# Sprawozdanie – Laboratoria 4.

Jakub Kogut

14 stycznia 2026

## 1 Wstęp

Na liście 4. zajmujemy się problemem *Max Flow*. Celem jest znalezienie maksymalnego przepływu w sieci przepływowej. W tym celu zaimplementowano algorytm *Edmonds-Karp*, który jest implementacją medoty *Ford-Fulkerson* wykorzystującą przeszukiwanie wszerz (BFS) do znajdowania ścieżek powiększających, oraz algorytm *Dinica*, który wykorzystuje koncepcję warstwowego grafu resztkowego do znajdowania blokujących przepływów, jak i prównanie z rozwiązaniem problemu jako zagadnienia programowania liniowego przez wykorzystanie *GLPK*.

## 2 Opis problemów

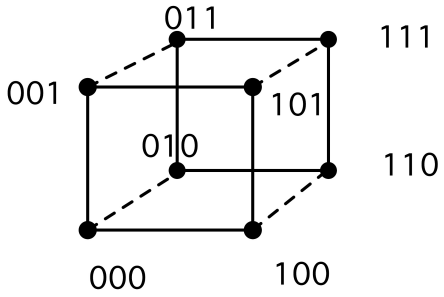
W zadaniach pojawiają się dwie różne rodziny grafów przepływowych, na których należy znaleźć maksymalny przepływ.

- **hiperkostki** – grafy zdefiniowane jako:

$$H_k = (N_k, A_k)$$

gdzie:

  - $N_k = \{0, 1, \ldots, 2^k - 1\}$ – zbiór wierzchołków reprezentowanych jako liczby całkowite od 0 do $2^k - 1$,
  - $A_k = \{(i, j) \in N_k \times N_k : e(i) - e(j) \geq 0 \wedge H(e(i) - e(j)) = 1\}$ – zbiór krawędzi, gdzie $e : \mathbb{N} \to \{0, 1\}^k$, $e(n)$ to reprezentacja wektorowa (postać binarna) liczby $n$, a $H(u, v)$ to odległość Hamminga między wektorami binarnymi $u$ i $v$.




Rysunek 1: Hiperkostka $H_3$

Grafy z tej roziny mają $2^k$ wierzchołków i $k \cdot 2^{k-1}$ krawędzi, są $k$-regularne. W zadaniu każda krawędź ma przypisaną losową, z rozkładem jednostanym, pojemność ze zbioru $\{1, \ldots, l\}$, gdzie $l = \max_{(i,l) \in A_k} \{H(e(i)), Z(e(i)), H(e(j)), Z(e(j))\}$.

- **nieskierowany graf dwudzielny** – grafy zdefiniowane jako:

$$G_{k,i} = (V_1 \cup V_2, E)$$

gdzie:
  - $V_1 = \{u_1, u_2, \ldots, u_{2^k}\}$ – zbiór wierzchołków po lewej stronie,
  - $V_2 = \{v_1, v_2, \ldots, v_{2^k}\}$ – zbiór wierzchołków po prawej stronie,
  - $E = \{(u_m, v_n) : 1 \leq m \leq 2^k, 1 \leq n \leq 2^k\}$ – zbiór krawędzi łączących wierzchołki z lewej i prawej strony.

# 3 Implementacja

W tej części do rozwiązania problemu zaimplementowałem 2 algorytmy: *Edmonds-Karp* oraz *Dinica*. Dodatkowo wykorzystałem bibliotekę *GLPK* do rozwiązania problemu jako zagadnienia programowania liniowego.

## 3.1 Metoda Forda–Fulkersona

Metoda Forda–Fulkersona jest ogólną procedurą wyznaczania maksymalnego przepływu w sieci przepływowej $G = (V, E)$ z wyróżnionym źródłem $s$, ujściem $t$ oraz pojemnościami $c(u, v) \geq 0$. Idea metody polega na wielokrotnym znajdowaniu *ścieżek powiększających* w grafie rezydualnym i zwiększaniu przepływu wzdłuż tych ścieżek, aż do momentu, gdy dalsze powiększanie nie jest możliwe.

