Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления (ИУ)

КАФЕДРА Теоретическая информатика и компьютерные технологии (ИУ-9)

Радиальное авторазмещение графа

Тришин Алексей

Группа ИУ9-51

Руководитель

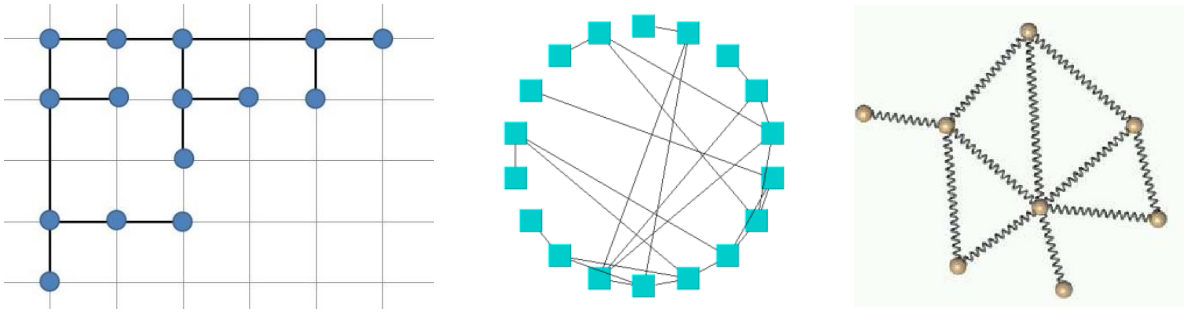
Вишняков И. Э.

Москва, 2017 г.

Оглавление

Введение

Существует множество способов визуализации графа, одни укладывают граф в виде hv-изображения на сетке, но получаются множественные изгибы рёбер (см. рисунок 1, а)), вторые располагают граф на окружности (см. рисунок 1, б)), однако возникает большое число пересечений рёбер, третьи используют пружинную модель, которая занимает много места на плоскости (см. рисунок 1, в)).



а) б) в)

Рисунок 1 – укладка графа в виде hv-изображения (а), на окружности (б), с помощью пружинной модели (в).

Визуализация графа помогает сделать понятной структуру графа. Однако множественные пересечения рёбер, их изгибы и неравномерная длина, наложение вершин друг на друга и асимметричность – всё это препятствует приемлемому восприятию человеком графа. Радиальная конфигурация отличается от других наличием нескольких окружностей, на которые размещаются вершины графа по определённому правилу. Данный способ наилучшим образом решает описанные проблемы.

Сразу стоит отметить тот факт, что визуализация с помощью радиального размещения предназначена, в первую очередь, для деревьев, так как их свойства наиболее хорошо подходят для представления данным способом. Хотя полученное дерево из графа (например, обходом в глубину или ширину) может и утратить информацию о рёбрах, тем не менее, после применения алгоритма получается визуально более понятное изображение графа, а удалённые рёбра можно восстановить.

Основным преимуществом данного подхода является наиболее приближённое соответствие евклидового расстояния между вершинами и их графо-теоретического расстояния, что, по факту, является одним из важнейших критериев «читаемости» схемы графа. Также радиальное размещение удовлетворяет другому, но не менее важному критерию, а именно равномерности заполняемости пространства. Все вершины наиболее равномерно располагаются на плоскости так, чтобы они не накладывались друг на друга и при этом расстояние между ними не было необоснованно велико.

Целью работы является программная реализация алгоритмов радиального авторазмещения графа с последующим сравнением их результатов между собой.

Глава 1. Обзор алгоритмов

1.1 Алгоритм фокусирования на вершине.

Основой данного алгоритма является метод многомерного шкалирования (МНШ) [1], цель которого – максимально эффективно разместить объекты, максимально сохраняя евклидово расстояние между ними. МНШ использует алгоритм минимизации некоторой функции, оценивающей качество получаемых вариантов отображения. Чаще всего используется функция стресса, которая в нашем случае будет выглядеть следующим образом:

$$\sigma(p) = \sum_{u,v} w_{uv}(d_{uv} - \|p(u) - p(v)\|)^2,$$

где, d_{uv} – суммарная длина рёбер на кратчайшем пути между вершинами u и v, а w_{uv} – это вес для вклада условия ошибки: $(d_{uv} - \|p(u) - p(v)\|)^2$, связанного с парой u, v.

