Autorzy:

Druszcz Bartłomiej

Polaczek Jakub

Grafy i Sieci

Wyznaczanie najszybszego przepływu

Sprawozdanie 3 – testy i analiza algorytmu

Treść zadania:

Sieć przepływowa ma jedno źródło i jedno ujście. Każdy łuk sieci ma określone dwa parametry całkowitoliczbowe: przepustowość oraz czas przepływu. Problem najszybszego przepływu polega na wyznaczeniu najszybszego sposobu przesłania ze źródła do ujścia zadanej wielkości towaru nie przekraczając przepustowości łuków. W ramach projektu należy opracować algorytm i program dla problemu najszybszego przepływu.

Rozważania wstępne:

W rozważanym grafie skierowanym G = (V,E) każdy łuk sieci posiada parametry e = (c, τ) – przepustowość oraz czas przepływu (opóźnienie). Wyróżnione są również dwa węzły sieci: źródło s oraz ujście t oraz objętość przepływu którą należy przetransportować przez sieć – F. Celem opracowanego algorytmu jest minimalizacja globalnego czasu T niezbędnego do przesłania F jednostek przepływu z źródła do ujścia.

Problem najszybszego przepływu jest blisko spokrewniony z klasycznym problemem maksymalnego przepływu, przedstawionym przez Forda-Fulkersona [4]. Algorytm Forda-Fulkersona pozwala wyznaczyć maksymalną przepustowość w sieci przepływowej. Generalizacja tego algorytmu, pozwalająca na uwzględnienie czasu i rozszerzenie zagadnienia o wymiar czasowy i dodanie parametrów reprezentujących opóźnienie przepływu w łuku nazywane jest problemem maksymalnego przepływu dynamicznego [1]. Rozwiązując problem maksymalnego przepływu dynamicznego iteracyjnie modyfikując horyzont czasowy przepływu T metodą bisekcji szukamy takiego czasu Tx, dla którego maksymalny przepływ wynosi F.

Tradycyjnie stosowanym rozwiązaniem problemu maksymalnego przepływu dynamicznego, będącego częścią rozwiązania zadania najszybszego przepływu, jest zastosowanie algorytmu Forda-Fulkersona do rozwiązywania przepływu w tzw. sieci rozszerzonej czasowo (ang. Time expanded network) zawierającej kopię przepływu i sieci dla każdego dyskretnego kroku czasowego. Jest to rozwiązanie dość kosztowne pamięciowo oraz obliczeniowo w porównaniu z najnowszymi metodami opierającymi się na wielu optymalizacjach tej metody m-in parametrycznym przeszukiwaniu grafu przy znajdowaniu ścieżek powiększających [2], czy zgrubnej dyskretyzacji czasowej [1]. Wszystkie te rozwiązania wychodzą jednak z tej samej metody i to właśnie to klasyczne podejście zastosujemy do rozwiązania zadania.

Maksymalny przepływ dynamiczny:

Rozwiązanie maksymalnego przepływu dynamicznego opiera się na iteracyjnym rozwiązywaniu statycznego przepływu algorytmem Forda-Fulkersona. Algorytm ten działa w opierając się o następujące mechanizmy:

- Tworzona jest tzw. Sieć residualna **Gf(V,Ef)** będąca początkowo kopią sieci przepływowej, w czasie działania algorytmu zmieniają się jednak przepustowości łuków tej sieci zgodnie ze wzorem $cf(u,v) = c(u,v) = f(u,v)$, gdzie f(u,v) jest przepływem przez dany łuk w sieci.
- Definiujemy ścieżkę powiększającą jako dowolną ścieżkę p ze źródła s do ujścia t w sieci residualnej i obliczamy związany z nią parametr $cf(p)$ będący minimalną wartością przepustowości krawędzi w danej ścieżce. Ścieżka może być wyznaczona przy pomocy przeszukania w głąb / wszerz (Algorytm Edmundsa-Karpa).