**Graf rezydualny.** Dla aktualnego przepływu $f$ definiuje się pojemność rezydualną

$$c_f(u, v) = c(u, v) - f(u, v),$$

czyli ile dodatkowego przepływu można jeszcze przesłać krawędzią $(u, v)$. Aby umożliwić modyfikację wcześniej wysłanego przepływu, do grafu rezydualnego dodaje się także krawędzie wsteczne: jeśli $f(u, v) > 0$, to w $G_f$ istnieje krawędź $(v, u)$ o pojemności $c_f(v, u) = f(u, v)$, pozwalająca „cofnąć" część przepływu. Graf rezydualny $G_f$ składa się z tych krawędzi $(u, v)$, dla których $c_f(u, v) > 0$.

**Ścieżka powiększająca i augmentacja.** Ścieżka $P$ od $s$ do $t$ w grafie rezydualnym $G_f$ nazywana jest ścieżką powiększającą. Dla takiej ścieżki wyznacza się *wąskie gardło*:

$$\Delta = \min_{(u,v) \in P} c_f(u, v),$$

czyli maksymalną wartość, o jaką można zwiększyć przepływ wzdłuż całej ścieżki, nie przekraczając pojemności. Następnie aktualizuje się przepływ:

- jeśli krawędź $(u, v)$ na ścieżce jest zgodna z kierunkiem oryginalnej krawędzi, zwiększa się $f(u, v)$ o $\Delta$,
- jeśli wykorzystano krawędź wsteczną $(v, u)$, to zmniejsza się $f(u, v)$ o $\Delta$ (co odpowiada cofnięciu części przepływu).

Po tej operacji zmieniają się pojemności rezydualne i w konsekwencji sam graf rezydualny $G_f$.

**Warunek zakończenia.** Procedura jest powtarzana, dopóki istnieje ścieżka powiększająca z $s$ do $t$ w grafie rezydualnym. Gdy $t$ staje się nieosiągalne z $s$ w $G_f$, nie ma już możliwości powiększenia przepływu i uzyskany przepływ jest maksymalny. Z twierdzenia max-flow min-cut wynika, że brak ścieżki powiększającej jest równoważny istnieniu przekroju o pojemności równej aktualnej wartości przepływu.

**Złożoność i uwagi.** Metoda Forda–Fulkersona nie precyzuje sposobu wyboru ścieżek powiększających. Dla pojemności całkowitych gwarantuje to zakończenie po skończonej liczbie augmentacji, lecz czas działania zależy od wyboru ścieżek. W skrajnym przypadku liczba iteracji może być bardzo duża. Dwa popularne uszczegółowienia tej metody to:

- Edmonds–Karp: wybiera najkrótszą ścieżkę (BFS), co daje złożoność $O(|V| \cdot |E|^2)$,
- Dinic: pracuje fazami na grafie warstwowym, osiągając $O(|V|^2 \cdot |E|)$ w ogólnym przypadku.

**Uwagi implementacyjne.** W implementacji praktyczne jest przechowywanie dla każdej krawędzi jej krawędzi odwrotnej (reverse), co umożliwia szybkie aktualizacje pojemności rezydualnych przy augmentacji. Wyszukiwanie ścieżki powiększającej można realizować dowolnym przeszukiwaniem grafu (DFS/BFS), zależnie od wybranej warianty metody.

## 3.2 Edmondsa-Karpa

Algorytm Edmondsa–Karpa jest wariantem metody Forda–Fulkersona wyznaczania maksymalnego przepływu w sieci przepływowej $G = (V, E)$ z wyróżnionym źródłem $s$ i ujściem $t$ oraz pojemnościami $c(u, v) \geq 0$. Kluczową cechą EK jest to, że w każdej iteracji wybiera on ścieżkę powiększającą o najmniejszej liczbie krawędzi, tzn. znajduje ją algorytmem BFS w grafie rezydualnym.

**Graf rezydualny.** Dla aktualnego przepływu $f$ definiuje się pojemność rezydualną

$$c_f(u, v) = c(u, v) - f(u, v),$$

a także krawędzie wsteczne, które umożliwiają cofanie wcześniej wysłanego przepływu: jeśli $f(u, v) > 0$, to w grafie rezydualnym istnieje krawędź $(v, u)$ o pojemności rezydualnej $c_f(v, u) = f(u, v)$. Zbiór krawędzi rezydualnych tworzy graf $G_f$.

