АЛГОРИТМЫ ВЫДЕЛЕНИЯ ГРУПП ОБЩЕНИЯ[1]

Лещёв Д.А.[2] , Сучков Д.В.[3], Хайкова С.П.[4], Чеповский А.А.[5]

Цель статьи: *разработка методов анализа графа взаимодействующих объектов на основе выделения неявных сообществ с целью решения задач поиска близости профилей и обмена, распространения информации между объектами.*

Метод: *импорт данных из социальных сетей с последующим построением взвешенного графа на основе выбранных атрибутов и весовой функции, соответствующей исходной задаче; выделение сообществ на построенном взвешенном графе и сравнение получаемых разбиений с результатами классических алгоритмов.*

Полученный результат: *разработаны и реализованы алгоритмы построения графа и импорта атрибутов пользователей, созданы соответствующие весовые функции, сконструированы структуры хранения данных, исследован и имплементирован алгоритм Louvain для взвешенных графов с учетом используемых структур данных, добавлены дополнительные гиперпараметры, повышающие качество разбиения графа на неявные сообщества пользователей. На примере социальной сети ВКонтакте построены специальные алгоритмы прохода по базе данных, программно-аппаратный комплекс применен на реальных данных, произведено сравнение результатов работы с классическими алгоритмами выделения сообществ.*

Ключевые слова: *структура графа, анализ социальной сети, выделение сообществ*

DOI: *10.21681/2311-3456-2019-4-61-71*

1. Введение

В данной работе авторами предложен метод построения и использования взвешенных графов для известной задачи выявления неявных сообществ в сетях взаимодействующих объектов[1-5]. В качестве базового примера продемонстрирован процесс импорта и обработки данных из популярной социальной сети ВКонтакте. При этом выделяемые сообщества отражают близость пользователей в контексте одной из актуальных задач, что обеспечивается применением соответствующей весовой функции на графе. Была использована следующая классификация исходной задачи:

- задача общей близости профилей пользователей;
- задача определения степени (очного) знакомства пользователей;
- задача обмена, распространения информации между пользователями.

2. Импорт данных для социальных графов на примере сети ВКонтакте

Социальные сети содержат в себе большое количество информации, которую можно использовать в исследовательских целях[6-8], а также служат важным инструментом для взаимодействия людей. Подтверждением этому служит социальная сеть ВКонтакте (http://vk.com), в которой люди ежедневно отправляют 10 миллиардов сообщений (http://vk.com/about). С целью подготовки информации к анализу для начала необходимо импортировать данные. ВКонтакте предоставляет специальный интерфейс (API), который позволяет получать информацию из базы данных vk.com. Далее рассмотрим некоторые особенности API ВКонтакте и виды графов, которые можно построить на основе данных из этой социальной сети.

2.1. Особенности API ВКонтакте

Для того, чтобы обращаться к базе данных vk.com, необходимо использовать ключ доступа (токен), своего рода «подпись» того, от чьего имени осуществляются запросы, и какие права доступа он выдал. Ключи доступа бывают трех видов:

- Ключ доступа пользователя
- Ключ доступа сообщества
- Сервисный ключ доступа

Ключ доступа пользователя необходим для работы со всеми методами API ВКонтакте, за исключением методов секции *Secure*. Используя такой ключ доступа, работа с API идет от имени пользователя, получившего ключ. Соответственно, можно получить общедоступную информацию, а также информацию, к которой данный пользователь имеет доступ, также возможно совершать действия от имени этого пользователя. Ключ доступа сообщества

1 Работа выполнена при финансовой поддержке РФФИ в рамках научных проектов № 16-29-09546, № 18-00-00606 (18-00-00233) и № 19-07-00806.

2 Лещёв Дмитрий Андреевич, бакалавр образовательной программы «Прикладная математика» МИЭМ НИУ ВШЭ, Москва, Россия. E-mail: daleschyov@edu.hse.ru.

3 Сучков Даниил Викторович, бакалавр образовательной программы «Прикладная математика» МИЭМ НИУ ВШЭ, Москва, Россия. E-mail: dvsuchkov@edu.hse.ru.

4 Хайкова Светлана Павловна, бакалавр образовательной программы «Прикладная математика» МИЭМ НИУ ВШЭ, Москва, Россия. E-mail: spkhaykova@yandex.ru.

5 Чеповский Александр Андреевич, кандидат физико-математических наук, доцент, Департамент прикладной математики МИЭМ НИУ ВШЭ, Москва, Россия. E-mail: aachepovsky@hse.ru.

обладает свойствами аналогично ключу доступа пользователя. Сервисный ключ доступа используется для запросов, которые не требуют авторизации пользователя или сообщества: методы категории *Secure* и, так называемые, открытые методы *(http://vk.com/dev/access_token)*. Авторами использовался сервисный ключ доступа, полученный для специального приложения ВКонтакте.

Для получения информации из базы данных vk.com используются https-запросы к специальному серверу. При этом синтаксис всех запросов и тип возвращаемых ими данных определены на стороне сервера. Для отправки запросов использовалась открытая библиотека Libcurl (*http://curl.haxx.se/libcurl/*). Ответом за запрос сервер всегда возвращает данные, представленные в формате JSON, который легко обработать.

2.2. Виды графов

Классический граф друзей, характеризующий связи пользователей вида «дружба», получается поиском в ширину от заданной вершины. Целесообразно либо заранее ограничивать глубину такого поиска при создании эго-графа, либо задавать вторую вершину, до которой идет поиск. Для нашей работы импортировались данные под более содержательные графы, позволяющие в итоге построить взвешенный граф пользователей, поэтому использовались доработанные варианты обхода графа: лайк-граф, граф комментариев и смешанный граф.

