АННОТАЦИЯ

В данном документе приведена пояснительная записка к программе "Программа для построения графов данных и поиска сообществ в них", предназначенной для построения графов данных и поиска сообществ в них.

В разделе "Введение" этого документа указаны наименование программы, краткое наименование программы и документы, на основании которых ведется разработка программы.

В разделе "Назначение и область применения" указаны функциональное назначение программы, эксплуатационное назначение программы и краткая характеристика области применения программы.

В разделе "Технические характеристики" содержится следующая информация: постановка задачи на разработку программы, описание применяемых математических методов и алгоритмов, алгоритмы функционирования программы и ее интерфейс, организация входных и выходных данных, описание и обоснование выбора состава технических и программных средств.

В разделе "Ожидаемые технико-экономические показатели" указаны предполагаемая потребность и полезность разработки.

Настоящий документ разработан в соответствии с требованиями следующих стандартов:

1) ГОСТ 19.101-77 Виды программ и программных документов [1];
2) ГОСТ 19.102-77 Стадии разработки [2];
3) ГОСТ 19.103-77 Обозначения программ и программных документов [3];
4) ГОСТ 19.104-78 Основные надписи [4];
5) ГОСТ 19.105-78 Общие требования к программным документам [5];
6) ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом [6];
7) ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению [7].

Изменения к данному Техническому заданию оформляются согласно ГОСТ 19.603-78 [8], ГОСТ 19.604-78 [9].

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

1.1. Наименование Программы

Название программы: "Программа для построения графов данных и поиска сообществ в них". Изложено определение графов данных и сообществ, применяемых алгоритмов, описан и обоснован выбор технических средств, описан интерфейс программы и организация входных и выходных данных.

1.2. Документ, на основе которого ведется разработка

Приказ № 2.3-02/1212-02 от 12.12.2018 «Об утверждении тем и руководителей курсовых работ студентов 3 курса Факультета компьютерных наук, образовательная программа «Прикладная математика и информатика».

Организация, утвердившая этот документ: Национальный исследовательский университет «Высшая школа экономики», Факультет компьютерных наук, образовательная программа «Прикладная математика и информатика».

Наименование темы разработки: "Сложные сети: граф связей и граф данных».

2. НАЗНАЧЕНИЕ И ОБЛАСТЬ ПРИМЕНЕНИЯ

2.1. Назначение Программы

Назначение разработки – построение графов данных и поиск сообществ в них.

2.2. Краткая характеристика области применения

Сообществами называются такие подмножества вершин, у которых много рёбер внутри подмножества и мало – с остальными вершинами[10, 11]. Существуют более строгие определения сообщества, но они не общеприняты. Поиск сообществ в сети важен, поскольку с его помощью можно изучить структуру сети, выявив в ней основные части и взаимодействия между ними.

Помимо рёбер на основе явно заданных связей между вершинами в сложной сети можно строить рёбра на основе метаданных, ассоциированных с вершинами, соединяя вершины со схожими метаданными, при этом в зависимости от конкретного типа графа данных схожесть вершин определяется по разному. Такой граф называется графом данных (proximity graph). Существует множество различных методов построения графа данных[12, 15, 16, 17] (граф относительного соседства, граф Габриеля и т.д.), некоторые из которых планируется реализовать.

Таким образом, данная программа может применяться при исследовании и работе с различными наборами данных для выявления их структуры -- выделения основных частей данных и связей между ними. Предполагается, что написанная библиотека будет использоваться для анализа данных и научных исследований. В рамках проекта написанная программа использовалась другими участниками проекта для сравнения различных характеристик графа связей и графа данных на примере некоторых датасетов. Например, Станислав Рыбин, студент 4 курса ПМИ, исследовал распределение степеней вершин, проверяя, сохраняется ли закон степенного распределения для графов данных сложных сетей.

3. ТЕХНИЧЕСКИЕ ХАРАКТЕРИСТИКИ

3.1. Постановка задачи на разработку программы

Целью проекта является разработка и создание библиотеки на языке C++ для построения графов данных и поиска сообществ в них, а так же интерфейса («обертки») на языке Python3 для этой библиотеки.