**Iteracja algorytmu.** W każdej iteracji:

1. Uruchamiamy BFS w grafie rezydualnym $G_f$ zaczynając od $s$, aby znaleźć najkrótszą (w liczbie krawędzi) ścieżkę powiększającą do $t$.
2. Jeśli $t$ jest nieosiągalne, to nie istnieje już ścieżka powiększająca i obecny przepływ jest maksymalny.
3. W przeciwnym razie wyznaczamy *wąskie gardło* ścieżki $P$:

$$\Delta = \min_{(u,v) \in P} c_f(u, v),$$

   a następnie powiększamy przepływ o $\Delta$ na krawędziach ścieżki $P$. Dla krawędzi zgodnych z kierunkiem zwiększamy $f(u, v)$, natomiast dla krawędzi wstecznych zmniejszamy $f(u, v)$ (co odpowiada przesuwaniu przepływu na alternatywne trasy).

Każde takie powiększenie zwiększa wartość przepływu o $\Delta$ i jest liczone jako jedna *ścieżka powiększająca* (użyte w eksperymentach jako licznik iteracji algorytmu).

**Złożoność.** W EK każda iteracja znajduje ścieżkę BFS w czasie $O(|E|)$. Można wykazać, że liczba iteracji jest ograniczona przez $O(|V| \cdot |E|)$, ponieważ odległość (w sensie BFS) najkrótszej ścieżki od $s$ do dowolnego wierzchołka w grafie rezydualnym nie maleje, a „krawędzie krytyczne" (saturujące się na ścieżkach) mogą zmieniać swój status ograniczoną liczbę razy. W konsekwencji całkowita złożoność czasowa wynosi

$$O(|V| \cdot |E|^2).$$

**Uwagi implementacyjne.** W implementacji wygodnie jest przechowywać sieć w postaci list sąsiedztwa z parami krawędzi: krawędzią *forward* o pojemności rezydualnej oraz krawędzią *reverse* umożliwiającą cofanie przepływu. BFS zapisuje poprzednika wierzchołka i indeks krawędzi, co pozwala po znalezieniu $t$ odtworzyć ścieżkę, wyznaczyć $\Delta$, a następnie zaktualizować pojemności rezydualne na krawędziach forward i reverse.

### 3.3 Algorytm Dinica

Algorytm Dinica (Dinic) służy do wyznaczania maksymalnego przepływu w sieci przepływowej $G = (V, E)$ ze źródłem $s$, ujściem $t$ oraz pojemnościami $c(u, v) \geq 0$. Podobnie jak w metodzie Forda–Fulkersona operuje on na grafie rezydualnym $G_f$, lecz przyspiesza działanie dzięki pracy na *grafie warstwowym* oraz wyszukiwaniu *przepływu blokującego*.

**Graf rezydualny.** Dla aktualnego przepływu $f$ definiuje się pojemności rezydualne $c_f(u, v)$ oraz krawędzie wsteczne o pojemności $f(u, v)$ (umożliwiające cofnięcie części przepływu). Krawędzie o $c_f(u, v) > 0$ tworzą graf rezydualny $G_f$.

**Faza: budowa grafu warstwowego (BFS).** W każdej fazie algorytm wykonuje BFS w $G_f$ od $s$ i wyznacza odległości (liczbę krawędzi) do pozostałych wierzchołków. Na tej podstawie definiuje się *graf warstwowy* $L$, zawierający tylko te krawędzie rezydualne $(u, v)$, które spełniają warunek:

$$\text{dist}(v) = \text{dist}(u) + 1.$$

W praktyce oznacza to, że w grafie warstwowym przepływ może iść wyłącznie „do przodu" pomiędzy kolejnymi warstwami. Jeśli BFS nie osiąga $t$ (tj. $\text{dist}(t) = -1$), to w $G_f$ nie istnieje ścieżka powiększająca i przepływ jest maksymalny.

**Faza: przepływ blokujący (DFS).** Mając graf warstwowy $L$, algorytm wielokrotnie wysyła przepływ z $s$ do $t$ wzdłuż krawędzi warstwowych za pomocą DFS (z ograniczeniem pojemności rezydualnych). Celem jest wyznaczenie *przepływu blokującego* (blocking flow), czyli takiego przepływu w $L$, po którym żadna ścieżka $s \to t$ w grafie warstwowym nie ma dodatniej przepustowości rezydualnej (co najmniej jedna krawędź na każdej ścieżce zostaje nasycona). W implementacji stosuje się tablicę wskaźników (np. `it[v]`), aby nie przeszukiwać w DFS wielokrotnie tych samych krawędzi, które zostały już „wyczerpane" w danej fazie.

