Piotr Pilny, Adam Pelikant
Wydział Informatyki i Zarządzania
Wyższa Szkoła Informatyki i Umiejętności
ul. Rzgowska 17a 93-008 Łódź
email: adam.pelikant@p.lodz.pl

WIZUALIZACJA I REORGANIZACJA GRAFÓW Z ZASTOSOWANIEM RELACYJNEJ BAZY DANYCH

Streszczenie – Celem niniejszej pracy jest omówienie metod wizualizacji i reorganizacji grafów nieskierowanych, nieważonych, spójnych, na płaszczyźnie. Przedstawiono dwa algorytmy, które zostały zaimplementowane za pomocą utworzonego oprogramowania Graph Visualizer. Aplikacja ma za zadanie ułożenie wierzchołków zadanego grafu w sposób jak najbardziej czytelny. Jako główne kryteria przyjęto jak najmniejszą liczbę przecięć krawędzi oraz równomierne i symetryczne ułożenie wierzchołków. Oprogramowanie umożliwia także interaktywne tworzenie i modyfikację grafu przez użytkownika. W pracy przedstawiono porównanie obydwu algorytmów, różnice między nimi oraz typowe zastosowania

Słowa kluczowe: Teoria grafów, reorganizacja grafów, wizualizacja grafów, sieci społecznościowe, typy użytkownika CLR, algorytm Fruchtermana-Reingolda, algorytm radialny, programowanie obiektowe w bazach danych

1 Wstęp

Algorytmy grafowe znalazły zastosowanie w olbrzymiej liczbie dziedzin. Wyszukiwanie połączeń komunikacyjnych, trasowanie pakietów w sieciach telekomunikacyjnych, układanie planów zajęć, to tylko kilka najpopularniejszych z wielu zastosowań. Istnieje szereg parametrów charakteryzujących każdą sieć, które opisują różne cechy grafu. Równie istotnym zagadnieniem z punktu widzenia analizy sieci jest jej wizualizacja. Istnieją różne metody realizujące to zadanie.

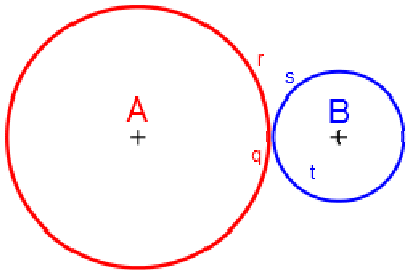
W niniejszej pracy przedstawiono dwie odmienne techniki wizualizacji grafów, z których każda posiada inne cechy i nadaje się do sieci o różnej wielkości:

1. Algorytm Fruchtermana-Reingolda [1] – bazuje na symulacji sił fizycznych. Jest szczególnie efektywny przy grafach relatywnie małych, ze względu na znaczną złożoność obliczeniową.

2. Algorytm kołowy (radialny) [2] – przekształca dowolny graf spójny do postaci drzewa, a następnie układa wierzchołki na koncentrycznych okręgach, których promień rośnie wraz z odległością danego wierzchołka od wierzchołka centralnego. Najlepsze rezultaty osiąga się przy grafach średniej i dużej wielkości.

Aplikacja umożliwia uruchomienie algorytmu Fruchtermana-Reingolda sekwencyjnie, co pozwala obserwować postęp działania i wygląd grafu na poszczególnych etapach. W przypadku algorytmu radialnego zastosowano animację, dzięki której wierzchołki przesuwają się płynnie z początkowego do docelowego położenia.

Na potrzeby niniejszej pracy stworzono dwa typy użytkownika (CLR) [3], [8]: SocNetCluster i SocNetPerson. Pierwsza z nich definiuje klaster (inaczej grono), czyli grupę osób zamieszkujących w pobliżu danego lokalnego centrum (np. dużego miasta). Klastrem jest obszar w kształcie koła okalający miasto, które staje się stolicą tego klastra. Drugi UDT definiuje osobę. Osoba zostaje przypisana do tego klastra, do którego granicy ma najmniejszą odległość (rysunek 1). Jeśli dwa klastry znacznie różniące się wielkością są położone w niewielkiej odległości od siebie, wówczas osoba, której lokalizacja jest w pobliżu obydwu klastrów, zostanie włączona do tego, którego granicy bliżej się znajduje. Na rysunku przedstawione zostały przykładowe klastry A i B oraz węzły sieci q, r, s, t. Gdyby za kryterium przyłączenia przyjąć odległość do środka klastra, wówczas węzeł q zostałaby włączona do klastra B, choć ewidentnie znajduje się w granicach klastra A. Z kolei węzeł r leży poza granicami obydwu klastrów, jednak – mimo, iż bliżej ma do środka klastra B – zostanie włączona do A, ze względu na mniejszą odległość do jego granicy.



Rys. 1. Przypisanie osoby do klastra

Opisana struktura jest podstawą do stworzenia modelu sieci społecznej. Jako źródło danych posłużyły tabele słownikowe, zawierające wykaz polskich nazwisk, imion, miast, wraz z licznościami w każdej z tych grup. Wzajemne powiązania między osobami (więzy znajomości) znakomicie nadają się do zaimplementowania w postaci grafu, którego wierzchoł-

kami są członkowie sieci społecznej, a powiązania z innymi osobami są jego krawędziami. Ze względu na charakter sieci społecznej graf jest nieskierowany. Oznacza to, że dwie osoby znają się wzajemnie (relacja znajomości jest dwukierunkowa). W celu opisania struktury danych utworzono dwie klasy obiektowe enkapsulowane do typu użytkownika po stronie serwera bazy danych.

Struktura obiektu SocNetCluster jest następująca:

Pola:

1) *id* - identyfikator klastra,
2) *name* - nazwa klastra,
3) *area* - powierzchnia klastra,
4) *population* - liczba ludności,
5) *cityLat* - szerokość geograficzna miasta bazowego klastra,
6) *cityLng* - długość geograficzna miasta bazowego klastra,
7) *lat* - szerokość geograficzna środka klastra,
8) *lng* - długość geograficzna środka klastra,
9) *radius* - promień klastra.

Metody:

1) *getDistanceToCluster* - oblicza odległość do innego klastra,
2) *getDistanceToPerson* - oblicza odległość od klastra do osoby,
3) *PersonCount* - zwraca liczbę osób w klastrze.

Natomiast struktura obiektu SocNetPerson ma postać:

Pola:

1) *cluster* - klaster, do którego przypisana została osoba,
2) *lastName* - nazwisko osoby,
3) *firstName* - imię osoby,
4) *male* - płeć osoby,
5) *city* - miejscowość zamieszkania,
6) *dateOfBirth* - data urodzenia osoby,
7) *lat* - szerokość geograficzna miasta,
8) *lng* - długość geograficzna miasta,
9) *email* - adres email osoby,
10) *interests* - hobby osoby,
11) *degree* - stopień wierzchołka (liczba znajomych osoby).

Metody:

1) *getDistanceToCluster* - oblicza odległość do klastra,
2) *getDistanceToOwnCluster* - oblicza odległość do własnego klastra,
3) *getDistanceToPerson* - oblicza odległość do innej osoby,
4) *IncreaseDegree* - zwiększa stopień wierzchołka o 1,
5) *DecreaseDegree* - zmniejsza stopień wierzchołka o 1.

W wyniku skompilowania projektu powstaje biblioteka SocNet.dll. W celu skorzystania z nowo utworzonego typu użytkownika, należy przeprowadzić kilka czynności po stronie SQL Servera [3], [8]. Na

początku konieczne jest wykonanie systemowej procedury włączającej rozszerzenie CLR. Następnie musimy stworzyć zestaw (ang. assembly). Zestaw może zawierać wiele typów użytkownika i elementów proceduralnych (funkcji, procedur, wyzwalaczy). Kolejnym krokiem jest utworzenie typów użytkownika, zawartych we włączonym przed chwilą zestawie.