При сборе информации о связях между пользователями для **лайк-графа** задается начальная вершина, которая записывается на нулевой уровень, и глубина d, равная числу итераций, а также фиксированное количество последних постов (ps) и/или количество последних фотографий (ph) Количество итераций для лайков на постах как и для лайков на фотографиях тогда будет равно d+1. Для лайков на записях на первой итерации программа совершает запрос с методом wall.get и параметром count=ps для начальной вершины. Если профиль открыт и сервер возвращает массив с заданным количеством id записей на стене пользователя, то массив добавляется в очередь. Далее программа проходит по массиву с id записей и выполняет запрос likes.getList, параметры: type=post, owner_id = текущий id пользователя и item_id = текущий id записи на стене. Полученный массив с id пользователей записывается на первый уровень. А id профиля, для которого был совершен запрос wall.get, записывается в массив пройденных вершин. На n-ой итерации программа проходит по массиву с id пользователей на уровне n-1 и, если id пользователя не содержится в массиве пройденных вершин, совершает аналогичные действия.

На d+1-ой итерации программа проходит по массиву на уровне d и выполняет аналогичные запросы без проверки на наличие в массиве пройденных вершин. Если для текущего id пользователя был получен массив, содержащий id записей, то программа собирает информацию о пользователях, лайкнувших записи, оставляя из них только те, которые содержатся в массиве пройденных вершин. Все полученные id пользователей хранятся в целочисленном массиве.

Для лайков на фотографиях выполняются аналогичные операции. Вместо метода wall.get используется метод photos.get, а в параметры запроса likes.getList передается type=photo.

Для того, чтобы собрать информацию о связях между пользователями для **графа комментариев**, необходимо указать начальную вершину, представленную в виде идентификатора (id) пользователя, которая записывается на нулевой уровень, глубину d, на которую необходимо собрать данные, количество последних постов (ps), для которых будет собираться информация о комментариях и количество первых комментариев (cm). Количество итераций тогда будет d+1.

Ключевым для нашей работы был **смешанный граф**, который формировался в зависимости от интересующей информации: друзья, лайки на постах, количество постов (ps), лайки на фотографиях и/или комментарии под записями, количество фотографий (ph). Изначально фиксируется начальная вершина, которая записывается на нулевой уровень, а также глубина d. Количество итераций тогда будет равно количеству выбранных типов графов, умноженному на d+1. Вся информация о связях в случае смешанного графа хранилась в отдельных массивах по типу. Алгоритм выполняет следующие операции:

1. Если выбран тип связи «друг», то импортируется информация о друзьях.
2. Если выбран тип связи «лайки под постами», то импортируется информация о лайках. Массив идентификаторов записей для каждого пользователя сохраняется.
3. Если выбран тип связи «лайки под фотографиями», то импортируется информация о лайках.
4. Если выбран тип связи «комментарии под записями» в случае, когда также выбран тип «лайки под постами», программа не собирает идентификаторы записей, в ином случае – собирает. Импортируется информация о комментариях.

Таким образом, авторами были получены соответствующие атрибуты вершин и связей для дальнейшего построения взвешенного графа.

3. Построение взвешенного графа

Для выделения сообществ необходимо построить связный граф, вершинами которого будут являться пользователи, а ребра отвечают в соответствии с исходной задачей за степень, интенсивность или характер взаимодействия между пользователями за счет поставленных им в соответствие коэффициентов – весов. Причем, для более активно взаимодействующих вершин вес на связывающем их ребре будет большим, нежели для менее активно взаимодействующих. Таким образом, между каждыми двумя пользователями будет определено обратное взвешенное расстояние (оно может оказаться и нулевым).

3.1 Подсчёт весов на рёбрах

Подсчёт обратного взвешенного расстояния (далее просто – расстояния) между двумя пользователями происходит за счёт добавления заданных заранее показателей (весов) за каждую близость в информации двух пользователей. В зависимости от исходной задачи по

УДК 004.421.2:519.178 DOI: 10.21681/2311-3456-2019-4-61-71

виду выделяемых сообществ значения этих показателей варьируются. Так же авторы ввели гиперпараметр «регуляризация», выбираемый отдельно для каждого графа: если расстояние между вершинами меньше значения регуляризации, то расстояние принимается равным нулю, что помогает удалять случайные связи.

В случае социальной сети ВКонтакте для подсчета расстояния использовался следующий массив весов, состоящий из 11 элементов, полученных при импорте данных:

1. Родственники – значение этого веса, добавляемое в расстояние между двумя пользователями с одинаковыми фамилиями.
2. Страна – значение этого веса, добавляемое в расстояние между двумя пользователями с одинаковой указанной страной.
3. Город – значение этого веса, добавляемое в расстояние между двумя пользователями, у которых есть пересечение в списках текущего и родного города. Т.е. родной или текущий город первого пользователя совпадает с родным или текущим городом второго в совокупности.
4. Школа – значение этого веса, добавляемое в расстояние между двумя пользователями, у которых совпадают школы. Т.е. в списке школ первого пользователя есть хотя бы одна из списка школ второго.
5. Университет – значение этого веса, добавляемое в расстояние между двумя пользователями, у которых совпадают университеты. Аналогично школам.
6. Факультет – значение этого веса, добавляемое в расстояние между двумя пользователями, у которых совпадают факультеты.
7. Общие друзья – значение этого веса, умноженное на число общих друзей между пользователями, добавляется к расстоянию между ними.
8. Комментарии под постами – значение этого веса, умноженное на суммарное число комментариев первого пользователя под постами на стене второго и наоборот, добавляется к расстоянию между ними.
9. Лайки под постами – значение этого веса, умноженное на суммарное число лайков первого пользователя под постами на стене второго и наоборот, добавляется к расстоянию между ними.
10. Лайки под фотографиями – значение этого веса, умноженное на суммарное число лайков, поставленных первым пользователем под фотографиями второго, и наоборот, добавляется к расстоянию между ними.
11. Вес общих групп – значение, добавляемое к расстоянию на основании групп и интересных страниц, на которые подписаны оба пользователя. Вычисляется как сумма весов по всем общим группам пользователей $\sum_{k=0}^{n} W_k$, где n – число общих групп у пользователей, W_k – вес k-й группы.