Реализованы функции построения следующих графов данных:

1. Граф k ближайших соседей.

Каждая вершина соединяется с её K ближайшими соседями.

2. Граф ε-шаров.

Соединяются пары вершин на расстоянии меньше заданного ε. При этом должна поддерживаться возможность не только явно задавать этот ε, но и автоматически находить его, передавая в функцию желаемое количество ребёр в построенном графе.

3. Граф относительного соседства (RNG, relative neighbourhood graph).

Две точки в пространстве, соответствующие объектам из набора данных, соединяются рёбром, если в пересечении двух гиперсфер с центрами в каждой из точек и радиусом, равным расстоянию между ними, нет других точек из набора данных.

4. Граф Габриеля.

Две точки в пространстве соединяются рёбром, если внутри гиперсферы, построенной на отрезке между ними как на диаметре нет других точек из набора данных.

5. Граф сфер влияния.

Для каждой точки строится гиперсфера с радиусом, равным расстоянию до её ближайшего соседа. Далее пара точек соединяется ребром, если соответствующие им гиперсферы пересекаются.

Реализованы следующие алгоритмы поиска сообществ:

1. Label propagation, синхронная и асинхронная модификации.[13]

2. Алгоритм CNM.[14]

3.2. Применяемые математические методы и алгоритмы программы

Будем рассматривать задачу разбиения вершин графа на непересекающиеся подмножества – сообщества, отметив что помимо неё существует также задача разбиения на пересекающиеся сообщества (soft clustering). Также будем подразумевать, что графы неориентированы.

Важнейший класс таких методов решения этой задачи основан на введении метрики качества

разбиения вершин на сообщества и оптимизации введённого функционала. В качестве такого функционала часто берут модулярность, определяемую следующим образом:

$$Q = \frac{1}{m} \sum_{i,j \in V} (A_{ij} - P_{ij})[c_i = c_j]$$

Здесь m – количество ребёр, V – множество вершин, A – матрица смежности графа, c_i – сообщество, к которому отнесена вершина i. P_{ij} – некоторая оценка A_{ij} для случайного графа, например (k_i k_j) / 2m , где k_i – степень вершины i. Второй сомножитель в формуле – индикатор некоторого выражения, а именно того, равны ли номера кластеров вершин i и j (другими словами, индикатор того, что вершины i и j принадлежат к одному кластеру).

3.2.1. Clauset-Newman-Moore Algorithm *(CNM)*

Задача оптимизации модулярности NP-полна, поэтому используемые на практике алгоритмы не ищут точное решение, а лишь стараются найти как можно лучшее приближение. Один из таких алгоритмов, CNM[5], создаёт по одному сообществу на каждую вершину, а затем на каждом шаге объединяет два таких сообщества, что при этом увеличение модулярности будет на данном шаге максимальным. Такие итерации продолжаются до тех пор, пока есть пара сообществ, объединение которых увеличит функцию модулярности. Таким образом, он жадно максимизирует функционал модулярности.

Процесс объединения сообществ можно изобразить лесом (графом, состоящим из нескольких деревьев), где вершины соответствуют сообществам, и потомки вершины соответствуют сообществам, объединением которых получилось соответствующее вершине сообщество. Такой граф называется дендрограммой. При использовании правильных структур данных данный алгоритм работает с временной сложностью O(n d log n), где n – размер графа, а d – высота дендрограммы. Поскольку для сложных сетей, которыми являются большинство встречающихся в реальной жизни графов (нейронные сети, графы цитирования научных публикаций, трофические сети, дорожные сети и т.д.), высота дендрограммы имеет порядок O(log n), сложность в таком случае получается равной $O(n \log^2 n)$.

