Wydział Matematyki i Nauk Informacyjnych
POLITECHNIKA WARSZAWSKA

Teoria Algorytmów i Obliczeń

Odległość między grafami

Piotr Kryczka, Arkadiusz Noster, Adam Ryl

29 listopada 2021

Spis treści

1 Sformułowanie problemu

1.1 Wstęp

Niniejszy dokument poświęcony został rozwiązaniu problemu odległości pomiędzy grafami w dwóch podejściach:

- przy pomocy zaproponowanej metryki i algorytmu dokładnego
- za pomocą algorytmu aproksymacyjnego.

Wiemy, że problem obliczania odległości między grafami jest NP-trudny, a zatem niestety nie istnieje algorytm działający w rozsądnym czasie dla dużych przypadków.

Na wejściu mamy dwa grafy G_1 i G_2 reprezentowane przez macierze sąsiedztwa odpowiednio $M_1 \in M^{m \times m}$ i $M_2 \in M^{n \times n}$, gdzie $m, n \in N$ i $m, n > 0$.

W zadaniu rozważamy grafy skierowane, zatem wyżej zdefiniowane macierze nie muszą być symetryczne, a wartości należą do zbioru $\{0, 1\}$, przy czym:

- $M[i, j] = 0$, jeśli nie istnieje krawędź z wierzchołka reprezentowanego przez i-ty wiersz do wierzchołka reprezentowanego przez j-tą kolumnę,
- $M[i, j] = 1$, jeśli taka krawędź istnieje.

1.2 Metryka

W proponowanych algorytmach zastosowany zostanie sposób mierzenia odległości między grafami oparty na maksymalnym wspólnym podgrafie [1].

Oznaczmy graf jako parę zbioru wierzchołków V i zbioru krawędzi E - $G = (V, E)$. **Podgrafem** grafu $G = (V, E)$ będziemy nazywać graf $S = (V_S, E_S)$ taki, że:

- $V_S \subseteq V$,
- $E_S = E \cap (V_S \times V_S)$.

Jeśli graf S jest podgrafem G, dla uproszczenia zapisu będziemy oznaczać tę zależność jako $S \subseteq G$.

Przyjmujemy, że **graf G jest wspólnym podgrafem grafów G_1 i G_2** jeśli $G \subseteq G_1$ oraz $G \subseteq G_2$, a zatem pewien podgraf G_1 oraz pewien podgraf G_2 są grafami izomorficznymi.

Powiemy, że **dwa grafy G i H są izomorficzne** jeśli istnieje bijekcja („przeetykietowani") wierzchołków grafu H wierzchołkom grafu G spełniająca następujący warunek: jeśli jakieś dwa wierzchołki są połączone krawędzią w jednym z tych grafów, to odpowiadające im wierzchołki w drugim grafie również łączy krawędź.

Konkretny **wspólny podgraf G jest maksymalny** dla grafów G_1 i G_2, jeśli nie istnieje wspólny podgraf G' mający więcej wierzchołków niż G.

Maksymalny wspólny podgraf dwóch grafów G_1 i G_2 będziemy oznaczać jako $mcs(G_1, G_2)$ (ang. *maximal common subgraph*). Zauważmy też, że $G = mcs(G_1, G_2)$ niekoniecznie musi być jednoznaczny, ponieważ jego wielkość określamy poprzez liczbę wierzchołków (oznaczenie: $|V_G|$ lub, dla uproszczenia zapisu, $|G|$).

Odległość pomiędzy dwoma niepustymi grafami obliczymy ze wzoru:

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{max(|G_1|, |G_2|)}.$$

W kolejnym rozdziale udowodnimy, że powyższa definicja odległości jest metryką, czyli spełnia następujące warunki:

1. $d(G_1, G_2) = 0 \iff G_1 = G_2$,
2. $d(G_1, G_2) = d(G_2, G_1)$,
3. $d(G_1, G_2) + d(G_2, G_3) \geqslant d(G_1, G_3)$ (tzw. nierówność trójkąta).

1.3 Dowód poprawności metryki

Pokażemy, że:

1. $0 \leqslant d(G_1, G_2) \leqslant 1$,
2. $d(G_1, G_2) = 0 \iff G_1$ i G_2 są izomorficzne,
3. $d(G_1, G_2) = d(G_2, G_1)$,
4. $d(G_1, G_3) \leqslant d(G_1, G_2) + d(G_2, G_3)$.

Dowód.

Właściwości 1., 2. i 3. wynikają bezpośrednio z definicji odległości w poprzednim rozdziale. Poniżej skupimy się jedynie na udowodnieniu nierówności trójkąta.

Przypadek A.

Przyjmijmy, że grafy $mcs(G_1, G_2)$ oraz $mcs(G_2, G_3)$ są rozłączne, czyli największy wspólny podgraf grafów $mcs(G_1, G_2)$ oraz $mcs(G_2, G_3)$ jest pusty.

Niech $m_{12} = mcs(G_1, G_2)$, analogicznie $m_{23} = mcs(G_2, G_3)$ i $m_{13} = mcs(G_1, G_3)$. Wtedy prawdziwa jest następująca nierówność:

$$m_{12} + m_{23} \leqslant |G_2|. \tag{1}$$

Nierówność trójkąta dla zdefiniowanej odległości prezentuje się następująco:

$$1 - \frac{m_{12}}{max(|G_1|, |G_2|)} + 1 - \frac{m_{23}}{max(|G_2|, |G_3|)} \geqslant 1 - \frac{m_{13}}{max(|G_1|, |G_3|)}. \tag{2}$$

Pokażemy, że lewa strona powyższej nierówności jest zawsze większa lub równa 1, co po przekształceniach daje:

$$max(|G_1|, |G_2|) \cdot max(|G_2|, |G_3|) \geqslant m_{12} \cdot max(|G_2|, |G_3|) + m_{23} \cdot max(|G_1|, |G_2|). \tag{3}$$

W tym celu rozpatrzymy kilka podprzypadków.

