МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«Сибирский федеральный университет»

На правах рукописи

Казаковцев Владимир Львович

АЛГОРИТМЫ УСКОРЕННОГО ПОИСКА В ВЕКТОРНЫХ БАЗАХ ДАННЫХ

2.3.1 - Системный анализ, управление и обработка информации, статистика

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель:
доктор технических наук,
профессор Ступина А.А.

Красноярск – 2026

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность. Задачи приближенного поиска ближайших соседей получили свое развитие в связи с широким распространением методов представления данных в виде высокоразмерных векторных признаков (наборов числовых характеристик объектов) и стремительным ростом объемов информации, обрабатываемой в современных информационных и интеллектуальных системах. Во многих прикладных областях, таких как семантический поиск, рекомендательные сервисы, анализ изображений и аудиосигналов, обнаружение аномалий и так далее, требуется многократное выполнение запросов на поиск близких объектов по метрике, однако точные методы становятся вычислительно затратными при большом количестве объектов в наборе данных и высокой размерности пространств признаков. Приближенные алгоритмы позволяют существенно сократить время ответа и потребление памяти при контролируемом снижении качества результатов, обеспечивая практическую реализуемость решений в условиях ограничений по задержке и стоимости вычислений.

Некоторые алгоритмы приближенного поиска ближайших соседей используют методы автоматической группировки данных, также называемые алгоритмами кластеризации. Эти алгоритмы решают задачу кластеризации, которая не имеет строгого формального определения, но неформальным образом может быть описана как задача разделения множества объектов на подмножества таким образом, чтобы объекты в одном подмножестве (кластере) были как можно сильнее схожи друг с другом и как можно сильнее отличались от объектов в других подмножества. Алгоритмы кластеризации входят в число базовых методов машинного обучения и анализа данных, которые находят широкое применение в различных областях, включая обработку результатов научных экспериментов, интернет-технологий, в частности обработку данных социальных сетей, обработку медицинских данных, финансы и другие. Рост объема и размерности собираемых и сохраняемых данных требует специальных методов и моделей для

их эффективной обработки. Кластеризация упрощает анализ больших и сложных для ручной обработки данных, структурируя их и позволяя исследователям и практикам сосредоточиться на значимых группах объектов, а также позволяет находить закономерности в данных и выявлять взаимосвязи между объектами.

Одним из методов приближенного поиска ближайших соседей, использующим кластеризацию, является инвертированный индекс (англ. Inverted File Index, IVF). Метод инвертированного индекса заключается в том, что все объекты в базе данных разбиваются на кластеры, и каждый кластер характеризуется его центроидом, которому в соответствие ставится список всех объектов внутри кластера. При поиске ближайших к вектору-запросу объектов базы данных на первом шаге выбирается несколько ближайших к вектору-запросу центроидов, а сам поиск ближайших соседей проводится не по всей базе данных, а внутри кластеров, определяемых этими центроидами, что позволяет значительно сократить количество необходимых для обработки запроса вычислительных ресурсов. Приближенный поиск ближайших соседей играет важную роль в современных информационных системах, таких как рекомендательные сервисы, компьютерное зрение и анализ данных. В условиях постоянно растущих объемов данных и задач поиска по сложным многомерным структурам, методы точного поиска не могут обеспечить приемлемую скорость поиска. Актуальность методов приближенного поиска обуславливается необходимостью быстрой обработки больших массивов информации, которые позволяют ускорить время отклика, сохраняя приемлемую точность поиска.

Развитие носителей информации и интернет-технологий ставит новые задачи, в том числе задачу обработки мультимодальных данных большого объема. Мультимодальными данными называют данные, представленные одновременно в двух и более модальностях (разнородных видах представления информации), например текст, изображение, аудио или видео, относящиеся к одному и тому же объекту или событию.