Соответственно, целью использования данного метода является минимизация функции стресса $\sigma(p)$, чтобы найти позиции вершин $p(v_i)$, максимально удовлетворяющие условию:

$$\|p(u) - p(v)\| \approx d_{uv}\ \forall u, v\ \in V.$$

Существует общепринятый факт, что конфигурации с небольшим значением функции стресса обычно структурно более информативны и эстетически приятны [11]. Причина положительных эстетических свойств низко нагруженного расположения заключается в том, что ни одной из вершин не отдается предпочтение. Существует способ уменьшения этого числа. Начиная с начальной конфигурации, генерируется улучшенная последовательность расположений. Причём данный процесс может быть запущен из любого начального расположения. В процессе уменьшения нагрузки новые позиции $\hat{p}(u) = (\hat{x}_u, \hat{y}_u)$ каждой вершины $u\ \in V$ могут быть вычислены из текущих позиций с помощью формул:

$$\hat{x}_u \leftarrow \frac{\sum_{v \neq u} w_{uv}\ (x_v + d_{uv}\ (x_u - x_v)\ b_{uv})}{\sum_{v \neq u} w_{uv}} \quad (1)$$

$$\hat{y}_u \leftarrow \frac{\sum_{v \neq u} w_{uv} \left(y_v + d_{uv} \left(y_u - y_v\right) b_{uv}\right)}{\sum_{v \neq u} w_{uv}} \quad (2),$$

где $b_{uv} = \begin{cases} \frac{1}{\|p(u) - p(v)\|}, & \text{если } \|p(u) - p(v)\| > 0 \\ 0, & \text{иначе} \end{cases}$.

Данные действия повторяются, пока не уменьшится нагрузка до определенного числа, или пока не пройдет определенное число шагов, или пока конфигурация не удовлетворит другому выбранному критерию. Легко показать, что нагрузка сгенерированных таким образом расположений не возрастает и стремится к локальному минимуму [10].

Если учитывать расстояние относительно только какой-то одной вершины, то данный алгоритм расположит граф радиально. Алгоритм, основанный на МНШ, можно применять как к дереву, так в общем случае и к графу. Явным преимуществом такого алгоритма является максимальное приближение графо-теоретического расстояния и евклидового, а также тот факт, что все рёбра приблизительно равны между собой. Однако данный способ требует изначального расположения всех вершин графа, что усложняет его использование. Также значительным недостатком является количество итераций, чтобы получить достаточно подходящее изображение. Очевидно, что с увеличением количества вершин увеличивается число необходимых итераций.

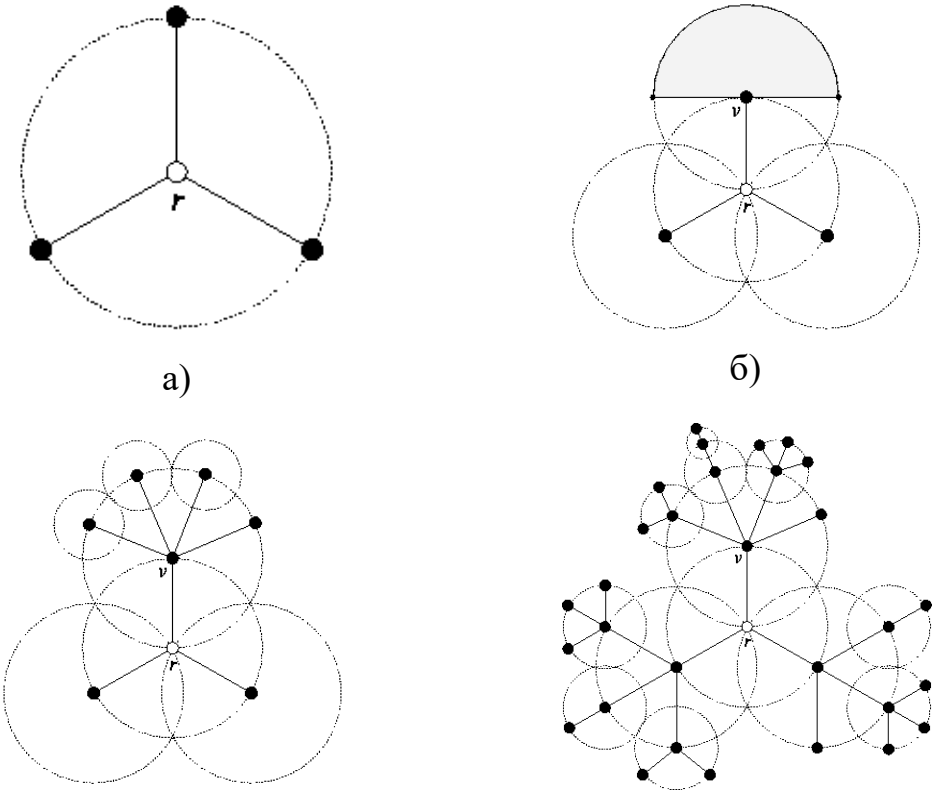
1.2 Алгоритм родителе-ориентированной модели.

