Projekt Zaliczeniowy - Sprawozdanie
Teoria Algorytmów i Obliczeń

Piotr Gugnowski, Karol Gutkowski, Kamil Kamiński, Dawid Kozaczuk

koordynator przedmiotu: prof. dr hab. inż. Władysław Homenda

Grudzień 2024

Spis treści

1 Wstęp

Dokument ten jest sprawozdaniem z projektu na zaliczenie przedmiotu Teoria Algorytmów i Obliczeń realizowanego podczas semestru zimowego 2024/2025 na wydziale Matematyki i Nauk Informacyjnych Politechniki Warszawskiej.

Projekt jest realizowany w wersji dla zwyczajnych grafów skierowanych (nie multigrafów).

Celem projektu było opracowanie pojęć: rozmiar grafu, metryka w zbiorze grafów, maksymalny cykl w grafie i liczba tych cykli, minimalne rozszerzenie grafu do grafu zawierającego cykl Hamiltona i liczbę cykli Hamiltona. Oznacza to, że dla podanych pojęć przygotowane zostały definicje, dokładne algorytmy oraz wielomianowe algorytmy aproksymacyjne (dla algorytmów, które w wersji dokładnej są wykładnicze). Algorytmy zostały uzasadnione, ich złożonośc została przeanalizowana oraz zostały na nich przeprowadzone testy wydajności.

2 Opracowanie pojęć

Graf (skierowany) - para zbiorów (V, E). Zbiór V jest zbiorem wierzchołków grafu. Za to E zbiorem krawędzi, czyli zbiorem par wierzchołków $E \subseteq V \times V = \{(u, v) \mid u, v \in V, u \neq v\}$, $uv \in E$ oznacza krawędź skierowaną od wierzchołka u do wierzchołka v.

Rozmiar grafu - to $|E|$, czyli liczność zbioru krawędzi E.

Ścieżka - dla grafu $G = (V, E)$ ścieżka to naprzemienny ciąg wierzchołków $v_1, v_2, ..., v_k$ taki, że dla każdego $i \in \{1, 2, ..., k-1\}$ jest $v_i v_{i+1} \in E$.

Cykl w grafie - dla grafu $G = (V, E)$ cyklem nazywamy ścieżkę $P = (v_1, v_2, ..., v_k)$, gdzie pierwszy i ostatni wierzchołek są takie same ($v_i = v_k$) oraz żaden wierzchołek (oprócz pierwszego) nie występuje na ścieżce więcej niż raz.

Maksymalny cykl w grafie - Maksymalny cykl w grafie to cykl, który zawiera najwięcej wierzchołków spośród wszystkich cykli w danym grafie. Może istnieć wiele takich cykli, które mają tę samą, maksymalną liczbę wierzchołków. Liczba takich cykli to liczbą maksymalnych cykli w grafie.

Cykl Hamiltona - cykl Hamiltona to taki cykl w grafie, który zawiera każdy wierzchołek z grafu dokładnie raz.

Minimalne rozszerzenie grafu do grafu zawierającego cykl Hamiltona - minimalne rozszerzenie to rozszerzanie grafu do grafu zawierającego cykl Hamiltona. Minimalne rozszerzenie to taki zbiór krawędzi, który zawiera najmniejszą ilość krawędzi, które należy dodać aby graf zawierał cykl Hamiltona.

Liczba cykli Hamiltona - liczbą cykli Hamiltona to liczba wszystkich możliwych unikalnych skierowanych cykli Hamiltona w grafie to znaczy, dla których kolejność odwiedzanych wierzchołków jest różna.

2.1 Metryka

Warunki metryki Metryka dla dowolnego niepustego zbioru X, jest to sfunckja $d : X \times X \to [0, \infty)$. Która dla dowolnych elementów $a, b, c \in X$ spełnia poniższe warunki:

1. identyczność nierozróżnialnych
$$d(a, b) = 0 \iff a = b,$$
2. symetria
$$d(a, b) = d(b, a),$$
3. nierówność trójkąta
$$d(a, b) \leq d(a, c) + d(c, b).$$

Definicja metryki dla grafów - Na przestrzeni grafów przyjęto, że funckja odległości d między grafami G_1 i G_2 zdefinowana jest następująco:
$d(G_1, G_2)$ = minimalna liczba zmian, których trzeba dokonać, aby otrzymać graf G_1 z grafu G_2 (uwzględniając równość izomorficzną). Te zmiany to:

- dodanie wierzchołka do G_2
- usunięcie wierzchołka z G_2
- dodanie krawędzi do G_2
- usunięcie krawędzi z G_2

Dowód poprawności metryki

- Nieujemność. Liczba zmian nie może być oczywiście ujemna, więc dla każego G_1 i G_2 mamy: $d(G_1, G_2) \geq 0$.
- Identyczność nierozróżnialnych. Dla grafów G_1 i G_2 jeśli $d(G_1, G_2) = 0$. Oznacza to, że nie trzeba dokonywać żadnych zmian aby otrzymać z grafu G_2 graf G_1.
- Symetria. Dla G_1 i G_2 wartość $d(G_1, G_2)$ jest sumą:
  - liczby zmian utożsamianych z dodaniem wierzchołka będzie tyle ile jest wierzchołków w G_1, których nie ma w G_2 ($|V(G_1) \backslash V(G_2)|$).
  - liczby zmian utożsamianych z usunięciem wierzchołka będzie tyle ile jest wierzchołków w G_2, których nie ma w G_1 ($|V(G_2) \backslash V(G_1)|$).
  - liczby zmian utożsamianych z dodaniem krawędzi będzie tyle ile jest krawędzi w G_1, których nie ma w G_2 ($|E(G_1) \backslash E(G_2)|$).

– liczby zmian utożsamianych z usunięciem krawędzi będzie tyle ile jest krawędzi w G_2, których nie ma w G_1 ($|E(G_2) \backslash E(G_1)|$).

Natomiast gdyby w identycznej kolejności rozważyć składowe wartości $d(G_2, G_1)$ to okazałoby się, że pierwsza składowa byłaby taka sama jak punkt drugi, druga taka sama jak punkt pierwszy, trzecia składowa taka sama jak punkt czwarty, czwarta składowa taka sama jak punkt trzeci.
Stąd widać, że wartości $d(G_1, G_2)$ i $d(G_2, G_1)$ są opisane przez sumę tych samych składników, co należało pokazać.

- Nierówność trójkąta. Należy pokazać, że dla dowolnych grafów G_1, G_2, G_3 mamy:

$$d(G_1, G_3) \leq d(G_1, G_2) + d(G_2, G_3).$$

Załóżmy, że prawdą jest, że:

$$d(G_1, G_3) \geq d(G_1, G_2) + d(G_2, G_3).$$

Z lewej mamy minimalną liczbę zmian potrzebnych do przekształcenia G_3 do G_1.
Z prawej strony mamy minimalną liczbę zmian potrzebnych do uzyskania G_1 przekształcając G_2 oraz minimalną liczbę zmian potrzebnych do uzyskania G_2 z G_3.
Stąd możemy zauważyć, że po prawej stronie mamy podaną formułę uzyskania G_1 przekształcając G_3 (najpierw do G_2, a z G_2 do G_1).
Skoro tak, to mamy sposób, który pozwoli uzyskać liczbę zmian mniejszą niż minimalna liczba zmian z lewej strony nierówności, co jest sprzecznością.
Skoro założenie jest fałszywe, to oryginalna nierówność jest prawdziwa, co należało pokazać.

3 Wyznaczanie odległości między grafami dla przyjętej metryki

3.1 Algorytm dokładny

Algorytm wyznaczania odległości między grafami $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$:

1. Ustal, który z grafów ma mniej wierzchołków, ten graf będziemy nazywać mniejszym, a drugi odpowiednio większym,
2. Wyznacz różnicę ilości wierzchołów między grafem mniejszym, a większym,
3. Rozszerz graf mniejszy o dodanie wierzchołków w ilości wyznaczonej różnicy,
4. Dla każdej permutacji wierzchołków grafu mniejszego (rozszerzonego do rozmiarów większego):
   (a) Wyznacz ilość różnych krawędzi pomiędzy grafem większym, a mniejszym (jeśli w dla danego wierzchołka o numerze j w danej permutacji istnieje krawędź w tylko jednym z grafów to liczymy to jako różnicę).

(b) Wybierz minimum z obecnie wyznaczonej wartości różnych krawędzi, a obecnie najniższą wartością. Jeśli minimum to 0, zakończ obliczenia.

5. Zwróć najmniejszą ilość różnych krawędzi powiększoną o liczbę różnicy wierzchołków między grafem większym, a mniejszym.

Poprawność algorytmu - obliczenie ilości różnych krawędzi dla każdej z permutacji (lub do uzyskania wartości 0), gwarantuje sprwadzenie każdego możliwego ustawienia wierzchołków grafu mniejszego (rozszerzonego o dodanie ilości wierzchołków równej różnicy ilości wierzchołków między grafami). Algorytm zwraca wartość minimalną ze wszystkich tych różnic co znaczy, że dobrze uwzględnia równość izomorficzną.

Złożonośc algorytmu Jeśli $n_1 = |V_1|$, a $n_2 = |V_2|$ niech $n = \max(n_1, n_2)$, wtedy wykonane jest $!n$ permutacji, porównanie różnic krawędzi jest w czasie $O(n^2)$. Złożoność algorytmu to $O(!n * n^2)$.

3.2 Algorytm aproksymacyjny

Algorytm aproksymacyjny wyznaczania odległości między grafami $G_1 = (V_1, E_1)$ oraz $G_2 = (V_2, E_2)$:

1. Ustal, który z grafów ma mniej wierzchołków, ten graf będziemy nazywać mniejszym, a drugi odpowiednio większym,

2. Wyznacz różnicę ilości wierzchołów między grafem mniejszym, a większym,

3. Rozszerz graf mniejszy o dodanie wierzchołków w ilości wyznaczonej różnicy,

4. Posortuje wierzchołki obu grafów (zachowując odpowiednie krawędzie) według stopni krawędzi wychodzących i wchodzących t.ż. wierzchołek $v_i > v_j$ gdy:

$$(\deg_{\text{in}}(v_i) > \deg_{\text{in}}(v_j)) \vee (\deg_{\text{in}}(v_i) = \deg_{\text{in}}(v_j) \wedge \deg_{\text{out}}(v_i) > \deg_{\text{out}}(v_j)).$$

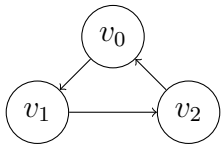
5. Analogicznie jak w algorytmie dokładnym wyznacz ilośc różnych krawędzi

6. Zwróć ilość różnych krawędzi powiększoną o różnicę ilości wierchołów między grafami.

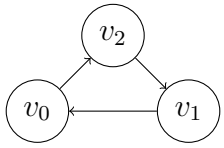
Poprawność algorytmu - Aby uzsykać wielomianową złożonośc algortymu aproksymacyjnego, zamiast wykonywać sprawdzenie dla wszystkich permutacji, obliczana jest liczba różnych krawędzi dla jednej permutacji. W czasie wielomianowym (algorytmy sortowania to zwykle $O(\log n)$ wykonujemy sortowanie wierzchołków co potencjalnie pozwala nam znaleźć lepsza permutację niż ta podana podstawowo na wejściu. Algorytm aproksymacyjny liczy ilość różnych krawędzi w ten sam sposób co algorytm dokładny. Uznano zatem aproksymowaną odległość za poprawnie wyznaczaną.

Złożonośc algorytmu Jeśli $n_1 = |V_1|$, a $n_2 = |V_2|$ niech $n = \max(n_1, n_2)$, to sortowanie wierzchołków jest w czasie $O(\log n)$, obliczenie ilości różnych krawędzi jest rzędu $O(n^2)$. Czyli algorytm jest złożoności $O(\log n + n^2) = O(n^2)$, więc jest wielomianowy.