Pseudokod metody Forda-Fulkersona dla statycznego przepływu prezentuje się następująco:

WHILE istnieje ścieżka powiększająca p należąca do grafu powiększającego:

Dla każdej krawędzi (u,v) należącej do p:

$f(u,v) := f(u,v) + cf(p)$

$f(v,u) := f(v,u) - cf(p)$

Praktyczna implementacja tego algorytmu uwzględnia przeszukiwanie sieci wszerz w celu znalezienia ścieżki o najmniejszej liczbie krawędzi nazywana jest algorytmem Edmundsa-Karpa i będzie zaimplementowana w projekcie. Złożoność obliczeniowa tego kroku wynosi O(VE^2), gdzie E jest liczbą krawędzi grafu, V ilością krawędzi.

Rozszerzenie tego algorytmu do problemu maksymalnego przepływu dynamicznego [3] o horyzoncie czasowym T zaczynamy od zdefiniowania sieci rozszerzonej czasowo **D(t).** Wierzchołki P_i^t(i =0 ,n; t=0,T) są wierzchołkami sieci **D(t)**. Krawędzie budujemy z łuków o nieskończonej przepustowości $P_i^t P_i^{t+1}(0 \le t \le T-1)$ oraz dla $i \neq j$ $P_i^t P_i^{t+t_{ij}}(0 \le t \le T - t_{ij})$. Utworzenie takiej rozszerzono czasowo sieci umożliwia sprowadzenie zagadnienia do maksymalnego przepływu statycznego. Taka sieć modeluje zarówno czas przepływu między węzłami sieci jak i pojemność danego węzła. Złożoność czasowa budowy takiej sieci wynosi O(VT + ET) , gdzie T to ilość kroków czasowych (horyzont) [5].

Kolejny krok algorytmu to opisany wcześniej algorytm Edmundsa-Karpa. W przypadku rozszerzonej czasowo sieci o horyzoncie T jego złożoność wyniesie O(VT(ET)^2).

Ostatnim krokiem jest rekonstrukcja sieci czasowo rozszerzonej do przepływu dynamicznego. W przypadku najszybszego przepływu interesuje nas parametr określający ilość przepływu który dotarł do ujścia w analizowanym czasie **Q**.

Najszybszy przepływ:

Główna pętla algorytmu opiera się na uruchamianiu algorytmu wyszczególnionego w poprzednim paragrafie dla zmienianego horyzontu czasowego T i wyszukiwaniu binarnym takiego Tx, Dla którego **Q=F** [1]. Otrzymany najszybszy przepływ należy następnie

zrekonstruować oraz zapisać w wyjściu z programu. Złożoność wyszukiwania binarnego wynosi $O(\log_2 n)$, a pojedynczej iteracji O(VT+ET + VT(ET)^2), zatem całego algorytmu powinna mieścić się w granicach $O(\log_2(VT+ET + VT(ET)^2))$

Wejście i wyjście programu:

Dane wejściowe zapisane będą w postaci listy w pliku tekstowym.

Format zapisu:

s, t, F, liczba_lukow, liczba_wierzcholkow
Vi, Vj, p, c
...

Gdzie:

Vi, Vj - wierzchołki grafu połączone łukiem
s - indeks wierzchołka źródłowego
t - indeks wierzchołka końcowego
F - ilość jednostek przepływu do przetransportowania
p - przepustowość łuku
c - czas przepływu przez łuk

Dane wyjściowe przedstawione będą w formie spisu przepływów pomiędzy węzłami. Format listy pozwoli na wierne odwzorowanie przepływu w ja najmniejszym rozmiarze wyjściowym.

Format wyjściowy:

czas	Vi	Vj	ilość
0	0	1	4
0	0	4	3
1	0	1	4
1	0	4	3

Tabela 1 - Format wyjściowy programu

Szczegóły implementacji:

Program opracowany zostanie w języku C++ w środowisku Windows/Linux. Do obsługi technicznej strony reprezentacji grafu użyta zostanie biblioteka LEMON pozwalająca na wydajną obsługę sieci tego typu. Biblioteka ta zawiera również podstawowe algorytmy grafowe, co może być przydatne przy opracowywaniu rozwiązania.

