САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА МАТЕМАТИЧЕСКОЙ ТЕОРИИ ИГР И СТАТИСТИЧЕСКИХ РЕШЕНИЙ

Белейченко Татьяна Александровна

Выпускная квалификационная работа бакалавра

Кластеризация на графах с качественной информацией на примере социальной сети

Направление 01014

Прикладная математика и информатика

Научный руководитель,
кандидат физ.-мат. наук,
старший преподаватель
Тур Анна Викторовна

Санкт-Петербург

2016

Содержание

Введение

В современном, стремительно развивающемся мире во многих производственных, научно-естественных, технических, экономических и др. областях стремительно растет объем данных. Вместе с тем возрастает потребность в упорядочении, выделении связанных групп по принципу схожести из огромного количества объектов. Именно такого рода проблему и помогает решить кластеризация. Информация, полученная вследствие кластеризации данных полезна и многогранна-вот лишь некоторые области практического применения качественных кластеров:

1. В области медицины используется кластеризация заболеваний, их симптомов, лечения.
2. В археологии устанавливаются таксономии каменных сооружений и древних объектов и т.д.
3. В маркетинге задачей кластеризации может являться выделение групп потребителей и конкурентов.
4. В социологии задача кластеризации - разбиение респондентов на однородные группы.

Ещё одной областью применения методов кластеризации данных является интернет-пространство. С его развитием продолжают активно набирать популярность такие социальные сети как «Вконтакте», «Одноклассники», «Facebook», и др. Количество пользователей, зарегистрированных в этих сетях, огромно и с каждым днем оно только увеличивается. Эти сети представляют собой богатый источник данных. Особый интерес вызывает структура графа, индуцированного по ссылкам дружбы. В представленной работе будут проанализированы некоторые методы кластеризации и применены к данным социальной сети «Вконтакте» с целью определения неявных сообществ среди друзей.

Социальная сеть представима в виде графа, в котором узлами являются пользователи, а наличие связи между узлами означает то, что пользователи находятся в «друзьях» друг у друга. При этом отсутствует какая-либо количественная мера, характеризующая эти связи. Такие структуры называют графами с качественной информацией. Под качественной информацией обычно понимаются некоторые данные о схожести узлов. Любая пара узлов либо наделена свойством схожести, либо нет. Кластеризация графов с качественной информацией подразумевает разбиение множества узлов графов на сообщества таким образом, чтобы в каждом кластере оказалось как можно больше схожих узлов, а между кластерами количество схожих узлов минимизируется.

Существует множество методов кластеризации графов с качественной информацией. Каждый их них предлагает некоторую количественную меру, характеризующую качество разбиения, и предлагает алгоритм оптимизирующий разбиение, согласно этой выбранной мере. Среди известных методов можно выделить:

- метод Betweenness: алгоритм на основе коэффициента “центральности по посредничеству”, определяемый как количество кратчайших путей между всеми парами вершин, проходящих через данное ребро [1];
- Multilevel: метод, основанный на многоуровневой оптимизации функции модулярности, с использованием эвристики, предложенной в статье [2];
- LabelPropogation: метод, основанный на присвоении меток к каждой вершине. Пошагово выбирается метка с максимальным повторением среди смежных вершин [3].

В работе рассмотрены два алгоритма с различными подходами к кластеризации: 1) «Алгоритм жадной оптимизации», предложенный в статье Ньюмана и Гирвана «Поиск и оценка структуры сообществ в сетях» в 2004г. (M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks); 2) «Осторожный алгоритм» (Algorithm Cautious) Бансаля и Блума,

рассмотренный в статье «Корреляция кластеризации» в 2002 г. (N .Bansal, A .Blum, S .Chawla, Correlation clustering).

Целью работы является реализация рассмотренных алгоритмов в программной среде, применение их к реальным данным, полученным из социальной сети «Вконтакте», анализ полученных результатов и сравнение работы двух алгоритмов кластеризации.

Обзор литературы

Наибольший интерес в последних опубликованных работах, касающихся кластеризации, вызывает такая структура сообществ, при которой плотность связей между объектами внутри каждой группы высокая, а между группами низкая. Проблема выделения таких сообществ внутри сети была достаточно хорошо изучена. Ранние методы, такие как алгоритм Керниган-Лин [4], спектральное разбиение [5,6] или иерархическая кластеризация [7] работают хорошо для специфических типов проблемы (в частности, бисекция графа или проблемы с хорошо определенными мерами сходства вершин), но плохо работают для более общих случаев.

Для решения этой проблемы ряд новых алгоритмов был предложен в последние годы. Так, Бансаль и Блум в статье [8] решают задачу MaxAgree (максимизация соглашений) и MinDisagree (минимизация рассогласований) и приводят алгоритм «Cautious», который далее в работе детально рассмотрен и применен на практике. Гирван и Ньюман [9,10] предложили алгоритм, который использует промежуточные рёбра в качестве метрики для определения границ сообществ. Этот алгоритм был успешно применен к различным сетям, в том числе сетям сообщений электронной почты, сетям сотрудничества между учеными и музыкантами, обменным и генным сетям. Однако, как отмечается в [10], алгоритм предъявляет высокие требования к вычислительным ресурсам, работает за $O(m^2 n)$ времени на произвольной сети с m рёбрами и n вершинами, или $O(n^3)$ времени на разреженном графе. Это ограничивает применение алгоритма к сетям с максимум несколько тысячью вершинами. В последнее время был предложен ряд быстрых алгоритмов [11,12,13]. В [12] представлен алгоритм, основанный на жадной оптимизации величины, известной как модулярность [10]. Этот метод работает хорошо, как для тестовых задач, так и на практике в реальных ситуациях, и работает быстрее, чем алгоритм Гирвана и Ньюмана. Простая

реализация выполняется за время $O((m+n)n)$ или $O(n^2)$ на разреженных графах.

В работе [10] применён новый алгоритм, который выполняет те же жадные оптимизации алгоритма [6] и, следовательно, дает одинаковые результаты для найденных сообществ. Однако, используя более сложные структуры данных, он работает гораздо быстрее, за время $O(mdlogn)$ где d-глубина “дендограммы”, описывающая структуру сети сообщества.

Постановка задачи

Дан неориентированный граф G=(V,W), где V-множество узлов/вершин, W-множество рёбер, соединяющих пары вершин из множества V.

Будем считать, что $(i, j)\in$W, если $i\in$V и $j\in$V связаны ребром, $(i, j)\notin$W, если узлы i и j не связаны.

Пусть A - матрица смежности графа G:

$$A_{ij} = \begin{cases} 1, & \text{если } (i,j)\in \text{W}, \\ 0, & \text{в обратном случае.} \end{cases}$$

Разбиением графа на сообщества будем называть такое разбиение множества его вершин, что P=$\{C_1 \ldots . C_k\}$, т.е. $\bigcup_{i=1}^{k} C_i$=V и $\text{C}_i \cap C_j = \varnothing$ $\forall$ $i\neq j\in$1,…,k.

