

# Poszukiwanie największej kliki w grafie

Anna Stępień  
Adam Stelmaszczyk

## Spis treści

<b>1</b>	<b>Zadanie</b>	<b>2</b>
<b>2</b>	<b>Założenia</b>	<b>2</b>
2.1	Dane wejściowe . . . . .	2
2.2	Dane wyjściowe . . . . .	2
2.3	Sytuacje wyjątkowe . . . . .	3
<b>3</b>	<b>Algorytm</b>	<b>3</b>
3.1	Pseudokod . . . . .	3
3.2	Opis działania . . . . .	5
3.3	Złożoność . . . . .	5
<b>4</b>	<b>Struktury danych</b>	<b>6</b>
<b>5</b>	<b>Testy</b>	<b>6</b>
5.1	Badanie poprawności zwracanych wyników . . . . .	6
5.2	Badanie czasu wykonania dla różnych typów grafów . . . . .	6
<b>6</b>	<b>Analiza zaimplementowanego algorytmu</b>	<b>6</b>
6.1	Analiza poprawności działania algorytmu . . . . .	7
6.2	Analiza czasów wykonania algorytmu . . . . .	7
<b>7</b>	<b>Podsumowanie i wnioski</b>	<b>23</b>
	<b>Dodatek A Zawartość katalogów</b>	<b>23</b>
	<b>Dodatek B Instrukcja obsługi</b>	<b>23</b>

# 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 największą 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:

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

$q_{i,j}$  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
```

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.

### 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. Próba 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 nieorientowanym. Domyślnie algorytm zwraca wszystkie maksymalne klik. 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:  $compsub \leftarrow \emptyset$ 
2:  $candidates \leftarrow V(G)$ 
3:  $not \leftarrow \emptyset$ 
4:  $cliques \leftarrow \emptyset$ 
5: function BRON_KERBOSCH( $compsub, candidates, not$ )
6:   if  $candidates = \emptyset$  and  $not = \emptyset$  then
7:      $cliques \leftarrow cliques \cup \{compsub\}$  ▷ Maksymalna klika
8:   else
9:     for each  $v$  in  $candidates$  do
10:       $candidates \leftarrow candidates \setminus \{v\}$ 
11:       $new\_compsub \leftarrow compsub \cup \{v\}$ 
12:       $new\_candidates \leftarrow candidates \cap neighbors(v)$ 
13:       $new\_not \leftarrow not \cap neighbors(v)$ 
14:      BRON_KERBOSCH( $new\_compsub, new\_candidates, new\_not$ )
15:       $compsub \leftarrow compsub \cup \{v\}$ 
16:    end for
17:  end if
18: end function
```

---

---

## 2 Algorytm Brona–Kerboscha (wersja rozszerzona)

---

```
1:  $compsub \leftarrow \emptyset$ 
2:  $candidates \leftarrow V(G)$ 
3:  $not \leftarrow \emptyset$ 
4:  $biggest\_clique \leftarrow \emptyset$ 
5: function BRON_KERBOSCH( $candidates, not$ )
6:   if  $candidates = \emptyset$  and  $not = \emptyset$  then
7:     if  $size(biggest\_clique) < size(compsub)$  then
8:        $biggest\_clique \leftarrow compsub$  ▷ Największa klika
9:     end if
10:  else
11:     $pivot \leftarrow vertex\_with\_maxdeg(candidates \cup not)$ 
12:     $candidates\_to\_check \leftarrow candidates \setminus neighbors(pivot)$ 
13:    for each  $v$  in  $candidates\_to\_check$  do
14:       $compsub \leftarrow compsub \cup \{v\}$ 
15:       $candidates \leftarrow candidates \setminus \{v\}$ 
16:       $new\_candidates \leftarrow candidates \cap neighbors(v)$ 
17:       $new\_not \leftarrow not \cap neighbors(v)$ 
18:      BRON_KERBOSCH( $new\_candidates, new\_not$ )
19:       $compsub \leftarrow compsub \setminus \{v\}$ 
20:       $not \leftarrow not \cup \{v\}$ 
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*<sup>1</sup>, 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

---

<sup>1</sup><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.