Необходимо отдельно указать процедуру подсчета веса отдельной группы. При выборе функции для веса общей группы произвольной численности исходим из того, что группы с маленьким числом участников должны давать значительно больший вклад в расстояние, нежели крупные. Таким образом, хотелось выделять какие-то локальные группы, которые состоят из близких с точки зрения исходной задачи людей, например, группа школьного класса или группа футбольной команды. Для этого было сделано предположение, какое значение веса группы мы бы хотели получить для какого числа ее участников, и, используя различные способы аппроксимации функции одной переменной, получена формула для группы произвольного размера. Исходные веса групп, взятые за ориентир, указаны в таблице 1:

Таблица 1

Аппроксимация функции веса групп

Число пользователей сообществе	Значения функции (с точностью до 3-х знаков)
10	95.838
100	67.400
200	47.401
300	38.580
500	29.764
1000	20.932
10000	6.501
1000000	0.627

3.2 Весовые функции

Рассмотрим три весовые функции, которые в большей степени учитывают тот или иной параметр, что соответствует различным исходным задачам:

- **Weights_normal** – усреднённые веса, учитывающие каждый из признаков в среднем; тут нет уклона в сторону какого-либо признака, отвечает исходной задаче общей близости профилей пользователей.
- **Weights_friends** – веса с уклоном в сторону дружбы, отвечает исходной задаче определения степени знакомства пользователей.
- **Weights_internet_connections** – веса с уклоном во взаимодействие между пользователями в социальной сети, отвечает исходной задаче обмена/распространения информации.

У весовой функции **Weights_normal** за точку отсчёта было принято присвоение веса 1 для людей, указавших одинаковую страну проживания. Общий город добавлял к расстоянию 1.5, так как проживание в одном городе даёт больший вклад в вероятность общения людей, чем общая страна. Аналогично, общий университет добавляет 2, а общий факультет добавляет 5. Может показаться, что вес слишком сильно вырос, но это сделано для того, чтобы выделять сообщество людей, обучающихся в одной группе университета. Вес школы также равен 5, так как число людей на факультете сопоставимо с числом людей в средней школе. Стоит уточнить, что совпадение университета, школы и факультета учитывалось только для людей, у которых разница в возрасте не

больше 4 лет. Вес одного общего друга, равный 0.33, был выбран из логики, что 15 общих друзей – это примерно школьный класс или группа в университете, а там вес равен 5. Следовательно, вес друзей должен быть равен $\frac{5}{15}=0.33$.

Если один пользователь оставил пост на стене другого пользователя, то к их расстоянию добавляется 3. Если человек оставил пост у тебя на стене, и ты его не удалил, значит, вы вероятно общаетесь на каком-то уровне. Это большая близость, нежели общий город или университет.

Лайки от одного пользователя под постом или фотографией другого пользователя имеют вес 2.5. Вес меньше веса для постов, так как лайки менее значимы и более часто встречаемы, нежели посты на стене пользователя, плюс их могут оставить совершенно сторонние люди или же фейковые аккаунты. Однако, чаще всего это всё-таки друзья и знакомые, поэтому вес немного больше веса для общего университета. При совпадении фамилий расстоянию между людьми добавлялось 10. Величина этого веса связана с тем, что у родственников вероятнее всего не будет совпадения по школе/университету/факультету, т.к. это будут люди разных поколений (например, родители и дети), поэтому это было необходимо компенсировать. С другой стороны, братья и сестры могут иметь одинаковые фамилии и учиться вместе, но тут и логично выглядит высокое общее значение веса.

Аналогично задаются веса для функции **Weights_friends**, с различием только в весе одного общего друга – в данном случае он равен 1. Таким образом, появляется возможность выделять лидеров сообществ (людей, у которых максимальное число общих друзей с другими людьми из данного сообщества), что позволяет компенсировать недостаток информации, полученной от других пользователей данного сообщества, например, из-за закрытого аккаунта.

Для весовой функции **Weights_internet_connections** за базу были взяты веса из **Weights_normal** и подправлены под соответствующую исходную задачу. А именно, факторы совпадающих страны, города, школы, университета и факультета становятся для нас менее значимыми и их значения превращаются в 1, 1, 4, 1 и 4 соответственно (было 1, 1.5, 5, 2 и 5). Базой для весов различных интернет взаимодействий был взят вес поста на стене пользователя, равный 3, оставшийся неизменным по сравнению с Weigths_normal. Один общий друг теперь добавляет 0.8 (вместо 0.33), лайк под постом и лайк под фотографиями получили веса 3 и 4 соответственно (вместо 2.5 и 2.5).

3.3 Построение итогового графа

После подсчета весов создается неориентированный взвешенный граф, где ребро существует между каждыми двумя пользователями, а его вес вычисляется с помощью одной из рассмотренных ранее функций и характеризует связь данных пользователей в соответствии с исходной задачей. Вершины нумеруются с нуля, то есть последняя вершина имеет номер на один меньше, чем число вершин.

Граф изначально хранится в виде списка смежности, то есть для каждой вершины поддерживается массив пар: набор вершин, смежных с данной вершиной, и весов инцидентных ребер. Далее для работы алгоритма по выделению сообществ более удобна иная структура данных для хранения графа. Нам потребуется часто вычислять степени вершин, находить всех соседей и веса между вершиной и ее соседями, поддерживать хранение и изменение весов вершин. Для того чтобы это делать максимально эффективно по времени и памяти, создадим 4 массива:

1. **Массив префиксных сумм степеней вершин** используется, чтобы вычислять степень любой заданной вершины и префиксную сумму степеней вершин, предшествующих данной по нумерации, что необходимо нам для использования последующих двух массивов.
2. **Массив ребер** – массив, имеющий длину, равную числу ребер (в данном случае считаем, что одно неориентированное ребро дает 2 ориентированных) в графе, где последовательно лежат номера соседей для каждой вершины. То есть, если мы узнаем степень вершины и префиксную сумму степеней вершин до данной, то мы сможем быстро найти номер элемента в массиве, хранящего номер первого соседа нашей вершины, и сколько их всего. Следовательно, мы получаем указатель на начало массива и количество элементов в нем, что необходимо для реализации прохода по массиву, заданному указателем.
3. **Массив весов** – массив, в котором последовательно лежат веса ребер, что позволяет нам, аналогично массиву ребер, находить вес ребра с первым соседом нашей вершины и, зная степень вершины, узнавать число соседей.
4. **Массив весов вершин** поддерживает веса вершин, которые появляются, когда на втором и более шагах алгоритма вершинами графа становятся сообщества, полученные на предыдущем шаге алгоритма. Вес такой вершины – сумма весов ребер внутри сообщества.