Utracone własności - wybór algorytmu aproksymacyjnego wiąże się z utratą własności metryki. Tracone jest założenie o identyczności nierozróżnialnych. Przy założeniu identyczności na bazie równości izomorficznej algorytm aproksymacyjny nie zachowuje tej własności w niektórych przypadkach. Poniżej prosty przypadek, gdy gdzie grafy G_1 i G_2, które są widocznie izomorficznie nierozróżnialne, a dla których aproksymacja odległości to 6. Jest tak gdyż stopnie wszystkich wierzchołów $(\deg_{\text{in}}(v_i), \deg_{\text{out}}(v_i)) = (1, 1)$. Podejście z sortowaniem zachowa oryginalną kolejność i obliczenie różnych krawędzi da nam 6.



Rysunek 1: graf G_1



Rysunek 2: graf G_2

Założenie symetrii pozostaje nadal prawdziwe, $d_{approx}(G_1, G_2) = d_{approx}(G_2, G_1)$. Sortowanie wierzchołków grafów G_1 oraz G_2 da zawsze tą samą wyjsciową kolejność. A więc porówananie różnicy krawędzi da także ten sam rezultat.

Także założenie nierówności trójkąta zostaje zachowane. Weźmy dowolne trzy grafy G_1, G_2 oraz G_3. Załóżmy, że $d_{approx}(G_1, G_3) \geq d_{approx}(G_1, G_2) + d_{approx}(G_2, G_3)$. Oznaczmy r_p jako ilość różnych krawędzi po posortowaniu wierzchołów grafu (jest to operacja porównania komórek rozszerzonych macierzy sąsiedztwa między grafami). Rozpisując prawą stronę nierówności mamy:

$$\begin{aligned} d_{\text{approx}}(G_1, G_2) + d_{\text{approx}}(G_2, G_3) &= |n_1 - n_2| + r_p(G_1, G_2) + |n_2 - n_3| + r_p(G_2, G_3) \\ &= |n_1 - n_2 + n_2 - n_3| + r_p(G_1, G_2) + r_p(G_2, G_3) \\ &= |n_1 - n_3| + r_p(G_1, G_2) + r_p(G_2, G_3). \end{aligned}$$

Rozpisując lewą stronę mamy:

$$d_{\text{approx}}(G_1, G_3) = |n_1 - n_3| + r_p(G_1, G_3).$$

Aby założona nierówność była prawdziwa to $r_p(G_1, G_3) \geq r_p(G_1, G_2) + r_p(G_2, G_3)$, ale jest to sprzeczne. Operacja r_p polega po prostu na porównaniu komórek rozszerzonych macierzy sąsiedztwa (i zsumowaniu ilości różnych komórek), dodanie dodatkwoego kroku porównania macierzy G_1 oraz G_2, a potem G_2 z G_3 nie może zmniejszyć nam ilości potrzebnych do zmiany wartości w rozszerzonej macierzy G_1 aby zrównwać ją z macierzą G_3. A więc mamy $d_{approx}(G_1, G_3) \leq d_{approx}(G_1, G_2) + d_{approx}(G_2, G_3)$

4 Znajdowanie minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie

4.1 Algorytm dokładny

Inicjalizacja algorytmu rekurencyjnego: Algorytm polega na zbadaniu każdej możliwej permutacji wszystkich wierzchołków w grafie oraz wyznaczeniu, między którymi wierzchołkami w poszczególnych ciągach brakuje krawędzi. Pierwszym krokiem jest inicjalizacja dwóch zmiennych monitorujących aktualny stan wywołań rekurencyjnych. Pierwszą z nich jest ciąg wierzchołków current_path, reprezentujący aktualnie badaną ścieżkę w grafie. Druga, current_extension, reprezentuje listę brakujących krawędzi w aktualnie badanej ścieżce (zapisanej w current_path). Ponadto, tworzone są zmienne smallest_extension do przechowywania najkrótszego dotychczas znalezionego rozszerzenia, a także hamilton_cycle_count do zliczania dotychczas znalezionych cykli Hamiltona. Algorytm jest wywołany iteracyjnie dla każdego wierzchołka grafu, na początku i-tej iteracji czyszczone są zmienne current_path i smallest_extension, po czym do pierwszej z nich zostaje dodany wierzchołek i. Następnie dla każdego wierzchołka o indeksie j różnym od i wywoływana jest następująca sekwencja instrukcji. Najpierw następuje sprawdzenie, czy istnieje krawędź do wierzchołka j z wierzchołka i. Jeśli nie, taka krawędź zostaje dodana do aktualnego rozszerzenia w zmiennej current_extension. Wierzchołek j zostaje dodany do aktualnej ścieżki current_path, po czym wywoływana jest funkcja rekurencyjna. Po wyjściu z tejże funkcji wierzchołek j zostaje zdjęty z current_path. Jeśli krawędź z i do j znajduje się w current_extension, to również jest ona usuwana.

Funkcja rekurencyjna - kolejne wywołania: Funkcja rekurencyjna na wejściu otrzymuje aktualną ścieżkę current_path oraz listę brakujących krawędzi w tej ścieżce smallest_extension. Następnie dla każdego wierzchołka k, który nie znajduje się na ścieżce current_path, wywoływana jest następna sekwencja instrukcji. Najpierw następuje sprawdzenie, czy istnieje krawędź do wierzchołka k z ostatniego wierzchołka w current_path. Jeśli nie, taka krawędź zostaje dodana do aktualnego rozszerzenia w zmiennej current_extension. Wierzchołek k zostaje dodany do aktualnej ścieżki current_path, po czym wywoływana jest funkcja rekurencyjna. Po wyjściu z tejże funkcji wierzchołek k zostaje zdjęty z current_path. Jeśli krawędź z ostatniego wierzchołka w current_path do k znajduje się w current_extension, to również jest ona usuwana.

Funkcja rekurencyjna - warunek stopu: Jeśli aktualna ścieżka current_path zawiera wszystkie wierzchołki grafu, nie istnieje możliwość dalszych wywołań funkcji rekurencyjnej, wówczas podejmowane są następujące działania:

- Do current_path zostaje dodany pierwszy wierzchołek, zamykając w ten sposób cykl. Jeśli z poprzedniego wierzchołka do aktualnego nie istniała krawędź w grafie, zostaje ona dodana do current_extension.

- W przypadku, gdy lista current_extension jest pusta, mamy do czynienia z cyklem Hamiltona. Ponieważ przeszukujemy cykle zaczynając z dowolnego wierzchołka, należy najpierw sprawdzić, czy aktualny cykl nie został już uwzględniony (po starcie z innego wierzchołka). Jeśli nie, liczba znalezionych cykli Hamiltona jest zwiększana o 1.

- Jeżeli lista current_extension nie jest pusta, a liczba znalezionych cykli Hamiltona wynosi 0, to porównujemy najmniejsze dotychczas znalezione rozszerzenie z aktualnym. Jeśli najmniejsze rozszerzenie w zmiennej smallest_extension jest puste, to nie znaleźliśmy jeszcze żadnego rozszerzenia do cyklu Hamiltona, zatem zmiennej smallest_extension jest przypisywane aktualne rozszerzenie z current_extension. Jeżeli smallest_extension nie jest puste, to porównujemy jego długość z aktualny rozszerzeniem z current_extension. W sytuacji, gdy current_extension jest krótszym rozszerzeniem, zmieniamy wartość smallest_extension na tę równą current_extension.

Złożoność: Opisany powyżej dokładny algorytm rekurencyjny ma złożoność O(n!). Związane jest to z faktem, że przechodząc przez kolejne ścieżki analizowana jest każda permutacja wierzchołków w grafie.

4.2 Algorytm aproksymacyjny

retryFactor: parametr podawany przed wykonaniem algorytmu. Parametr ten określa, ile razy niżej opisany algorytm ponowi próbę znalezienia najmniejszego rozszerzenia grafu do grafu zawierającego cykl Hamiltona oraz ile razy ponowi próbę znalezienia cyklu Hamiltona. Jego minimalna wartość to 1, zalecane jest, aby jego wartość nie przekraczała 16, aby zachować rozsądny czas wykonania algorytmu. W zaimplementowanym algorytmie przyjęto, że maksymalną dopuszczalną wartością retryFactor będzie 15, a minimalną 1.

Opis: Znajdowanie minimalnego rozszerzenia grafu Ta część algorytmu polega na wielokrotnym wykonaniu (liczba wykonań to średnia ilości krawędzi wychodzących pomnożone przez retryFactor) procedury **podążania losową ścieżką po grafie**.

- Najpierw wybierany jest losowy wierzchołek.

- Następnie wybierany jest losowy nieodwiedzony sąsiadujący wierzchołek wybranego wierzchołka po czym następuje przejście do tego wierzchołka.

- Przejścia są powtarzane do momentu, aż w trakcie działania procedury znajdzie się wierzchołek, który nie ma krawędzi wychodzących do nieodwiedzonego wierzchołka. Wtedy następuje dodanie krawędzi.

- Krawędź pozwala przejść dalej do nieodwiedzonego wierzchołka z najmniejszą ilością krawędzi wchodzących z nieodwiedzonych do tej pory wierzchołków.

- Po takim przejściu następuje kontynuowanie podążanie losową ścieżką.

- Gdy odwiedzone zostaną wszystkie wierzchołki następuje przejście do początkowego wierzchołka. Jeśli brakuje krawędzi to jest dodana.
- Rozszerzenie grafu to zbiór dodanych krawędzi podczas procedury budowania ścieżki.
- Wybierany jest najmniej liczby zbiór spośród zbiorów tworzonych podczas każdej próby budowania ścieżki.

Opis: Znajdowanie liczby cykli Hamiltona w rozszerzonym grafie Ta część algorytmu polega na zliczaniu ścieżek, którymi udało się przejść podczas próby przejścia przez wszystkie wierzchołki dokładnie raz oraz takich, dla których istnieje krawędź w grafie od ostatniego do pierwszego wierzchołka. Szukanie ścieżki jest wykonywane wielokrotnie (ilość wywołań to: rozmiaru grafu pomnożony przez max(1, ln(retryFactor))), każda unikatowa ścieżka (mająca unikalną kolejność wierzchołków, zaczynając od wierzchołka z najmniejszym indeksem) jest zapisywana. Na początku wiadomo o jednym cyklu Hamiltona, tym, którym podążano przy tworzeniu najmniejszego rozszerzenia do grafu zawierającego cykl Hamiltona.

Bez rozszerzenia:

- Jeśli grafu nie trzeba było rozszerzać, aby zawierał cykl Hamiltona każdy punkt rozpoczęcia budowania ścieżki jest równie dobry, dlatego też każda ścieżka zaczyna się od wierzchołka o indeksie 0.
- Wykonywana jest **procedura podążania losową ścieżką** (inna niż dla części algorytmu znajdującej rozszerzenie).
- W przypadku przejścia do wierzchołka, z którego nie można już przejść do nieodwiedzonego wierzchołka sprawdzane jest czy wszystkie wierzchołki zostały odwiedzone i czy istneje krawędź w grafie z ostatniego do pierwszego wierzchołka. Jeśli warunki są spełnione to cykl jest dodawany do zbioru unikalnych cykli.

Z rozszerzeniem: Jeśli graf trzeba było rozszerzyć, to jedna z dodanych krawędzi była niezbędna, aby w ogóle jakikolwiek cykl Hamiltona istniał w grafie. Dlatego też opłaca się rozpoczynać budowę ścieżek od krawędzi należących do rozszerzenia grafu. Dalej algorytm działa tak jak dla grafu, którego nie trzeba było rozszerzać.

Uzasadnienie złożoności: Uwagi Uzasadnienie będzie polegać na podaniu wszystkich rzeczywistych kroków podczas wykonywania algorytmu i dopisywaniu po kroku jego złożoności. Przyjęto, że $G = (V, E)$, $|V| = n$, $|E| = m$. Finalna złożoność jest łatwa do zauważenia przez wybranie największego złożenia złożoności kroku i złożoności pętli, w których dany krok się znajduje.