Чтобы добиться такой временной сложности, необходимо поддерживать разреженную матрицу ΔQ_{ij} – увеличение функционала модулярности для каждой пары сообществ (i, j), между которыми есть хотя бы одно ребро, массив a_i – количество концов ребёр в каждом из

сообществ, а так же отображение (map) $H_{i,}$ для каждого столбца матрицы ΔQ_{ij} хранящее её максимальный элемент. При этом такое отображение должно быть упорядоченным, чтобы можно было не только быстро находить и обновлять элемент по ключу, но и быстро. Например, для такой цели подходит сбалансированное дерево поиска. Также необходимо хранить граф, в котором вершины – сообщества, рёбра проведены между теми парами сообществ, между которыми есть рёбра в исходном графе, при этом для каждое ребро графа сообществ обладает весом, равным количеству ребёр между данной парой сообществ, либо (во взвешенном случае) сумме их весов.

Описанные структуры позволяют на каждом шаге быстро найти пару сообществ, объединение которых даёт максимальный прирост модулярности и объединить их в одно. Более подробное описание работы алгоритма и доказательство его временной сложности можно найти в упомянутой выше статье.

3.2.2. Label propagation

Другой подход к поиску сообществ основан на запуске случайного процесса, в результате работы получается разбиение множества вершин на сообщества. label propagation[6] – один из таких алгоритмов. Вначале каждой вершине присваивается своё сообщество, а затем на каждой итерации для каждой вершины считается число её соседей из каждого сообщества, и выбирается сообщество с максимальным числом соседей (если таких сообществ несколько, то случайно выбранное из них). В зависимости от модификации либо сообщество вершины обновляется сразу после подсчёта (асинхронный вариант), либо сначала для всех вершин выбираются новые сообщества, а потом они одновременно обновляются после итерации по всем вершинам (синхронный вариант), либо сообщество вершины обновляется сразу, но вершины перебираются в таком порядке, что вершины из одного сообщества идут подряд (полу-синхронный вариант). Итерации продолжаются до тех пор, пока количество поменявших сообщество вершин на каждом шаге не станет меньше некоторого числа, выбираемого в зависимости от данных, либо пока не будет привышено ограничение на максимальное число итераций.

Каждая итерация имеет временную сложность O(n + m), где n и m – количество вершин и ребёр в графе соответственно, при этом число итераций, необходимых, чтобы процесс сошёлся, невелико даже для больших n. Таким образом, данный алгоритм является масштабируемым.

3.3. Алгоритмы функционирования программы и ее интерфейс

В интерфейсе для языка C++ используются следующие определённые типы данных:

Граф хранится как вектор из векторов рёбер для каждой вершины, при этом ребро представляет собой пару из индекса второго конца ребра и веса ребра:

```
struct Edge {
    int ind;
    float w;
};
using Graph = vector<vector<Edge>>;
```

Структура, которую возвращает алгоритм label propagation, представляет собой пару векторов. Первый – метки (номера сообществ) для каждой из вершин, а второй – количество изменённых меток на каждой итерации (этот вектор отражает сходимость алгоритма):

```
using LabelPropRes = std::pair<vector<int>, vector<int>>;
```

```
class FastMatrix;
```

Класс FastMatrix – класс для хранения двумерного массива вещественных чисел. В отличие от vector<vector<float>>, он хранит все данные в одном одномерном массиве, но при этом поддерживает удобную индексацию двумя индексами. Таким образом, в данной структуре данных не хранятся для каждой строки матрицы её размер, вместимость и указатель на первый элемент, что существенно экономит расход памяти. Кроме того, поскольку данные идут подряд, они лучше кешируются процессором.

Для поддержки интерфейса для языка Python3 написана программа на языке C++, считывающая запрос к библиотеке из входного файла, вызывающая функцию из библиотеки, выбранную в запросе и записывающая результат её выполнения в выходной файл. Модуль на языке Python3 содержит те же функции, что и библиотека на языке C++, но выполняет их, записывая аргументы в входной файл, вызывая описанную выше программу и считывая результат из выходного файла. Входные и выходные файлы содержат данные в текстовом формате. Текстовый формат был выбран постольку, поскольку он является более удобным для программирования и отладки по сравнению с бинарным форматом данных. С другой стороны, он менее эффективен, но это имеет малое значение, так как время передачи данных относительно мало по сравнению с временем работы алгоритмов.

Таблица с интерфейсом реализованных функций приведена в приложении. Также интерфейс функций можно найти в документации, прилагающейся к исходному коду программы.