Такая структура данных позволяет реализовать методы для нахождения степени вершины, взвешенной степени вершины, узнавать есть ли у заданной вершины петля и, если она есть, то ее вес, суммарный вес графа и добавлять веса сразу всем вершинам при запуске нового шага алгоритма.

4. Анализ взвешенного графа

Одним из общепризнанных методов выделения неявных сообществ является нахождение такого разбиения графа на сообщества, при котором достигается максимальное значение модулярности, характеризующей разницу в плотности полученных кластеров в сравнении с условно ожидаемой [9-11].Более точно, модулярность Q определяется следующим образом:

$$Q=\frac{1}{2m}\sum_{i,j}\left[A_{ij}-\frac{k_i k_j}{2m}\right]\delta\left(c_i,c_j\right), \qquad (1)$$

где m – количество ребер,

A_{ij} – элемент матрицы смежности графа

k_i – степень вершины i,

c_i – номер класса, к которому принадлежит вершина,

$$\delta(c_i, c_j) = \begin{cases} 0, c_i = c_j; \\ 1, c_i \neq c_j. \end{cases}$$

Для подсчета модулярности взвешенного графа определим понятие взвешенной степени вершины как сумму весов всех ребер, инцидентных вершине. Тогда модулярность для взвешенного графа считается по формуле:

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j), \quad (2)$$

где m – сумма весов всех ребер в графе,

A_{ij} – вес ребра между вершинами i и j

k_i – взвешенная степень вершины i,

c_i – номер класса, к которому принадлежит вершина,

$$\delta(c_i, c_j) = \begin{cases} 0, c_i = c_j; \\ 1, c_i \neq c_j. \end{cases}$$

Оптимизация функционала модулярности происходит в известном иерархическом алгоритме Louvain[12], реализация его версии для взвешенных графов была использована авторами для выделения сообществ. Далее покажем, как именно он применяется в нашем случае.

4.1 Выделение сообществ

Мы реализовали структуру оценки качества, которая поддерживает массив разбиения вершин на сообщества, где согласно алгоритму Louvain изначально каждая вершина принадлежит своему собственному сообществу. Данная структура предоставляет методы для исключения и вставки вершин в сообщества, вычисления выигрыша от перемещения любой заданной вершины в соседнее с ней сообщество и вычисления общей модулярности графа при текущем разбиении. Для использования и реализации данных методов используется три массива, где первый массив поддерживает для каждой вершины номер сообщества, к которому она принадлежит на данном этапе. Другие массивы отвечают за то, чтобы для каждого сообщества хранить сумму взвешенных степеней вершин, находящихся в этом сообществе, и сумму весов между вершинами данного сообщества. Поддержка этих массивов позволяет быстро вычислять выигрыш от перехода из одного сообщества в другое и пересчитывать общую модулярность графа. Рассмотрим работу данной структуры подробнее.

Назовем эти массивы $tot[]$ и $in[]$ соответственно, тогда, если мы хотим добавить вершину $node$ в сообщество $comm$, и при этом мы рассчитали, что суммарный вес ребер между нашей вершиной $node$ и ее соседями из сообщества comm равен d, то нам нужно выполнить следующие действия:

К сумме весов между вершинами $in[]$ сообщества $comm$ мы добавляем сумму веса петли вершины $node$, если она есть, и удвоенного d, так как граф неориентированный и в сумме весов веса всех ребер, составляющих d, требуется учесть дважды: как вес ребра (i, j) и вес ребра (j, i).

$$in[comm] = in[comm] + 2 * d + \\ + вес\_петли(node)$$

При этом к сумме взвешенных степеней вершин $tot[]$ сообщества $comm$ нам требуется только прибавить взвешенную степень вершины $node$:

$$tot[comm] = tot[comm] + \\ + взвешенная\_степень(node)$$

Следовательно, для удаления вершины $node$ из сообщества $comm$, нам также потребуется вычислить d и провести аналогичные процедуры, только используя вычитание из $in[comm]$ и $tot[comm]$.

Зная, как поддерживаются и изменяются массивы $tot[]$ и $in[]$ мы можем вычислять выигрыш при перемещении вершины $node$ в сообщество $comm$, если мы также вычислили расстояние d между вершиной и сообществом по формуле

$$gain = d - tot[comm] * \\ * степень\_вершины\ (node) / \\ / суммарный\_вес\_графа$$

Тогда модулярность графа можно вычислить как[12]:

$$Q = \sum_{i=0}^{Число\ вершин} \frac{in[i] - tot[i]^2}{m}\theta(tot[i]),$$

где m – сумма весов всех ребер в графе,

A_{ij} – вес ребра между вершинами i и j

$$\theta(x) = \begin{cases} 1, x > 0 \\ 0, x \leq 0 \end{cases} \quad (3)$$

Реализация одного шага алгоритма заключается в следующем. Расставив вершины в случайном порядке, осуществляется проход по каждой из них. После чего исключается вершина из ее сообщества, выбирается наилучшее по приросту модулярности сообщество из числа инцидентных нашей вершине. Далее вершина добавляется в найденное сообщество. Такой проход по

всем вершинам продолжается пока после окончания очередного шага выполняются 3 условия:

1. Хотя бы одна вершина была перемещена из своего сообщества в другое.
2. Общая модулярность графа возросла не менее, чем на ε, где ε – гиперпараметр, задаваемый пользователем.
3. Число проходов не превысило заданное, где заданное максимальное число проходов – гиперпараметр.

