Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт прикладной математики и механики

Работа допущена к защите
Директор ВШПМиВФ
_____ Л.В. Уткин
«_____» _____ 2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
РАБОТА БАКАЛАВРА
РАЗРАБОТКА МЕТОДА АВТОМАТИЧЕСКОГО РАЗБИЕНИЯ ИЗОБРАЖЕНИЙ МОЗГА ПЛОДОВОЙ МУШКИ НА ОБЛАСТИ

по направлению подготовки 01.03.02 «Прикладная математика и информатика»
Направленность (профиль) 01.03.02_04 «Биоинформатика»

Выполнил
студент гр. 3630102/70401 А.С. Вяткин

Руководитель
доцент,
к. б. н., ВШПМиВФ, ИПММ К.Н. Козлов

Консультант
по нормоконтролю Л.А. Арефьева

Санкт-Петербург
2021 г.

САНКТ-ПЕТЕРБУРГСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО
Институт прикладной математики и механики

УТВЕРЖДАЮ
Руководитель образовательной программы
«Прикладная математика и информатика»
_____ К.Н. Козлов
«20» декабря 2020 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы

студенту Вяткину Александру Сергеевичу, гр. 3630102/70401

1. Тема работы: Разработка метода автоматического разбиения изображений мозга плодовой мушки на области.
2. Срок сдачи студентом законченной работы: июнь 2021.
3. Исходные данные по работе:
   Данные о двух видах мозгов плодовой мушки.
   Инструментальные средства:
   - Язык программирования: C++, Python3
   - Библиотеки: Boost, GTest
   - Среды разработки: CLion, Jupyter notebook

   Ключевые источники литературы:
   - Vincent D. Blondel, Jean-Loup Guillaume, Renaud Lambiotte, Etienne Lefebvre (2008). Fast unfolding of communities in large networks. J. Stat. Mech. P10008
   - V.A. Traag, L. Waltman, and N.J. van Eck Centre (2019). From Louvain to Leiden: guaranteeing well-connected communities.
   - Ulrik Brandes (2008). On Variants of Shortest-Path Betweenness Centrality and their Generic Computation.
4. Содержание работы (перечень подлежащих разработке вопросов):
   В работе предполагается решить следующие задачи:
   1. Введение. Обоснование актуальности.

2. Постановка задачи. Описание данных.
3. Описание проведенных экспериментов.
4. Описание алгоритмов, реализованных в работе.
5. Описание различных модификаций.
6. Результаты и их сравнительный анализ.
7. Выводы.
8. Заключение.

5. Дата выдачи задания: 20.12.2020

Руководитель ВКР _____ К.Н. Козлов

Задание принял к исполнению _____

Студент _____ А.С. Вяткин

РЕФЕРАТ

На 46 с., 23 рисунка, 8 таблиц,
КЛЮЧЕВЫЕ СЛОВА: ДРОЗОФИЛА, МОЗГ, КЛАСТЕРИЗАЦИЯ, ГРАФЫ, МОДУЛЬНОСТЬ, МАШИННОЕ ОБУЧЕНИЕ.

В данной работе был разработан метод автоматического разбиения изображения мозга плодовой мушки на области. Данный алгоритм ускоряет и упрощает анализ функциональных областей мозга плодовой мушки.

Был проведен сравнительный анализ классических алгоритмов кластеризации, таких как: к-средних, *BIRCH* и иерархического алгоритма кластеризации, который работает снизу вверх, объединяя кластеры с минимальным средним квадратическим отклонением, с реализованным в данной работе алгоритмом. Реализованный в данной работе алгоритм является генетическим. Были придуманы и реализованы несколько видов мутаций:

- выделяющая мутация
- разделяющая мутация, имеющая две модификации: случайную, которая случайно делит пополам кластер и детерминированную, удаляющую ребра, через которые проходит наибольшее количество кратчайших путей в данном графе, пока граф не перестанет быть связным
- склеивающая мутация

Была разработана метрика, по которой сравнивались качества разбиений. Также была разработана параллельная версия генетического алгоритма, заметно ускорившая его работу.

Была установлена несостоятельность классификаторов из пакета *scikit-learn*. Были проведены численные эксперименты по разбиению мозгов плодовых мушек для шести экспериментальных образцов. Разработанный генетический алгоритм показал свое превосходство над классическими алгоритмами кластеризации.

ABSTRACT

46 pages, 23 figures, 8 tables,
KEYWORDS: DROSOPHILA, BRAIN, CLUSTERIZATION, GRAPHS, MODULARITY, MACHINE LEARNING.

In this work, a method was developed for automatically splitting an image of a fruit fly brain in an area. This algorithm speeds up and simplifies the analysis of functional areas of the fruit fly brain.

A comparative analysis of classical clustering algorithms, such as k-means, *BIRCH* and a hierarchical clustering algorithm, which works from the bottom up, combining clusters with the minimum standard deviation, with the algorithm implemented in this work, was carried out. The algorithm implemented in this work is genetic. Several types of mutations were invented and implemented:

- extracting mutation
- separating mutation, having two modifications: random, which randomly bisects the cluster and deterministic, which removes the edges through which the greatest number of shortest paths in a given graph passes until the graph is no longer connected
- merging mutation

A metric was developed by which the quality of the partitions was compared. A parallel version of the genetic algorithm was also developed, which significantly accelerated its work.

An inconsistency of the classifiers from the *scikit-learn* package has been fixed. Numerical experiments were performed to break the brains of fruit flies for six experimental samples. The developed genetic algorithm has shown its superiority over the classical clustering algorithms.

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

NEC Neuron expression clusters, кластеры экспрессии нейронов.

MEL *Drosophila Melanogaster*, дрозофила меланогастер.

SIM *Drosophila Simulans*, дрозофила симуланс.

DBSCAN Density-based spatial clustering of applications with noise, пространственная кластеризация приложений с шумом на основе плотности.

OPTICS Ordering points to identify the clustering structure, упорядочивание точек для выявления кластерной структуры.

BIRCH Balanced iterative reducing and clustering using hierarchies, сбалансированное итеративное сокращение и кластеризация с использованием иерархий.

KNN K nearest neighbours, к ближайших соседей.

SVM Support vector machine, машина опорных векторов.

ВВЕДЕНИЕ

Человечество еще очень плохо понимает, как работает головной мозг человека, но это знание очень важно для лечения таких заболеваний головного мозга, как болезнь Альцгеймера[8], деменция и другие, которые сейчас являются неизлечимыми, а с возросшей средней продолжительностью жизни человека[9], данные болезни могут лишить его дееспособности достаточно рано, что ведет к большим экономическим убыткам и социальным проблемам.

На данном этапе развития технологий мы не можем моделировать и анализировать достаточно детально такие большие данные, как головной мозг человека, поэтому люди тестируют алгоритмы выделения структурных областей головного мозга на модельных организмах. В данной работе такой модельный организм - плодовая мушка[10].

На данный момент автоматизированных алгоритмов решения задачи выделения функциональных областей мозга - не существует. Сейчас эксперт кластеризует данные руками. Он относит каждую точку - набор нейронов мозга объекта - к соответствующему кластеру. Это очень трудоемкая работа даже для такого маленького организма, как плодовая мушка. Ее мозг насчитывает всего лишь несколько тысяч таких вершин. Соответственно, существует потребность в разработке программного пакета, который автоматизирует процесс кластеризации.

Разработать автоматизированный способ выделения функциональных областей мозга плодовой мушки по полученным данным.

Для решения поставленной задачи требуется выполнить следующие подзадачи:

- Предобработать данные, а именно построить граф на точках из трехмерного пространства. Для достижения этой цели разработать программу.
- Модифицировать существующие алгоритмы кластеризации для получения удовлетворительных результатов. Основные требования к алгоритмам - один кластер полученного разбиения не должен содержать более одного кластера реального разбиения, потому что разделить кластер сложнее, чем объединить.
- Придумать метрику качества полученного разбиения.
- Проанализировать полученные результаты.

ГЛАВА 1. ПОСТАНОВКА ЗАДАЧИ И ОПИСАНИЕ ДАННЫХ

1.1. Биологическое описание

Дрозофил выращивали на стандартной среде при 25°C в режиме нахождения 12 часов при свете, и 12 часов в темноте.

Культуральные пробирки использовались однократно, чтобы избежать псевдоповторностей, так, что каждый генотип и пол были отобраны из отдельных популяций дрозофил. Плотность каждой культуры дрозофил контролировали, и одному и тому же числу родителей давали 24 часа для откладки яиц. Самцы мух были собраны через 11 дней, незадолго до эклозии, чтобы облегчить препарирование, поскольку мозг взрослой мухи, как правило, плавает в жидкости.

При препарировании вначале аккуратно отделялась ткань головы и ротовой аппарат, чтобы избежать повреждения мозга. Для переноса мозга в пробирку для последующего окрашивания использовали пипетку. Для каждого вида (*Drosophila melanogaster* и *Drosophila simulans*) извлекали мозг в трех повторностях, всего было получено 6 препаратов мозга.

Различия в формировании паттерна нейронов в NEC (кластерах экспрессии нейронов) между NEC у самцов оценивались для поздних куколок (стадия P15). Дрозофил на поздней стадии куколки легко идентифицировать и препарировать, что позволяет проводить синхронизированное по развитию сравнение между генотипами на более крупных выборках. Помимо этого важно, что NEC на стадии P15 мало отличаются от таковых в центральной нервной системе взрослых самцов[11]. Таким образом, получение изображений на этой стадии позволяет выявить различия в структуре зрелых NEC между видами дрозофил.

Был оптимизирован разработанный ранее протокол HCR глазного диска для мозга дрозофилы. Этот высокопроизводительной метод флуоресцентного окрашивания мРНК позволяет оценить различия в картинах экспрессии генов между видами дрозофил с разрешением в одну клетку.

Каждый образец был отсканирован в постериорно-антериорном направлении с использованием конфокальной системы Zeiss LSM 780 с объективом PlanApochromat 20×/1.40 и водной иммерсией. Изображения получали в программном обеспечении ZEN в режиме lambda, который позволяет улучшать детекцию сигнала путем измерения полного спектра эмиссии от каждой длины волны возбуждения. Все образцы сканировали с практически неизменными параметрами:

сила лазера и прирост. При этом они настраивались таким образом, чтобы избежать насыщения пикселей.

1.2. Формат хранения данных

В этой секции будет приведено описание собранных ранее данных. Каждый мозг представлен в виде таблицы с 9ю колонками. Формат хранения данных - *.csv*, где в качестве разделителя используется символ - запятая. Ниже приведено детальное описание каждой из колонок:

- id - целое неотрицательное число, являющееся уникальным номером вершины. Каждая вершина представляет собой несколько нейронов мозга плодовой мушки, полученных при помощи специального оборудования.
- x - вещественное число, представляющее собой одну из координат в трехмерном пространстве полученной вершины.
- y - вещественное число - ордината вершины.
- z - вещественное число - аппликата вершины.
- q - вещественное число, характеризующее интенсивность свечения флуоресценции в данной точке.
- bra - натуральное число, являющееся уникальным номером мозга.
- zone - целое число, уникальный номер функциональной области мозга. В каждом полушарии их 14, также есть кластер, имеющий номер 8, он не принадлежит ни одному из полушарий, он находится ровно посередине.
- h - категориальный признак, принимающий 3 уникальных значения: l, r и m. Смысл этих обозначений - принадлежность вершины к левому, правому или соответственно никакому - центральной области мозга - полушарию.
- spec - категориальный признак, имеющий 2 возможных значения: m и s. m - значит, что данный мозг принадлежит плодовой мушке типа MEL, a s - принадлежность к типу SIM.

Визуализация данных мозга MEL 2(Рис. 1.1), полученная при помощи пакета для языка *r - rgl*. Всего имеется 6 мозгов - 3 MEL и 3 SIM. Мозги двух типов отличаются достаточно сильно, например, это видно из сравнения изображения мозга MEL 2(Рис. 1.1) и SIM 2(Рис. 1.2). Качественно они отличаются симметрией между полушариями.

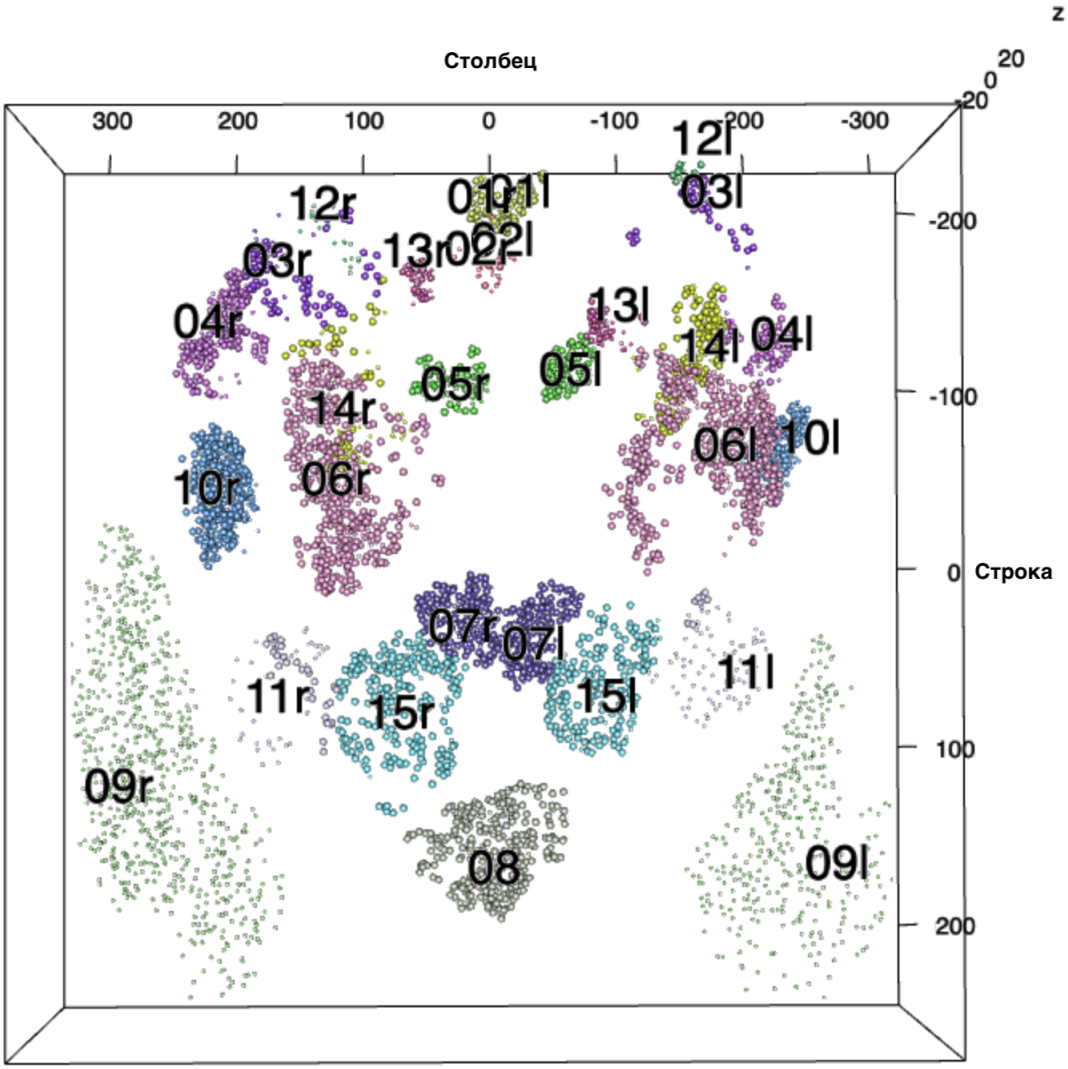


Рис.1.1. MEL 2

1.3. Постановка задачи

Цель данной работы разработка метода автоматического выделения кластерной структуры данных. На вход подается файл в формате .csv, на выходе должно получиться разбиение данных на кластеры. Каждой вершине из .csv файла, должно быть сопоставлено некоторое число, являющееся уникальным идентификатором кластера. В данной работе в качестве уникальных идентификаторов кластеров будут использоваться целые неотрицательные числа.

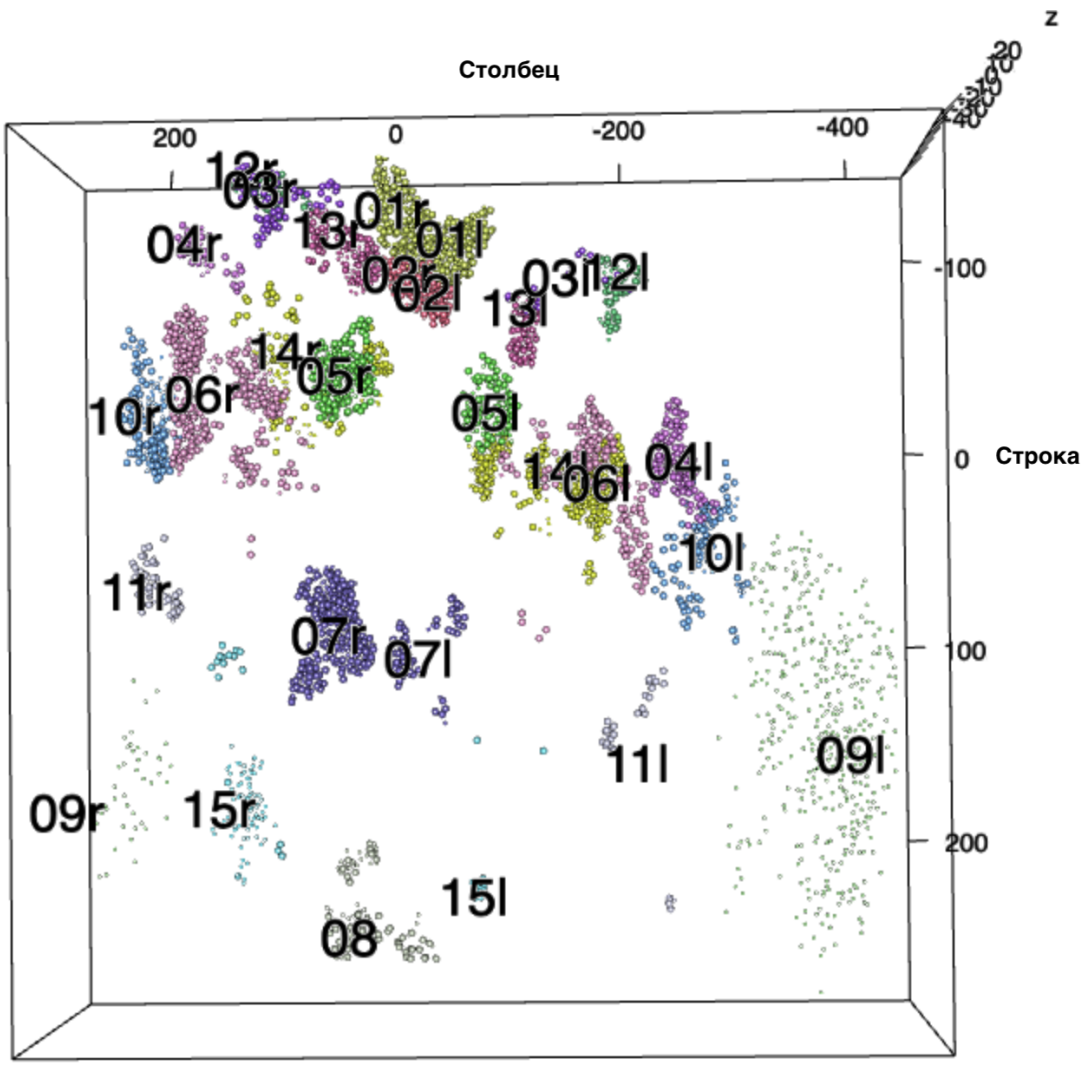


Рис.1.2. SIM 2

1.4. Существующие подходы к решению решению

Конкретно эта задача - новая, но для задач кластеризации существует множество разработанных методов. Наиболее популярные из которых можно, например, найти в пакете *scikit-learn*. Их можно разделить на несколько категорий:

- метрические: *K-means*, *DBSCAN*[12], *OPTICS*[13]
- иерархические: *BIRCH*[3]

В данной работе был выбран следующий подход к решению этой задачи:

- первый этап - построения графа на трехмерных данных.
- второй этап - выделение кластерной структуры данных, при помощи специализированных методов.

ГЛАВА 2. ОПИСАНИЕ ЭТАПОВ РАБОТЫ С ДАННЫМИ, МЕТРИК И АЛГОРИТМОВ

Петлями в данной работе называются ребра, которые соединяют вершину саму с собой.

Мультиребрами называются кратные ребра, то есть это ребра, которые имеют одни и те же конечные вершины.

Синглтон разбиением будем называть разбиения множества вершин графа, в котором каждой вершине соответствует свое сообщество(кластер).

2.1. Модульность

На вход модульности подается граф $G(V, E)$ и разбиение(2.1) множества вершин V на непересекающиеся подмножества в объединении дающее все множество вершин V(2.2). Это стандартная метрика при анализе кластерной структуры графов[16].

Качественный смысл этой метрики - это выделение кластеров в графе, у которых среднее число ребер внутри сообщества больше квадрата среднего числа ребер во всем графе.

Максимизация этой метрики в общем случае - NP-трудная задача[14]. Переборное решение, проверяющее каждое возможное разбиение графа на подмножества, работать будет очень долго, так как количество всевозможных разбиений растет как число Белла - B_n[15]. Соответственно требуется использовать эвристический алгоритм, который будет иметь хорошее соотношение значения метрики - качества разбиения - к скорости работы.

Также про данную метрику известно, что для неориентированных графов без мультиребер и петель значение модульности ограниченно следующими значениями: $-0.5 \leqslant Modularity(G(V, E), C) \leqslant 1$. Также стоит отметить, что значение 1 может быть и не достижимо на конкретном графе.

$$C = \bigcup_{i=1}^{n} C_i, \tag{2.1}$$

где $C_i \cap C_j = \varnothing$, при $i \neq j$ и $C_i \cap C_j = C_i$, при $i = j$.

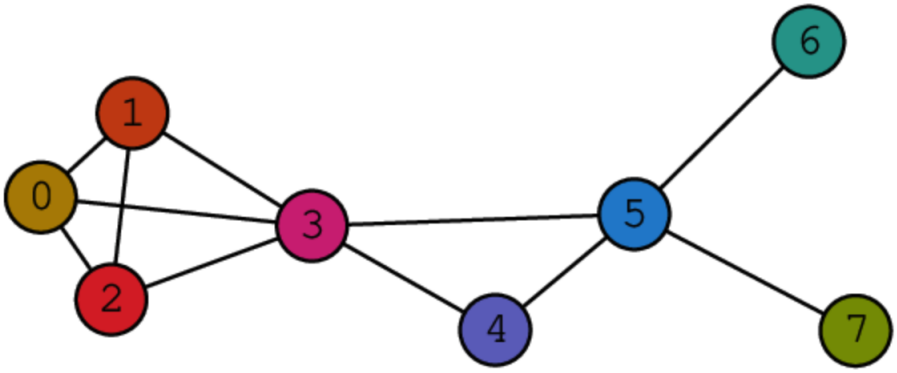


Рис.2.1. Синглтон разбиение

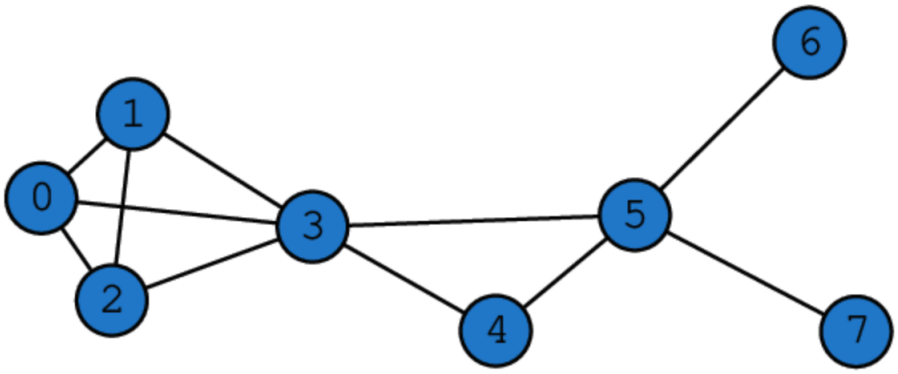


Рис.2.2. Граф - один кластер

$$Modulatity(G(V,E),C) = \sum_{c \in C} \left(\frac{|E_c^{in}|}{|E|} - \left(\frac{\sum_{v \in C} deg(v)}{2\,|E|} \right)^2 \right), \qquad (2.2)$$

где $n = |C|$ количество сообществ в графе, $|E_c^{in}| = \{(v,u) \in E | v \in c, u \in c\}$ количество ребер внутри сообщества, $|E|$ количество ребер в графе, $deg(v)$ степень вершины v.

При разбиении множества вершин на сообщества каждое из которых представляет собой одну вершину(Рис. 2.1) модульность приблизительно равна -0.153. В случае, когда все вершины относятся к одному множеству(Рис. 2.2) модульность равна 0. На последней иллюстрации(Рис. 2.3) она с точностью до трех знаков после запятой равна 0.281. Для данного графа лучшего разбиения не получить, то есть это один из тех случаев, когда модульность для данного графа не может в точности достичь 1.

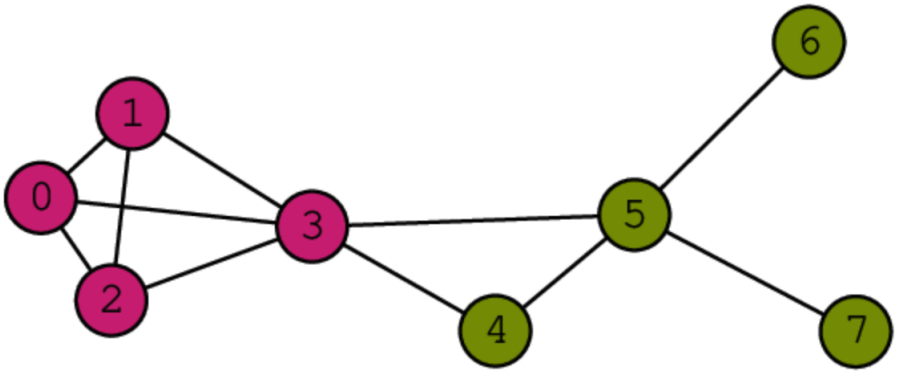


Рис.2.3. Лучшее разбиение

2.2. Внешняя метрика

Данная метрика(2.3) была придумана для того, чтобы сравнивать экспертное разбиение мозга плодовой мушки с разбиением, полученным алгоритмом. Она принимает значение от 0 до 1. Значение равное 1 она принимает, когда все кластеры полученного разбиения полностью содержатся в каком-нибудь кластере экспертного разбиения.

$$metric(P') = \frac{\sum_{i=1}^{m} \max_{j=1,n}(P'_i \cap P_j)}{\sum_{i=1}^{n} |P_i|}, \tag{2.3}$$

где P - экспертное разбиение, P' - разбиение полученное алгоритмом, $P = \bigcup_{i=1}^{n} P_i$, при $|P| = n$ $P' = \bigcup_{i=1}^{m} P'_i$, при $|P'| = m$.

2.3. Этапы работы с данными

В двух последующих секциях будут описана предобработка данных - построение графа по точкам из трехмерного пространства и алгоритмы, которые для полученного графа разбивают его множество вершин на сообщества.

2.4. Графопостроитель

Данный модуль проекта реализован на *C++*[7], так как для построения графа необходимо посчитать попарные расстояния между вершинами, и если вершин в мозге достаточно много, а их может быть несколько тысяч, то интерпретируемые языки, такие как *Python*, могут не справиться за удовлетворительное время. Данный

алгоритм строит мультиграф, количество ребер между парой вершин в котором интерпретируется, как вес ребра. Ниже описаны параметры, которые алгоритм имеет:

- mean_cluster_size - положительное вещественное число, являющееся характеристикой среднего размера кластера, а точнее его радиусом. В случае, когда расстояние между вершинами превышает радиус, то ребро не будет построено.
- number_of_weights - натуральное число, показывающее на сколько открытых полуинтервалов разобьется отрезок от 0 до mean_cluster_size.
- weights - натуральные числа, являющиеся весами ребер.
- p - вещественное положительное число или бесконечность, параметр в расстоянии Минковского.

Данный алгоритм проходится двойным вложенным циклом по всем вершинам и находит расстояние между всеми уникальными парами вершин, которых $binomial(2, |V|), |V|$, где количество вершин в графе. Дальше для каждого полученного расстояния проверяется не больше ли оно среднего размера кластера, если это так, то этого ребра не будет в графе, но если расстояние все-таки оказалось меньше среднего размера кластера, то тогда бинарным поиском[17] находится нужный вес ребра и он вместе с уникальными идентификаторами вершин записывается в список смежности. Далее этот список смежности будут обрабатывать алгоритмы кластеризации графов.

2.5. Алгоритмы

В данной секции будут приведены алгоритмы использованные для кластеризации графа, полученного на предыдущем этапе. Все описанные ниже алгоритмы реализованы на *C++* в одной библиотеке, с использованием умных указателей, мув-семантики, трансформеров и многих других возможностей современного *C++*[7], также в данном приложении использовались библиотеки: *Boost* - для парсинга командной строки и для записи логгов, *GTest* - для покрытия тестами кода. Отметим, что данное приложение поддерживает многопоточное исполнение, что было реализовано при помощи стандартных возможностей современного *C++*: *future*[7]. Выбор пал на данную стандартную библиотеку из-за простоты и удобства в использовании.

Данный модуль проекта имеет следующие параметры командной строки:

- input - строка, хранящая путь к файлу, содержащему граф записанный в виде списка смежности, где каждая строка заполняется следующим образом: два уникальных идентификатора вершин и количество ребер между этой парой вершин. Эти три числа, разделены пробелом.
- output - строка, хранящая путь к файлу, а котором храниться результат работы кластеризаторов. В первой строке храниться служебная информация - значение модульности данного разбиения. Далее построчно хранятся уникальные идентификаторы вершин, принадлежащие текущему сообществу. Они также разделены пробелами.
- multi - при указании данного параметра, граф считывается как мультиграф.
- algo - строка, которая говорит какой алгоритм вызвать. Этот параметр имеет следующие возможные значения: *louvain*, *leiden*, *k-nearest*, *genetic*.
- log - строка, хранящая путь в котором будет сохраняться лог работы алгоритмов.
- verbose - целое неотрицательное число, которое говорит как много служебной информации сохранять. Чем больше число, тем больше служебной информации будет записано. Максимальное значение - 6.
- seed - целое число, являющееся семенем для генератора случайных чисел, если его не указывать, то будет сгенерировано случайное семя, при помощи процедуры *std::random_device*.

2.5.1. Алгоритм, идея которого вдохновлена алгоритмом к-средних

Алгоритм очень сильно похож на алгоритм к-средних, одно из отличий - способ подсчета расстояния между вершинами графа.

У данного алгоритма есть три параметра:

- k - параметр, отвечающий за максимальное расстояние до вершины, которая будет считаться соседом данного центра.
- min_number_of_clusters - минимальное количество кластеров, или что тоже самое минимальное количество центров кластеризации.
- max_number_of_clusters - максимальное количество кластеров, или начал кластеризации.

Сначала в диапазоне от min_number_of_clusters до max_number_of_clusters будет случайно выбрано реальное количество центров кластеризации.

Дальше генерируется количество центров кластеризации, которые выбираются из следующим образом: сначала все вершины сортируются по убыванию степени, после этого достаточное количество первых вершин - столько, какова степень первой вершины в отсортированном списке - случайно перемешиваются, и первые вершины в количестве штук, равному количеству центров кластеризации, становятся центрами кластеризации.

После этого граф обходится обходом в ширину с началом в каждом центроиде, пока расстояние до текущей вершины в обходе не будет больше k. При этом проверяется, что данная вершина не принадлежит уже какому-то кластеру. Каждому обойденному набору вершин присваивается свое уникальное значение идентификатора кластера.

После этого все не обойденные вершины обходятся(если такие есть) обходом в ширину с центром в первой обнаруженной необойденной вершине. Всем вершинам, которые были обойдены таким образом присваивается свой уникальный идентификатор кластера. Это продолжается, пока все вершины не будут иметь свой идентификатор кластера.

2.5.2. Алгоритм Лювейна

Алгоритм немного модифицирован по сравнению с классическим алгоритмом[1], а именно добавлены следующие параметры, которые могут менять поведение алгоритма:

- is_initialized - булев параметр, если его значение *true*, то считается, что у всех вершин в переданном графе, уже есть свои кластеры. В противном случае, вызывается стандартная процедура инициализации, о которой будет написано подробнее чуть ниже(*SingletonPartition*).
- is_stochastic - этот параметр тоже принимает булевы значения, если его значение *true*, то вызывается процедура *StochasticMoveNode*. В противном случае, если его значение *false*, то вызывается процедура *DeterministicMoveNode*.

Дальше в этой секции будет описан алгоритм Лювейна с небольшим количеством изменений по сравнению с классическим алгоритмом[1]. При значении *false* параметра *is_initialized* вызывается процедура *SingletonPartition*.

При вызове *SingletonPartition* каждая вершина графа переносится в свой кластер. Дальше пока вершины двигаются при помощи процедуры *MoveNodes*,

которая для каждой вершины вызывает одну из процедур: *StochasticMoveNode* или *DeterministicMoveNode*. В *DeterministicMoveNode* текущая вершина перемещается в сообщества ее соседей и также пробуется переместить текущую вершину в синглтон комьюнити. При этом запоминается сообщество, в которое при перемещении данной вершины модульность увеличивается на наибольшее значение, если увеличивающего метрику перемещения не найдено, то вершина остается в своем сообществе. *StochasticMoveNode* отличается только тем, что тут отсутсвует жадная часть, то есть вершина перемещается в комьюнити не с наибольшим увеличением значения метрики, а в первое, при перемещении в которое данной вершины значение модульности увеличилось.

Также стоит заметить, что модульность пересчитывать каждый раз с нуля для всего графа достаточно дорого, поэтому модульность считается для каждого сообщества отдельно и потом складывается по всем сообществам, что дает общую модульность. При перемещении вершины по соседним сообществам, запоминается прирост, который дает эта вершина при попадании в текущее сообщество, что позволяет достаточно дешево(с точки зрения временных затрат, скорее имеется в виду быстро) пересчитывать модульность для каждого сообщества, а значит и графа в целом.

Следующим этапом алгоритма строится аггрегированный граф. Этим занимается процедура *AggregateGraph*. Она каждое сообщество превращает в вершину с количеством петель равному количеству внутренних ребер в сообществе, а количество внешних ребер равно количеству ребер равное исходному количеству внешних для сообщества ребер. Исходный граф превращается в неориентированные граф с петлями и мультиребрами.

В тот момент, когда не остается вершин, которые еще могут быть передвинуты, так чтобы модульность увеличивалась, цикл, описанный выше, останавливается и вызывается процедура *Flat*, которая аггрегированный граф возвращает в исходное состояние с сохранением приобретенных индикаторов кластеров у каждой из вершин.

2.5.3. Алгоритм Лейдена

Данный алгоритм[2] является улучшением алгоритма Лювейна[1]. Они делят общую идею, поэтому они имеют одинаковую асимптотику. Авторами этого алгоритма были обнаружены несколько проблем алгоритма Лювейна: сообщества

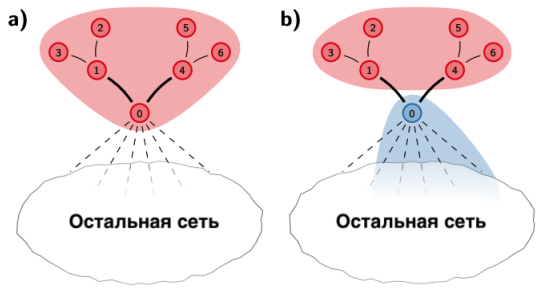


Рис.2.4. Несвязное сообщество

иногда могут быть несвязными(эта основная проблема, с которой борются авторы. С этой целью была разработана процедура *RefinePartition*)(Рис. 2.4) и при помощи улучшенной процедуры *MoveNodes*, которую они назвали *MoveNodesFast*, получаются также более связные сообщества.

После движения узлов сети могло получиться состояние изображенное на рисунке (Рис. 2.4) a), и уже через некоторое количество итераций состояние разбиения могло измениться и перейти в состояние, изображенное на рисунке(Рис. 2.4) b). В дальнейшем красное сообщество, так как оно хорошо связано(с точки зрения модульности), не будет меняться. И это приведет к появлению несвязных кластеров в разбиении.

Никаких дополнительных параметров, как у алгоритма Лювейна[1], у данного алгоритма нет, так как он не использовался, как генератор первой популяции для генетики, которая описана ниже.

Первый этап работы похож на аналогичный этап в алгоритме Лювейна. Сначала инициализируется разбиение графа при помощи процедуры *SingletonPartition*, где каждой вершини присваивается свой уникальный идентификатор сообщества.

Следующие этапы повторяются пока разбиение меняется.

Сначала вызывается процедура *MoveNodesFast*. На первой итерации двигаются все узлы, а потом только соседи тех узлов, которые были перемещены на предыдущей итерации, а также эти соседи не должны быть узлами, которые принадлежат сообществу, в которое текущий узел перемещается. Все эти узлы добавляются в очередь и на каждой итерации движения один узел забирается из очереди и он двигается при помощи процедуры *DeterministicMoveNode*, которая жадно перемещает узел в сообщество, дающее наибольший прирост модульности. Если ни одно сообщество не дает положительного прироста, то текущий узел возвращается в сообщество, из которого он был взят. Это продолжается пока очередь узлов не опустеет.

При пересчете модульности разбиения графа применяется та же идея, которая описанная в алгоритме Лювейна1. Точно также сохраняются дельты, которые позволяют быстро пересчитывать модульность каждого сообщества, что позволяет в итоге быстро пересчитать модульность разбиения в целом.

На следующем этапе вызывается процедура *RefinePartition*. Она делает новое синглтон разбиение текущего графа. Дальше для каждого сообщества из текущего разбиения применяется процедура *MergeNodesSubset*. Для каждого узла, для которого выполняется условие гамма-связности(2.4), и также если он все еще находится в синглтон комьюнити, выбирается одно из сообществ, являющееся подмножеством S с вероятностью пропорциональной его связности. Здесь имеется в виду та же гамма связность, только для сообщества.

v - гамма-связна, если:

$$v \in S, E(v, S - v) \geqslant \gamma|v| * (|S| - |v|), \tag{2.4}$$

где $E(G, P)$ - количество ребер между G и P, $|v|$ - количество вершин, которое было аггрегированно в данную, S - текущее улучшаемое сообщество.

На данном этапе выполняется процедура *AggregateGraph*, которая создает из каждого сообщества узел мультиграфа с петлями, где петли - внутренние ребра текущего сообщества. На этом заканчивается цикл.

По выходу из цикла вызывается процедура *Flat*, которая возвращает граф в исходное состояние, с сохранение приобретенных уникальных идентификаторов сообществ.

2.5.4. Генетический алгоритм

Данный алгоритм называется генетическим, потому что идея, лежащая в его основе позаимствована у природы, точнее позаимствован эволюционный механизм организмов. Данный алгоритм имеет наибольшее число параметров:

- min_size_of_population - натуральное число, являющееся нижней оценкой на размер первой популяции. Этот параметр указывается в специальном конфигурационном файле.
- max_size_of_population - натуральное число, являющееся верхней оценкой на размер первой популяции. Значение этого параметра должно быть не меньше, чем значение параметра min_size_of_population. Этот параметр указывается в специальном конфигурационном файле.

- max_number_of_iterations - натуральное число, являющееся верхней границей количества итераций генетического алгоритма. Этот параметр указывается в специальном конфигурационном файле.
- number_of_threads - натуральное число, являющееся верхней границей количества потоков. Оно будет меньше, например, если количество индивидуумов в популяции меньше запрашиваемого количества потоков. Этот параметр указывается в специальном конфигурационном файле.
- percentage_of_mutants - вещественное число от 0 до 1, говорящее какое количество мутантов, в процентном соотношении к размеру популяции, может быть максимально получено. Этот параметр указывается в специальном конфигурационном файле.
- epsilon - положительное вещественное число, являющееся точностью с которой модульности двух различных разбиения графа на сообщества считаются совпадающими. Этот параметр указывается в специальном конфигурационном файле.
- schedule - натуральное число, которое говорит сколько итераций генетического алгоритма должны иметь одинаковое значение модультности с точностью до epsilon, чтобы считать, что генетический алгоритм сошелся. Этот параметр указывается в специальном конфигурационном файле.
- refine_frequency - натуральное число, показывающее с какой частотой или раз сколько в итераций будет производиться жадное улучшение всех индивидуумов в популяции. Под жадным улучшением понимается вызов процедуры *StochasticLouvain* - версии алгоритма Лювейна[1] с стохастическим движением узлов и с параметром is_initialized установленным в *true*. Этот параметр указывается в специальном конфигурационном файле.
- mutation_frequency - натуральное число, показывающее раз в сколько итераций будут производиться мутации. Этот параметр указывается в специальном конфигурационном файле.
- save_intermediate - параметр принимающий значения 0 или 1. Если его значение 0, то промежуточные разбиения, полученные генетическим алгоритмом сохраняться не будут. Если значение 1, то будет сохраняться промежуточный результат(под промежуточным результатом понимается полученное разбиение) для лучшего по модульности индивидуума с точностью до epsilon. Этот параметр указывается в специальном конфигурационном файле.

- prefix - строка, являющаяся префиксом или путем по которому будут сохраняться промежуточные результаты, если save_intermediate установлен в 1. Имена файлов будут генерироваться следующим образом: *prefix<iteration_number>.out*. Этот параметр указывается в специальном конфигурационном файле.
- mut - строка, в которой описывается количество и порядок применения мутаций. Реализованы мутации трех типов: *merging*, *extracting* и *separating*. В качестве значения этого параметра можно указать любое количество мутаций, каждая из которых имеет свое уникальное имя. Имена мутаций разделяются пробелами. Это параметр командной строки.
- config - строка, в которой храниться путь к конфигурационному файлу. Это параметр командной строки.

Алгоритм начинает с генерации размера популяции в диапазоне от min_size_of_population до max_size_of_population включительно.

Дальше вызывается генератор первой популяции, который генерирует сколько-то первых индивидуумов - разбиений графа.

После этого запускается основной цикл генетического алгоритма, который сначала скрещивает разбиения, вызовом процедуры *Cross*. Эта процедура принимает два разбиения и для каждого сообщества из этих двух разбиений пересекает их, при этом их пересечение выделяется в отдельное сообщество. *Cross* вызывается попарно для всех уникальных пар индивидуумов. Результаты скрещивания сохраняются как потомки.

Далее применяются мутации. Генерируется реальное количество мутантов на данной итерации: от 0 до percentage_of_mutants умноженный на размер популяции, обе границы включаются. Далее мутации применяются в том порядке и количестве, в котором они указаны в параметре mut. Каждый вид мутаций и их параметры будут описаны ниже.

Предпоследним этапом является жадная оптимизация полученной новой популяции. Вызывается процедура *StochasticLouvain*.

Последним этапом является сортировка полученной популяции по убыванию модульности.

2.5.5. Выделяющая мутация

Эта мутация была придумана для выделения из сообществ со слабой плотностью(имеется в виду относительно остальных сообществ), вершин с наименьшей степенью связности. Цель - увеличение плотности сообществ.

У этой мутации есть следующие параметры, которые задаются в специальном конфигурационном файле:

- min_extracting_communities - натуральное число, являющееся нижней границей количества сообществ, из которых будут выделяться плохо связанные узлы.
- max_extracting_communities_part - вещественное число - оценка верхней границы количества сообществ, являющееся процентом от общего числа сообществ для данного разбиения, из которых будут выделены плохо связанные узлы.
- min_extracting_nodes - натуральное число - нижняя граница количества узлов, которые будут выделены из сообщества.
- max_extracting_nodes_part - вещественное число - оценка верхней границы(процент от общего числа вершин в каждом сообществе) числа узлов, которые будут выделены из сообщества.

На первом этапе генерируется число сообществ, из которых будут выделяться вершины. Диапазон из которого будет выбираться количество сообществ: min_extracting_communities до max_extracting_communities_part умноженное на количество сообществ в текущем разбиении.

На следующем этапе сообщества сортируются по возрастанию плотности.

Дальше для нескольких первых сообществ в цикле выбирается количество узлов, каждый из которых будет отнесен к своему сообществу(синглтон сообщество). Диапазон - от min_extracting_nodes до max_extracting_nodes_part умноженное на количество узлов в комьюнити. Обе границы включительно.

На следующем этапе в цикле выделяются узлы из сообщества.

2.5.6. Склеивающая мутация

Эта мутация была придумана как антипод к предыдущей мутации. Цель объединение произвольных сообществ в более крупные.

У этой мутации также есть несколько параметров, которые необходимо задать в специальном конфигурационном файле:

- min_merging_communities - натуральное число большее двух, нижняя граница граница значения переменной, которая отвечает за количество сообществ, которые могут быть объединены.
- max_merging_part - вещественное число, оценка верхней границы значения переменной, которая отвечает за количество сообществ, которые могут быть объединены. Значение этого параметра - процент от общего числа сообществ в разбиении.

На первом этапе проверяется возможно ли что-то объединить, то есть количество сообществ в текущем разбиении больше одного.

Следующий этап - случайное перемешивание всех сообществ, чтобы в дальнейшем выбрать первые сколько-то сообществ. Это количество определяется на следующем этапе.

На этом этапе случайно генерируется число от min_merging_communities до max_merging_part умножить на количество сообществ в разбиении. Обе границы включаются.

На последнем этапе объединяется столько сообществ в одно, каково было значение сгенерированного числа на предыдущем шаге.

2.5.7. Разделяющая мутация

Идея этой мутации - разделение сообщества на два или более подсообщества с достаточно большим числом вершин. Основная разница с выделяющей мутацией - выделяющая разделяет на одно большое и несколько синглтон комьюнити.

У данной мутации есть несколько параметров:

- min_separating_communities - натуральное число, нижняя граница числа сообществ, которые будут разделены на подсообщества.
- max_separating_communities_part - вещественное число, процент от общего числа сообществ в данном разбиении, которое при умножении на общее число сообществ является верхней границей числа сообществ, которые будут разделены на подсообщества.
- engine - строка, которая принимает два уникальных значения: *girvan–newman*[6] и *random-split*. Это два основных бэкэнд алгоритма, которые применяются для разделения сообщества на несколько. Первый алгоритм разделяет на связные подсообщества, если исходное сообщество

было связным. Второй алгоритм делит на два подмножества примерно одинаковой мощности, но не обязательно связанные.

Основной алгоритм можно разделить на следующие этапы:

Сначала генерируется число сообществ, которые будут разделены на подсообщества. Это число генерируется в диапазоне от min_separating_communities до max_separating_communities_part умноженное на количество сообществ в данном разбиении.

Далее сообщества в разбиении случайно перемешиваются и дальше первые несколько сообществ, в количестве, равном сгенерированному ранее числу, будут взяты для разделения.

Следующий этап - непосредственно разделение. Вызывается соответствующий алгоритм, который вернет массив массивов уникальных идентификаторов вершин. Каждый такой массив идентификаторов - это новое сообщество.

На последнем этапе обновляется соответствующим образом разбиение.

Теперь опишем два существующих бэкэнд алгоритма:

- *random-split* - сначала все вершины сообщества перемешиваются. После этого первая половина вершин остается в сообществе, а вторая половина переностится во вновь созданное сообщество. Этот алгоритм стохастический, его можно использовать для получения более разнообразной популяции.
- *girvan-newman* - можно выделить следующие этапы этого алгоритма:
  Первым шагом при помощи обхода в ширину в цикле для каждой вершины, как начала обхода, находим кратчайшее число путей от данной вершины до всех оставшихся и запоминаем это число.
  На втором шаге обновляем *edge betweenness*[6]. Начиная с вершин, которые были обойдены позже всех пересчитываем по следующему правилу вес ребра: если это листовая вершина обхода, то делим общее число путей из каждого родителя на общее число путей до данного ребенка и записываем прибавляем это к текущему значению метрики, хранящемуся в данном ребре, если вершина не листовая, то проделывается та же самая процедура, только в конце к значению метрики еще прибавляется единица.
  Последним этапом - ребра имеющие максимальное значение *edge betweenness* помечаются как удаленные.
  Это продолжается пока граф остается связными. Как только граф стано-

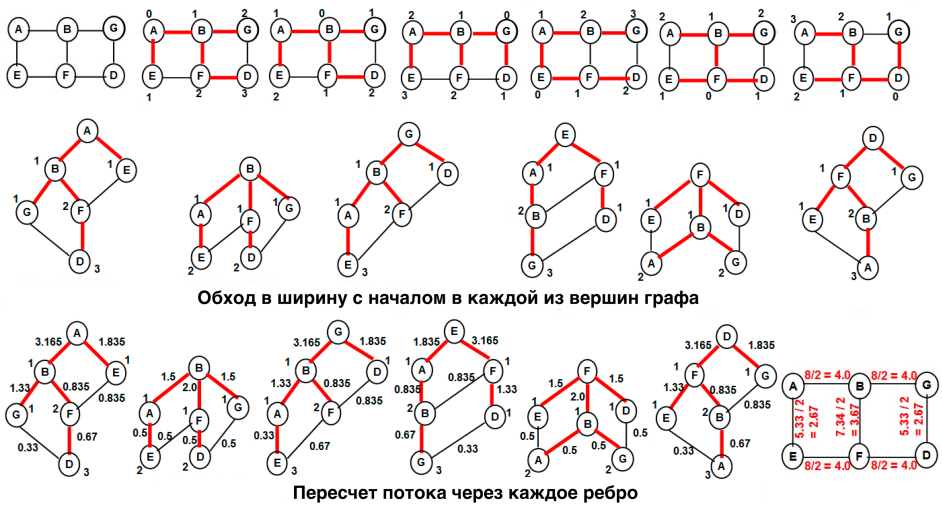


Рис.2.5. Иллюстрация работы алгоритма Джирвана-Ньюмана

вится несвязными - функция заканчивает работу и возвращает разбиение на подсообщества(массив массивов уникальных идентификаторов вершин).

На данной иллюстрации, разобран пример работы алгоритма Джирвана–Ньюмана[18](Рис. 2.5). Во второй строке строится дерево обхода для каждой из вершин, в третьей строке показан пересчет значения *edge betweenness* для каждого случая.

ГЛАВА 3. ПРОВЕДЕННЫЕ ЧИСЛЕННЫЕ ЭКСПЕРИМЕНТЫ

В данной главе приводится несколько численных экспериментов, проведенных в данной работе. Были опробованы классические алгоритмы[5] реализованные в пакете *scikit-learn*: *AgglomerativeClustering*, *BIRCH*[3] и *K-Means*. Также была попытка применить классические алгоритмы классификации, также реализованные в пакете *scikit-learn*: *SVM*[19], *K-NearestNeighbours* и *RandomForest*[20]. В последующих секциях будут описаны эксперименты с графопостроителем. В данной секции все тесты проводятся на половинах мозгов, для этого все файлы с данными были заранее разделены на половины по признаку h.

3.1. Кластеризаторы

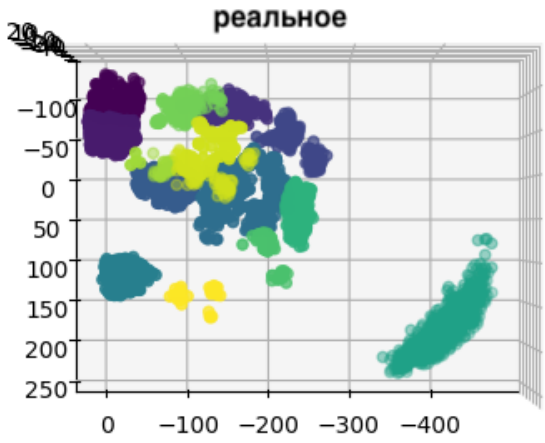
В этой секции приводятся иллюстрации и значения метрик классических алгоритмов кластеризации.

3.1.1. K-Means

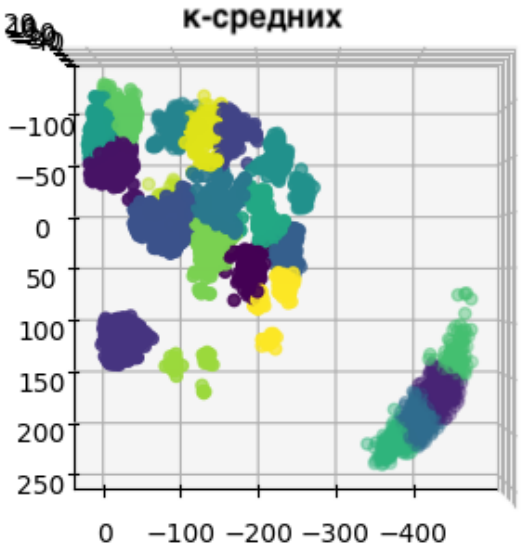
В данном алгоритме выбирается несколько центров, количество которых на каждой итерации остается неизменным. Далее положения этих центров итеративно пересчитывается. Это продолжается пока не выполнится условия критерия остановки.

Данный алгоритм был запущен со следующими параметрами:

- количество кластеров - 21. Всего кластеров в одном полушарии 14, за исключением центрального. Данное количество кластеров было выбрано с целью разбиения некоторых сообществ на более мелкие составляющие так, чтобы объединить эти кластеры было несложно руками, но при этом кластеризация получалась более точная с точки зрения внешней метрики.
- инициализационный алгоритм - *k-means++*[4]. Самый быстрый из доступных методов инициализации центроидов, дающий достаточно хорошее первое приближение. Основная его идея в максимизации расстояния между центрами.

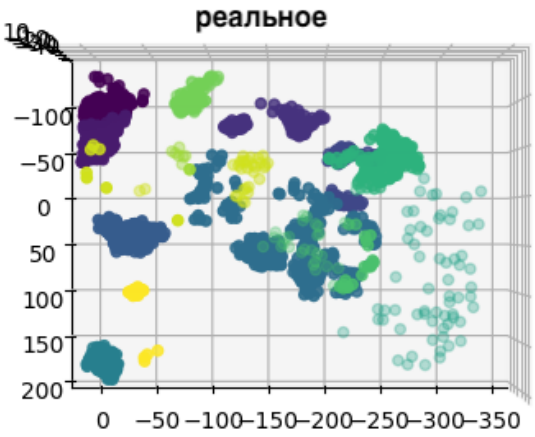


a) Реальное разбиение мозга MEL 3

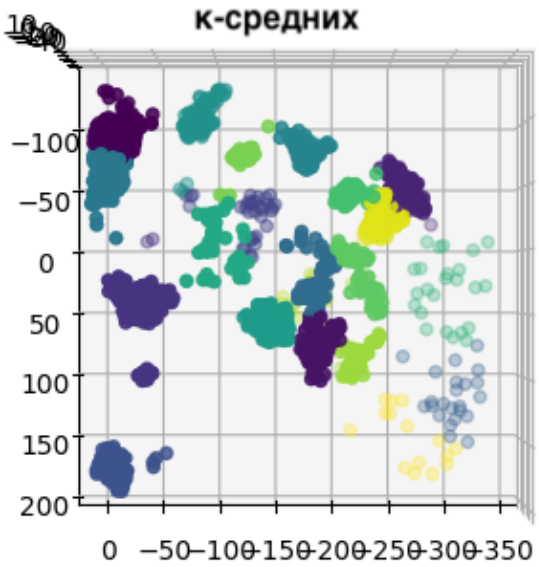


b) Разбиение мозга MEL 3, полученное алгоритмом *K-Means*

Рис.3.1. Сравнение результатов работы алгоритма *K-Means* с реальным разбиением мозга мухи MEL 3



a) Реальное разбиение мозга SIM 3



b) Разбиение мозга SIM 3, полученное алгоритмом *K-Means*

Рис.3.2. Сравнение результатов работы алгоритма *K-Means* с реальным разбиением мозга мухи SIM 3

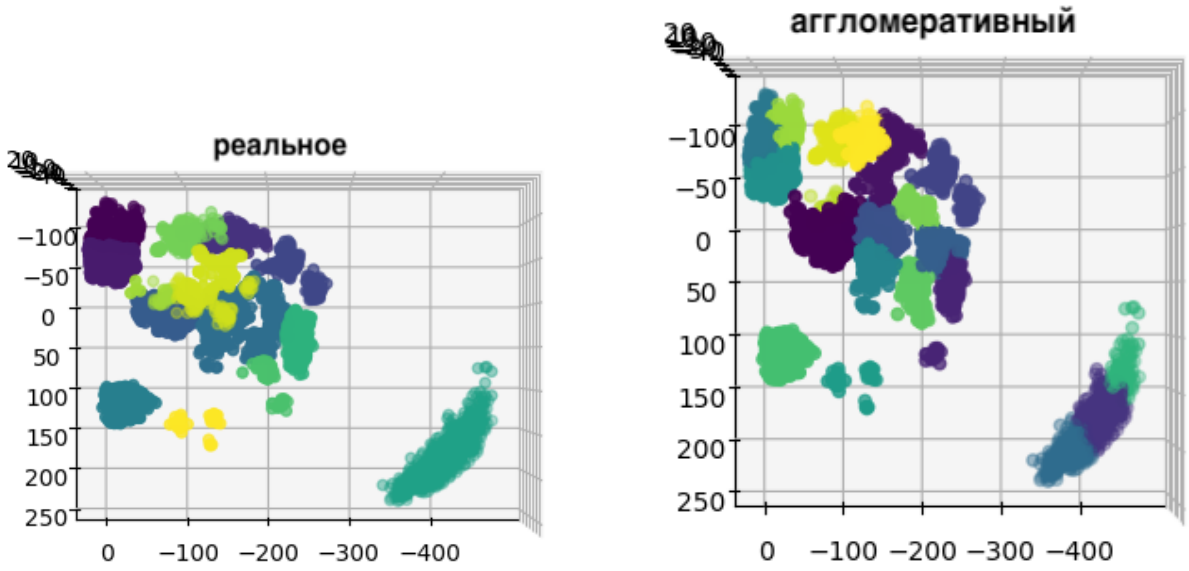
Видно, что для иллюстрации кластеризации мозга MEL 3 алгоритм к-средних справился достаточно хорошо(Рис. 3.1). Он смог разбить, например, область в левом верхнем углу на несколько подобластей. Также стоит отметить, что некоторые кластеры по центру были разбиты совершенно не верно. Тоже самое можно сказать о разбиении мозга SIM 3(Рис. 3.2).

3.1.2. AgglomerativeClustering

Иерархический алгоритм кластеризации, работающий снизу вверх: он объединяет кластеры в соотвествии с некоторым оптимизируемым критерием.

Данный алгоритм был запущен со следующими параметрами:

- количество кластеров - 21. Всего кластеров в одном полушарии 14, за исключением центрального. Данное количество кластеров было выбрано с целью разбиения некоторых сообществ на более мелкие составляющие так, чтобы объединить эти кластеры было несложно руками, но при этом кластеризация получалась более точная с точки зрения внешней метрики.
- алгоритм определяющий связность сообществ на каждой итерации - *ward*. Объединяются кластеры, имеющие наименьшую дисперсию.



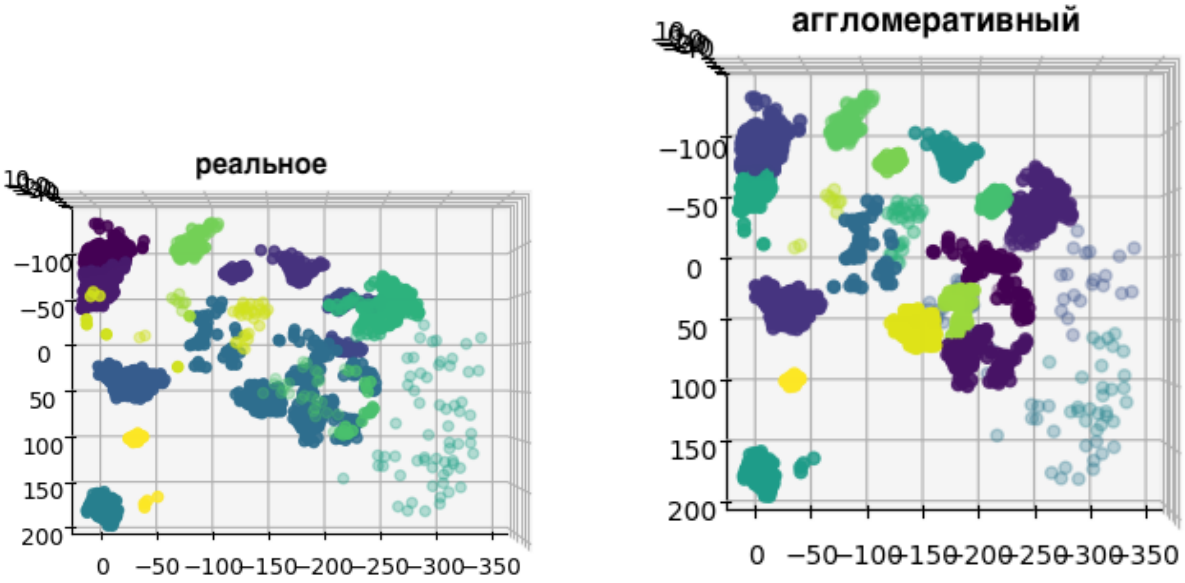
a) Реальное разбиение мозга MEL 3

b) Разбиение мозга MEL 3, полученное алгоритмом *agglomerativeClustering*

Рис.3.3. Сравнение результатов работы алгоритма *agglomerativeClustering* с реальным разбиением мозга мухи MEL 3

3.1.3. BIRCH

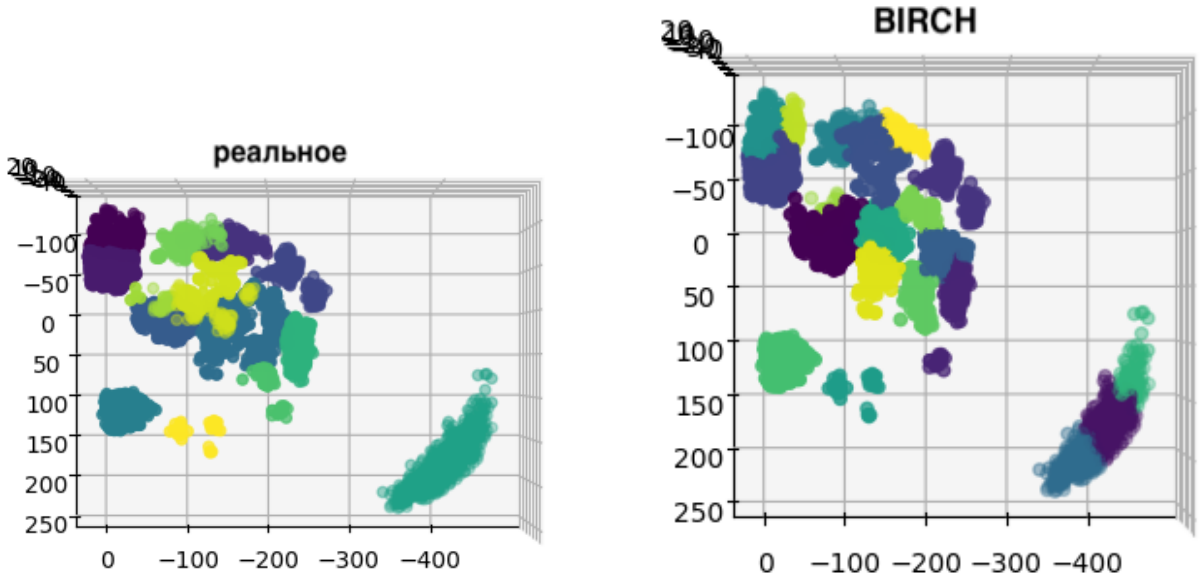
Иерархический алгоритм кластеризации, основным преимуществом которого является масштабируемость[3].



a) Реальное разбиение мозга SIM 3

b) Разбиение мозга SIM 3, полученное алгоритмом *agglomerativeClustering*

Рис.3.4. Сравнение результатов работы алгоритма *agglomerativeClustering* с реальным разбиением мозга мухи SIM 3

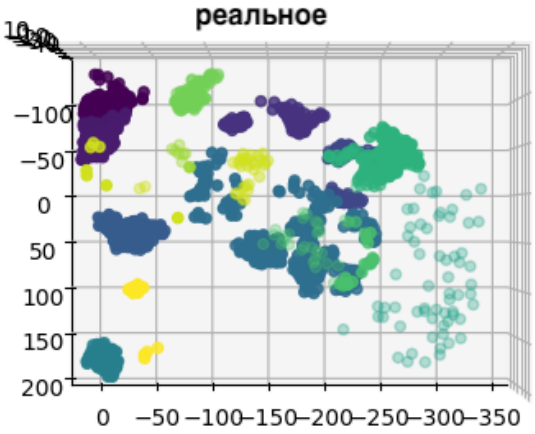


a) Реальное разбиение мозга MEL 3

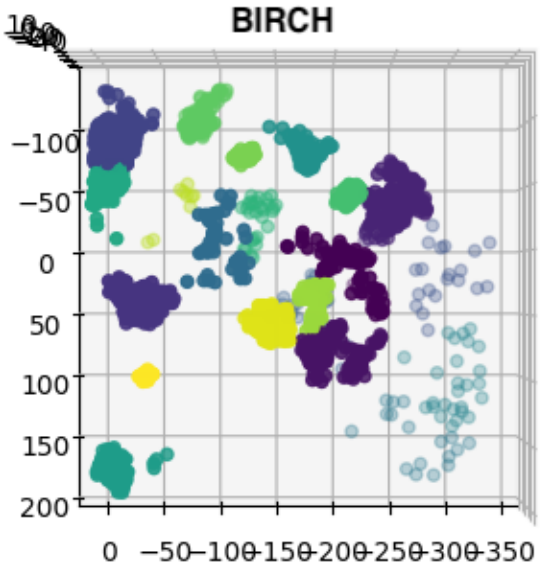
b) Разбиение мозга MEL 3, полученное алгоритмом *BIRCH*

Рис.3.5. Сравнение результатов работы алгоритма *BIRCH* с реальным разбиением мозга мухи MEL 3

У всех классических алгоритмов кластеризации есть существенный недостаток - многие кластеры из экспертного разбиения представляются несколькими кластерами из полученного разбиения. Даже при условии, что значение метрики достаточно высокое это плохо для разбиения в целом(Таблица 3.1). Ведь разделить полученные кластеры, например, при помощи маски - достаточно трудная задача.



a) Реальное разбиение мозга SIM 3



b) Разбиение мозга SIM 3, полученное алгоритмом *BIRCH*

Рис.3.6. Сравнение результатов работы алгоритма *BIRCH* с реальным разбиением мозга мухи SIM 3

Алгоритм	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
K-Means	0.904	0.925	0.909	0.916	0.892	0.916
AgglomerativeClustering	0.922	0.915	0.89	0.949	0.89	0.926
BIRCH	0.916	0.915	0.89	0.94	0.896	0.926

Таблица 3.1

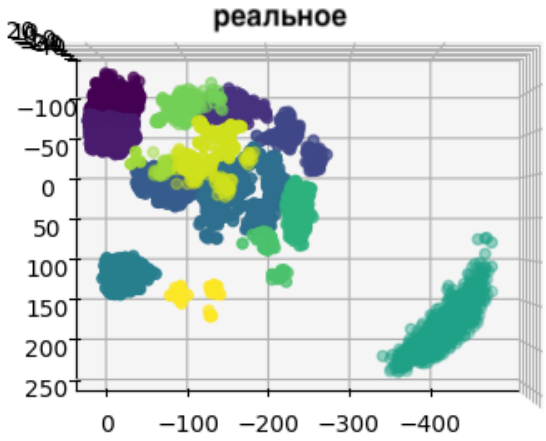
Таблица значений внешней метрики на левых полушариях соответствующего мозга для кластеризаторов

3.2. Классификаторы

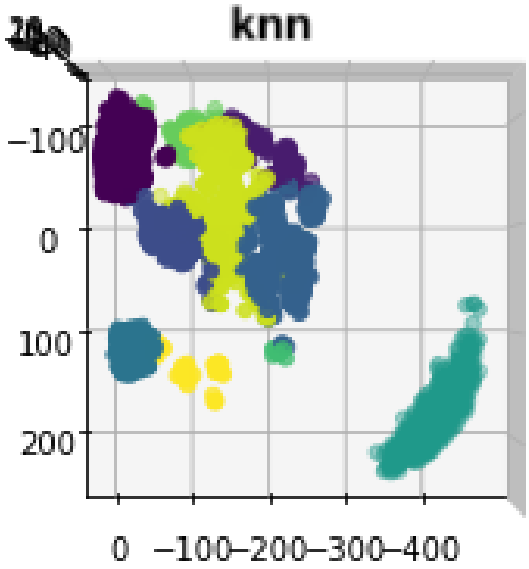
Все классификаторы показали достаточно плохой результат даже по сравнению с классическими алгоритмами кластеризации.

В качестве обучающей выборки бралось полушарие другого модельного организма с соответствующим типом: MEL или SIM.

3.2.1. K-NearestNeighbours

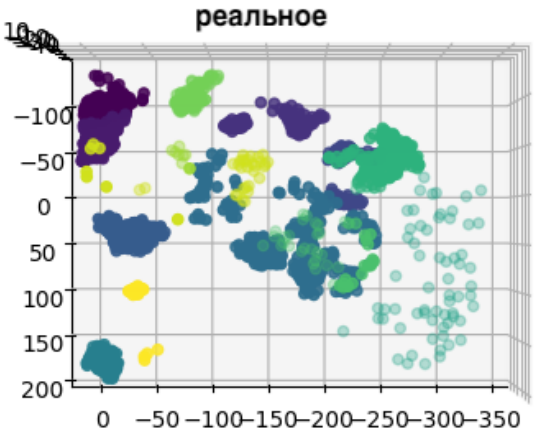


a) Реальное разбиение мозга MEL 3

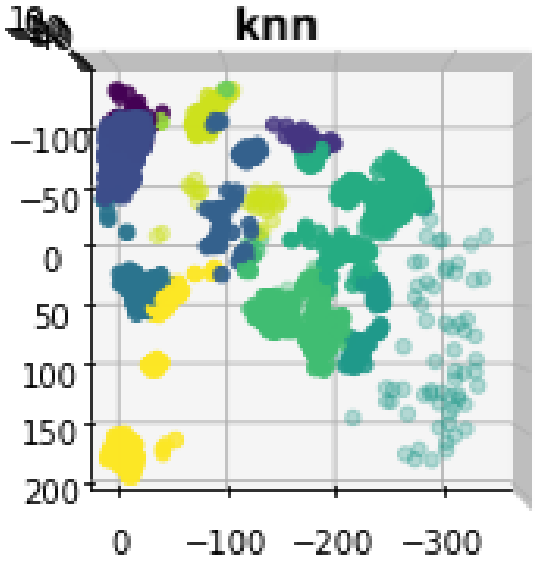


b) Разбиение мозга MEL 3, полученное алгоритмом *KNN*

Рис.3.7. Сравнение результатов работы алгоритма *KNN* с реальным разбиением мозга мухи MEL 3



a) Реальное разбиение мозга SIM 3



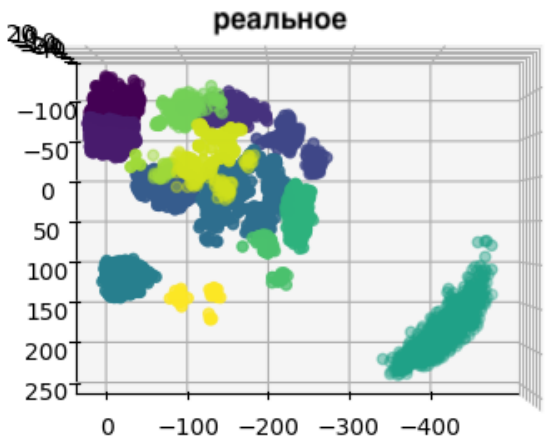
b) Разбиение мозга SIM 3, полученное алгоритмом *KNN*

Рис.3.8. Сравнение результатов работы алгоритма *KNN* с реальным разбиением мозга мухи SIM 3

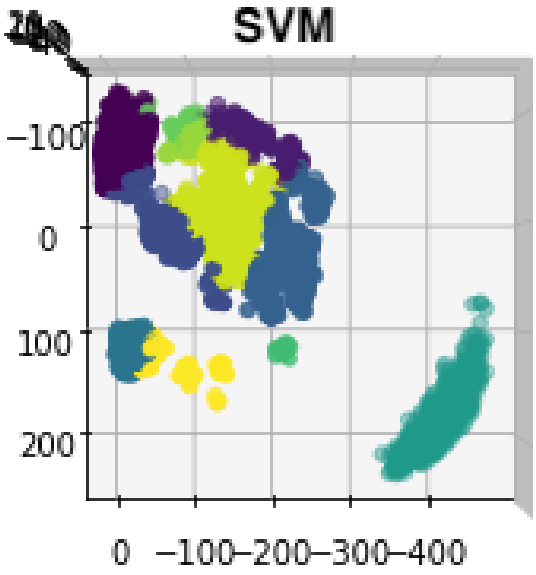
Для мозга MEL 3 видно несколько очевидных на глаз проблем(Рис. 3.7): кластер в верхнем левом углу не разделился на два, центральный ярко-зеленый кластер на реальном разбиении сильно отличается от кластера в полученном разбиении. Мозг SIM 3 классифицировался незначительно лучше мозга MEL 3(Рис. 3.8).

3.2.2. SVM

Здесь приведены результаты классификации при помощи алгоритма *SVM*[19] на мозгах MEL 3 и SIM 3.

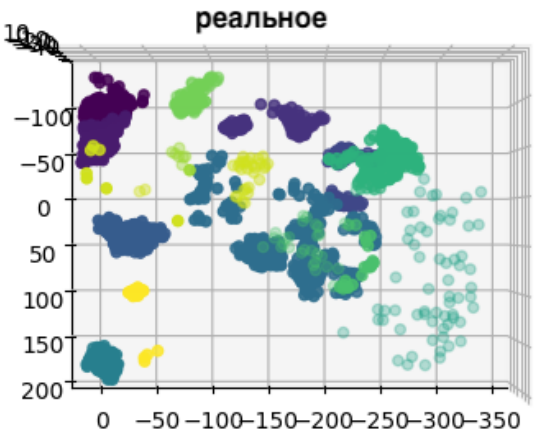


a) Реальное разбиение мозга MEL 3

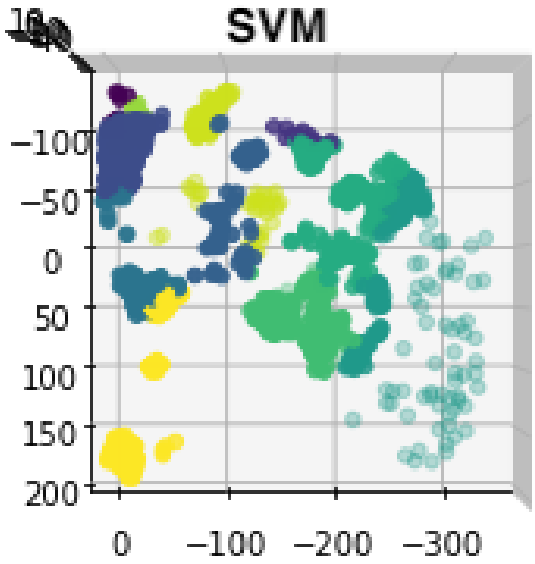


b) Разбиение мозга MEL 3, полученное алгоритмом *SVM*

Рис.3.9. Сравнение результатов работы алгоритма *SVM* с реальным разбиением мозга мухи MEL 3



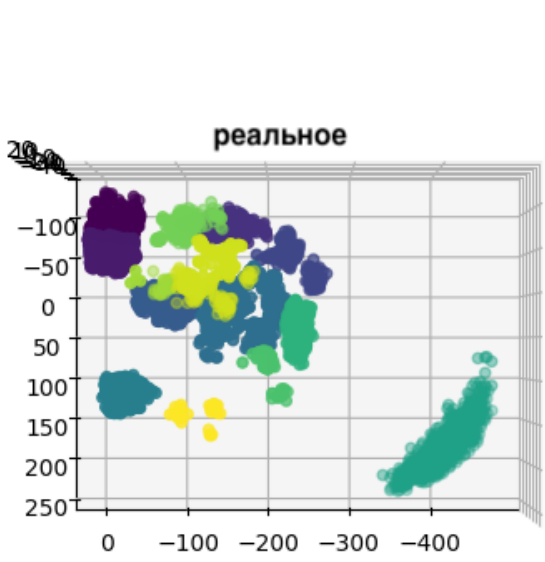
a) Реальное разбиение мозга SIM 3



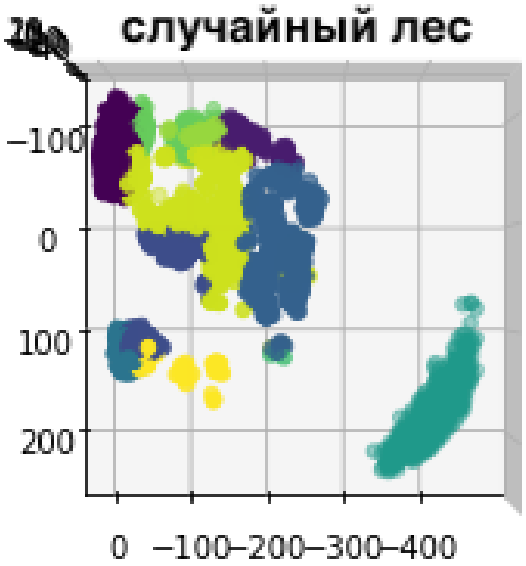
b) Разбиение мозга SIM 3, полученное алгоритмом *SVM*

Рис.3.10. Сравнение результатов работы алгоритма *SVM* с реальным разбиением мозга мухи SIM 3

3.2.3. Random Forest

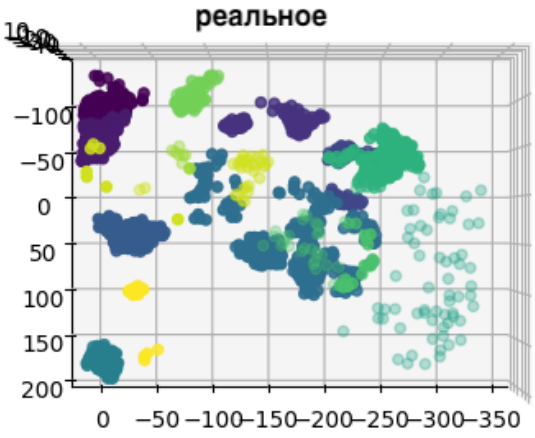


a) Реальное разбиение мозга MEL 3

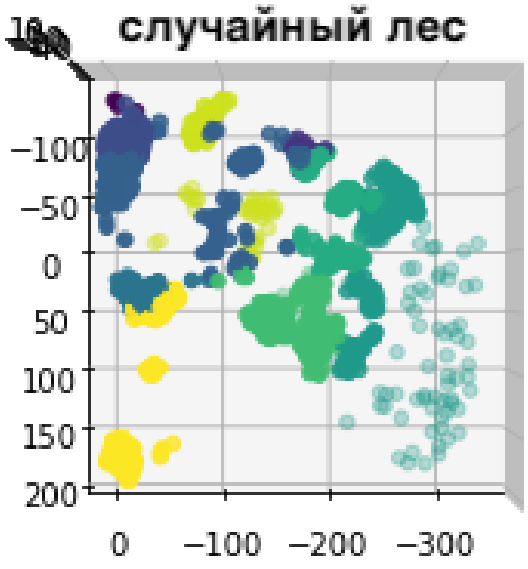


b) Разбиение мозга MEL 3, полученное алгоритмом *RandomForest*

Рис.3.11. Сравнение результатов работы алгоритма *RandomForest* с реальным разбиением мозга мухи MEL 3



a) Реальное разбиение мозга SIM 3



b) Разбиение мозга SIM 3, полученное алгоритмом *RandomForest*

Рис.3.12. Сравнение результатов работы алгоритма *RandomForest* с реальным разбиением мозга мухи SIM 3

В целом полученные разбиения визуально напоминают реальные разбиения, но основной вывод, который можно сделать: симметрии в данных, даже если брать соответствующие полушария разных мозгов мух одной породы - не достаточно,

чтобы получить удовлетворительные(близкие к реальным) разбиения данных. Также стоит отметить, что данный подход не безнадежен. Его можно использовать в дополнение к разработанному методу кластеризации. В результате кластеризации генетическим алгоритмом получается очень много кластеров, некоторые из кластеров реального разбиения подразбиваются на несколько более мелких. При помощи методов кластеризации можно попробовать объединить эти кластеры в 1. Например так: если значение внешней не уменьшилось, то объединяем кластеры в один.

Алгоритм	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
KNN	0.684	0.652	0.679	0.818	0.685	0.65
SVM	0.677	0.707	0.63	0.672	0.658	0.682
RandomForest	0.649	0.658	0.672	0.787	0.59	0.686

Таблица 3.2

Таблица значений внешней метрики на левых полушариях соответствующего мозга для классификаторов

Лучше всех классификаторов(имеется в виду в среднем) показал себя *KNN*, хуже - *SVM*(Таблица 3.2).

3.3. Эксперименты с графопостроителем

Первый опыт, проведенный в этой секции - построение графа с мультиребрами, количество которых интерпретируется, как вес ребра. Были опробованы разные веса: единичные, линейные, квадратные, кубические и экспоненциальные. В итоге веса у ребер были выбраны единичными, другими словами граф стал графом без мультиребер.

Почти на всех мозгах при использовании графа без мультиребер значение внешней метрики значительно превышает значение метрики при остальных инициализациях весов(количества мультиребер). За исключением одной флуктуации: на мозге SIM 4 - линейные веса показали себя лучше остальных(Таблица 3.3). Видимо в реальных данных расстояния между нейронами мозга плодовой мушки не важны. Имеется в виду, что важно наличие связи, но не важна протяженность связи(расстояние между нейронами).

Тип весов	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
единичные	0.908	0.928	0.927	0.918	0.892	0.939
линейные	0.811	0.871	0.823	0.918	0.808	0.956
квадратичные	0.735	0.868	0.823	0.883	0.701	0.926
кубические	0.639	0.865	0.823	0.846	0.608	0.926
экспоненциальные	0.639	0.868	0.823	0.787	0.639	0.91

Таблица 3.3

Таблица значений внешней метрики на левых полушариях соответствующего мозга для генетического алгоритма при различных весах

Второй эксперимент - подбор параметра расстояния Минковского. Лучше всего показало себя манхэттенское расстояние(Таблица 3.4). Четко прослеживается зависимость между увеличением значения параметра расстояния Минковского и уменьшением значения внешней метрики.

Значение параметра	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
Манхэттенское	0.898	0.931	0.928	0.918	0.892	0.95
Евклидово	0.823	0.906	0.823	0.869	0.777	0.932
Бесконечное	0.722	0.861	0.823	0.836	0.704	0.911

Таблица 3.4

Таблица значений внешней метрики на левых полушариях соответствующего мозга для генетического алгоритма при разных значениях параметра p расстояния Минковского

Оба эксперимента независимые, то есть перебирались не все возможные пары - веса и параметр расстояния, при этом алгоритм был генетический.

3.4. Эксперименты с графовыми кластеризаторами

Ниже приведена сводная таблица сравнения времени работы(Таблица 3.6) и значений модульности для алгоритмов Лювейна и Лейдена(Таблица 3.5). Второй получает незначительно большие значения метрики, но при этом работает заметно медленнее первого. Поэтому алгоритм Лювейна в своей стохастической модификации был выбран для инициализации генетического алгоритма.

В среднем для всех полушарий соответствующих мозгов абсолютная разность значений модульности: 0.012.

Алгоритм	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
Лювейн	0.823	0.819	0.787	0.842	0.807	0.884
Лейден	0.832	0.844	0.796	0.848	0.804	0.903

Таблица 3.5

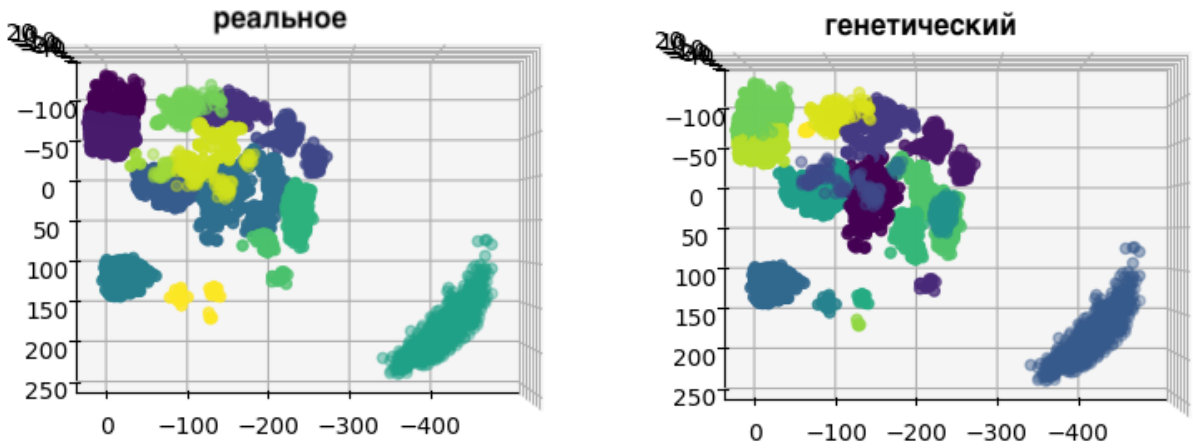
Таблица значений модульности на левых полушариях соответствующего мозга соответствующего графового кластеризатора

Алгоритм	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
Лювейн	220	156	90	153	264	147
Лейден	432	241	152	218	492	205

Таблица 3.6

Таблица времен работы соответствующих алгоритмов

Время работы указано в миллисекундах. В среднем алгоритм Лейдена[2] работает в 1.65 раз медленнее алгоритма Лювейна[1].

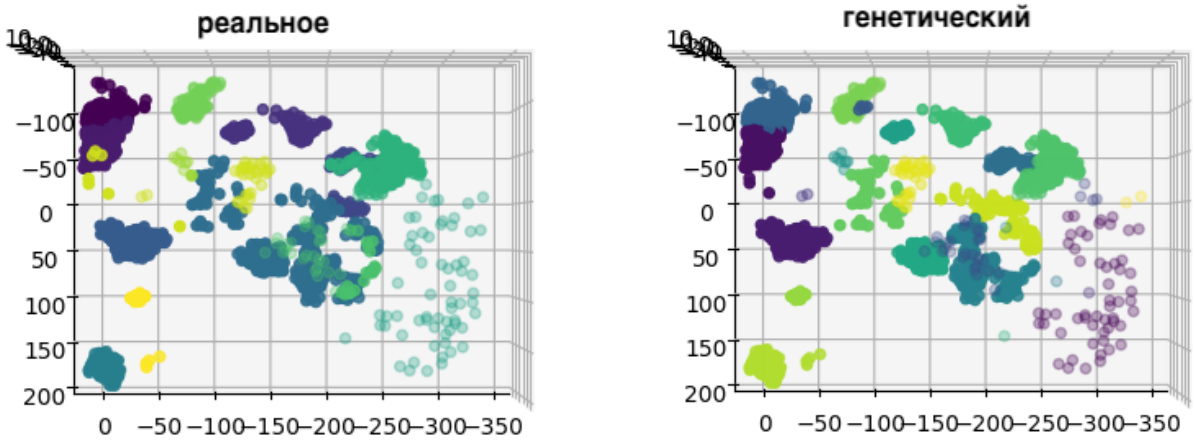


a) Реальное разбиение мозга MEL 3

b) Разбиение мозга MEL 3, полученное генетическим алгоритмом

Рис.3.13. Сравнение результатов работы генетического алгоритма с реальным разбиением мозга мухи MEL 3

Результаты лучшего классического алгоритма кластеризации(Таблица 3.1) и генетического алгоритма сопоставимы по значению внешней метрики(Таблица 3.7). Визуально результаты кластеризации, полученные генетическим алгоритмом сильно похожи на экспертное разбиение. Например, это достаточно хорошо наблюдается на иллюстрации разбиения мозга MEL 3(Рис. 3.13).



a) Реальное разбиение мозга SIM 3

b) Разбиение мозга SIM 3, полученное генетическим алгоритмом

Рис.3.14. Сравнение результатов работы генетического алгоритма с реальным разбиением мозга мухи SIM 3

Алгоритм	*MEL 1*	*MEL 2*	*MEL 3*	*SIM 2*	*SIM 3*	*SIM 4*
Лучший классический(3.1))	0.922	0.925	0.909	0.949	0.896	0.926
Генетический	0.906	0.926	0.927	0.918	0.892	0.94

Таблица 3.7

Сравнение значений внешней метрики лучшего классического алгоритма кластеризации и разработанного генетического алгоритма

На каждой из нижеприведенных иллюстраций приведено разбиение кластеров исходного разбиения кластерами, полученными генетическим алгоритмом(Рис. 3.15). Из данных иллюстраций видно, что есть несколько кластеров полученного разбиения, которые можно считать шумом в силу малости вклада: 3ий, 5ый, 8ой, 12ый, 16ый, 20ый и 26ой. Также стоит отметить, что 5ый кластер исходного был идеально определен, но есть и проблемные кластеры, дающие сопоставимый вклад в несколько кластеров, например, 6ой кластер.

3.5. Результаты распараллеливания генетического алгоритма

В реализованном приложении была выбрана стратегия распараллеливания на основе задач[7]. Это сильно упрощает написание параллельных приложений, так как вся ответственность за распределение данных между потоками, очередностью запуска потоков - ложится на плечи стандартной библиотеки.

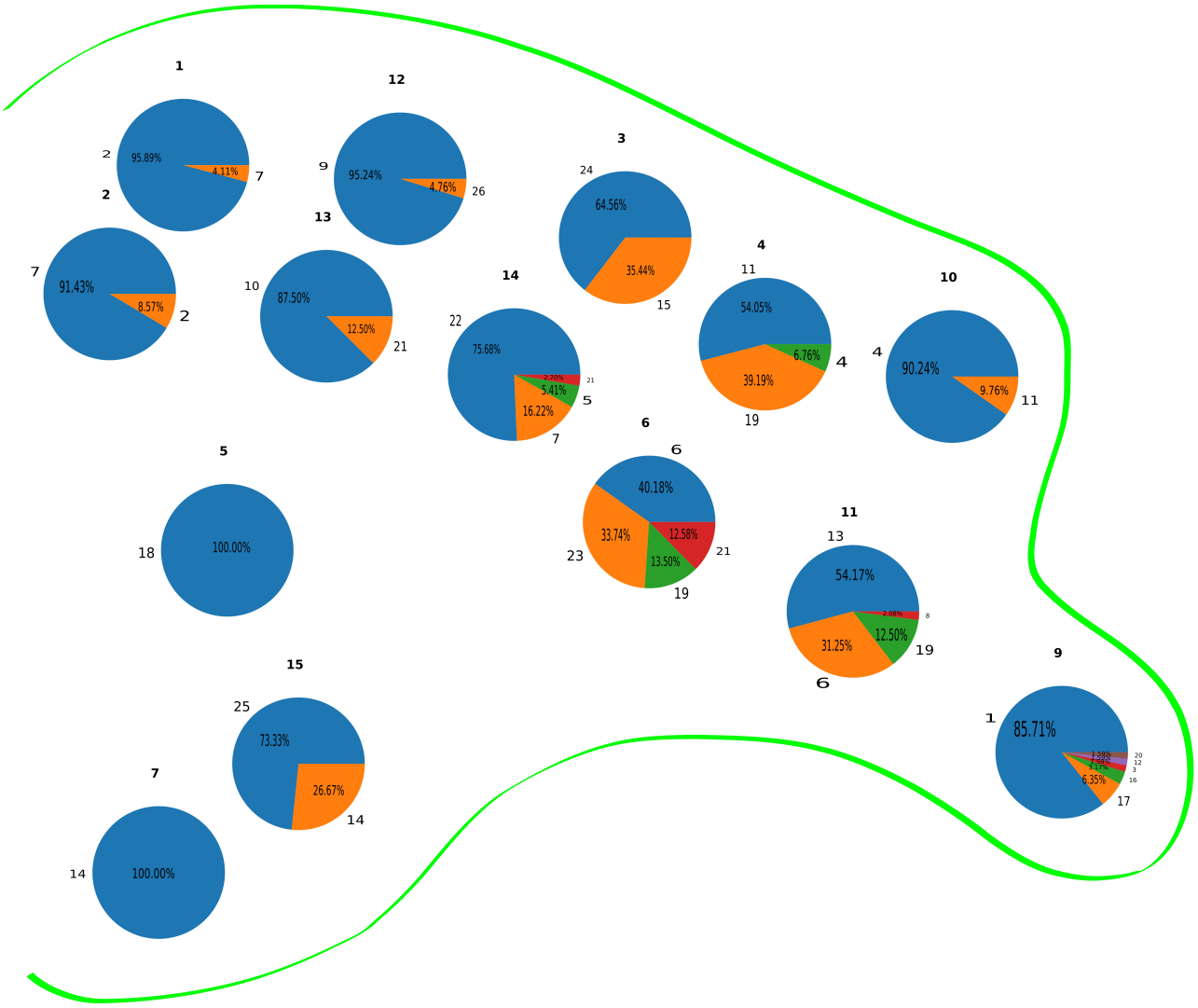
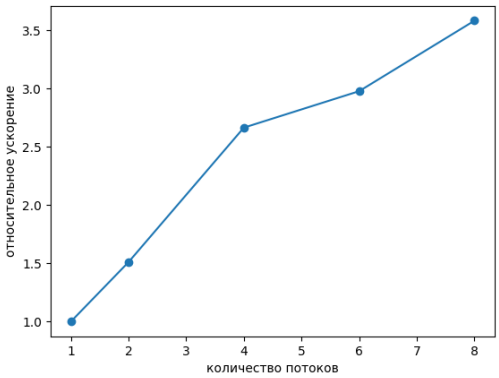


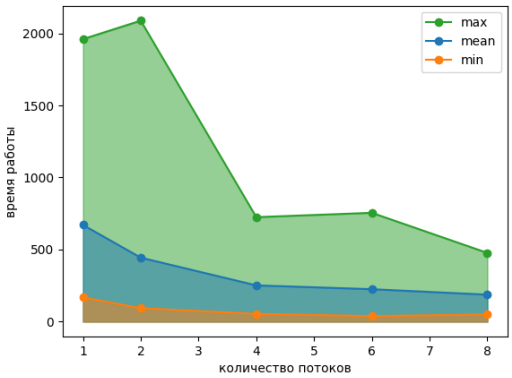
Рис.3.15. Состав каждого кластера реального разбиения

Ниже приведен график времени работы при определенном количестве потоков(Рис. 3.16). Время усреднялось для нескольких запусков при разных значениях некоторых параметров. Эксперименты проводились на рабочей станции 8xIntel(R) Xeon(R) CPU E5-1620 v2 @ 3.70GHz, 64 Гб ОЗУ.

Видно, что с увеличением количества потоков среднее время работы выходит на плато. В какой-то момент среднее время работы должно начать расти. Это связано с накладными расходами на создание потоков и с приближением числа потоков к количеству физических ядер.



a) Относительное ускорение работы генетического алгоритма в зависимости от количества потоков



b) Время работы генетического алгоритма

Рис.3.16. Результаты распараллеливания

ЗАКЛЮЧЕНИЕ

Генетический алгоритм в большинстве случаев показал себя достаточно хорошо. Даже значение внешней метрики на разбиениях, полученных генетическим алгоритмом, достаточно часто превышает значение внешней метрики на разбиениях тех же полушарий, полученных классическими алгоритмами.

Большая часть кластеров генетическим алгоритмом, разработанным в ходе данной работы, определяется достаточно точно. На некоторых мозгах генетический алгоритм получает достаточно много кластеров - 62, но при этом крупных кластеров выделяется немного. Крупным считается кластер, имеющий больше 4 вершин внутри себя. Все не крупные кластеры можно считать шумом в данных. То есть данный алгоритм не только хорошо справляется с точки зрения модульности, внешней метрики и реального разбиения на кластеры(имеется в виду визуальная близость полученного разбиения к экспертному), но и способен находить шум в данных.

В некоторых мозгах два кластера экспертного разбиения представлены одним кластером генетического алгоритма. В дальнейшем можно подобрать параметры графопостроителя так, чтобы кластеров получалось больше и к этому результату применить распознавание шаблонов или накладывания масок. Самый простой способ - для каждого кластера экспертного разбиения найти его центр и среднее квадратическое отклонение точек кластера от центра. Тоже самое проделать для разбиения, полученного генетическим алгоритмом. После этого можно попробовать найти кластеры из генетического алгоритма, которые полностью включаются в кластеры экспертного разбиения. Имеется в виду, что центры находятся в области, которую покрывает кластер из экспертного разбиения, а удвоенный радиус генетического кластера меньше разности радиуса экспертного кластера и расстояния между центрами кластеров.

ВЫВОДЫ

Разработан комбинированный алгоритм(генетический), инициализирующий первую популяцию при помощи алгоритма Лювейна[1], использующий на каждой итерации мутации всех трех видов в следующем порядке: выделяющая, разделяющая, склеивающая, который позволяет автоматически выделять области мозга плодовой мушки по количественным данным об экспрессии генов, извлеченным из трехмерных изображений мозга. В среднем на собранных данных точность получается сравнимая с экспертом(0.919).

Индивидуальные отличия образцов и различия между *Drosophila melonagaster* и *Drosophila simulans* оказывают влияние на эффективность автоматической кластеризации.

Параллелизация алгоритма позволяет сократить время обработки и повысить удобство использования.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

[1] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, J. Stat. Mech. Theory Expr. (2008). Fast unfolding of communities in large networks.

[2] V.A. Traag, L. Waltman, N.J. van Eck. Theory Expr. (2019). From Louvain to Leiden: guaranteeing well-connected communities.

[3] T. Zhang, R. Ramakrishnan, M. Livny. (1997). BIRCH: A New Data Clustering Algorithm and Its Applications. Data Mining and Knowledge Discovery 1, 141–182.

[4] D.Arthur and S. Vassilvitskii. (2007). K-means++: the advantages of careful seeding. 1027–1035.

[5] Rokach, Lior, and Oded Maimon. (2005). Clustering methods. Data mining and knowledge discovery handbook. Springer US, 321-352.

[6] Andreia Sofia Teixeira, Pedro T Monteiro, João A Carriço, Mário Ramirez, Alexandre P Francisco. (2008). Spanning edge betweenness.

[7] Scott Meyers. (2016). Effective C++.

[8] Burns A., Iliffe S. (2009). Alzheimer's disease. BMJ.

[9] James W. Vaupel, Kristín G. v. Kistowski (2005). Broken Limits to Life Expectancy. Science.

[10] Alan Mullan, Aleksandra Marsh. (2019). Advantages of using Drosophila Melanogaster as a Model Organism.

[11] Billeter, Jean-Christophe, Adriana Villella, Jane B. Allendorfer, Anthony J. Dornan, Michael Richardson, Donald A. Gailey, Stephen F. Goodwin. (2006). Isoform–Specific Control of Male Neuronal Differentiation and Behavior in Drosophila by the Fruitless Gene. Current Biology.

[12] Schubert, E., Sander, J., Ester, M., Kriegel, H. P., Xu, X. (2017). DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. ACM Transactions on Database Systems.

[13] Ankerst, Mihael, Markus M. Breunig, Hans-Peter Kriegel, Jörg Sander. (1999). OPTICS: ordering points to identify the clustering structure. ACM Sigmod Record, 49-60.

[14] Knuth D. (1974). Postscript about NP-hard problems. ACM SIGACT News, 15–16.

[15] Asai Nobuhiro, Kubo Izumi, Kuo Hui-Hsiung. (2000). Bell numbers, log-concavity, and log-convexity. Acta Applicandae Mathematicae, 79–87.

[16] Baldwin C.Y., Clark K.B. (2000). Design Rules: The power of modularity 63–92.

[17] Williams, Jr., Louis F. (1976). A modification to the half-interval search (binary search) method. Proceedings of the 14th ACM Southeast Conference, 95–101.

[18] M. Girvan, M. E. J. Newman. (2002). Community structure in social and biological networks. National Acad Sciences.

[19] Cortes Corinna, Vapnik Vladimir N. (1995). Support-vector networks. Machine Learning, 273–297.

[20] Ho, Tin Kam. (1995). Random Decision Forests. Proceedings of the 3rd International Conference on Document Analysis and Recognition 278–282.