Изложенное выше свидетельствует о перспективности развития алгоритмов автоматической группировки данных для решения широкого круга задач, включая

разработку методов кластеризации мультимодальных данных и алгоритмов приближенного поиска ближайших соседей.

Степень научной разработанности проблемы. Среди важнейших алгоритмов поиска ближайших соседей можно выделить предложенный Ю. Мальковым и Д. Яшуниным HNSW (англ. Hierarchical Navigable Small World), который использует многослойную иерархическую структуру графов. Другим наиболее распространенным алгоритмом является алгоритм поиска по обратному индексу (англ. Inverted File Index, IVF), который для ограничения области поиска использует кластерную структуру данных. Сложно выделить одного или нескольких авторов этого алгоритма, однако важный вклад в развитие этого подхода внесли такие ученые, как Л. Ванг, А. Бабенко, С. Брух и другие.

Самой распространенной моделью кластеризации является модель *k*-средних, предложенная в 1957 году Г. Штейнгаузом. С. Ллойд позднее предложил алгоритм, который известен как алгоритм Ллойда или процедура *k*-средних. Значительный вклад в развитие теории кластеризации внесли Ц. Дрезнер, О. Алп, Э. Эркут и Н. Младенович, считающиеся сегодня классиками в этой области. А.Н. Антамошкин и другие отечественные ученые развили теорию эвристических алгоритмов кластеризации. Также важно отметить Л. Кауфмана и П. Руссива, представивших модель *k*-медоид, а также Х. Хамахера, П. Хансена и Ю. Кочетова, которые развивали метод чередующихся окрестностей (Variable Neighbourhood Search). В.В. Шаламовым, В.А. Ефимовой, С.Б. Муравьевым и А.Ф. Фильченковым был разработан алгоритм для автоматического выбора алгоритма кластеризации и настройки его гиперпараметров MASSCAH (Multi-armed simultanous selection of clustering algorithm and its hyperparameters).

Красноярской научной школой: А.Н. Антамошкиным, Л.А. Казаковцевым, А.А. Ступиной, Г.Ш. Шкабериной и др. был внесен значительный вклад в развитие эволюционных алгоритмов кластеризации, жадных агломеративных эвристических процедур, различных операторов мутации. В.И. Голованом, В.В. Орловым и Л.А. Казаковцевым были развиты методы выделения однородных

партий электрорадиоизделий для систем с повышенными требованиями к качеству, однородности и отказоустойчивости микросхем.

К. Ли, М. Чжаном, Д.Г. Андерсеном и Ю. Хе предложен алгоритм адаптивного поиска приближенных ближайших соседей для методов HNSW и IVF. Авторы рассматривают расстояния между объектами и центрами кластеров и исходя из этих данных их алгоритм определяет глубину поиска.

Обработка мультимодальных данных является быстро развивающейся областью, в которой сложно выделить основные, фундаментальные работы. Ч. Чен и др. предложили метод агрегации модальностей на основе энкодер-декодерной архитектуры, использующий для агрегации модальностей метод глубокого обучения, то есть многослойные нейронные сети, требующие обучения на размеченных данных и настройки параметров обучения. Другие подходы, также использующие глубокое обучение, были предложены в работах М. Аль Рахала, В. Ванга и других. В. Ванг с соавторами показал достаточно эффективную модель автоэнкодера для объединения нескольких модальностей в единое векторное представление (эмбеддинг), однако, такая модель также требует обучения или дообучения на размеченных данных, а также не может быть эффективно реализована на вычислительном кластере. Н. Шривастава и Р.Р. Салахутдинов предложили для создания единого векторного представления использовать две машины Больцмана, то есть две рекуррентных нейронных сети, которые также требуют обучения на размеченных данных и больших вычислительных ресурсов. Метод ADAPT, разработанный Г.С. Лбовым и соавторами в 1980-х гг., является принципиально иным подходом: все пространство поиска представляется как конечное множество, где каждая переменная может принимать конечный набор значений.