Для построения графов данных были выбраны следующие алгоритмы:

1) Для построения ε-шаров был перебираются всевозможные пары вершин и для каждой считается расстояние, сравнивается с ε и если оно меньше, то пара вершин соединяется ребром.

Также можно было использовать структуры данных, быстро отвечающие на запрос «найти точки из заданного множества на расстоянии не более R», например, R-дерево [18], однако такие структуры эффективно работают только на данных малой размерности.

Для того, чтобы не настраивать вручную параметр ε, можно передать примерное нужное количество ребёр в графе и ε будет настроен автоматически. Для этого строится выборка из равновероятных пар вершин. На основе этой выборки находится значение ε, для которого доля пар в вершин в выборке на меньшем расстоянии примерна равно доле выбранных пар вершин в графе, при которой получается требуемое количество рёбер.

2) Для построения графа K ближайших соседей для каждой вершины перебираются все остальные вершины и поддерживается приоритетная очередь с K ближайшими соседями вершины среди рассмотренных.

Также можно было использовать структуры данных, быстро отвечающие на запрос «найти K ближайших точек из заданного множества для данной точки», например, KD-дерево[19], однако такие структуры эффективно работают только на данных малой размерности. Так, KD-дерево для d-мерных данных имеет сложность запроса $O(n^{1-1/d})$ и при этом большую константу при данной асимптотической сложности.

3) Для построения графа Габриеля и графа относительного соседства были выбраны тривиальные алгоритмы, работающие за $O(n^3)$. Существуют более оптимальные алгоритмы, но они работают лишь в случае 2 или 3 измерений[20, 21]. Так же существуют более оптимальные алгоритмы для метрики L_∞[20]. Есть алгоритмы, которые ассимптотически быстрее по времени, но имеют больший расход памяти, то есть $O(n^2)$, что более критично, чем процессорное время[19].

4) Для построения графа сфер влияния был сначала для каждой вершины находится расстояние до её ближайшего соседа, а после для каждой пары вершин расстояние между ними сравнивается с суммой расстояний до их ближайших соседей, и если сумма больше, то вершины соединяются ребром.

3.4. Выбор технических и программных средств

Программа была разработана и проверена на функционирование под управлением операционной системы Linux, дистрибутива Ubuntu 18.04, архитектуры x86_64, однако является кросплатформенной на стадии исходного кода и может использоваться на

компьютере с любой операционной системой и архитектурой.

Исходными языками данной разработки являются языки C++ и Python3.

Для набора исходного кода использовался текстовый редактор vim.

В качестве системы контроля версий использовался git. Удалённый репозиторий хранился на сайте github.com.

Для использования библиотеки при программировании на языке C++ требуется компилятор языка C++, поддерживающий стандарт C++14. При использовании интерфейса библиотеки на языке Python3 требуется интерпретатор языка Python3.

Выбор вышеописанных алгоритмов поиска сообществ обоснован тем, что они масштабируемы, то есть ассимптотически временная сложность в зависимости от размера графа относительно невелика, тем, они поддерживают графы, в которых ребра имеют различные веса, а также тем, что результат их работы не сильно зависит от настраиваемых параметров, следовательно они просты в использовании.

3.5. Тестирование и экспериментальные результаты

Тестирование алгоритмов проводилось на трёх типах данных.

1) Графы или наборы точек (для поиска сообществ или построения графа данных соответственно) из нескольких вершин, результат работы на которых сравнивался с ожидаемым ответом. Тем самым проверялась корректность работы алгоритмов.

2) Синтетические данные.