Данный алгоритм [4] применяется к дереву. В нём используется родителе-ориентированная модель представления графа. Основная суть заключается в том, что вершины с одним и тем же родителем лежат в одной системе координат, верно и обратное – вершины с разными родителями лежат в разных системах координат.

Определяется родителе-ориентированная модель графа следующим образом. Для любой вершины v дерева T полярные координаты v даны в системе координат, которая определяется следующим образом:

- Если v – корень дерева T: начало координат совпадает с началом координат плоскости, луч нулевого градуса совпадает с положительным направлением оси OX;
- Если v – дочерняя вершина корня дерева T: корень дерева является началом координат, а луч нулевого градуса – лучом из корня, имеющим то же направление, что и положительное направление оси OX;
- Иначе: родитель вершины – начало координат, а луч нулевого градуса проходит через предка родителя данной вершины.

Алгоритм работает следующим образом, дано дерево T, сначала корневая вершина r дерева T помещается в центр плоскости, а её дети $v_1, \dots, v_n$ равномерно размещаются на окружности с центром в r (см. рисунок 2, а)). Затем на окружности с центром в v_i определяется дуга, на которой могут лежать дети v_i (см. рисунок 2, б)). Второй шаг повторяется рекурсивно до тех пор, пока все вершины не будут расположены на плоскости (см. рисунок 2, в), г)).



а) б)

в) г)

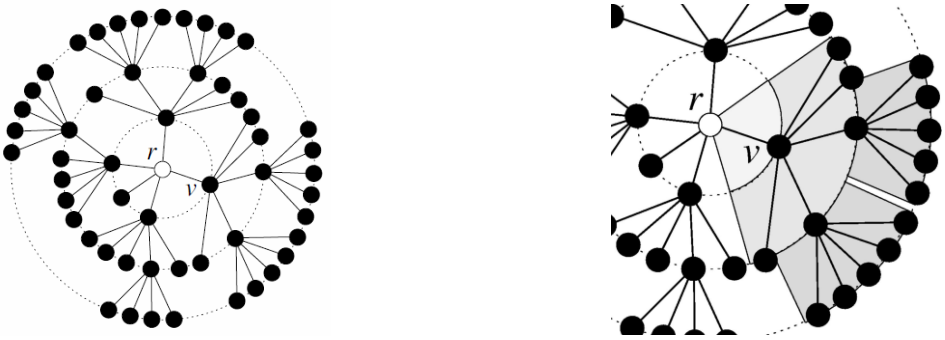
Рисунок 2 – процесс расположения вершин.

Преимуществом такого алгоритма является то, что все вершины одного родителя равноудалены от своего него, что облегчает восприятие структуры дерева и отлично подходит для отображения иерархии. Применяемая здесь модель данных уменьшает возможное количество пересечений вершин (однако не исключает их полностью), что также делает изображение более понятным.

Явным недостатком является постепенно уменьшающиеся длины рёбер от поколения к поколению, что с одной стороны отражает графо-теоретическое расстояние до корня дерева (чем меньше ребро, тем дальше вершина находится от корня), но с другой – создает возможности для наложения вершин.

1.3 Простой алгоритм радиального размещения.

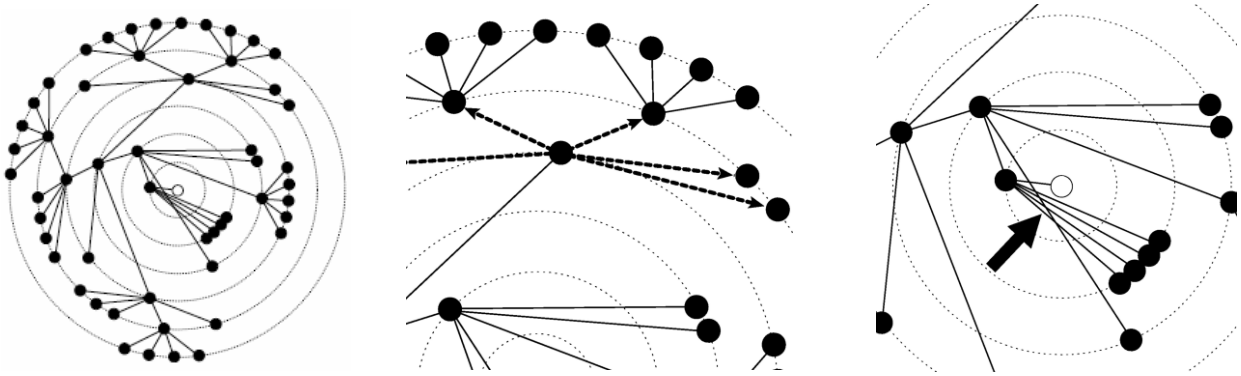
Данный алгоритм [6] может применяться для любого дерева. Также, как и в предыдущем алгоритме, корень дерева r помещается в центр плоскости. Затем пространство вокруг корневой вершины делится среди её потомков: размер сектора кольца для потомка c вершины v зависит от количества листовых вершин в поддереве с корнем в c, которое пропорционально числу листовых вершин в поддереве с корнем в вершине v. Каждая вершина помещается в центр её кольцевого сектора в соответствии с её глубиной в дереве (см. рисунок 3 а), б)). Алгоритм продолжается до тех пор, пока не будут вычислены позиции каждой вершины.