Несмотря на обширные накопленные знания, методики и технологии, в существующих методах кластерного анализа есть недостатки. Так, современные задачи требуют агрегации разных типов данных, так как информация об одиночном объекте может быть представлена более чем одним вектором. Также следует заметить, что в области эволюционных алгоритмов есть поле для

исследований: могут быть разработаны более точные и стабильные алгоритмы, а также более специализированные алгоритмы кластеризации на основе эволюционных алгоритмов.

Целью настоящей работы является повышение эффективности алгоритмов приближенного поиска ближайших соседей.

Для достижения цели были поставлены следующие **задачи:**

1. Провести анализ проблем, возникающих при применении методов приближенного поиска ближайших соседей, а также анализ методов, позволяющих повысить быстродействие алгоритмов приближенного поиска ближайших соседей без потери точности.

2. Разработать алгоритм классификации запросов приближенного поиска ближайших соседей с использованием IVF-индекса, основанный на оценке доли эффективных (результативных) кластеров на начальных этапах поиска, позволяющий определить сложность запроса и предсказать количество кластеров, в которых находятся ближайшие соседи.

3. Разработать адаптивный алгоритм поиска данных в векторной базе данных на основе IVF-индекса с использованием классификатора сложности запросов по результатам предварительного поиска, ускоряющий процесс поиска ближайших соседей без потери качества.

4. Разработать алгоритмы на основе жадной агломеративной процедуры, которые бы представляли компромисс между достигаемым значением целевой функции и необходимыми для выполнения вычислительными и временными ресурсами, а также работали стабильно и конкурентоспособно по сравнению с другими алгоритмами (по внешним и внутренним мерам качества кластеризации).

5. Разработать меру расстояния, агрегирующую расстояния между объектами, рассчитанные по отдельным модальностям, не требующую единого для всех модальностей векторного представления.

6. На основе разработанной меры расстояния построить модель кластеризации, позволяющую применять алгоритмы кластеризации и

приближенного поиска ближайших соседей к мультимодальным данным без использования моделей глубокого обучения.

Методология и методы исследования. В качестве методологической базы исследования использовались существующие работы по алгоритмам приближенного поиска ближайших соседей, построению индекса в векторных базах данных и кластерному анализу (из областей машинного обучения и анализа данных), компьютерное моделирование, методы математической статистики (включая оценку распределений и тесты значимости), системного анализа, теории размещения и теории оптимизации (из сферы исследования операций и оптимизационных задач).

Новые научные результаты:

1. Предложен новый алгоритм классификации запросов по уровню сложности для приближенного поиска ближайших соседей с использованием IVF-индекса, позволяющий определять требуемую для достижения целевого показателя полноты область поиска, на основе числа результативных кластеров после начального этапа поиска.

2. Предложен новый адаптивный алгоритм поиска ближайших соседей в векторной базе данных на основе IVF-индекса, отличающийся от известных использованием классификатора сложности запросов на основе результатов предварительного поиска, использование которого позволяет повысить среднюю эффективность поиска.

3. Предложены новые эволюционные алгоритмы решения задачи *k*-средних, отличающиеся от известных оператором мутации, основанным на ускоренной жадной агломеративной процедуре, и позволяющие повысить точность решения задачи кластеризации.

4. Предложена новая модель кластеризации мультимодальных данных, которая, в отличие от известных моделей, позволяет напрямую, без приведения к единому векторному виду, применять алгоритмы кластеризации к мультимодальным данным.

Положения, выносимые на защиту:

1. Предложен классификатор сложности запросов, позволяющий оценить число кластеров, поиск в которых в среднем обеспечивает требуемое значение полноты (Recall) для каждого класса запросов. Классификатор обеспечивает точность (accuracy) определения класса сложности запроса на уровне 0,81.

2. Новый адаптивный алгоритм поиска ближайших на основе IVF-индекса обеспечивает ускорение выполнения запросов на 10–30% на наборах данных до 1 миллиарда объектов.