Podprzypadek A.1 $|G_1| \geqslant |G_2| \geqslant |G_3|$

Nierówność (3) przyjmuje wtedy postać:

$$|G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_2| + m_{23} \cdot |G_1|. \tag{4}$$

Wykorzystując nierówność (1) dostajemy:

$$|G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_1| + m_{23} \cdot |G_1| \geqslant m_{12} \cdot |G_2| + m_{23} \cdot |G_1|.$$

Podprzypadek A.2 $|G_1| \geqslant |G_3| \geqslant |G_2|$

Nierówność (3) przyjmuje wtedy postać:

$$|G_1| \cdot |G_3| \geqslant m_{12} \cdot |G_3| + m_{23} \cdot |G_1|. \tag{5}$$

Ponownie wykorzystując nierówność (1) dostajemy:

$$|G_1| \cdot |G_3| \geqslant |G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_1| + m_{23} \cdot |G_1| \geqslant m_{12} \cdot |G_3| + m_{23} \cdot |G_1|.$$

Pozostałe podprzypadki, korzystające kolejno z założeń:

- $|G_2| \geqslant |G_1| \geqslant |G_3|$,
- $|G_2| \geqslant |G_3| \geqslant |G_1|$,
- $|G_3| \geqslant |G_1| \geqslant |G_2|$,

- $|G_3| \geqslant |G_2| \geqslant |G_1|$

można udowodnić w sposób analogiczny.

Przypadek B.

Przyjmijmy, że największy wspólny podgraf $mcs(G_1, G_2)$ oraz $mcs(G_2, G_3)$ nie jest pusty. Oznaczmy:

$$m = |mcs(mcs(G_1, G_2), mcs(G_2, G_3))| > 0.$$

Wynika z tego, że istnieje również maksymalny wspólny podgraf grafów G_1 i G_3, którego rozmiar jest większy lub równy m. Idąc dalej, możemy dojść do następujących wniosków:

$$m_{12} + m_{23} - m \leqslant |G_2|, \quad m \leqslant m_{12}, \quad m \leqslant m_{23}. \tag{6}$$

Pokażemy, że:

$$1 - \frac{m_{12}}{max(|G_1|, |G_2|)} + 1 - \frac{m_{23}}{max(|G_2|, |G_3|)} \geqslant 1 - \frac{m}{max(|G_1|, |G_3|)}, \tag{7}$$

co ostatecznie będzie implikować zachodzenie nierówności trójkąta dla zdefiniowanej odległości, czyli również poprawność metryki.

Oczywiście powyższa nierówność po przekształceniach jest równoważna:

$$\begin{aligned} max(|G_1|, |G_2|) \cdot max(|G_2|, |G_3|) \cdot max(|G_1|, |G_3|) \geqslant\ & m_{12} \cdot max(|G_2|, |G_3|) \cdot max(|G_1|, |G_3|) \\ & + m_{23} \cdot max(|G_1|, |G_2|) \cdot max(|G_1|, |G_3|), \\ & - m \cdot max(|G_1|, |G_2|) \cdot max(|G_2|, |G_3|). \end{aligned} \tag{8}$$

Ponownie rozpatrzmy kilka podprzypadków.

Podprzypadek B.1 $|G_1| \geqslant |G_2| \geqslant |G_3|$

Nierówność (8) przyjmuje postać:

$$|G_1| \cdot |G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_1| \cdot |G_2| + m_{23} \cdot |G_1| \cdot |G_1| - m \cdot |G_1| \cdot |G_2|,$$

co może zostać uproszczone do postaci:

$$|G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_2| + m_{23} \cdot |G_1| - m \cdot |G_2| = (m_{12} - m) \cdot |G_2| + m_{23} \cdot |G_1|. \tag{9}$$

Z nierówności (6) dostajemy:

$$|G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_1| + m_{23} \cdot |G_1| - m \cdot |G_1| = (m_{12} - m) \cdot |G_1| + m_{23} \cdot |G_1|, \tag{10}$$

z czego dostajemy nierówność (9), ponieważ $|G_1| \geqslant |G_2|$ oraz $|m_{12}| \geqslant m$.

Podprzypadek B.2 $|G_1| \geqslant |G_3| \geqslant |G_2|$

Nierówność (8) przyjmuje postać:

$$|G_1| \cdot |G_1| \cdot |G_3| \geqslant m_{12} \cdot |G_1| \cdot |G_3| + m_{23} \cdot |G_1| \cdot |G_1| - m \cdot |G_1| \cdot |G_3|,$$

co może zostać uproszczone do postaci:

$$|G_1| \cdot |G_3| \geqslant m_{12} \cdot |G_3| + m_{23} \cdot |G_1| - m \cdot |G_3|. \tag{11}$$

Ponownie z nierówności (6) (i analogicznie jak w podprzypadku B.1) dostajemy:

$$|G_1| \cdot |G_3| \geqslant |G_1| \cdot |G_2| \geqslant m_{12} \cdot |G_1| + m_{23} \cdot |G_1| - m \cdot |G_1| \geqslant m_{12} \cdot |G_3| + m_{23} \cdot |G_1| - m \cdot |G_3|. \tag{12}$$

Pozostałe podprzypadki ponownie można udowodnić w sposób analogiczny.