а) б)

Рисунок 3 – радиальное расположение дерева с корнем r (а) и разделение уровней кольцевого сектора v (б).

Очевидным достоинством такого подхода является наглядность определения значения глубины вершин в дереве, однако, кольцевые сектора могут ограничивать расположение вершин так, что их расположение может быть не оптимально с точки зрения визуального определения родителя определённых вершин, так как длины рёбер между родителем и его детьми могут быть разными (см. рисунок 4 а), б)). Также данное представление полностью не исключает пересечение рёбер (см. рисунок 4 а), в)), что также является важным критерием при выборе алгоритма.



а) б) в)

Рисунок 4 – проблемы простого алгоритма радиального размещения, результат – (а), разная длина рёбер (б), пересечение рёбер (в)

2. Разработка алгоритмов

Пусть $G = (V, E)$ – простой неориентированный граф. Обозначим мощность набора вершин и ребер $n = |V|$ и проиндексируем вершины числами: $V = \{v_1, \ldots, v_n\}$. Каждая вершина имеет поле для подписи. Вершины и их подписи могут быть разного размера. На вход алгоритмам даётся граф G, на выходе должно получиться изображение Г, где каждой вершине v_i взаимно и однозначно сопоставляется её позиция на плоскости $p(v_i) = (x_{v_i}, y_{v_i})$, причём необходимо минимизировать пересечения вершин и их подписей.

На вход программа должна получать:

- количество вершин, подписи для каждой вершины (по требованию), рёбра, в виде пары вершин, которые должны соединяться ребром. (В качестве файла)
- номер алгоритма визуализации.

После работы каждого из алгоритмов каждая вершина будет иметь свои координаты на плоскости.

Также после основной части алгоритмов родителе-ориентированной модели и простого радиального размещения необходимо удалить пересечения вершин, увеличив радиус, на котором они находятся. Так как эти алгоритмы работают с деревом, полученным из графа, то необходимо добавить возможность запоминания удалённых связей между вершинами, чтобы впоследствии их можно было восстановить и по требованию вывести на экран.

В ходе разработки были спроектированы следующие структуры данных:

- Вершина. Хранит в себе данные о координатах (в декартовой и полярной системах координат), размере, глубине в дереве, родителе, своих дочерних вершинах, является ли она корнем, а также свою подпись.
- Подпись вершины. В данной структуре данных находится информация о позиции на плоскости и размере самой подписи.

- Матрица. Необходима для алгоритма фокусирования на вершине.
- Граф. Хранит в себе массив вершин, корень дерева, радиусы, на которых лежат вершины, массив вершин по глубине в дереве, а также свой радиус и центр.

Также были разработаны следующие модули:

- Поиск в ширину. Вспомогательный модуль, который запускает обход в ширину от вершины и запоминает расстояния до пройденных вершин от стартовой.
- Визуализация. Содержит в себе граф и текущий номер алгоритма визуализации. Отвечает за вывод графа на экран.
- Алгоритмы. Содержит в себе сами алгоритмы.

Приложение должно работать по следующей схеме:

1. Инициализация приложения. На данном этапе считываются входные данные, инициализируется граф.
2. Размещение. Получение из графа дерева обходом в ширину из центра графа (или из выбранной пользователем вершины), затем вычисление позиций вершин графа на плоскости в зависимости от алгоритма и перевод координат в систему координат окна.
3. Визуализация. Инициализация окна приложения, а также инициализация обработчиков нажатия клавиш для перерисовки графа с использованием другого алгоритма или с указанием нового корня дерева.

