МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение  
высшего образования   
 **«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

***К*афедра вычислительных технологий**

**КУРСОВАЯ РАБОТА**

**Алгоритмы визуализации динамических графов для анализа сложных сетей**

Работу выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А. Пасько

(подпись, дата) (инициалы, фамилия)

Факультет компьютерных технологий и прикладной математики курс 3

Направление 02.03.02 – «Фундаментальная информатика и  
информационные технологии»

Научный руководитель доц.,   
канд.тех. наук, доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Т.А. Приходько

(подпись, дата) (инициалы, фамилия)

Нормоконтролер ст. преп.,   
канд.техн. наук \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е.Е. Полупанова

(подпись, дата) (инициалы, фамилия)

Краснодар 2018

СОДЕРЖАНИЕ

[Введение 3](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350409)

[1 Общие методы построения динамических сетей 5](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350410)

[2 Алгоритмы визуализации динамических графов 7](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350411)

[2.1 Общие математические понятия 7](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350414)

[2.2 Силовая модель 12](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350415)

[2.3 Пружинная модель………………………………………………….. ....12](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350415)

[2.4 Недостатки алгоритмов…………………………………………….. ....12](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350415)

[3 Реализация алгоритмов 14](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350417)

[3.1Выполнение анализа 15](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350418)

[ЗАКЛЮЧЕНИЕ 21](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350447)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 23](file:///C:\Users\Админ\Desktop\6%20СЕМЕСТР\все%20предметы\5%20семестр\Курсовая%201\Пасько_КР1.docx#_Toc501350448)

ПРИЛОЖЕНИЕ …………………………………………………………………..

ВВЕДЕНИЕ

Граф - это одна из основных структур данных, которая моделирует большое количество объектов и связей между ними.

При визуализации сетей связи очень широко используется понятие "граф", ведь сложно представить формальное описание сети без данной структуры.

В век информационных технологий количество информации с каждым днем растет, поэтому роль графов в науке сложно переоценить. Именно графы используются в химии для установления связей между молекулами и атомами, в физике для моделирования взаимодействий между телами, в биологии при помощи графов моделируют поведение бактерий, социологи используют графы для изучения связей и закономерностей в обществе, а в программировании являются одной из главных структур данных.

Визуализация – один из важнейших компонент при моделировании графов. Она активно развивается еще с 80-х годов, т.к представляет собой удобный инструмент анализа данных. Задача визуализации заключается в том, чтобы представить граф в простом и наглядном виде, подчеркивающем его структуру и топологические свойства. Существуют различные методы визуализации графов, многие из которых опираются на определенную топологию: планарные графы, деревья. Соответственно, разные алгоритмы предназначены для различных видов графов и могут накладывать свои особые ограничения на расположение вершин, на вид ребер (прямые, кривые или ортогональные линии). Также стоит отметить, что от выбора алгоритма напрямую зависит полученный результат, ведь алгоритмы для построения больших графов будут не эффективны для небольших значений, что соответственно справедливо и наоборот. Но поскольку сеть не всегда имеет классическую топологию (звезда, кольцо, и пр.), необходимо выбирать алгоритм размещения общего графа, то есть алгоритм, не зависящий от вида графа, одинаково успешно располагающий различные структуры. Выбирая алгоритм для визуализации, следует точно определить свойства визуализируемого объекта и критерии оценки генерируемых алгоритмом изображений. Критерии, которые необходимо предъявить к графу, более подробно будут описаны в следующем пункте.

Курсовая работа посвящена современным алгоритмам визуализации, основанном на физических аналогиях. Наиболее гибкими алгоритмами, позволяющие строить простые неориентированные графы, являются силовые алгоритмы, также известные под названием пружинных.