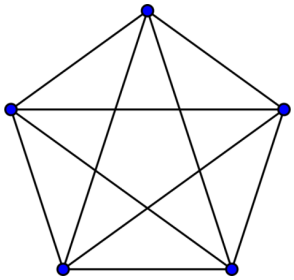
Podczas tworzenia typów użytkownika przyjęto, że separatorem poszczególnych składowych będzie znak '|'. Tak zapisany łańcuch znaków jest przesyłany przez SQL Server do CLR, który uruchamia kod zarządzany skompilowany do biblioteki SocNet.dll. Program dokonuje analizy łańcucha (odpowiada za to statyczna metoda Parse) i tworzy obiekt, który następnie zwraca do serwera. Cała operacja wykonywana jest w kontekście serwera.

2 Elementy teorii grafów

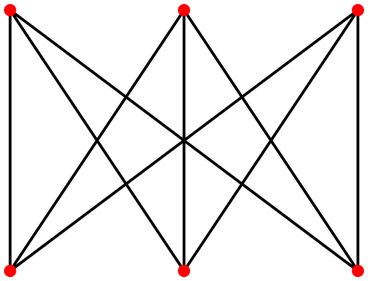
Grafem G nazywamy uporządkowaną parę G = (V, E), gdzie V to zbiór wierzchołków, E to zbiór krawędzi. Wierzchołki mogą być połączone krawędziami. Każda krawędź łączy ze sobą dwa wierzchołki. Możliwe jest połączenie wierzchołka krawędzią z samym sobą. Wówczas mamy do czynienia z pętlą. Jeśli możliwe jest przejście z każdego wierzchołka do pozostałych, to taki graf nazywamy spójnym. Graf pełny to taki, w którym każda para wierzchołków jest połączona krawędzią. Każdej krawędzi można przypisać wartość, stanowiącą jej wagę. Mówimy wówczas o grafie ważonym. Stosuje się je m.in. do reprezentacji map drogowych, gdzie każdy wierzchołek reprezentuje inną miejscowość, zaś drogi są krawędziami. W tym wypadku kosztem krawędzi może być np. dystans lub czas przejazdu. Innym przykładem zastosowania grafów ważonych są sieci telekomunikacyjne. Jeśli rutery lub inne punkty dostępowe do sieci potraktujemy jako wierzchołki, a łącza pomiędzy nimi jako krawędzie, wówczas kosztem krawędzi może być przepustowość danego łącza. Jeśli krawędzie posiadają zwrot, to graf zawierający takie krawędzie nazywamy grafem skierowanym. Trasą nazywamy ciąg krawędzi E_1, E_2, …, E_n taki, że koniec krawędzi E_i jest początkiem krawędzi E_{i+1}. Trasę, w której krawędzie nie powtarzają się, nazywamy drogą. Graf jest cykliczny (posiada cykle), jeśli istnieje droga, której początek i koniec są równe. Grafy spójne, nieskierowane, nie posiadające cykli (acykliczne) są równoważne drzewom. Stopniem wierzchołka określamy liczbę sąsiadujących z nim krawędzi. Podgrafem grafu G nazywamy graf powstały przez usunięcie z grafu G pewnej liczby wierzchołków (razem ze wszystkimi przyległymi do niego krawędziami) lub krawędzi. Rozszerzenie grafu to operacja polegająca na zastąpieniu krawędzi drogą złożoną z co najmniej dwóch nowych krawędzi. Grafem

rozszerzonym grafu G nazywamy graf powstały przez co najmniej jedno rozszerzenie grafu G.

Istotnym z punktu widzenia wizualizacji parametrem grafu jest planarność. Graf jest planarny, jeśli możliwe jest takie jego przedstawienie na płaszczyźnie, żeby krawędzie grafu nie przecinały się. Zgodnie z twierdzeniem Kuratowskiego [4], graf jest planarny wtedy i tylko wtedy, kiedy nie zawiera podgrafu będącego grafem rozszerzonym K_5 (graf pełny o pięciu wierzchołkach) lub $K_{3,3}$ (graf pełny dwudzielny o sześciu wierzchołkach, z których trzy są połączone z każdym z pozostałych trzech). Grafy K_5 i $K_{3,3}$ przedstawiono odpowiednio na rysunkach 2 i 3.



Rys. 2. Graf K_5



Rys. 3. Graf $K_{3,3}$

Przedstawienie na płaszczyźnie w sposób czytelny dla odbiorcy grafu planarnego jest łatwiejsze, ze względu na możliwość jego wizualizacji bez przecinających się krawędzi.

3 Algorytmy wizualizacji grafów

3.1 Algorytm Fruchtermana-Reingolda

Peter Eades zaprezentował w 1984 r. [5] algorytm wizualizacji grafów polegający na zastosowaniu modelu sił fizycznych oddziałujących na wierzchołki. Główna idea polega na traktowaniu wierzchołków jako ładunków elektrycznych o tym samym znaku, które oddziałują na siebie wzajemnie określoną siłą odwrotnie proporcjonalną do odległości między nimi, oraz krawędzi jako sprężyn, które przyciągają połączone wierzchołki z siłą wprost proporcjonalną do odległości. P. Eades w swojej propozycji nie uwzględnił prawa Hooke'a. W zamian zastosował własną formułę do obliczenia sił wywieranych przez sprężyny. Algorytm należy do grupy tzw. algorytmów ukierunkowanej siły (ang. force-directed graph drawing).

Jednym z rozszerzeń wymienionego algorytmu był model, który przedstawili w 1989 r. Tomihisa Kamada i Satoru Kawai [6]. Zaproponowali oni metodę, w której wyznacza się idealną odległość dla każdej pary wierzchołków, proporcjonalną do najkrótszej ścieżki między nimi. Kamada i Kawai sformułowali problem ułożenia grafu jako proces redukcji całkowitej energii układu sprężyn (krawędzi / ścieżek) łączących stalowe pierścienie (wierzchołki). W grafie spójnym dla każdej pary wierzchołków istnieje ścieżka. Sprężynami w układzie Kamada-Kawai są zatem ścieżki, a nie krawędzie (w szczególnych wypadkach ścieżka może składać się tylko z jednej krawędzi). Jednocześnie autorzy zrezygnowali z traktowania wierzchołków jako ładunków oddziałujących na siebie nawzajem, na rzecz sprężyn działających przeciwstawnymi siłami (odpychającymi i przyciągającymi, w zależności od odległości). Jeśli dystans między wierzchołkami jest zbyt duży w stosunku do idealnego, sprężyna przyciąga wierzchołki. Analogicznie, w wypadku zbyt blisko siebie położonych wierzchołków, siła sprężyny je oddala. Im bliższa idealnej jest odległość między wierzchołkami, tym całkowita energia jest mniejsza i układ się stabilizuje.

Thomas M. J. Fruchterman i Edward M. Reingold [1] zaprezentowali w 1991 r. swoje rozwinięcie algorytmu force-directed opartego na idei Eades'a. W ich ujęciu, przy obliczaniu siły odpychania brane są pod uwagę wszystkie pary wierzchołków, z możliwością co najwyżej ograniczenia obliczeń do pewnego arbitralnie ustalonego obszaru w obrębie danego wierzchołka, w celu zmniejszenia złożoności obliczeniowej algorytmu. Poprzez zastosowanie tzw. symulowanego wyżarzania (ang. simulated annealing), można osiągnąć zadowalający rezultat zanim wykonane zostaną wszystkie przewidziane iteracje. Stosuje się dodatkową składową – temperaturę, która po każdej iteracji jest obniżana. Układ stabilizuje się w miarę postępu działania algorytmu.