Rozwiązanie problemu zostało podzielone na dwa podzagadnienia – generację grafu wejściowego oraz rozwiązanie problemu najszybszego przepływu. W tym celu utworzono dwie klasy – QFSolver do rozwiązania problemu najszybszego przepływu oraz GraphGenerator do tworzenia sieci o zadanej liczbie wierzchołków.

Generacja grafu wejściowego

Graf wejściowy generowany jest w sposób losowy – dodawane są krawędzie ze stałym prawdopodobieństwem, następnie w pętli przy pomocy funkcji bibliotecznej do weryfikacji czy graf jest silnie spójny dodawane są kolejne krawędzie. Implementacja biblioteczna w bibliotece LEMON opiera się na przeszukiwaniu wierzchołków w głąb między sobą w pętli i poszukiwaniu niespójnych zbiorów wierzchołków. Przy generacji gafu wejściowego każda krawędź ma

przypisaną losową przepustowość oraz opóźnienie. Losowana jest również wartość przepływu oraz węzły wejściowe i wyjściowe. Stosowane zakresy zmienności to 1-100 dla opóźnień oraz pojemności, 100-1000 dla wartości przepływu.

Rozwiązanie problemu najszybszego przepływu

Program wczytuje sieć przepływową trzech struktur zaimplementowanych w bibliotece LEMON:

- (ListDigraph) baseGraph służąca do przechowania informacji o wierzchołkach i łukach
- (ArcMap) speedMap przechowująca odwzorowanie prędkości (opóźnienia poszczególnych łuków
- (ArcMap) capacityMap przechowująca odwzorowanie przepustowości łuków

Procedura rozwiązania problemu najszybszego przepływu odbywa się przy pomocy kolejnych odwołań do metody największego przepływu i wyszukiwaniu optymalnego horyzontu czasowego metodą bisekcji. W poniższym przykładzie nazwy zmiennych zostały spolszczone na potrzeby pseudokodu:

```
SOLVE:

Granica_dolna := 0
Granica gorna := maksymalne_opoznienie_w_sieci +
maksymalna_pojemnosc_luku_sieci / wartość_przeplywu
WHILE(MAX_PRZEPLYW (granica_gorna) < wartość_przeplywu)
{
     Granica_dolna = granica_gorna
     Granica_gorna *= 2
}
WHILE (granica_gorna – granica_dolna > 1) // wyszukanie rozwiązania
przy pomocy bisekcji
{
     Srodek := SREDNIA(granica_gorna, granica_dolna)
     IF(MAX_PRZEPLYW(srodek) < wartość_przeplywu)
     THEN
          granica_dolna = srodek
     ELSE
          Granica_gorna = srodek
}
RETURN mapa_przeplywu // Mapa rozwiazywana przy okazji funkcji MAXFLOW
```

Funkcja MAX_PRZEPLYW oblicza problem maksymalnego przepływu w sieci rozszerzonej czasowo i jej pseudokod prezentuje się następująco:

```
MAX_PRZEPLYW (horyzont):

ROZSZERZ_CZASOWO(horyzont)
```

```
LEMON::EDMONDS_KARP(rozszerzona_siec, zrodlo, uplyw)

RETURN mapa_przeplywu, wartość_przeplywu
```