**Schemat algorytmu.** Algorytm powtarza fazy:

1. BFS w $G_f$ i budowa warstw dist$[\cdot]$.
2. Dopóki istnieje możliwość wysłania dodatniego przepływu w $L$: DFS zwiększający przepływ (augmentacja) wzdłuż krawędzi spełniających warunek warstwowy.

Po zakończeniu fazy graf warstwowy nie zawiera już ścieżki $s \to t$ o dodatniej przepustowości; następnie wykonywany jest kolejny BFS, który albo wydłuża najkrótszą odległość do $t$, albo stwierdza brak ścieżki. To gwarantuje postęp i zakończenie algorytmu z przepływem maksymalnym.

**Złożoność.** Dla ogólnego grafu skierowanego klasyczna złożoność algorytmu Dinica wynosi

$$O(|V|^2 \cdot |E|).$$

W praktyce jest on zwykle znacząco szybszy niż Edmonds–Karp, ponieważ w jednej fazie (dla ustalonych warstw) potrafi „hurtowo" zwiększyć przepływ do momentu zablokowania wszystkich ścieżek w grafie warstwowym.

**Uwagi implementacyjne.** Podobnie jak w EK wygodna jest reprezentacja krawędzi w parach (forward/reverse) w liście sąsiedztwa. BFS wyznacza tablicę `level` (odległości warstw), a DFS korzysta z `it[v]` (aktualny indeks krawędzi), co zmniejsza liczbę powtórnych przejść po tych samych krawędziach w obrębie jednej fazy. W pomiarach można raportować m.in. liczbę uruchomień BFS (liczba faz) oraz liczbę wywołań DFS (prób wysłania przepływu).

# 4 Eksperymenty i wyniki

W zadaniach należało przeprowadzić eksperymenty porównujące czas działania zaimplementowanych algorytmów na różnych typach grafów przepływowych. Tutaj znajdują się ich wyniki oraz analiza.

## 4.1 Zadanie 1.

## 4.2 Zadanie 1.

**Cel.** Celem zadania 1 było wyznaczenie maksymalnego przepływu w sieciach będących skierowanymi hiperkostkami $H_k$ dla kolejnych wartości $k$ oraz porównanie praktycznego czasu działania algorytmów Edmondsa–Karpa i Dinica. Dodatkowo, dla mniejszych instancji, porównano wyniki z rozwiązaniem uzyskanym przy pomocy GLPK (traktując problem jako programowanie liniowe).

**Metodyka.** Dla każdej wartości $k$ generowano instancję hiperkostki z losowymi pojemnościami krawędzi (z wykorzystaniem stałego ziarna `seed`, aby zapewnić powtarzalność). Każdy pomiar wykonano kilkukrotnie, a następnie uśredniono czasy. Dla algorytmów własnych (EK i Dinic) mierzono czas wykonania programu, natomiast dla GLPK mierzono czas rozwiązania modelu w `glpsol` na wygenerowanym pliku `.mod`. Porównania wykonywano na tych samych danych wejściowych.

**Wyniki.** Na Figure 2 przedstawiono zależność czasu działania od parametru $k$ (w skali logarytmicznej na osi czasu). Widać, że wraz ze wzrostem rozmiaru instancji czas działania rośnie znacząco, a algorytm Dinica jest wyraźnie szybszy od Edmondsa–Karpa dla większych wartości $k$. Dla małych instancji dodatkowo porównano wynik z GLPK, który zwraca tę samą wartość maksymalnego przepływu, jednak jego czas rośnie szybko wraz z rozmiarem modelu.