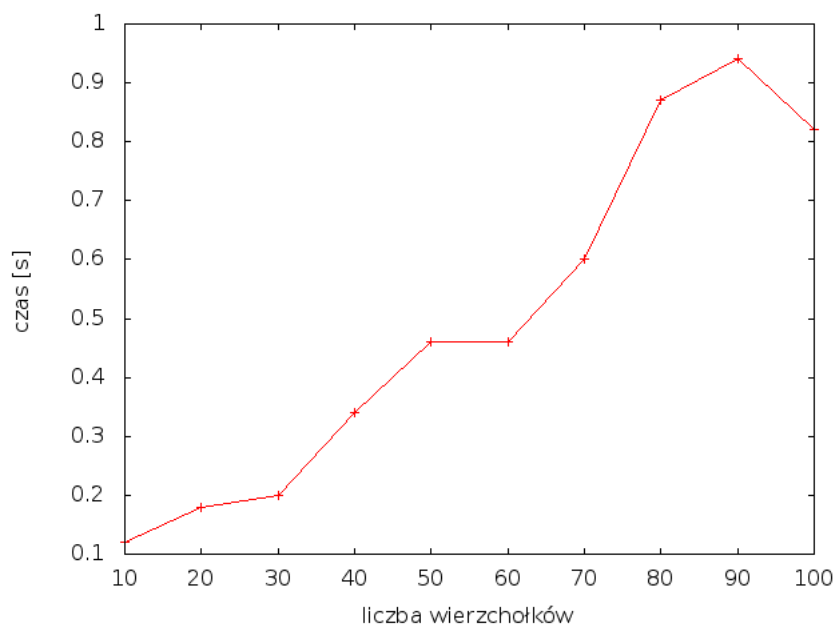
## 6.2 Analiza czasów wykonania algorytmu

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	Liczba największych klik	Czas [s]
10	5	0.12
20	2	0.18
30	2	0.20
40	8	0.34
50	16	0.46
60	37	0.46
70	53	0.60
80	1	0.87
90	3	0.94
100	3	0.82



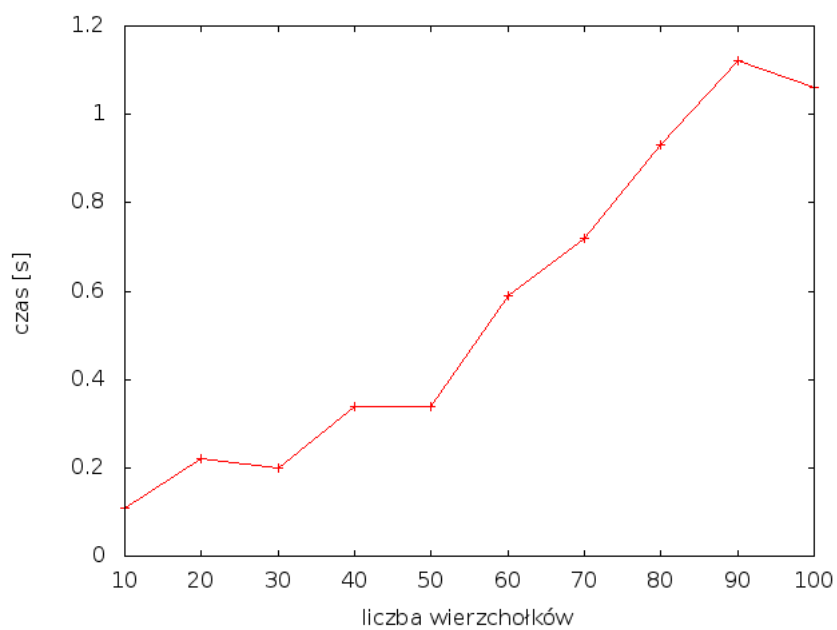
Rysunek 1: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,1



## Zestaw grafów o gęstości 0,2

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	9	0.11
20	1	0.22
30	5	0.20
40	5	0.34
50	10	0.34
60	22	0.59
70	1	0.72
80	1	0.93
90	4	1.12
100	10	1.06

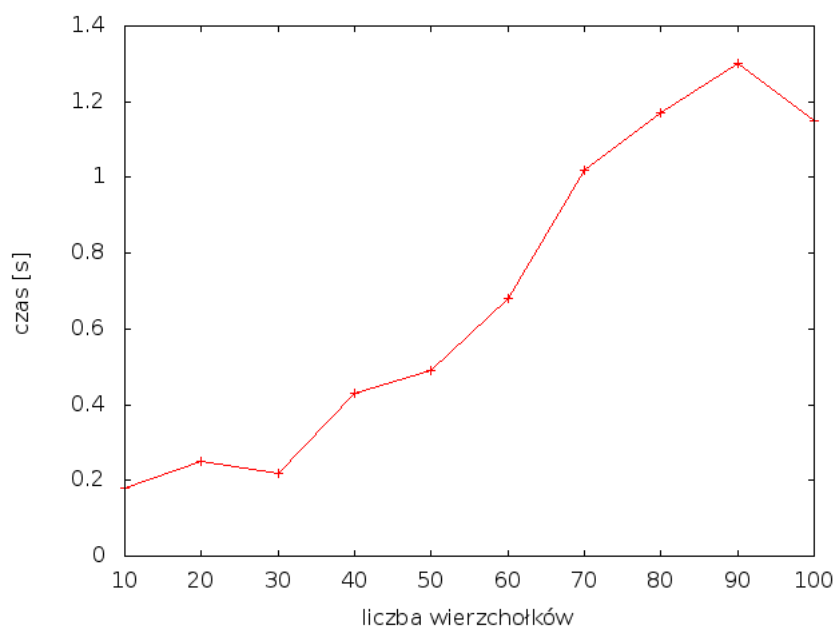


Rysunek 2: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,2

## Zestaw grafów o gęstości 0,3

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	3	0.18
20	2	0.25
30	20	0.22