Przebieg algorytmu Fruchtermana-Reingolda w postaci pseudokodu można przedstawić następująco:

```
G=(V,E)
area=W*L (W-szerokość_ramki, L-długość_ramki)
temp=max (W,L)/10
k=√area/(|V|)
funkcja f_a (x)=x^2/k
funkcja f_r (x)=k^2/x
for i=1 to n do begin
   {oblicz siły odpychania}
   for v in V do begin
     v.disp=0
     for u in V do
       if (u≠v)  then begin
         δ=v.pos-u.pos
```

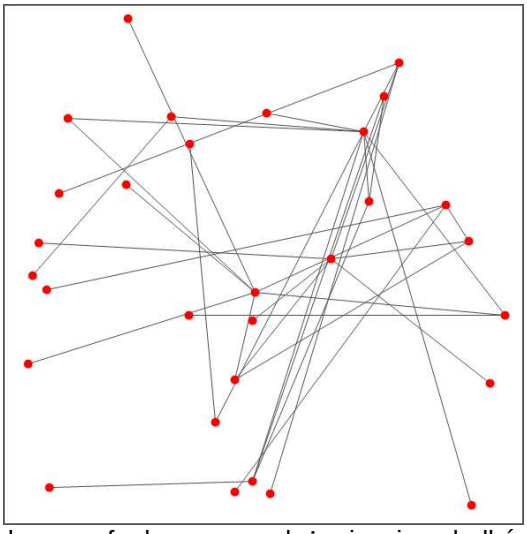
```
            v.disp=v.disp+(δ/|δ| )*f_r (|δ|)
          end
        end
        {oblicz siły przyciągania}
        for e in E do begin
          δ=e.v.pos-e.u.pos
          e.v.disp=e.v.disp-(δ/|δ| )*f_a (|δ|)
          e.u.disp=e.u.disp+(δ/|δ| )*f_a (|δ|)
        end
        {oblicz nowe pozycje wierzchołków}
        for v in V do begin
          v.pos=v.pos+((v.disp)/|v.disp| )*min(v.disp,t)
        end
        {zmniejsz temperaturę (chłodzenie)}
        t=cool(t)
      end
    end
```

Sposób działania algorytmu jest następujący. Współrzędne wierzchołków są inicjalizowane losowymi wartościami. Obliczana jest powierzchnia (area) obszaru do rysowania. Zmienna temp (temperatura) inicjalizowana jest jedną dziesiątą długości dłuższego boku powierzchni. Obliczana jest wartość współczynnika k, który ogranicza przesunięcie. Następnie w n iteracjach wykonywane są kolejno:

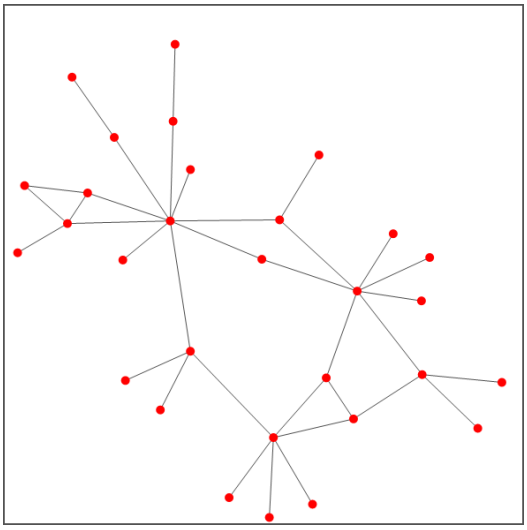
1. Obliczenie wektorów przesunięć dla każdego wierzchołka w stosunku do pozostałych (siła odpychania).
2. Obliczenie wektorów przesunięć dla każdej pary wierzchołków połączonych krawędzią (przyciąganie).
3. Obliczenie nowych położeń wszystkich wierzchołków, z uwzględnieniem wektorów przesunięć. W tym kroku brana jest pod uwagę wartość temperatury (t).
4. Zmniejszenie temperatury (chłodzenie).

Rezultat działania algorytmu Fruchtermana-Reingolda na losowym grafie składającym się z 30 wierzchołków przedstawiają rysunki 4 i 5.

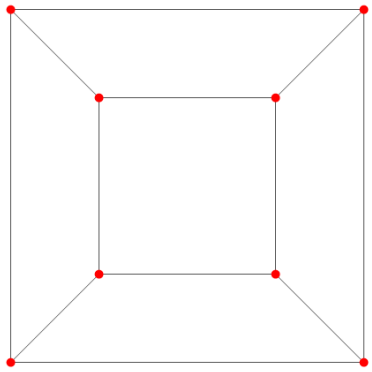
Złożoność obliczeniowa algorytmu wynosi dla jednej iteracji. Z tego względu jego przydatność, szczególnie dla rozwiązań interaktywnych, ogranicza się do grafów relatywnie niewielkich, o liczbie wierzchołków nieprzekraczającej kilkuset. Dla grafu o 100 wierzchołkach i 150 krawędziach algorytm musi wykonać 10 150 operacji. Przy 5 000 wierzchołków i 7 500 krawędzi liczba ta rośnie do 25 007 500. Cechą algorytmu Fruchtermana-Reingolda jest równomierna dystrybucja wierzchołków i dobre rezultaty nawet przy dość gęstych grafach. Algorytm ten nie zapewnia wizualizacji grafu planarnego bez przecięć. Nie zawsze jednak takie zachowanie jest pożądane, ponieważ niektóre grafy są czytelniejsze mimo nieplanarnego wyglądu. Taki przykład ilustrują rysunki 6 i 7.



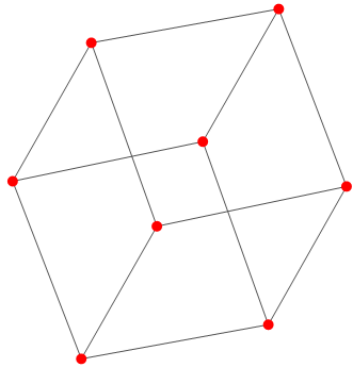
Rys. 4. Przykładowy graf o losowym położeniu wierzchołków



Rys. 5. Graf ułożony algorytmem Fruchtermana-Reingolda



Rys. 6. Graf planarny



Rys. 7. Graf planarny ułożony algorytmem Fruchtermana-Reingolda

3.2 Algorytm radialny

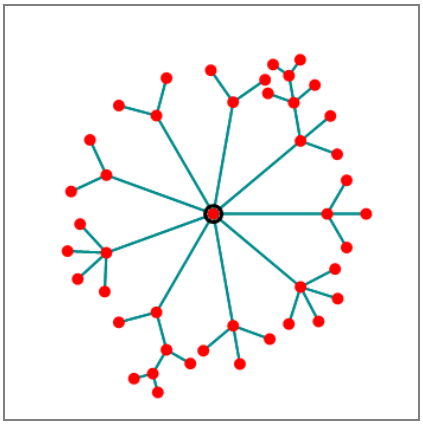
W pewnych zastosowaniach atrakcyjnym sposobem przedstawienia grafu jest ułożenie wierzchołków na jednym bądź kilku okręgach. Różne odmiany tego typu algorytmów oferują różne sposoby wizualizacji:

1. Wierzchołki układane są na jednym okręgu. Czytelność takiego układu jest ograniczona do grafów o bardzo małej liczbie wierzchołków (<100). W szczególności przy grafach gęstych uzyskanie estetycznego wyglądu może okazać się niewykonalne.
2. Wierzchołki pozycjonowane są na koncentrycznych okręgach, o promieniach zależnych od odległości w grafie danego wierzchołka od wierzchołka centralnego. Stosuje się przekształcanie grafu w

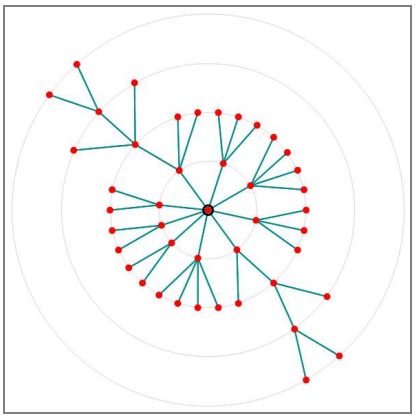
drzewo, najczęściej za pomocą algorytmu Breadth-First Search (BFS). Korzeń drzewa staje się wierzchołkiem centralnym (2).