Для реализации любого алгоритма необходимы функции заполнения графа, получения из него дерева (обходом в глубину или ширину), подсчёта его максимальной глубины, нахождения центра и радиуса графа и функции отрисовки/перерисовки графа и преобразования координат вершин на плоскости в координаты окна. Также для вершин нужны функции, задающие их с помощью декартовых и полярных координат.

2.1 Разработка алгоритма фокусирования на вершине.

Для визуализации графа используют следующий весовой коэффициент: $w_{uv} = d_{uv}^{-2}$. Так малые расстояния учитываются лучше, чем большие. Функцию стресса $\sigma(p)$ мы будем считать функцией стресса расстояний и переобозначим ее как $\sigma_W(p)$. Нижний индекс обозначает, что стресс определяется как матрица $W = (w_{uv})_{uv} \in \mathbb{R}^{n\times n}$. Данная модель расширяется вторым набором весов $Z = (z_{uv})_{uv} \in \mathbb{R}^{n\times n}$, который используется для функции стресса ограничений, определённой как:

$$\sigma_Z(p) = \sum_{u,v} z_{uv}(d_{uv} - \|p(u) - p(v)\|)^2.$$

Теперь, наша задача сводится к минимизации обеих функций, будем уменьшать значения стресса расстояний и ограничений одновременно. Вначале вершины могут перемещаться свободно без рассмотренных ограничений, минимизируя просто $\sigma_W(p)$. Затем ограничения предоставляют всё больше и больше контроля расположения, динамически изменяя коэффициенты в данной комбинации, учитывая на каждой итерации сначала один критерий, а затем другой. Результирующую функцию стресса можно записать следующим образом:

$$\sigma_{(1-t)\cdot W + t\cdot Z} = (1-t)\cdot\sigma_W(p) + t\cdot\sigma_Z(p),$$

где $t = 0, \frac{1}{k}, \frac{2}{k}, \dots, \frac{k-1}{k}, 1$; k – количество итераций, определяемое пользователем.

Легко включить данную функцию в минимизацию стресса изменяя правила (1) и (2) на

$$\hat{x}_u \leftarrow \frac{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})\cdot(x_v + d_{uv}\,(x_u - x_v)\,b_{uv})}{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})},$$

$$\hat{y}_u \leftarrow \frac{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})\cdot(y_v + d_{uv}\,(y_u - y_v)\,b_{uv})}{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})}.$$

После того, как описан общий подход, применяемый в данном алгоритме, можно непосредственно перейти к конкретно радиальному

размещению графа. Идея заключается в том, что k-ому уровню дерева соответствует k-ая окружность.

Чтобы применить эту идею на практике матрица весовых ограничений учитывает те пары вершин, с которыми соединена ребром корневая вершина, скажем v_i, веса всех остальных вершин обращаются в нуль. Имеется матрица $D = (d_{uv})_{uv} \in \mathbb{R}^{n\times n}$ сумм длин рёбер, которые лежат на кратчайшем пути между вершинами u и v. Также есть матрица $W = (w_{uv})_{uv} \in \mathbb{R}^{n\times n}$, где $w_{uv} = {d_{uv}}^{-2}$, а матрица весовых ограничений $Z = (z_{uv})_{uv}$ имеет ненулевые элементы только на i-ой строке и столбце:

$$Z = \begin{bmatrix} 0 & \cdots & 0 & w_{v_1v_i} & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & \vdots & 0 & \ddots & \vdots \\ 0 & \dots & 0 & w_{v_{i-1}v_i} & 0 & \cdots & 0 \\ w_{v_iv_1} & \cdots & w_{v_iv_{i-1}} & 0 & w_{v_iv_{i+1}} & \cdots & w_{v_iv_n} \\ 0 & \cdots & 0 & w_{v_{i+1}v_i} & 0 & \cdots & 0 \\ \vdots & \ddots & 0 & \vdots & 0 & \ddots & \vdots \\ 0 & \cdots & 0 & w_{v_nv_i} & 0 & \cdots & 0 \end{bmatrix}.$$

В данной работе инициализировать начальное положение вершин будем с помощью других двух алгоритмов. Итоговым преобразованием для получения результата являются формулы:

$$\hat{x}_u \leftarrow \frac{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})\cdot (x_v + d_{uv}\,(x_u - x_v)\,b_{uv}}{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})},$$

$$\hat{y}_u \leftarrow \frac{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})\cdot (y_v + d_{uv}\,(y_u - y_v)\,b_{uv}}{\sum_{v\neq u}((1-t)\cdot w_{uv} + t\cdot z_{uv})}.$$

2.2 Разработка алгоритма родителе-ориентированной модели.