Uzasadnienie złożoności: Znajdowanie minimalnego rozszerzenia grafu O(n^3)

- Wyznaczana jest liczba powtórzeń **procedury budowania ścieżki**. Jest to średnia ilość krawędzi wychodzących dla wierzchołków w grafie zaokrąglona w górę (dla 0 krawędzi 1) pomnożona przez współczynnik powtórzeń (retryFactor). O(n)

- Wykonanie poniższej logiki tyle razy, ile wynosi wyznaczona liczba powtórzeń O(n)
  - Wyznaczany jest wierzchołek startowy losowo. O(1)
  - Wykonanie procedury **podążania losową ścieżką** O(n^2) (Wyjaśnienie poniżej)
  - Sprawdzenie czy podążanie wyznaczoną ścieżką wymagało najmniejszej ilości dodanych krawędzi do grafu. Jeśli nie to przejście do kolejnej iteracji. O(1)
  - Przypisanie dodanych krawędzi jako najmniejszego rozszerzenia. O(1)
  - Ustawienie cyklu tak, by zaczynał się od najmniejszego indeksu O(n)
  - Dodanie przebytej ścieżki do zbioru cykli Hamiltona odpowiadających najmniejszemu rozszerzeniu O(1)
  - Przerwanie pętli jeśli najmniejsze rozszerzenie ma 0 krawędzi. O(1)

Uzasadnienie złożoności: Podążanie losową ścieżką O(n^2) Jest to funkcja rekurencyjna, która przyjmuje referencję do zbioru odwiedzonych wierzchołków, wierzchołek startowy, listę relacji i ilość wierzchołków grafu, zbiór dodanych krawędzi, wektor kolejno odwiedzonych wierzchołków, obecnie *odwiedzany* wierzchołek. Obecnie odwiedzany wierzchołek musi być wcześniej nieodwiedzony, więc górnym ograniczeniem ilości wywołań funkcji jest ilość wierzchołków w grafie. O(n) (W wywołaniu wykonywana jest poniższa logika.)

- Dodaj obecnie odwiedzany wierzchołek do zbioru odwiedzonych wierzchołków. O(1)
- Dodaj obecnie odwiedzany wierzchołek na koniec obecnie tworzonej ścieżki O(1)
- Sprawdź czy w grafie istnieje krawędź do wierzchołka startowego z *odwiedzanego* wierzchołka. O(1)
- Wybierz losowego nieodwiedzonego sąsiada odwiedzanego wierzchołka. O(n)
- Usuń krawędzie powiązane z odwiedzanym wierzchołkiem, jeśli odwiedzany wierzchołek nie jest wierzchołkiem startowym. O(n)
- Jeśli nie udało się wybrać losowego sąsiedniego i nieodwiedzonego wierzchołka to wybierany jest losowy wierzchołek spośród nieodwiedzonych, który ma najmniej krawędzi wchodzących z nieodwiedzonych wierzchołków. O(n)
- Jeśli nie udało się znaleść takiego wierzchołka to znaczy, że odwiedzono już wszyskie wierzchołki. Jeżeli brakuje krawędzi do początkowego wierzchołka to jest ona dodawana. Rekurencja się kończy. O(1)
- Jeśli udało się znaleść taki wierzchołek to dodawana jest krawędź od odwiedzanego wierzchołka do wybranego wierzchołka. O(1)
- Wywołana jest rekurencja dla wybranego w tym wywołaniu następnego wierzchołka. O(1)

Uzasadnienie złożoności: Znajdowanie cykli hamiltona w rozszerzonym grafie O(n^5) Pierwszy cykl Hamiltona był znaleziony podczas znajdowania minimalnego rozszerzenia grafu. Jeśli graf nie wymaga rozszerzenia, by istniał w nim cykl Hamiltona to wykonana jest poniższa procedura.

- Wyznaczona jest liczba iteracji, będąca liczbą krawędzi w grafie pomnożoną przez max(1, ln(retryFactor)). O(n)
- Wykonanie poniższych kroków tyle razy, ile wynosi liczba iteracji. O(n^2)
  - Wywołane jest **podążanie losową ścieżką** z wierzchołka o indeksie 0 w grafie. Różnica względem poprzedniego podążania losową ścieżką jest taka, że nie są usuwane krawędzie wychodzące z odwiedzonego wierzchołka oraz niepowodzenie przy wyborze sąsiedniego nieodwiedzonego wierzchołka kończy procedurę podążania ścieżką. O(n^2)
  - Jeśli wyznaczona ścieżka odwiedza wszystkie wierzchołki i istnieje krawędź w grafie z ostatniego do wierzchołka 0 to ścieżka ta jest dodawana do zbioru unikalnych cykli Hamiltona znalezionego rozszerzenia grafu. O(1)

Do szukania pozostałych cykli, jeśli minimalne rozszerzenie zawiera krawędź, wykorzystana jest poniższa procedura.

- Do grafu dodane są krawędzie ze znalezionego rozszerzenia. O(n)
- Dla każdej krawędzi z rozszerzenia wykonana jest poniższa logika. O(n)
  - Wyznaczona jest liczba iteracji będąca iloczynem retryFactor i ilości krawędzi w grafie. O(1)
  - Dla każdej iteracji wykonane są poniższe kroki. O(n^2)
    - Wykonane jest podążanie losową ścieżką w taki sposób jak dla przypadków, w których nie było dodatkowych krawędzi w rozszerzeniu grafu, z tym, że w pierwszym przejściu zagwarantowane jest przejście po krawędzi z rozszerzenia, a wierzchołkiem startowym jest wierzchołek, z którego wychodzi krawędź z rozszerzenia grafu. O(n^2)
    - Jeśli wyznaczona ścieżka odwiedza wszystkie wierzchołki i istnieje krawędź w grafie z ostatniego do wierzchołka 0 to ścieżka ta jest dodawana do zbioru unikalnych cykli Hamiltona znalezionego rozszerzenia grafu. O(1)

Dowód poprawności - Znajdowanie minimalnego rozszerzenia grafu: Uwagi Jako, że jest to algorytm aproksymacyjny, w którym kosztem zwiększenia wydajności jest brak gwarancji, że znalezione rozszerzenie jest rozszerzeniem minimalnym, to zamiast dowodzenia, że algorytm znajduje minimalne rozszerzenie grafu zostanie pokazane, że algorytm znajduje rozszerzenie grafu do grafu zawierającego cykl Hamiltona.

Dowód poprawności - Znajdowanie minimalnego rozszerzenia grafu Przyjęto, że $G = (V, E)$, $|V| = n$, $|E| = m$.

Rozważmy pierwsze powtórzenie, wykonana jest wtedy procedura podążania losową ścieżką.
W trakcie wywołania tej procedury został wyznaczony zbiór krawędzi, który w pierwszej iteracji zostanie przypisany jako najmniejsze rozszerzenie. Każde następne wywołanie iteracji co najwyżej może podać mniejszy zbiór krawędzi, ale również mający tą własność, co poprzedni zbiór, czyli **będący rozszerzeniem grafu do grafu z cyklem Hamiltona**.

Dowód poprawności - Znajdowanie minimalnego rozszerzenia grafu: wykonanie procedury podążania losową ścieżką zwróci rozszerzenie grafu do grafu z cyklem Hamiltona Jest to procedura rekurencyjna, która kończy się dopiero wtedy, jeśli niemożliwe będzie wybranie losowego wierzchołka spośród nieodwiedzonych, który ma najmniej krawędzi wchodzących z nieodwiedzonych wierzchołków. Oznacza to, że w trakcie wykonania procedury został odwiedzony każdy wierzchołek.
Za każdym razem, kiedy wybierany jest następny wierzchołek, to jest on wybierany spośród nieodwiedzonych wierzchołków.
Z poprzednich obserwacji można zauważyć, że każdy wierzchołek jest odwiedzany dokładnie raz.
Kolejność odwiedzanych wierzchołków tworzona przy każdym wywołaniu rekurencyjnym poprzez zapisanie odwiedzanego wierzchołka na koniec *obecnie tworzonej ścieżki*. Także kolejność odwiedzonych wierzchołków wraz z krawędzią od ostatniego do pierwszego wierzchołka (krawędź ta, jeśli nie istniała w grafie to tuż przed zakończeniem rekurencji jest ona dodawana do rozszerzenia grafu) tworzą cykl Hamiltona w grafie, do którego dodawano krawędzie za każdym razem, kiedy nie udało się wybrać losowego sąsiedniego i nieodwiedzonego wierzchołka. Te dodane krawędzie są właśnie szukanym rozszerzeniem grafu.

Dowód poprawności - Znajdowanie liczby cykli hamiltona: Uwagi Jako, że jest to algorytm aproksymacyjny to tracimy gwarancję, że zostaną odnalezione wszystkie cykle Hamiltona, można natomiast pokazać, że znajdowane listy wierzchołków dodawane do zbioru unikalnych cykli rzeczywiście reprezentują istniejące w grafie cykle Hamiltona.

Dowód poprawności - Znajdowanie liczby cykli hamiltona W przypadku wykonania tego algorytmu występują dwie sytuacje:

- szukanie cykli dla grafu, który miał cykl Hamiltona bez rozszerzania,
- szukanie cyklu dla grafu, do którego zostało dodane rozszerzenie zawierające przynajmniej jedną krawędź.

Natomiast w obu przypadkach wykonywana jest procedura podążania losową ścieżką oraz dodanie tej ścieżki do zbioru unikalnych cykli Hamiltona.
Różnica polega tylko na ilości prób szukania tych cykli oraz wyborze wierzchołka startowego (oraz w tym, że dla przypadku, gdy rozszeszenie grafu zawiera krawędzie drugi

w kolejności odwiedzany wierzchołek będzie drugim wierzchołkiem do którego wchodzi rozważana krawędź z rozszerzenia grafu).
Należy więc się przyjżeć procedurze podążania losową ścieżką, aby uwidocznić, że dodawane ścieżki do zbioru unikalnych cykli Hamiltona rzeczywiście reprezentują istniejący w grafie cykl Hamiltona.
Procedura podążania losową ścieżką jest rekurencyjna, odwiedzane są tylko wcześniej nieodwiedzone wierzchołki, a jako następny w kolejności wierzchołek wybierany jest tylko wierzchołek sąsiadujący.
Jeśli takiego sąsiadującego wierzchołka nie ma to rekurencja zostaje zakończona.
Wyznaczona ścieżka reprezentuje cykl Hamiltona jeśli więc wszystkie wierzchołki zostały odwiedzone i istnieje krawędź od ostatniego do pierwszego wierzchołka (z tej ścieżki) w grafie.
A dokładnie te warunki są dodatkowo sprawdzane po wywołaniu procedury podążania losową ścieżką przy próbie dodania ścieżki do zbioru unikalnych cykli Hamiltona.
Tak więc rzeczywiście wszystkie ścieżki znajdujące się w zbiorze cykli Hamiltona poprawnie reprezentują cykle Hamiltona.

5 Znajdowanie maksymalnego cyklu w grafie oraz liczby maksymalnych cykli w grafie

5.1 Algorytm dokładny

Inicjalizacja algorytmu rekurencyjnego: Algorytm polega na zbadaniu każdej możliwej ścieżki o różnych wierzchołkach w grafie, znalezieniu w ten sposób cykli oraz wyznaczeniu najdłuższego z nich. Pierwszym krokiem jest inicjalizacja zmiennej current_path, która pomaga monitorować aktualny ciąg wywołań rekurencyjnych, reprezentuje ona aktualnie badaną ścieżkę w grafie. Ponadto, tworzone są zmienne longest_cycles do przechowywania znalezionych najdłuższych cykli w grafie, a także longest_cycle_length do zapisywania informacji o długości najdłuższego znalezionego dotychczas cyklu. Algorytm jest wywołany iteracyjnie dla każdego wierzchołka grafu, na początku i-tej iteracji do czyszczona jest zmienna current_path, po czym zostaje do niej dodany wierzchołek i. Następnie dla każdego wierzchołka j, dla którego istnieje krawędź wchodząca z wierzchołka i, wywoływana jest następująca sekwencja instrukcji. Najpierw wierzchołek j zostaje dodany do zmiennej current_path. Następnie wywoływana jest funkcja rekurencyjna. Po wyjściu z tejże funkcji wierzchołek j zostaje zdjęty z current_path.

Funkcja rekurencyjna - kolejne wywołania: Funkcja rekurencyjna na wejściu otrzymuje aktualną ścieżkę current_path. Pierwszym krokiem jest sprawdzenie, czy istnieje krawędź wychodząca z ostatniego wierzchołka ze ścieżki current_path do pierwszego wierzchołka na tejże ścieżce. Jeśli tak, to zostaje wywołany blok kodu opisany w następnym akapicie dotyczącym warunku stopu. Następnie dla każdego wierzchołka k, który nie znajduje się na liście current_path oraz dla którego istnieje krawędź wchodząca z ostatniego wierzchołka z current_path, wywoływana jest następująca sekwencja instrukcji. Najpierw wierzchołek k zostaje dodany do zmiennej current_path. Następnie wywoływana jest funkcja rekurencyjna. Po wyjściu z tejże funkcji wierzchołek k

zostaje zdjęty z current_path.

Funkcja rekurencyjna – warunek stopu: Jeśli w badanym grafie istnieje krawędź z ostatniego wierzchołka ze ścieżki current_path do pierwszego wierzchołka tej ścieżki, daje to możliwość zamknięcia cyklu. W tej sytuacji przed dalszymi wywołaniami rekurencyjnymi sprawdzamy utworzony w ten sposób cykl. Jeśli jego długość jest mniejsza niż maksymalna dotychczas znaleziona długość cyklu zapisana w longest_cycle_length, nowy cykl pomijamy. Jeżeli długość nowego cyklu jest równa wartości longest_cycle_length, sprawdzamy najpierw, czy dany cykl znajduje się już na liście najdłuższych cykli zapisanych w longest_cycles (mogliśmy już znaleźć ten cykl zaczynając przechodzenie po grafie startując z innego wierzchołka), jeśli nie, to dodajemy nowy cykl do longest_cycles. W sytuacji, gdy znaleziony cykl jest dłuższy od dotychczas najdłuższego, resetujemy listę najdłuższych cykli longest_cycles i dodajemy do niej nowo znaleziony cykl, zaś wartość longest_cycle_length ustawiamy na długość nowego najdłuższego cyklu.

Złożoność: Opisany powyżej dokładny algorytm rekurencyjny ma złożoność O(n!). Związane jest to z faktem, że w pesymistycznym przypadku przechodząc przez kolejne ścieżki analizowana jest każda permutacja wierzchołków w grafie.

5.2 Algorytm aproksymacyjny

Wstępne przekształcenie grafu: Przed rozpoczęciem przeszukiwania grafu w głąb (DFS) wykonywane jest wstępne przekształcenie grafu. W tym etapie usuwane są wszystkie liście (wierzchołki o stopniu 1), a proces ten trwa do momentu, aż graf będzie składał się wyłącznie z cykli. Celem tego kroku jest pominięcie niepotrzebnych wywołań przeszukiwania oraz unikanie „ślepych uliczek" w grafie.

Tablice pomocnicze: Algorytm wykorzystuje następujące tablice pomocnicze:

- **Numbers** – przechowuje odległość wierzchołka od początku przeszukiwania. Dzięki temu można określić długość cyklu po napotkaniu już odwiedzonego wierzchołka.
- **Parent** – przechowuje numer wierzchołka, z którego dotarliśmy do bieżącego wierzchołka. Wierzchołki nieodwiedzone i początek przeszukiwania mają przypisaną wartość -1.
- **Visited** – przechowuje informacje o tym, czy dany wierzchołek został już odwiedzony podczas przeszukiwania.

Zmienne pomocnicze: Dodatkowo w algorytmie wykorzystywane są:

- **lastLargestCycle** – przechowuje długość najdłuższego dotychczas znalezionego cyklu.
- **Resulting_cycles** – przechowuje wszystkie cykle maksymalnej długości znalezione do tej pory.

Rozpoczęcie przeszukiwania: Dla każdego wierzchołka grafu odbywa się oddzielne przeszukiwanie w głąb w którym:

- Resetowane są tablice pomocnicze (poza informacjami o znalezionych cyklach).
- Rozpoczynane jest przeszukiwanie w głąb od bieżącego wierzchołka, przypisując mu wartość 1 w tablicy **Numbers**.
- Kolejne wierzchołki są wybierane w losowej kolejności, co zwiększa różnorodność przeszukiwanych tras.

Przeszukiwanie w głąb: Podczas przeszukiwania:

- Każdemu nieodwiedzonemu sąsiadowi przypisywany jest bieżący wierzchołek jako rodzic w tablicy **Parent**.
- Tablica **Numbers** jest aktualizowana — bieżącemu wierzchołkowi przypisywana jest wartość o 1 większa niż jego rodzic (z wyjątkiem wierzchołka początkowego).
- Tablica **Visited** jest aktualizowana, aby oznaczyć bieżący wierzchołek jako odwiedzony.

Wykrywanie cyklu: Jeśli w trakcie przeszukiwania napotkamy już odwiedzony wierzchołek, algorytm:

- Oblicza długość cyklu, korzystając z tablicy **Numbers**.
- Jeśli długość cyklu jest większa lub równa najdłuższemu dotychczas znalezionemu cyklowi:
  - Cykl jest weryfikowany poprzez cofanie się po tablicy **Parent**.
  - Jeśli cykl jest poprawny i dłuższy, aktualizowana jest zmienna **lastLargestCycle**, a zbiór cykli jest resetowany.
  - Jeśli cykl jest równy, jest dodawany do zbioru cykli maksymalnej długości, o ile nie jest duplikatem.

Zakończenie przeszukiwania: Po zakończeniu przeszukiwania grafu z każdego możliwego wierzchołka algorytm zwraca:

- Długość najdłuższego znalezionego cyklu.
- Kolekcję wszystkich cykli o tej długości.

Złożoność obliczeniowa:

Pesymistyczna analiza: W najgorszym przypadku:

- Algorytm wykonuje n przeszukań w głąb.
- W każdej iteracji DFS:

– Każdy wierzchołek (n) i każda krawędź (m) są odwiedzane dokładnie raz.
– Sprawdzanie poprawności cyklu wymaga $O(n)$, a porównanie z kolekcją wynikową $O(n \cdot m)$.

W najgorszym przypadku dla każdej krawędzi wykonujemy $O(n + n \cdot m)$, co daje:

$$O(n(n + m \cdot (n + n \cdot m))) = O(n^3 \cdot m^2).$$

Średnia analiza: W praktyce algorytm znajduje długi cykl już w pierwszych iteracjach, co ogranicza koszt dalszych operacji. Średni koszt wynosi:

$$O(n \cdot (n + m)) = O(n^2 + n \cdot m) = O(n^3).$$

Dowód poprawności:

Cel algorytmu: Jako, że jest to algorytm aproksymacyjny, zamiast dowiedzenia, że algorytm znajduje wszystkie maksymalne cykle w grafie, zostanie pokazane, że algorytm znajduje poprawne cykle w grafie o ile takie istnieją, oraz rozważa bardzo różne trasy, na których mogą istnieć potencjalne cykle.

Usuwanie liści: Proces transformacji grafu, w którym usuwane są krawędzie przylegające do liści, pozostawia graf zawierający jedynie cykle (jeśli graf początkowo zawierał cykle).
W takim grafie, przeszukiwanie DFS z dowolnego wierzchołka:

- Może zakończyć się jedynie napotkaniem już odwiedzonego wcześniej wierzchołka (nie ma "ślepych uliczek").
- Gwarantuje to znalezienie cyklu, o ile w grafie pozostały jakiekolwiek cykle po usunięciu liści.

Z tego wynika, że algorytm zawsze znajdzie przynajmniej jeden cykl, jeśli istnieje w grafie przekształconym przez funkcję usuwającą liście.

Losowy wybór sąsiadów: Losowe wybieranie sąsiadów zwiększa różnorodność tras rozpatrywanych w przeszukiwaniu.
Dzięki temu algorytm ma większe szanse przeanalizować różne potencjalne ścieżki w grafie, podczas gdy:

- Dobieranie na podstawie np. stopnia wierzchołka lub otoczenia wierzchołka powodowałoby, że dla każdego wywołania DFS przeszukiwanie grafu zachowywałoby podobną kolejność wyboru sąsiadów.

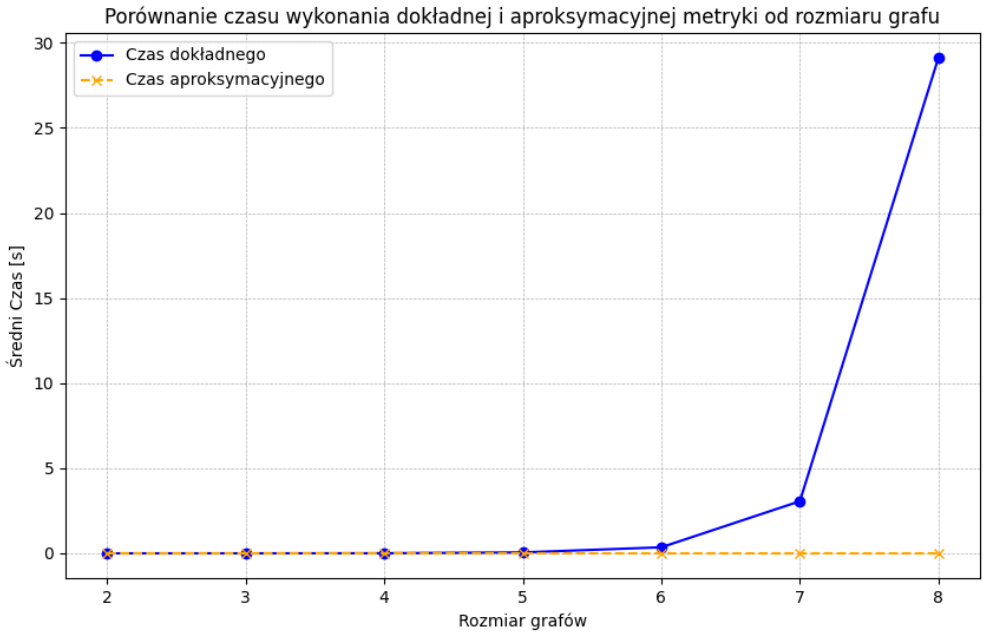
Dla każdego wierzchołka w grafie takich losowań będzie tyle, ile wywołań DFS, czyli tyle, ile wierzchołków w grafie.
Oznacza to, że dla większych grafów rozważane jest więcej możliwości na jeden wierzchołek. Zapewnia to rozważenie bardzo różnych tras przy kolejnych przeszukiwaniach w głąb grafu.

6 Testy wydajności

Dla czterech algorytmów (wszystkich po za liczeniem rozmiaru grafu), zaimplementowaliśmy algorytm dokładny o złożoności wykładniczej oraz algorytm aporksymacyjny wielomianowy. W tej sekcji prezentujemy wyniki testów wydajności porównania wersji algorytmów dla odpowiednich problemów.

6.1 Porównanie algorytmów: Wyznaczanie odległości między grafami dla przyjętej metryki

Do porównania algorytmu aproksymacyjnego oraz dokładnego dla liczenia metryki użyte zostały 42 grafy. Było 6 grup grafów w rozmiarach kolejno 2, 3, 4, 5, 6, 7 oraz 8. Do testów metryki były brane wszystkie możliwe pary grafów tych samych rozmiarów, a na koniec brana była średnia z obliczonych czasów.

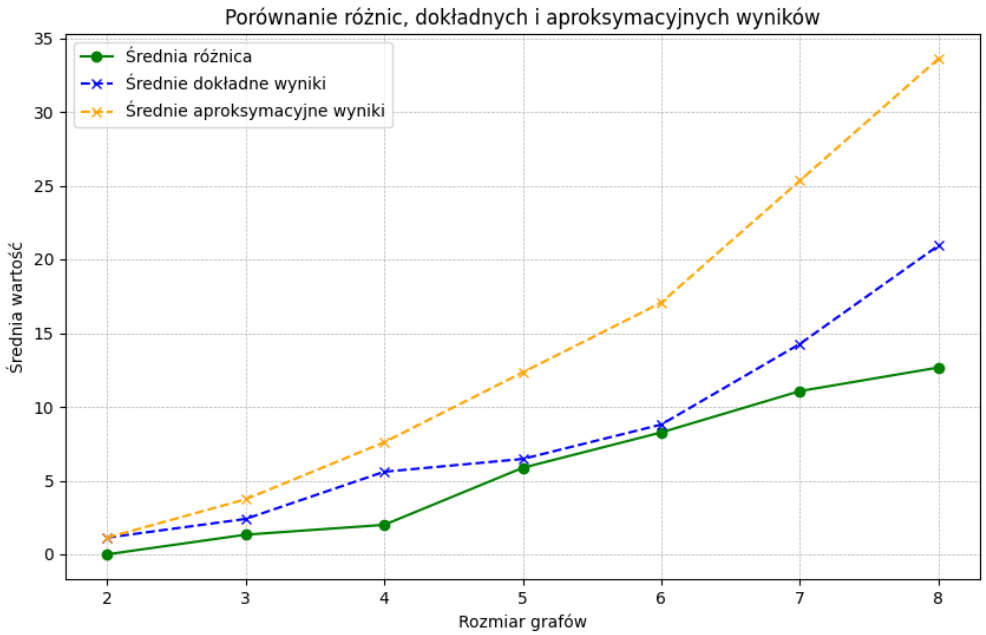


Rysunek 3: Wykres porównania czasu wykonania algorytmu dokładnej metryki oraz algorytmu metryki aproksymacyjnej. Do testów były brane pary grafów o tej samej ilości wierzchołków.

Obserwacje: Algorytm dokładny dla grafów do rozmiaru 8 wykonuje obliczenia w akceptowalnym czasie (tj. poniżej kilku minuty na parę grafów), dla większych rozmiarów grafów czas oczekiwania został uznany przez nas za zbyt duży. Algorytm aproksymacyjny zachowywał bardzo podobne czasy nie przekraczając około 50ms na wykonanie algorytmu dla pary grafów.

Wnisoki: Dla relatywnie małych grafów (do 8 wierzchołków) algorytm dokładny gwarantuje najlepsze wyniki. Lecz dla większych grafów czas wykonania będzie dla większości użytkowników nieakceptowalny, wtedy według nas lepszym sposobem będzie użycie

algorytmu aproksymacyjnego dla którego czas wykonania pozostaje relatywnie mały wraz ze wzrostem rozmiarów grafów.



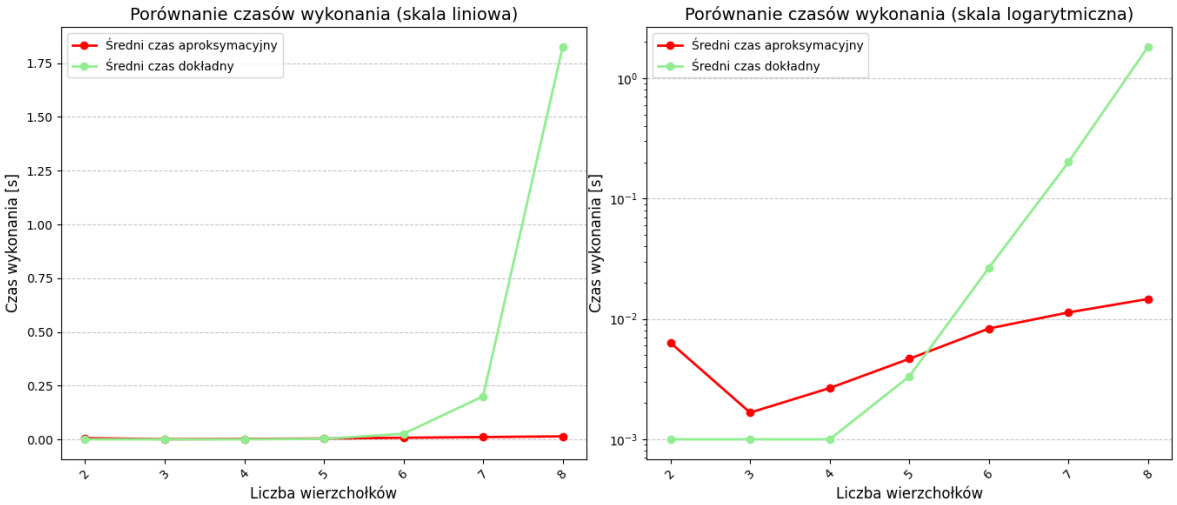
Rysunek 4: Wykres średnich dokładnych i aproksymacyjnych wyników oraz średniej różnicy między dwoma.

Obserwacje: Różnica wyników między algorytmem dokładnym, a aproksymacyjnym rośnie wraz ze zwiększaniem rozmiarów grafów.

Wnioski: Algorytm aproksymacyjny traci na dokładności wraz ze wzrostem rozmiarów grafów, jest to spodziewany efekt. Dla większych grafów istnieje więcej permutacji wierzchołków grafów, a co za tym idzie mniejsza szansa ze obrana przez nas aproksymacja sortowaniem wierzchołów po stopniach będzie blisko dokładnego wyniku.

6.2 Porównanie algorytmów: Znajdowanie minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie

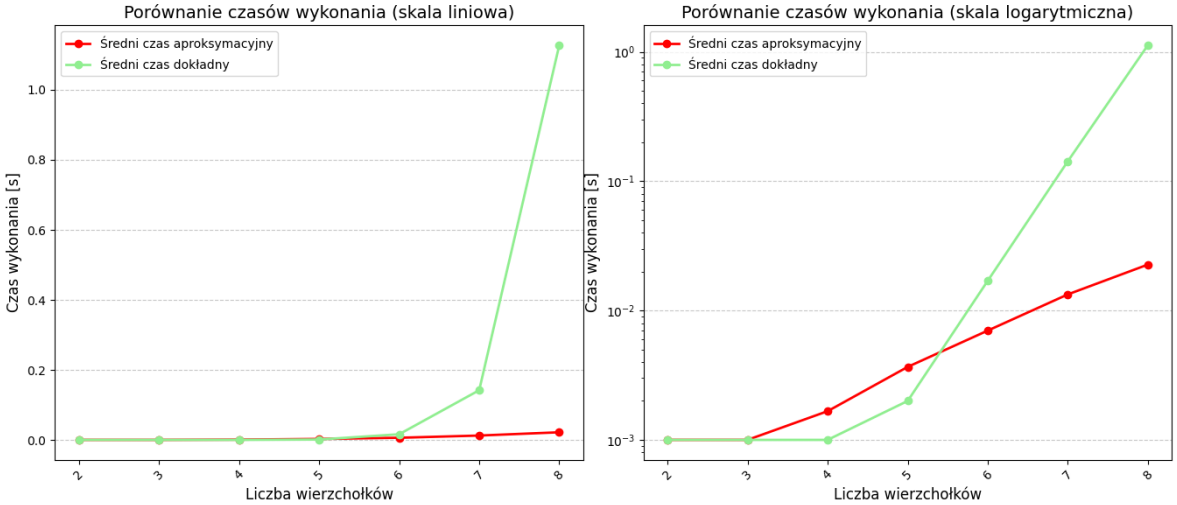
Przy porównaniu z algorytmem dokładnym, algorytm aproksymacyjny będzie wykonywany z parametrem **retryFactor** ustawionym na 15.



Rysunek 5: Wykresy czasów wykonania algorytmów aproksymacyjnego i dokładnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 7 grafach od 2 do 8 wierzchołków.

Obserwacje: W przypadku grafów od 2 do 5 wierzchołków średni czas wykonania funkcji przez algorytm dokładny jest niższy od średniego czasu działania algorytmu aproksymacyjnego. Jednakże, tempo wzrostu dla algorytmu dokładnego jest wykładnicze i dla grafów od 6 wierzchołków średni czas wykonania rośnie kilkukrotnie z każdym dodanym wierzchołkiem, dla 8 wierzchołków wynosi już kilka sekund. Tymczasem czas działania algorytmu aproksymacyjnego rośnie znacznie wolniej, dla grafu o 8 wierzchołkach osiąga wartości rzędu setnych części sekundy.

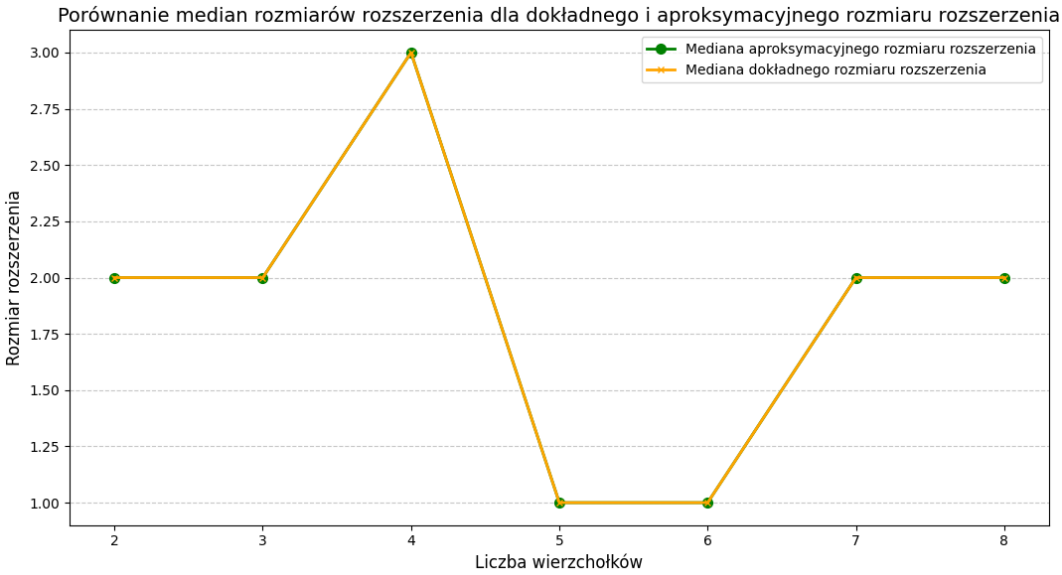
Wnioski: W przypadku grafów bardzo małych prostota obliczeń wykonywanych przez algorytm dokładny sprawia, że czas wykonania jest szybszy niż dla algorytmu wykładniczego, który dodatkowo podejmuje wielokrotne próby. Jednakże, dla większych grafów dokładny algorytm wykładniczy szybko staje się zbyt wolny i niepraktyczny, podczas gdy wielomianowy algorytm aproksymacyjny utrzymuje bardzo dobre czasy wykonania.



Rysunek 6: Wykresy czasów wykonania algorytmów aproksymacyjnego i dokładnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{3}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 7 grafach od 2 do 8 wierzchołków.

Obserwacje: Podobnie jak dla grafów o gęstości 25 procent krawędzi, przy liczbie wierzchołków nie większej niż 5 czasy uzyskiwane przez algorytm aproksymacyjny są nie lepsze niż dla algorytmu dokładnego, są to jednak w obu przypadkach wartości rzędu tysięcznych części sekundy. Dla grafów o 6 lub więcej wierzchołkach widoczny jest wykładniczy wzrost czasu wykonania algorytmu precyzyjnego wraz ze wzrostem liczby wierzchołków. Wzrost czasu działania funkcji aproksymacyjnej jest znacznie mniejszy, dla 8 wierzchołków różnica osiąga już 2 rzędy wielkości.

Wnioski: Czasy działania algorytmu dokładnego są bardzo podobne jak w przypadku grafów o mniejszej gęstości, ma to związek z faktem, że krawędzie nie odgrywają w nim bardzo dużej roli. Ponownie, dla mniejszych grafów algorytm wykładniczy jest nieznacznie szybszy, jednak już dla 8 wierzchołków staje się niezbyt praktyczny przez czas działania i znacznie lepszym rozwiązaniem okazuje się zastosowanie algorytmu aproksymacyjnego o złożoności wielomianowej.

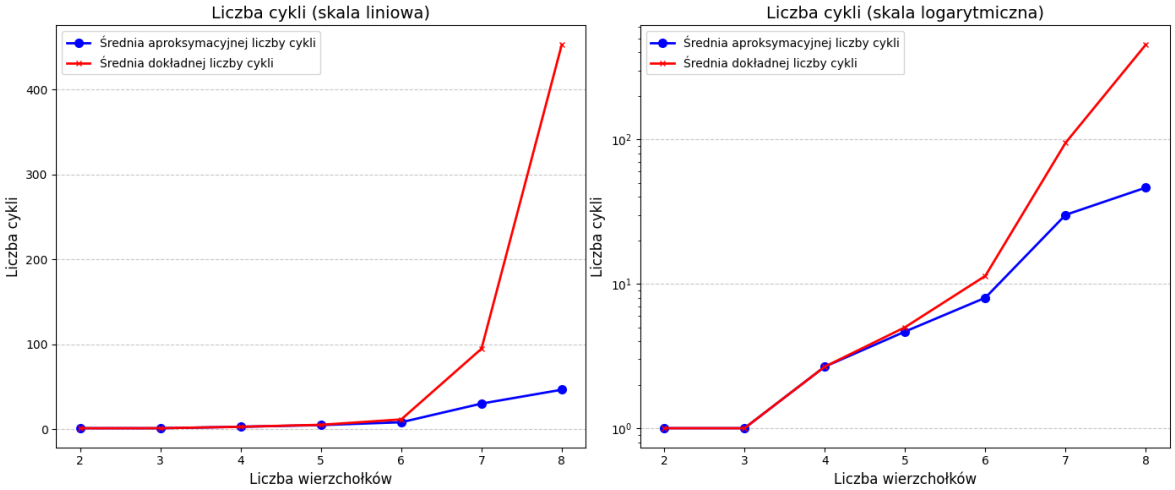


Rysunek 7: Wykres median rozmiarów najmniejszego rozszerzenia wyznaczonych algorytem aproksymacyjnym i dokładnym znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 7 grafach od 2 do 8 wierzchołków.

Obserwacje: Przy dogłębniejszym przyjrzeniu się rezultatom zauważono, że dla wszystkich grafów z danych testowych rozmiar najmniejszych rozszerzeń grafu był równy dla algorytmu aproksymacyjnego i algorytmu dokładnego. Jest to powód, dla którego wykresy median są na siebie nałożone.

Wnioski: Dla niewielkich rozmiarów grafu, algorytm aproksymacyjny z parametrem retryFactor = 15 wydaje się być równie dobry do wyznaczania minimalnego rozszerzenia do algorytm dokładny. Należy jednak pamiętać, że algorytm aproksymacyjny nie daje gwarancji, że znalezione minimalne rozszerzenie na pewno będzie najmniejsze.

Dla grafów gęstych, to znaczy o rozmiarach bliskich $\frac{3}{4}$ wszystkich możliwych krawędzi do utworzenia grafu dla podanej liczby wierzchołków, za każdym razem najmniejsze rozszerzenie miało 0 krawędzi dla obu algorytmów.

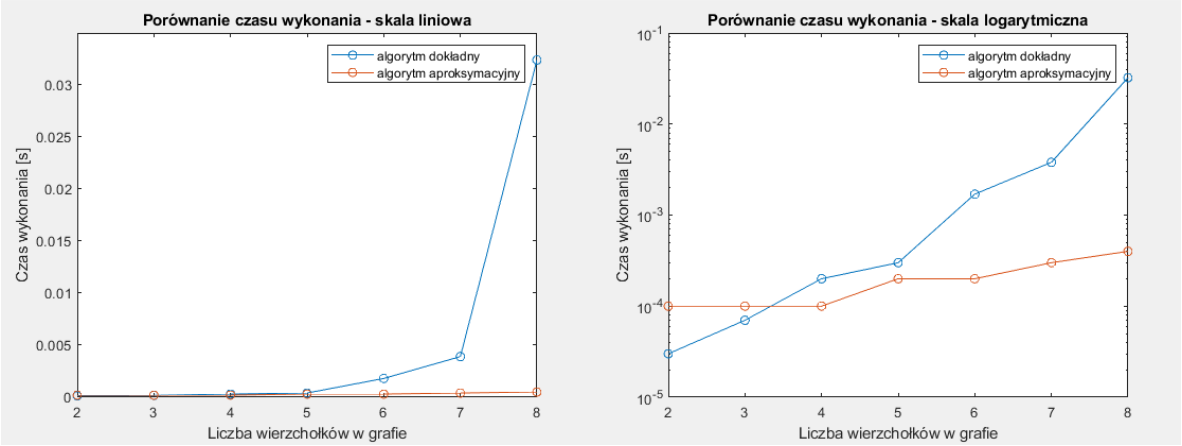


Rysunek 8: Wykres średnich liczb cykli Hamiltona wyznaczonych algorytem aproksymacyjnym i dokładnym znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{3}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Wszystkie rezultaty były wyznaczane dla grafów nierozszerzonych (a więc dla tych samych grafów), ponieważ minimalne rozszerzenie wynosiło za każdym razem 0. Pomiary oparte na 7 grafach od 2 do 8 wierzchołków.

Obserwacje: Dla grafów od 2 do 4 wierzchołków oba algorytmy zwróciły taką samą liczbę znalezionych cykli Hamiltona. Rozbieżności zaczynają się pojawiać dla większych grafów, od 5 wierzchołków w górę. Z początku względne różnice nie są duże, jednak rosną one w bardzo szybkim tempie. Dla grafu o 8 wierzchołkach uzyskane wyniki mają już inny rząd wielkości.

Wnioski: Dla grafów o bardzo małych rozmiarach wykonanie kilku prób algorytmem aproksymacyjnym jest wystarczające, aby znaleźć wszystkie cykle Hamiltona w grafie. Jednakże, wraz ze wzrostem rozmiaru grafu, skuteczność znajdowania cykli Hamiltona w algorytmie aproksymacyjnym znacznie spada. Niemniej jednak, algorytm aproksymacyjny zwraca wyniki w czasie wielomianowym, więc kosztem mniejszej ilości znalezionych cykli Hamiltona można go stosować dla znacznie większych grafów niż w przypadku algorytmu dokładnego.

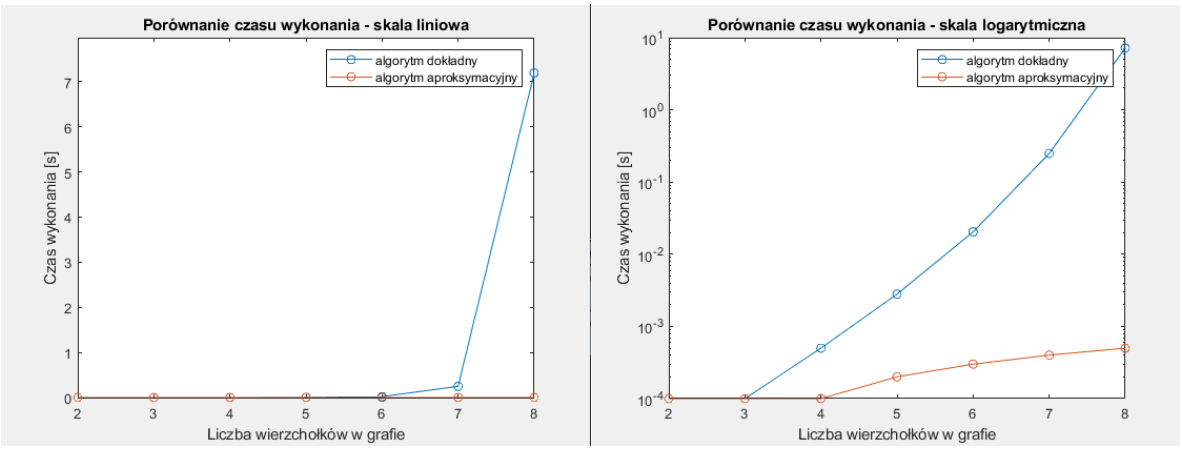
6.3 Porównanie algorytmów: Znajdowanie maksymalnego cyklu w grafie oraz liczby maksymalnych cykli w grafie



Rysunek 9: Wykres średnego czasu wykonania funkcji znajdowania najdłuższych cykli oraz ich liczby przy pomocy algorytmu dokładnego oraz aproksymacyjnego. Dane testowe stanowią losowe grafy regularne bez pętli o liczbie krawędzi równej połowie liczby krawędzi w grafie pełnym o takiej samej liczbie wierzchołków, po jednym grafie od 2 do 8 wierzchołków.

Obserwacje: Dla grafów od 2 do 5 wierzchołków czas wykonania był prawie identyczny, a dla liczby wierzchołków od 6 do 8 czas wykonania algorytmu dokładnego był o rzędy wielkości większy od czasu wykonania algorytmu aproksymacyjnego.

Wnioski: Algorytm apsokrymacyjny skutecznie zmniejsza czas znalezienia największego cyklu w grafie dla zestawu danych testowych i osiąga złożoność wielomianową.

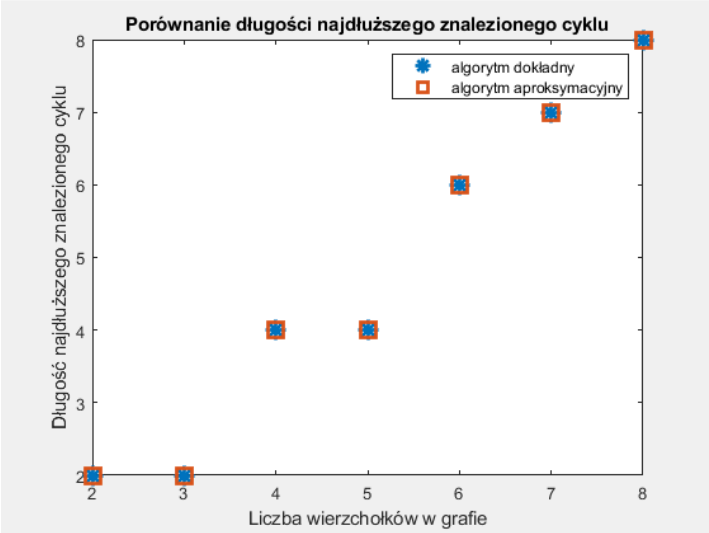


Rysunek 10: Wykres średnego czasu wykonania funkcji znajdowania najdłuższych cykli oraz ich liczby przy pomocy algorytmu dokładnego oraz aproksymacyjnego. Dane testowe stanowią grafy pełne bez pętli, po jednym grafie od 2 do 8 wierzchołków.

Obserwacje: Czas wykonania dla grafów pełnych jest rośnie szybciej niż dla grafów o mniejszej liczbie krawędzi i już od grafu o 4 wierzchołkach jest on klika razy większy dla algorytmu dokładnego niż dla algorytmu aproksymacyjnego, a dla liczby wierzchołków

od 5 do 8 czas wykonania algorytmu dokładnego był już o rzędy wielkości większy od czasu wykonania algorytmu aproksymacyjnego i rósł on o wiele szybciej niż w przypadku grafu o mniejszej liczbie krawędzi z pierwszego testu.

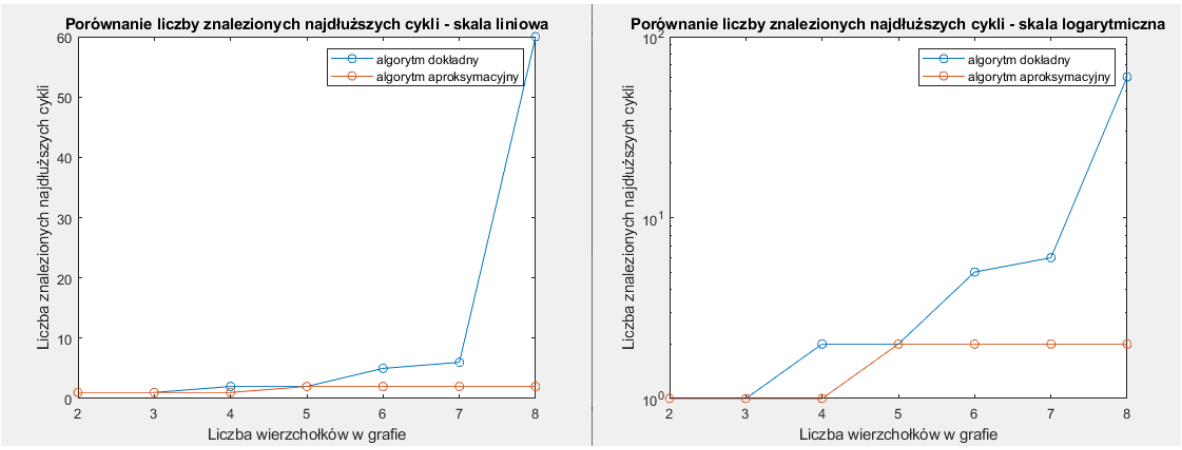
Wnioski: Czas wykonania algorytmu dokładnego jest dużo większy w przypadku grafów o dużej liczbie krawędzi, takich jak grafy pełne, co oznacza że jest niepraktyczny w przypadku dużych grafów o dużej liczbie krawędzi. Algorytm aproksymacyjny wykonywał się wolniej niż w przypadku grafów o mniejszej liczbie krawędzi ale w dalszym ciągu czas wykonania był tego samego rzędu wielkości.



Rysunek 11: Wykres zależności długości najdłuższego znalezionego cyklu w zależności od liczby wierzchołków w grafie, dla algorytmu dokładnego oraz aproksymacyjnego. Dane testowe stanowią losowe grafy regularne bez pętli o liczbie krawędzi równej połowie liczby krawędzi w grafie pełnym o takiej samej liczbie wierzchołków, po jednym grafie od 2 do 8 wierzchołków.

Obserwacje: Dla każdego grafu długość najdłuższego cyklu w grafie jest identyczna dla algorytmu dokładnego i algorytmu aproksymacyjnego.

Wnioski: Algorytm aproksymacyjny skutecznie znajduje najdłuższy cykl w grafie dla grafów zawierających do 8 wierzchołków.

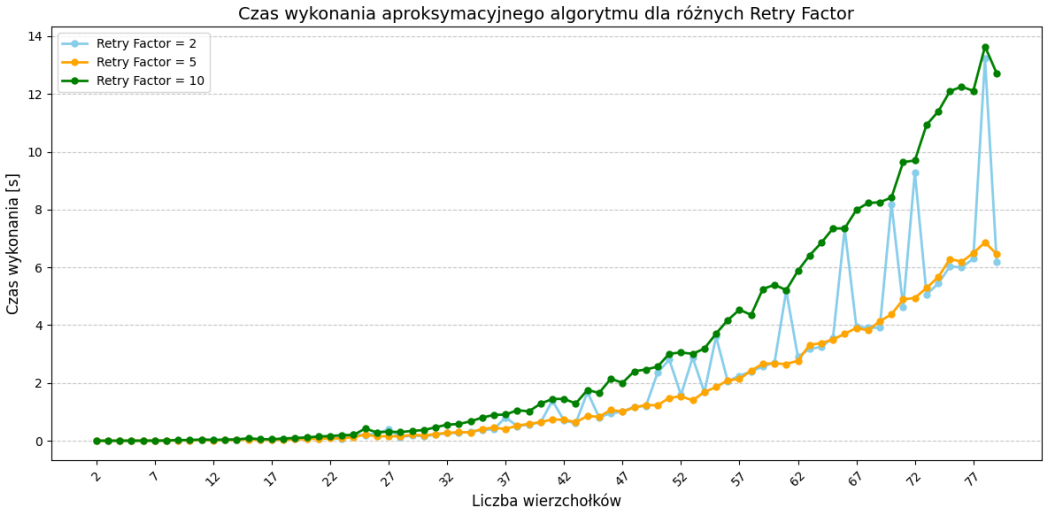


Rysunek 12: Wykres zależności liczby znalezionych cykli o największej długości w zależności od liczby wierzchołków w grafie, dla algorytmu dokładnego oraz aproksymacyjnego. Dane testowe stanowią losowe grafy regularne bez pętli o liczbie krawędzi równej połowie liczby krawędzi w grafie pełnym o takiej samej liczbie wierzchołków, po jednym grafie od 2 do 8 wierzchołków.

Obserwacje: Dla grafów zawierających do 8 wierzchołków algorytm aproksymacyjny znajduje zawsze tylko kilka maksymalnych cykli, przez co dla grafu o 8 wierzchołkach znajduje już o kilkadziesiąc cykli mniej od algorytmu dokładnego.

Wnioski: Algorytm aproksymacyjny dla dużych grafów znajduje o wiele mniej maksymalnych cykli w grafie niż algorytm dokładny.

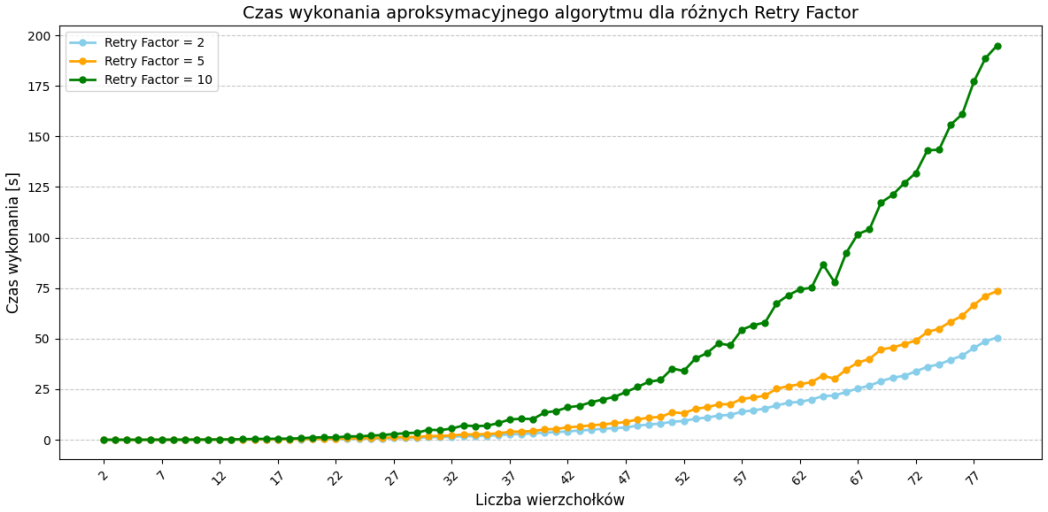
6.4 Testy wydajności dla aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie



Rysunek 13: Wykres czasów wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie dla **retryFactor** = 2, 5, 10. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Algorytm jest zależny od parametru retryFactor, dlatego też warto jest sprawdzić, jak wygląda czas wykonania dla różnych wartości tego parametru. Na rysunku 13 widać, że dla wartości 2 algorytm jest bardzo niestabilny i czasy wykonania zbliżają się dla niektórych grafów do czasów dla parametru retryFactor 10. Widać, że dla małych grafów wybór większej wartości retryFactor nie zwiększa znacząco czasu wykonania algorytmu. Natomiast dla większych liczb wierzchołków (>33), czasy wykonania dla retryFactor 1 i 2 są zbliżone, z tym, że dla niektórych liczb wierzchołków czasy wykonania dla retryFactor są znacząco dłuższe.

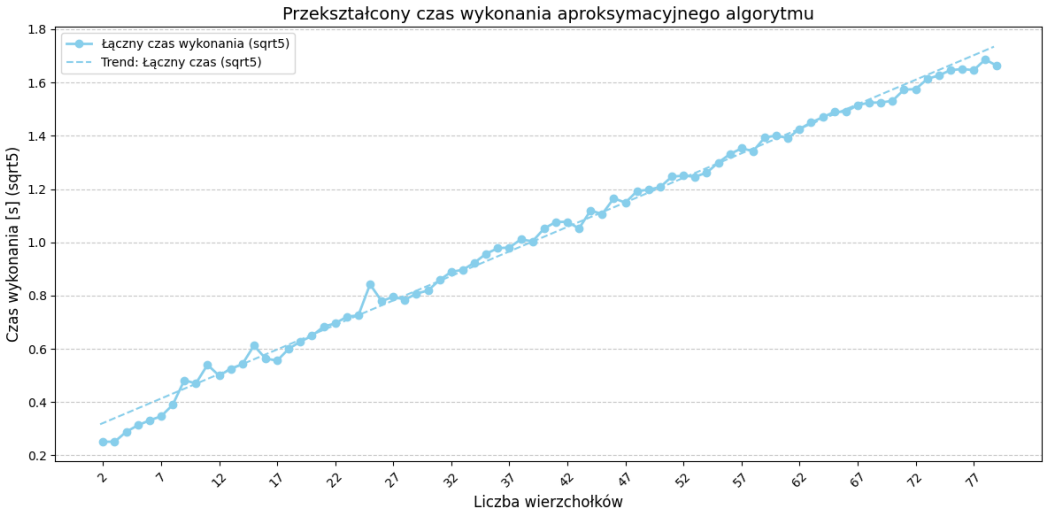
Wnioski: Jeżeli czas wykonania dla grafów rzadkich musi być krótki, ale nie musi być najkrótszy możliwy, to użycie retryFactor = 5, będzie o wiele lepszym wyborem niż wybór wartości 2, ponieważ czasy wykonania są zbliżone (a w ogólności nawet czasy dla retryFactor = 5 są krótsze, bo nie występują nagłe zwiększania czasu wykonania do poziomów z retryFactor = 10), a jak później zostanie zaobserwowane, wykonanie z retryFactor = 5 zwraca bardziej jakościowe rozwiązania. Dla grafów rzadkich wykonania algorytmów są akceptowalne dla wszystkich przetestowanych wartości retryFactor, ponieważ czasy wykonania nie przekraczają minuty.



Rysunek 14: Wykres czasów wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie dla **retryFactor** = 2, 5, 10. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{3}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Wykres na rysunku 14 pokazuje, że dla grafów gęstych, wykonanie algorytmu aproksymacyjnego jest stabilne również dla parametru retryFactor = 2. W przeciwieństwie do grafów rzadkich, które przetestowano na rysunku 13 wykonanie algorytmu jest znacząco krótsze dla większych liczb wierzchołków (>62) przy wybraniu retryFactor = 2 zamiast 5. Należy też zauważyć, że dla retryFactor = 10, czas wykonania przekracza minutę dla grafów z liczbą wierzchołków większą niż 60, co można uznać za czas nieakceptowalny. Dla retryFactor = 5 czas nieakceptowalny jest osiągany dopiero dla liczby wierzchołków większej niż 76.

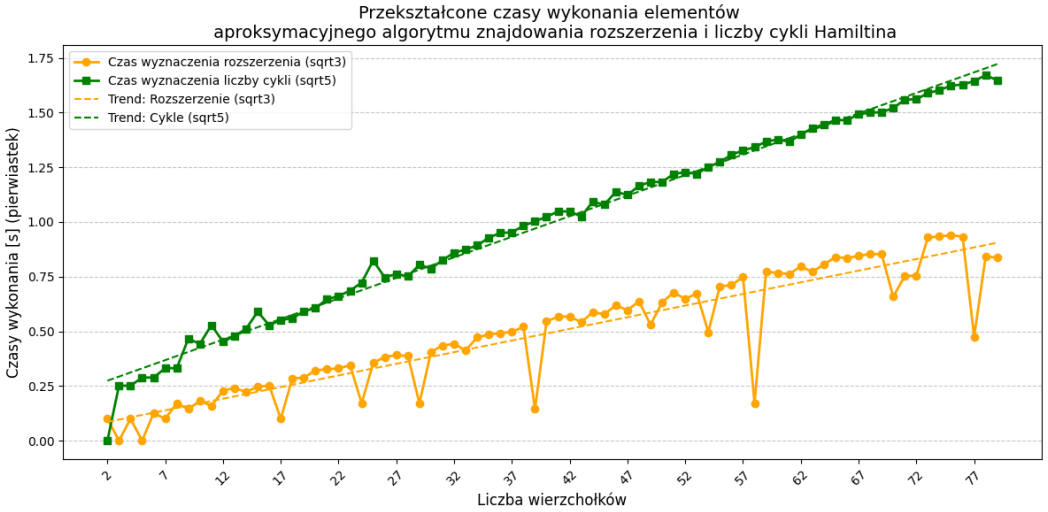
Wnioski Jeżeli przy wykonywaniu algorytmu ważne jest, aby szukać rozwiązań dla większych grafów (liczba wierzchołków > 60) to zalecane jest, aby korzystać z mniejszej wartości retryFactor. Dla przetestowanych wartości widać, że lepszym rozwiązaniem będzie wybór retryFactor = 2, ponieważ nawet dla grafu z liczbą wierzchołków 79 (maksymalną przetestowaną) czas wykonania nie przekroczy minuty.



Rysunek 15: Wykres czasów wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie przekształcony o pierwiastek z 5. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Wykres czasów wykonania przekształcony o pierwiastek z 5 przedstawiony na liniowych osiach jest zbliżony do liniowej linii trendu.

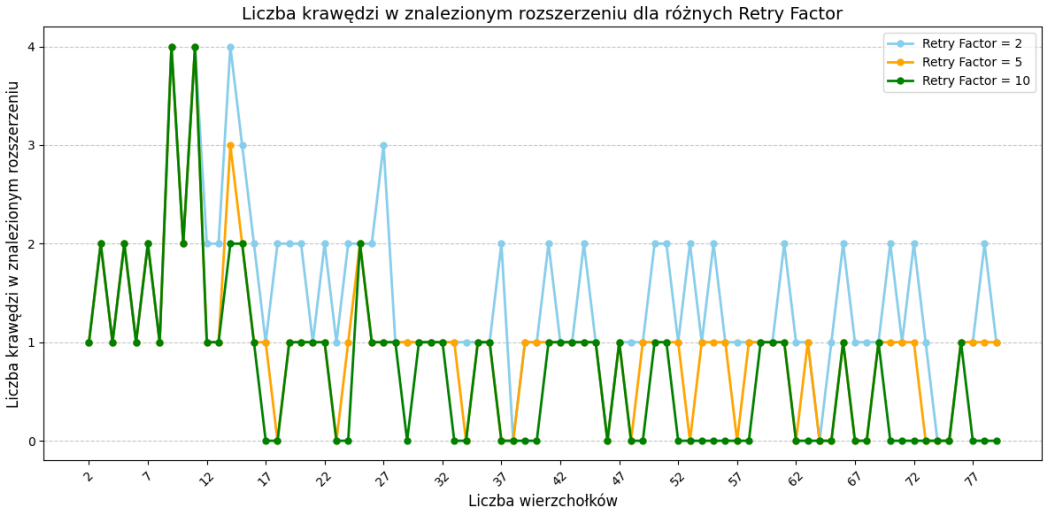
Wnioski: Układanie się przekształconego wykresu w prostą dodatkowo utwierdza, że złożoność całego algorytmu to $O(n^5)$.



Rysunek 16: Wykres przekształconych czasów wykonania składowych algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie. Cześć znajdująca rozszerzenie przekształcona o pierwiastek z 3, a cześć znajdująca liczbę cykli Hamiltona przekształcona o pierwiastek z 5. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Wykres czasów wykonania części znajdującej liczbę cykli Hamiltona przekształcony o pierwiastek z 5 przedstawiony na liniowych osiach jest zbliżony do liniowej linii trendu. Linia trendu dla części szukającej minimalne rozszerzenie grafu jest nieco poniżej większości punktów wykresu przekształconych czasów wykonania, natomiast wynika to z tego, że uwzględniane są w niej również czasy wykonania, dla których została zastosowana optymalizacja, polegająca na szybszym zakończeniu algorytmu jeżeli zostanie znaleziony cykl Hamiltona nie korzystający z dodawanych krawędzi. Można też przyjąć, że punkty wykresu (dla których nie została zastosowana optymalizacja) układają się wzdłuż pewnej linii.

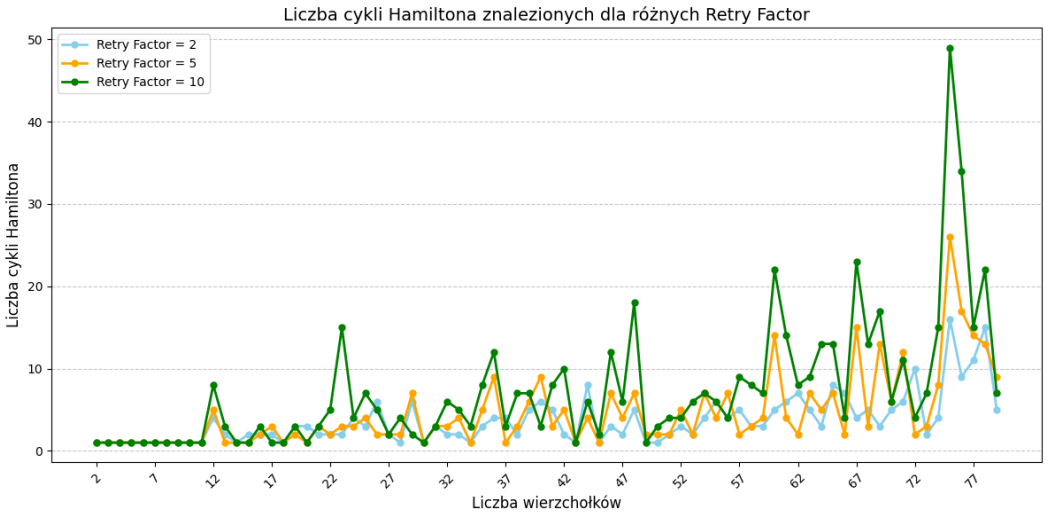
Wnioski: Fakt, że przedstawione przekształcone wykresy układają się w sposób zbliżony do linii prostych sugeruje, że rzeczywiście złożoność części znajdującej rozszerzenie to $O(n^3)$, a części znajdującej liczbę cykli Hamiltona to $O(n^5)$.



Rysunek 17: Wykres liczby krawędzi w znalezionym minimalnym rozszerzeniu podczas wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie dla **retryFactor** = 2, 5, 10. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Algorytm nigdy nie zwraca rozszerzenia grafu zawierającego mniejszą ilość krawędzi niż najmniejsze rozszerzenie gwarantujące istnienie cyklu Hamiltona w grafie. Oznacza to, że jeśli dla różnych retryFactor rozmiar znalezionego rozszerzenia jest różny, to większe rozszerzenia grafu są dalej od optymalnego rozwiązania. Na rysunku 17 widać, że użycie większego retryFactor prawdopodobnie zapewni bardziej jakościowe rozwiązanie w sensie zwracającym rozszerzenie, które ma liczbę krawędzi bliższą lub równą liczbie krawędzi najmniejszego możliwego rozszerzenia. Dla retryFactor 2 znalezione rozszerzenie ma aż o 2 krawędzie więcej niż rozszerzenie znalezione dla retryFactor. Użycie retryFactor = 5 dla rozważanych danych pozwalało uzyskiwać rozszerzenia, które miały co najwyżej o jedną krawędź więcej względem rozwiązań uzyskanych z użyciem retryFactor = 10.

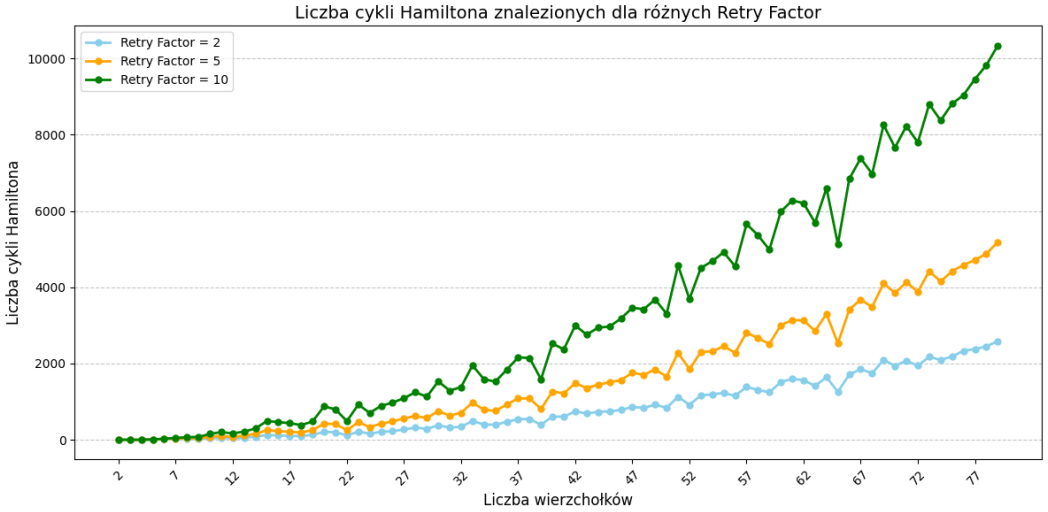
Wnioski: Charakterystyka działania algorytmu zaobserwowana na wykresie 17 sugeruje, że w przypadku, kiedy ważne jest zarówno szybkie wykonanie, jak i bardziej jakościowe rozszerzenia grafu (to znaczy mające ilość krawędzi bliższą minimalnej liczbie dodanych krawędzi potrzebnych do utworzenia cyklu Hamiltona w grafie) to wybranie retryFactor = 5 będzie najrozsądniejsze. Generalnie można przyjąć, że wybór większego retryFactor pozwoli uzyskać bardziej jakościowe rozszerzenia grafu.



Rysunek 18: Wykres liczb cykli Hamiltona znalezionych podczas wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie dla **retryFactor** = 2, 5, 10. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{1}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Można zauważyć, że liczba znalezionych cykli dla retryFactor wynoszącego 5 lub 2 jest zbliżona. Można też zauważyć, że liczba znalezionych cykli dla retryFactor = 10 jest większa.

Wnioski: Fakt, że liczba znalezionych cykli nie zwiększa się przy zwiększeniu parametru retryFactor do 5 najpewniej wynika z tego, że ta część algorytmu odpowiedzialna za znajdowanie liczby cykli zależy od ln(retryFactor) zaokrąglonego w dół, co daje tą samą wartość dla 5 i 2. Dlatego też jeżeli przy wykonaniu algorytmu dla rzadkich grafów istotne jest, aby uzyskać większą liczbę cykli Hamiltona w rozszerzonym grafie, należy podawać wartości retryFactor większą niż 7.



Rysunek 19: Wykres liczb cykli Hamiltona znalezionych podczas wykonania algorytmu aproksymacyjnego znajdowania minimalnego rozszerzenia do grafu zawierającego cykl Hamiltona oraz liczbę cykli Hamiltona w rozszerzonym grafie dla **retryFactor** = 2, 5, 10. Dane pomiarowe to grafy, o rozmiarach bliskich $\frac{3}{4}$ wszystkich możliwych krawędzi do utworzenia grafie dla podanej liczby wierzchołków. Pomiary oparte na 78 grafach od 2 do 79 wierzchołków.

Obserwacje: Im większy retryFactor, tym większa liczba znalezionych cykli Hamiltona w rozszerzonym grafie.

Wnioski: Jeżeli przy wykonaniu algorytmu dla gęstych grafów ważne jest, aby uzyskać większą liczbę znalezionych cykli Hamiltona w rozszerzonym grafie, to należy użyć większej wartości retryFactor.

7 Wnioski podsumowujace

Realizując projekt, zaimplementowano kluczowe algorytmy pozwalające dokonać analizy strukturalnej grafów i porównywania grafów. Implementując algorytmy, można było zastosować w praktyce wiedzę z teorii grafów.

Wszystkie algorytmy dokładne w stworzonych implementacjach mają złożoność obliczeniową większą niż wielomianową. Dlatego też ich użyteczność okazała się być znikoma dla większych grafów. Przykładowo już dla grafów z kilkunastoma wierzchołkami czas wykonania algorytmów dokładnych był nieakceptowalny. Pomimo tego algorytmy te zachowują tę przewagę, że gwarantują zwrócenie optymalnych wyników.

Natomiast algorytmy aproksymacyjne zostały zaimplementowane w ten sposób, że ich złożoność obliczeniowa jest wielomianowa. Przekłada się to na o wiele szybsze czasy wykonania algorytmów. Umożliwia to uzyskiwanie wyników w rozsądnych czasach dla większych grafów. Algorytmy te zostały oparte o podejmowanie decyzji niegwarantujących uzyskania najbardziej optymalnych wyników, co powoduje utratę dokładności, ale pozwoliło uzyskać szybsze wykonania. Jeżeli szybkość czasu wykonania jest kluczowa, to dodatkowo w aproksymacyjnym algorytmie znajdującym rozszerzenie grafu do grafu zawierającego cykl Hamiltona można zmniejszyć wartość parametru retryFactor. Oczywiście będzie to powodować uzyskiwanie mniej optymalnych wyników.

Użyteczność algorytmów dokładnych i aproksymacyjnych można podsumować następującymi stwierdzeniami. Algorytmy dokładne są jedynym wyborem, jeżeli gwarancja uzyskania optymalnego rozwiązania jest wymagana. Natomiast jeżeli ta gwarancja nie jest konieczna, to algorytmy aproksymacyjne okażą się użyteczną alternatywą ze względu na o wiele szybsze czasy wykonania.