Если хотя бы одно из условий нарушается, то это считается критерием остановки –текущий шаг алгоритма закончен. После этого граф изменяется так, что вершинами в нем становятся текущие сообщества, появлялся внутренний вес новых вершин и следующий шаг алгоритма запускается уже на новом графе.

4.2 Примеры работы на графах ВКонтакте

В качестве примера был импортирован из сети ВКонтакте граф реального пользователя, чтобы иметь возможность оценить качественно результат выделения сообществ. Ожидаемыми сообществами являлись: университет, школа, двор и футбольная лига университета. Сообщества были выделены описанным выше алгоритмом для всех трёх весовых функций, после чего результаты разбиения сравнили с разбиением, полученным алгоритмом Blondel [11] для невзвешенных графов.

Для того чтобы показать различия разбиения при изменении выбора весов, взяты одинаковые для всех из них гиперпараметры (регуляризация = 25, ε = 1e-2, максимальное число итераций не ограничено). Получающиеся графы имеют 245 вершин и 2052 ребер.

На всех примерах присутствует часть вершин-«фейков», не имеющих никаких ребер из-за регуляризации, т.е. удаления в силу слишком маленьких весов.

4.2.1 Работа Weights_normal и Weights_friends

Для начала рассмотрим сообщества численностью больше 10 человек и сравним, насколько они соответствуют действительности. Получаем следующий граф и результат:

Таблица 2

Соответствие найденных кластеров действительным (weights_normal)

Название сообщества	Соответствие действительности	Численность
Cluster_1	Университет	48
Cluster_2	Школа	37
Cluster_3	Двор	22
Cluster_4	Футбольная лига университета	15
Cluster_5	Подмножество университета	10
Cluster_6	Подмножество двора	10

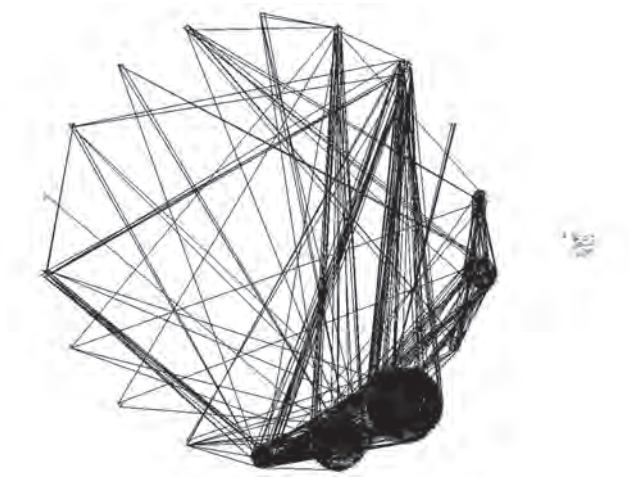

Рисунок 1. Разбиение, полученное с весовой функцией weights_normal

Функция **Weights_friends** на данном графе дает практически такой же результат, лишь немного меняя положение людей между сообществами-подмножествами одного сообщества.

4.2.2 Работа Weights_internet_connections

При весах **Weights_internet_connection** больший вес имеют не общие параметры пользователей: школа, университет, и т.п., а общение пользователей внутри сети, что позволило выделить локальные сообщества внутри больших. На примере видно, что сообщество университета разделилось на 3 более маленьких с большой плотностью связей между сообществами.

Таблица 3

Соответствие найденных кластеров действительным (weights_internet_connection)

Название сообщества	Соответствие действительности	Численность
Cluster_1	Школа	36
Cluster_2	Университет	31
Cluster_3	Двор	24
Cluster_4	Футбольная лига университета	17
Cluster_5	Подмножество тесно общающихся людей из университета	13
Cluster_6	Другое подмножество тесно общающихся людей из университета	13

Кроме указанных в таблице сообществ, содержащих более 10 пользователей, на рисунке справа, рядом с сообществом двора, можно увидеть 2 маленьких сообщества, которые в свою очередь являются подмножествами этого большого сообщества.

УДК 004.421.2:519.178 DOI: 10.21681/2311-3456-2019-4-61-71

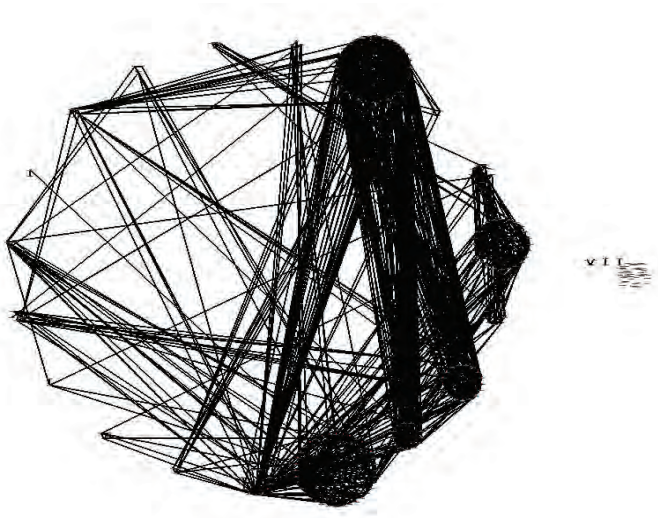

Рисунок 2. Разбиение, полученное с весовой функцией weights_internet_connection

4.2.3 Классическое разбиение

Алгоритм Блонделя хуже справляется с данным графом, так как он не использует взвешенности графа и ориентируется только на наличие ребер меду пользователями. Таким образом, ему удается выделить лишь 2 больших сообщества, связи между которыми были удалены с помощью регуляризации. Однако внутри сообществ их по-прежнему огромное число, поэтому Блондель не делит их на меньшие.

Также приведем таблицу сравнения работы алгоритма Louvain на 2-х весовых функциях (weights_normal и weights_internet_connection) и аналогов известных алгоритмов Blondel и Infomap[13-15] (табл 4).