В теории также рассматриваются алгоритмы для визуализации общих графов, т.е предназначенные для визуализации графов заранее неизвестной топологией. Основное отличие таких графов заключается в том, что они изменяются с течением времени: могут появиться как новые значения, так и исчезнуть старые, также возможны изменения с атрибутами вершин и ребер. Можно привести большое количество примеров, где явно это отражено. Например, визуализация социального графа Twitter или Facebook. Основные изменения графа-это появление новых знакомств. Сложность данной задачи заключается в том, что необходимо ясно и четко подчеркнуть происходящие изменения в графе. На сегодняшний день имеются различные методы визуализации, каждый из которых имеет свою специфику и особенности. В курсовой работе рассматривается только теоретическая часть визуализации динамических графов, т.к данная задача будет рассматриваться в следующей работе.

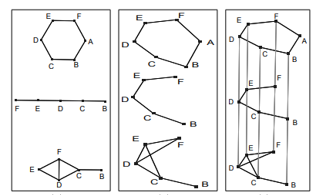
1 Общие методы построения динамических сетей

Для изображения динамических сетей будем использовать метод послойной визуализации или 2.5D-визуализация, который состоит из следующих шагов:

а) Период времени, рассматриваемый нами, разбивается на непересекающиеся подпериоды. Затем строится последовательность графов, моделирующая исходные данные за каждый подпериод. Количество выбранных подпериодов зависит от предметной области. Например, в нашем случае, с социальными сетями можно анализировать изменения за неделю.

б) Для каждого построенного графа строится укладка на плоскости.

в) В результате, финальная визуализация получается путем объединения всех укладок, т.е соединения вершин, которые соответствуют одинаковым объектам в соседних графах. (рис.1).



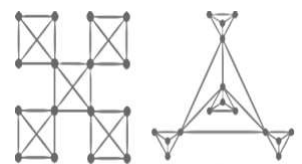
(а) (b) (c)

Рисунок 1 - граф, состоящий из трех слоев. (a) Укладки без учета ментальной карты. (b) Сохранение ментальной карты. (c) Послойная визуализация.

При визуализации графов возникает ряд трудностей. Первое – важно знать, как представлены исходные данные графа. Не всегда граф, подающийся на вход удовлетворяет удобочитаемости, т.к большое количество пересечений ребер не позволяет проанализировать модель.

Второе – очень часто исходные данные модели представлены в текстовом виде, например, XML, где отсутствует информация о вершинах графовой модели, поэтому эта проблема остается ключевой в системах визуализации данных.

При построении визуализации важно учитывать ряд критериев: во-первых - равномерное распределение вершин и ребер, минимизация пересечений и симметричность. Но также следует отметить, что не всегда минимизация пересечений приводит к улучшению читаемости изображения, все зависит от случая и поставленной задачи. На рис. 2 приведены два изображения одного и того же планарного графа. Однако очевидно, что следует искать некоторый баланс между различными критериями.



1. (b)

Рисунок2 - граф, представленный в двух различных укладках:(a), (b).

Изображение графа на рис. 2 (a) имеет 5 пересечений ребер, а изображение на рис. 2(b) не имеет пересечений. Если мы, по каким-то причинам, заинтересованы в планарности изображения, то, безусловно, размещение вершин, приведенное на рис.2(b), подходит больше. Но если нас не интересует планарность, изображение (a) с пересечениями, представляется более удобным для восприятия.

Во-вторых – последовательность укладок должна сохранять ментальную карту, т.е графы должны сохранять схожесть между собой. Будем считать, что ментальная карта сохраняется, если вершины графа приближенно сохраняют свои координаты в последовательных укладках.

В-третьих – суммарная площадь, занимаемая графом, должна быть минимальна, но доля свободного места не должна быть меньше некоторого предела.

В-четвертых – количество наложений вершин друг на друга должно быть минимальным.

Также стоит отметить, что алгоритм автоматической укладки графа должен быть эффективным по времени исполнения.

Существует множество методов построения укладок графов на плоскости:

- алгоритмы на основе физической модели,

-аналитические алгоритмы,

-генетические алгоритмы.