3. Wierzchołki układane są na okręgach o promieniach malejących wraz z odległością od wierzchołka centralnego (korzenia), o centrach w wierzchołku-rodzicu. Istotnym ograniczeniem układu jest szybkie zmniejszanie się promieni i spadek czytelności w miarę wzrostu głębokości drzewa (liczby poziomów).

Algorytm zaproponowany przez [7] nosi nazwę parent-centered radial layout. Przykładowy graf ułożony takim algorytmem przedstawiono na rysunku 8. Dla porównania na rysunku 9 zaprezentowany został efekt działania algorytmu radialnego na tym samym grafie.

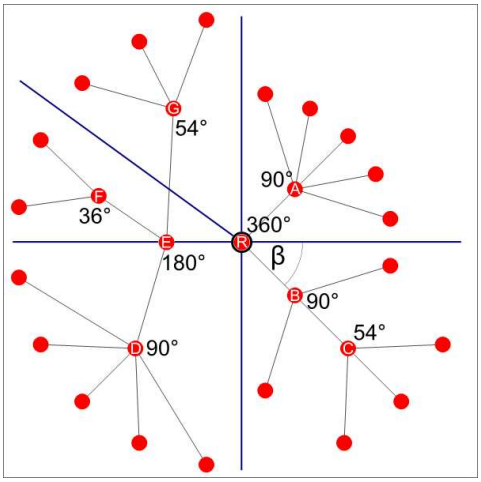


Rys. 8. Przykładowy graf ułożony algorytmem parent-centered radial layout



Rys. 9. Graf w układzie radialnym

Algorytm wizualizacji oparty na koncentrycznych okręgach jest czytelny nawet przy dużych grafach. W pierwszym kroku należy dokonać wyboru wierzchołka centralnego (korzenia). Domyślnie korzeniem staje się wierzchołek o największym stopniu (liczbie krawędzi). Następnie graf przekształcany jest w drzewo przy użyciu algorytmu Breadth-First Search (BFS). Wierzchołek główny pozycjonowany jest na środku ramki i przypisuje mu pełny kąt (360°). Każdemu potomkowi przydzielany jest kąt środkowy, czyli wycinek przeznaczony dla jego poddrzewa (nazwijmy go α) proporcjonalny do ilorazu liczby liści należących do jego poddrzewa w stosunku do łącznej liczby liści w grafie. Postępowanie to powtarza się dla wszystkich wierzchołków z wyjątkiem liści. Znając szerokość α, ustala się jego granice w następujący sposób: półprosta łącząca rodzica z korzeniem staje się dwusieczną α wierzchołka-dziecka. Przydzielony obszar dzieli się na równe części dla wszystkich bezpośrednich potomków rozpatrywanego wierzchołka. Każdy z wierzchołków-dzieci pozycjonowany jest w taki sposób, aby półprosta łącząca go z wierzchołkiem centralnym stanowiła dwusieczną kąta α przydzielonego temu wierzchołkowi. Przykład przedstawiono na rysunku 10.



Rys. 10. Schemat podziału powierzchni dla układu radialnego

Wierzchołkiem głównym (korzeniem) jest wierzchołek oznaczony literą R. Posiada on pełny kąt (360°). W grafie (drzewie) jest 20 liści (wierzchołków, które nie posiadają potomków). Wierzchołki A, B i D mają przydzielone po 90°, ponieważ każdy posiada w swoim poddrzewie 5 liści (5 / 20 * 360 = 90), przy czym wierzchołki A i D posiadają tylko liście. Wierzchołek B ma 2 liście oraz poddrzewo z korzeniem w wierzchołku C. Poddrzewo C zawiera 3 liście, zatem dla niego α = 3 / 20 * 360 = 54°. Dla wierzchołka E kąt α = 180°, ponieważ w jego

poddrzewie znajduje się 10 liści (5 dla D, 2 dla F, 3 dla G). Analogicznie, poddrzewa F i G otrzymują wartości α odpowiednio 36° (2 liście: 2 / 20 * 360 = 36) i 54° (2 liście: 3 / 20 * 360 = 54).

Graf rysujemy począwszy od korzenia. Umiejscawiamy go w środku ramki. Definiujemy β: kąt między odcinkiem łączącym korzeń z rysowanym wierzchołkiem a dodatnią półosią OX, wyznaczany zgodnie z ruchem wskazówek zegara (z zakresu 0° – 360°).

Dla pierwszego rysowanego wierzchołka będącego dzieckiem danego rodzica wartość β obliczamy następująco:

$\beta = \beta_r - \alpha_r / 2 + \alpha / 2$, gdzie:

β_r – kąt β rodzica,

α_r – kąt α rodzica.

Kąt α dla wierzchołka głównego wynosi 360°, a β wynosi 90°.

Dla kolejnych dzieci danego rodzica β wyliczamy ze wzoru:

$\beta = \beta_p + \alpha_p / 2 + \alpha / 2$, gdzie:

β_p – kąt β ostatnio rysowanego wierzchołka,

α_p – kąt α ostatnio rysowanego wierzchołka.

Dla przykładowego grafu z rysunku 10 postępujemy następująco: dla wierzchołka A wyznaczamy kąt β:

$\beta_A = \beta_r - \alpha_r / 2 + \alpha / 2 = 90 - 360 / 2 + 90 / 2 = -45°$

Bierzemy kolejny wierzchołek (B) i znów obliczamy β, tym razem z drugiego wzoru:

$\beta_B = \beta_p + \alpha_p / 2 + \alpha / 2 = -45 + 90 / 2 + 90 / 2 = 45°$

Dla wierzchołka E:

$\beta_E = \beta_p + \alpha_p / 2 + \alpha / 2 = 45 + 90 / 2 + 180 / 2 = 180°$

Dla wierzchołka D (zmiana rodzica; rodzicem D jest E):

$\beta_D = \beta_r - \alpha_r / 2 + \alpha / 2 = 180 - 180 / 2 + 90 / 2 = 135°$

Przechodząc drzewo wszerz, postępujemy analogicznie dla wszystkich wierzchołków, aż dotrzemy do ostatniego liścia.

Przebieg algorytmu BFS w postaci pseudokodu ma następującą postać:

```
G=(V,E)
s-korzeń drzewa
Q-kolejka priorytetowa
funkcja Q.pop()-funkcja zdejmująca wierzchołek z
kolejki
funkcja Q.push(v)-funkcja wkładająca wierzchołek v do
kolejki
for v in V do begin
   kolor[u]=biały
   odległość[u]=∞
   rodzic[u]=NULL
end
kolor[s]=szary
odległość[s]=0
```

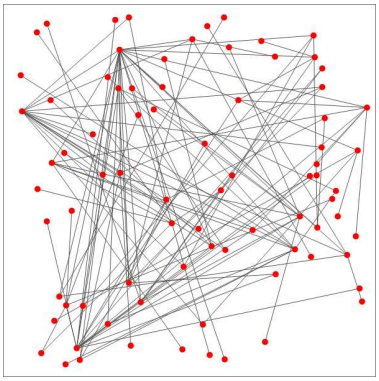
```
rodzic[s]=NULL
Q.push(s)
while Q niepuste do begin
   u=Q.pop()
   for v in lista sąsiedztwa u do begin
      if (kolor[v]==biały)  then begin
         kolor[v]=szary
         odległość[v]=odległość[u]+1
         rodzic[v]=u
         Q.push(v)
      end
      kolor[u]=czarny
   end
end
```

Złożoność obliczeniowa algorytmu w pesymistycznym wariancie wynosi $O(|V|+|E|)$. Efektem wykonania powyższego kodu jest powstanie drzewa o korzeniu w s. Wierzchołek ten stanie się wierzchołkiem głównym (korzeniem) układu radialnego. Drzewo nie zawiera cykli (pomiędzy każdą parą wierzchołków istnieje dokładnie jedna ścieżka).