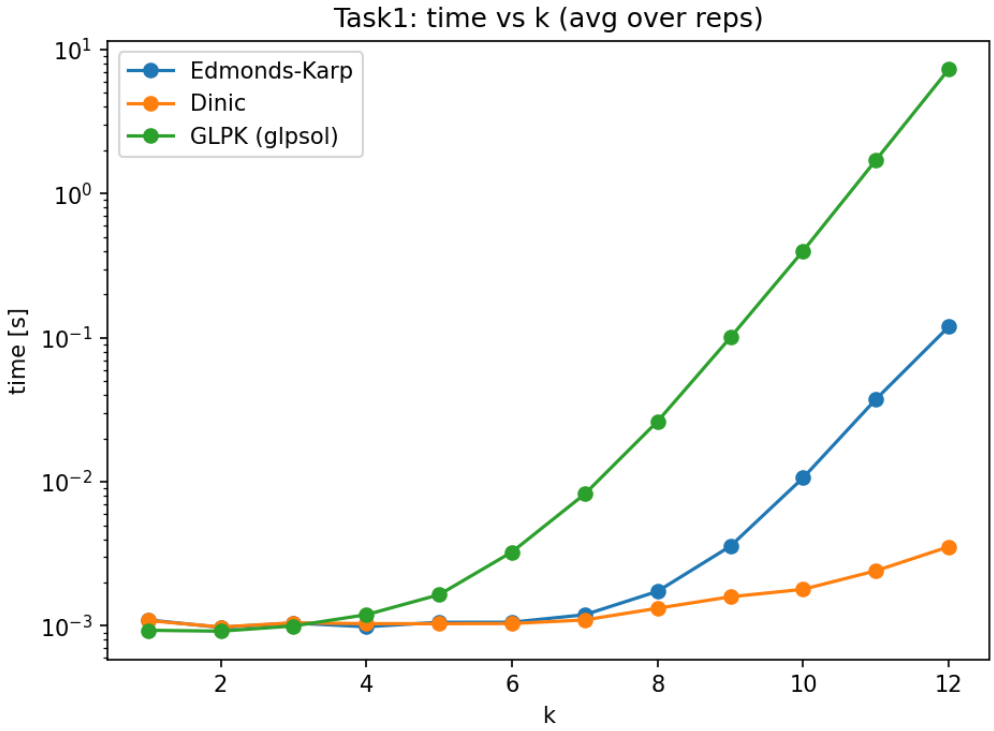
**Poprawność.** W testach porównawczych uzyskane wartości maksymalnego przepływu dla Edmondsa–Karpa i Dinica były identyczne dla tych samych instancji (ten sam `seed`). Dla mniejszych wartości $k$ wynik był dodatkowo zgodny z wartością obliczoną przez GLPK.

**Dodatkowa obserwacja.** Wartość maksymalnego przepływu zależy od wylosowanych pojemności, dlatego w eksperymentach istotne jest użycie stałego ziarna losowości. Dla pełności na Figure 3 pokazano przykładową zależność wartości maksymalnego przepływu od $k$.

**Podsumowanie.** Dla hiperkostek $H_k$ wraz ze wzrostem $k$ rośnie rozmiar sieci ($|V| = 2^k$, $|E| = k \cdot 2^{k-1}$), co silnie wpływa na czas obliczeń. W praktyce Dinic okazał się najbardziej efektywny czasowo dla większych instancji, natomiast Edmonds–Karp jest prostszy, ale skaluje się gorzej.

## 4.3 Zadanie 2.

**Cel.** Celem zadania 2 było znalezienie maksymalnego skojarzenia w losowym grafie dwudzielnym $G_{k,i}$ o rozmiarach $|V_1| = |V_2| = 2^k$, w którym każdy wierzchołek po lewej stronie ma dokładnie $i$ sąsiadów po prawej stronie. Problem sprowadzono do maksymalnego przepływu przez standardową




Rysunek 2: Zadanie 1: porównanie czasu działania (średnio z kilku uruchomień) w funkcji $k$. Oś czasu w skali logarytmicznej.

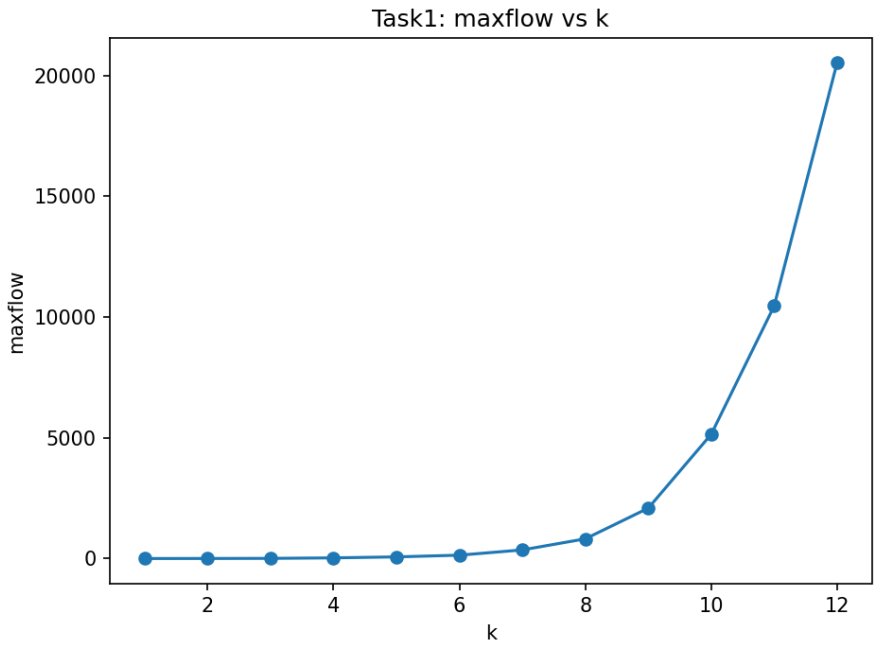
konstrukcję sieci: źródło $s$ połączone z każdym $u \in V_1$ krawędzią o pojemności 1, krawędzie $u-v$ (dla $(u, v) \in E$) o pojemności 1 oraz krawędzie z $v \in V_2$ do ujścia $t$ o pojemności 1. W takiej sieci wartość maksymalnego przepływu jest równa rozmiarowi maksymalnego skojarzenia.

**Metodyka.** Dla wybranych wartości $k$ oraz stopnia $i$ generowano losowe grafy dwudzielne, używając stałego `seed` w celu powtarzalności. Dla każdej pary parametrów wykonywano kilka uruchomień i uśredniano czasy. Porównano algorytmy Edmondsa–Karpa i Dinica, a dla mniejszych instancji dodatkowo rozwiązanie przez GLPK (poprawność wartości maksymalnego przepływu/matchingu).

**Wyniki czasowe.** Na Figure 4 przedstawiono zależność czasu działania od stopnia $i$ (uśrednioną po rozpatrywanych wartościach $k$ oraz powtórzeniach). Oś czasu jest w skali logarytmicznej. Wraz ze wzrostem $i$ graf staje się gęstszy (więcej krawędzi $V_1 \times V_2$), co zwykle zwiększa koszt obliczeń. W praktyce Dinic wypada szybciej od Edmondsa–Karpa dla większych instancji, co jest zgodne z oczekiwaniami dotyczącymi skalowania obu metod.

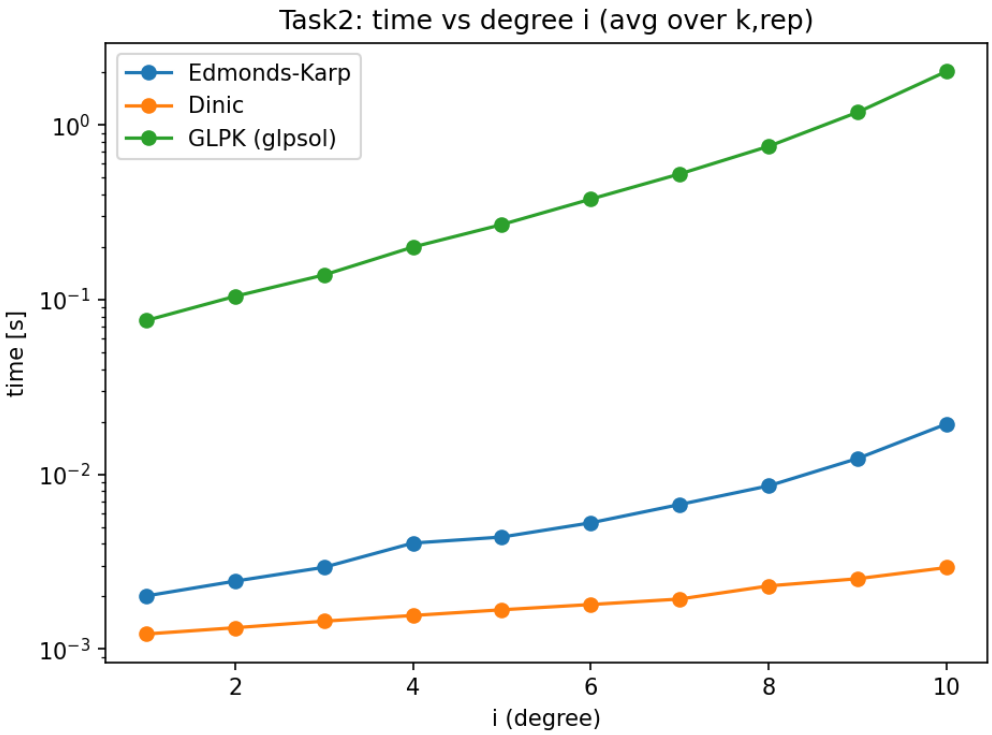
**Wielkość skojarzenia.** Rozmiar maksymalnego skojarzenia zależy od parametrów $k$ i $i$ oraz konkretnej realizacji losowania. Dla małych $i$ graf może nie zawierać doskonałego skojarzenia, natomiast przy większym $i$ rośnie szansa na skojarzenie bliskie $2^k$. Dla ilustracji na Figure 5 pokazano przykładową zależność rozmiaru skojarzenia od $i$.

**Poprawność.** Dla tych samych instancji (ten sam `seed`) Edmonds–Karp i Dinic zwracały identyczny rozmiar skojarzenia. Dla mniejszych przypadków wynik był dodatkowo zgodny z GLPK, co potwierdza poprawność redukcji matchingu do max-flow oraz implementacji algorytmów.

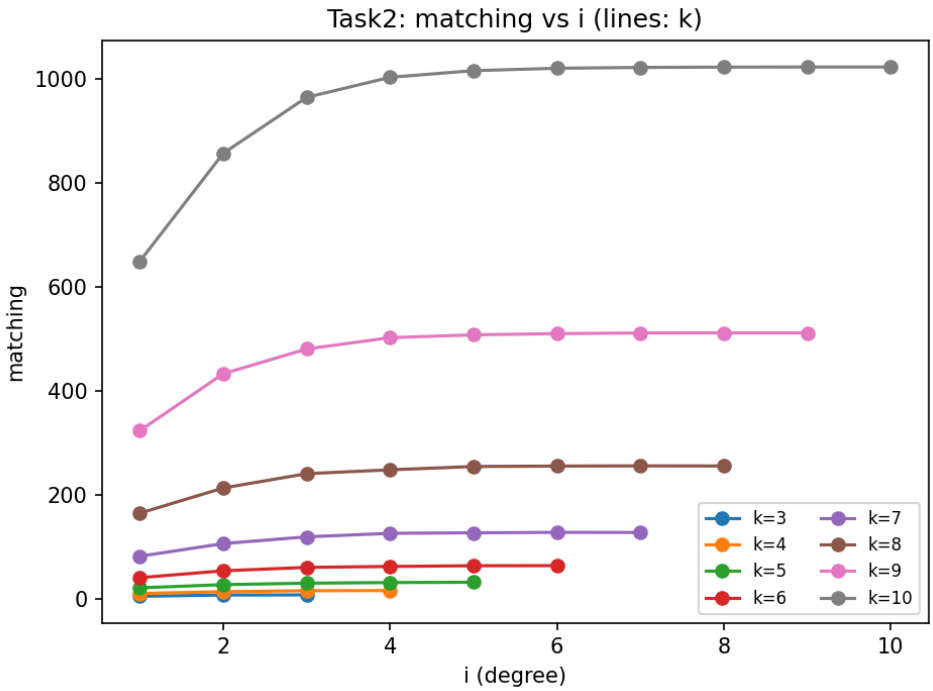



Rysunek 3: Zadanie 1: przykładowa wartość maksymalnego przepływu w funkcji $k$ (dla ustalonego `seed`).

**Podsumowanie.** Zadanie 2 pokazuje praktyczne różnice w wydajności algorytmów max-flow na sieciach powstających z grafów dwudzielnych. Wraz ze wzrostem stopnia $i$ rośnie liczba krawędzi w sieci, co wpływa na czas obliczeń. Dinic w praktyce lepiej skaluje się na gęstszych instancjach niż Edmonds–Karp, zachowując tę samą wartość rozwiązania.




Rysunek 4: Zadanie 2: porównanie czasu działania w funkcji stopnia $i$ (średnio z kilku uruchomień), oś czasu w skali logarytmicznej.




Rysunek 5: Zadanie 2: przykładowy rozmiar maksymalnego skojarzenia w funkcji stopnia $i$ (dla wybranych wartości $k$ i ustalonego `seed`).