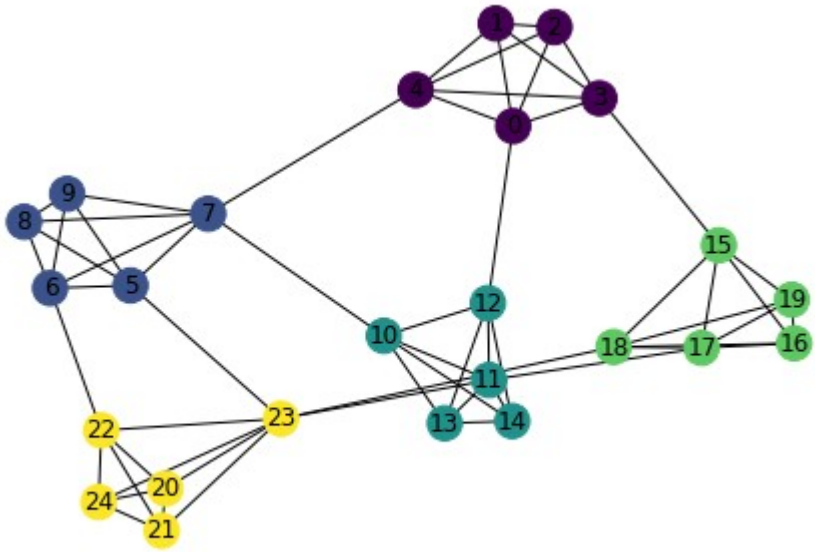
Для поиска сообществ использовалась модель SBM (stochastic block model[22]), когда вероятность наличия ребра между каждой из пар вершин задаётся в матрице, все выборы рёбер независимы, а после берётся граф из такого распределения. Для моделирования графа со структурой сообществ достаточно рассмотреть частный случай, когда задаются размеры сообществ, вероятность ребра внутри сообщества и вероятность ребра между сообществами.

Для построения графов данных использовался следующий вид данных. Сначала выбирались несколько центров кластеров. Потом для каждой точки выбирался случайный кластер и точка генерировалась из многомерного нормального распределения с центром в центре этого кластера.

Результат работы проверялся с помощью визуализации библиотеки networkx.

Примеры визуализации:

1. Поиск сообществ в графе, сгенерированном с помощью SBM. 5 сообществ по 5 вершин, вероятность ребра внутри сообщества 97%, между сообществами – 3%. Вершины, отнесённые к одному сообществу, показаны одним цветом.



2. Построение графа данных по данным из нескольких кластеров. Размерность данных 4, два кластера с центрами (0, 0, 0, -1) и (0, 0, 0, 1), матрицы ковариации многомерного нормального распределения единичные. Вершины, соответствующие точкам, сгенерированным из одного кластера, показаны одним цветом. Изображены граф K ближайших соседей, граф относительного соседства и граф Габриеля.

Рис. 1. Граф K ближайших соседей, визуализация.

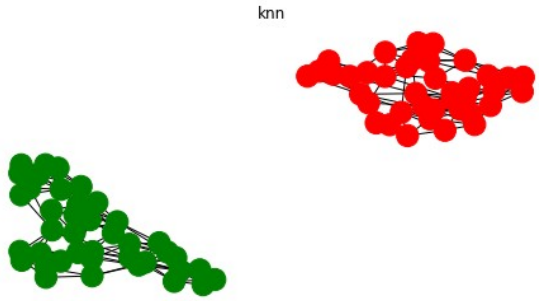


Рис. 2. Граф относительного соседства, визуализация.

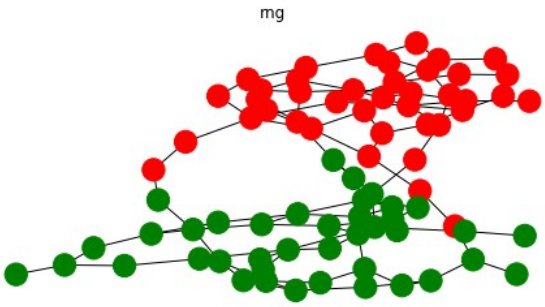
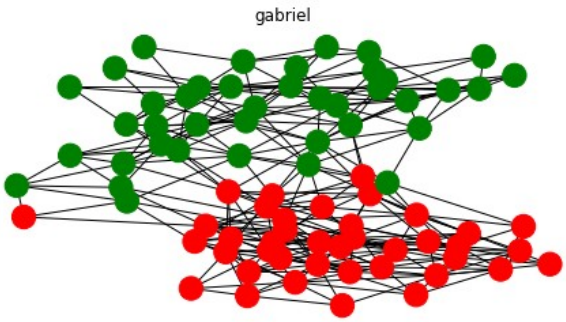


Рис. 3. Граф Габриеля, визуализация.