40	1	0.43
50	1	0.49
60	29	0.68
70	1	1.02
80	2	1.17
90	8	1.30
100	9	1.15

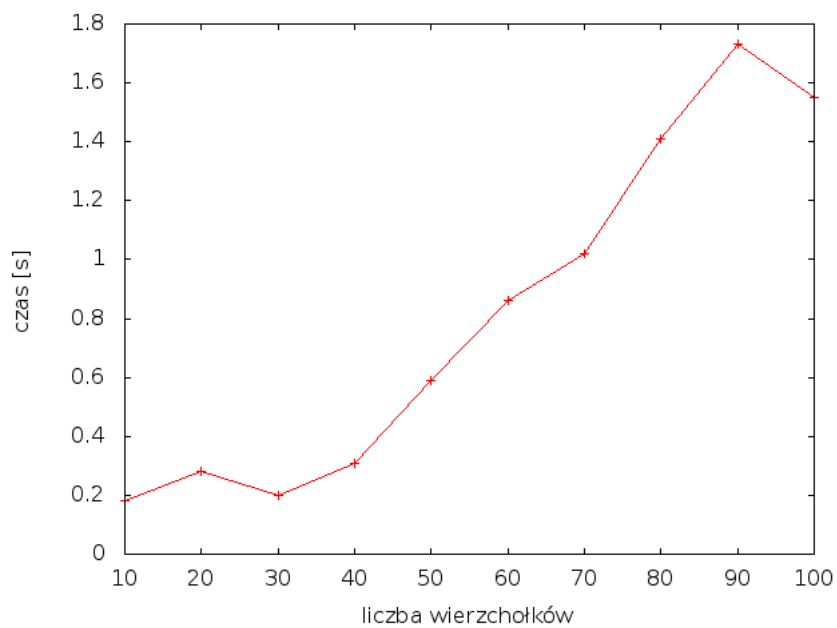


Rysunek 3: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,3

## Zestaw grafów o gęstości 0,4

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	7	0.18
20	10	0.28
30	6	0.20
40	6	0.31
50	10	0.59
60	32	0.86
70	5	1.02
80	5	1.41
90	9	1.73
100	49	1.55

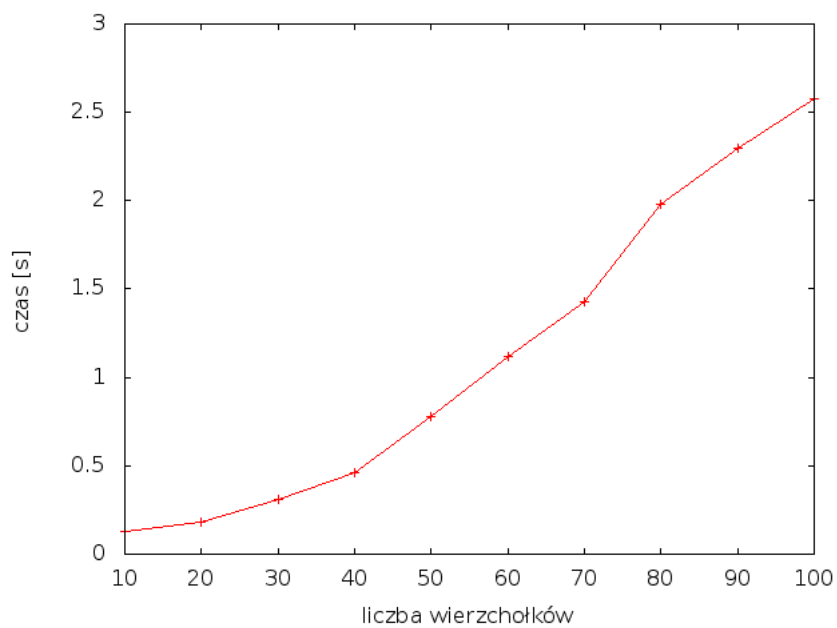


Rysunek 4: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,4

## Zestaw grafów o gęstości 0,5

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	12	0.13
20	3	0.18
30	11	0.31
40	2	0.46
50	3	0.78
60	4	1.12
70	7	1.43
80	7	1.98
90	2	2.30
100	17	2.58

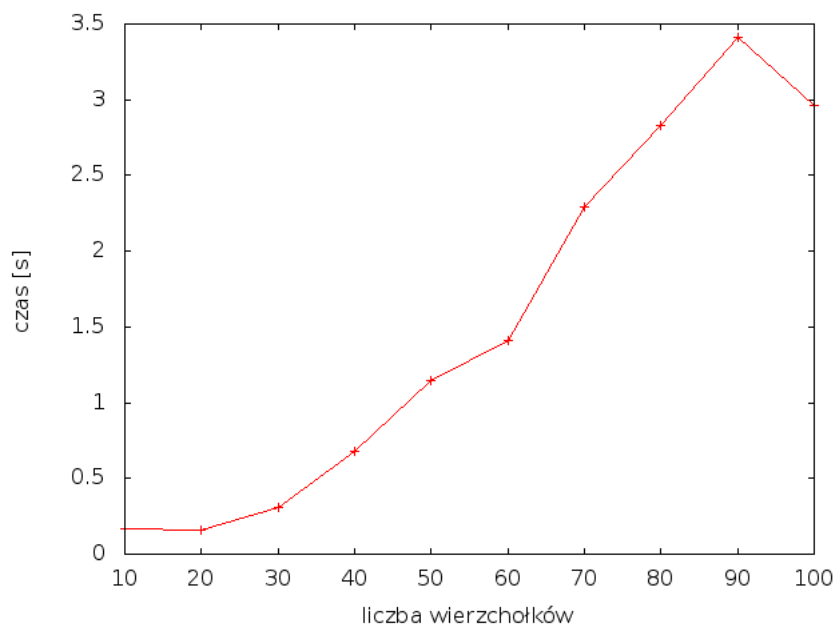


Rysunek 5: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,5