Одной из возможных мер качества разбиения является значение модулярности, предложенное Ньюманом и Гирваном в 2002 году в ходе разработки алгоритма кластеризации вершин графа [9]. Опишем это понятие.

Предположим, что вершины разделены в сообщества таким образом, что вершина $v\in$V принадлежит сообществу c_v. Тогда количество рёбер, расположенных внутри сообществ равно:

$$\frac{\sum_{vw} A_{vw}\delta(c_v,c_w)}{\sum_{vw} A_{vw}} = \frac{1}{2m}\sum_{vw} A_{vw}\,\delta(c_v, c_w)\,, \qquad (1)$$

где δ-функция $\delta(i, j)$ такова, что $\delta(i, j)$=1 если i=j и 0 в обратном случае.
Здесь через m обозначено общее количество рёбер в графе:

$$m = \frac{1}{2}\sum_{vw} A_{vw}.$$

Величина, заданная в (1) будет принимать большие значения для

разбиений сети, имеющих много рёбер, находящихся внутри сообществ. Но эта величина не является хорошей мерой качества разбиения, т.к. принимает своё наибольшее значение равное 1 в тривиальном случае, когда все вершины принадлежат единственному сообществу.

В работе [10] предложено от величины, описанной в (1) (доли связей внутри сообществ), отнять ожидаемую долю связей, если бы рёбра были размещены случайно:

$$Q=\frac{1}{2m}\sum_{vw}\left[A_{vw}-\frac{k_v k_w}{2m}\right]\delta(c_v,c_w).$$

Полученное значение Q авторы назвали модулярностью. Через k_v обозначена степень вершины v, т.е. количество рёбер, исходящих из этой вершины v:

$$k_v=\sum_w A_{vw}.$$

Если количество рёбер внутри сообщества не отличается от ожидаемого количества для рандомизированной сети, то тогда эта величина будет равна 0.

Алгоритм, предложенный в работе [10] предполагает строить разбиения графа так, чтобы величина Q была максимальной.

Нашей задачей является реализация предложенного алгоритма и применение его к реальным данным, полученным из социальной сети «Вконтакте». Будет рассмотрен граф с количеством узлов, равным 533. Также интересно сравнить полученные результаты с результатами работы других известных алгоритмов кластеризации. Поэтому в работе также приведён «Осторожный алгоритм» кластеризации Бансаля и Блума, рассмотренного в статье «Корреляция кластеризации» [8].

Глава 1: Алгоритм жадной оптимизации

В данной главе опишем сам алгоритм и предоставим его реализацию в программной среде на реальных данных из социальной сети «Вконтакте».

Пусть e_{ij} - доля рёбер, соединяющих вершины сообщества i с вершинами сообщества j :

$$e_{ij} = \frac{1}{2m}\sum_{vw} A_{vw}\,\delta(c_v, i)\delta(c_w, j).$$

Через a_i определим часть концов рёбер, принадлежащих вершинам в сообществе i, т.е.:

$$a_i = \frac{1}{2m}\sum_{v} k_v\,\delta(c_v, i).$$

Величину $\delta(c_v, c_w)$ можно представить в виде $\delta(c_v, c_w) = \sum_i \delta(c_v, i)\delta(c_w, i)$, тогда:

$$Q = \frac{1}{2m}\sum_{vw}\left[A_{vw} - \frac{k_v k_w}{2m}\right]\sum_i \delta(c_v, i)\delta(c_w, i) =$$

$$\sum_i \left[\frac{1}{2m}\sum_{vw} A_{vw}\,\delta(c_v, i)\delta(c_w, i) - \frac{1}{2m}\sum_v k_v\,\delta(c_v, i)\frac{1}{2m}\sum_w k_w\,\delta(c_w, i)\right] = \sum_i\left(e_{ii} - a_i^2\right).$$

1.1 Описание алгоритма

Опишем работу алгоритма. На каждой итерации действие алгоритма заключается в поиске пары сообществ, в результате объединения которых величина Q изменится максимально. В качестве начального разбиения выбирается такое разбиение, при котором каждый узел представляет собой сообщество. Один способ предусмотреть (и реализовать) процесс объединения сообществ – подумать о сети как о мультиграфе, в котором целое сообщество представлено вершиной, связки рёбер соединяют одни вершины с другими, а ребра внутренних сообществ как самостоятельные

рёбра. Матрица смежности этого мультиграфа имеет элементы $A'_{ij} = 2me_{ij}$ и соединение двух сообществ i и j соответствует замене i-ых и j-ых строк и столбцов их суммой. В алгоритме [12] эта операция явно выполнена на всей матрице, но, если же матрица смежности является разреженной, операция может быть проведена более эффективно с использованием структур данных для разреженных матриц. К сожалению, вычисление ΔQ_{ij} и нахождение пары i, j с самым большим значением ΔQ_{ij}, становится затратным по времени.

Соединение двух сообществ, между которыми нет рёбер, не может повлиять на увеличение значения Q. Нам необходимо рассматривать только значения ΔQ_{ij} для пар i, j, которые соединены одним или более рёбрами. Кроме того, алгоритм использует эффективную структуру данных для отслеживания наибольших значений ΔQ_{ij}. Эти улучшения результата в значительной мере сказываются на экономии памяти и времени.

В общей сложности сохраняются три структуры данных:

1. Разреженная матрица, содержащая ΔQ_{ij} для каждой пары i, j сообществ с, как минимум, одним ребром между ними.
2. Множество H, содержащее наибольшие элементы каждой строки матрицы ΔQ_{ij} вместе с метками i, j соответствующих пар сообществ.
3. Обычный векторный массив с элементами a_{ij}.

Как описано выше, мы начинаем с каждой вершины, являющейся единственным членом сообщества, в котором $e_{ij} = \frac{1}{2m}$, если i и j связаны ; $e_{ij} = 0$ в обратном случае и $a_i = \frac{k_i}{2m}$. Таким образом, мы изначально положили, что

$$\Delta Q_{ij} = \begin{cases} \frac{1}{2m} - \frac{k_i k_j}{2m^2}, & \text{если } i,j \text{ связны,} \\ 0, & \text{в обратном случае} \end{cases}$$

(2)

и

$$a_i = \frac{k_i}{2m}, \tag{3}$$

для каждого i.

Наш алгоритм может быть определён следующим образом:

1. Вычисляем начальные значения ΔQ_{ij} и a_i согласно формулам (2) и (3) и заполняем множество H наибольшими элементами каждой строки матрицы ΔQ.
2. Выбираем наибольший элемент ΔQ_{ij} из H, объединяем соответствующие сообщества, обновляем матрицу ΔQ, множество H и a_i (как описано ниже) и увеличиваем величину Q на соответствующее максимальное значение ΔQ_{ij}.
3. Повторяем шаг 2 до тех пор, пока максимальная величина ΔQ_{ij} принимает положительное значение.