3. Новые алгоритмы решения задачи *k*-средних позволяют более точно решать задачу *k*-средних, за счет чего обеспечивается построение IVF-индекса, который позволяет ускорить выполнение запросов на 0,5-1%.

4. Новая модель автоматической группировки мультимодальных данных позволяет напрямую применять классические алгоритмы кластеризации с эффективностью не менее 70% (по индексу Рэнда) без построения дополнительных к существующим векторным представлениям модальностей структур данных.

Теоретическая значимость результатов. Теоретическая значимость работы состоит в развитии методов автоматической группировки объектов, развитии методов поиска данных в векторных базах данных, а также в расширении инструментария методов кластерного анализа для мультимодальных данных, в том числе больших мультимодальных данных.

Практическая значимость результатов. Предложенные модели и алгоритмы могут применяться для задач разделения объектов на группы и поиска данных в информационных системах, работающих с разнородными наборами данных в прикладных областях. Разработанный эволюционный алгоритм и адаптивный алгоритм поиска могут эффективно применяться в системах управления векторными базами данных. Использование эволюционных алгоритмов типа 1+λ, а также упрощенного жадного алгоритма позволяет снизить долю неверно определенных ближайших соседей в векторных базах данных, хранящих эмбеддинги. Разработанный адаптивный алгоритм приближенного

поиска ближайших соседей и классификатор сложности запросов применяются в векторных базах данных с сотнями миллионов объектов для приближенного поиска ближайших соседей. Разработанная мера расстояния в пространстве мультимодальных данных позволяет вычислять расстояния между мультимодальными объектами, что позволяет не только адаптировать существующие алгоритмы кластеризации для такого пространства, но и сравнивать расстояния между собой, делая возможным приближенный поиск ближайших соседей в пространстве мультимодальных данных.

Исследование было выполнено при поддержке Министерства науки и высшего образования Российской Федерации в рамках государственного задания №FEFE-2020-0013 «Развитие теории самоконфигурирующихся алгоритмов машинного обучения для моделирования и прогнозирования характеристик компонентов сложных систем», при поддержке Мегагранта «Гибридные методы моделирования и оптимизации в сложных системах»№075-15-2022-11-21, а также гранта Фонда Содействия Инновациям по программе «Код-ИИ» и хозяйственного договора с техкомпанией «Хуавей» (разработка эффективного алгоритма поиска объектов в векторной базе данных).

Апробация. Основные положения и результаты диссертационной работы докладывались на международных конференциях и семинарах: научный семинар Омского филиала института математики им. С.Л. Соболева СО РАН «Математическое моделирование и дискретная оптимизация» (2026), International Workshop on Mathematical Models and their Applications (IWMMA 2025), Hybrid methods of modeling and optimization in complex systems (HMMOCS 2022, 2024), 2021 3rd International Conference on Advanced Information Science and System, AISS 2021, 2020 International Conference on Control, Robotics and Intelligent System, CCRIS 2020.

Структура работы. Диссертационная работа изложена на 142 страницах и состоит из введения, четырех глав, заключения, списка литературы из 150 источников, 23 таблиц, 26 рисунков и двух приложений.

1 МЕТОДЫ ПРИБЛИЖЕННОГО ПОИСКА БЛИЖАЙШИХ СОСЕДЕЙ

1.1 Постановка задачи приближенного поиска ближайших соседей

Поиск ближайшего соседа (NNS) – задача оптимизации, которая включает в себя поиск вектора, наиболее похожего на заданный вектор (вектор запроса) среди векторов в исходном наборе данных (векторов). Сходство между объектами обычно определяется функцией различия, которая определяет расстояние между любыми векторами и классами в метрическом пространстве R^M. Это пространство состоит из набора векторов с функцией расстояния $d(x_i, q_i)$. Чем больше значение функции расстояния, тем меньше сходство между вектором запроса и вектором из исходного набора данных. Расстояние порядка p между двумя точками определяется с помощью расстояния Минковского (l^p-нормой) $d_p(x_i, q_i) = \left(\sum_{i=1}^{M}|x_i - q_i|^p\right)^{\frac{1}{p}}$ [1], где x и q – векторы характеристик, а M – размерность вектора. Разновидности расстояния Минковского зависят от параметра p [1, 2]: p=1 соответствует манхэттенскому расстоянию, а p=1 соответствует расстоянию Евклида.