Funkcja Edmonds-Karp jest bibliotecznie zaimplementowanym rozwiązaniem problemu maksymalnego przepływu w sieci przepływowej, jej implementacja jest zgodna z opisanym wcześniej algorytmem Edmondsa-Karpa. Funkcja ROZSZERZ_CZASOWO tworzy graf rozszerzony czasowo na bazie bazowego grafu (baseGraph) oraz map opóźnień i pojemności.

```
ROZSZERZ_CZASOWO (horyzont):

graf_rozszerzony =
UTWORZ_GRAF_O_N_WIERZHOLKACH(baseGraph.iloscWierzholkow)
FOR EACH wierzchołek o indeksie ID w graf_rozszerzony:
{
      przesuniecieID : = baseGraph.iloscWierzholkow
      IF istnieje wierzchołek o indeksie (ID + przesuniecieID)
      THEN
            Graf_rozszerzony.dodajLuk(ID, ID + przesuniecieID,
            nieskonczonosc) //Dodaj luk o nieskończonej pojemności
            miedzy sasiednimi plastrami czasowymi grafu rozszerzonego
}

FOR slice := 0 do horyzont:
{
      FOR EACH luk w baseGraph:
      {
            przesuniecieID := baseGraph.iloscWierzholkow *
            luk.opoznienie
            IF istnieje wierzchołek o indeksie (ID + przesuniecieID)
            THEN
                  Graf_rozszerzony.dodajLuk(ID, ID + przesuniecieID,
                  luk.pojemnosc) //Dodaj luk o pojemności bazowego luku
                  miedzy plastrami czasowymi grafu rozszerzonego
                  odległymi o opoznienie

      }

}

RETURN graf_rozszerzony
```

Testowanie wydajnościowe algorytmu:

Poprawność działania algorytmu weryfikowana będzie na dwa sposoby. Pierwszy uwzględnia ręczne rozwiązanie zadania dla trywialnych przypadków oraz porównanie wyników

uzyskanych przez algorytm. Druga metoda zakłada stworzenie losowych, większych sieci oraz porównanie rozwiązania z algorytmem przepływu o minimalnym koszcie zaimplementowanym w ramach używanej biblioteki. Zgodnie z założeniami zagadnienia, czas uzyskany przez algorytm najszybszego przepływu powinien być taki sam lub krótszy od algorytmu najmniejszego kosztu.

Do testów działania algorytmu został napisany specjalny program w języku C++. Program ten generuje graf o losowych parametrach. Zdefiniowano w nim następujące parametry:

n – liczba wierzchołków,
m – liczba krawędzi skierowanych,
pmax – maksymalna przepustowość krawędzi,
tmax – maksymalny czas przepływu przez krawędź,
F – zadana objętość do przetransferowania między źródłem a ujściem,
s – numer wierzchołka początkowego (źródło),
t - numer wierzchołka końcowego (ujście).

W programie zdefiniowano trzy tablice reprezentujące:
- macierz sąsiedztwa grafu,
- przepustowości krawędzi,
- czas przepływu przez krawędź.

- Generacja macierzy sąsiedztwa:

Założeniem projektu jest, że z dowolnego wierzchołka można przetransportować określoną objętość F do innego wierzchołka w tym grafie. Aby ten warunek był spełniony zawsze wszystkie wierzchołki muszą tworzyć cykl. Przy zastosowaniu cyku wykorzystano n krawędzi. Pozostałe m-n krawędzi rozmieszczono losowo w grafie uwzględniając fakt, że na istniejącej krawędzi nie można już postawić innej krawędzi skierowanej.

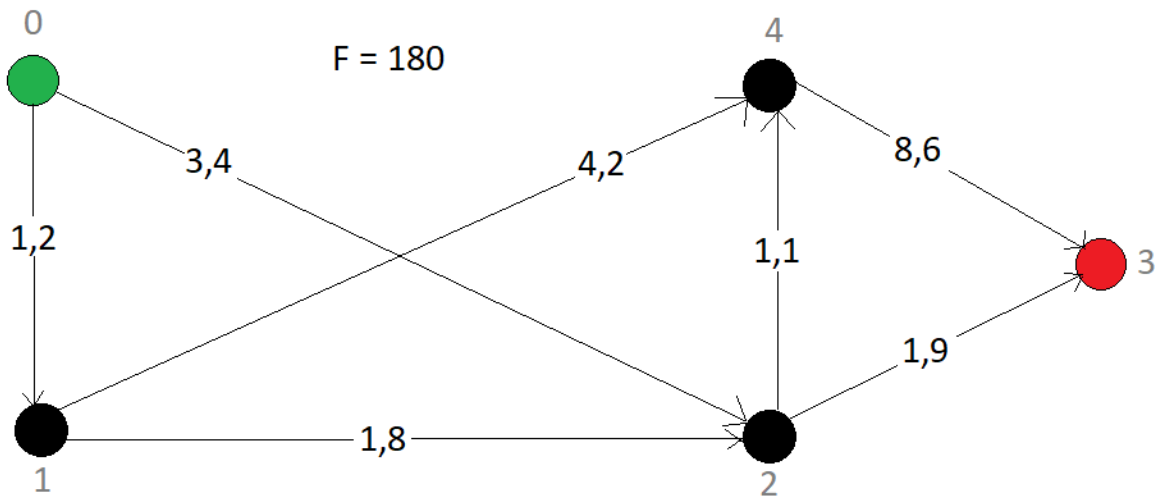
- Generacja przepustowości krawędzi i czasu przepływu przez krawędź:

Dla wszystkich istniejących krawędzi wygenerowano losowe wartości z przedziału od 1 do odpowiednio pmax i tmax.

Tak wygenerowany graf testowy został zapisany w pliku tekstowym w formacie opisanym w podpunkcie *Wejście wyjście programu*. Kolejne kolumny będą oddzielone tabulatorami.

Testowanie poprawności algorytmu, wyniki:

Testy poprawności algorytmu przeprowadzono zgodnie ze schematem przedstawionym poniżej. Parametry przy łukach sieci oznaczają kolejno przepustowość i czas przepływu przez krawędź.



Graf ten w ustalonym poprzednio formacie ma postać:

0	3	20	7	5
0	1	1	2	
0	2	3	4	
1	2	1	8	
1	4	4	2	
2	4	1	1	
2	3	1	9	
4	3	8	6	

Tabela 2 - Wejściowa sieć przepływowa w formacie tekstowym

Oznacza to przepływ 20 jednostek przepływu z węzła 0 do węzła 3 dla grafu o pięciu wierzchołkach i siedmiu łukach. Kolejne wiersze oznaczają kolejne łuki wraz z pojemnością i opóźnieniem.

Plik wynikowy (w formie tabeli) uzyskany przez algorytm wygląda następująco:

czas	vi	vj	F
0	0	1	1
0	0	2	3
1	0	1	1
1	0	2	3
2	0	1	1
3	0	2	3
3	1	4	1
4	0	1	1
4	0	2	3
4	1	4	1
5	0	1	1
6	1	4	1

6	2	4	1
6	2	3	1
6	4	3	1
7	0	1	1
7	1	4	1
7	2	4	1
8	2	3	1
8	4	3	2
8	0	1	1
9	1	4	1
9	2	4	1
9	2	3	1
9	4	3	2
9	0	1	1
10	1	4	1
10	2	4	1
10	2	3	1
11	4	3	2
11	1	4	1
12	2	4	1
12	2	3	1
12	4	3	2
13	1	4	1
13	2	4	1
13	4	3	2
14	2	4	1
15	4	3	2
16	4	3	2