2 Алгоритмы визуализации динамических графов

2.1 Общие математические понятия

В данной курсовой работе будут представлены следующие алгоритмы:

* Силовой алгоритм
* Пружинный алгоритм

Введем необходимые математические обозначения, т.к далее будут рассмотрены математические идеи алгоритмов.

Граф G состоит из множества вершин V и множества ребер E⊆V×V. Количество вершин в графе обозначается через n=|V|, количество ребер – m=|E|. Вершины графа V={,…,}, также будем обозначать индексы V={1,…,n}. Для вершины v вводится ее вес , аналогично для ребра (u, v) вес обозначатся через .

Под укладкой L графа G= (V, E) понимаем отображение L: V-> вершин графа в множество точек d-мерного пространства. Нарисовать граф т.е указать координаты его вершин, а ребра изобразить прямыми отрезками между вершинами. Позицию вершины будем обозначать через или , ее координаты - ,…,. Под расстоянием между вершинами будем понимать евклидово расстояние между соответствующими точками и обозначать через ||-||. Для динамической визуализации требуется изобразить последовательность графов ,,…,, которые описывают некоторые данные за последовательный промежуток времени. У каждой вершины имеется некая метка . Также будем считать, что = тогда и только тогда, когда выбранные вершины u, v соответствуют одному объекту исходных данных.

2.2 Силовая модель

Алгоритмы, принадлежащие к классу силовых, считают наиболее гибкими для размещения вершин общих неориентированных графов. Как правило, такие алгоритмы используют только информацию, которая содержится в структуре самого графа, не предполагая, что пригодятся дополнительная информация. Такие графы удовлетворяют критериям эстетичности, т.е симметричны, а также, очень часто, не имеют пересечений ребер для изображения планарных графов.

Традиционный силовой алгоритм визуализации графов разработал и опубликовал Питер Идс (Peter Eades) в 1984 году. В этом алгоритме граф представляется в виде физической системы, в которой вершины графа соответствуют заряженным частицам, а ребра – пружины между ними, и действуют, как силы притяжения, так и отталкивания.

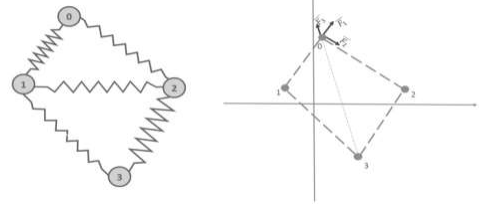


Рисунок 3 - Граф как физическая модель [2]

На рисунках изображена физическая модель графа. Видно, что вершины стремятся принять исходной положение, соответственно на смежные вершины, расположенные друг от друга дальше, чем заданная идеальная длина ребра, действует сила притяжения. Например, на вершину с индексом 0 действуют три силы со стороны остальных вершин. Силы со стороны вершин с индексами 1 и 2 складываются из силы, действующей на вершину со стороны пружины, и электрическую силу, действующую на вершины как на одноименно заряженные электрические заряды, а со стороны вершины с индексом 3 действует лишь электрическая сила, поскольку вершины 0 и 3 не связаны ребром. В данной модели силы, действующие на вершины со стороны пружин, как притягивают, так и отталкивают вершины друг от друга, стремясь достичь оптимальной длины ребра. Электрические силы добавлены, чтобы избежать наложения несмежных вершин. Основным минусом данного алгоритма является ограничение на размер визуализируемого графа (до 150 вершин).

Как и представлено выше, силовая модель состоит из вершин, которые представляют собой заряженные частицы, между которыми действуют две силы: притяжения и отталкивания. Силы отталкивания действуют между всеми парами вершин и обратно пропорциональны расстоянию между вершинами, они разъединяют близко находящиеся вершины, т.е в хорошей укладке “слипшихся” вершин не обнаружится. В свою очередь, силы притяжения действуют между вершинами, которые соединены ребром, в конечной укладке такие вершины будут находиться рядом. В общем виде энергию системы можно выразить следующим образом:

+= (1)

Первое слагаемое – энергия притяжения, второе – энергия отталкивания между вершинами. Показатели степени a, rR и коэффициенты ,R определяют степень влияния каждого компонента на итоговую укладку. Общепринятыми ограничениями являются: , a ≥ 0, r ≤ 0, т.е силы притяжения растут, а отталкивания убывают с увеличением расстояния.