3) Реальные наборы данных. Были выбраны три набора данных, у которых есть как явно заданные связи между вершинами, так и метаданные для каждой вершины.

1. Граф, в котором вершины – товары на amazon.com, а рёбра проведены между товарами, часто покупаемыми вместе.

2. Набор данных «6 degrees of Francis Bacon» – социальный граф, в котором вершины – некоторые люди, жившие во Франции в 17 и 18 веках, а ребром соединены пары людей, про которых известно, что они были знакомы.

3. Граф пользователей Твиттера, в котором две вершины соединены неориентированным ребром, если один из пользователей подписан на другого (или оба пользователя подписаны друг на друга).

Таблица 1. Размеры данных.

Данные	**Количество вершин**	**Количество рёбер**
6 degrees of Francis Beacon	15801	171408
Twitter	112416	308927
Amazon.com	548552	987942

Поскольку эти наборы данных имеют большой размер, на них проводилось тестирование времени работы алгоритмов.

Таблица 2. Время построения графов данных, в секундах.

--	**6 degrees of Francis Bacon**	**Twitter**	**Amazon**

Граф ε-шаров (|E| ≈ 8 |V|)	57	291	12884
Граф k ближ. сосед. (k=8)	297	6533	95690
Граф сфер влияния	46	697	15012
Граф отн. соседства	924	--	--
Граф Габриеля	2324	--	--

Таблица 3. Время работы label propagation (100 итераций), в секундах

--	**6 degrees of Francis Bacon**	**Twitter**	**Amazon**
Исходный граф	14	52	229
Граф ε-шаров (|E| ≈ 8 |V|)	9	69	448
Граф k ближ. сосед. (k=8)	7	81	451
Граф сфер влияния	3	42	313
Граф отн. соседства	5	--	--
Граф Габриеля	11	--	--

4. ОЖИДАЕМЫЕ ТЕХНИКО-ЭКОНОМИЧЕСКИЕ ПОКАЗАТЕЛИ

Данная программа может применяться при исследовании различных наборов данных для выявления их структуры -- выделения основных частей данных и связей между ними.
В рамках данной работы расчёт экономической эффективности не предусмотрен. Продукт будет распространяться бесплатно.

5. ИСТОЧНИКИ, ИСПОЛЬЗОВАННЫЕ ПРИ РАЗРАБОТКЕ

1. ГОСТ 19.101-77 Виды программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

2. ГОСТ 19.102-77 Стадии разработки. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

3. ГОСТ 19.103-77 Обозначения программ и программных документов. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

4. ГОСТ 19.104-78 Основные надписи. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

5. ГОСТ 19.105-78 Общие требования к программным документам. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

6. ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом.// Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

7. ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

8. ГОСТ 19.603-78 Общие правила внесения изменений. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

9. ГОСТ 19.604-78 Правила внесения изменений в программные документы, выполненные печатным способом. // Единая система программной документации. – М.: ИПК Издательство стандартов, 2001.

10. Fortunato, S. Community detection in networks: A user guide / Santo Fortunato, Darko Hric // Physics Reports 659, 1-44, 2016.

11. Fortunato, S. Community detection in graphs / Santo Fortunato // Physics Reports 486, 75-174, 2010.

12. Beck, M. Computational Discrete Geometry / Matthias Beck.

13. Raghavan, U. Near linear time algorithm to detect community structures in large-scale networks / Usha Nandini Raghavan, Reka Albert, Soundar Kumara // Physical Review E 76, 2007 .

14. Clauset, A. Finding community structure in very large networks / Aaron Clauset, M. E. J. Newman,and Cristopher Moore // Phys. Rev. E 70, 2004.

15. Mitchell, J. 32 proximity algorithms / Joseph S. B. Mitchell, Wolfgang Mulzer // 2016.

16. Zemel, R. Proximity Graphs for Clustering and Manifold Learning / Richard S. Zemel //

Advances in Neural Information Processing Systems 17, 2005.