Przebieg algorytmu radialnego w postaci pseudokodu ma postać:

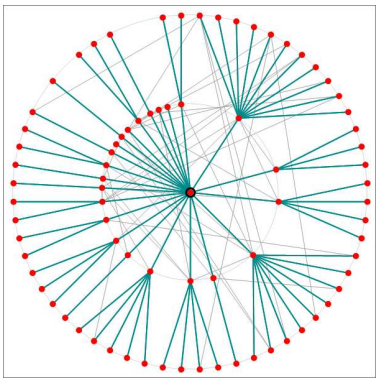
```
G=(V,E)
DFS(G,v)-funkcja przekształcająca graf G w drzewo o
korzeniu w v
D – głębokość drzewa
LVL-poziom wierzchołka (odległość od korzenia)
L-liczba liści w całym poddrzewie danego wierzchołka
W,H - wymiary (szerokość i wysokość) ramki do
wizualizacji grafu
R – promień pierwszego okręgu (najbliższego
korzeniowi)
T=DFS(G,v)
R=min (W,H)/(2*D)
Arc=0
LastAngle=0
ParentArc=0
v.x=W/2
v.y=H/2
v.angle=π/2
{przechodzimy drzewo wszerz, począwszy od dzieci
korzenia do liści}
for i=2 to count(T) do begin
   Arc=(T[i].L)/(v.L)*2*π
   ParentArc=(T[i].parent.L)/(v.L)*2*π
   if (T[i].parent≠T[i-1].parent)  then begin
      T[i].angle=T[i].parent.angle-ParentArc/2+Arc/2
```

```
      else then begin
        T[i].angle=LastAngle+LastArc/2+Arc/2
      end
      T[i].x=v.x+R*T[i].LVL*cos (T[i].angle)
      T[i].y=v.y+R*T[i].LVL*sin (T[i].angle)
      LastAngle=T[i].angle
      LastArc=Arc
    end
```



Rys. 11. Przykładowy graf o losowym położeniu wierzchołków

Złożoność obliczeniowa algorytmu (z pominięciem etapu tworzenia drzewa algorytmem BFS) wynosi $O(|V|)$. Jest to więc algorytm bardzo szybki, działający czasie liniowym, co predestynuje go do zastosowań w wizualizacjach interaktywnych. Na rysunku 11 przedstawiono przykładowy, 80-wierzchołkowy graf, o losowym położeniu wierzchołków.



Rys. 12. Graf ułożony algorytmem radialnym

Na rys.12 przedstawiono ten sam graf ułożony algorytmem radialnym. Krawędzie włączone do drzewa zostały wyróżnione innym kolorem, zaś pozostałe – narysowane w kolorze jasnoszarym, w celu poprawy czytelności wizualizacji. Pokazano także okręgi, na których umiejscowione zostały wierzchołki.

4 Program Graph Visualizer

Graph Visualizer to aplikacja stworzona w celu zaprezentowania algorytmów prezentacji grafów, sposobów implementacji struktur i pokazania problemów występujących w tych procesach, z koncepcjami ich rozwiązania. Dokonano wyboru dwóch algorytmów wizualizacji, które zostały zaimplementowane w aplikacji:

1. Algorytm Fruchtermana-Reingolda,
2. Algorytm radialny

Decydując się na wybór tych dwóch algorytmów, kierowano się przede wszystkim kryterium przydatności do odzwierciedlenia struktury sieci społecznej (lub jej fragmentu). Obydwa algorytmy nadają się do tego celu: przedstawiają grafy spójne, nieskierowane, nieważone, cykliczne, a sieci społeczne charakteryzują się tymi cechami.

Aplikacja z założenia wyświetla graf pobrany z bazy danych, w której został zapisany w tabelach za pomocą typów użytkownika SocNetCluster i SocNetPerson. Dla wygody użytkownika i w celu ułatwienia testowania przedstawionych rozwiązań dla różnych grafów, program wyposażony jest w zestaw narzędzi umożliwiających interaktywną edycję grafów:

- tworzenie/usuwanie/przesuwanie wierzchołków,
- tworzenie/usuwanie krawędzi,
- modyfikację koloru i wielkości wierzchołków,
- usuwanie całego grafu,
- usuwanie podgrafu grafu niespójnego,
- generowanie losowych grafów,
- losowy układ wierzchołków,
- zmianę wielkości ramki do rysowania grafów (z funkcją automatycznego dopasowania do wielkości okna).

Program pracuje w trybie MDI (Multi Document Interface) – interfejs użytkownika składa się z głównego okna aplikacji oraz okien-dokumentów. Dzięki temu można pracować niezależnie na kilku dokumentach i porównywać efekty działania różnych algorytmów lub różne grafy.

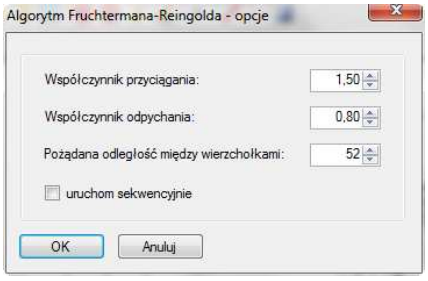
Aplikacja umożliwia zapis grafu do pliku we własnym formacie i późniejszy odczyt. Ponadto zaimplementowano zapis całej ramki z grafem do pliku w formacie PNG (Portable Networks Graphics).

5 Implementacja algorytmów i środowisko programowania

Program został napisany w języku C#, przy użyciu środowiska Microsoft Visual Studio 2008 Professional Edition. C# jest nowoczesnym, obiektowym językiem programowania, zapewniającym automatyczne zarządzanie pamięcią, udostępniającym wiele gotowych szablonów i klas do wykorzystania. Microsoft Visual Studio to bardzo rozbudowane środowisko tworzenia oprogramowania, wspierające pisanie w językach .NET, ale również w C++ i C++/CLI. Zapewnia ono wsparcie programisty poprzez sugerowanie składni (w tym własnych klas i ich składowych), dokańczanie słów kluczowych i nazw klas, typów, zmiennych, metod. Środowisko dysponuje zaawansowanym debuggerem, pozwalającym na śledzenie kodu krok po kroku, zastawianie pułapek, podgląd wartości zmiennych. Zestaw typów projektów pozwala na przyspieszenie pracy poprzez stworzenie szablonu projektu (np. aplikacja konsolowa, okienkowa, internetowa, usługa sieciowa, biblioteka DLL i in.).

6 Implementacja algorytmu Fruchtermana-Reingolda

Po wciśnięciu przycisku lub wybraniu odpowiedniej pozycji z menu, wyświetla się formularz służący do ustawienia parametrów działania algorytmu. Zawiera on 3 liczby, określające odpowiednio: współczynnik przyciągania, współczynnik odpychania i pożądaną odległość między wierzchołkami. Wartości te są wstępnie obliczone na podstawie parametrów grafu i wielkości ramki. Można je modyfikować w celu wzmocnienia lub osłabienia oddziaływania sił na wierzchołki. Dodatkowo na formularzu widnieje przycisk opcji „uruchom sekwencyjnie”, którego zaznaczenie spowoduje krokowe działanie algorytmu, z kilkudziesięciomilisekundowymi przerwami między poszczególnymi iteracjami. Zyskujemy dzięki temu efekt animacji, ukazujący kolejne położenia wierzchołków. Okno ustawień pokazano na rysunku 13.

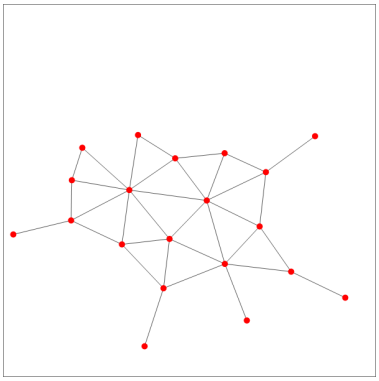


Rys. 13. Okno opcji algorytmu Fruchtermana-Reingolda

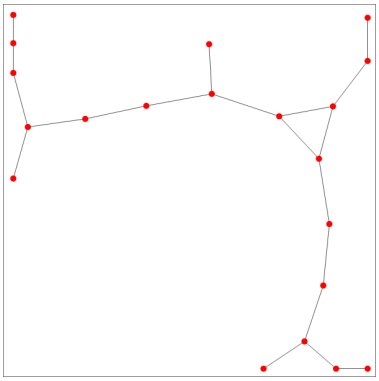
Po zatwierdzeniu ustawień przyciskiem OK, aplikacja tworzy instancję klasy FruchtermanReingold i wywołuje metodę layoutGraph z odpowiednimi argumentami.