Структура наших данных позволяет нам выполнить обновление в шаге 2 довольно быстро. Первым делом, заметим, что в корректировке нуждаются только несколько элементов матрицы ΔQ. Если объединяем сообщества i и j, то строка, соответствующая сообществу i, удаляется из матрицы ΔQ, а обновить необходимо только строку, соответствующую сообществу j. Правила обновления следующие. Если сообщество k связано с i и j, тогда:

$$\Delta Q'_{jk} = \Delta Q_{ik} + \Delta Q_{jk}.$$

Если k связано с i, но не связано с j, тогда:

$$\Delta Q'_{jk} = \Delta Q_{ik} - 2a_j a_k.$$

Если k связано с j, но не связано с i, тогда:

$$\Delta Q'_{jk} = \Delta Q_{jk} - 2a_i a_k.$$

Заметим, что Q имеет только один пик за алгоритм, так как после того, как наибольшее значение ΔQ становится отрицательным, далее все значения ΔQ могут только уменьшаться. Поэтому работа алгоритма останавливается, как только максимальное значение ΔQ_{ij} становится отрицательным.

1.2. Реализация алгоритма

Описанный алгоритм был реализован на языке JavaScript. Первая часть программы посвящена выгрузке данных из социальной сети «Вконтакте». Для конкретного пользователя считывается множество его друзей и определяются связи между ними. Строится матрица смежности. Вторая часть посвящена непосредственной реализации алгоритма. В третьей части программы реализуется визуализация полученных результатов.

Перейдем к подробному описанию каждого их модулей программы.

Структура программы:

Для запуска программы необходимо открыть файл **index.html** в любом из браузеров. В данном файле содержится описание пользовательского интерфейса в виде html-разметки и подключены используемые модули (JavaScript файлы).

Основные модули:

1. **main.js** – содержит логику взаимодействия пользователя с интерфейсом. К примеру, реакция на нажатие кнопок.
2. **vkFriends.js** – содержит методы для работы с API VK, такие как, авторизация, получения списка друзей и др.
3. **algorithms.js** – содержит реализацию алгоритма кластеризации.
4. **graph.js** – содержит методы для отображения данных в виде графа.

Описание работы программы:

1. Откройте файл **index.html** в любом из браузеров.
2. В поле «**Client Id**» введите Id вашего приложения, зарегистрированного в VK. Узнать его или создать новое можно на странице: https://vk.com/apps?act=manage
3. Нажмите на кнопку «**Login**» и откроется окно с запросом логина и пароля для входа в VK, если они у вас не закешированны в браузере, и запрос на разрешения получения информации о ваших друзьях. После

подтверждения всех запросов, вас переадресует на страницу, в адресной строке которой, будет указан access_token для работы с API VK.

4. Введите access_token в поле «**access token**» и нажмите на кнопку «**Определить друзей**». Загрузятся данные о ваших друзьях и построится матрица смежности.
5. Нажмите кнопку «**Определить связи**» и запуститься алгоритм кластеризации результат которого отобразится на странице в виде графа.

Для визуализации данных используется D3 JavaScript-библиотека. В программе так же используются: библиотека underscore.js – для работы с множествами (пересечение, разность) и CSS фреймворк Materialize – для стилей визуальных компонент.

Результат выполнения программы для конкретного пользователя (автора диплома) отображен на Рис.1.

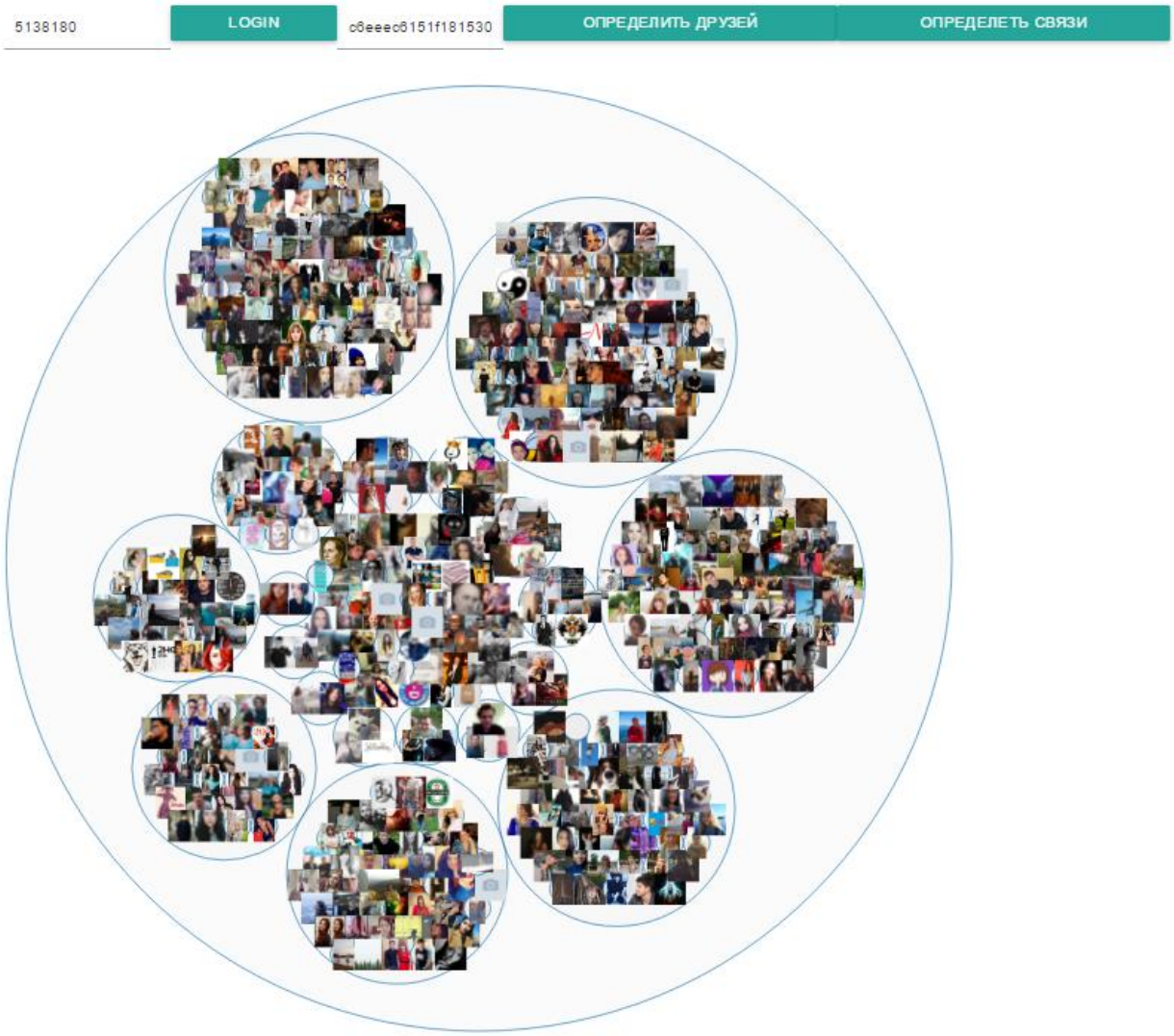

Рис.1. Результат применения жадного алгоритма

533 друга автора были разбиты на 8 основных (достаточно больших) кластеров и несколько кластеров с малым количеством пользователей. Полученные результаты соответствуют ожидаемому разбиению друзей на группы. Например, в один из кластеров попали все пользователи, которые являются друзьями автора по школе, в другой – попали друзья из детского сада и т.д. Программа также тестировалась на множестве друзей других пользователей «Вконтакте», с порядком данных, начиная от 200, где были получены схожие ожидаемые результаты.

Таким образом, можно сделать вывод о том, что рассмотренный алгоритм применим для кластеризации данных с порядком, схожим с указанным в нашем реализованном примере.

Программный код реализации алгоритма предоставлен в Приложении1.

Глава 2: «Осторожный» алгоритм

Рассмотрим ещё один алгоритм кластеризации на графах с качественной информацией. В статье «Корреляция кластеризации» («Correlation Clustering») [8] авторы рассматривают две задачи: MaxAgree (максимизация соглашений внутри кластеров и минимизация рассогласований между ними) и MinDisagree (минимизация рассогласований внутри кластеров и максимизация соглашений между кластерами). Задачи являются эквивалентными с точки зрения оптимальности, но различными с точки зрения аппроксимации.

2.1. Описание Алгоритма

В отличие от предыдущего алгоритма, «Осторожный» алгоритм производит кластеризацию на полных графах. Пусть G=(V,E) – неориентированный невзвешенный полный граф, состоящий из двух множеств: множества V≠Ø, элементы которого называются вершинами или узлами графа, и множества E неупорядоченных пар из множества V, элементы которого называются рёбрами или связями.

Через e(u,v) обозначим метку (+ или −) ребра (u,v): e(u,v)=+, если вершины обладают свойством схожести, e(u,v)= −, если нет.

Через $N^+(u)=\{u\}\cup\{v:e(u,v)=+\}$ и $N^-(u)=\{v:e(u,v)=-\}$ обозначают "положительных" и "отрицательных" соседей вершины u соотвественно.

Определение [8]. Вершина u называется δ-хорошей по отношению к C, где C⊂V, если она удовлетворяет следующим условиям:

- $|N^+(u)\cap C|\geq(1-\delta)|C|$
- $|N^+(u)\cap(V\setminus C)|\leq\delta|C|$

Набор C называется δ-чистым, если все $u\in C$ являются δ-хорошими по отношению к C.

Перейдем к описанию алгоритма.

«Осторожный» алгоритм (Algorithm Cautious):

1)Выбирается произвольная вершина u и выполняются следующие операции:

(a) Пусть A(u)=N$^+$(u)

(b) (Шаг удаления вершины): Пока $\exists x \in$A(u) такой, что x является «3δ-bad» по отношению к A(u), то A(u)=A(u)\{x}

(c) (Шаг возвращения вершины): Пусть Y={y| $y \in$V, y является «7δ-good» по отношению к A(u)}. Тогда A(u)=A(u)UY

2) Удаляем элементы полученного множества A(u) из множества всех вершин графа и повторяем процедуру до тех пор, пока ни одной вершины не останется или пока все преобразованные A(u) не останутся пустыми. В последнем случае оставшиеся на выходе вершины являются одиночными узлами.

Авторы предлагают значение параметра δ заданным и равным по величине 1/44.

2.2. Реализация алгоритма

Для рассматриваемой задачи кластеризации данных, полученных из социальной сети «Вконтакте» будем считать, что вершины u и v обладают свойством «схожести» (e(u,v)=+), если между ними имеется дружественная связь, в противном случае, считаем, что e(u,v)= $-$.

Структура программы:

Структура программы совпадает со структурой программы, описанной в первой главе, с той лишь разницей, что в main.js по нажатию на "Определить связи" вызывается другой метод кластеризации, и в algorithms,js реализован «осторожный» алгоритм.

Описание работы программы:

1. Откройте файл **index.html** в любом из браузеров.
2. В поле «**Client Id**» введите Id вашего приложения, зарегистрированного в VK. Узнать его или создать новое можно на странице: https://vk.com/apps?act=manage
3. Нажмите на кнопку «**Login**» и откроется окно с запросом логина и пароля для входа в VK, если они у вас не закешированны в браузере, и запрос на разрешения получения информации о ваших друзьях. После подтверждения всех запросов, вас переадресует на страницу, в адресной строке которой, будет указан access_token для работы с API VK.
4. Введите access_token в поле «**access token**» и нажмите на кнопку «**Определить друзей**». Загрузятся данные о ваших друзьях и построится матрица смежности.
5. Введите значение δ и нажмите "Определить связи"
6. Можно изменить значение δ и нажать "Определить связи".

Полученные результаты:

При рекомендуемом значении параметра $\delta=\frac{1}{44}$ (Рис.2.) в результате кластеризации 533 друзей размер максимально полученного сообщества оказался равным трём. А, к примеру, при использовании параметра $\delta=\frac{1}{4}$ (Рис. 3.), множество друзей оказалось разбитым на два сообщества. Из полученных результатов можно сделать вывод, что «осторожный» алгоритм плохо работает на графах с количеством вершин равным 200-600. Таким образом, можно сделать вывод о том, что рассмотренный алгоритм не применим для кластеризации данных с порядком, схожим с указанным в нашем реализованном примере.

Программный код реализации алгоритма предоставлен в Приложении 2.

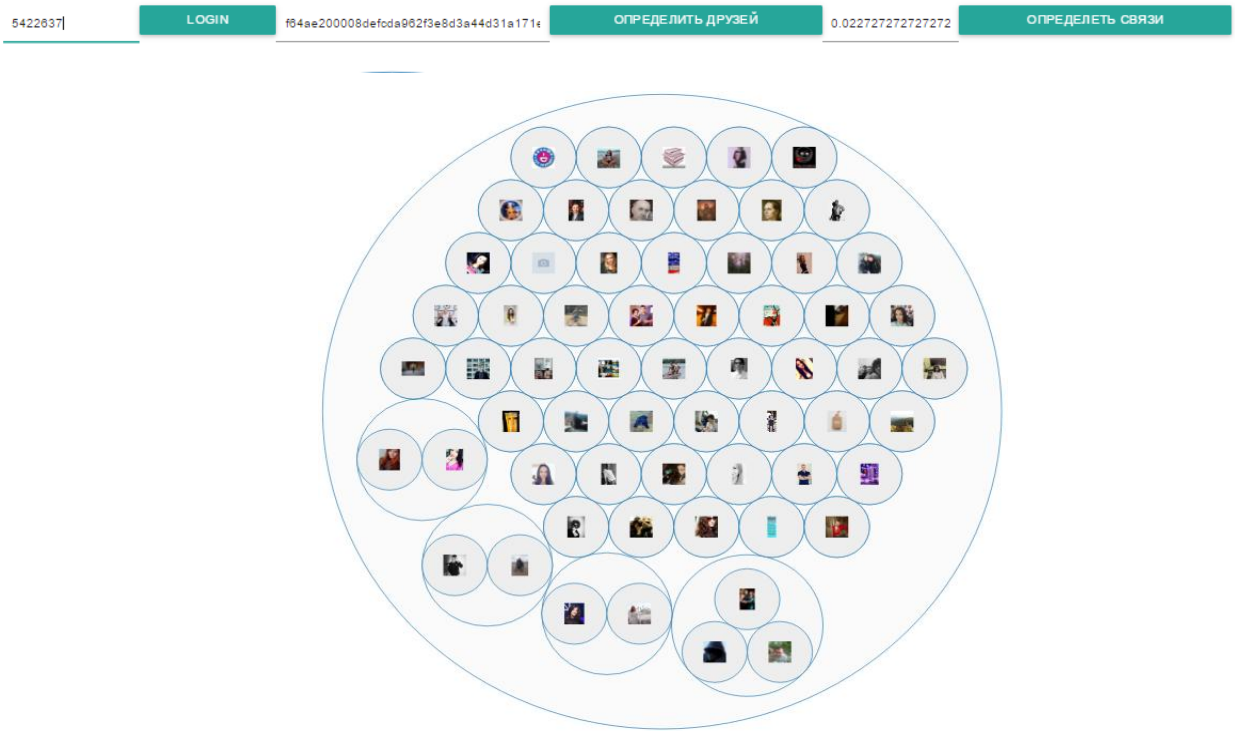

Рис.2. Результат применения «осторожного» алгоритма при $\delta=\frac{1}{44}$

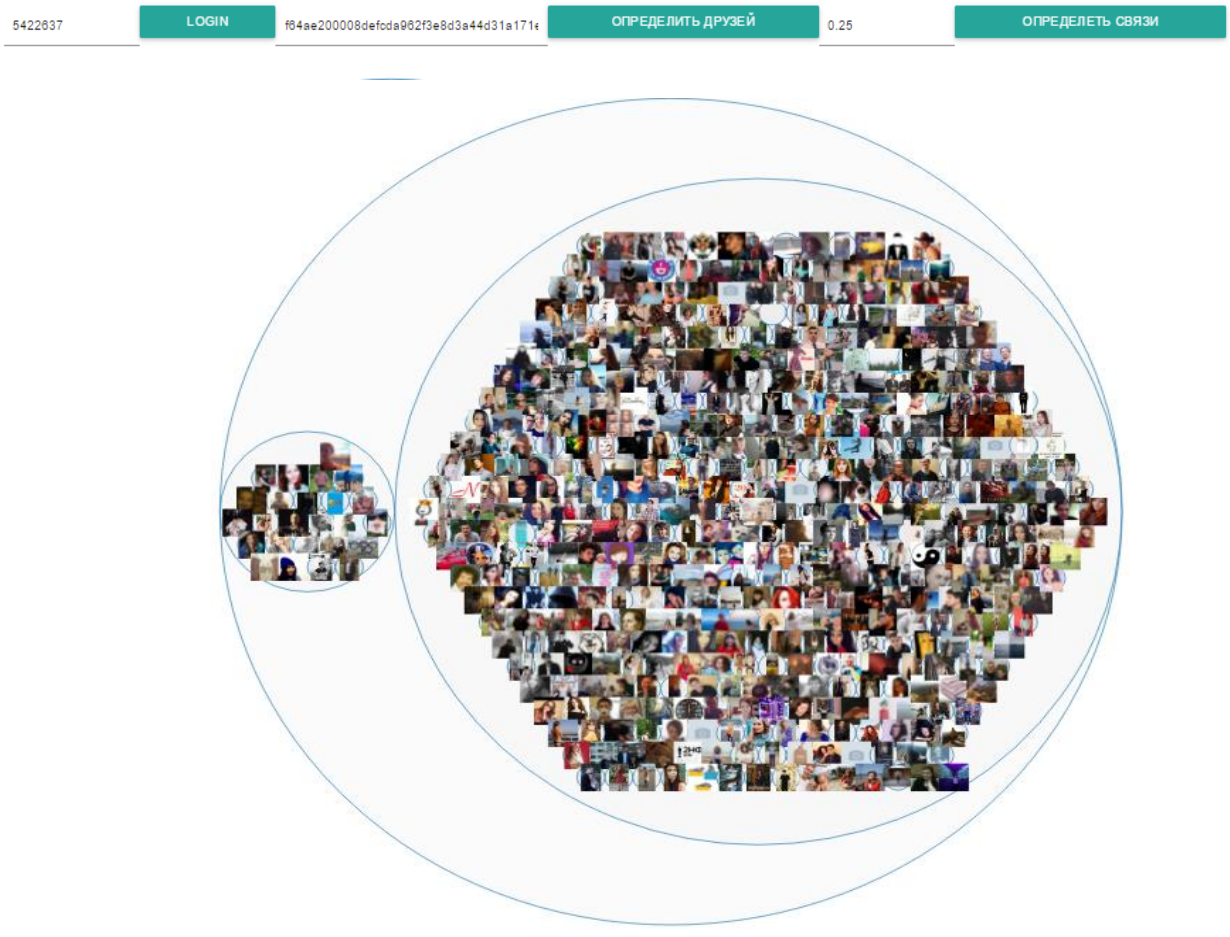

Рис.3. Результат применения «осторожного» алгоритма при $\delta=\frac{1}{4}$

Заключение

В работе были реализованы два алгоритма кластеризации в программной среде. Полученные программы были применены к реальным данным из социальной сети «Вконтакте». Проведен анализ полученных результатов и сравнение работы двух рассмотренных алгоритмов.

В результате работы алгоритма жадной оптимизации было получено разбиение на кластеры, отражающие принадлежность друзей к группам знакомств, сформированных по принадлежности к социальным группам. Их условно можно назвать «университет», «школа», «детский сад», «танцевальная школа» и так далее. Работа программы была протестирована для пользователей с числом друзей от 200 до 600. В каждом из случаев, полученные результаты хорошо отображали ожидаемые.

В результате работы «осторожного» алгоритма было получено разбиение на кластеры. Однако взаимосвязь между друзьями, соответствующими вершинам одного кластера данного разбиения, определить не удалось. И результат работы данного алгоритма не нашел практического применения для задач кластеризации с числом узлов 200-600.

Таким образом, было установлено, что алгоритм жадной оптимизации может быть успешно применен в практических исследованиях для выявления неявных сообществ среди пользователей социальной сети.

Список литературы

[1] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks.Proceedings of the National Academy of Sciences, 99(12):7821–7826, 2002.

[2] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment 2008(10):P10008, 2008. http://www.arxiv.org/abs/0803.0476

[3] Usha Nandini Raghavan, R´eka Albert, and Soundar Kumara. Near linear time algorithm to detect community structures in largscale networks.Physical Review E, 76(3):036106, 2007. http://www.arxiv.org/abs/0709.2938

[4] B. W. Kernighan and S. Lin, An efficient heuristic procedure for partitioning graphs.Bell System Technical Journal 49, 291–307 (1970).

[5]] M. Fiedler, Algebraic connectivity of graphs.Czech.Math. J.23, 298–305 (1973).

[6] A. Pothen, H. Simon, and K.-P. Liou, Partitioning sparse matrices with eigenvectors of graphs. SIAM J. Matrix Anal. Appl. 11, 430–452 (1990).

[7] J. Scott,Social Network Analysis: A Handbook. Sage, London, 2nd edition (2000).

[8] N .Bansal, A .Blum, S .Chawla, Correlation clustering, in: Proceedings of 43rd FOCS, 2002, pp .238–247

[9] M. Girvan and M. E. J. Newman, Community structure in social and biological networks. Proc. Natl. Acad. Sci. USA 99, 7821–7826 (2002).

[10] M. E. J. Newman and M. Girvan, Finding and evaluating community structure in networks. Phys. Rev. E 69, 026113 (2004).

[11] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, Defining and identifying communities in networks. Proc. Natl. Acad. Sci. USA 101, 2658–2663 (2004).

[12] M. E. J. Newman, Fast algorithm for detecting community structure in networks. Phys. Rev. E 69, 066133 (2004).

[13]] F. Wu and B. A. Huberman, Finding communities in linear time: A physics approach. Eur. Phys. J. B 38, 331–338 (2004).

Приложение 1.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<script src="js\lib\jquery.js"></script>
<script src="js\lib\materialize.js"></script>
<script type="text/javascript" src="js\lib\underscore.js"></script>
<script type="text/javascript" src="js\lib\d3.v3.js" charset="utf-8"></script>
<script type="text/javascript" src="js\vkFriends.js"></script>
<script type="text/javascript" src="js\graph.js"></script>
<script type="text/javascript" src="js\algorithms.js"></script>
<script type="text/javascript" src="js\main.js"></script>
<link rel="stylesheet" href="css\materialize.css">
<link rel="stylesheet" href="css\style.css">
<body>
  <div class="progress">
      <div class="indeterminate"></div>
  </div>
  <div class="row alert alert-success"><span></span></div>
  <div class="row alert alert-error"><span></span></div>
  <div class="row">
    <input class="col s2 m1" id="clientId" placeholder="Client Id"></input>
    <a class="col s2 m1 waves-effect waves-light btn" id="login" href="">Login</a>
    <input class="col s4 m2" id="token" placeholder="access token"></input>
    <a id="findFriends" class="col s2 m2 waves-effect waves-light btn">Определить
друзей</a>
    <a id="getRelations" class="col s2 m2 waves-effect waves-light btn margin-
left">Определеть связи</a>
  </div>
  <div id="graphs">
    <div id="circlePacking"></div>
    <div id="forceLayout" class="hide"></div>
  </div>
</body>
```

main.js

```
$(document).ready(function() {

  function sleep(milliseconds) {
    var start = new Date().getTime();
    for (var i = 0; i < 1e7; i++) {
      if ((new Date().getTime() - start) > milliseconds){
        break;
      }
    }
  };

  var showAlert = function(message, type){
    $('.alert').hide();
    $('.alert-'+ type + ' span').html(message);
    $('.alert-'+ type).show();
  };

  var hideAlerts = function(){
    $('.alert').hide();
  };
```

```
          var showProgress = function(){
            $('.progress').show();
          };

          var hideProgress = function(){
            $('.progress').hide();
          }

          $('#login').click(function() {
            var link = 'https://oauth.vk.com/authorize?client_id=' + $('#clientId').val()
+
'&scope=friends,offline&redirect_uri=https://oauth.vk.com/blank.html&display=page&v=5.50&
response_type=token';
            window.open(link);
          });

          var friends;
          $('#findFriends').click(function() {
            hideAlerts();
            showProgress();
            findFriends($('#token').val())
              .done(function (data){
                friends = $.grep(data.response, function(item, i) {
                  return !item.deactivated;
                });

                var uids = $.map(friends, function(friend) {return friend.uid; });
                $.each(friends, function(index, friend) {
                  if(index > 0 && (index % 2 === 0))
                    sleep(1000);

                  findCommonFriends($('#token').val(),friend.uid)
                    .done(function(data){
                      if(data.error) {
                        showAlert(data.error.error_msg, 'error');
                      }
                      friend.friends = $.grep(data.response, function(item, i) {
                        return _.contains(uids, item);
                      });
                    })
                    .fail(function (jqXHR, textStatus, errorThrown) {
                      showAlert(jqXHR.responseText, 'error');
                    });
                });

                showAlert('Найдено друзей: <strong>' + friends.length + '</strong>.',
'success');
                hideProgress();
              })
              .fail(function (jqXHR, textStatus, errorThrown) {
                showAlert(jqXHR.responseText, 'error');
                hideProgress();
              });
          });

          $('#getRelations').click(function() {

            hideAlerts();
            if(!friends || friends.length == 0) {
              showAlert('Не удалось определить друзей.', 'error');
              return;
            }

            showProgress();
```

```
            var users = $.map(friends, function(friend) { return { id: friend.uid,
neighbors: friend.friends} });
            var communities = Algorithms.findCommunity(users);
            var groups = {name: "Пользователь", size: communities.length, children: []};
            $.each(communities, function(index, community) {
              var group = { name: community.length, size: community.length };
              group.children = $.map(community, function(id) {
                var user = friends.find(function(element, index, array) { return
element.uid == id; });
                user.name = (user.last_name || '' ) + ' ' + (user.first_name || '');
                user.size = 100;
                user.img = user.photo_200_orig;
                return user;
              });
              groups.children.push(group);
            });

            showCirclePacking(groups);
            $('.progress').hide();
          });

          $('#graphTypes li a').click(function(){
            $('#graphs div').hide();
            var type = $(this).data('type');
            var navItem = $('#' + type);
            navItem.show();
            $('#graphTypes li').removeClass('active');
            $(this).parent().addClass('active');
          });
        });
```

vkFriends.js

```
        var findFriends = function (token) {
               return $.ajax({
                type: 'GET',
                      dataType: 'jsonp',
                data: {},
                url:
'https://api.vk.com/method/friends.get?order=name&fields=nickname,photo_200_orig,educatio
n&access_token=' + token
            });
        };

        var findCommonFriends = function(token, friendUid) {
               return $.ajax({
                       type: 'GET',
                             dataType: 'jsonp',
                       data: {},
                       async: false,
                       url:   'https://api.vk.com/method/friends.getMutual?target_uid='   +
friendUid + '&access_token=' + token
            });
        };
```

algorithms.js

```
        Algorithms = (function() {
               var    nodes = [];
               var neighbors = [];
               var graph;
```