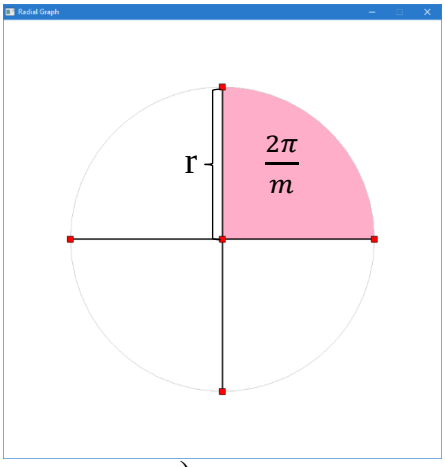
Алгоритм определения координат вершин формально описывается следующим образом. Дано дерево T, корневая вершина имеет координаты $(0,0)$, для каждой вершины v пусть $v_1, \dots, v_m$ – дочерние вершины v. $\forall i \in \{1, \dots, m\}$ координаты вершины v_i (даны в системе координат родителе-ориентированной модели (см. рисунок 5)):

- Если v – корень дерева T: $(\frac{2\pi i}{m}, r)$, $r > 0$ – предопределённое пользователем значение (см. рисунок 5, а)).

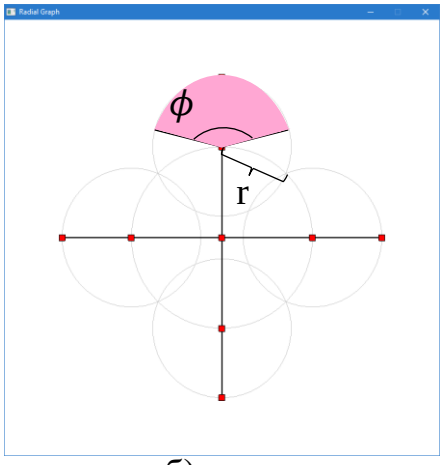
- Иначе: $\left(\pi - \frac{\phi}{2} + \frac{\phi i}{m} + \frac{\phi}{2m}, r\right)$, $\phi -$ преопределённое значение больше нуля (см. рисунок 5, в)), а r – это:
    - Значение координаты r вершины v, если v не имеет сестёр;
    - Иначе, радиус окружности с центром в v, которая пересекает среднюю точку на их общей окружности между v и её ближайшей вершиной на том же уровне, у которой родитель тот же, что и у v.

После вычисления позиций всех вершин необходимо удалить, по возможности, все пересечения вершин или их подписей путём увеличения длины ребра, на котором лежат пересекающиеся вершины.

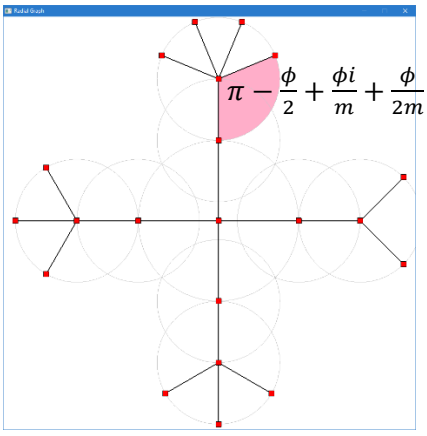
К набору функций, необходимых для реализации данного алгоритма, относятся функция проверки пересечений вершин и их устранения, а также функция поиска ближайшей сестры вершины.



а)



б)



в)

Рисунок 5 – процесс расположения вершин на плоскости.

2.3 Разработка простого алгоритма радиального размещения

Здесь приведём формальный общий алгоритм нахождения позиций вершин для радиального размещения. Дано дерево T, вершины $v \in V(T)$, углы α и β, которые определяют угол кольцевого сектора вершины v. Алгоритм просчитывает позицию каждого ребёнка c вершины v в плоскости Г. $R_0 -$ значение радиуса первой окружности. Первоначальные значения α и β – 0 и 2π соответственно.

Также как в предыдущем алгоритме, после нахождения позиций всех вершин графа нужно минимизировать пересечения вершин и их подписей. Алгоритм запускается от корня дерева и работает следующим образом:

1. Если текущая вершина v – корень дерева, то она помещается в начало координат и угол кольцевого сектора равен 2π.
2. Для каждой дочерней вершины c:
   2.1. Значение угла кольцевого сектора c пропорционально отношению количеству листовых вершин в поддереве с корнем c к количеству листовых вершин с поддеревом с вершиной v. Вершина c помещается в центр ее кольцевого сектора.
   2.2. Если у c есть дочерние вершины, то пункт 2 повторяется для каждой её дочерней вершины.