Szereg doświadczeń przeprowadzonych z użyciem aplikacji Graph Visualizer pozwolił dobrać optymalne wartości współczynników odpychania i przyciągania oraz pożądanej odległości pomiędzy wierzchołkami. Nie są to jednak wartości uniwersalne. W zależności od struktury grafu należy je modyfikować. W praktyce wystarcza zmiana tego ostatniego współczynnika (pożądanego odstępu).

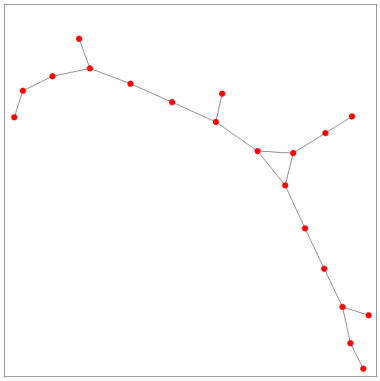
Efekt działania algorytmu na grafach o tej samej liczbie wierzchołków (20), ale o różnej gęstości, z zastosowaniem domyślnych ustawień pożądanego odstępu, obrazują rysunki 14, 15 i 16.



Rys. 14. Graf A o relatywnie dużej gęstości, ułożony algorytmem Fruchtermana-Reingolda z domyślnymi ustawieniami

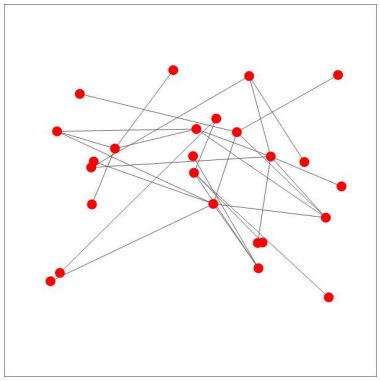


Rys. 15. Graf B (20 wierzchołków) o relatywnie małej gęstości, ułożony algorytmem Fruchtermana-Reingolda z domyślnymi ustawieniami

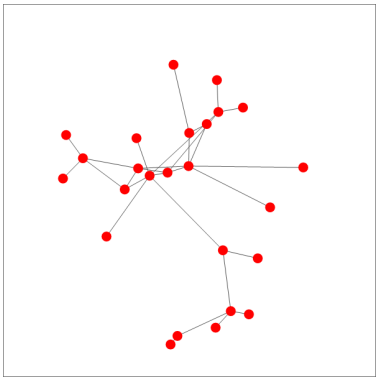


Rys. 16. Graf B ułożony algorytmem Fruchtermana-Reingolda ze zmienionymi ustawieniami

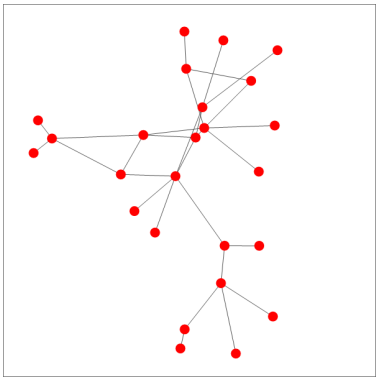
Wyraźnie widać, że skrajne wierzchołki grafu nierównomiernego, o dużo mniejszej gęstości, próbują wyjść poza obszar rysowania. Wynika to ze wzoru na obliczenie pożądanej odległości, który zakłada równomierny rozkład wierzchołków na powierzchni, czyli ułożenie wierzchołków jak na szachownicy. Oczywistym rozwiązaniem jest zmniejszenie parametru k – pożądanego odstępu. Na rysunku 16 widać zdecydowaną poprawę osiągniętą dzięki zmianie parametru k.



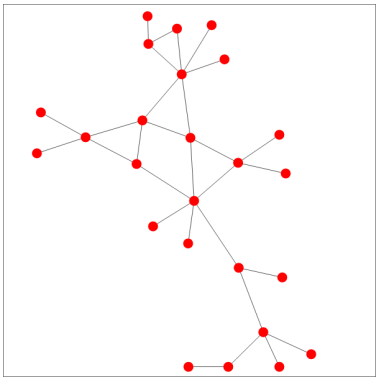
Rys. 17. Sekwencyjne działanie algorytmu Fruchtermana-Reingolda. Stan początkowy



Rys. 18. Sekwencyjne działanie algorytmu Fruchtermana-Reingolda. Iteracja 20

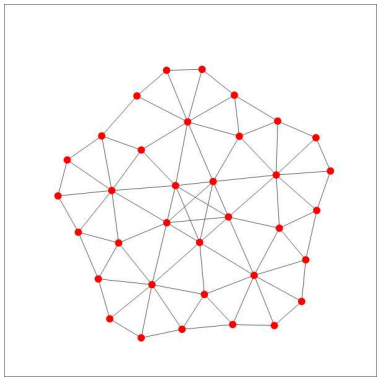


Rys. 19. Sekwencyjne działanie algorytmu Fruchtermana-Reingolda. Iteracja 40

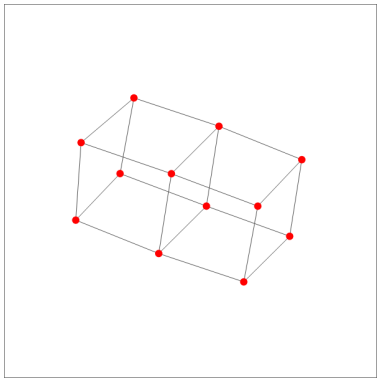


Rys. 20. Sekwencyjne działanie algorytmu Fruchtermana-Reingolda. Iteracja 124 - stan końcowy

Rysunki 17 - 20 przedstawiają stan (wygląd) losowego grafu o 25 wierzchołkach po wykonaniu wybranych iteracji. Przedstawienie wszystkich kroków zajęłoby zbyt dużo miejsca, ponieważ algorytm wykonał 124 iteracje. Wybrane sekwencje dobrze ilustrują przebieg algorytmu.



Rys. 21. Rezultat algorytmu Fruchtermana-Reingolda



Rys. 22. Rezultat algorytmu Fruchtermana-Reingolda

Pewne grafy dobrze poddają się algorytmowi Fruchtermana-Reingolda, który daje bardzo dobre rezultaty. W wypadku innych grafów rezultat może być dużo mniej efektowny i czytelny. Przykładowe wizualizacje różnych grafów o szczególnych walorach estetycznych (równomierny rozkład wierzchołków, zachowanie symetrii), ułożonych omawianym algorytmem, pokazano na rysunkach 21 - 22. Warto zwrócić uwagę na dodatkową, niezamierzoną cechę algorytmu Fruchtermana-Reingolda: niektóre grafy przedstawiane są trójwymiarowo .

Przeprowadzono testy wydajności algorytmu na losowych grafach o różnej liczbie wierzchołków i krawędzi. Testy składały się z 10 iteracji kolejno dla grafów o 50, 100, 500 i 1 000 wierzchołków. Przed każdą iteracją losowany był nowy graf. Liczba krawędzi nie jest ustalana deterministycznie i zależy wyłącznie od przebiegu algorytmu losującego. Czas trwania każdej iteracji oraz czasy minimalne, średnie i maksymalne w każdej grupie przedstawiono w tabeli 1. Pokazano również wygenerowaną w poszczególnych iteracjach liczbę krawędzi grafu.

Słabą stroną algorytmu Fruchtermana-Reingolda jest jego duża złożoność obliczeniowa. Zależy ona kwadratowo od liczby wierzchołków i liniowo od liczby krawędzi. Otrzymane wyniki potwierdzają tę cechę oraz tezę, że przydatność algorytmu ogranicza się jedynie do grafów relatywnie małych (o niewielkiej liczbie wierzchołków).