Powyżej udowodniono nierówność trójkąta dla odległości między grafami zdefiniowanej jako

$$d(G_1, G_2) = 1 - \frac{|mcs(G_1, G_2)|}{max(|G_1|, |G_2|)}.$$

Co za tym idzie udowodniono, że **tak zdefiniowana odległość jest poprawną metryką**.

2 Algorytm dokładny

2.1 Opis algorytmu oraz analiza poprawności

Bez straty ogólności przyjęto, że $|G_1| \leqslant |G_2|$. W celu wyznaczenia liczby wierzchołków największego wspólnego podgrafu grafów G_1 i G_2 rozpatrzone zostaną wszystkie możliwe podgrafy grafu G_1. Podczas rozpatrywania każdego podgrafu zostanie sprawdzone, czy graf G_2 zawiera dany podgraf. Wybrany zostanie, który ma najwięcej wierzchołków.

Aby rozwiązać ten problem skorzystano z metody backtrackingu (algorytmu z nawrotami). Niech $|G_1| = n$. Zatem należy sprawdzić 2^n przypadków, poprzez wybieranie lub odrzucanie kolejnych z n wierzchołków.

Funkcja sprawdza wszystkie przypadki, wywołując w sposób rekurencyjny samą siebie:

- dodając w i-tym wywołaniu i-ty wierzchołek,
- nie dodając i-tego wierzchołka.

W przypadku gdy i-ty wierzchołek jest dodawany do listy wierzchołków podgrafu, będącego kandydatem na największy wspólny podgraf, sprawdzane jest przy pomocy pewnej funkcji (opisanej niżej), czy graf G_2 zawiera dany podgraf. Jeżeli nie, to kolejne podgrafy z tym wierzchołkiem nie są już sprawdzane, ponieważ dodając kolejne wierzchołki i tak nie zostaną uzyskane podgrafy grafu G_2. Warunkiem stopu jest sytuacja, gdy głębokość wywołań funkcji wynosi n.

Aby uzyskać liczbę wierzchołków największego wspólnego podgrafu, przy każdym wywołaniu funkcji zwracana jest liczba wierzchołków największego spośród podgrafów składających się z dodanych przy wywołaniu wierzchołków.

Poniżej przedstawiono pseudokod odpowiadający wyżej opisanemu algorytmowi.

```
1  GetMaximumCommonSubgraphVertices(G1, G2, Index, ConsideredVertices):
2      // Stop condition
3      if Index == G1.Length:
4          return ConsideredVertices
5
6      // Vertex G1[Index] is not considered
7      CandidateVertices1 = GetMaximumCommonSubgraphVertices(G1, G2,
8                               Index + 1, ConsideredVertices)
9
10     // Vertex G1[Index] is considered
11     ConsideredVertices.Push(Index)
12     if IsCommonSubgraph(G1, G2, ConsideredVertices):
13         CandidateVertices2 = GetMaximumCommonSubgraphVertices(G1, G2,
14                                  Index + 1, ConsideredVertices)
15     else:
16         CandidateVertices2 = Vertices.EmptyList
17     ConsideredVertices.Pop()
18
19     return (CandidateVertices1.Length > CandidateVertices2.Length ?
                                       CandidateVertices1 : CandidateVertices2)
```

Warto wspomnieć, że funkcję tę należy wywołać w następujący sposób:

$$GetMaximumCommonSubgraphVertices(G1, G2, 0, Vertices.EmptyList),$$

gdzie przyjęto, że:

- $G1$ i $G2$ to macierze sąsiedztwa grafów wejściowych,
- 0 - to początkowy stan iteratora niezbędnego do zakończenia obliczeń (poprzez warunek stopu),
- lista wierzchołków (początkowo pusta) będzie zawierać informację, które z wierzchołków G_1 mają być rozpatrywane przy danym wywołaniu.

Stwierdzenie, czy graf G_2 zawiera konkretny podgraf grafu G_1 nastąpi po sprawdzeniu wszystkich możliwych permutacji wierzchołków grafu G_2, ograniczając je do liczby wierzchołków podgrafu grafu G_1.

2.2 Analiza złożoności

Niech $|G_1| = n$. Wtedy sprawdzenie wszystkich możliwych podgrafów grafu G_1 ma złożoność $O(2^n)$. Wynika to z tego, że pesymistycznie - w przypadku gdy liczba wierzchołków maksymalnego podgrafu jest jest równa n - stos wywołań będzie wysokości n, czyli liczba rozpatrywanych przypadków będzie równa 2^n.

Dodatkowo dla każdego wywołania sprawdzane jest istnienie wspólnego podgrafu. Jest one określone złożonością $O(m!)$, gdzie $m = |G_2|$. Niech dany będzie podgraf grafu G_1 o k wierzchołkach. Wiadomo jest, że $k \leqslant n \leqslant m$. Liczba permutacji wierzchołków grafu G_2 ograniczona do zbioru liczności k równa jest $\frac{m!}{(m-k)!}$. Zatem w pesymistycznym przypadku - gdy $k = m$ - wartość tego wyrażenia wynosi $m!$.

Biorąc pod uwagę obie złożoności: $O(2^n) \cdot O(m!)$, gdzie $n \leqslant m$, stwierdza się złożoność rzędu $O(m!)$, gdzie m oznacza liczbę wierzchołków grafu o większej liczbie wierzchołków.

2.3 Przykłady ilustrujące działanie algorytmu

Poniżej przedstawiono przykłady przedstawiające sposób działania algorytmu dokładnego. Dla przykładów przedstawiono macierze sąsiedztwa wejściowych grafów wraz z indeksami. Następnie zapisano kolejne kroki wywołań wraz ze zwracanymi wartościami funkcji. Parametry są opisane następująco:

1. *index* - indeks rozważanego wierzchołka grafu *G1*,
2. *g1Vertices* - lista rozważanych indeksów grafu *G1*,
3. *g2Vertices* - lista odpowiadających indeksów grafu *G2* - nie istotna dla algorytmu, a jedynie dla zobrazowania wyników.

Dodatkowo przedstawiono macierze sąsiedztwa znalezionych maksymalnych podgrafów wraz z ich indeksami.

2.3.1 Przykład nr 1

```
  | 0 1 2       | 0 1 2
--+------     --+------
0 | 0 1 1     0 | 0 1 1
1 | 0 0 0     1 | 0 0 0
2 | 0 0 0     2 | 0 0 0
```

```
Wywołanie: index=0, g1Vertices=[], g2Vertices=[].
   Wywołanie bez v_0: index=1, g1Vertices=[], g2Vertices=[].
      Wywołanie bez v_1: index=2, g1Vertices=[], g2Vertices=[].
         Wywołanie bez v_2: index=3, g1Vertices=[], g2Vertices=[].
            Warunek stop. Zwracane: (g1Vertices=[], g2Vertices=[]).
         Wywołanie z v_2: index=3, g1Vertices=[2], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[2], g2Vertices=[0]).
         Zwracane: (g1Vertices=[2], g2Vertices=[0]).
      Wywołanie z v_1: index=2, g1Vertices=[1], g2Vertices=[0].
         Wywołanie bez v_2: index=3, g1Vertices=[1], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[1], g2Vertices=[0]).
         Wywołanie z v_2: index=3, g1Vertices=[1, 2], g2Vertices=[1, 2].
            Warunek stop. Zwracane: (g1Vertices=[1, 2], g2Vertices=[1, 2]).
         Zwracane: (g1Vertices=[1, 2], g2Vertices=[1, 2]).
      Zwracane: (g1Vertices=[1, 2], g2Vertices=[1, 2]).
   Wywołanie z v_0: index=1, g1Vertices=[0], g2Vertices=[0].
      Wywołanie bez v_1: index=2, g1Vertices=[0], g2Vertices=[0].
         Wywołanie bez v_2: index=3, g1Vertices=[0], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[0], g2Vertices=[0]).
         Wywołanie z v_2: index=3, g1Vertices=[0, 2], g2Vertices=[0, 1].
            Warunek stop. Zwracane: (g1Vertices=[0, 2], g2Vertices=[0, 1]).
         Zwracane: (g1Vertices=[0, 2], g2Vertices=[0, 1]).
      Wywołanie z v_1: index=2, g1Vertices=[0, 1], g2Vertices=[0, 1].
         Wywołanie bez v_2: index=3, g1Vertices=[0, 1], g2Vertices=[0, 1].
            Warunek stop. Zwracane: (g1Vertices=[0, 1], g2Vertices=[0, 1]).
         Wywołanie z v_2: index=3, g1Vertices=[0, 1, 2], g2Vertices=[0, 1, 2].
            Warunek stop. Zwracane: (g1Vertices=[0, 1, 2], g2Vertices=[0, 1, 2]).
         Zwracane: (g1Vertices=[0, 1, 2], g2Vertices=[0, 1, 2]).
      Zwracane: (g1Vertices=[0, 1, 2], g2Vertices=[0, 1, 2]).
   Zwracane: (g1Vertices=[0, 1, 2], g2Vertices=[0, 1, 2]).
```

```
  | 0 1 2       | 0 1 2
--+------     --+------
0 | 0 1 1     0 | 0 1 1
1 | 0 0 0     1 | 0 0 0
2 | 0 0 0     2 | 0 0 0
```

Dla tego przykładu znaleziono maksymalny podgraf o wielkości 3, zatem dległość wynosi 0,00.

2.3.2 Przykład nr 2

```
  | 0 1 2        | 0 1 2
--+------      --+------
0 | 0 1 0      0 | 0 1 0
1 | 1 0 0      1 | 0 0 1
2 | 0 0 0      2 | 0 1 0

Wywołanie: index=0, g1Vertices=[], g2Vertices=[].
   Wywołanie bez v_0: index=1, g1Vertices=[], g2Vertices=[].
      Wywołanie bez v_1: index=2, g1Vertices=[], g2Vertices=[].
         Wywołanie bez v_2: index=3, g1Vertices=[], g2Vertices=[].
            Warunek stop. Zwracane: (g1Vertices=[], g2Vertices=[]).
         Wywołanie z v_2: index=3, g1Vertices=[2], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[2], g2Vertices=[0]).
         Zwracane: (g1Vertices=[2], g2Vertices=[0]).
      Wywołanie z v_1: index=2, g1Vertices=[1], g2Vertices=[0].
         Wywołanie bez v_2: index=3, g1Vertices=[1], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[1], g2Vertices=[0]).
         Wywołanie z v_2: index=3, g1Vertices=[1, 2], g2Vertices=[0, 2].
            Warunek stop. Zwracane: (g1Vertices=[1, 2], g2Vertices=[0, 2]).
         Zwracane: (g1Vertices=[1, 2], g2Vertices=[0, 2]).
      Zwracane: (g1Vertices=[1, 2], g2Vertices=[0, 2]).
   Wywołanie z v_0: index=1, g1Vertices=[0], g2Vertices=[0].
      Wywołanie bez v_1: index=2, g1Vertices=[0], g2Vertices=[0].
         Wywołanie bez v_2: index=3, g1Vertices=[0], g2Vertices=[0].
            Warunek stop. Zwracane: (g1Vertices=[0], g2Vertices=[0]).
         Wywołanie z v_2: index=3, g1Vertices=[0, 2], g2Vertices=[0, 2].
            Warunek stop. Zwracane: (g1Vertices=[0, 2], g2Vertices=[0, 2]).
         Zwracane: (g1Vertices=[0, 2], g2Vertices=[0, 2]).
      Wywołanie z v_1: index=2, g1Vertices=[0, 1], g2Vertices=[1, 2].
         Wywołanie bez v_2: index=3, g1Vertices=[0, 1], g2Vertices=[1, 2].
            Warunek stop. Zwracane: (g1Vertices=[0, 1], g2Vertices=[1, 2]).
         Brak wywołania z v_2 - brak wspólnego podgrafu.
         Zwracane: (g1Vertices=[0, 1], g2Vertices=[1, 2]).
      Zwracane: (g1Vertices=[0, 1], g2Vertices=[1, 2]).
   Zwracane: (g1Vertices=[0, 1], g2Vertices=[1, 2]).

    | 0 1        | 1 2
  --+----      --+----
  0 | 0 1      1 | 0 1
  1 | 1 0      2 | 1 0
```

Dla tego przykładu znaleziono maksymalny podgraf o wielkości 2, zatem dległość wynosi 0,33.

3 Algorytm przybliżony

Do wyznaczania przybliżonego dystansu między grafami zostanie użyta metryka zbliżona do tej zastosowanej w algorytmie dokładnym. Różnica polega na tym, że zamiast maksymalnego wspólnego podgrafu, wyznaczany będzie największy wspólny podgraf, który uda się znaleźć zmodyfikowaną do celów projektu, zachłanną wersją algorytmu VF2 [2].

Sens obydwu metryk jest zbliżony, gdyż miarą dystansu będzie ta sama cecha pary grafów - liczba wierzchołków ich wspólnego podgrafu. Wybór tej metryki pozwoli zatem na lepsze porównanie obydwu algorytmów.

3.1 Zmodyfikowany algorytm VF2

3.1.1 greedyVF2

Zdefiniujmy funkcję `greedyVF2`, która przyjmuje następujące argumenty:

1. graf $G_1 = (V_1, E_1)$,
2. graf $G_2 = (V_2, E_2)$ taki, że $|V_2| \geqslant |V_1|$,
3. $M \subset V_1 \times V_2$ taki, że M to funkcja, która jest taką bijekcją, że jeśli para wierzchołków jest połączona krawędzią w jednym z grafów, to odpowiadające im wierzchołki w drugim grafie również łączy krawędź o tym samym zwrocie. Inaczej ujmując - po przeetykietowaniu zgodnym z funkcją M, te wierzchołki tworzą równe sobie podgrafy.

Funkcja zwraca liczbę wierzchołków największego znalezionego wspólnego podgrafu.

Oznaczmy dodatkowo M_1, M_2 jako odpowiednio zbiór wierzchołków odpowiednio G_1, G_2, które występują w krotkach przynależących do M. Funkcja wykonuje poniższe kroki.

1. Znajdź zbiór kandydatów $C \subset (V_1 \times V_2)$ do dodania do M - funkcja `findCandidates`.
2. Sprawdzaj, czy kolejni kandydaci $c \in C$ mogą dołączyć do M, tak, by zachować jego definicję:
   (a) jeśli któryś kandydat c może dołączyć do M, to zwróć wartość zwróconą przez rekursywne wywołanie `greedyVF2(G_1, G_2, M')`, gdzie $M' = M \cup \{c\}$,
   (b) jeśli żaden z kandydatów nie może dołączyć do M, to zwróć liczbę elementów M, odpowiadającą liczbie wierzchołków największego znalezionego wspólnego podgrafu.

3.1.2 findCandidates

Funkcja `findCandidates` przyjmuje te same argumenty, co `greedyVF2` i zwraca zbiór $C \subset (V_1 \times V_2)$ kandydatów do dodania do M.

Zdefiniujmy:

- T_1^{out} jako zbiór wierzchołków $v \in G_1 - M_1$, które posiadają krawędź skierowaną do wierzchołków należących do M_1,
- T_1^{in} jako zbiór wierzchołków $v \in G_1 - M_1$, do których istnieje krawędź skierowana od wierzchołków należących do M_1.

Odpowiednio mamy też definicje dla T_2^{out} oraz T_2^{in}. Następnie:

1. jeśli T_1^{out} oraz T_2^{out} są niepuste, funkcja zwraca $C = T_1^{out} \times t_2$, gdzie t_2 jest losowo wybranym elementem ze zbioru T_2^{out},
2. jeśli T_1^{in} oraz T_2^{in} są niepuste, funkcja zwraca $C = T_1^{in} \times t_2$, gdzie t_2 jest losowo wybranym elementem ze zbioru T_2^{in},
3. jeśli żadne z powyższych nie zachodzi oraz $G_1 - M_1$ jest niepuste, to funkcja zwraca $C = G_1 - M_1 \times v_2$, gdzie v_2 jest losowo wybranym elementem ze zbioru $G_2 - M_2$,
4. jeśli żadne z powyższych nie zachodzi, to zwracany jest zbiór pusty.

