Poszukiwanie największej kliki w grafie

Anna Stępień
Adam Stelmaszczyk

Spis treści

1 Zadanie

Kliką grafu nazywamy podgraf, w którym każde dwa wierzchołki są ze sobą połączone. Maksymalną kliką nazywamy klikę, do której nie można dodać ani jednego wierzchołka więcej, tak aby razem z nią nadal tworzył klikę. Największą kliką nazywamy klikę o największej liczbie wierzchołków. Celem zadania jest implementacja wybranego algorytmu znajdującego największa klikę w grafie oraz analiza otrzymanych wyników.

2 Założenia

Realizowana aplikacja będzie pracowała w trybie konsolowym i będzie przyjmowała pliki z danymi przekazane na strumień wejściowy.
W projekcie zostanie wykorzystany zmodyfikowany algorytm Brona–Kerboscha [1], dokładniej opisany w sekcji 3. Do implementacji zadania wykorzystany zostanie język Java.

2.1 Dane wejściowe

Wejściem dla algorytmu jest graf nieskierowany dany macierzą o n wierszach i n kolumnach:

$$\begin{matrix} q_{0,0} & q_{1,0} & \cdots & q_{n-1,0} \\ q_{0,1} & q_{1,1} & \cdots & q_{n-1,1} \\ \vdots & \vdots & \ddots & \vdots \\ q_{0,n-1} & q_{1,n-1} & \cdots & q_{n-1,n-1} \end{matrix} , q_{i,j} \in \{0,1\}, 0 \leqslant i,j < n$$

$q_{i,j}$ równe 0 oznacza, że wierzchołki i oraz j nie są połączone krawędzią. W przeciwnym razie, wierzchołki są połączone.

Macierz jest dana w pliku tekstowym, w którym kolejne $q_{i,j}$ w wierszu j są oddzielone co najmniej jednym znakiem białym. Przez znak biały rozumiemy spację lub tabulator. $q_{i,j}$ różne od 0 będą traktowane jak 1.
Poniżej przedstawiono przykładowy, poprawny plik wejściowy.

```
0 1 0 0
1 0 1 1
0 1 0 1
0 1 1 0
```

2.2 Dane wyjściowe

Wyjściem jest niepusty zbiór numerów wierzchołków, które tworzą największą klikę w podanym grafie. Wierzchołki numerujemy od 0 do $n-1$. W grafie może istnieć więcej niż jedna największa klika. W takim przypadku algorytm zwróci pierwszą ze znalezionych klik.

2.3 Sytuacje wyjątkowe

Problemami, które mogą wystąpić podczas działania aplikacji są:

- błędny format danych wejściowych,
- przepełnienie stosu spowodowane zbyt głębokim poziomem rekurencji.

  W przypadku, gdy algorytm otrzyma na wejściu błędne dane np. liczba wierszy macierzy będzie niezgodna z zadeklarowaną na początku pliku z danymi, użytkownik zostanie poinformowany o zaistniałej sytuacji a dalsze działanie programu zostanie przerwane.

  Ze względu na rekurencyjny charakter algorytmu Brona–Kerboscha może się zdarzyć, iż dla pewnych danych wejściowych algorytm nie będzie w stanie zwrócić wyniku ze względu na ograniczoną pojemność stosu. Próbą rozwiązania tego problemu mogłaby być iteracyjna implementacja algorytmu.

3 Algorytm

Algorytm Brona–Kerboscha jest rekurencyjnym algorytmem z nawrotami, który umożliwia poszukiwanie maksymalnych klik w zadanym grafie niezorientowanym. Domyślnie algorytm zwraca wszystkie maksymalne kliki. W algorytmie wprowadzona zostanie zmiana, dzięki której zwracana będzie największa ze znalezionych maksymalnych klik, charakteryzująca się największą liczbą wierzchołków.

3.1 Pseudokod

Na poniższym listingu przedstawiona została podstawowa wersja algorytmu Brona–Kerboscha.
Poniżej przedstawiona została zmodyfikowana wersja algorytmu, która zostanie wykorzystana do realizacji zadania.

1 Algorytm Brona–Kerboscha (wersja podstawowa)

```
1: compsub ← ∅
2: candidates ← V(G)
3: not ← ∅
4: cliques ← ∅
5: function BRON_KERBOSCH(compsub, candidates, not)
6:     if candidates = ∅ and not = ∅ then
7:         cliques ← cliques ∪ {compsub}                    ▷ Maksymalna klika
8:     else
9:         for each v in candidates do
10:            candidates ← candidates \ {v}
11:            new_compsub ← compsub ∪ {v}
12:            new_candidates ← candidates ∩ neighbors(v)
13:            new_not ← not ∩ neighbors(v)
14:            BRON_KERBOSCH(new_compsub, new_candidates, new_not)
15:            compsub ← compsub ∪ {v}
16:        end for
17:    end if
18: end function
```

2 Algorytm Brona–Kerboscha (wersja rozszerzona)

```
1: compsub ← ∅
2: candidates ← V(G)
3: not ← ∅
4: biggest_clique ← ∅
5: function BRON_KERBOSCH(candidates, not)
6:     if candidates = ∅ and not = ∅ then
7:         if size(biggest_clique) < size(compsub) then
8:             biggest_clique ← compsub                     ▷ Największa klika
9:         end if
10:    else
11:        pivot ← vertex_with_maxdeg(candidates ∪ not)
12:        candidates_to_check ← candidates \ neighbors(pivot)
13:        for each v in candidates_to_check do
14:            compsub ← compsub ∪ {v}
15:            candidates ← candidates \ {v}
16:            new_candidates ← candidates ∩ neighbors(v)
17:            new_not ← not ∩ neighbors(v)
18:            BRON_KERBOSCH(new_candidates, new_not)
19:            compsub ← compsub \ {v}
20:            not ← not ∪ {v}
21:        end for
22:    end if
23: end function
```

3.2 Opis działania

Istotą działania przedstawionego algorytmu jest utrzymywanie trzech rozłącznych zbiorów: *compsub*, *candidates* oraz *not*.
Algorytm Brona–Kerboscha znajduje maksymalne kliki składające się ze wszystkich wierzchołków należących do zbioru *compsub*, niektórych należących do zbioru *candidates*, i z żadnego, który należy do zbioru *not*.
Poniżej przedstawiona została charakterystyka każdego ze zbiorów wykorzystywanych przez algorytm:

- *compsub*

  do zbioru należą wszystkie wierzchołki grafu, które tworzą powstającą klikę.

- *candidates*

  do zbioru należą wierzchołki grafu, które mogą posłużyć do rozszerzenia zbioru *compsub*.

- *not*

  do zbioru należą te wierzchołki, które były już wcześniej wykorzystane do rozszerzenia zbioru *compsub*.

Należy zauważyć, iż wszystkie wierzchołki, które są połączone z każdym wierzchołkiem należącym do zbioru *compsub* znajdują się albo w zbiorze *candidates* albo *not*.

Zmodyfikowana wersja algorytmu Brona–Kerboscha wprowadza pojęcie wierzchołka zwrotnego (dalej oznaczanego *pivot*), który wybierany jest ze zbioru $candidates \cup not$ jako wierzchołek o największym stopniu.

W każdym rekurencyjnym wywołaniu algorytmu rozważane są wierzchołki należące do zbioru *candidates*. Jeśli zbiory *candidates* i *not* są puste, sprawdzane jest czy znaleziona maksymalna klika (oparta na wierzchołkach ze zbioru *compsub*) jest większa od największej dotychczas znalezionej kliki. Jeśli tak, to znaleziona klika staje się największą, w przeciwnym wypadku największa klika pozostawiana jest bez zmian. W przypadku, gdy zbiory *candidates* i *not* nie są puste, dla każdego wierzchołka ze zbioru $candidates \setminus neighbors(pivot)$ następuje rekurencyjne wywołanie algorytmu, w którym bieżący wierzchołek v dodawany jest do zbioru *compsub* i usuwany ze zbioru *candidates*, a w zbiorach *candidates* i *not* pozostawiane są tylko te wierzchołki grafu, które są sąsiadami wierzchołka v. Następnie, wierzchołek v jest dodawany do zbioru *not* jako już wykorzystany do rozszerzenia kliki oraz usuwany ze zbioru *compsub*.
Wynikiem działania algorytmu jest zbiór *biggest_clique*, który początkowo inicjowany jest jako zbiór pusty. W przypadku, gdy znaleziona zostanie największa klika, zbiór ten zawiera wierzchołki ją tworzące.

3.3 Złożoność

Pesymistyczna złożoność przedstawionego algorytmu wynosi $O(3^{n/3})$ i wynika z górnego ograniczenia na liczbę maksymalnych klik w grafie o n wierzchołkach.

4 Struktury danych

Graf Do reprezentacji grafu zostanie wykorzystana macierz sąsiedztwa, zaimplementowana jako dwuwymiarowa tablica wartości boolowskich.

Zbiory wierzchołków (*compsub*, *candidates*, *not*, *biggest_clique*) Zbiory przechowujące wierzchołki zostaną zaimplementowane jako klasa $Vertices$ dziedzicząca po klasie $TreeSet$ języka Java.

5 Testy

Istotną częścią realizowanego zadania jest przeprowadzenie testów związanych zarówno z poprawnością zwracanych wyników jak również wpływem danych wejściowych na czas wykonania algorytmu.

5.1 Badanie poprawności zwracanych wyników

Do weryfikacji poprawności zwracanych przez algorytm wyników zostanie wykorzystana biblioteka igraph[1], która udostępnia m.in funkcję wyznaczającą maksymalne kliki w zadanym grafie. Podczas testowania planujemy wykorzystać dane zwrócone przez bibliotekę igraph jako rozwiązania referencyjne, które następnie posłużą do porównania z wynikami otrzymanymi przez zaimplementowany algorytm. Rozwiązanie, a więc największa klika zwrócona przez algorytm jest poprawna wtedy, gdy znajduje się na liście rozwiązań referencyjnych.
Proces generowania rozwiązań referencyjnych oraz porównywania wyników zostanie zautomatyzowany.

5.2 Badanie czasu wykonania dla różnych typów grafów

Z punktu widzenia analizy zaimplementowanego algorytmu istotne jest zbadanie jego zachowania dla różnych typów grafów. W szczególności przeprowadzone zostaną eksperymenty na zestawach grafów o zróżnicowanej gęstości.

6 Analiza zaimplementowanego algorytmu

W ramach części badawczej projektu zrealizowane zostały dwa rodzaje testów: testy wykazujące poprawność implementacji algorytmu oraz testy badające jego zachowanie dla różnych typów grafów.
W obu eksperymentach wykorzystano następujące zestawy grafów:

- zestaw grafów losowych o gęstości 0,1
- zestaw grafów losowych o gęstości 0,2
- zestaw grafów losowych o gęstości 0,3
- zestaw grafów losowych o gęstości 0,4

[1] `http://igraph.sourceforge.net/`

- zestaw grafów losowych o gęstości 0,5
- zestaw grafów losowych o gęstości 0,6
- zestaw grafów losowych o gęstości 0,7
- zestaw grafów losowych o gęstości 0,8
- zestaw grafów losowych o gęstości 0,9
- zestaw grafów pełnych

W każdym zestawie znalazły się grafy o liczbie wierzchołków: 10, 20, 30, 40, 50, 60, 70, 80, 90, 100.
Podczas analizy zbadane zostały dwa przypadki:

- wpływ liczby wierzchołków dla grafów o ustalonej gęstości,
- wpływ gęstości dla grafów o ustalonej liczbie wierzchołków.

6.1 Analiza poprawności działania algorytmu

Przed przystąpieniem do właściwej analizy działania algorytmu została zweryfikowana poprawność jego implementacji. Do testowania poprawności działania programu wykorzystana została biblioteka igraph. Szczegółowy schemat testowania został przedstawiony w rozdziale 5.1. Testy zostały przeprowadzone na zestawie grafów opisanych na początku niniejszego rozdziału i nie wykazały błędów ani problemów z działaniem algorytmu.

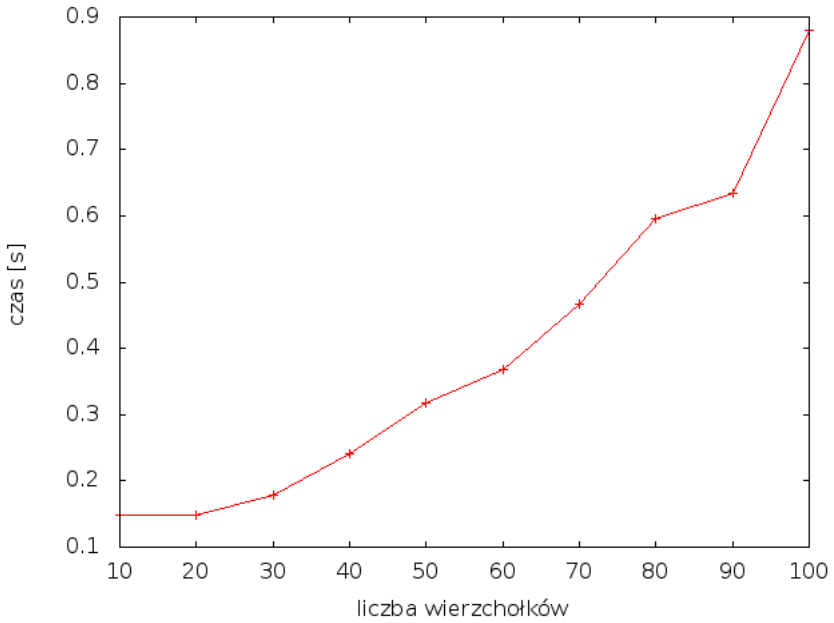
6.2 Analiza czasów wykonania algorytmu

Dla każdego zestawu grafów dokonane zostały pomiary czasu działania algorytmu. Wyniki zostały zamieszczone poniżej.

Zestaw grafów o gęstości 0,1

Tabela 1: Statystyki czasowe dla zestawu grafów o gęstości 0,1

Liczba wierzchołków	Czas [s]
10	0,148
20	0,149
30	0,179
40	0,242
50	0,318
60	0,369
70	0,467
80	0,595
90	0,633
100	0,880

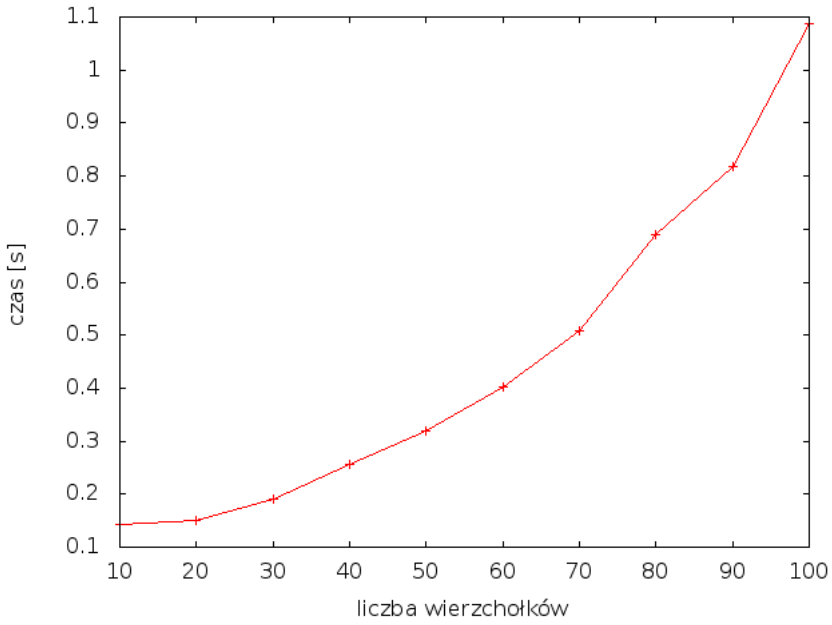


Rysunek 1: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,1

Zestaw grafów o gęstości 0,2

Tabela 2: Statystyki czasowe dla zestawu grafów o gęstości 0,2

Liczba wierzchołków	Czas [s]
10	0,143
20	0,150
30	0,191
40	0,255
50	0,319
60	0,402
70	0,509
80	0,689
90	0,817
100	1,088

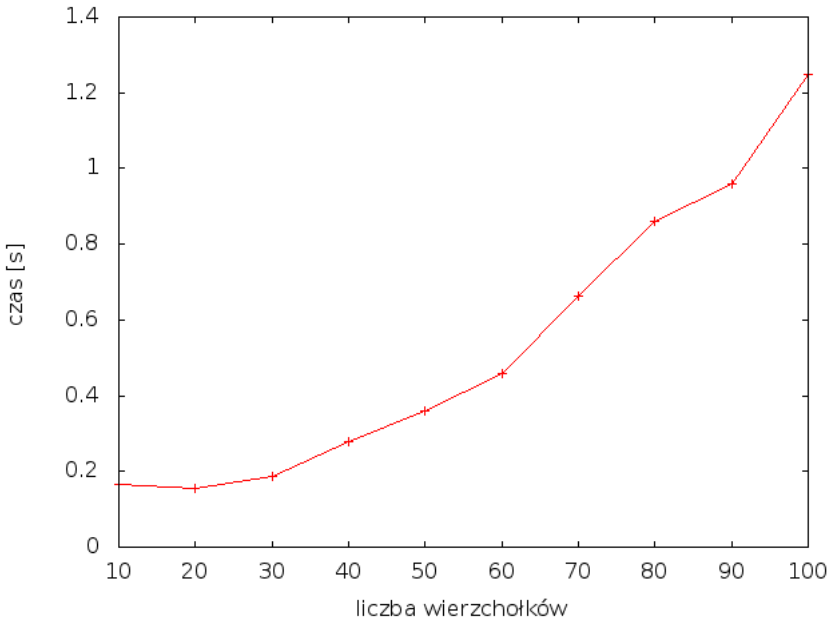


Rysunek 2: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,2

Zestaw grafów o gęstości 0,3

Tabela 3: Statystyki czasowe dla zestawu grafów o gęstości 0,3

Liczba wierzchołków	Czas [s]
10	0,164
20	0,155
30	0,187
40	0,280
50	0,361
60	0,457
70	0,664
80	0,859
90	0,958
100	1,248

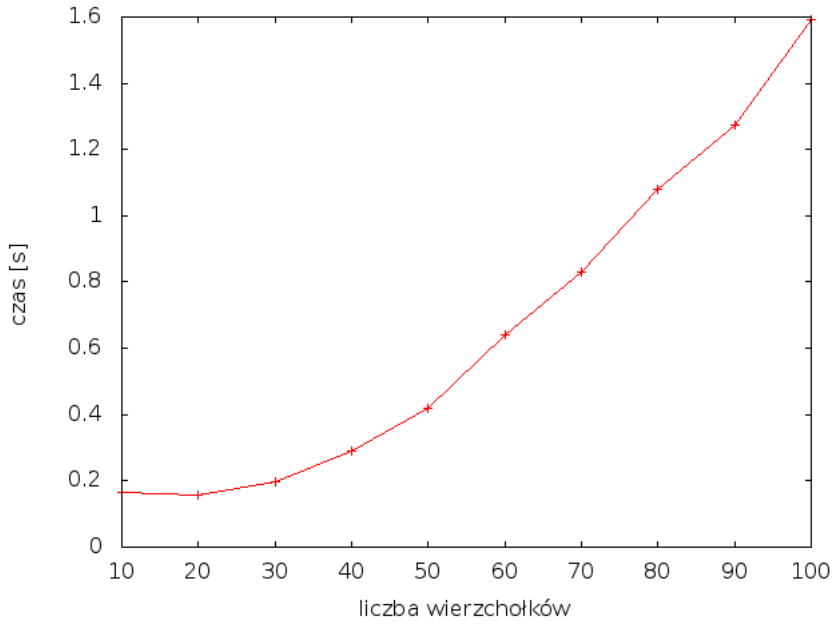


Rysunek 3: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,3

Zestaw grafów o gęstości 0,4

Tabela 4: Statystyki czasowe dla zestawu grafów o gęstości 0,4

Liczba wierzchołków	Czas [s]
10	0,164
20	0,156
30	0,197
40	0,291
50	0,420
60	0,642
70	0,830
80	1,079
90	1,272
100	1,591

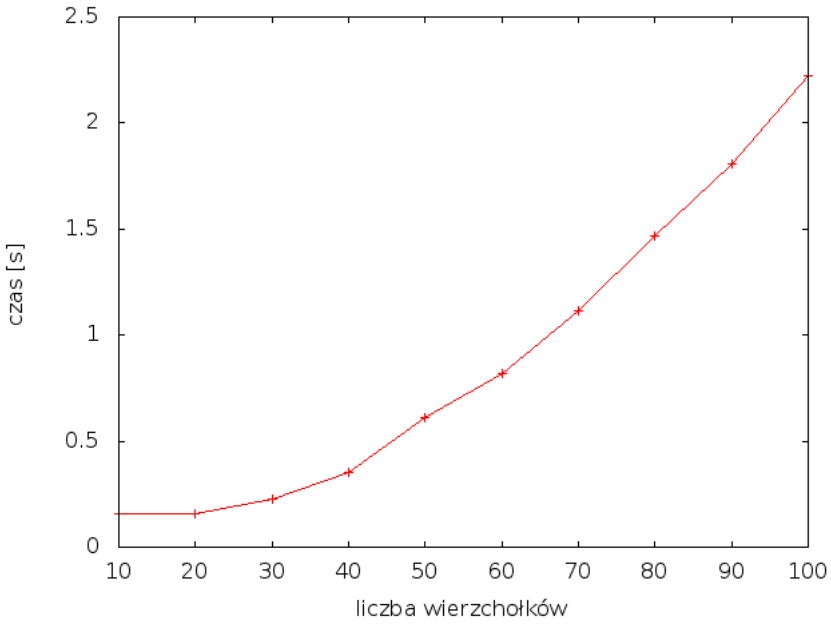


Rysunek 4: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,4

Zestaw grafów o gęstości 0,5

Tabela 5: Statystyki czasowe dla zestawu grafów o gęstości 0,5

Liczba wierzchołków	Czas [s]
10	0,159
20	0,157
30	0,227
40	0,353
50	0,610
60	0,819
70	1,114
80	1,465
90	1,809
100	2,222

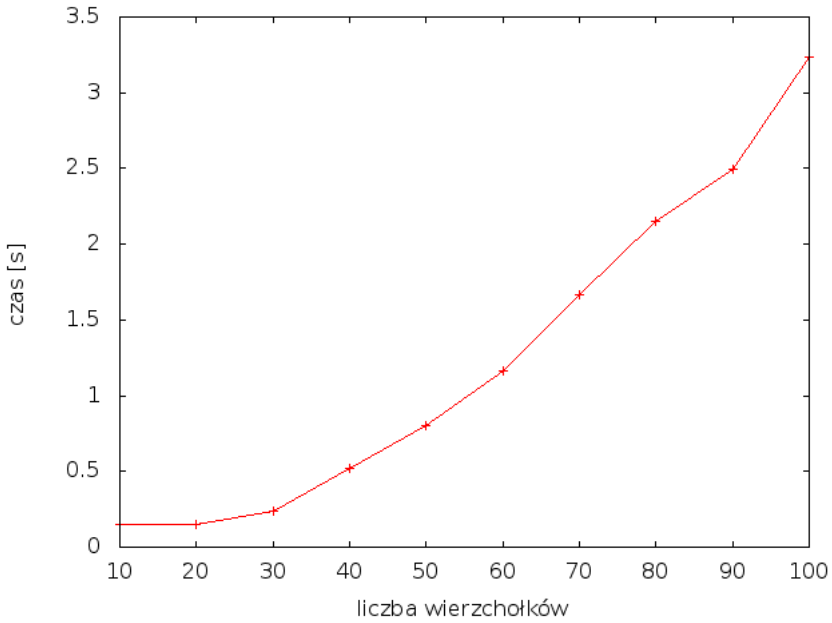


Rysunek 5: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,5

Zestaw grafów o gęstości 0,6

Tabela 6: Statystyki czasowe dla zestawu grafów o gęstości 0,6

Liczba wierzchołków	Czas [s]
10	0,154
20	0,153
30	0,239
40	0,522
50	0,806
60	1,168
70	1,667
80	2,154
90	2,496
100	3,232

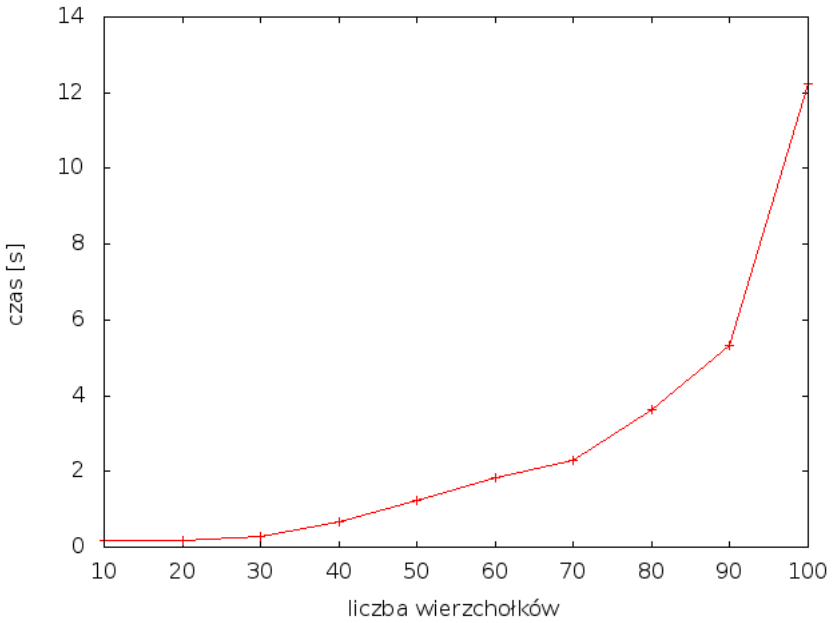


Rysunek 6: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,6

Zestaw grafów o gęstości 0,7

Tabela 7: Statystyki czasowe dla zestawu grafów o gęstości 0,7

Liczba wierzchołków	Czas [s]
10	0,167
20	0,166
30	0,290
40	0,657
50	1,249
60	1,850
70	2,304
80	3,645
90	5,338
100	12,227

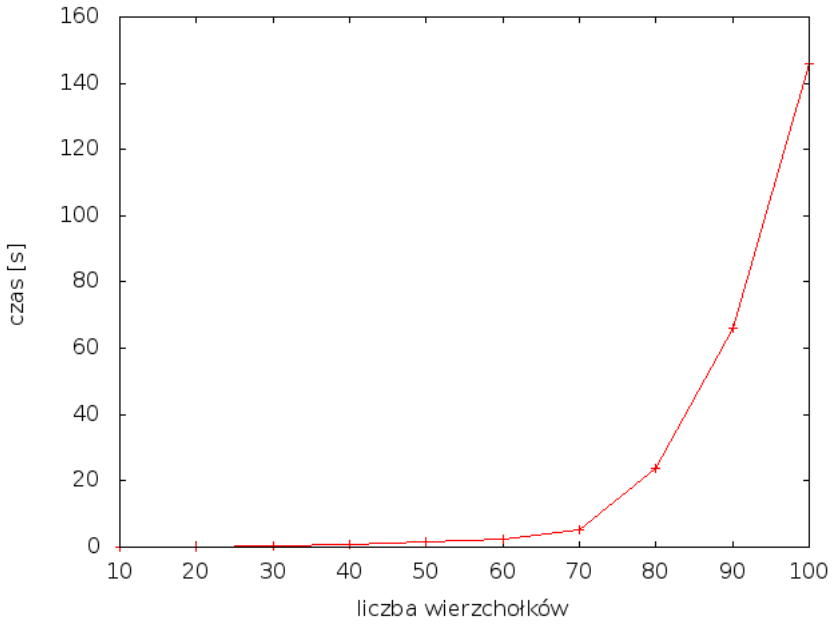


Rysunek 7: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,7

Zestaw grafów o gęstości 0,8

Tabela 8: Statystyki czasowe dla zestawu grafów o gęstości 0,8

Liczba wierzchołków	Czas [s]
10	0,125
20	0,171
30	0,440
40	0,995
50	1,780
60	2,581
70	5,430
80	23,747
90	66,287
100	145,761

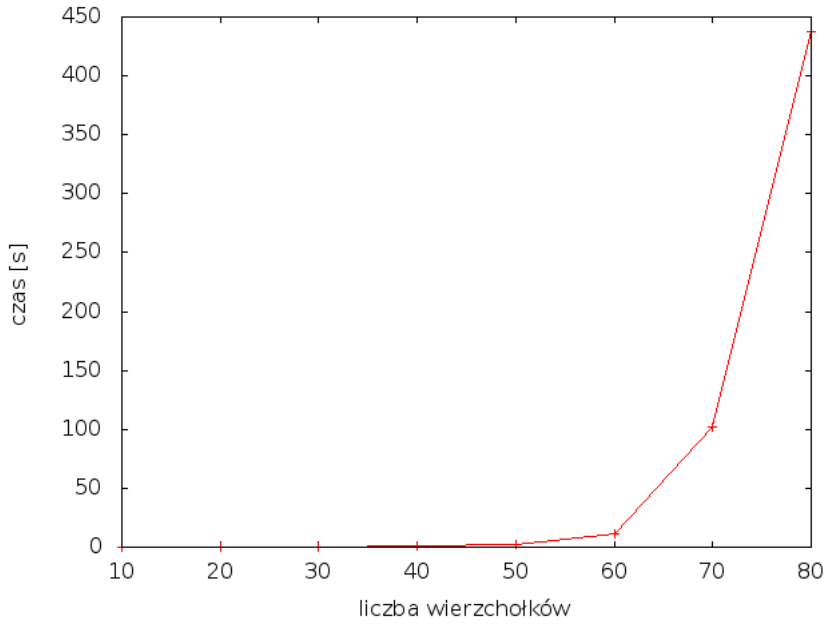


Rysunek 8: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,8

Zestaw grafów o gęstości 0,9

Tabela 9: Statystyki czasowe dla zestawu grafów o gęstości 0,9

Liczba wierzchołków	Czas [s]
10	0,129
20	0,167
30	0,433
40	1,265
50	2,287
60	11,706
70	95,545
80	580,295
90	3672,374
100	10056,890

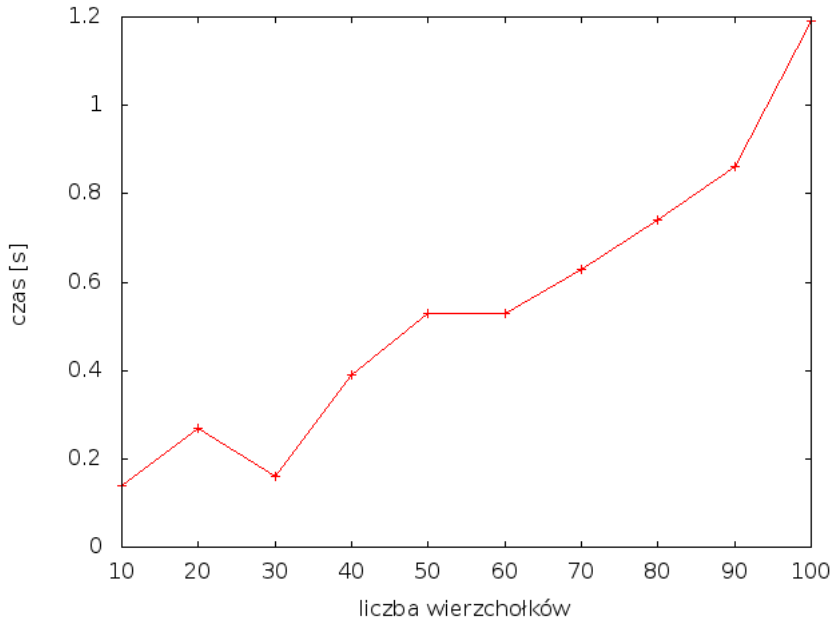


Rysunek 9: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,9

Zestaw grafów pełnych

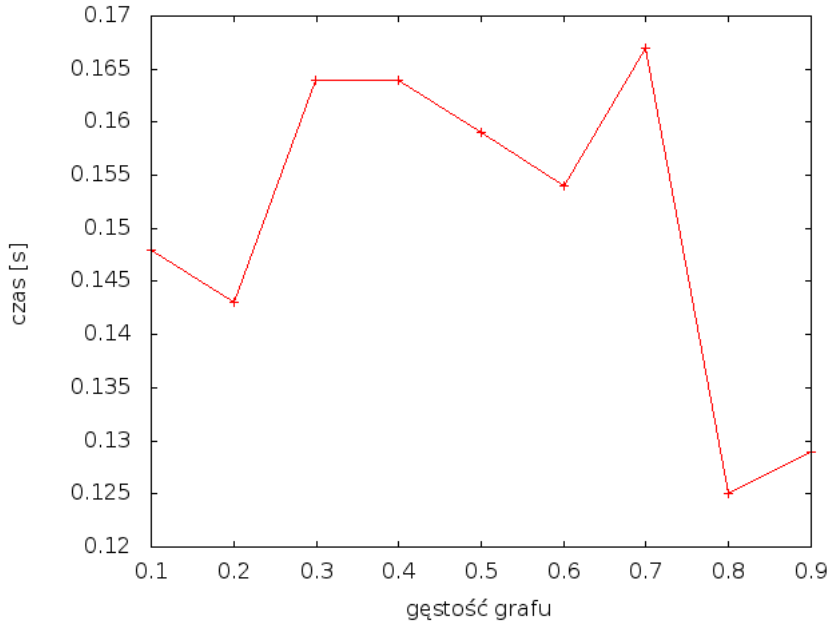
Tabela 10: Statystyki czasowe dla zestawu grafów pełnych

Liczba wierzchołków	Czas [s]
10	0,14
20	0,27
30	0,16
40	0,39
50	0,53
60	0,53
70	0,63
80	0,74
90	0,86
100	1,19



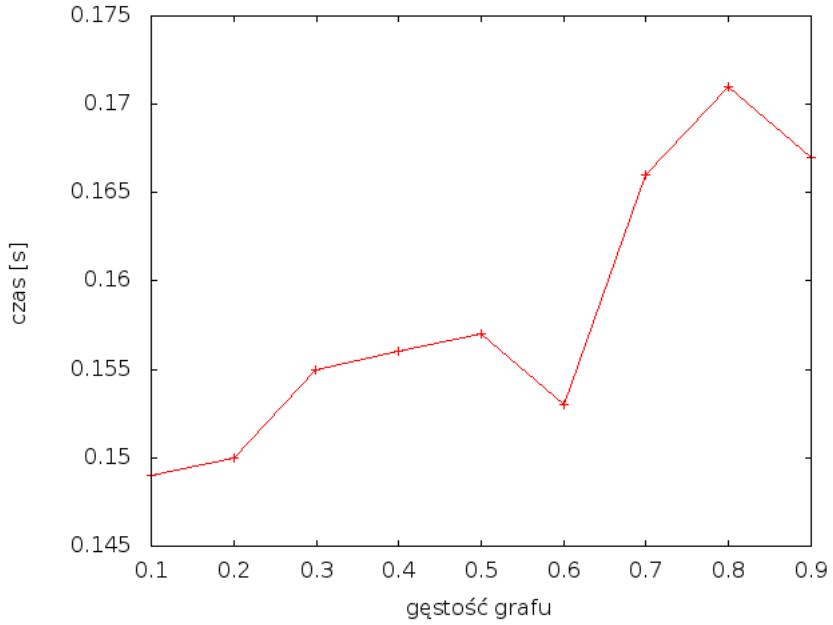
Rysunek 10: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów pełnych

Zestaw grafów o 10 wierzchołkach



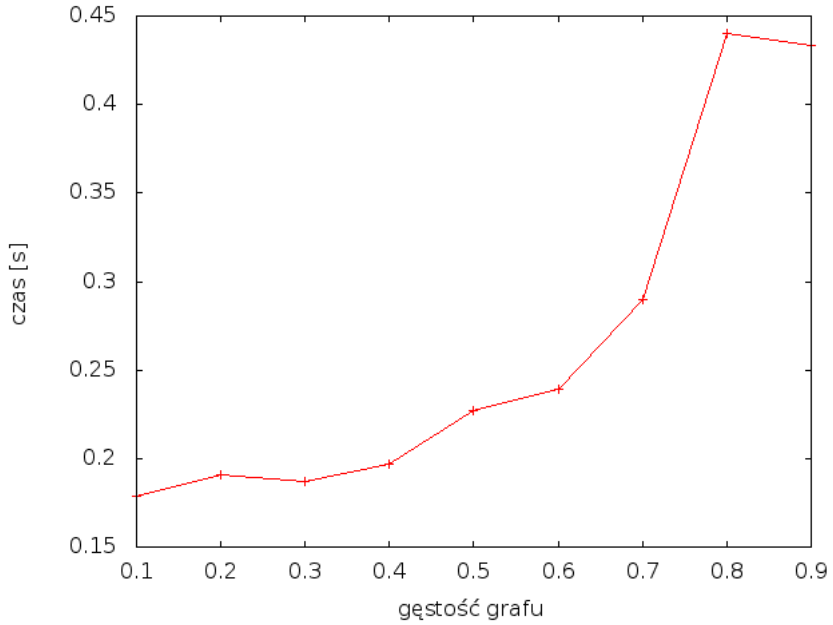
Rysunek 11: Wykres zależności czasu wykonania od gęstości dla grafów o 10 wierzchołkach

Zestaw grafów o 20 wierzchołkach



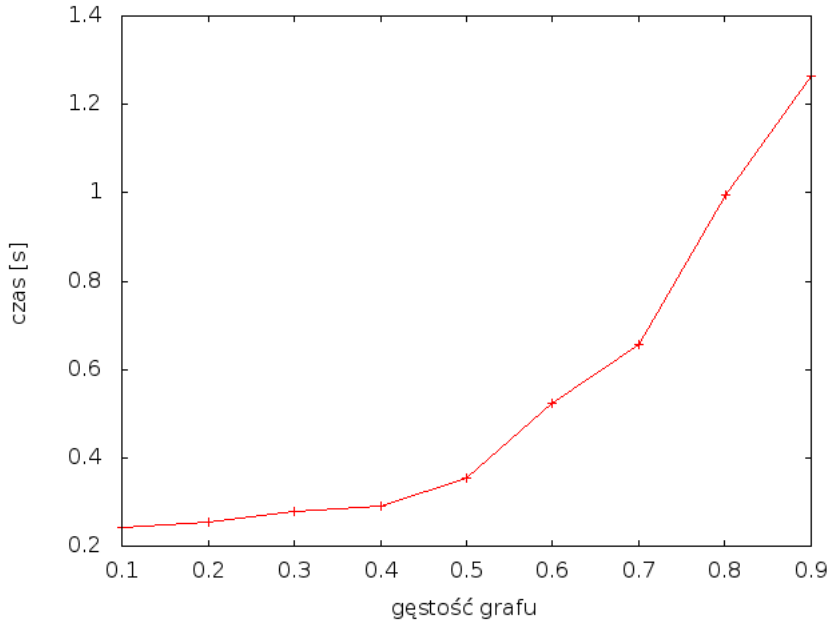
Rysunek 12: Wykres zależności czasu wykonania od gęstości dla grafów o 20 wierzchołkach

Zestaw grafów o 30 wierzchołkach



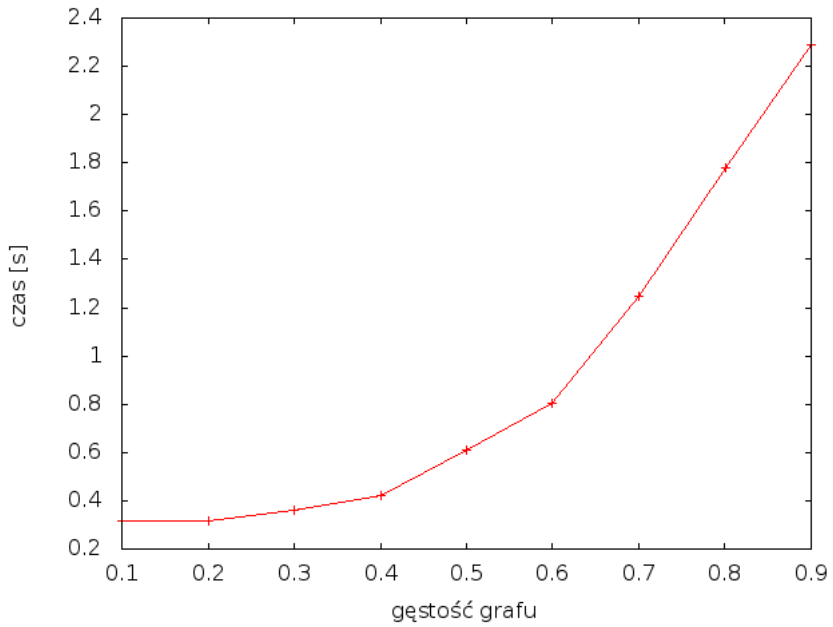
Rysunek 13: Wykres zależności czasu wykonania od gęstości dla grafów o 30 wierzchołkach

Zestaw grafów o 40 wierzchołkach



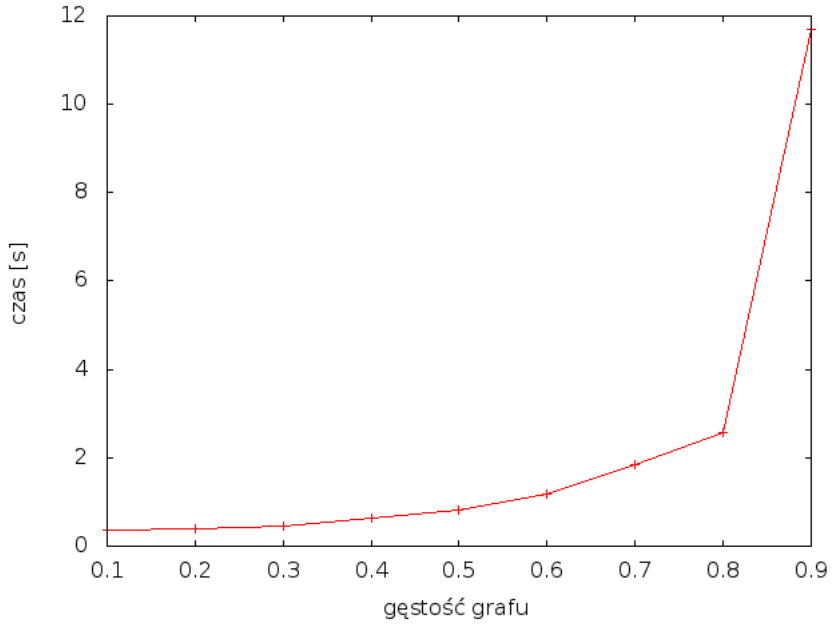
Rysunek 14: Wykres zależności czasu wykonania od gęstości dla grafów o 40 wierzchołkach

Zestaw grafów o 50 wierzchołkach



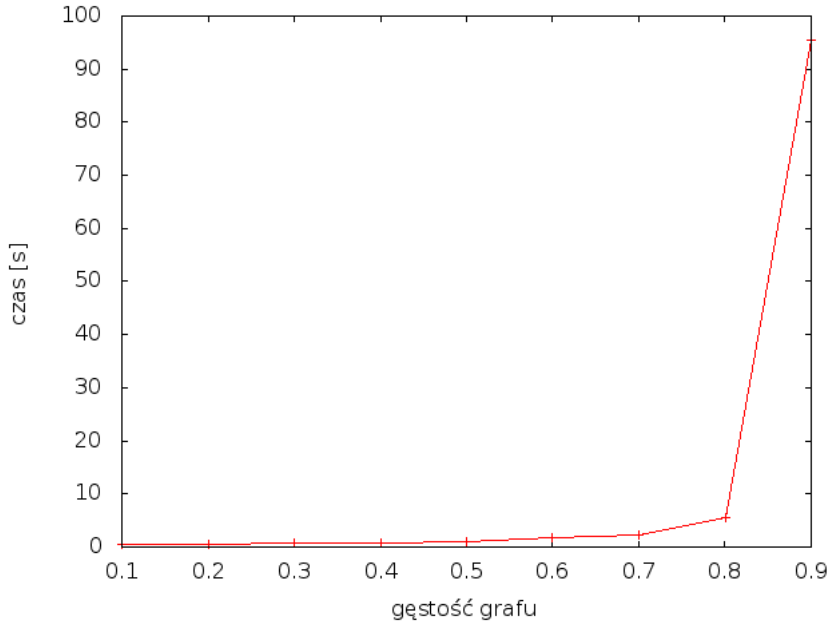
Rysunek 15: Wykres zależności czasu wykonania od gęstości dla grafów o 50 wierzchołkach

Zestaw grafów o 60 wierzchołkach



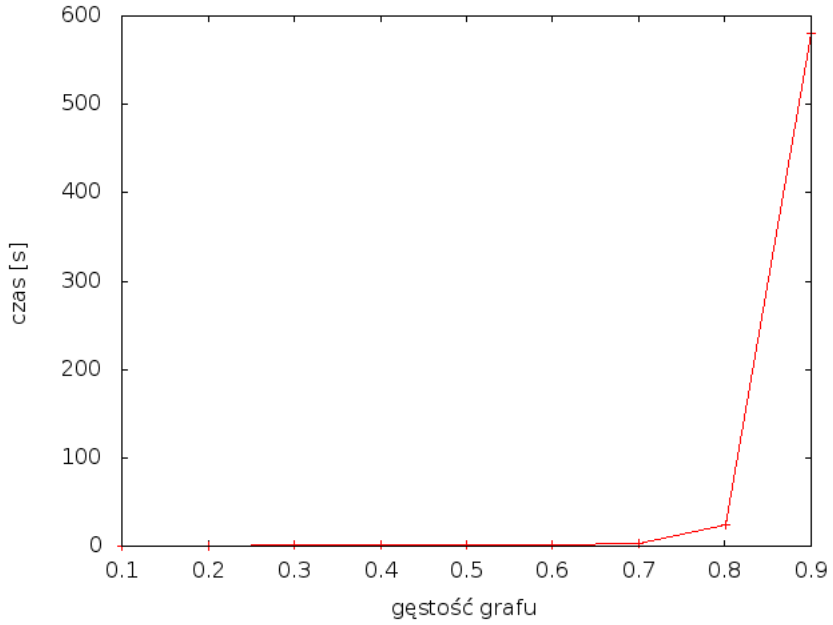
Rysunek 16: Wykres zależności czasu wykonania od gęstości dla grafów o 60 wierzchołkach

Zestaw grafów o 70 wierzchołkach



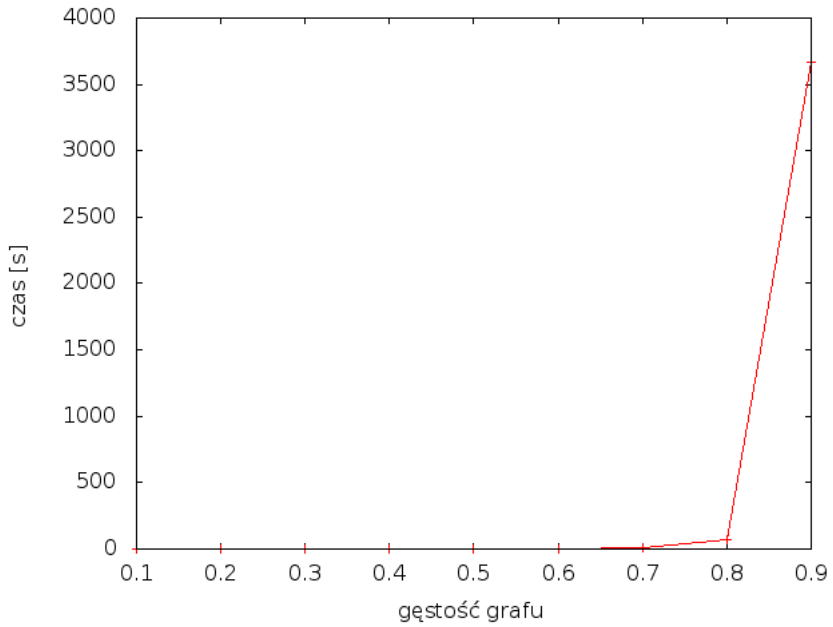
Rysunek 17: Wykres zależności czasu wykonania od gęstości dla grafów o 70 wierzchołkach