```
		var initGrapth = function(_graph){
			graph = _graph;
			$.each(graph, function(index, node) {
				nodes.push(node.id);
				neighbors[node.id] = node.neighbors.slice();
			});
		};

		var findCommunity = function(_graph){
			if(!graph)
				initGrapth(_graph);

			var m = 0;
			$.each(nodes, function(index, node) {
				m += neighbors[node] ? neighbors[node].length : 0;
			});
			m /= 2;

			var a = {};
			$.each(nodes, function(index, node) {
				a[node] = neighbors[node] ? neighbors[node].length / (2 * m) :
0;
			});

			var deltaQ = {};
			$.each(nodes, function(i, nodeI){
				deltaQ[nodeI] = {nodes: [nodeI]};
				$.each(nodes, function(j, nodeJ){
					deltaQ[nodeI][nodeJ] = (neighbors[nodeI].indexOf(nodeJ)
!= -1) ? 1/(2*m) - a[nodeI] * a[nodeJ] : 0;
				});
			});

			while (true) {
				var maxI = -1;
				var maxJ = -1;
				var maxDeltaQ = 0;
				for(var nodeI in deltaQ) {
					nodeI = parseInt(nodeI);
					for(var nodeJ in deltaQ){
						nodeJ = parseInt(nodeJ);
						if(nodeI == nodeJ)
							continue;

						if(deltaQ[nodeI][nodeJ] >= maxDeltaQ){
							maxDeltaQ = deltaQ[nodeI][nodeJ];
							maxI = nodeI;
							maxJ = nodeJ;
						}
					}
				}

				if(maxDeltaQ <= 0 || Object.keys(deltaQ).length === 1)
					break;

				for(var k in deltaQ) {
					k = parseInt(k);
					if( k == maxI || k == maxJ)
						continue;

					var isNeighborsWithI = isNeighbors(deltaQ[maxI].nodes,
deltaQ[k].nodes);
					var isNeighborsWithJ = isNeighbors(deltaQ[maxJ].nodes,
```

```
deltaQ[k].nodes);

                                    deltaQ[maxJ][k] =  isNeighborsWithI && isNeighborsWithJ
                                           ? deltaQ[maxI][k] + deltaQ[maxJ][k]
                                           : isNeighborsWithI && !isNeighborsWithJ
                                                  ? deltaQ[maxI][k] - 2 * a[maxJ] * a[k]
                                                  : isNeighborsWithJ && !isNeighborsWithI ?
deltaQ[maxJ][k] - 2 * a[maxI] * a[k] : 0;

                                    deltaQ[k][maxJ] = deltaQ[maxJ][k];
                                    var row = deltaQ[k];
                                    delete row[maxI];
                             }
                             deltaQ[maxJ].nodes                                                 =
deltaQ[maxJ].nodes.concat(deltaQ[maxI].nodes);
                             delete deltaQ[maxJ][maxI];
                             delete deltaQ[maxI];
                             a[maxJ] = a[maxJ] + a[maxI];
                             a[maxI] = 0;
                      }

                      var communities = [];
                      for(var item in deltaQ) {
                             communities.push(deltaQ[item].nodes);
                      };

                      return communities;
               };

               var isNeighbors = function(source, targer) {
                      var res = false;
                      $.each(source, function(index, sourceNode) {
                             if (_.intersection(neighbors[sourceNode], targer).length > 0)
{
                                    res = true;
                                    return false;
                             }
                      });

                      return res;
               };

               return {
                      findCommunity: findCommunity
               }
        }());
```

graph.js

```
        var showCirclePacking = function(data) {
          var diameter = 960,
              format = d3.format(",d");

          var pack = d3.layout.pack()
              .size([diameter - 4, diameter - 4])
              .value(function(d) { return d.size; });

          $('#circlePacking').empty();
          var svg = d3.select("#circlePacking").append("svg")
              .attr("width", diameter)
              .attr("height", diameter)
            .append("g")
```

```
            .attr("transform", "translate(2,2)");

        var node = svg.datum(data).selectAll(".node")
            .data(pack.nodes)
          .enter().append("g")
            .attr("class", function(d) { return d.children ? "node" : "leaf node"; })
            .attr("transform", function(d) { return "translate(" + d.x + "," + d.y +
")"; });

        node.append("title")
            .text(function(d) { return d.name; });

        node.append("circle")
            .attr("r", function(d) { return d.r; })
            .style("fill", "#eee");

        var images = node.filter(function(d) { return !d.children;
}).append("svg:image")
            .attr("xlink:href", function(d) { return d.photo_200_orig;})
            .attr("x", function(d) { return -16;})
            .attr("y", function(d) { return -16;})
            .attr("height", 32)
            .attr("width", 32);

        // make the image grow a little on mouse over and add the text details on click
        var setEvents = images
                .on( 'mouseenter', function() {
                  // select element in current context
                  d3.select( this )
                    .transition()
                    .attr("xlink:href", function(d) { return d.photo_200_orig;})
                    .attr("x", function(d) { return -128;})
                    .attr("y", function(d) { return -128;})
                    .attr("height", 256)
                    .attr("width", 256)
                    .style("z-index", 10);
                })
                // set back
                .on( 'mouseleave', function() {
                  d3.select( this )
                    .transition()
                    .attr("xlink:href", function(d) { return d.photo_200_orig;})
                    .attr("x", function(d) { return -16;})
                    .attr("y", function(d) { return -16;})
                    .attr("height", 32)
                    .attr("width", 32)
                    .style("z-index", 1);
                });

        d3.select(self.frameElement).style("height", diameter + "px");
      };
```

Приложение 2.

index.html

```
<!DOCTYPE html>
<meta charset="utf-8">
<script src="js\lib\jquery.js"></script>
<script src="js\lib\materialize.js"></script>
<script type="text/javascript" src="js\lib\underscore.js"></script>
<script type="text/javascript" src="js\lib\d3.v3.js" charset="utf-8"></script>
<script type="text/javascript" src="js\vkFriends.js"></script>
<script type="text/javascript" src="js\graph.js"></script>
<script type="text/javascript" src="js\algorithms.js"></script>
<script type="text/javascript" src="js\main.js"></script>
<link rel="stylesheet" href="css\materialize.css">
<link rel="stylesheet" href="css\style.css">
<body>
  <div class="progress">
      <div class="indeterminate"></div>
  </div>
  <div class="row alert alert-success"><span></span></div>
  <div class="row alert alert-error"><span></span></div>
  <div class="row">
    <input class="col s2 m1" id="clientId" placeholder="Client Id"></input>
    <a class="col s2 m1 waves-effect waves-light btn" id="login" href="">Login</a>
    <input class="col s4 m2" id="token" placeholder="access token"></input>
    <a id="findFriends" class="col s2 m2 waves-effect waves-light btn">Определить
друзей</a>
    <input id="delta" class="col s2 m1" placeholder="delta"
value="0.0227272727272727"></input>
    <a id="getRelations" class="col s2 m2 waves-effect waves-light btn">Определеть
связи</a>
  </div>
  <div id="graphs">
    <div id="circlePacking"></div>
    <div id="forceLayout" class="hide"></div>
  </div>
</body>
```

main.js