Tabela. 1. Testy wydajności algorytmu Fruchtermana-Reingolda

Lp.	**50 wierzchołków**		**100 wierzchołków**		**500 wierzchołków**		**1 000 wierzchołków**	
	liczba krawędzi	czas [ms]	liczba krawędzi	czas [ms]	liczba krawędzi	czas [ms]	liczba krawędzi	czas [ms]
1	71	46	155	140	819	2 839	1 591	11 341
2	79	32	153	125	766	2 839	1 539	11 356
3	80	31	150	125	779	2 886	1 617	11 294
4	72	31	152	109	762	2 854	1 567	11 310
5	75	31	156	109	769	2 839	1 555	11 279
6	78	31	137	109	789	2 839	1 555	11 342
7	73	32	160	125	770	2 855	1 574	11 310
8	73	31	154	125	791	2 840	1 558	11 310
9	73	31	158	109	766	2 839	1 548	11 357
10	66	31	153	109	794	2 840	1 568	11 310
Min	**66**	**31**	**137**	**109**	**762**	**2 839**	**1 539**	**11 279**
Avg	**74**	**32,7**	**152,8**	**118,5**	**780,5**	**2 847**	**1 567,2**	**11 320,9**
Max	**80**	**46**	**160**	**140**	**819**	**2 886**	**1 617**	**11 357**

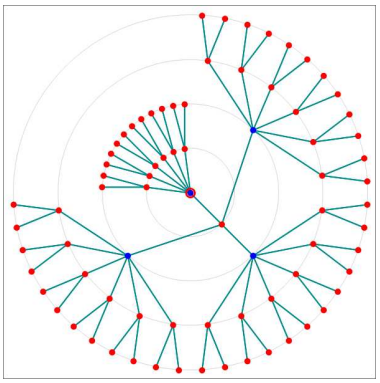
7 Implementacja algorytmu radialnego

Uruchomienie wizualizacji grafu w układzie radialnym rozpoczyna się od utworzenia instancji klasy RadialView i wywołaniu metody layoutGraph z odpowiednimi argumentami. Ważnym argumentem z punktu widzenia rezultatu algorytmu jest wierzchołek, który stanie się korzeniem drzewa i zostanie wyświetlony w centrum układu. Domyślnie korzeniem staje się wierzchołek o największym stopniu (tzw. lider). Aplikacja przed uruchomieniem metody layoutGraph przeszukuje graf i ustala lidera.

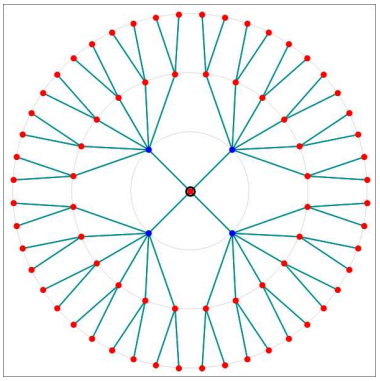
Warto zaznaczyć dodatkową cechę zaimplementowanego rozwiązania. W czasie wyświetlania układu radialnego kliknięcie dowolnego

wierzchołka powoduje jego ustawienie jako nowy korzeń drzewa i przebudowę widoku.

Implementacja domyślnie ustawia wierzchołek o największym stopniu jako korzeń drzewa. Nie zawsze jest to rozwiązanie optymalne. Przykładowy graf, który prezentuje się po zmianie wierzchołka centralnego przedstawiają rysunki 23 (ustawienie domyślne – korzeniem jest lider) i 24 (ręczny wybór korzenia - układ bardziej wyważony).

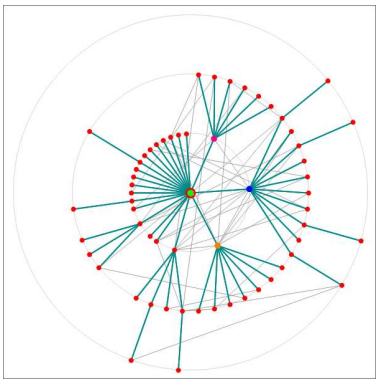


Rys. 23. Rezultat działania algorytmu radialnego - domyślny wybór korzenia

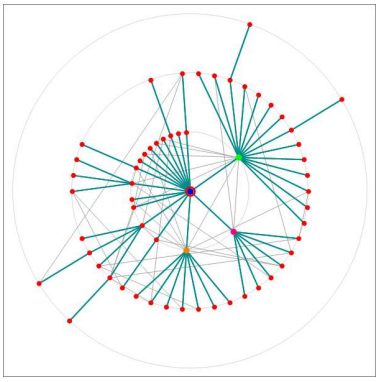


Rys. 24. Rezultat działania algorytmu radialnego - ręczny wybór korzenia

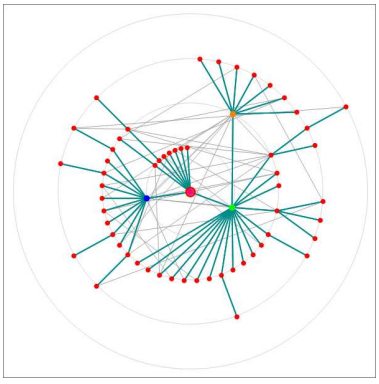
Na rysunkach 24, 25, 26 i 27 zaprezentowano losowy graf złożony z 60 wierzchołków w układzie radialnym, z różnymi korzeniami. Aby umożliwić prześledzenie zmian, kolejne wierzchołki centralne mają inne kolory od pozostałych.



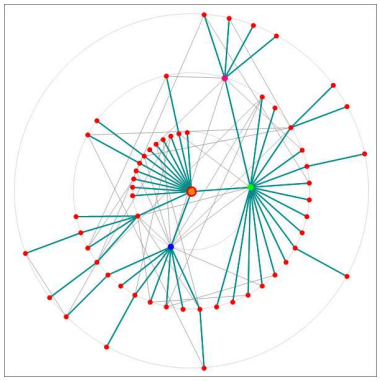
Rys. 25. Graf 60-wierzchołkowy w układzie radialnym – domyślnie



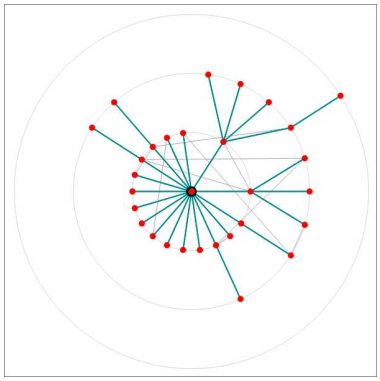
Rys. 26. Graf 60-wierzchołkowy w układzie radialnym – ręczny wybór korzenia



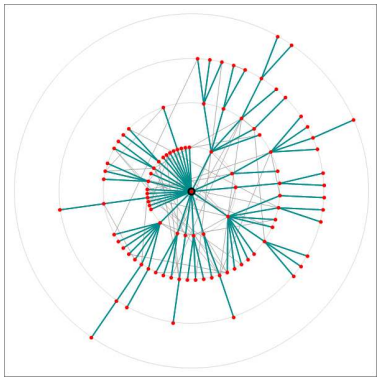
Rys. 27. Graf 60-wierzchołkowy w układzie radialnym – ręczny wybór korzenia



Rys. 28. Graf 60-wierzchołkowy w układzie radialnym – ręczny wybór korzenia



Rys. 29. Losowy graf 30-wierzchołkowy w układzie radialnym



Rys. 30. Losowy graf 100-wierzchołkowy w układzie radialnym

Jakość i skuteczność algorytmu dla grafów różnych wielkości ilustrują rysunki 28 i 29. Dla grafów bardzo dużych celowe jest oczywiście albo zwiększenie powierzchni prezentacji, albo zmniejszenie wielkości wierzchołków, w celu wyeliminowania nakładania się wierzchołków na siebie.