Задача поиска ближайшего соседа может быть описана в метрическом пространстве следующим образом [1, 3]: Дано множество N точек и функция расстояния $d(x_i, q_i)$ между точками x_i, q_i. Пара (N, d) определяет метрическое пространство, если d обладает такими характеристиками, как рефлексивность, неотрицательность, симметрия и в нем выполняется неравенство треугольников. Более подробная постановка задачи в евклидовом пространстве приведена ниже [1].

Точный NNS: Дано множество N точек в M-мерном пространстве R^M ($N \subset R^M$). Необходимо построить структуру данных, которая для любой заданной точки $q \in R^M$ находит точку из N, находящуюся на наименьшем расстоянии до q.

Это определение для небольшого набора данных с низкой размерностью описывает задачу, алгоритмы решения которой имеют сублинейную (или даже

логарифмическую) временную сложность выполнения запроса, но для большого набора данных с высокой размерностью сложность обработки запроса растет экспоненциально. Аппроксимация может уменьшить экспоненциальную сложность до полиномиальной [4]. Ближайший сосед в таком случае [1] определяется следующим образом.

Приближенный NNS: Дано множество N точек в M-мерном пространстве R^M ($N \subset R^M$). Необходимо построить структуру данных, которая для любой заданной точки $q \in R^M$, находит точку из N, находящуюся на расстоянии не более чем в c раз дальше, чем до p, где p – это ближайшая до запроса точка из N.

Для приближенного NNS наиболее часто используемым показателем качества является полнота (обозначается Recall@K), которая представляет собой долю истинных (точных) ближайших соседей среди K найденных соседей.

В современной литературе предлагаются два подхода к решению задачи поиска ближайшего соседа, которые можно разделить на точные (линейные методы, методы разбиения пространства) и приближенные (графовые методы, методы, основанные на хешировании, файлы аппроксимации векторов и поиск, основанный на сжатии или кластеризации).

Идея линейного подхода заключается в последовательном вычислении расстояния от вектора запроса до каждого вектора в наборе данных. Вычислительная сложность этого подхода пропорциональна количеству и размерности данных, $O(MN)$, где M – это размерность вектора, а N – общее количество векторов в наборе данных [5]. Как результат, производительность заметно снижается с ростом размерности [6, 7]. Хотя существуют некоторые эвристики для снижения вычислительных затрат [5], этот метод плохо масштабируется для задач высокой размерности и больших объемов.

Для решения задачи поиска сходства в многомерных векторных пространствах были предложены различные древовидные структуры для разбиения пространства, такие как KD-деревья и их вариации [8, 9, 10], R-деревья и их вариации [11, 12, 13], B+-деревья [14, 15, 16] и деревья покрытий [17, 18]. Основная идея этих структур заключается в сужении пространства поиска путем

разбиения его плоскостями или выровненными по осям областями. Расстояния рассчитываются только для соседей, которые лежат по ту же сторону плоскости, что и вектор запроса. Такая древовидная организация данных уменьшает количество кандидатов для поиска ближайшего соседа. Однако производительность этих методов также ухудшается для многомерных и крупномасштабных данных, поскольку построение таких структур данных обычно требует значительного времени и памяти [5].