Основная задача данных алгоритмов – это расположение узлов (это может быть 2D или 3D пространства) так, чтобы все ребра были примерно одинаковой длины, и как можно меньше пересекающихся ребер. Как только силы на узлах и ребрах определены, поведение всего графа можно моделировать так, как если бы это была физическая система. Силы, действующие на узлы, либо сближают их, либо отдаляют. Эти действия длятся до тех пор, пока система не придет в состояние механического равновесия, т.е позиции вершин и ребер становятся неизменными. Как раз в таком равновесии используют позиции узла для изображения графа.

В задаче динамической визуализации графов необходимо обеспечить схожесть укладок графов. Под словом «схожесть» понимается приближенное сохранение позиций вершин с одинаковыми метками в разных графах. Для этого мы строим новый граф, являющийся объединением всех других графов последовательности с добавлением новых ребер: каждая пара вершин с одинаковыми метками соединяется ребром. В итоге получим граф-объединение: = (), где = и = U{(u,v)⊆=}. Вспомогательные ребра между различными графами будут обеспечивать силы притяжения между вершинами с одинаковыми метками и исходные объекты будут стремиться занять одинаковые позиции во всех слоях итоговой послойной укладки, что соответствует сохранению ментальной карты. Дополнительная составляющая энергии для работы с динамическими графами выражается так:

=,где>0 и m (2)

Одной из составляющих данного алгоритма является минимизация энергии. Общая энергия системы для последовательности графов выглядит следующим образом:

U=++ (3)

Или другими словами – можно рассмотреть, как функцию n\*d, где n-это вершины, а и d - координаты. Для того, чтобы найти минимум, необходимо применить метод градиентного спуска. Первоначально каждой вершине присваиваются случайные координаты. На каждой итерации алгоритма предполагаем, что все вершины фиксированы, кроме одной. Будем сдвигать эту вершину в направлении антиградиента так, чтобы максимально уменьшить энергию системы. Данный шаг применяется для всех вершин в графе. Завершение работы алгоритма произойдет тогда, когда энергия достигнет своего локального минимума. Но т.к в этой курсовой работе рассматривается только визуализация графов (без послойной визуализации), то результатов построения таких укладок не будет.

Изображения графов, полученные при помощи силовых алгоритмов, как правило, удовлетворяют критерию эстетичности, характеризуются симметричностью, а также, с большой вероятностью, не имеют пересечений ребер для изображений планарных графов, что позволяет сделать вывод о соответствии генерируемых изображений определенным ранее критериям. Этот алгоритм, минимизируя энергию системы, сдвигает вершины в направлении действующей на них силы. Положение вершин, в котором энергия системы минимальна, является искомым размещением. Алгоритм направлен на то, чтобы обеспечить расположение смежных вершин на достаточно близком расстоянии. Данный алгоритм находит симметричное расположение (насколько это возможно при равной длине ребра). В этой модели силы, действующие на вершины со стороны пружин, как притягивают, так и отталкивают вершины друг от друга, стремясь достичь оптимальной длины ребра. Т.о в «силовом» методе граф рассматривается как система пружин - ребер и шарниров с вершинами – одноименными электрическими зарядами, между которыми действуют силы отталкивания.

2.3 Пружинная модель

Пружинная модель – одна из популярных физических моделей рисования графов. В этой модели ребра – это пружины, при растяжении и сжатии которых возникают силы упругости, действующие по закону Гука; ребра-пружины стремятся вернуть свою первоначальную длину. Укладка графа при помощи пружинной модели эквивалентна методу многомерного шкалирования (MDS, multidimensional scaling), перейдем к его описанию.

Метод MDS позволяет расположить объекты в пространстве некоторой небольшой размерности таким образом, чтобы достаточно точно воспроизвести заданные расстояния между ними. Задача многомерного шкалирования ставится так: по заданной матрице расстояний D = ()∈ между объектами найти их позиции , . . . ,, ∈ так, чтобы выполнялось || − || = для всех i, j ∈ [1, n]. Также матрицу расстояний D накладывают ограничения: матрица должна быть симметричной = , имеют неотрицательные элементы ≥ 0 и нули на главной диагонали = 0.

На практике задача 1 не всегда имеет решение, поэтому разумно искать позиции объектов, приближенно сохраняющие заданные расстояния, т.е. такие, что || − || ≈ . Мерой качества приближения является функция стресс-функция:

Stress = . (4)

Если разрешить варьировать вклад каждого слагаемого в общую сумму, то получится следующая взвешенная задача многомерного шкалирования:

по матрицам расстояний D = ()∈ и весов W = () ∈ найти позиции , . . . , ∈ объектов, минимизирующих стресс-функцию:

min. (5)

При использовании метода MDS для визуализации графов в качестве D чаще всего берут матрицу кратчайших расстояний между вершинами. В результате вершины, соединенные ребром, будут расположены рядом, а вершины, имеющие большое теоретико-графовое расстояние, будут отдалены друг от друга. Вес обычно обратно пропорционален расстоянию между вершинами (= ,q<0). В таком виде алгоритм MDS подобен пружинной модели.

Важно отметить, что в «пружинном методе» графу сопоставляется некоторая упругая механическая система, в которой точкам соответствуют вершины графа, а «пружинам» – упругим силовым связям – ребра графа.

2.4 Недостатки алгоритмов

Современные алгоритмы рисования графов строят хорошие укладки для простых графов с известной структурой данных. Но как правило, на практике реальные сети имеют сложную структуру и работают не так, как хотелось бы. Рассмотрим основные проблемы в области визуализации сложных сетей:

* Одна из ключевых проблем визуализации сложных сетей проявляется в размещении вершин с разной степенью. Вершины, которые имеют большое количество ребер, оказываются в центре изображения и за счет этого плотность вершин в центре изображения заметно возрастает. Причины такого расположения две: малое среднее расстояние между вершинами и несимметричное распределение степеней вершин.
* Сейчас, как правило, вершины графа невозможно изобразить в виде точек, т.к каждой вершине ассоциирована некоторая текстовая или графическая информация, при этом вершины имеют различные размеры и не должны пересекаться и накладываться друг на друга.
* В зависимости от того, какой размер имеют вершины графа, следует особый выбор его ребер. Прямые линии, соединяющие соответствующие точки, могут пересекать метки вершин, тем самым снижая удобочитаемость графа.

3 Реализация алгоритмов

В курсовой работе были реализованы два алгоритма укладки графа на плоскости. Выбранные алгоритмы – модифицированные аналоги тех алгоритмов, которые описаны в пунктах 1.1-1.2.

На практике силовые алгоритмы чаще всего используют, работая с JS (Java Script), когда обращаются к библиотеке D3.js, которая использует в совокупности такие технологии, как HTML, SVG, CSS и D3. В D3 представлены удобные утилиты для обработки и загрузки массивов данных и создания DOM-элементов.

3.1 Выполнение анализа

Для алгоритмов, имеющих 50-500 вершин полученные результаты имеют хорошие показатели, т.е практически одинаковая длина ребра, равномерное распределение вершин, проявление симметричности. Стоит отметить, что группа этих алгоритмов достаточна проста в понимании, ведь все основано на физических аналогиях.

Далее приведен пример визуализации графа, используя JS, HTML, CSS.

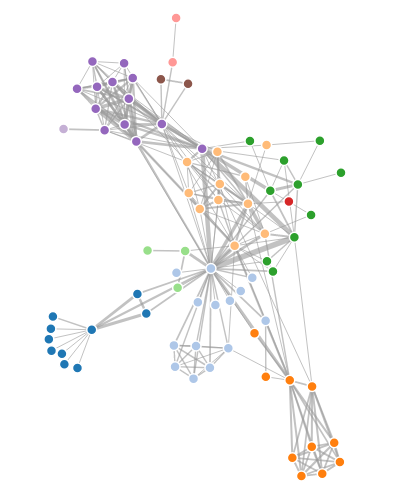


Рисунок – 4. Название

В курсовой работе также были созданы алгоритмы, написанные на языке Python. Для реализации алгоритмов были использованы: библиотека tkinter, которая позволяет визуализировать все составляющие графа, составляющие модуля pygame, библиотека math, np, остальные функции были написаны вручную, используя вышеописанные алгоритмы.

|  |
| --- |
| Результаты силового алгоритма |
| C:\Users\AMD\Desktop\тесты\sourse.py_text1_12.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_11.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_10.PNG  C:\Users\AMD\Desktop\тесты\sourse.py_text1_9.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_8.PNG    C:\Users\AMD\Desktop\тесты\sourse.py_text1_6.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_5.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_4.PNG  C:\Users\AMD\Desktop\тесты\sourse.py_text1_3.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_2.PNG C:\Users\AMD\Desktop\тесты\sourse.py_text1_1.PNG |
|  |

Рисунок - 5. Результаты пружинного алгоритма на основе одинаковых входных данных

Было создано 6 разновидностей графов (отличия заключались лишь в количестве вершин и ребер). Каждый из графов отработал по 14 раз на каждом из алгоритмов. В ходе работы было зафиксировано время выполнения приведенных алгоритмов и определено среднее значение времени в секундах. Многочисленные подсчеты показали, что разница между силовым и пружинным алгоритмами составляет 20 секунд в пользу пружинного.

Не мешало бы привести график – зависимость времени исполнения различных алгоритмов от числа вершин.

Стоит отметить, что данные методы относятся к одному и тому же разряду силовых алгоритмов. Отличие заключается лишь в некоторых модификациях формул и добавление в один из используемых в алгоритмов (в пружинный) электрической силы, которая также способствует отталкивать частицы, тем самым на практике были видны соответствующие результаты.

Как правило, при оценке графа ссылаются на общие критерии, которые приведены выше. Оценивая полученные результаты (рисунок - -), можно отметить, что укладки графов на данных алгоритмах соответствуют критериям эстетичности. Но при большом количестве связей укладки графа выражены неявно, это видно из рисунков (рис 7.), (рис.8).

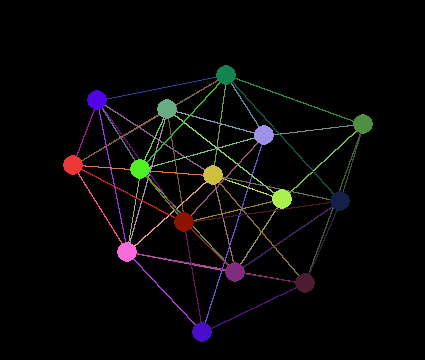


Рисунок -7

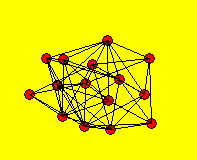


Рисунок - 8

ЗАКЛЮЧЕНИЕ

В завершении отметим, что одним из современных подходов визуализации графов является использование Force Directed методов, то есть методов, основанных на физических аналогиях. Эти методы предназначены в основном для визуализации неориентированных графов. Следует также отметить, что в отечественной литературе методы визуализации графов практически не описаны. Зарубежными учеными наиболее активно разрабатываются так называемые Force-Directed методы.

В курсовой работе были реализованы два метода Force Directed. Данные раскладки графов показали, что визуализация с большим количеством вершин и ребер графа будет не эффективна. Основная цель работы заключалась в выяснении, какой из модифицированных методов больше всего будет подходить для послойной визуализации сложных динамических графов, изменяющихся со временем, которые предполагается реализовать в дальнейшей работе.

Таким образом, на основании экспериментов, был сделан вывод, что пружинный алгоритм метода Force Directed является наиболее оптимальным (по времени или по другому критерию?).

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Пупырев С.Н, Модели, алгоритмы и программный комплекс визуализации сложных сетей.

|  |
| --- |
| 2 Пупырев С.Н., Тихонов А.В., Визуализация динамических графов для анализа сложных сетей.  3 Соколов Г.В., Анализ алгоритмов автоматической укладки графов на плоскости в рамках задачи визуализации моделей на графах. |

ПРИЛОЖЕНИЕ

Листинг программы №1:  
import tkinter  
import random  
import math  
import time  
  
start\_time = time.clock()  
d=[]  
alpha = 4.0  
beta = .0001  
k = 1.0  
#торможение  
eta = .9  
delta\_t = .01  
m = len(d)  
root = tkinter.Tk()  
canvas = tkinter.Canvas(root, width=1000, height=800, background="yellow")  
canvas.pack()  
x = []  
v = []  
ids = []  
def move\_oval(i):  
 newx = int(x[i][0] \* 500)  
 newy = int(x[i][1] \* 500)  
 canvas.coords(ids[i], newx - 5, newy - 5, newx + 5, newy + 5)  
  
for i in range(m):  
 xi = [random.random(), random.random()]  
 x.append(xi)  
 v.append([0.0, 0.0])  
 id = canvas.create\_oval(245, 245, 255, 255, fill="red")  
 ids.append(id)  
 move\_oval(i)  
lids = []  
def move\_line(id, xi, xj):  
 canvas.coords(id,  
 int(xi[0] \* 500),  
 int(xi[1] \* 500),  
 int(xj[0] \* 500),  
 int(xj[1] \* 500))  
for i in range(m):  
 for j in range(m):  
 if d[i][j] != 0:  
 id = canvas.create\_line(0, 0, 0, 0)  
 lids.append(id)  
 move\_line(id, x[i], x[j])  
def Coulomb\_force(xi, xj):  
 dx = xj[0] - xi[0]  
 dy = xj[1] - xi[1]  
 ds2 = dx \* dx + dy \* dy  
 ds = math.sqrt(ds2)  
 ds3 = ds2 \* ds  
 if ds3 == 0.0:  
 const = 0  
 else:  
 const = beta / (ds2 \* ds)  
 return [-const \* dx, -const \* dy]  
def Hooke\_force(xi, xj, dij):  
 dx = xj[0] - xi[0]  
 dy = xj[1] - xi[1]  
 ds = math.sqrt(dx \* dx + dy \* dy)  
 dl = ds - dij  
 const = k \* dl / ds  
 return [const \* dx, const \* dy]  
def move():  
 ekint = [0.0, 0.0]  
 for i in range(m):  
 Fx = 0.0  
 Fy = 0.0  
 for j in range(m):  
 if j == 1:  
 continue  
 dij = d[i][j]  
 Fij = 0.0  
 if dij == 0.0:  
 Fij = Coulomb\_force(x[i], x[j])  
 else:  
 Fij = Hooke\_force(x[i], x[j], dij)  
 Fx += Fij[0]  
 Fy += Fij[1]  
 v[i][0] = (v[i][0] + alpha \* Fx \* delta\_t) \* eta  
 v[i][1] = (v[i][1] + alpha \* Fy \* delta\_t) \* eta  
 ekint[0] = ekint[0] + alpha \* (v[i][0] \* v[i][0])  
 ekint[1] = ekint[1] + alpha \* (v[i][1] \* v[i][1])  
 print ("total kinetic energy: %lf" % math.sqrt(ekint[0] \* ekint[0] + ekint[1] \* ekint[1]))  
 for i in range(m):  
 x[i][0] += v[i][0] \* delta\_t  
 x[i][1] += v[i][1] \* delta\_t  
 move\_oval(i)  
 li = 0  
 for i in range(m):  
 for j in range(m):  
 if d[i][j] != 0:  
 id = lids[li]  
 move\_line(id, x[i], x[j])  
 li += 1  
 root.after(1, move)  
root.after(1, move)  
root.mainloop()  
print (time.clock() - start\_time, "seconds")

Листинг программы №2:

import time  
import pygame  
import numpy as np  
start\_time = time.clock()  
pygame.init()  
screen\_w, screen\_h = screen\_size = 1000, 800  
screen = pygame.display.set\_mode(screen\_size)  
clock = pygame.time.Clock()  
running = True  
c\_white = (255, 255, 255)  
c\_black = ( 0, 0, 0)  
c\_red = (255, 0, 0)  
d = []  
graph=np.array(d)  
class Body:  
 def \_\_init\_\_(self, id, pos=None, mass=10):  
 if pos == None:  
 self.pos = np.multiply(np.random.random(2) - .5, screen\_size)  
 else:  
 self.pos = np.array(pos).astype(float)  
 self.id = id  
 self.mass = mass  
 self.vel = np.array([0, 0]).astype(float)  
 self.color = (255\*np.random.random(3)).astype(int)  
bodies = [Body(id) for id in range(graph.shape[1])]  
def edge\_exists(body\_a, body\_b):  
 a = body\_a.id  
 b = body\_b.id  
 return graph[a, b] or graph[b, a]  
def avg\_pos(bods):  
 avg = np.array([0., 0.])  
 for body in bods:  
 avg += body.pos  
 return avg / len(bods)  
def unit(v):  
 return (v / np.linalg.norm(v))  
def update\_body\_physics(bods, dt):  
 const\_charge = 100000  
 const\_spring = .001  
 spring\_equ\_len = 10  
 for a, body\_a in enumerate(bods):  
 f\_net = np.array([0., 0.])  
 for b, body\_b in enumerate(bods):  
 if a == b: continue  
 ab = body\_b.pos - body\_a.pos  
 \_ab = np.linalg.norm(ab)  
 # spring force  
 f\_s = np.array([0., 0.])  
 if edge\_exists(body\_a, body\_b):  
 f\_s = const\_spring \* unit(ab) \* (\_ab - spring\_equ\_len)\*\*(+2)  
 # electric force  
 f\_c = const\_charge \* unit(ab) \* \_ab\*\*(-2)  
 f\_net += f\_s - f\_c  
 body\_a.vel += (f\_net / body\_a.mass) \* dt  
 body\_a.vel \*= .99  
 body\_a.pos += body\_a.vel \* dt  
def scr\_pos(body, p\_avg):  
 pos = body.pos.copy() + np.array(screen\_size) / 2 - p\_avg  
 pos[1] = screen\_h - pos[1]  
 return pos.astype(int)  
while running:  
 for event in pygame.event.get():  
 if event.type == pygame.QUIT:  
 running = False  
 dt\_ms = clock.tick(60)  
 dt\_s = dt\_ms \* .001  
 update\_body\_physics(bodies, dt\_s)  
 # draw the circles  
 screen.fill((c\_black))  
 p\_avg = avg\_pos(bodies)  
 for body\_a in bodies:  
 pos\_a = scr\_pos(body\_a, p\_avg)  
 for body\_b in bodies:  
 if (body\_a.id == body\_b.id) or not edge\_exists(body\_a, body\_b): continue  
 pos\_b = scr\_pos(body\_b, p\_avg)  
 pygame.draw.line(screen, (body\_a.color + body\_b.color)/2, pos\_a, pos\_b)  
 for body in bodies:  
 pos = scr\_pos(body, p\_avg)  
 pygame.draw.circle(screen, body.color, pos, 10)  
 pygame.display.update()  
print (time.clock() - start\_time, "seconds")