Набор необходимых функций аналогичен предыдущему случаю за исключением поиска ближайшей вершины на том же уровне того же родителя.

3. Реализация программы радиального авторазмещения

Для реализации поставленной задачи использовался язык программирования Java [9] с установленной библиотекой LWJGL [8] для визуализации графа, дающей возможность для работы с OpenGL и библиотекой GLFW для работы с окном в Java.

Для каждой структуры данных и модуля, описанного в разделе 2, создаётся отдельный класс. Также создаются классы для каждого из алгоритмов, которые являются контейнерами для вычислений. Программа запускается из командной строки: graph.jar {1} {2} -i -s:

{1} - путь к текстовому файлу с графом.

{2} - номер алгоритма:

1. Алгоритм фокусирования на вершине, основанный на алгоритме родителе-ориентированной модели.
2. Алгоритм фокусирования на вершине, основанный на простом алгоритме радиального размещения.
3. Алгоритм родителе-ориентированной модели.
4. Алгоритм простого радиального размещения.

Ключ -i, указывается в случае, если индексирование вершин начинается не с 0, а с 1.

Ключ -s, указывается в случае, если вводятся подписи для вершин.

Формат файла, содержащего граф:

n

m

[sign1]

…

[signN]

i_1 j_1

…

i_m j_m

где n – количество вершин, m – количество рёбер, signI – подпись i-ой вершины (указывается при ключе -s), i_k j_k – ребро, которое соединяет i-ую и j-ую вершины.

Программа работает следующим образом:

1. Создаётся объект класса Graph и заполняется входными данными. Во время заполнения создаются объекты класса Vertex, и вычисляется центр графа.
2. В созданном объекте вызывается метод useAlgorithm(int type), туда передаётся указанный пользователем алгоритм. Внутри метода вызывается статическая функция useAlgorithm(Graph graph) нужного алгоритма, происходит вычисление позиций вершин графа.
3. Затем создаётся объект класса Drawer, который используется для вывода на экран графа путём вызова у данного объекта метода startLoop(), предварительно сконвертировав координаты вершин в оконные. Также внутри этого объекта создаются обработчики нажатий клавиш для взаимодействия с приложением. При нажатии у объекта класса Graph вызывается метод rebuild(int type), если выбран другой алгоритм, или rebuild(double x, double y, int type), если выбрана новая вершина в качестве корневой. После этого выполняются действия пункта 2.

Пользователь может выбирать вершину графа в качестве нового корня дерева, от которого запустится обход в ширину (нажатие мышью на вершину или её подпись). Он может выводить на экран или скрывать вспомогательные окружности (клавиша R), а также рёбра графа, которые были удалены после получения дерева (клавиша D). Также пользователь может изменять алгоритм размещения графа (клавиши 1 – алгоритм фокусирования на вершине, основанный на алгоритме родителе-ориентированной модели, 2 – алгоритм фокусирования на вершине, основанный на простом радиальном размещении, 3 – алгоритм родителе-ориентированной модели, 4 – простой алгоритм радиального размещения).

Размер окна определяется при первом рисовании графа, также в качестве констант задаются минимальный и максимальный размер окна, изначальные размеры вершины и подписи (значения размеров можно менять по необходимости), количество итераций для алгоритма фокусирования на вершине.

4. Тестирование алгоритмов.

Так как основной целью работы является получение радиального размещения графа, то тестирование и сравнение алгоритмов происходило по эстетическим критериям, которые отражают качество изображения графа [12]. В качестве эстетических критериев использовался следующий список [13]:

- Количество пересекающихся рёбер.
- Занимаемое пространство изображением.
- Вариация длины рёбер.

Были проведены исследования [14], которые выявили, что наиболее важный критерий из списка – количество пересекающихся рёбер. Также достаточно сильным критерием является вариация длин рёбер, так как при приблизительно равных рёбрах структура графа становится более понятной.

Параметрами графа, по которым строятся наборы тестов являются:

- Количество вершин.
- Количество рёбер.
- Максимальная степень вершины.
- Средняя степень вершин.

Описание графов, применяемых в тестировании, представлено в таблице 1.

Количество вершин	Количество рёбер	Макс. Степень вершины	Средняя степень вершин
32	98	8	5
62	159	12	5
129	378	8	5
204	535	9	5

Таблица 1. Описание графов, применяемых в тестировании.

Результаты критериев после работы алгоритмов на графах в таблице 2,3,4,5:

Количество пересекающихся рёбер	Мин. длина	Макс. длина	Ширина изображения	Высота изображения
0	97.84	153.41	1132.93	415.5
0	99.99	175.7	1224.44	1388.46
0	79.33	166.59	3278.78	2091.46
0	63.58	180.4	3185.05	5500.47

Таблица 2. Результат для алгоритма фокусирования на вершине на основе алгоритма родителе-ориентированной модели.

Количество пересекающихся рёбер	Мин. длина	Макс. длина	Ширина изображения	Высота изображения
0	98.19	158.8	896.41	931.54
0	99.99	175.4	1523.81	1126.73
0	67.33	181.21	2200.51	3164.57
0	63.87	165.29	4348.51	4668.01

Таблица 3. Результат для алгоритма фокусирования на вершине на основе простого алгоритма радиального размещения.

Количество пересекающихся рёбер	Мин. длина	Макс. длина	Ширина изображения	Высота изображения
3	19633.00	150.0	443.60	344.05
2	10.253	150.00	526.65	395.06
10	11.00	150.00	984.0	625.81
27	11.00	150.00	1394.46	1086.75

Таблица 4. Результат для алгоритма родителе-ориентированной модели.

Количество пересекающихся рёбер	Мин. длина	Макс. длина	Ширина изображения	Высота изображения
0	32.39	153.84	459.79	354,00
6	27.00	252.11	518.18	534.76
25	11.000	429.01	824.19	686.37
37	10.500	835.74	1339.55	1171.53

Таблица 5. Результат для простого алгоритма радиального размещения.

В результате тестирования было выявлено, что наилучший результат по критерию непересекаемости рёбер показал алгоритм фокусирования на вершине. Однако он занял больше всего места на плоскости. Также разница между наибольшим ребром и наименьшим не так велика, по сравнению с другими алгоритмами. Исходя из результатов тестирования можно сказать, что алгоритм фокусирования на вершине наиболее эффективно размещает радиально граф.

Заключение

В ходе данной курсовой работы была разработана и реализована программа для радиального авторазмещения графа. Были изучены и применены три алгоритма для решения поставленной задачи. Также были выявлены их преимущества и недостатки, проведено тестирование и сравнение результатов алгоритмов. Однако не удалось решить проблему с выводом текста для подписей, так стандартные средства OpenGL не позволяют тривиально разрешить такую задачу. Также осталась неразрешённой проблема определения количества итераций в алгоритме фокусирования на вершине, так как для точного результата необходимо найти зависимость между этим количеством и количеством вершин в графе, а также уровнем глубины дерева.

Список литературы

[1] I. Borg and P. Groenen. Modern Multidimensional Scaling. Springer, 2005.

[2] Brandes U. and Pitch Chr. More Flexible Radial Layout. Journal of Graph Algorithms and Applications p.157-173. 2011.

[3] Gansner E. R., Koren Y., and North S. Graph drawing by stress majorization. In Proceedings of the 11th International Symposium in Graph Drawing (GD'03), volume 2912 of Springer LNCS, p. 239–250, 2004.

[4] Pavlo A., Homan Chr., Schull J.. A parent-centered radial layout algorithm for interactive graph visualization and animation.

[5] Yee Ka-Ping, Fisher D., Dhamija R., Marti A. Hearst. Animated exploration of dynamic graphs with radial layout. In Proceedings of the IEEE Symposium on Information Visualization, p. 43–50, 2001.

[6] Pavlo A. Interactive, Tree-Based Graph Visualization, 2006.

[7] Источник набора графов – http://networkrepository.com/networks.php. Дата обращения (18.10.2017)

[8] Документация библиотеки LWJGL – https://javadoc.lwjgl.org

[9] Документация языка Java – https://docs.oracle.com/javase/8/docs/

[10] de Leeuw J. Convergence of the majorization method for multidimensional scaling. Journal of Classification, volume 5, issue 2:163–180, 1988.

[11] Kobourov S., Pupyrev S, Saket B.. Are crossings important for drawing large graphs? In Graph Drawing (GD 2014), volume 8871 of LNCS, pages 234–245. Springer, 2014

[12] Batini C., Furlani L., Nardelli E. What is a good diagram? a pragmatic approach. In Proceedings of the Fourth International Conference on Object-Oriented and Entity-Relationship Modelling, pages 312–319, Washington, DC, USA, 1985. IEEE Computer Society. ISBN 0-444-87951-X.

[13] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. Graph Drawing: Algorithms for the Visualization of Graphs. Prentice Hall, Upper Saddle River, New Jersey, 1999.

[14] Helen C. Purchase. The effects of graph layout. In OZCHI ’98: Proceedings of the Australasian Conference on Computer Human Interaction, page 80. IEEE Computer Society, 1998. ISBN 0-8186-9206-5.