В [19] авторы исследуют влияние размерности на эффективность поиска ближайшего соседа. С увеличением размерности возникает так называемое «проклятие размерности»: дисперсия расстояний между объектами снижается, расстояния между ближними объектами приближаются к расстояниям между самыми удаленными объектами. Для преодоления «проклятия размерности» был предложен подход, основанный на приближенном поиске ближайшего соседа (англ. Approximate Nearest Neighbours search, ANN, ANNs) [6]. Ключевое преимущество ANN заключается в способности эффективно вычислять приближенные решения, что приводит к существенному повышению скорости и сокращению использования памяти по сравнению с методами точного поиска. В некоторых случаях размер набора данных делает полностью точный поиск вычислительно невозможным в течение разумного времени. В других ситуациях разница между точным и приближенным результатами поиска ближайшего соседа может быть минимальной или незначительной

В [20, 21, 22] авторы представили методы графов близости, такие как Navigable Small Worlds (NSW), а затем расширили этот подход с помощью Hierarchical Navigable Small Worlds (HNSW) в [23]. Эти методы основаны на концепции малого мира (Small World), которая относится к свойству графов, где каждый узел имеет ближние связи со своими соседями (обычно около $\log(N)$ шагов), а также несколько дальних связей, которые способствуют эффективному глобальному исследованию. Алгоритм HNSW создает многослойную иерархическую структуру графа для эффективной индексации и поиска ближайших соседей в многомерных пространствах. В отличие от точного

алгоритма *K*-ближайших соседей (KNN) [24], который выполняет полный поиск данных, HNSW хорошо масштабируется для больших наборов данных.

Методы на основе хеширования представляют собой еще один подход к решению проблемы поиска приближенного ближайшего соседа [25]. В этой парадигме многомерные точки данных отображаются в компактные двоичные или целочисленные представления, называемые хеш-кодами, которые существенно короче по длине исходных векторов признаков. Хотя методы хеширования обеспечивают значительный выигрыш в эффективности с точки зрения как скорости вычислений, так и использования памяти, определенная степень потери информации, по сути вносится в процесс кодирования, особенно когда данные проецируются в пространство меньшей размерности. Алгоритмы хеширования для ANN-поиска можно в целом разделить на две основные категории: независимые от данных методы, такие как локально-чувствительное хеширование (Locality-Sensetive Hashing, LSH) [26], и зависимые от данных подходы, такие как локально-сохраняющее хеширование (Locality-Preserving Hashing, LPH) [23]. LSH [26] обеспечивает эффективный поиск похожих элементов в крупномасштабных базах данных за счет увеличения вероятности коллизий хеш-функций для соседних точек данных. Этот метод относится к категории недетерминированных алгоритмов, которые не дают детерминированных гарантий точности результата. Вместо этого они обеспечивают высокую вероятность получения точного или достаточно точного приближенного решения.

Приближенный поиск ближайших соседей – важный инструмент для работы с большими объемами данных в современных базах данных, применяемый для поиска схожих объектов. В условиях экспоненциального роста объемов многомерных данных (векторы признаков в компьютерном зрении, в задачах обработки естественного языка, в рекомендательных системах) точный поиск k ближайших соседей посредством полного сканирования становится вычислительно неприемлемым. Приближенный поиск ближайших соседей применяется в интерактивных системах, где критична скорость ответа.

В работах [27, 28] авторы представили эффективный метод индексации для многомерных баз данных, использующих ANN поиск для обработки запросов, основанный на методе файла аппроксимации векторов (VA-файла). Этот подход к фильтрации поддерживает эффективный поиск ближайшего соседа, разбивая векторное пространство гиперплоскостями для аппроксимации расстояний между кластерами. Другие исследователи представили файл аппроксимации частичных векторов [29], предназначенный для эффективной обработки запросов на частичное сходство в любом подпространстве, даже если конкретное подпространство заранее неизвестно.