3.1.3 Obliczanie metryki

Wartość metryki przybliżonej dla takiej pary skierowanych grafów $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, że $V_2 \geqslant V_1$ obliczana jest następująco:

$$d(G_1, G_2) = 1 - \frac{greedyVF2(G_1, G_2)}{max(|V_1|, |V_2|)}.$$

3.2 Analiza poprawności i zastosowania algorytmu

Cechą decydującą o poprawności algorytmu jest to, czy zwraca on liczbę wierzchołków pewnego podgrafu. W algorytmie zbiór M bezpośrednio korzysta z definicji grafów izomorficznych, więc musi wskazywać na prawidłowy podgraf w każdej iteracji. Stąd, bazując na definicji metryki, można stwierdzić, że każdy uzyskany w ten sposób wynik będzie nie mniejszy niż ten uzyskany algorytmem dokładnym.

Dzięki zastosowanym optymalizacjom, znajdywane wartości metryki powinny być bardziej zbliżone wartościom wyznaczonym przez algorytm dokładny niż miałoby to miejsce przy bardziej losowym wyznaczaniu wspólnych podgrafów. Można to wywnioskować z tego, że w niezmodyfikowanej wersji algorytm VF2 okazuje się znacznie szybszy niż inne znane algorytmy do wyznaczania maksymalnych wspólnych podgrafów, w tym algorytm Ullmanna.[2]. Skoro te same heurystyki zostały użyte przy algorytmie zachłannym, to szansa na wybranie bardziej optymalnego zestawu wierzchołków powinna być większa.

Decyzje podjęte przy projektowaniu algorytmu sprawiają, że uzyskane wartości nie spełniają formalnej definicji metryki. Uzyskane wartości powinny jednak być skorelowane z wartościami uzyskiwanymi z użyciem algorytmu dokładnego, umożliwiając jednocześnie szacowanie wartości tej metryki dla grafów, których wielkości uniemożliwiają praktyczne zastosowanie dokładnego algorytmu.

Losowość, która została zastosowana przy generowaniu par kandydatów do dodania do M sprawia, że ponowne wywoływanie funkcji może skutkować znalezieniem większego wspólnego podgrafu. Użytkownik algorytmu będzie zatem mógł wywoływać algorytm wielokrotnie i wybrać największą uzyskaną wartość celem uzyskania lepszych rezultatów.

3.3 Złożoność

W związku z zachłannym wyborem dodawanych do M wierzchołków, algorytm iteruje po wszystkich wierzchołkach raz, sprawdzając istnienie krawędzi do każdego z pozostałych wierzchołków. Pozostałe operacje wewnątrz pętli mają złożoność stałą, stąd złożoność czasowa to $O(n * m)$, gdzie n i m są liczbami wierzchołków grafów, które porównujemy.

W czasie wykonywania programu poza danymi wejściowymi potrzebne jest przechowywanie dodawanych do znalezionego podzbioru wierzchołków, stąd otrzymana złożoność pamięciowa to $O(n)$, gdzie n jest liczbą wierzchołków mniejszego z grafów.

4 Implementacja

Implementacja rozwiązania została przeprowadzona w języku C# przy użyciu platformy .NET w zintegrowanym środowisku programistycznym Visual Studio (wersja 2019). W niniejszej sekcji opisano strukturę programu oraz zmiany względem pierwotnych ustaleń.

4.1 Struktura rozwiązania

Kod źródłowy podzielono na cztery moduły (projekty) stanowiące całość rozwiązania:

- GraphDistance,
- GraphDistanceTests,
- ReportGenerator,
- Analyzer.

4.1.1 GraphDistance

W projekcie GraphDistance wyobrędniono dwa foldery:

- Graphs,
- Algorithms.

Folder Graphs zawiera pliki, w których zdefiniowano klasy:

- AdjacencyMatrix - klasa reprezentująca macierz sąsiedztwa grafu wraz z pomocniczymi metodami,
- Graph - klasa reprezentująca graf wraz z pomocniczymi metodami,
- GraphFile - statyczna klasa zawierająca między innymi metody: pozwalającą odczytać dane porównywanych grafów z pliku .txt oraz zapisującą graf do pliku .txt,
- Errors - statyczna klasa zawierająca opisy błędów, które mogą się pojawić podczas działania metod z wyżej wypunktowanych klas.

Dodatkowo, w pliku Generator zaimplementowano klasy i metody pozwalające na generowanie losowych grafów, co zostanie wykorzystane w projekcie Analyzer do przygotowania danych do wykresów porównawczych.

W folderze Algorithms znajdują się pliki zawierające implementację zastosowanych algorytmów opisanych w poprzednich sekcjach, a w szczególności:

- IDistanceFinder - interfejs zawierający deklarację metody FindDistance, pozwalającej znaleźć odległość miedzy grafami,
- folder Exact, w którym znajduje się plik z implementacją algorytmu dokładnego,
- folder GreedyVF2, w którym znajdują się pliki implementujące algorytm przybliżony.

4.1.2 GraphDistanceTests

Projekt GraphDistanceTests zawiera testy jednostkowe pozwalające sprawdzić poprawność metod pomocniczych zdefiniowanych w klasach AdjacencyMatrix oraz Graph.

Testy jednostkowe zaimplementowano również dla metody FindDistance z klasy ExactDistanceFinder, ponieważ jako jedyny algorytm wyszukiwania odległości zwraca pewną, dokładną wartość, a wyniki jego działania można precyzyjnie przewidzieć.

4.1.3 ReportGenerator

ReportGenerator jest projektem startowym przygotowanej aplikacji konsolowej. Zawiera implementację interfejsu użytkownika oraz niezbędne klasy i metody pozwalające na bezproblemowe poruszanie się użytkownika po aplikacji, a także czytelne wyświetlanie otrzymywanych wyników.

4.1.4 Analyzer

Projekt Analyzer jest projektem pomocniczym, umożliwiającym wygenerowanie do pliku .csv danych pozwalających na stworzenie czytelnych wykresów porównawczych dla algorytmów stosowanych do obliczania odległości między grafami.

4.2 Zmiany względem pierwotnej wersji

Podczas przygotowywania rozwiązania nie przeprowadzono żadnych kluczowych zmian względem pierwotnej wersji dokumentacji przygotowanej na jej podstawie implementacji.

Jedyną ważną zmianą było przerobienie interfejsu IDistanceFinder i metod go implementujących tak, aby zwracały (poza obliczonym dystansem) listę krotek, która określa wierzchołki indeksy wierzchołków obu grafów, które składają się na znaleziony maksymalny wspólny podgraf. Zostanie to wykorzystane przy wyświetlaniu wyników na konsoli.

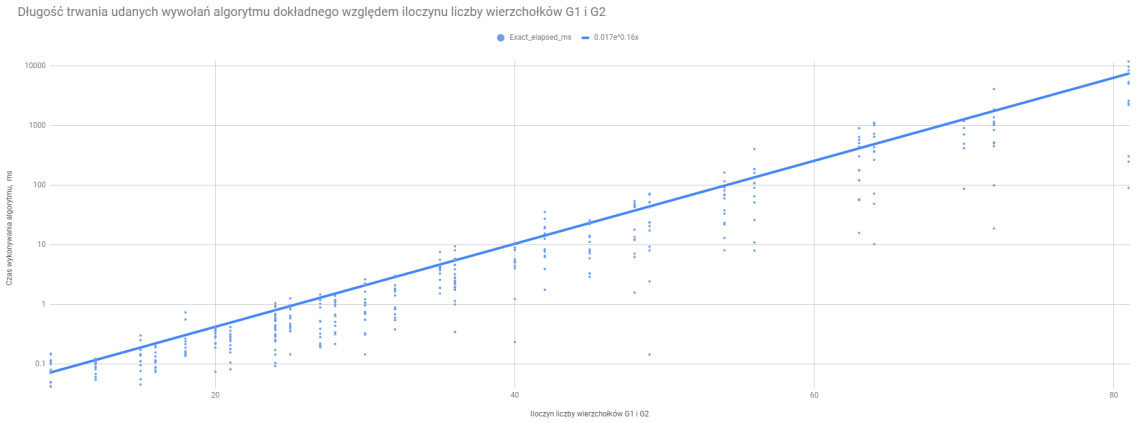
5 Testowanie i porównywanie zastosowanych algorytmów

Przy użyciu projektu Analyzer wygenerowano dane do wykresów porównawczych zaproponowanych algorytmów wyszukiwania odległości między grafami.

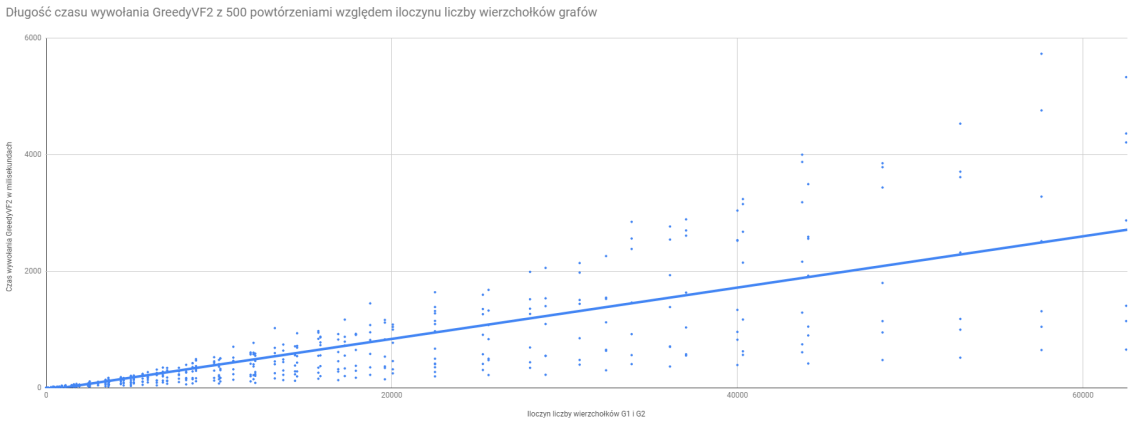
Łącznie wygenerowano około 900 różnorodnych par grafów, do porównania których zastosowano algorytm dokładny oraz algorytm GreedyVF2 z 1, 10, 100 i 500 podejściami. Na potrzeby testowe przyjęto, że timeout - czas, po którym algorytm przerwie obliczenia niezależnie od stopnia zaawansowania - będzie wynosić 30 sekund.

Algorytmy wydają się być poprawne - dla znanych wielkości największego podgrafu algorytm dokładny zwraca wartość dokładną metryki, a GreedyVF2 wartość większą lub równą metryce z algorytmu dokładnego. Na podstawie analizy uzyskanych wyników wywnioskowano, że:

