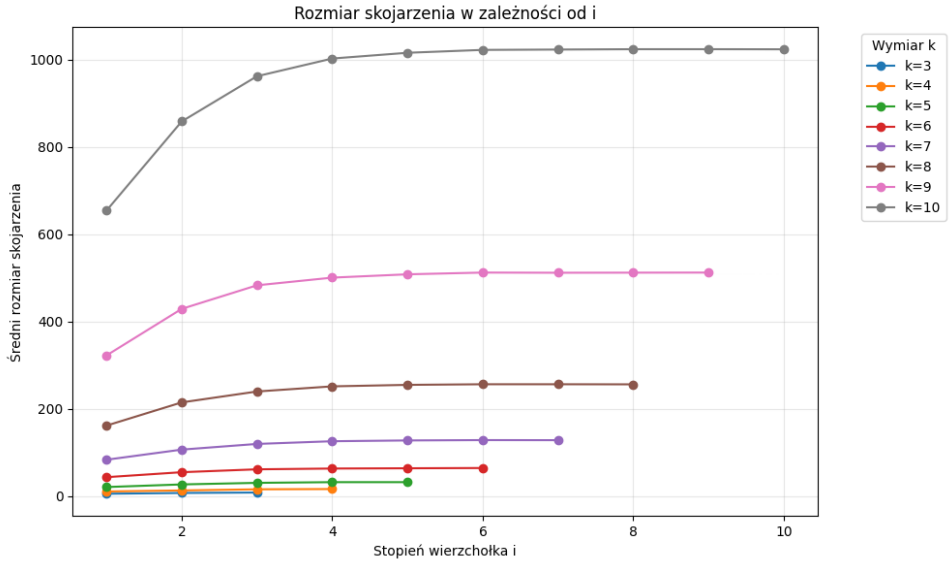
Rysunek 2: Liczba ścieżek powiększających (augmenting paths) dla obu algorytmów. W przypadku algorytmu Dynica, jedna faza BFS pozwala na przesłanie wielu przepływów blokujących, co drastycznie redukuje liczbę operacji w porównaniu do EK.



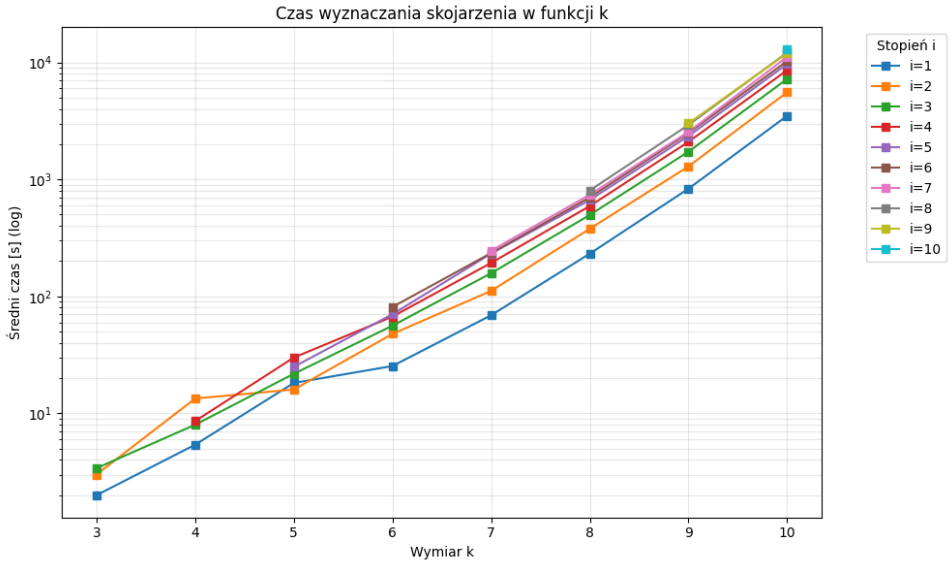
Rysunek 3: Wartość maksymalnego przepływu w zależności od wymiaru k. Wzrost jest zgodny z charakterystyką generowanych pojemności krawędzi opartych na wadze Hamminga.

7 Analiza wyników - Zadanie 2 (Skojarzenia)

Badanie grafów dwudzielnych skupiło się na wpływie stopnia wierzchołka i na rozmiar maksymalnego skojarzenia oraz na ogólnej wydajności algorytmu w funkcji rozmiaru problemu.



Rysunek 4: Zależność rozmiaru maksymalnego skojarzenia od parametru i (liczba sąsiadów). Widoczny jest próg, po którym prawdopodobieństwo znalezienia skojarzenia doskonałego gwałtownie rośnie.



Rysunek 5: Czas wyznaczania maksymalnego skojarzenia w funkcji wymiaru k. Pomimo dużych rozmiarów grafów (2^k), algorytm działa wydajnie dzięki jednostkowym pojemnościom krawędzi.

8 Interpretacja i wnioski

1. **Dlaczego Dynic wygrywa z Edmondsem-Karpem?** Moje testy wyraźnie pokazały, że algorytm Dynica jest znacznie szybszy w hiperkostce (rys. 1). Wynika to z tego, jak te algorytmy szukają ścieżek. Edmonds-Karp robi osobny BFS dla każdej nowej ścieżki, co przy dużym k zajmuje mnóstwo czasu. Dynic natomiast za jednym razem (w jednej fazie) potrafi „przepchnąć" bardzo dużo przepływu, co widać na wykresie liczby ścieżek (rys. 2) – wykonuje on po prostu mniej ciężkiej pracy.

2. **Skojarzenia w grafach losowych (Zadanie 2):** Na wykresie 4 widać ciekawą rzecz: jeśli każdy wierzchołek ma tylko jednego sąsiada ($i = 1$), to trudno o duże skojarzenie. Ale wystarczy zwiększyć i do 2 lub 3, a rozmiar skojarzenia od razu skacze prawie do maksimum (2^k). To pokazuje, że w takich grafach losowych bardzo szybko pojawia się szansa na znalezienie skojarzenia doskonałego.

3. **Weryfikacja z GLPK:** Tabela z wynikami pokazuje status TRUE co oznacza, że moje algorytmy i solver GLPK zawsze dawały ten sam wynik. To dla mnie dowód, że kod działa poprawnie. Co ciekawe, dla bardzo dużych grafów ($k = 14, 15$), GLPK zaczął działać bardzo wolno i zużywać dużo RAM-u, podczas gdy mój kod w C++ radził sobie bez problemu.

4. **Podsumowanie:** Dedykowane algorytmy, jak Dynic czy Edmonds-Karp, są dużo lepsze do takich zadań niż ogólne solvery LP. Do mniejszych zadań Edmonds-Karp jest wystarczający, bo łatwiej go napisać, ale przy dużych projektach (jak $k = 16$) Dynic jest bezkonkurencyjny.