Здесь для изучения степени совпадений разбиений, полученных применением Louvain с двумя весовыми функциями и классических алгоритмов, подсчитана нормализованная взаимная информация Normalized Mutual Information [16].

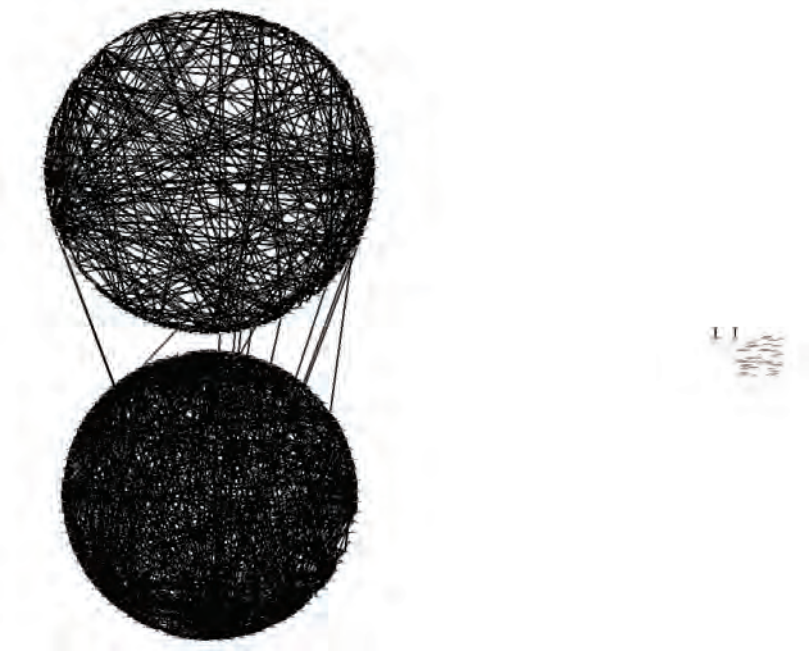

Рисунок 3. Разбиение, полученное алгоритмом Блонделя

Таблица 4

Сравнение работы алгоритмов выделения сообществ

Алгоритм	Число кластеров	Кластеры численностью:				Метрика NMI (за правильное взято разбиение алгоритма Infomap)	Метрика NMI (за правильное взято разбиение алгоритма Louvain с весами weights_ normal)
		10+	5-10	2-5	1(Фейки)		
Louvain (weights_normal)	77	6	2	12	57	0.811	1
Louvain (weights_internet_conection)	72	6	4	12	50	0.816	0.880
Blondel	55	2	0	5	50	0.868	0.750
Infomap	60	4	1	5	50	1	0.811

4.2.4. Зависимость разбиения от параметра регуляризации

Ниже представлен граф (рис.4), получающийся, если выбрать регуляризацию достаточно большой (равной 100 в данном случае) – граф разделяется на несколько компонент связности, остается только малая часть ребер (74). Обратный случай показан на рис. 5, тут есть регуляризация выбрана слишком маленькой (равной 5). В таком случае в графе остается много связей с «фейками», то есть шум, который мешает выделению сообществ. Количество ребер достигает 21284.

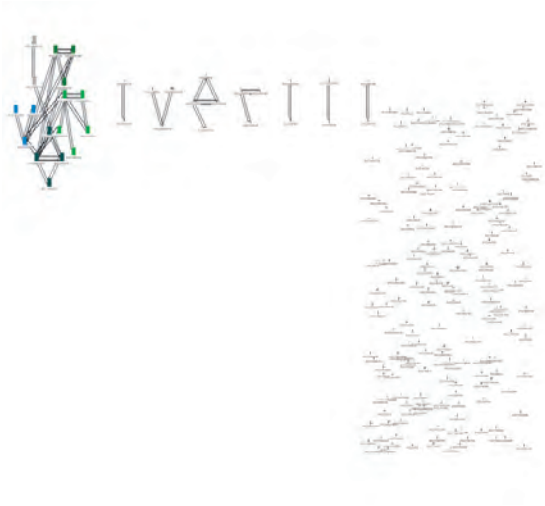

Рисунок 4. Разбиение, полученное при регуляризации = 100

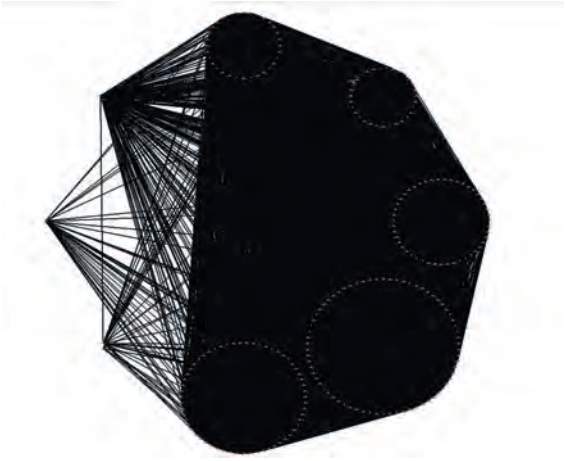

Рисунок 5. Разбиение, полученное при регуляризации = 5

4.2.5 Зависимость разбиения от параметра ε

Рассмотрим теперь зависимость разбиения на сообщества в зависимости от выбранной константы ε. Данный параметр останавливает перемещение вершин между сообществами, если после очередного перемещения выигрыш оказался меньше ε. Пример иллюстрирует поведение алгоритма при достаточно больших и маленьких значениях параметра: при большом значении (ε=10) переход вершин останавливается практически в самом начале алгоритма и успевает соединить в сообщества только малую часть вершин. При маленьком значении алгоритм наоборот работает (практически) до самого конца, то есть перестает соединять сообщества между собой, только когда выигрыш от объединения становится отрицательным, поэтому мы получаем кластеры школы, университета, и т д без сообществ, являющихся подмножествами других сообществ.

УДК 004.421.2:519.178 DOI: 10.21681/2311-3456-2019-4-61-71

Рисунок 6. Значение ε = 10

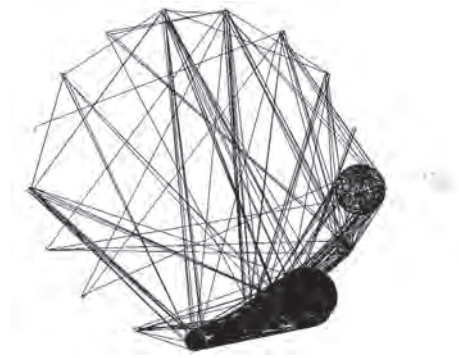

Рисунок 7. Значение ε = 1e-3

4.3 Пример работы на графе из Twitter

Так же был рассмотрен подграф сообщений из социальной сети Twitter, состоящий из 7000 вершин и 16146 ребер, построенных на основе постов и комментариев пользователей целевых сообществ с 2017 года. При этом 60% ребер в графе имели вес 1, поэтому регуляризация тут взята равной 0, а параметр $\boldsymbol{\varepsilon}$ – равен 0,02.

Было проведено сравнение работы алгоритмов Louvain, Infomap, Blondel по метрике NMI на данном графе, которое показало, что алгоритмы выделяют схожие сообщества, но разбиение Louvain немного отличается от двух других, так как он работает со взвешенным графом. Важно отметить, что в итоге алгоритм на взвешенном графе выделяет сообщества с большим числом пользователей, чем алгоритмы, работающие на невзвешенном графе (таб. 5), что отличается от рассмотренного ранее случая для ВКонтакте и не дает возможности вычленить более узкие сообщества.

Мы считаем, что основной причиной данного отличия наряду иным характером графа является подбор конкретных весов, выставленных за действия пользователей в сети Twitter. Таким образом, можно констатировать, что данный граф из иной социальной сети является примером того, насколько важно подбирать базовые веса в соответствии с характером графа взаимодействующих объектов и исходной задачей.

5. Выводы

Для целей анализа сетей взаимодействующих объектов актуальным является формирование методики, сочетающей в себе процедуру импорта данных, учет атрибутов объектов, предобработку данных, построение взвешенного графа и применение алгоритмов выделения неявных сообществ.

В рамках данной работы проанализирована популярная социальная сеть ВКонтакте, построены специальные алгоритмы прохода по ее базе данных, реализованы программные процедуры для предобработки и хранению полученных данных.

Спроектирован и реализован процесс построения и анализа взвешенных графов, соответствующих процессам взаимодействия пользователей социальных сетей. Исследован и имплементирован алгоритм Louvain для взвешенных графов с учетом используемых структур данных. Введены регуляризации и дополнительные гиперпараметры, повышающие качество разбиения графа на неявные сообщества пользователей. Программно-аппаратный комплекс применен на реальных данных из социальной сети ВКонтакте и Twitter, полученные разбиения графов приведены в работе.

Произведено сравнение результатов работы с классическими алгоритмами выделения сообществ, по итогам которого можно сделать вывод о целесообразности сформированного подхода и высоком качестве получаемых результатов. Направлением дальнейших исследований видится уточнение подходов по формированию базовых весов и весовых функций для разных сетей взаимодействующих объектов.

Таблица 5

Сравнение работы алгоритмов выделения сообществ

Алгоритм	Число кластеров	Кластеры численностью				NMI with Louvain	NMI with Infomap
		200+	100-200	10-100	Меньше 10		
Louvain	2293	4	4	20	2265	1	0.871
Infomap	2368	4	2	46	2731	0.871	1
Blondel	2023	4	4	31	2652	0.836	0.896

Литература

1. Newman M., Girvan M. Finding and evaluating community structure in networks // Physical Review E. 2004. Vol. 69. No 2. P. 1-15.
2. Girvan M., Newman M. Community structure in social and biological networks // Proceedings of the National Academy of Sciences. 2002. Vol. 99. No 12. P. 7821-7826.
3. Wu F., Chen L., Wang J., Alhajj R. Biomolecular Networks and Human Diseases // BioMed Research International. 2014. Vol. 2014. P. 1-2.
4. Ahuja M.S., Singh J., Neha Practical Applications of Community Detection // International Journal of Advanced Research in Computer Science and Software Engineering. 2016. Vol. 6. No 4. Pp. 412-415.
5. М.И. Коломейченко, И.В. Поляков, А.А. Чеповский, А.М. Чеповский Выделение сообществ в графе взаимодействующих объектов Фундаментальная и прикладная математика, том 21. №3. стр. 131-139, (2016).
6. M. Roth, A. Ben-David, D. Deutscher. Suggesting Friends Using the Implicit Social Graph — KDD'10, July 25–28, 2010, Washington, DC, USA., 2010.
7. Ahuja M.S., Singh J., Neha Practical Applications of Community Detection // International Journal of Advanced Research in Computer Science and Software Engineering. 2016. Vol. 6. No 4. Pp. 412-415.
8. Т. В. Соколова, А.А. Чеповский. Анализ профилей сообществ социальных сетей. Системы высокой доступности. Т. 14, № 3. стр. 82-86. (2018).
9. Newman M. Modularity and community structure in networks // Proceedings of the National Academy of Sciences. 2006. Vol. 103. No 23. P. 8577-8582.
10. Blondel V.D., Guillaume J.L., Lambiotte R., Lefebvre E. Fast unfolding of communities in large networks // Journal of Statistical Mechanics: Theory and Experiment. 2008. No 10. P10008. 12 p
11. А.О. Орлов, А.А. Чеповский О свойствах модулярности и актуальных корректировках алгоритма Блонделя. Вестник Новосибирского государственного университета. Серия: Информационные технологии.. Т. 15. вып. 3. стр. 64-73. (2017).
12. Que X., Checconi F., Petrini F., Gunnels J. Scalable Community Detection with the Louvain Algorithm // 29th IEEE International Parallel & Distributed Processing Symposium, May 25-29, 2015.
13. Rosvall M., Bergstrom C. T. Maps of random walks on complex networks reveal community structure // Proc. Natl. Acad. Sci. USA. 2008. Vol.105. №4. P. 1118–1123.
14. Rosvall M., Bergstrom C. T., Axelsson D. The map equation // The European Physical Journal Special Topics. 2009. Vol. 178. №1. P. 13–23.
15. Kolomeychenko M.I., Chepovskiy A.A., Chepovskiy A.M. An Algorithm for Detecting Communities in Social Networks Journal of Mathematical Sciences vol. 211 No. 3. pp. 310-318 (2015).
16. Danon L., Duch J., Diaz-Guilera A., Arenas A. Comparing community structure identification // Journal of Statistical Mechanics: Theory and Experiment, No. 9, 01.09.2005, p. 219-228.

ALGORITHMS TO REVEAL COMMUNICATION GROUPS

Leschyov D.A.[6], Suchkov D.V.[7], Khaykova S.P.[8], Chepovskiy A.A.[9]

The purpose of the study: development of methods for analyzing the graph of interacting objects based on the detection of implicit communities in order to solve the problems of searching for the proximity of profiles and the exchange, distribution of information between objects.

Method: importing data from social networks with the subsequent construction of a weighted graph based on the selected attributes and the weight function corresponding to the original task; detection of communities on the constructed weighted graph and comparison of the obtained partitions with the results of classical algorithms.

Results: algorithms to construct graphs and to import attributes were developed and implemented, weight functions created, data structures were constructed, Louvain algorithm for weighted graphs was investigated and implemented with the according to data structures, additional hyper parameters that improve the quality of the standard graph partition by implicit user communities were added. On the example of the social network VKontakte, special algorithms

6. Dmitry Leschyov, Bachelor of Education Program "Applied Mathematics", Moscow Institute of Electronics and Mathematics (MIEM HSE), Moscow, Russia. E mail: daleschyov@edu.hse.ru

7. Daniel Suchkov, Bachelor of Education Program "Applied Mathematics", Moscow Institute of Electronics and Mathematics (MIEM HSE), Moscow, Russia. E mail: dvsuchkov@edu.hse.ru

8. Svetlana Khaykova, Bachelor of Education Program "Applied Mathematics", Moscow Institute of Electronics and Mathematics (MIEM HSE), Moscow, Russia. E mail: spkhaykova@yandex.ru

9. Alexander Chepovskiy, Associate Professor, Department of Applied Mathematics, Moscow Institute of Electronics and Mathematics (MIEM HSE), Street, Moscow, Russia. E mail: aachepovsky@hse.ru

for database crawling are built, the software and hardware complex is applied on real data, and the results of work are compared with the classical algorithms for allocating communities.

Keywords: *graph structure, social network analysis, community detection.*

References

1. Newman M., Girvan M. Finding and evaluating community structure in networks // Physical Review E. 2004. Vol. 69. No 2. P. 1-15.
2. Girvan M., Newman M. Community structure in social and biological networks // Proceedings of the National Academy of Sciences. 2002. Vol. 99. No 12. P. 7821-7826.
3. Wu F., Chen L., Wang J., Alhajj R. Biomolecular Networks and Human Diseases // BioMed Research International. 2014. Vol. 2014. P. 1-2.
4. Ahuja M.S., Singh J., Neha Practical Applications of Community Detection // International Journal of Advanced Research in Computer Science and Software Engineering. 2016. Vol. 6. No 4. Pp. 412-415.
5. Kolomeychenko M.I., Polyakov I.V., Chepovskiy A.A., Chepovskiy A.M. Vydelenie soobshchestv v grafe vzaimodejstvuyushchih ob»ektov Fundamental›naya i prikladnaya matematika, Vol. 21. No3. pp. 131-139, (2016). (in Russian)
6. M. Roth, A. Ben-David, D. Deutscher. Suggesting Friends Using the Implicit Social Graph — KDD'10, July 25–28, 2010, Washington, DC, USA., 2010.
7. Ahuja M.S., Singh J., Neha Practical Applications of Community Detection // International Journal of Advanced Research in Computer Science and Software Engineering. 2016. Vol. 6. No 4. Pp. 412-415.
8. Sokolova T.V., Chepovskiy A.A. Analiz profilej soobshchestv social'nyh setej. Sistemy vysokoj dostupnosti. Vol. 14, no 3. pp. 82-86. (2018). (in Russian)
9. Newman M. Modularity and community structure in networks // Proceedings of the National Academy of Sciences. 2006. Vol. 103. No 23. P. 8577-8582.
10. Blondel V.D., Guillaume J.L., Lambiotte R., Lefebvre E. Fast unfolding of communities in large networks // Journal of Statistical Mechanics: Theory and Experiment. 2008. No 10. P10008. 12 p
11. Orlov A.O., Chepovskiy A.A. O svojstvah modulyarnosti i aktual›nyh korrektirovkah algoritma Blondelya. Vestnik Novosibirskogo gosudarstvennogo universiteta. Seriya: Informacionnye tekhnologii. Vol. 15. no. 3. pp. 64-73. (2017). (in Russian)
12. Que X., Checconi F., Petrini F., Gunnels J. Scalable Community Detection with the Louvain Algorithm // 29th IEEE International Parallel & Distributed Processing Symposium, May 25-29, 2015.
13. Rosvall M., Bergstrom C. T. Maps of random walks on complex networks reveal community structure // Proc. Natl. Acad. Sci. USA. 2008. Vol.105. №4. P. 1118–1123.
14. Rosvall M., Bergstrom C. T., Axelsson D. The map equation // The European Physical Journal Special Topics. 2009. Vol. 178. №1. P. 13–23.
15. Kolomeychenko M.I., Chepovskiy A.A., Chepovskiy A.M. An Algorithm for Detecting Communities in Social Networks Journal of Mathematical Sciences vol. 211 No. 3. pp. 310-318 (2015).
16. Danon L., Duch J., Diaz-Guilera A., Arenas A. Comparing community structure identification // Journal of Statistical Mechanics: Theory and Experiment, No. 9, 01.09.2005, p. 219-228.