Другие методы поиска с помощью ANNs для многомерных данных включают кодирование данных в компактные коды на основе векторного квантования [30]. В основе этого подхода лежит сжатие или кластеризация. Пространство данных сначала разлагается в декартово произведение некоторых подпространств низкой размерности, а затем объекты квантуются в каждом подпространстве отдельно [25]. Многие работы, посвященные изучению поиска с помощью ANN, основаны на Product Quantization (PQ) [25, 31-35].

В [36] авторы представили подход к адаптивному поиску методом приближенного ближайшего соседа, который использует только статические признаки запроса. В отличие от AdaptNN [37], который опирается на характеристики времени выполнения, алгоритм [36] предсказывает условия завершения поиска на основе локальной внутренней размерности (LID) запроса до начала выполнения. Это устраняет трудоемкий процесс выбора признаков и упрощает обучение модели. Этот алгоритм интегрирован с двумя ведущими подходами индексации на основе ANN: IMI (Inverted Multi-Index) [38] и HNSW (Hierarchical Navigable Small World) [39].

В векторных базах данных (например, PostgreSQL pgvector [40] и других) поиск включает в себя поиск данных, приблизительно соответствующих вектору запроса [41]. Цель поиска состоит в том, чтобы определить векторы данных, наиболее близкие к вектору запроса, используя алгоритмы приближенного ближайшего соседа [42]. Основное преимущество такого подхода заключается в

его способности значительно повысить производительность поиска при незначительном снижении точности запроса.

В отличие от автономных алгоритмов поиска на основе ANN, алгоритм поиска на основе ANN, реализованный в составе СУБД, не может хранить индекс полностью в оперативной памяти, поскольку СУБД может испытывать значительные параллельные нагрузки, связанные с множеством различных таблиц и индексов. Кроме того, СУБД часто считывает данные с диска через свой менеджер буферов небольшими блоками (4–16 КБ). Блоки такого размера не могут хранить много многомерных векторов FP32 (блок размером 8 КБ может хранить только 15 128-мерных векторов FP32). Это поднимает проблему сжатия векторных данных в индексах поиска на основе ANN. Решение этой проблемы приводит к более эффективному хранению векторов на единицу индекса и, таким образом, экономит дорогостоящие операции чтения с диска.

Существующие исследования [43] показывают, что одним из основных факторов, влияющих на скорость поиска ANN, является качество построенного индекса. Например, достижение целевых уровней полноты может потребовать различного количества вычислений расстояний вектор-вектор. В нашем исследовании мы фокусируемся на задачах, которые должны быть решены с высоким значением полноты (Recall ≥0,99 для 100 ближайших соседей), что означает, что мы можем использовать определенные типы алгоритмов [44]. В худшем случае необходимо будет вычислить N расстояний (где N – количество векторов в индексированном наборе данных), что означает, что индекс практически бесполезен. Быстрый рост использования искусственного интеллекта и увеличение объемов обрабатываемых данных определяют высокую востребованность РСУБД. Также наблюдается нехватка алгоритмов для приближенного поиска ближайших соседей в РСУБД, которые не полностью полагаются на качество построенного индекса.

Одной из задач настоящей работы является разработка быстрого алгоритма приближенного поиска ближайшего соседа, обеспечивающего высокую полноту (не менее Recall@100 = 0,99). Высокие значения полноты необходимы для

сравнительного анализа приближенного поиска ближайшего соседа, поскольку они показывают, насколько эффективно алгоритм извлекает истинные ближайшие совпадения из набора данных. Низкая полнота может негативно повлиять на пользовательский опыт и безопасность в таких приложениях, как рекомендательные системы или системы выявления мошеннических действий (и других аномалий), поэтому обеспечение качества результатов становится критически важным. Предполагается, что помимо качества построенного индекса и организации структуры данных, результат обработки запроса может быть улучшен на этапе поиска по построенному индексу путем оценки уровня сложности поискового запроса. Для поиска на основе кластеризации мы вводим долю продуктивных кластеров (т. е. кластеров, которые дают нам хотя бы одного ближайшего соседа) среди всех отсканированных кластеров в качестве нового критерия для оценки уровня сложности на первом этапе поиска.