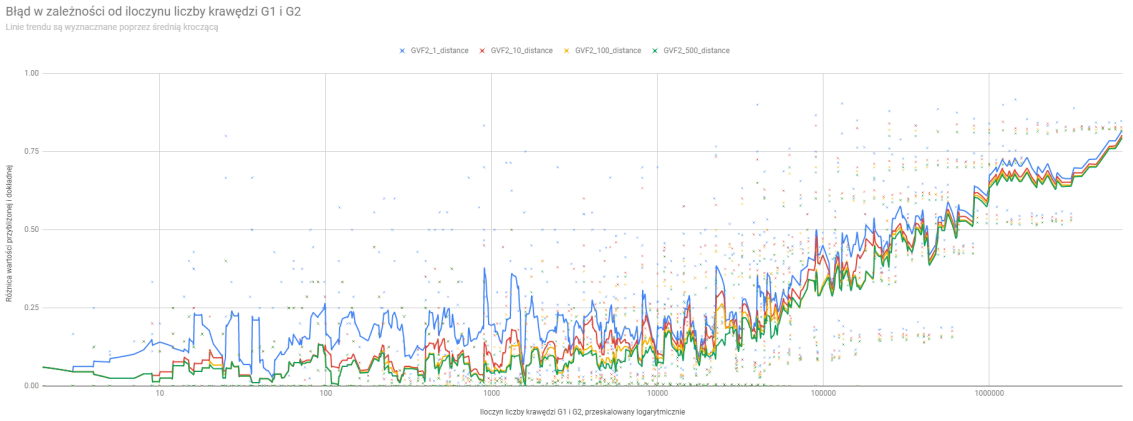
- w przypadku, gdy w obydwu grafach istnieje skierowany cykl Hamiltona idealne okazuje się zastosowanie algorytmu przybliżonego GreedyVF2, który zwraca bardzo szybko dokładne wyniki,
- w przypadku rozważania pary grafów (G1, podgraf G1 z przeetykietowanymi wierzchołkami) warto używać algorytmu dokładnego ze względu na dokładne wyznaczenie wartości metryki, pomimo braku szans na skończenie obliczeń przed upłynięciem timeoutu dla większych grafów,
- w przypadku losowości porównywanych grafów ciężko dojść do jednoznacznych wniosków, poza stwierdzeniem oczywistego faktu, że im więcej prób w algorytmie GreedyVF2, tym mniejszy błąd pojawi się podczas obliczania złożoności,
- algorytm GreedyVF2 jest bardzo wydajny - nawet dla bardzo dużych grafów kończy obliczenia w mniej niż sekundę, natomiast zwiększenie liczby powtórzeń zwiększa dokładność wyznaczanego dystansu.



Rysunek 1: Czas trwania udanych wywołań algorytmu dokładnego względem iloczynu liczby wierzchołków. Czas trwania obliczeń algorytmu dokładnego faktycznie jest ekspotencjalny.

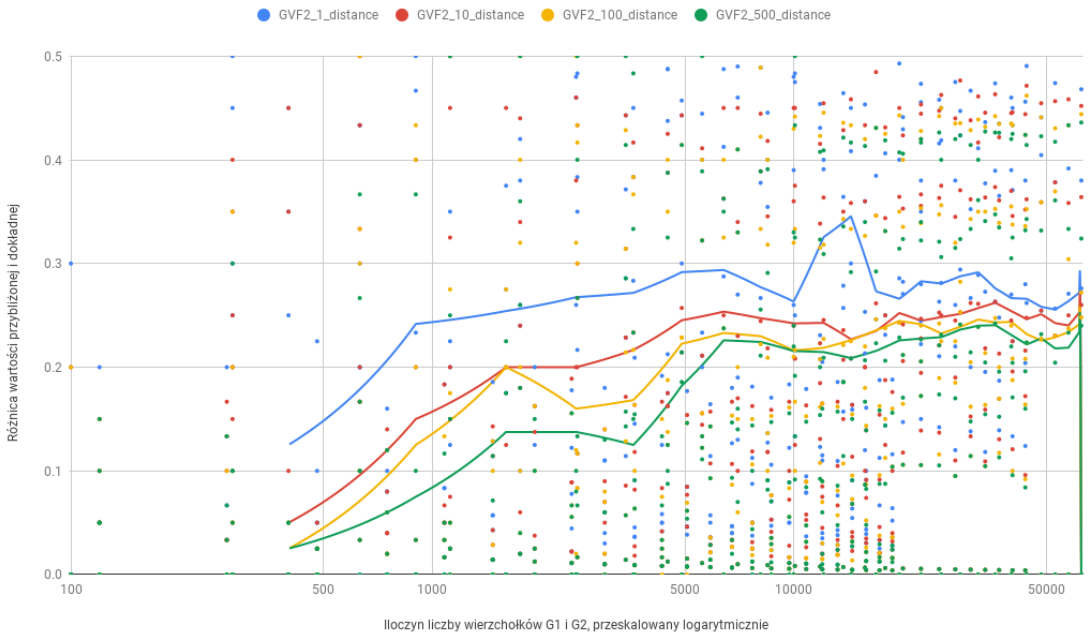


Rysunek 2: Czas wywołania GreedyVF2 z 500 powtórzeniami względem iloczynu liczby wierzchołków. Czas wykonania faktycznie rośnie liniowo wraz ze wzrostem iloczynu liczby wierzchołków grafu.



Rysunek 3: Błąd w zależności od iloczynu liczby krawędzi G1 i G2.

Rysunek 4: Błąd w zależności od iloczynu liczby wierzchołków G1 i G2.



Rysunek 5: Wartość różnicy pomiędzy dystansami obliczonymi algorytmami przybliżonymi z różnymi liczbami prób, a wartością uzyskaną za pomocą algorytmu dokładnego.

6 Literatura

[1] H. Bunke i K. Shearer, "A graph distance metric based on the maximal common subgraph," *Pattern Recognition Letters*, t. 19, nr. 3, s. 255–259, 1998, ISSN: 0167-8655.

[2] P. Foggia, C. Sansone i M. Vento, "An Improved Algorithm for Matching Large Graphs," *Proc of the 3rd IAPR-TC-15 International Workshop on Graph-based Representation*, sty. 2001.

7 Załączniki

- [Arkusz z analizą porównawczą algorytmów dokładnego i przybliżonego], dostęp z dnia 29.11.2021r.