## Zestaw grafów o gęstości 0,6

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	1	0.17
20	3	0.16
30	15	0.31
40	1	0.68
50	1	1.15
60	34	1.41
70	5	2.29
80	1	2.83
90	3	3.41
100	48	2.96

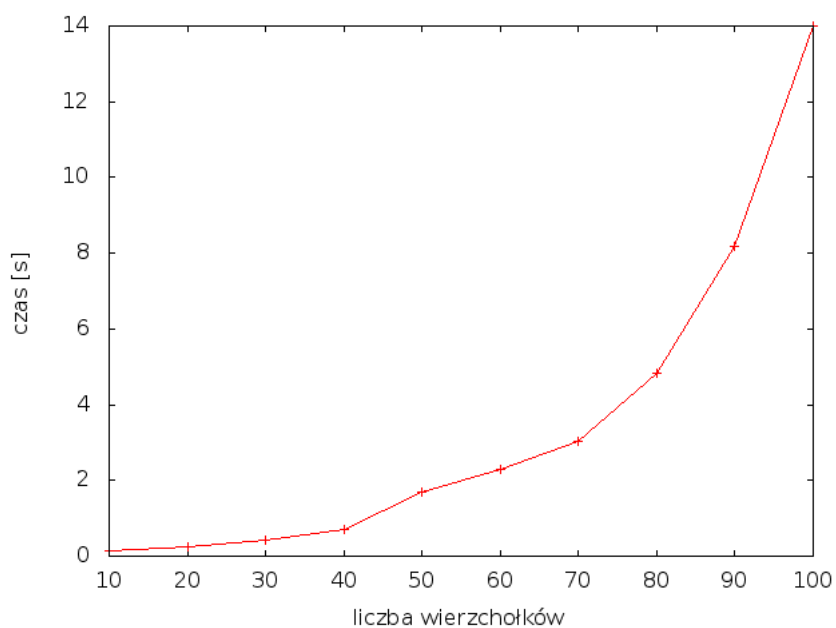


Rysunek 6: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,6

## Zestaw grafów o gęstości 0,7

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	4	0.13
20	2	0.24
30	23	0.42
40	2	0.72
50	16	1.68
60	24	2.29
70	1	3.04
80	92	4.83
90	39	8.17
100	52	13.99

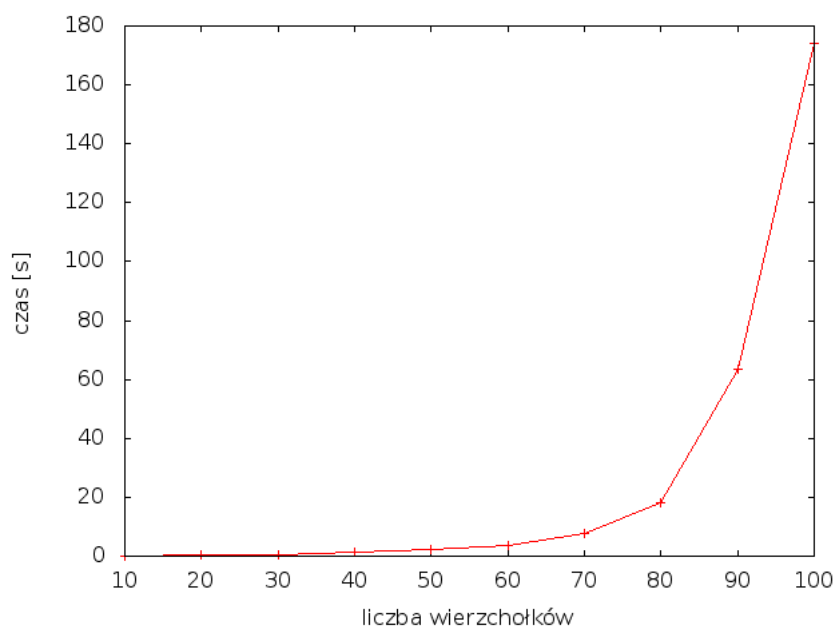


Rysunek 7: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,7

## Zestaw grafów o gęstości 0,8

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	3	0.13
20	38	0.28
30	15	0.51
40	2	1.25
50	47	2.45
60	17	3.41
70	5	7.49
80	9	18.25
90	58	63.27
100	29	174.22

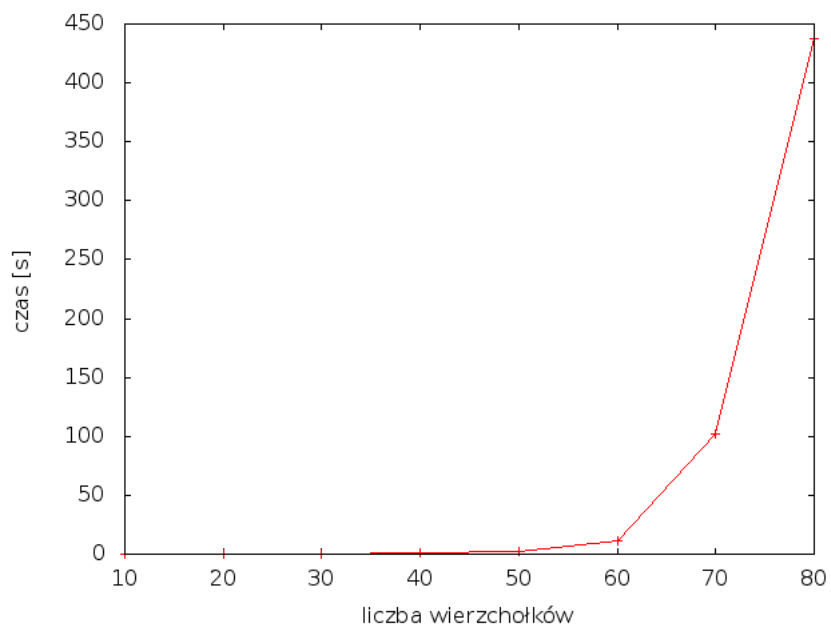


Rysunek 8: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,8

## Zestaw grafów o gęstości 0,9

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

Liczba wierzchołków	Liczba największych klik	Czas [s]
10	2	0.14
20	2	0.22
30	2	0.56
40	5	1.47
50	30	2.46
60	67	11.49
70	127	102.40
80	106	437.49
90	204	3486.84
100	111	9667.70



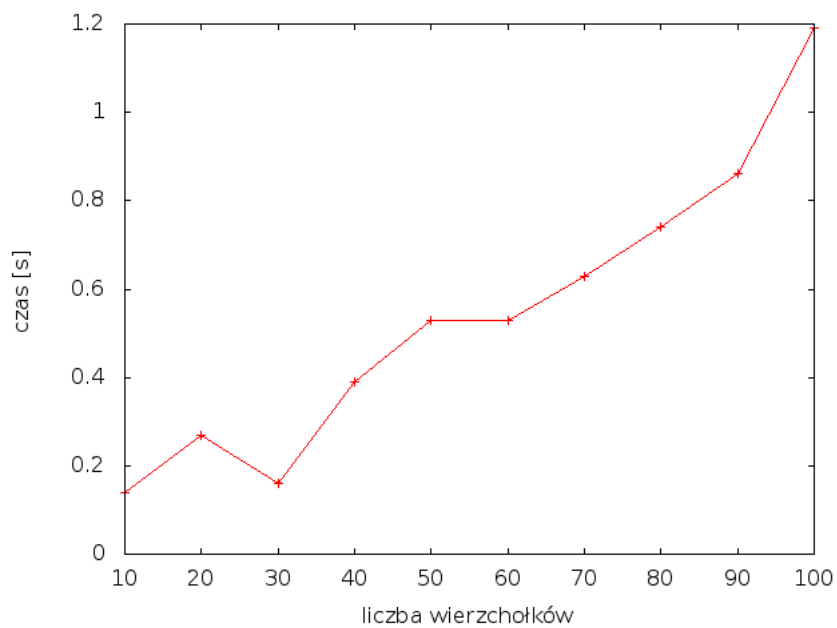
Rysunek 9: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów o gęstości 0,9



## Zestaw grafów pełnych

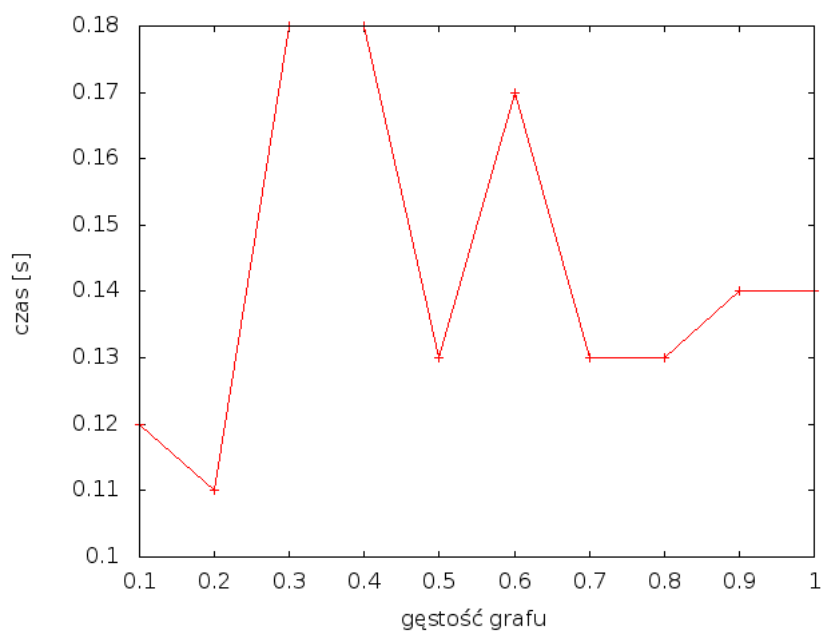
Tabela 10: Statystyki czasowe dla zestawu grafów pełnych

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



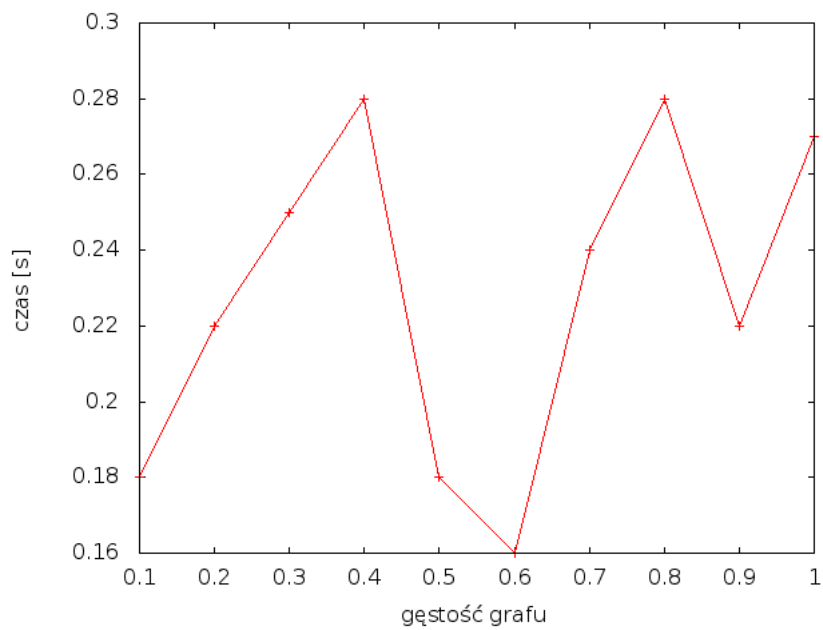
Rysunek 10: Wykres zależności czasu wykonania od liczby wierzchołków dla grafów pełnych

### Zestaw grafów o 10 wierzchołkach



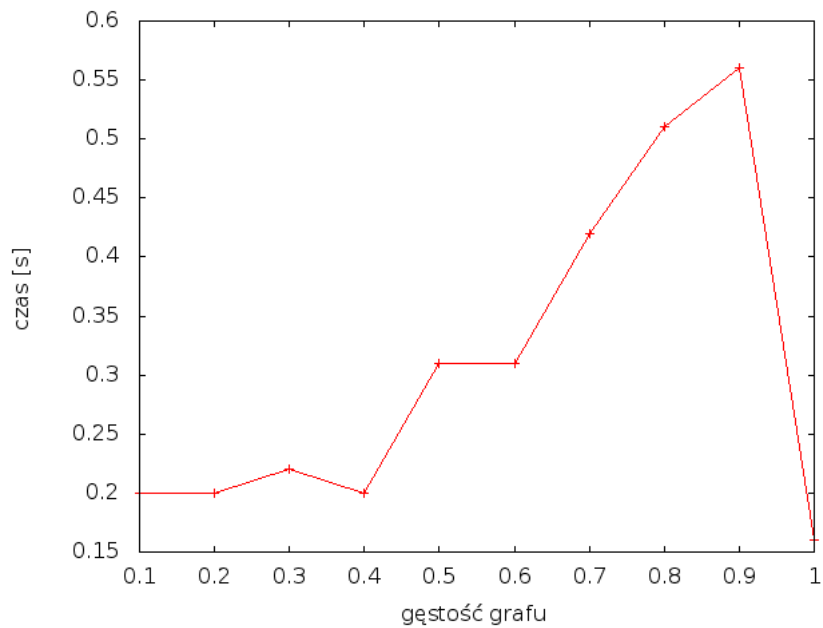
Rysunek 11: Wykres zależności czasu wykonania od gęstości dla grafów o 10 wierzchołkach

### Zestaw grafów o 20 wierzchołkach



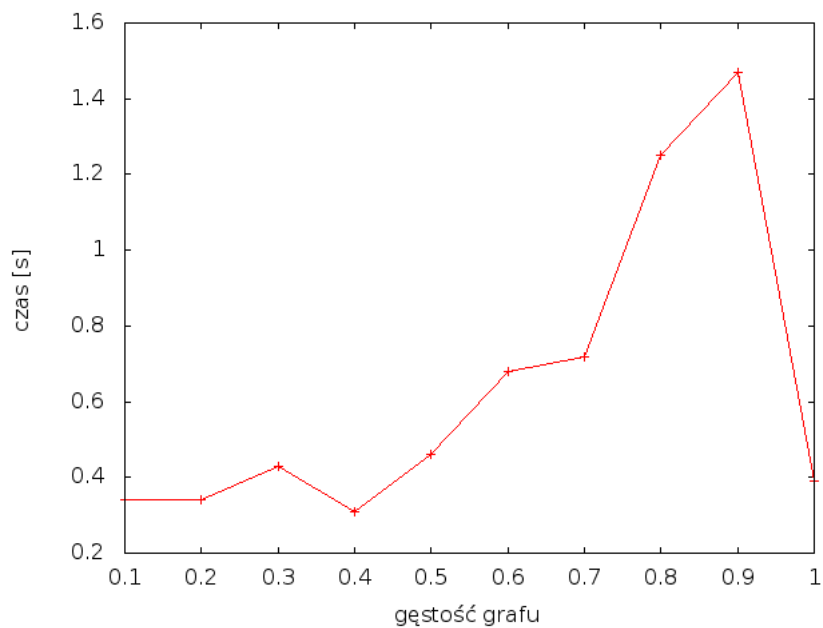
Rysunek 12: Wykres zależności czasu wykonania od gęstości dla grafów o 20 wierzchołkach

### Zestaw grafów o 30 wierzchołkach



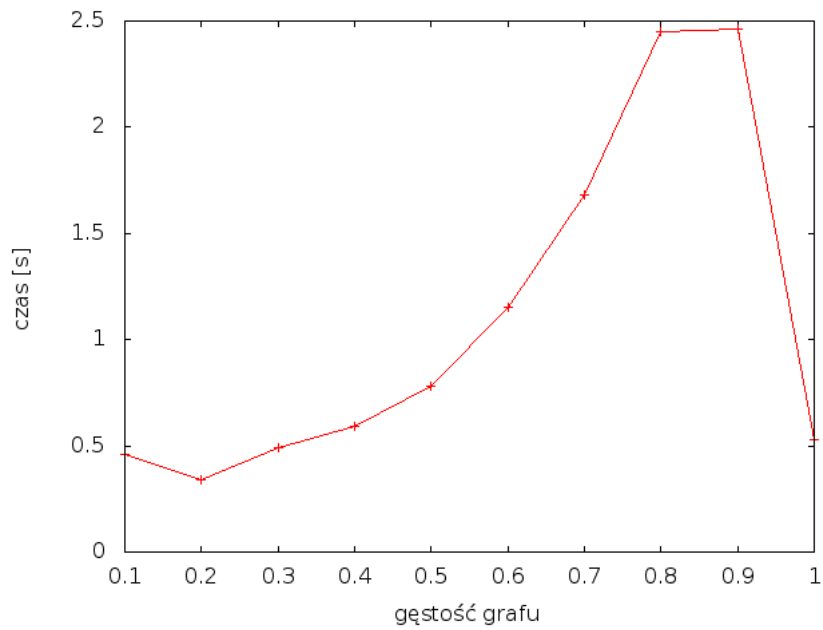
Rysunek 13: Wykres zależności czasu wykonania od gęstości dla grafów o 30 wierzchołkach

### Zestaw grafów o 40 wierzchołkach



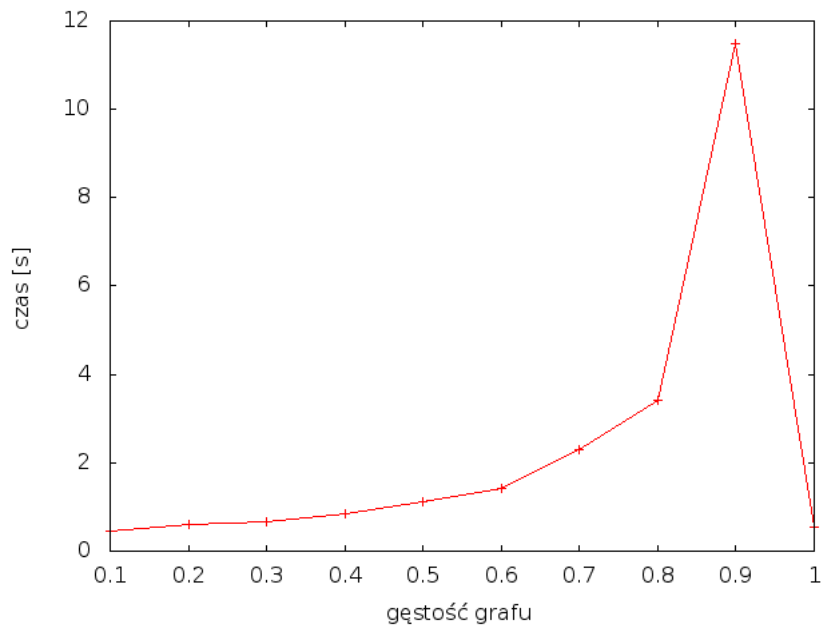
Rysunek 14: Wykres zależności czasu wykonania od gęstości dla grafów o 40 wierzchołkach

### Zestaw grafów o 50 wierzchołkach



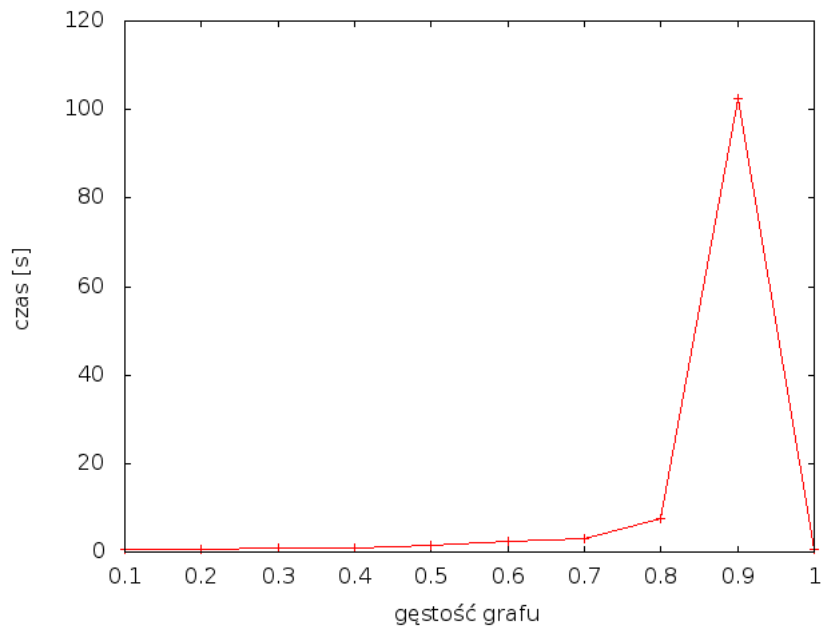
Rysunek 15: Wykres zależności czasu wykonania od gęstości dla grafów o 50 wierzchołkach