```
$(document).ready(function() {

  function sleep(milliseconds) {
    var start = new Date().getTime();
    for (var i = 0; i < 1e7; i++) {
      if ((new Date().getTime() - start) > milliseconds){
        break;
      }
    }
  };

  var showAlert = function(message, type){
    $('.alert').hide();
    $('.alert-'+ type + ' span').html(message);
    $('.alert-'+ type).show();
  };

  var hideAlerts = function(){
```

```
          $('.alert').hide();
        };

        var showProgress = function(){
          $('.progress').show();
        };

        var hideProgress = function(){
          $('.progress').hide();
        }

        $('#login').click(function() {
          var link = 'https://oauth.vk.com/authorize?client_id=' + $('#clientId').val()
+
'&scope=friends,offline&redirect_uri=https://oauth.vk.com/blank.html&display=page&v=5.50&
response_type=token';
          window.open(link);
        });

        var friends;
        $('#findFriends').click(function() {
          hideAlerts();
          showProgress();
          findFriends($('#token').val())
            .done(function (data){
              friends = $.grep(data.response, function(item, i) {
                return !item.deactivated;
              });

              var uids = $.map(friends, function(friend) {return friend.uid; });
              $.each(friends, function(index, friend) {
                if(index > 0 && (index % 2 === 0))
                  sleep(1000);

                findCommonFriends($('#token').val(),friend.uid)
                  .done(function(data){
                    if(data.error) {
                      showAlert(data.error.error_msg, 'error');
                    }
                    friend.friends = $.grep(data.response, function(item, i) {
                      return _.contains(uids, item);
                    });
                  })
                  .fail(function (jqXHR, textStatus, errorThrown) {
                    showAlert(jqXHR.responseText, 'error');
                  });
              });

              showAlert('Найдено друзей: <strong>' + friends.length + '</strong>.',
'success');
              hideProgress();
            })
            .fail(function (jqXHR, textStatus, errorThrown) {
              showAlert(jqXHR.responseText, 'error');
              hideProgress();
            });
        });

        $('#getRelations').click(function() {
          hideAlerts();
          if(!friends || friends.length == 0) {
            showAlert('Не удалось определить друзей.', 'error');
            return;
          }
```

```
            showProgress();
             var delta = parseFloat($('#delta').val());
             var groups = Algorithms.oldAlgorithm(friends, delta);
             showCirclePacking(groups);

            $('.progress').hide();
          });

          $('#graphTypes li a').click(function(){
            $('#graphs div').hide();
            var type = $(this).data('type');
            var navItem = $('#' + type);
            navItem.show();
            $('#graphTypes li').removeClass('active');
            $(this).parent().addClass('active');
          });
        });
```

vkFriends.js

```
        var findFriends = function (token) {
               return $.ajax({
                type: 'GET',
                      dataType: 'jsonp',
                data: {},
                url:
'https://api.vk.com/method/friends.get?order=name&fields=nickname,photo_200_orig,educatio
n&access_token=' + token
            });
        };

        var findCommonFriends = function(token, friendUid) {
               return $.ajax({
                        type: 'GET',
                              dataType: 'jsonp',
                        data: {},
                        async: false,
                        url:   'https://api.vk.com/method/friends.getMutual?target_uid='    +
friendUid + '&access_token=' + token
            });
        };
```

algorithms.js

```
        Algorithms = (function() {
               var oldAlgorithm = function(friends, delta) {
                      var isGood = function(users, n, c, delta) {
                             if(_.intersection(n, c).length < (1 - delta) * c.length)
                                    return false;

                        return _.intersection(n, _.difference(users, c)).length <= delta *
c.length;
               };

               var used = [];
                   var groups = {name: "Пользователь", children: []};
                   var index = 0;
                   var users = $.map(friends, function(friend) { return friend.uid; });
                   while(users && users.length > 0 && index < users.length) {
```

```
                    var v = users[index];
                    var  f  =  friends.find(function(element,  index,  array)  {  return
element.uid == v; }).friends || [];
                    var a = $.grep(f, function(id){
                     return !_.contains(used, id);
                    });
                    a.push(v);
                    var removed = true;
                    while (removed) {
                      removed = false;
                      $.each(a, function(index, x) {
                            var fr = friends.find(function(element, index, array) { return
element.uid == x; }).friends || [];
                            var n = $.grep(fr, function(id) {
                                  return !_.contains(used, id);
                            });
                            n.push(x);
                            if (!isGood(users, n, a, 3*delta)) {
                                  var i = a.indexOf(x);
                            if (i > -1) {
                                        a.splice(i, 1);
                            }

                                 removed = true;
                                 return false;
                            }
                      });
                    }

                    $.each(users, function(index, user) {
                      if (_.contains(a, user))
                        return true;

                            var fr = friends.find(function(element, index, array) { return
element.uid == user; }).friends || [];
                      var n = $.grep(fr, function(id) {
                            return !_.contains(used, id);
                      });

                      n.push(user);
                      if (isGood(users, n, a, 7*delta))
                        a.push(user);
                    });

                    if (a && a.length > 0) {
                      var group = {};
                      group.name = a.length;
                      group.children = $.map(a, function(id) {
                        var i = users.indexOf(id);
                          if (i > -1) {
                            users.splice(i, 1);
                            used.push(id);
                          }
                        var user = friends.find(function(element, index, array) { return
element.uid == id; });
                          user.name = (user.last_name || '' ) + ' ' + (user.first_name ||
'');

                        user.size = 100;
                        return user;
                      });
                      groups.children.push(group);
                      group.size = a.length;
                      index = 0;
                    }
```

```
                    else {
                      index++;
                    }
                  }

                  return groups;
                };

                return {
                       oldAlgorithm: oldAlgorithm
                }
        }());
```

graph.js

```
        var showCirclePacking = function(data) {
          var diameter = 960,
              format = d3.format(",d");

          var pack = d3.layout.pack()
              .size([diameter - 4, diameter - 4])
              .value(function(d) { return d.size; });

          $('#circlePacking').empty();
          var svg = d3.select("#circlePacking").append("svg")
              .attr("width", diameter)
              .attr("height", diameter)
            .append("g")
              .attr("transform", "translate(2,2)");

          var node = svg.datum(data).selectAll(".node")
              .data(pack.nodes)
            .enter().append("g")
              .attr("class", function(d) { return d.children ? "node" : "leaf node"; })
              .attr("transform", function(d) { return "translate(" + d.x + "," + d.y +
")"; });

          node.append("title")
              .text(function(d) { return d.name; });

          node.append("circle")
              .attr("r", function(d) { return d.r; })
              .style("fill", "#eee");

          var     images     =     node.filter(function(d)     {     return     !d.children;
}).append("svg:image")
              .attr("xlink:href",  function(d) { return d.photo_200_orig;})
              .attr("x", function(d) { return -16;})
              .attr("y", function(d) { return -16;})
              .attr("height", 32)
              .attr("width", 32);

          // make the image grow a little on mouse over and add the text details on click
          var setEvents = images
                  .on( 'mouseenter', function() {
                    // select element in current context
                    d3.select( this )
                      .transition()
                      .attr("xlink:href",  function(d) { return d.photo_200_orig;})
                      .attr("x", function(d) { return -128;})
                      .attr("y", function(d) { return -128;})
                      .attr("height", 256)
```

```
                .attr("width", 256)
                .style("z-index", 10);
            })
            // set back
            .on( 'mouseleave', function() {
              d3.select( this )
                .transition()
                .attr("xlink:href",  function(d) { return d.photo_200_orig;})
                .attr("x", function(d) { return -16;})
                .attr("y", function(d) { return -16;})
                .attr("height", 32)
                .attr("width", 32)
                .style("z-index", 1);
            });

  d3.select(self.frameElement).style("height", diameter + "px");
};
```