Zestaw grafów o 80 wierzchołkach



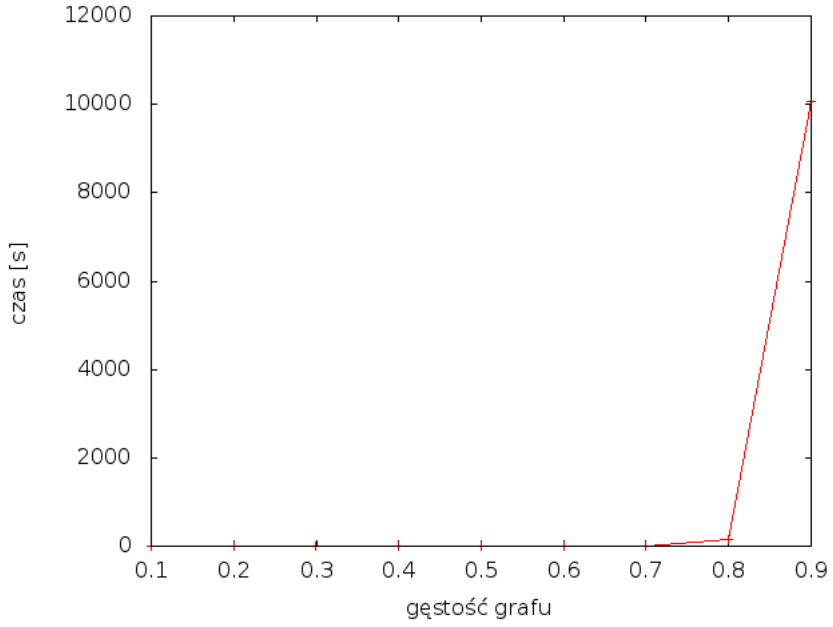
Rysunek 18: Wykres zależności czasu wykonania od gęstości dla grafów o 80 wierzchołkach

Zestaw grafów o 90 wierzchołkach



Rysunek 19: Wykres zależności czasu wykonania od gęstości dla grafów o 90 wierzchołkach

Zestaw grafów o 100 wierzchołkach



Rysunek 20: Wykres zależności czasu wykonania od gęstości dla grafów o 100 wierzchołkach

7 Podsumowanie i wnioski

W ramach projektu zrealizowana została aplikacja umożliwiająca znalezienie największej kliki w zadanym grafie. Zaimplementowany algorytm został przetestowany pod kątem poprawnego działania jak również wykonane zostały badania mające na celu zbadanie jego zachowania dla grafów o zróżnicowanej gęstości.
W wyniku eksperymentu przeprowadzonego na zestawie grafów losowych obejmującego grafy o liczbie wierzchołków 10, 20, 30, 40, 50, 60, 70, 80, 100 oraz gęstościach z przedziału 0,1 – 1 uzyskano rezultaty czasowe zaprezentowane w rozdziale 6.2.
Analizując otrzymane wyniki można zauważyć, iż dla grafów o gęstościach z przedziału 0,1 – 0,6 czas wykonania wzrasta w sposób zbliżony do liniowego wraz ze wzrostem liczby wierzchołków grafu. Odmienna sytuacja ma miejsce dla grafów o gęstości z przedziału 0,7 – 0,9, dla których czas wykonania algorytmu rośnie wykładniczo wraz ze wzrostem liczby wierzchołków. Prawidłowość ta jest szczególnie widoczna dla grafów o gęstości 0,9.
Osobną klasę grafów stanowią grafy pełne, dla których czasy wykonania zależą, podobnie jak w przypadku grafów rzadkich, w sposób zbliżony do liniowego od liczby wierzchołków.
Na podstawie analizy wyników można stwierdzić, że zaimplementowany algorytm silnie zależy zarówno od rozmiaru grafu jak i jego gęstości. W przypadku grafów rzadkich oraz pełnych czasy uzyskiwane przez algorytm nie przekroczyły kilku sekund dla grafów o 100 wierzchołkach. Niestety, w przypadku jednocześnie dużej gęstości i liczby wierzchołków grafu ujawnia się wykładnicza złożoność algorytmu i czas działania drastycznie wzrasta (czego przykładem jest wynik dla grafu o 100 wierzchołkach i gęstości 0,9 sięgający prawie 3 godzin).

A Zawartość katalogów

Opis zawartości poszczególnych katalogów:

- `gis-kliki-java` - implementacja algorytmu będącego przedmiotem projektu,
- `gis-c` - pomocniczy program generujący rozwiązania referencyjne z wykorzystaniem biblioteki igraph,
- `tests` - pliki dla testów weryfikujących poprawność implementacji:
  - `correctness` - pliki z grafami wykorzystanymi do weryfikacji poprawności algorytmu,
  - `performance` - pliki z grafami wykorzystanymi podczas analizy czasów wykonania algorytmu i jego poprawności,
- `doc` - dokumentacja projektu.

B Instrukcja obsługi

Aplikacja działa w trybie konsolowym. W celu jej uruchomienia należy wykonać następujące kroki:

1. Skompilować kod Javy. W tym celu wystarczy uruchomić skrypt `gis-kliki-java/build.sh`.
2. Przygotować pliki wejściowe.
3. Uruchomić program jednym z możliwych sposobów:
   - Ręczne uruchomienie. W katalogu `gis-kliki-java` należy wykonać następujące polecenie:
     `java -cp bin main/Main < plik_wejściowy`
   - Uruchomienie z wykorzystaniem skryptu. Wraz z programem dostarczone są dwa skrypty ułatwiające uruchamianie i testowanie aplikacji:
     - `run_basic.sh`
       W wyniku działania zwracany jest czas wykonania oraz znaleziona największa klika.
       Przykład wykorzystania: `run_test.sh graf1_in.txt graf2_in.txt` (na wejście można przekazać wiele plików)
     - `run_test.sh`
       Skrypt umożliwia działanie aplikacji w dwóch trybach: standardowym (jak w przypadku `run_basic.sh`) oraz rozszerzonym, weryfikującym znalezione rozwiązanie z rozwiązaniami referencyjnymi. Jeśli plik wejściowy nazywał się np. `123_in.txt`, to skrypt poszukuje rozwiązania w pliku o nazwie `123_out.txt`. W przypadku nie znalezienia pliku z rozwiązaniem aplikacja zostaje uruchomiona w trybie standardowym.
       Przykład wykorzystania: `run_test.sh graf1_in.txt graf2_in.txt` (na wejście można przekazać wiele plików)

Referencje

[1] Bron Coen and Kerbosch Joep. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.