### Zestaw grafów o 60 wierzchołkach



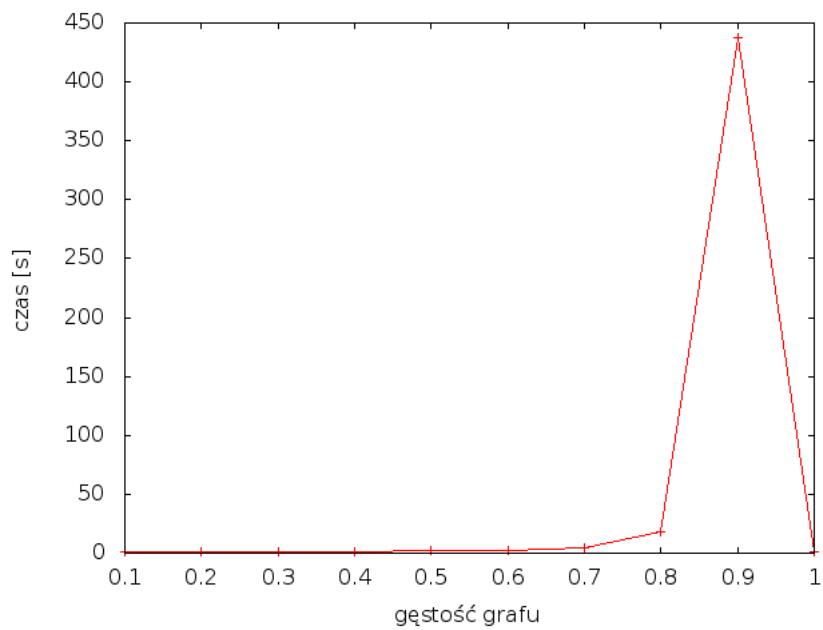
Rysunek 16: Wykres zależności czasu wykonania od gęstości dla grafów o 60 wierzchołkach

### Zestaw grafów o 70 wierzchołkach



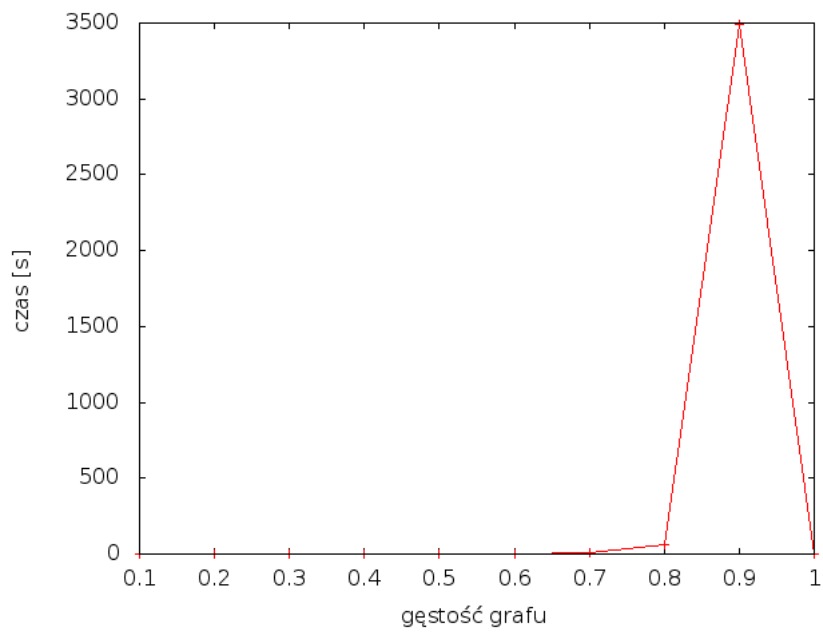
Rysunek 17: Wykres zależności czasu wykonania od gęstości dla grafów o 70 wierzchołkach

### Zestaw grafów o 80 wierzchołkach



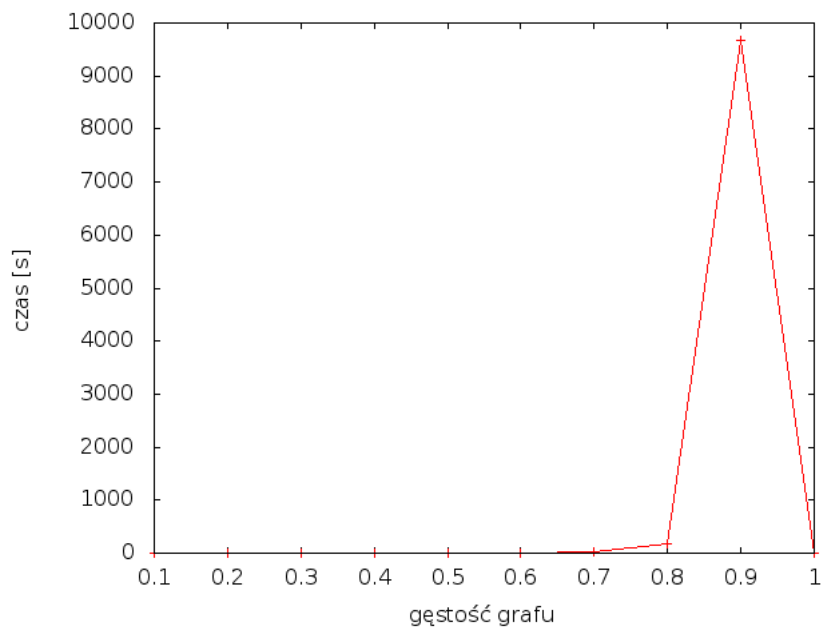
Rysunek 18: Wykres zależności czasu wykonania od gęstości dla grafów o 80 wierzchołkach

### 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 klikli 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 – 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:

- java-workspace - 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 podczas analizy czasów wykonania algorytmu i jego poprawności,
  - performance - pliki z grafami wykorzystanymi do weryfikacji wstępnej poprawności algorytmu.
- 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 `java-workspace/build.sh`.
2. przygotować plik/pliki wejściowe
3. uruchomić program jednym z możliwych sposobów:
  - proste uruchomienie  
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_basic.sh graf1 graf2` (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 rozwiązaniem referencyjnym. Skrypt poszukuje rozwiązanie w pliku o zaczynającym się taką samą nazwą jak plik wejściowy i kończącym się przyrostkiem `out`. W przypadku nie znalezienia pliku z rozwiązaniem aplikacja zostaje uruchomiona w trybie standardowym.  
Przykład wykorzystania: `run_test.sh graf1 graf2` (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.