Tabela 3 - Wyjściowe dane przepływu

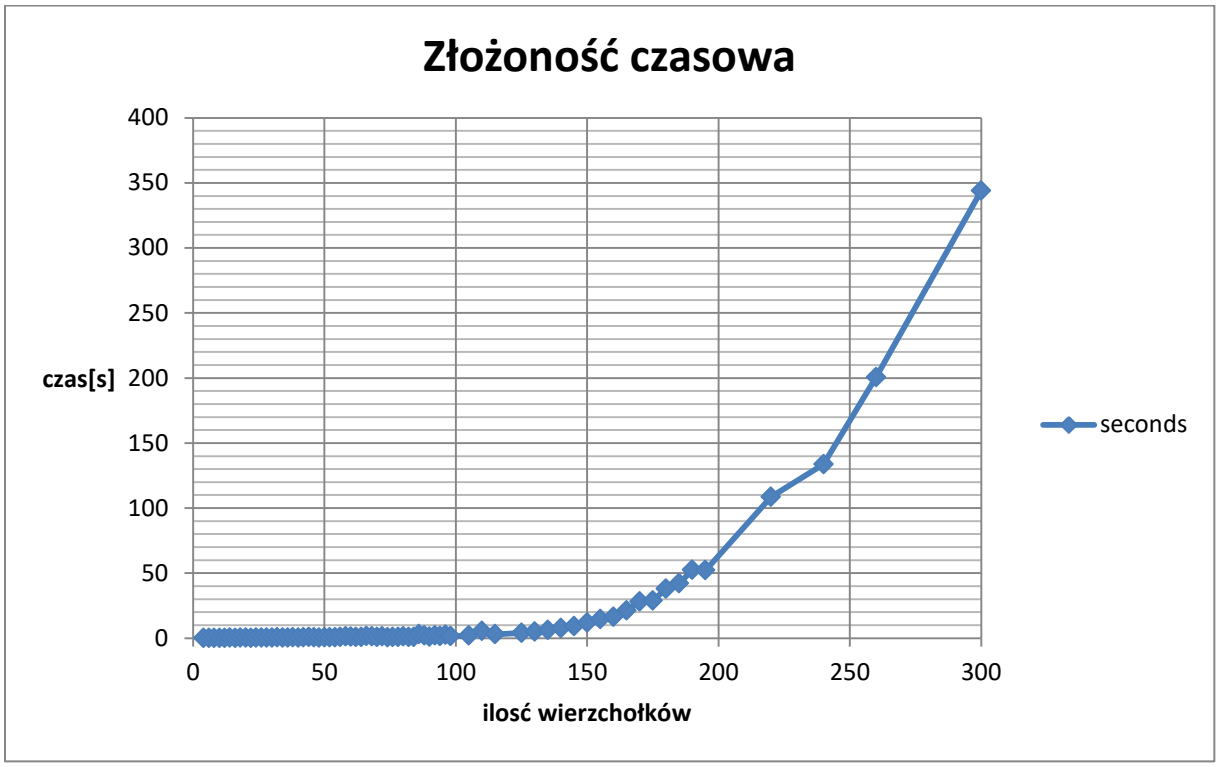
Zapisane są w tym formacie poszczególne przepływy między węzłami – opóźnienia po jakim docierają do kolejnych węzłów musimy wywnioskować z parametrów sieci. Przy głębszej analizie możemy określić, że jest to rozwiązanie poprawne, gdyż uwzględnia podstawowe prawo przepływu – suma wejść równa się sumie wyjść.

Przykładowa interpretacja: W chwili 0 wysłane jest z wierzchołka 0 trzy jednostki w kierunku wierzchołka 2 oraz jedna do 1. Z geometrii sieci wiemy, że dotrą one z opóźnieniem kolejno 4 oraz 3. W krokach czasowych 1 oraz 2 jest analogicznie – przepływ nie dotarł jeszcze do kolejnych węzłów. Dopiero w kroku 3 wysłane jest jedna jednostka z wierzchołka 1 do 4 łukiem o opóźnieniu 2. Proces jest kontynuowany aż cały przepływ znajdzie się w wierzchołku trzecim.

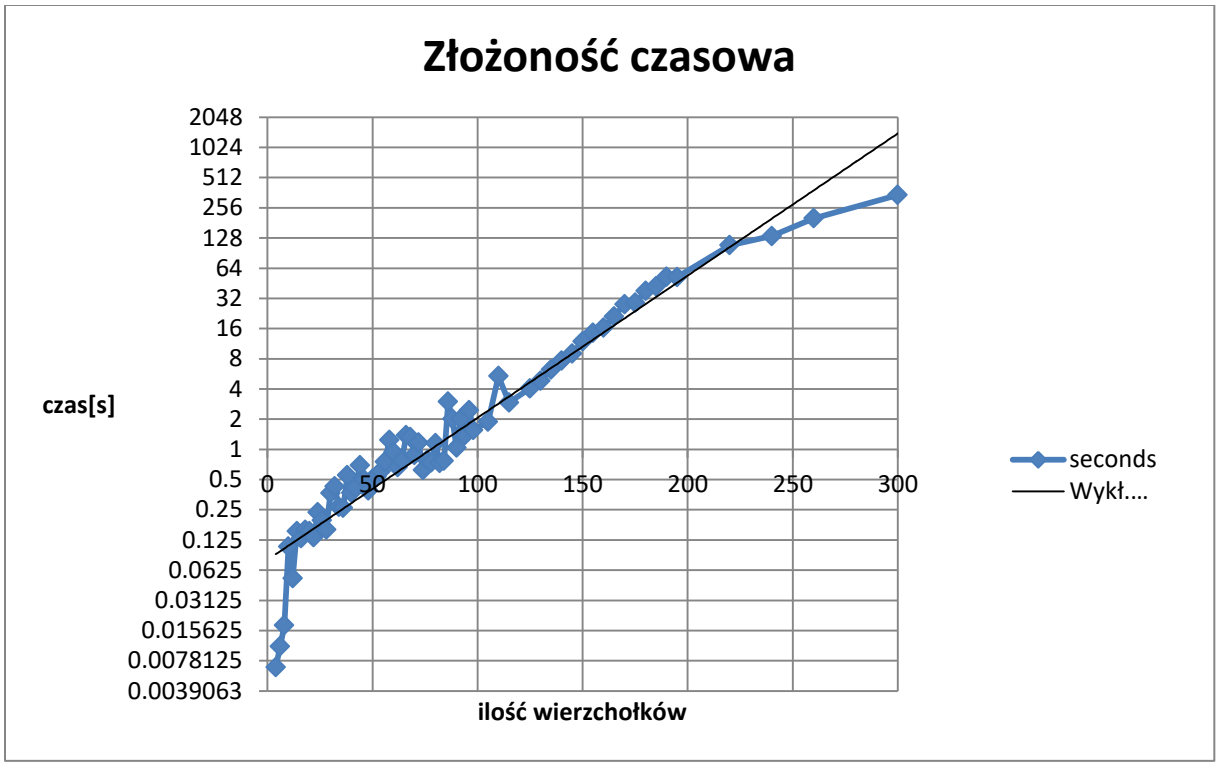
Testowanie wydajności algorytmu, wyniki:

Testy wydajności przeprowadzone zostały na komputerze o procesorze Intel Core i7-4810MQ. Biblioteka LEMON jest jednowątkowa, uruchomiany był jeden test sekwencyjnie. Badano zależność czasu wyszukiwania rozwiązania (mierzoną w sekundach czasu pracy

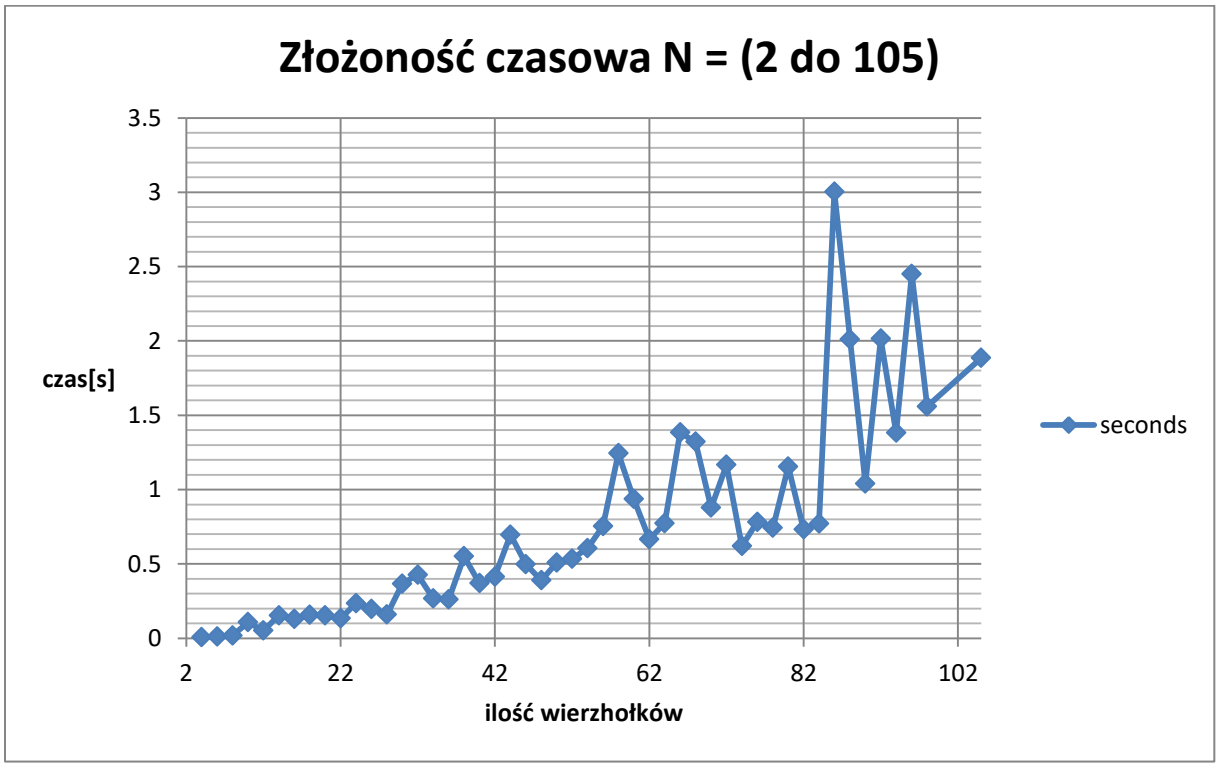
procesora nad zadaniem) od ilości wierzchołków w sieci generowanej zgodnie z procedurą przedstawioną wcześniej. Badano zależność od 2 do 300 wierzchołków uśredniając po 50 pomiarów w zakresie 2 – 100 wierzchołków, 30 w zakresie 100 – 200 oraz po 3 w zakresie 200 -300. Wyniki przedstawiono na poniższych wykresach. Rozdzielczość z jaką mierzono czas wynosi 1ms. Jako, że generowane grafy posiadały ograniczoną wartość przepływu w granicach 100 – 1000 nie badany był wpływ horyzontu czasowego na długość rozwiązania.



Rysunek 1 - Złożoność czasowa w funkcji ilości wierzhołków



Rysunek 2 Zależność złożoności czasowej od ilości wierzchołków w ujęciu skali logarytmicznej o podstawie 2



Rysunek 2 - Zbliżenie zależności złożoności czasowej dla zakresu wierzchołków od 2 do 105

Jak widać na powyższym wykresie (Rysunek 2) złożoność czasowa jest zależnością wykładniczą, tak jak w omawianej wcześniej analizie złożoności.

Wnioski:

Opracowany został algorytm rozwiązujący problem najszybszego przepływu w czasie wykładniczym. Przeprowadzone zostały testy zależności czasu rozwiązania problemu od ilości wierzchołków w grafie silnie spójnym, skierowanym. Wyniki okazały się zgodne z przewidywaną wydajnością. Nie zbadano wpływu horyzontu czasowego na złożoność rozwiązania.

Implementacja algorytmu oparta została o bibliotekę do obliczeń grafowych LEMON w środowisku C++, wykorzystano zaimplementowane e niej algorytmy przeszukiwania w głąb oraz Edmunda-Karpa.

Poprawność algorytmu zweryfikowana została manualnie śledząc wyjście programu i porównując z uprzednio przygotowanym, prostym grafem wejściowym.

Cytowane prace

[1] L. Fleischer and M. Skutalla, “Quickest flows over time,” *Society for Industrial and Applied Mathematics,* pp. 1600-1630, 2007.

[2] R. E. Burkard, K. Dlaska i B. Klinz, „The quickest flow problem,” *ZOR Methods Models,* p. 31–58., 1993.

[3] L. R. Ford i D. Fulkerson, „Constructing maximal dynamic flows from static flows,” The Rand Corporation, Santa Monica, California, 1958.

[4] L. R. Ford i D. Fulkerson, Flows in Networks, Santa Monica, California: Rand corporation, 1962.

[5] M. A. Fonoberova i D. D. Lozovanu, „The maximum flow in dynamic networks,” *Computer Science Journal of Moldova,* nr 387,396, 2004.