Podobnie jak w wypadku implementacji Fruchtermana-Reingolda, procedura układu radialnego również poddana została testom wydajności. Ze względu na niemierzalność rezultatów przy grafach mniejszych niż 1 000 wierzchołków (uzyskane wyniki w milisekundach często wynosiły 0, zmierzenie czasów z większą dokładnością było niemożliwe), do testów wykorzystano grafy o liczbie wierzchołków odpowiednio 1 000, 2 000 i 5 000. Pozostałe kryteria nie zmieniły się. Wykonano 10 iteracji dla każdej z wymienionych wielkości grafów. W każdej iteracji losowany był nowy graf. Na czas testów efekt animacji, który sztucznie wydłużał czas trwania operacji (płynne przemieszczanie) został wyłączony. Wyniki przedstawia tabela 2.

Tabela. 2. Testy wydajności algorytmu radialnego

Lp.	**1 000 wierzchołków**		**2 000 wierzchołków**		**5 000 wierzchołków**	
	liczba krawędzi	czas [ms]	liczba krawędzi	czas [ms]	liczba krawędzi	czas [ms]
1	1 581	32	3 199	156	7 798	921
2	1 551	47	3 202	140	7 880	921
3	1 534	31	3 144	140	7 907	936
4	1 542	47	3 213	156	7 938	936
5	1 563	31	3 221	140	7 837	920
6	1 560	46	3 122	156	7 835	920
7	1 569	31	3 206	141	7 853	921
8	1 587	31	3 149	141	7 978	936
9	1 609	31	3 138	156	7 874	936
10	1 518	47	3 130	140	7 986	936
Min	**1 518**	**31**	**3 122**	**140**	**7 798**	**920**
Avg	**1 561,4**	**37,4**	**3 172,4**	**146,6**	**7 888,6**	**928,3**
Max	**1 609**	**47**	**3 221**	**156**	**7 986**	**936**

W porównaniu z algorytmem Fruchtermana-Reingolda, algorytm radialny działa w czasie liniowym w funkcji liczby wierzchołków i krawędzi. Zdecydowanie lepiej nadaje się do prezentacji grafów o dużej liczbie wierzchołków, również w zastosowaniach interaktywnych. Średni czas ułożenia grafu zawierającego 1 000 wierzchołków jest tego samego rzędu co czas ułożenia algorytmem Fruchtermana-Reingolda grafu o 50 wierzchołkach. Jest to różnica kolosalna. Porównywalnej wielkości grafy (1 000 wierzchołków) były przetwarzane przez każdy z algorytmów w średnim czasie odpowiednio 11,32 s (algorytm Fruchtermana-

Reingolda) i 37,4 s (algorytm radialny). Ten ostatni jest zatem ~300 razy szybszy dla tej wielkości grafu. Oczywiście porównanie to będzie coraz korzystniejsze dla algorytmu radialnego wraz ze wzrostem wielkości grafu, ze względu na kwadratową złożoność algorytmu Fruchtermana-Reingolda.

8 Podsumowanie

Wiele zjawisk występujących we współczesnym świecie jest opisywane przez sieci. Mogą to być nie tylko sieci fizyczne – drogowa, energetyczna, komputerowe ale również w mniejszym lub większym stopniu wirtualne – znajomi w portalu społecznościowym, połączenia email. Większość z nich generuje bardzo złożone struktury, których analiza wymaga w pierwszym kroku ich wizualizację, w takiej postaci aby była ona czytelna dla operatora. Stąd bardzo istotnym elementem staje się narzędzie do wizualizacji, a w przypadku mało czytelnej postaci wyjściowej do jej reorganizacji poprawiającej jakość prezentacji.

W pracy przedstawiona została metoda tworzenia takiego narzędzia z zastosowaniem języka wyższego rzędu realizującego dwa algorytmy reorganizacji sieci: Fruchtermana-Reingolda oraz radialny. Omówiony został zestaw cech obu algorytmów predestynujący je do grafów o określonych właściwościach i poziomie komplikacji. Pokazane zostały również wyniki testów wydajności świadczące o dużej efektywności oprogramowanych algorytmów, ze wskazaniem na algorytm radialny w przypadku bardziej złożonych grafów.

W przypadku grafów, w których nie ma możliwości określenia położenia węzłów, mechanizm reorganizacji może być wykorzystany jako pierwszy krok w procesie ich dalszej analizy. Przypisanie wynikających z reorganizacji współrzędnych może być wstępem do przeprowadzenia grupowania węzłów z zastosowaniem metod klasteryzacji. Również w przypadku grafów o dobrze określonych danych dotyczących położenia węzłów, reorganizacja i wynikające z niej alternatywne współrzędne mogą być użyteczne w analizie grafów stanowiąc materiał porównawczy dla ich domyślnej organizacji.

9 Literatura

[1] Thomas M. J. Fruchterman, Edward M. Reingold, *Graph drawing by force-directed placement.* Software—Practice And Experience. 1991, Vol. 21.

[2] J. S.l. Graham, *NicheWorks-Interactive Visualization of Very Large Graphs,* Wills, Journal of Computational and Graphical Statistics, 1997, Vol. 8.

[3] P. Pilny A. Pelikant, *Typy użytkownika CLR - wprowadzenie obiektowości do relacyjnej bazy danych*. Zeszyty Naukowe Wyższej Szkoły Informatyki w Łodzi. 2012, Tom 11, 3.

[4] K. Kuratowski, *Sur le problème des courbes gquches en Topologie*. Fundamenta Mathematicae. 1930, 15.

[5] P. Eades, *A Heuristic for Graph Drawing*, 42, 1984.

[6] T. Kamada, S. Kawai, *An algorithm for drawing general undirected graphs*. Information Processing Letters, 1989.

[7] A. Pavlo, Ch. Homan, J. Schull, *A parent-centered radial layout algorithm for interactive graph visualization and animation*. Rochester : RIT Scholar Works, 2006.

[8] E. Konopka, A. Pelikant, *Funkcje i typy użytkownika CLR w zadaniach statystycznych*, Zeszyty Naukowe Wyższej Szkoły Informatyki w Łodzi, Vol. 11, Nr 2, 2012 ss. 5-30

[9] Robin J. Wilson, *Wprowadzenie do teorii grafów*, Wydawnictwo Naukowe PWN, Warszawa, 2007.

[10] A. Fronczak, P. Fronczak, *Świat sieci złożonych: Od fizyki do Internetu*, Wydawnictwo Naukowe PWN, Warszawa, 2009.

[11] D. J. Watts, S.H. Strogatz, *Collective dynamics of "small-world" networks*, Nature, Vol. 393, 440-442, 1998.

[12] A.-L. Barabási, R Albert, *Emergence of scaling in random networks*, Science, Vol. 286, 509-512, 1999.

[13] Z. Tarapaty, *Czy sieci rządzą światem?-Od Eulera do Barabasiego*, WAT, Warszawa, 2012.

[14] A. Hejlsberg, M. Torgersen, S. Wiltamuth, P. Golde, *Język C#. Programowanie*. Wydanie III. Microsoft .NET Development / Helion, 2010.

[15] D. Mendrala, P. Potasiński, M. Szeliga, D. Widera, *Serwer SQL 2008*. Administracja i programowanie / Helion, 2009.

[16] A. Pelikant. *MS SQL Server. Zaawansowane metody programowania*, Helion, 2014

[17] Stephen C. Perry, *C# i .NET* , Helion, 2006.

VISUALIZATION AND REORGANIZATION OF GRAPHS USING RELATIONAL DATABASE

Summary – The purpose of this paper is to discuss methods of visualization and the reorganization of not directed, not weighted, coherent graphs on a plane. Shows two algorithms that are implemented using created software - Graph Visualizer. The application is designed to put vertex of specified graph in the most readable form. As the main criteria assumed the smallest number of intersections and even and symmetrical arrangement of vertices. The software also allows interactively create and modify graph by user. At work shows a comparison of the two algorithms, the differences between them, and they typical applications.

Keywords: Graph theory, graphs reorganization, graphs visualization, social networks, user defined data types CLR, Fruchterman- Reingold algorithm, radial algorithm, object-oriented programming in databases.