17. Proximity graphs: E,δ, Δ,χ and ω / Bose, P., Dujmovi´c, V., Hurtado, F., Iacono, J., Langerman, S., Meijer, H., Sacrist´an,V., Saumell, M., Wood, D.R. // International Journal of Computational Geometry & ApplicationsVol. 22, No. 05, pp. 439-469, 2012.

18. Guttman, A. R-Trees: A Dynamic Index Structure for Spatial Searching / Antonin Guttman // SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984.

19. Katajainen, J. An almost naive algorithm for finding relative neighbourhood graphs in L_p metrics / Jyrki Katajainen, Olli Nevalainen // Informatique théorique et applications, tome21, no2,p. 199-215, 1987

20. O'Rourke, J. Computing the relative neighborhood graph in L_1 and L_{inf} metrics / Joseph O'Rourke // Pattern Recognition, Volume 15, Issue 3, p.189-192, 1982.

21. Liotta, G. Low degree algorithms for computing and checking Gabriel graphs / Guiseppe Liotta // Pattern Recognition, 1996.

22. Mossel, E. Stochastic Block Models and Reconstruction / Elchanan Mossel, Joe Neeman, Allan Sly // arXiv preprint, 2012.

ПРИЛОЖЕНИЕ 1

Таблицы с описанием функций

Таблица 1. Интерфейс функций на языке C++

имя	тип	аргументы	назначение
eps_graph	vector<vector<int>>	const FastMatrix &data, int n_edges	Построить граф пар вершин на расстоянии не более эпсилон.
eps_graph_n_edges	vector<vector<int>>	const FastMatrix &data, int n_edges	Построить граф пар вершин на расстоянии не более эпсилон, однако эпсилон подбирается автоматически так, чтобы в полученном графе было примерно n_edges рёбер
knn_graph	vector<vector<int>>	const FastMatrix &data, size_t k	Построить граф k ближайших соседей
rng_graph	vector<vector<int>>	const FastMatrix &data	Построить граф относительного соседства
gabriel_graph	vector<vector<int>>	const FastMatrix &data	Построить граф Габриеля
influence_graph	vector<vector<int>>	const FastMatrix &data	Построить граф сфер влияния
label_propagation	LabelPropRes	const Graph &graph, int max_iter, int min_delta, int async=false	Поиск сообществ методом label propagation. Передаются условия остановки (максимальное количество итераций и минимальное количество замен меток на итерации), а так же аргумент, отвечающий за то, синхронная или асинхронная модификация алгоритма будет использоваться. Возвращается пара из двух векторов. В первом – номер сообщества для каждой из вершин, а во второй количество изменений номеров на каждой итерации.
CNM	vector<int>	const Graph &graph	Поиск сообществ алгоритмом CNM. Возвращается вектор из

			номера сообщества для каждой из вершин.

Таблица 2 Интерфейс функций на языке Python3.

имя	аргументы	назначение
eps_graph	data, epsilon	Построить граф
eps_graph_n_edges	data, n_edges	Построить граф пар вершин на расстоянии не более эпсилон, однако эпсилон подбирается автоматически так, чтобы в полученном графе было примерно n_edges рёбер
knn_graph	data, k	Построить граф k ближайших соседей
rng_graph	data	Построить граф относительного соседства
gabriel_graph	data	Построить граф Габриеля
influence_graph	data	Построить граф сфер влияния
label_propagation	graph, max_iter, min_delta, f_async=false	Поиск сообществ методом label propagation. Передаются условия остановки (максимальное количество итераций и минимальное количество замен меток на итерации), а так же аргумент, отвечающий за то, синхронная или асинхронная модификация алгоритма будет использоваться. Возвращается пара из двух векторов. В первом – номер сообщества для каждой из вершин, а во второй количество изменений номеров на каждой итерации.
CNM	graph	Поиск сообществ алгоритмом CNM. Возвращается вектор из номеров сообщества для каждой из вершин.

ЛИСТ РЕГИСТРАЦИИ ИЗМЕНЕНИЙ

Изм.	Номера листов (страниц)				Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа и дата	Подпись	Дата
	измененных	замененных	новых	аннулированных					

Изм.	Лист	№ докум.	Подп.	Дата
RU.17701729.XX.XX-03 81				
Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата