Wydział Matematyki
i Nauk Informacyjnych

POLITECHNIKA WARSZAWSKA

Teoria algorytmów i obliczeń

Projekt zaliczeniowy

Piotr Jacak
Jakub Kindracki
Wiktor Kobielski
Ernest Mołczan

Koordynator: prof. dr hab. inż. Władysław Homenda

Semestr zimowy 2025/2026

Spis treści

1 Wstęp **3**

2 Definicje pojęć **4**

3 Rozmiar multigrafu **5**

4 Metryka w zbiorze wszystkich multigrafów **7**

5 Minimalne rozszerzenie multigrafu **8**
5.1 Algorytm dokładny dla problemu izomorfizmu podgrafu 8
5.1.1 Dowód poprawności 9
5.1.2 Złożoność obliczeniowa 9
5.2 Aproksymacyjne minimalne rozszerzenie multigrafu 11
5.2.1 Opis algorytmu 11
5.2.2 Złożoność obliczeniowa 12

6 Minimalne rozszerzenie multigrafu zawierającego m kopii podgrafu P **14**
6.1 Motywacja i sformułowanie problemu 14
6.2 Definicje formalne 15
6.3 Algorytm dokładny 17
6.3.1 Przegląd algorytmu 17
6.3.2 Szczegółowy opis algorytmu 17
6.3.3 Pseudokod algorytmu 21
6.4 Dowód poprawności algorytmu 22
6.5 Analiza złożoności obliczeniowej 23
6.5.1 Złożoność czasowa 23
6.5.2 Złożoność pamięciowa 25
6.5.3 Charakterystyka algorytmu 26
6.6 Przykład działania algorytmu 26
6.7 Optymalizacje i możliwe ulepszenia 29
6.7.1 Optymalizacje implementacyjne 29
6.7.2 Heurystyki i algorytmy aproksymacyjne 29
6.8 Podsumowanie 31

7 Bibliografia **32**

1 Wstęp

Niniejsza praca stanowi sprawozdanie z projektu zrealizowanego w ramach przedmiotu **Teoria algorytmów i obliczeń**. Przedmiotem badań są algorytmy operujące na multigrafach, ze szczególnym uwzględnieniem problematyki izomorfizmu podgrafów oraz minimalnych rozszerzeń grafów.

Głównym celem projektu jest opracowanie, analiza teoretyczna oraz implementacja algorytmów rozwiązujących dwa ściśle powiązane problemy. Pierwszym z nich jest weryfikacja, czy dany multigraf H jest izomorficzny z n podgrafami multigrafu G. Drugim, kluczowym zagadnieniem, jest wyznaczenie *minimalnego rozszerzenia* multigrafu G do postaci G', która zawiera co najmniej n podgrafów izomorficznych z H.

Realizacja powyższych celów wymagała formalnego zdefiniowania oraz uzasadnienia kilku fundamentalnych pojęć. W pracy zaproponowano autorskie lub bazujące na literaturze definicje:

- *rozmiaru multigrafu*,
- *metryki* w zbiorze multigrafów,
- *minimalnego rozszerzenia* multigrafu.

Pojęcia te stanowią podstawę do dalszej analizy algorytmicznej oraz oceny kosztu operacji.

W ramach pracy przeprowadzono analizę złożoności obliczeniowej opracowanych algorytmów. Zgodnie z założeniami projektu, w przypadku gdy algorytmy dokładne charakteryzują się złożonością wykładniczą, przedstawiono również propozycje algorytmów aproksymacyjnych o złożoności wielomianowej.

2 Definicje pojęć

Definicja 1 (Graf)**.** Grafem nazywamy parę $G = (V, E)$, gdzie V jest zbiorem wierzchołków, a $E \subseteq V \times V = \{(u, v) : u, v \in V \wedge u \neq v\}$ jest zbiorem krawędzi. Dla każdej pary wierzchołków $u, v \in V$ istnieje co najwyżej jedna krawędź łącząca wierzchołki u i v.

Definicja 2 (Multigraf)**.** Multigrafem nazywamy graf, w którym pomiędzy dowolnymi dwoma różnymi wierzchołkami $u, v \in V$ może istnieć więcej niż jedna krawędź.

Definicja 3 (Graf skierowany)**.** Grafem skierowanym nazywamy parę $G = (V, E)$, gdzie V jest zbiorem wierzchołków, a $E \subseteq V \times V = \{(u, v) : u, v \in V \wedge u \neq v\}$ jest zbiorem krawędzi. Krawędzie w grafie skierowanym mają określony kierunek, co oznacza, że krawędź (u, v) jest różna od krawędzi (v, u). Definicja jest analogiczna dla multigrafów.

Definicja 4 (Izomorfizm grafów)**.** Dwa grafy $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$ są izomorficzne, wtedy i tylko wtedy, gdy istnieje bijekcja $f : V_1 \to V_2$, taka że dla każdej krawędzi $(u, v) \in E_1$ zachodzi $(f(u), f(v)) \in E_2$. Definicja ta jest analogiczna dla multigrafów i grafów skierowanych.

Definicja 5 (Podgraf)**.** Graf $H = (V_H, E_H)$ nazywamy podgrafem grafu $G = (V_G, E_G)$, wtedy i tylko wtedy, gdy $V_H \subseteq V_G$ oraz $E_H \subseteq E_G$. Definicja ta jest analogiczna dla multigrafów i grafów skierowanych.

Definicja 6 (Graf atrybutowy)**.** Graf $G = (V, E, f)$ nazywamy grafem atrybutowym, gdzie V jest zbiorem wierzchołków, a $E \subseteq V \times V = \{(u, v) : u, v \in V \wedge u \neq v\}$ jest zbiorem krawędzi. Dla każdej pary wierzchołków $u, v \in V$ istnieje co najwyżej jedna krawędź łącząca wierzchołki u i v. $f : E \to \Sigma_E$ jest funkcją, przypisującą etykiety wszystkim krawędziom w grafie G.

Definicja 7 (Macierz sąsiedztwa)**.** Macierzą sąsiedztwa multigrafu $G = (V, E)$ nazywamy macierz A, której pole $A_{uv} = k$, wtedy i tylko wtedy, gdy istnieje k krawędzi $(u, v) \in E$. W przypadku gdy nie istnieje żadna krawędź pomiędzy wierzchołkami u i v, to $A_{uv} = 0$.

3 Rozmiar multigrafu

Definicja 8 (Rozmiar multigrafu)**.** Rozmiarem $S(G)$ multigrafu $G = (V, E)$ nazywamy parę liczb naturalnych $(|V|, |E|)$, gdzie $|V|$ oznacza liczbę wierzchołków, a $|E|$ liczbę krawędzi w multigrafie G.

Zakładamy, że liczby wierzchołków i krawędzi są zapisanymi wcześniej stałymi, więc obliczenie rozmiaru multigrafów jest operacją o złożoności czasowej $O(1)$.

Definicja 9 (Porządek w zbiorze wszystkich multigrafów)**.** Niech G_1 i G_2 będą dwoma multigrafami. Mówimy, że G_1 jest mniejszy, lub równy G_2 wtedy i tylko wtedy, gdy:

$$|V_1| < |V_2| \vee (|V_1| = |V_2| \wedge |E_1| \leq |E_2|)$$

Żeby udowodnić poprawność powyższej definicji porządku wykazujemy, że spełnia ona trzy wymagane własności:

- **Zwrotność**:

  $$S(G) \leq S(G)$$

  Dla dowolnego multigrafu $G = (V, E)$, zachodzi $|V| = |V| \wedge |E| = |E|$. Więc w szczególności spełnia on warunek $|V| = |V| \wedge |E| \leq |E|$ z definicji porządku. Stąd $S(G) \leq S(G)$.

- **Przechodniość**:

  $$S(G_1) \leq S(G_2) \wedge S(G_2) \leq S(G_3) \Rightarrow S(G_1) \leq S(G_3)$$

  Weźmy dowolne trzy multigrafy $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ oraz $G_3 = (V_3, E_3)$ takie, że $S(G_1) \leq S(G_2)$ oraz $S(G_2) \leq S(G_3)$.

  Załóżmy, że $S(G_1) \geq S(G_3)$. Z definicji to implikuje, że $|V_1| > |V_3| \vee (|V_1| = |V_3| \wedge |E_1| > |E_3|)$.

  Z założeń wiemy też, że $|V_2| > |V_1|$, lub $|V_2| = |V_1| \wedge |E_2| \geq |E_1|$.

  W pierwszym przypadku z założeń wynika, ze $|V_2| > |V_3|$, co stoi w sprzeczności z $S(G_2) \leq S(G_3)$.

  W drugim przypadku, z założeń wynika, że $|V_2| = |V_3|$ oraz $|E_2| > |E_3|$, co również stoi w sprzeczności z $S(G_2) \leq S(G_3)$.

  W obu przypadkach dochodzimy do sprzeczności, więc nasze początkowe założenie było fałszywe. Stąd $S(G_1) \leq S(G_3)$.

- **Antysymetryczność**:

$$S(G_1) \leq S(G_2) \wedge S(G_2) \leq S(G_1) \Rightarrow S(G_1) = S(G_2)$$

Weźmy dowolne dwa multigrafy $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$ takie, że $S(G_1) \leq S(G_2)$ oraz $S(G_2) \leq S(G_1)$. Z definicji porządku, z pierwszego założenia wynika, że $|V_1| < |V_2| \vee (|V_1| = |V_2| \wedge |E_1| \leq |E_2|)$. Z drugiego założenia wynika, że $|V_2| < |V_1| \vee (|V_2| = |V_1| \wedge |E_2| \leq |E_1|)$.

Jeśli $|V_1| < |V_2|$, to z drugiego założenia wynika, że $|V_2| < |V_1|$, co jest sprzeczne. Analogicznie, jeśli $|V_2| < |V_1|$, to z pierwszego założenia wynika, że $|V_1| < |V_2|$, co również jest sprzeczne. Zatem musi zachodzić $|V_1| = |V_2|$.

Wtedy z pierwszego założenia wynika, że $|E_1| \leq |E_2|$, a z drugiego, że $|E_2| \leq |E_1|$. Stąd $|E_1| = |E_2|$.

W rezultacie mamy $S(G_1) = S(G_2)$.

4 Metryka w zbiorze wszystkich multigrafów

Definicja 10 (Metryka w zbiorze multigrafów)**.** Niech $\mathcal{G}$ będzie zbiorem wszystkich multigrafów. **Metryką** w zbiorze $\mathcal{G}$ nazywamy funkcję:

$$d : \mathcal{G} \times \mathcal{G} \to \mathbb{N}_0$$

Wartość $d(G_1, G_2)$ nazywamy **odległością** między multigrafami G_1 i G_2, a definiujemy ją, jako **minimalną** liczbę operacji dodawania lub usuwania pojedynczej krawędzi lub wierzchołka, za pomocą których można przekształcić graf G_1 w graf izomorficzny z G_2.

Powyższa definicja spełnia następujące własności metryki:

- **Identyczność nierozróżnialnych**:
  Dla dowolnych multigrafów G_1 oraz G_2, $d(G_1, G_2) = 0$ wtedy i tylko wtedy, gdy G_1 jest izomorficzny z G_2. Wynika to bezpośrednio z definicji naszej metryki.

- **Symetria**:
  Dla dowolnych multigrafów G_1 oraz G_2, $d(G_1, G_2) = d(G_2, G_1)$. Dodawanie i usuwanie krawędzi lub wierzchołków jest operacją odwracalną, więc liczba operacji potrzebnych do przekształcenia G_1 w G_2 jest równa liczbie odwrotnych operacji potrzebnych do przekształcenia G_2 w G_1.

- **Nierówność trójkąta**:
  Dla dowolnych multigrafów G_1, G_2 oraz G_3, $d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3)$. Oznacza to, że najkrótsza droga między dwoma multigrafami nie może być dłuższa niż droga przechodząca przez trzeci multigraf. Jest to prawda, ponieważ każda sekwencja operacji przekształcających G_1 w G_2 oraz G_2 w G_3 może być złożona w jedną sekwencję przekształcającą G_1 w G_3.

5 Minimalne rozszerzenie multigrafu

5.1 Algorytm dokładny dla problemu izomorfizmu podgrafu

Mając dane dwa grafy G i H, chcemy znaleźć podgrafy G izomorficzne do H. Do rozwiązania tego problemu posłuży nam algorytm, który wykorzystuje procedurę Backtrackingu do sprawdzania struktury grafów.
Przed przejściem do algorytmu, zdefiniujmy sobie struktury przydatne nam do implementacji. Niech $n_G = |V(G)|$ - ilość wierzchołków w grafie G oraz $n_H = |V(H)|$ - ilość wierzchołków w Grafie H.

Opis algorytmu:

1. Inicjalizacja macierzy sąsiedztwa grafów G i H odpowiednio $S_G \in \mathbb{N}^{n_G \times n_G}$ i $S_H \in \mathbb{N}^{n_H \times n_H}$. Wartość $S[i,j]$, to ilość krawędzi pomiędzy i-tym, a j-tym wierzchołkiem dla danego grafu.

2. Inicjalizacja kandydatów - Zdefiniujmy sobie listę $mozliwe\_dopasowania$, $len(mozliwe\_dopasowania) = n_H$, gdzie pod i-tym indeksem, będziemy mieli listę możliwych dopasowań dla wierzchołka $i \in V(H)$.
   Algorytm Ullmana dla grafów prostych zakłada inicjalizację:
   $u \in V(G), u \in mozliwe\_dopasowania[i] \iff deg_G(u) \geq deg_H(i)$
   Jest ona działającą inicjalizacją dla multigrafów, jednak w celach optymalizacji algorytmu, możemy zmienić tę inicjalizację tak, aby zmniejszyć liczbę potencjalnych dopasowań, a co za tym idzie zmniejszyć liczbę gałęzi, które będzie musiał przejść algorytm. Możemy zauważyć, że w macierzach sąsiedztwa na głównej przekątnej pod indeksami $[i,i]$ znajduje się liczba pętli danego wierzchołka, zatem naszym warunkiem będzie także $S_G[i,i] \geq S_H[i,i]$. Biorąc to wszystko razem, otrzymujemy
   $u \in mozliwe\_dopasowania[i] \iff (deg_G(u) \geq deg_H(i)) \wedge (S_G[i,i] \geq S_H[i,i]))$

3. Dla każdej krawędzi, która istnieje między już dopasowanymi wierzchołkami z H, sprawdź czy istnieje krawędź między ich dopasowaniami z G i czy ilość krawędzi między dopasowaniami jest większa lub równa niż ilość krawędzi między wierzchołkami. Można to osiągnąć przez przejrzenie wszystkich par już dopasowanych wierzchołków i krawędzi między nimi. Jeśli nie, zwróć False

4. Sprawdź, czy wszystkie wierzchołki nie zostały już dopasowane. Jeśli tak, zwróć True.

5. Dla każdego wierzchołka $v \in mozliwe\_dopasowania[i]$, jeśli v *not in* $dopasowania$, przypisz $dopasowania[i] = v$ oraz wywołaj funkcję ponownie dla następnego wierzchołka $\in V(H)$ z przekazaną kopią. W przypadku wyniku True z tej funkcji, zwróć True, w przypadku False, $dopasowania[i] = null$ i przejdź do następnego kroku tej pętli.

6. W przypadku niedopasowania po wszystkich iteracjach pętli, zwróć False.

5.1.1 Dowód poprawności

Najpierw zbadajmy, czy algorytm dobrze inicjalizuje *mozliwe_dopasowania*. W tym celu rozbijmy wszystkie 3 warunki. Pierwszy warunek mówi o tym, że potencjalne dopasowanie v dla wierzchołka u, musi mieć stopień co najmniej równy stopniowi wierzchołka u. Gdyby tak nie było, w grafie G nie istniałaby co najmniej jedna krawędź wychodząca z v, która istniałaby w H i wychodziłaby z u, zatem v nie mogłoby być dopasowaniem dla u. Drugi warunek mówi o tym, że liczba pętli dla v musi być co najmniej równa liczbie pętli dla u. Idea jest taka sama jak warunku pierwszego, gdyby warunek nie był spełniony, nie istniałaby co najmniej jedna pętla da danego wierzchołka, a co za tym idzie, nie mógłby on być dopasowaniem dla u.

Dalej w algorytmie, przechodzimy po kolei po wierzchołkach z H. Najpierw sprawdzamy, czy struktura się zgadza dla tych wierzchołków, do których znaleźliśmy już dopasowania. Jeśli choć 1 krawędź istniejąca w H pomiędzy dwoma wierzchołkami nie będzie istnieć między ich dopasowaniami w G, algorytm wychodzi z tej ścieżki dopasowań i szuka innych, zatem działa poprawnie.

Następnie sprawdzamy wszystkie z możliwych dopasowań dla danego wierzchołka, zatem sprawdzając tak wszystkie wierzchołki, mamy pewność, że przejdziemy po wszystkich możliwych permutacjach.

5.1.2 Złożoność obliczeniowa

Zauważmy, że inicjalizacja *mozliwe_dopasowania* w taki sposób, że dla każdego wierzchołka $u \in V(H)$ możliwym dopasowaniem są wszystkie $v \in V(G)$, to algorytm przejdzie po wszystkich poddrzewach, zatem w przypadku pesymistycznym do 1 wierzchołka wykona n_G potencjalnych dopasowań, do drugiego $n_G - 1$, ..., a do n_H-tego, $(n_G - n_H + 1)$ dopasowań. Zatem mamy

$$\underbrace{(n_G)(n_G-1)\dots(n_G-n_H+1)}_{n_H \text{ razy}} \leq n_G^{n_H}$$

W każdej takiej pętli wykonujemy sprawdzenie, czy struktura grafu się zgadza, (krok 4). Zauważmy, że wykonamy tam i^2 porównań, gdzie i to indeks aktualnie obliczanego wierzchołka. Wiemy że $i < n_H$, zatem możemy ograniczyć tę operację: $i^2 < n_H^2$. W sumie możemy stwierdzić, że złożoność tego algorytmu wyniesie $O(n_G^{n_H} n_H^2)$

5.2 Aproksymacyjne minimalne rozszerzenie multigrafu

Do problemu można zastosować pewną modyfikację algorytmu LeRP (Length-R Paths), opracowanego przez Freda W DePiero oraz Davida Krouta [1]. Algorytm opiera się na założeniu, że o podobieństwie strukturalnym dwóch wierzchołków można wnioskować na podstawie porównania liczby ścieżek (*sygnatur*) o długości r w ich sąsiedztwie.

Modyfikacja algorytmu jako argumenty przyjmuje dwa multigrafy $G_1 = (V_1, E_1)$ i $G_2 = (V_2, E_2)$, maksymalną długość ścieżki R oraz liczbę szukanych kopii grafu k. Im większa długość ścieżki R, tym algorytm jest dokładniejszy. Zwraca natomiast przekształcenie $f(g_{1i}) = g_{2k}$, gdzie $g_{1i} \in G_1$ i $g_{2k} \in G_2$, które opisuje najlepszy (w rozumieniu aproksymacji) wspólny podgraf G_1 oraz G_2.
Oznaczmy przez $H = (V_H, E_H)$ najlepszy wspólny podgraf G_1 oraz G_2. Do multigrafu H należy dołożyć zbiór krawędzi X, tak aby grafy G_1 oraz $H' = (V_H, E_H \cup X)$ były izomorficzne. Wówczas multigraf $G_3 = (V_2, E_2 \cup X)$ będzie minimalnym rozszerzeniem G_2, aby ten zawierał G_1 jako podgraf. Przez minimalne rozszerzenie, rozumie się minimalną liczbę dodanych krawędzi do grafu.
Następnie usuwamy z początkowego grafu G_2 wierzchołki podgrafu H i powtarzamy proces dla grafów G_1 i $G_2' = (V_2 - V_H, E_2')$, aby znaleźć k rozdzielnych kopii G_1 w grafie G_2.

5.2.1 Opis algorytmu

Algorytm można podzielić na kilka etapów.

1. W pierwszym kroku, należy wykonać transformację multigrafów wejściowych $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$ na grafy atrybutowe G_1' oraz G_2'. Transformacja przebiega w następujący sposób:

   - Zbiory wierzchołków pozostają bez zmian ($V_1' = V_1, V_2' = V_2$).
   - Dla każdej pary wierzchołków (u, v) w G_1 (i analogicznie w G_2): Jeśli między u a v w G_1 istnieje k równoległych krawędzi, to w G_1' tworzona jest pojedyncza krawędź (u, v) z atrybutem $k \in \mathbb{N}^+$.

2. W etapie drugim, dla obu grafów G_1' i G_2' obliczane są potęgi ich macierzy sąsiedztwa, odpowiednio A^r i B^r aż do maksymalnej długości R. Wartość A^r_{ij} w macierzy A^r reprezentuje liczbę ścieżek o długości dokładnie r z wierzchołka i do wierzchołka j (na ścieżkach mogą się powtarzać wierzchołki i krawędzie). W

kontekście omówionej transformacji, macierz A jest macierzą, gdzie A_{ij} przechowuje atrybut k - liczbę równoległych krawędzi między wierzchołkami i i j w multigrafie G_1.

3. Każdy wierzchołek $g_{1i} \in G'_1$ można porównać z każdym wierzchołkiem $g_{2k} \in G'_2$, stosując przykładowo podobieństwo cosinusowe między histogramem wartości w wierszu macierzy A^r_i a histogramem wartości w wierszu macierzy B^r_k. Następnie tworzymy macierz z wartościami podobieństw między wierzchołkami. Wierzchołki najbardziej podobne zostają ziarnem mapowania.

4. Następnie iteracyjnie (przykładowo DFS), algorytm porównuje sąsiadów wierzchołków zmapowanych. Do następnego mapowania, wybiera tych sąsiadów, do których liczba ścieżek o długości co najwyżej R jest największa i jest równa dla obu wierzchołków z dwóch grafów. Algorytm sprawdza również, czy wybrane wierzchołki nie zostały już zmapowane.

5. Zmapowane wierzchołki w grafie G'_2 są usuwane, tworząc nowy graf G''_2 i proces jest powtarzany dla grafów G'_1 oraz G''_2. W ten sposób znajdowane jest k rozdzielnych kopii G_1 w minimalnym rozszerzeniu G_2.

5.2.2 Złożoność obliczeniowa

Złożoność pesymistyczna omówionej modyfikacji algorytmu LeRP jest wielomianowa i wynosi $O(N^2 \cdot (N + E) \cdot D^2 \cdot R \cdot k)$, gdzie:

- N to liczba wierzchołków w grafach (zakładając, że oba grafy mają rozmiar rzędu N).
- E to liczba krawędzi w grafach (zakładając, że oba grafy mają liczbę krawędzi rzędu N).
- D to średni stopień wierzchołków w grafach.
- R to maksymalna długość ścieżki brana pod uwagę.
- k to liczba szukanych kopii w minimalnym rozszerzeniu

Uzasadnienie złożoności: porównanie par wierzchołków wymaga N^2 porównań. Każdy wierzchołek ma średnio D sąsiadów, więc porównywanie sąsiadów daje D^2 dodatkowych operacji. Analiza różnych długości ścieżek to czynnik R. Podczas budowania dopasowania, algorytm iteracyjny ma złożoność $(N + E)$. Czynnik k odpowiada za znalezienie k kopii mniejszego multigrafu G_1.

W kontekście tego algorytmu aproksymacyjnego nie rozważano formalnego dowodu poprawności. Dostarczono empiryczną gwarancję jakości - algorytm testowano na dużych zbiorach danych, wykazując, że algorytm konsekwentnie zwraca wyniki bliskie optimum w praktyce.

6 Minimalne rozszerzenie multigrafu zawierającego m kopii podgrafu P

6.1 Motywacja i sformułowanie problemu

W poprzednich sekcjach zajmowaliśmy się problemem weryfikacji istnienia pojedynczego podgrafu izomorficznego z danym wzorcem. W praktycznych zastosowaniach często pojawia się jednak bardziej ogólne zagadnienie: jak minimalnie rozszerzyć graf G, aby zawierał on m rozłącznych kopii grafu wzorcowego P?

Problem ten ma istotne zastosowania w dziedzinach takich jak:

- **Projektowanie sieci**: Zapewnienie redundancji przez istnienie wielu izomorficznych podsieci
- **Analiza struktur molekularnych**: Identyfikacja powtarzających się motywów strukturalnych
- **Analiza sieci społecznych**: Wykrywanie grup o podobnej strukturze relacji
- **Optymalizacja grafów**: Minimalne modyfikacje zachowujące pożądane właściwości strukturalne

Formalne sformułowanie problemu:
Dane:

- Multigraf skierowany $G = (V_G, E_G)$ o n wierzchołkach (graf "duży")
- Multigraf skierowany $P = (V_P, E_P)$ o k wierzchołkach, gdzie $k \leq n$ (graf "mały", wzorzec)
- Liczba naturalna $m \geq 1$ - wymagana liczba kopii

Zadanie:

- Znaleźć minimalny zbiór krawędzi E_{add} taki, że graf $G' = (V_G, E_G \cup E_{add})$ zawiera co najmniej m rozłącznych wierzchołkowo podgrafów izomorficznych z P

6.2 Definicje formalne

Definicja 11 (Rozszerzenie multigrafu)**.** Niech $G = (V_G, E_G)$ i $G' = (V_{G'}, E_{G'})$ będą multigrafami. Mówimy, że G' jest **rozszerzeniem** G, jeśli:

1. $V_G \subseteq V_{G'}$ (zbiór wierzchołków G jest podzbiorem wierzchołków G')
2. $E_G \subseteq E_{G'}$ (zbiór krawędzi G jest podzbiorem krawędzi G')

Uzasadnienie: Definicja jest naturalna i oparta na relacji inkluzji zbiorów. Zgodna z intuicją, że rozszerzenie grafu polega na dodaniu nowych wierzchołków i/lub krawędzi przy zachowaniu struktury oryginalnego grafu. Jest spójna z definicją podgrafu (G jest podgrafem G').

Definicja 12 (Koszt rozszerzenia)**.** Niech $G = (V_G, E_G)$ i $G' = (V_{G'}, E_{G'})$ będą multigrafami takimi, że G' jest rozszerzeniem G. **Kosztem rozszerzenia** $\gamma(G, G')$ nazywamy parę liczb naturalnych:

$$\gamma(G, G') = (|V_{G'} \setminus V_G|, |E_{G'} \setminus E_G|)$$

gdzie:

- $|V_{G'} \setminus V_G|$ to liczba dodanych wierzchołków
- $|E_{G'} \setminus E_G|$ to liczba dodanych krawędzi

Uzasadnienie: Koszt uwzględnia dwie podstawowe operacje rozszerzania grafu. W kontekście naszego algorytmu skupiamy się głównie na dodawaniu krawędzi, zakładając stałą liczbę wierzchołków ($V_G = V_{G'}$).

Definicja 13 (Porządek leksykograficzny na kosztach)**.** Dla dwóch kosztów (v_1, e_1) i (v_2, e_2) definiujemy porządek leksykograficzny:

$$(v_1, e_1) < (v_2, e_2) \iff v_1 < v_2 \vee (v_1 = v_2 \wedge e_1 < e_2)$$

Uzasadnienie: Porządek leksykograficzny priorytetyzuje minimalizację liczby dodanych wierzchołków, a następnie krawędzi.

Definicja 14 (Osadzenie k-wierzchołkowe)**. Osadzenie k-wierzchołkowe** grafu P w grafie G definiujemy przez parę (C, π), gdzie:

1. $C \subseteq V_G$ jest **k-kombinacją** - podzbiorem wierzchołków takim, że $|C| = k = |V_P|$

2. $\pi : V_P \to C$ jest **k-permutacją** - bijekcją mapującą wierzchołki P na wierzchołki C

Para (C, π) definiuje potencjalne osadzenie P w G poprzez podgraf indukowany przez C z odpowiednim mapowaniem wierzchołków.

Uzasadnienie: Formalizuje procedurę przeszukiwania przestrzeni możliwych osadzeń. Bezpośrednio odpowiada implementacji (kombinacje i permutacje w kodzie). Liczba możliwych osadzeń wynosi $\binom{n}{k} \times k!$.

Definicja 15 (Brakujące krawędzie dla osadzenia)**.** Niech $G = (V_G, E_G)$ i $P = (V_P, E_P)$ będą multigrafami, gdzie $|V_P| = k \leq |V_G| = n$. Niech (C, π) będzie osadzeniem k-wierzchołkowym P w G. **Zbiorem brakujących krawędzi** dla osadzenia (C, π) nazywamy multizbiór:

$$\Delta((C, \pi), G, P) = \{(\pi(u), \pi(v)) : u, v \in V_P\}$$

z krotnościami:

$$\text{mult}_\Delta(\pi(u), \pi(v)) = \max(0, A_P[u][v] - A_G[\pi(u)][\pi(v)])$$

gdzie A_P, A_G są macierzami sąsiedztwa odpowiednio grafów P i G.

Uzasadnienie: Umożliwia kwantyfikację odległości między potencjalnym osadzeniem a rzeczywistym izomorfizmem. Stanowi podstawę algorytmu konstrukcji minimalnego rozszerzenia. Uwzględnia krotności krawędzi (multigrafowość).

Definicja 16 (Minimalne rozszerzenie zawierające m kopii podgrafu P)**.** Niech $G = (V_G, E_G)$ i $P = (V_P, E_P)$ będą multigrafami, gdzie $|V_P| \leq |V_G|$, oraz niech $m \geq 1$ będzie liczbą naturalną. **Minimalnym rozszerzeniem** G zawierającym m kopii P nazywamy multigraf $G' = (V_{G'}, E_{G'})$ spełniający następujące warunki:

1. G' jest rozszerzeniem G (tj. $V_G \subseteq V_{G'}$, $E_G \subseteq E_{G'}$)

2. G' zawiera co najmniej m podgrafów parami rozłącznych wierzchołkowo, z których każdy jest izomorficzny z P

3. Koszt rozszerzenia $\gamma(G, G')$ jest minimalny w sensie porządku leksykograficznego wśród wszystkich rozszerzeń spełniających warunki 1 i 2

Uzasadnienie: Wymóg rozłączności wierzchołkowej m kopii zapewnia, że są to rzeczywiście odrębne podgrafy. W implementacji zakładamy $V_G = V_{G'}$ (nie dodajemy wierzchołków), więc minimalizujemy tylko liczbę dodanych krawędzi. Definicja jest operacyjna i pozwala na konstrukcję algorytmów.

6.3 Algorytm dokładny

6.3.1 Przegląd algorytmu

Algorytm oparty jest na pełnym przeszukiwaniu przestrzeni rozwiązań. Strategia polega na:

1. Wygenerowaniu wszystkich możliwych osadzeń grafu P w grafie G
2. Obliczeniu brakujących krawędzi dla każdego osadzenia
3. Rozważeniu wszystkich możliwych m-tuple osadzeń dla m kopii
4. Wybraniu osadzenia minimalizującego liczbę dodanych krawędzi

Algorytm składa się z dwóch głównych faz:

- **Faza 1:** Generowanie osadzeń i macierzy brakujących krawędzi
- **Faza 2:** Znajdowanie minimalnego rozszerzenia dla m kopii

6.3.2 Szczegółowy opis algorytmu

Faza 1: Generowanie osadzeń i macierzy brakujących krawędzi
Krok 1: Generowanie k-kombinacji wierzchołków G
Dla grafu G o n wierzchołkach generujemy wszystkie możliwe k-kombinacje wierzchołków, gdzie $k = |V_P|$. Każda kombinacja $C_j \subseteq V_G$ reprezentuje potencjalny zbiór wierzchołków, na które można zmapować graf P.

- *Wejście:* $G = (V_G, E_G)$, $|V_G| = n$, $k = |V_P|$
- *Wyjście:* Zbiór wszystkich k-kombinacji $\{C_1, C_2, \ldots, C_N\}$, gdzie $N = \binom{n}{k}$
- *Implementacja:* Funkcja `combinationsK()` generuje kombinacje iteracyjnie w porządku leksykograficznym
- *Złożoność:* $O\left(\binom{n}{k} \times k\right)$

Krok 2: Generowanie permutacji wierzchołków P
Dla grafu P o k wierzchołkach generujemy wszystkie możliwe permutacje. Każda permutacja $\pi_i : V_P \to V_P$ reprezentuje potencjalne uporządkowane mapowanie wierzchołków P na wierzchołki kombinacji C_j.

- *Wejście:* $P = (V_P, E_P)$, $|V_P| = k$

- *Wyjście:* Zbiór wszystkich permutacji $\{\pi_1, \pi_2, \ldots, \pi_M\}$, gdzie $M = k!$
- *Implementacja:* Funkcja `permutations()` wykorzystuje algorytm Heapa
- *Złożoność:* $O(k! \times k)$

Krok 3: Obliczanie brakujących krawędzi

Dla każdej pary (C_j, π_i) obliczamy listę krawędzi, które należy dodać do G, aby podgraf indukowany przez C_j z mapowaniem π_i był izomorficzny z P.

Dla każdej pary wierzchołków $(u, v) \in V_P \times V_P$:

- Obliczamy obrazy: $u' = C_j[\pi_i(u)]$, $v' = C_j[\pi_i(v)]$
- Liczba krawędzi w P: $e_P = A_P[\pi_i(u)][\pi_i(v)]$
- Liczba krawędzi w G: $e_G = A_G[u'][v']$
- Brakujące krawędzie: $\Delta = \max(0, e_P - e_G)$
- Dodajemy Δ kopii krawędzi (u', v') do listy brakujących krawędzi

Wynik zapisujemy w macierzy missingEdgesMatrix$[i][j]$ jako listę brakujących krawędzi dla permutacji i i kombinacji j.

- *Złożoność pojedynczego obliczenia:* $O(k^2)$
- *Całkowita złożoność:* $O\left(\binom{n}{k} \times k! \times k^2\right)$
- *Pamięć:* $O\left(\binom{n}{k} \times k! \times k^2\right)$ w najgorszym przypadku

Faza 2: Znajdowanie minimalnego rozszerzenia dla m kopii

Krok 4: Generowanie m-kombinacji osadzeń

Generujemy wszystkie możliwe sposoby wyboru m różnych kombinacji wierzchołków spośród $N = \binom{n}{k}$ dostępnych kombinacji. Każda taka m-kombinacja reprezentuje wybór m rozłącznych wierzchołkowo podzbiorów dla m kopii grafu P.

- *Liczba m-kombinacji:* $\binom{\binom{n}{k}}{m}$
- *Złożoność:* $O\left(\binom{\binom{n}{k}}{m} \times m\right)$

Krok 5: Generowanie m-krotek permutacji

Dla każdej m-kombinacji podzbiorów, generujemy wszystkie możliwe m-krotki permutacji. Każda m-krotka $(i_1, i_2, \ldots, i_m)$ określa, jaką permutację stosujemy dla każdej z m kopii.

- *Liczba m-krotek:* $(k!)^m$
- *Implementacja:* Funkcja `productSequences()` generuje produkt kartezjański
- *Złożoność:* $O((k!)^m \times m)$

Krok 6: Obliczanie unii zbiorów krawędzi

Dla każdej konfiguracji (m-kombinacja podzbiorów, m-krotka permutacji) obliczamy minimalny zbiór krawędzi potrzebny do stworzenia m kopii grafu P.

Kluczowa obserwacja: jeśli wielokrotne kopie wymagają tej samej krawędzi (u, v) z krotnościami $k_1, k_2, \ldots, k_m$, wystarczy dodać $\max(k_1, k_2, \ldots, k_m)$ kopii tej krawędzi, ponieważ krawędzie mogą być współdzielone między kopiami.

Algorytm:

1. Inicjalizujemy mapę częstości: `edgeFrequencyMap = {}`
2. Dla każdej z m kopii ($t = 1, \ldots, m$):
   (a) Pobieramy brakujące krawędzie dla kopii t
   (b) Tworzymy lokalną mapę częstości krawędzi dla tej kopii
   (c) Aktualizujemy globalną mapę: dla każdej krawędzi e, ustawiamy `edgeFrequencyMap[e] = max(edgeFrequencyMap[e], localFrequency[e])`
3. Konwertujemy mapę częstości na listę krawędzi
4. Jeśli rozmiar listy jest mniejszy niż dotychczasowe minimum, aktualizujemy rozwiązanie

- *Złożoność:* $O\left(\binom{n}{k}_m \times (k!)^m \times m \times k^2\right)$

6.3.3 Pseudokod algorytmu

Algorithm 1 MinimalGraphExtension(G, P, m)

Require: $G = (V_G, E_G)$ - multigraf "duży", $|V_G| = n$
Require: $P = (V_P, E_P)$ - multigraf "mały", $|V_P| = k$
Require: m - liczba wymaganych kopii P
Ensure: Lista krawędzi do dodania do G

```
// Faza 1: Generowanie osadzeń
combinations ← GenerateCombinations(V_G, k)
indexToSubset ← IndexMap(combinations)
permutations ← GeneratePermutations(V_P)
indexToPermutation ← IndexMap(permutations)

missingEdgesMatrix ← Array[|indexToPermutation|][|indexToSubset|]

for i ← 0 to |indexToPermutation| − 1 do
    for j ← 0 to |indexToSubset| − 1 do
        π ← indexToPermutation[i]
        C ← indexToSubset[j]
        missingEdgesMatrix[i][j] ← [ ]
        for u ← 0 to k − 1 do
            for v ← 0 to k − 1 do
                u' ← C[π[u]]
                v' ← C[π[v]]
                e_P ← A_P[π[u]][π[v]]
                e_G ← A_G[u'][v']
                Δ ← max(0, e_P − e_G)
                for t ← 0 to Δ − 1 do
                    missingEdgesMatrix[i][j].append((u', v'))
                end for
            end for
        end for
    end for
end for

// Faza 2: Znajdowanie minimalnego rozszerzenia
mCombinations ← GenerateCombinations(indexToSubset.keys, m)
minimalEdges ← null
minimalSize ← ∞
```



```
for each {j_1, ..., j_m} in mCombinations do
    for each (i_1, ..., i_m) in ProductSeq(indexToPermutation.keys, m) do
        edgeFreqMap ← {}
        for t ← 0 to m − 1 do
            missingEdges ← missingEdgesMatrix[i_t][j_t]
            localFreq ← {}
            for each e in missingEdges do
                localFreq[e] ← localFreq[e] + 1
```

6.4 Dowód poprawności algorytmu

[Poprawność algorytmu MinimalGraphExtension] Algorytm MinimalGraphExtension zwraca minimalną listę krawędzi do dodania do G, aby zawierał m rozłącznych wierzchołkowo kopii P.

Dowód. Dowód podzielimy na trzy części: kompletność, poprawność i minimalność.

Część 1: Kompletność (algorytm znajduje rozwiązanie, jeśli istnieje)

Załóżmy, że istnieje rozszerzenie G' grafu G zawierające m rozłącznych wierzchołkowo kopii P, osiągnięte przez dodanie zbioru krawędzi E_{add}.

1. Dla każdej z m kopii P w G' istnieje:
   - k-kombinacja wierzchołków $C_t \subseteq V_G$ $(t = 1, \ldots, m)$
   - Permutacja $\pi_t : V_P \to V_P$
   - Takie że podgraf G' indukowany przez C_t z odpowiednim mapowaniem jest izomorficzny z P
2. Kombinacje C_t są rozłączne ($C_i \cap C_j = \emptyset$ dla $i \neq j$), ponieważ kopie są rozłączne wierzchołkowo
3. Algorytm generuje wszystkie możliwe m-kombinacje k-podzbiorów (linia 30)
4. Dla każdej m-kombinacji, algorytm generuje wszystkie możliwe m-krotki permutacji (linia 31)
5. Zatem algorytm rozważy kombinację $\{C_1, \ldots, C_m\}$ i krotkę permutacji $(\pi_1, \ldots, \pi_m)$ odpowiadającą rzeczywistemu rozwiązaniu
6. Dla tej kombinacji i krotki permutacji, algorytm obliczy dokładnie te same krawędzie, które są w E_{add} (linie 33-48)

Część 2: Poprawność (dodane krawędzie są wystarczające)

Dla dowolnej m-kombinacji podzbiorów i m-krotki permutacji rozważanej przez algorytm:

1. Dla każdej z m kopii (linie 33-42):
   - Algorytm oblicza brakujące krawędzie zapisane wcześniej w $missingEdgesMatrix$
   - Te krawędzie są dokładnie tymi, których brakuje do stworzenia izomorfizmu (z Fazy 1)

2. Operacja maksimum na krotnościach (linia 41) zapewnia:

   - Jeśli wielokrotne kopie potrzebują tej samej krawędzi (u, v) z krotnościami $k_1, k_2, \ldots$, dodajemy $\max(k_1, k_2, \ldots)$ kopii
   - To jest *wystarczające*, bo krawędzie mogą być współdzielone między kopiami
   - To jest *konieczne*, bo każda kopia wymaga odpowiedniej krotności

3. Po dodaniu krawędzi z $addedEdges$ do G:

   - Każda z m kopii ma wystarczającą liczbę krawędzi między każdą parą wierzchołków
   - Każda kopia jest izomorficzna z P

Część 3: Minimalność (algorytm znajduje minimum)

1. Algorytm przeszukuje wszystkie możliwe sposoby osadzenia m kopii P w G (linie 30-31)
2. Dla każdego sposobu oblicza minimalną liczbę krawędzi potrzebnych do realizacji tego osadzenia (linie 33-48)
3. Wybiera osadzenie wymagające najmniejszej liczby krawędzi (linie 49-52)
4. Nie istnieje sposób osadzenia m kopii P w G wymagający mniej krawędzi, bo wszystkie sposoby zostały rozważone

Zatem algorytm jest poprawny - zwraca minimalną liczbę krawędzi wystarczających do stworzenia m rozłącznych wierzchołkowo kopii P w G. □

6.5 Analiza złożoności obliczeniowej

6.5.1 Złożoność czasowa

Oznaczmy:

- $n = |V_G|$ - liczba wierzchołków dużego grafu G
- $k = |V_P|$ - liczba wierzchołków małego grafu P
- m - liczba wymaganych kopii P w G

Rozbicie na fazy:

1. **Faza 1a - Generowanie kombinacji:** $O\left(\binom{n}{k} \times k\right)$
   - Liczba kombinacji: $\binom{n}{k}$
   - Koszt generowania jednej kombinacji: $O(k)$
2. **Faza 1b - Generowanie permutacji:** $O(k! \times k)$
   - Liczba permutacji: $k!$
   - Koszt generowania jednej permutacji: $O(k)$
3. **Faza 1c - Obliczanie missingEdgesMatrix:** $O\left(\binom{n}{k} \times k! \times k^2\right)$
   - Dla każdej z $\binom{n}{k}$ kombinacji
   - Dla każdej z $k!$ permutacji
   - Porównanie k^2 par krawędzi
4. **Faza 2a - Generowanie m-kombinacji osadzeń:** $O\left(\binom{\binom{n}{k}}{m} \times m\right)$
   - Liczba m-kombinacji: $\binom{\binom{n}{k}}{m}$
5. **Faza 2b - Główna pętla przeszukiwania:**

$$O\left(\binom{\binom{n}{k}}{m} \times (k!)^m \times m \times k^2\right)$$

   - Dla każdej m-kombinacji podzbiorów: $\binom{\binom{n}{k}}{m}$
   - Dla każdej m-krotki permutacji: $(k!)^m$
   - Dla każdej z m kopii: m
   - Obliczanie częstości krawędzi: $O(k^2)$ dla jednej kopii

Całkowita złożoność czasowa:

$$T(n,k,m) = O\left(\binom{n}{k} \times k! \times k^2\right) + O\left(\binom{\binom{n}{k}}{m} \times (k!)^m \times m \times k^2\right)$$

Dominujący składnik dla małych m (gdy $m \ll \binom{n}{k}$):

$$T(n,k,m) = O\left(\binom{n}{k} \times k! \times k^2\right)$$

Dominujący składnik dla większych m:

$$T(n,k,m) = O\left(\binom{\binom{n}{k}}{m} \times (k!)^m \times m \times k^2\right)$$

Oszacowania asymptotyczne:

- $\binom{n}{k} = O\left(\frac{n^k}{k!}\right)$ - wielomianowe względem n dla stałego k
- $k!$ - silniowe względem k
- $\binom{\binom{n}{k}}{m} \approx O((n^k)^m)$ dla dużych n
- $(k!)^m$ - wykładnicze względem m

Przykład obliczeniowy:
Dla $k = 3$, $n = 10$, $m = 2$:

- $\binom{10}{3} = 120$
- $3! = 6$
- $\binom{120}{2} = 7140$
- $(3!)^2 = 36$
- Oszacowanie operacji Fazy 1: $120 \times 6 \times 9 \approx 6{,}480$
- Oszacowanie operacji Fazy 2: $7140 \times 36 \times 2 \times 9 \approx 4{,}630{,}080$
- Łącznie: $\approx 4{,}6$ miliona operacji

6.5.2 Złożoność pamięciowa

Główne struktury danych:

1. **missingEdgesMatrix:** $O\left(\binom{n}{k} \times k! \times k^2\right)$
   - Tablica 2D o wymiarach $k! \times \binom{n}{k}$
   - Każda komórka zawiera listę brakujących krawędzi (w najgorszym $O(k^2)$ krawędzi)
2. **indexToSubset:** $O\left(\binom{n}{k} \times k\right)$

- Przechowuje $\binom{n}{k}$ kombinacji, każda długości k

3. **indexToPermutation:** $O(k! \times k)$
   - Przechowuje $k!$ permutacji, każda długości k
4. **Zmienne tymczasowe w głównej pętli:** $O(k^2)$

Całkowita złożoność pamięciowa:

$$S(n, k, m) = O\left(\binom{n}{k} \times k! \times k^2\right)$$

Uwaga: Złożoność pamięciowa jest niezależna od m (nie przechowujemy wszystkich m-krotek, generujemy je leniwie).

6.5.3 Charakterystyka algorytmu

Klasa złożoności:

- Problem jest **NP-trudny** (redukcja z problemu izomorfizmu podgrafów)
- Algorytm dokładny ma złożoność **wykładniczą** względem k (ze względu na $k!$)
- Algorytm ma złożoność **wielomianowo-wykładniczą** względem m

Praktyczne ograniczenia:

- Algorytm jest wykonalny dla małych wartości k ($k \leq 6 - 7$) i $n \leq 20$
- Dla większych wartości k lub n algorytm staje się niepraktyczny
- Wartość m ma mniejszy wpływ na czas wykonania niż k (dla małych m)

6.6 Przykład działania algorytmu

Dane wejściowe:

Graf G z 8 wierzchołkami (macierz sąsiedztwa 8×8):

$$A_G = \begin{pmatrix} 2 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Graf P z 3 wierzchołkami (macierz sąsiedztwa 3×3):

$$A_P = \begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

Liczba wymaganych kopii: $m = 2$
Faza 1: Generowanie osadzeń
Krok 1: Generowanie 3-kombinacji z 8 wierzchołków

- Liczba kombinacji: $\binom{8}{3} = 56$
- Przykłady: $C_0 = \{0, 1, 2\}$, $C_1 = \{0, 1, 3\}$, $\ldots$, $C_{55} = \{5, 6, 7\}$

Krok 2: Generowanie permutacji wierzchołków P

- Liczba permutacji: $3! = 6$
- Wszystkie permutacje: $\pi_0 = [0, 1, 2]$, $\pi_1 = [0, 2, 1]$, $\pi_2 = [1, 0, 2]$, $\pi_3 = [1, 2, 0]$, $\pi_4 = [2, 0, 1]$, $\pi_5 = [2, 1, 0]$

Krok 3: Obliczanie brakujących krawędzi
Dla $C_0 = \{0, 1, 2\}$ i $\pi_0 = [0, 1, 2]$:

- Podgraf G indukowany przez $\{0, 1, 2\}$:

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

- P z permutacją π_0 (bez zmian):

$$\begin{pmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{pmatrix}$$

- Brakujące krawędzie: missingEdgesMatrix[0][0] = [ ] (puste - grafy identyczne!)

Dla $C_{55} = \{5, 6, 7\}$ i $\pi_0 = [0, 1, 2]$:

- Podgraf G indukowany przez $\{5, 6, 7\}$:

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

- Brakujące krawędzie: missingEdgesMatrix[0][55] = [(5, 5), (5, 5), (5, 7), (6, 6), (6, 6), (7, 5), (7, 7), (7

Faza 2: Znajdowanie 2 kopii
Krok 4-6: Rozważamy 2-kombinację $\{C_0, C_{55}\} = \{\{0, 1, 2\}, \{5, 6, 7\}\}$
Te kombinacje są rozłączne wierzchołkowo ✓
Dla 2-krotki permutacji (π_0, π_0):

- Kopia 1: missingEdgesMatrix[0][0] = [ ] (0 krawędzi)
- Kopia 2: missingEdgesMatrix[0][55] = [. . .] (8 krawędzi)
- edgeFrequencyMap zawiera 8 krawędzi z Kopii 2
- Rozmiar listy: 8

Po przeszukaniu wszystkich konfiguracji, algorytm może znaleźć lepsze rozwiązanie (np. dwie inne kombinacje wymagające łącznie mniej krawędzi).

Wynik przykładowy:

```
Minimalna liczba krawędzi do dodania: 0 (lub inna wartość)
Dodane krawędzie: []
Osadzenie kopii 1: wierzchołki {0,1,2}
Osadzenie kopii 2: wierzchołki {5,6,7} (po ewentualnym dodaniu krawędzi)
```

6.7 Optymalizacje i możliwe ulepszenia

6.7.1 Optymalizacje implementacyjne

1. **Wczesne przerywanie (pruning)**
   - Jeśli podczas budowania $addedEdgesList$ rozmiar przekroczy dotychczasowe minimum, przerwij obliczenia dla tej konfiguracji
   - Oszczędność czasu w pesymistycznych przypadkach
2. **Wykorzystanie symetrii grafu**
   - Jeśli graf P ma symetrie (automorfizmy), wiele permutacji da identyczne wyniki
   - Można zredukować liczbę rozważanych permutacji poprzez grupowanie klas równoważności
3. **Sortowanie kombinacji według óbiecującychćech**
   - Najpierw rozważaj kombinacje, które już mają wiele krawędzi odpowiadających P
   - Może prowadzić do znalezienia dobrego rozwiązania wcześniej
4. **Przetwarzanie równoległe**
   - Obliczenia dla różnych m-kombinacji są niezależne
   - Możliwość równoległego przetwarzania na wielu rdzeniach/maszynach

6.7.2 Heurystyki i algorytmy aproksymacyjne

1. Algorytm zachłanny (Greedy)

Idea: Zamiast rozważać wszystkie możliwe m-kombinacje osadzeń, wybieraj zachłannie następne najlepsze osadzenie.

Algorytm:

- Dla pierwszej kopii P: znajdź osadzenie wymagające najmniej krawędzi
- Dodaj te krawędzie do G
- Dla kolejnych kopii: znajdź osadzenie wymagające najmniej nowych krawędzi (biorąc pod uwagę już dodane)

- Powtórz m razy

Złożoność: $O\left(m \times \binom{n}{k} \times k! \times k^2\right)$
Jakość: Nie gwarantuje optymalności, ale może dać dobre przybliżenie w krótszym czasie.

2. Algorytm genetyczny
Idea: Ewolucyjne poszukiwanie dobrego rozwiązania.
Komponenty:

- **Populacja:** Zbiór kandydatów rozwiązań (m-krotki osadzeń)
- **Funkcja przystosowania:** Liczba krawędzi do dodania (minimalizowana)
- **Operatory:** Krzyżowanie (wymiana osadzeń między rozwiązaniami), mutacja (losowa zmiana osadzenia)
- **Selekcja:** Wybór najlepszych rozwiązań do następnego pokolenia

Zalety: Możliwość znalezienia dobrych rozwiązań dla dużych instancji.
Wady: Brak gwarancji optymalności, wymaga tuningu parametrów.

3. Symulowane wyżarzanie (Simulated Annealing)
Idea: Iteracyjne ulepszanie rozwiązania z możliwością akceptacji gorszych rozwiązań.
Algorytm:

- Start: losowa m-krotka osadzeń
- Iteracyjnie: modyfikuj losowo jedno osadzenie
- Akceptuj jeśli poprawia rozwiązanie lub z prawdopodobieństwem zależnym od "temperatury"
- "Temperatura"maleje z czasem, redukując akceptację gorszych rozwiązań

Zalety: Pozwala na wyjście z lokalnych minimów.

4. Programowanie całkowitoliczbowe (ILP)
Idea: Sformułuj problem jako program całkowitoliczbowy.
Zmienne:

- $x_{ij} \in \{0, 1\}$ - czy osadzenie i jest wybrane dla kopii j
- $y_e \in \mathbb{N}_0$ - liczba kopii krawędzi e do dodania

Ograniczenia:

- Każda kopia musi mieć dokładnie jedno osadzenie: $\sum_i x_{ij} = 1$ dla każdego j
- Osadzenia muszą być rozłączne wierzchołkowo
- Dla każdej krawędzi e: $y_e \geq \max_j\{\Delta_{ij}(e) \cdot x_{ij}\}$ gdzie $\Delta_{ij}(e)$ to krotność e w brakujących krawędziach osadzenia i dla kopii j

Cel: Minimalizuj $\sum_e y_e$
Zalety: Optymalne rozwiązanie (jeśli solver zakończy się w rozsądnym czasie).
Wady: Może być wolne dla dużych instancji.

6.8 Podsumowanie

W niniejszej sekcji przedstawiono kompleksowe podejście do problemu minimalnego rozszerzenia multigrafu zawierającego m kopii podgrafu wzorcowego:

- **Sformułowano problem** i uzasadniono jego praktyczne znaczenie
- **Zdefiniowano formalnie** kluczowe pojęcia: rozszerzenie, koszt rozszerzenia, osadzenie k-wierzchołkowe, brakujące krawędzie, minimalne rozszerzenie zawierające m kopii
- **Opracowano algorytm dokładny** oparty na pełnym przeszukiwaniu przestrzeni rozwiązań
- **Udowodniono poprawność** algorytmu (kompletność, poprawność, minimalność)
- **Przeprowadzono szczegółową analizę złożoności**:
  - Złożoność czasowa: $O\left(\binom{n}{k} \times k! \times k^2 + \binom{\binom{n}{k}}{m} \times (k!)^m \times m \times k^2\right)$
  - Złożoność pamięciowa: $O\left(\binom{n}{k} \times k! \times k^2\right)$
  - Problem jest NP-trudny
- **Zaprezentowano przykład działania** algorytmu na konkretnych danych
- **Zaproponowano optymalizacje** implementacyjne i algorytmy aproksymacyjne (zachłanny, genetyczny, simulated annealing, ILP)

Algorytm dokładny jest praktyczny dla małych wartości k ($k \leq 6 - 7$) i $n \leq 20$. Dla większych instancji zaleca się stosowanie algorytmów aproksymacyjnych lub heurystyk.

7 Bibliografia

Literatura

[1] Fred DePiero and David Krout. An algorithm using length-r paths to approximate subgraph isomorphism. *Pattern Recognition Letters*, 24(1):33–46, 2003.