1.2 Методы приближенного поиска ближайших соседей, основанные на графах

Графовые методы используют внутреннюю структуру данных, представляя их в виде графа, где узлы соответствуют точкам данных, а ребра символизируют близость или сходство. Такое представление обеспечивает быструю навигацию и поиск в наборе данных, снижая вычислительную сложность и улучшая масштабируемость по сравнению с традиционными алгоритмами поиска ближайшего соседа.

Метод Navigating Spreading-out Graph (NSG) [45] является ведущим методом индексации на основе графов, близко приближаясь к методу Monotonic Relative Neighborhood Graph (MRNG), обеспечивая близкую к логарифмической сложность поиска при ограниченном времени построения. NSG, наряду с другими методами на основе графов, такими как FANNG [46], NSW [22] и HNSW [39], использует поиск по первому наилучшему совпадению для обработки запросов. Среди других методов на основе графов выделяют [37, 47, 48-52]. Эти методы

ориентированы в первую очередь на построение и оптимизацию самого индекса графа.

DiskANN [51] – система, которая объединяет инкрементальный алгоритм графа ANN в оперативной памяти с фреймворком для хранения графа на SSD. DiskANN – передовая система ANN поиска, разработанная для эффективной работы на SSD, использующая алгоритм индексации на основе графа, известный как Vamana.

Vamana – это алгоритм создания графовых индексов, обладающих меньшим диаметром по сравнению с традиционными алгоритмами, такими как NSG (Navigating Spreading-out Graph) и HNSW (Hierarchical Navigable Small World). Это свойство позволяет DiskANN минимизировать количество последовательных чтений с диска, необходимых во время операций поиска. Графы, созданные Vamana, также эффективны для поиска в памяти. Их производительность сравнима или превосходит современные алгоритмы поиска в памяти, такие как HNSW и NSG. Для очень больших наборов данных DiskANN может использовать меньшие индексы Vamana для перекрывающихся разделов данных и объединять их в один индекс. Такой подход обеспечивает производительность поиска, почти эквивалентную полному индексу, созданному для всего набора данных, что облегчает обработку данных, которые не могут поместиться в память все сразу. DiskANN совместим с методами векторного сжатия, такими как Product Quantization. Система хранит как индекс графа, так и векторы набора данных на диске, а сжатые векторы кэшируются в памяти, оптимизируя производительность и эффективность хранения. DiskANN, работающий на основе алгоритма Vamana, предлагает масштабируемое и эффективное решение для крупномасштабных задач ANN, эффективно преодолевая разрыв между подходами, использующими оперативную память, и подходами, основанными на диске. DiskANN может индексировать и запрашивать миллиард точек набора данных со 100-мерными векторами на рабочей станции с 64 ГБ оперативной памяти. Он достигает более 95% полноты на первом шаге с задержками менее 5 миллисекунд [51].

Алгоритм Hierarchical Navigable Small World (HNSW) [39] представляет собой инкрементальный метод, создающий иерархическую структуру, подобную списку пропусков. Каждый уровень этой иерархии представляет собой граф «навигируемого малого мира» (NSW) [3, 22]. В графе NSW узлы в основном соединены со своими непосредственными соседями, но общая структура спроектирована так, чтобы быть навигируемой, то есть любой узел можно достичь за относительно небольшое количество переходов. HNSW создает несколько слоев графов NSW, каждый из которых служит надмножеством предыдущего. Количество вершин в каждом слое геометрически увеличивается сверху вниз, при этом нижний слой содержит все входные точки.

Граф ближайших соседей, основанный на иерархической кластеризации (HCNNG) [53], включает на первом шаге случайный выбор двух векторов (точек) p и p' и разбиение входных данных на части по близости к этим точкам. Кластер считается сформированным, когда количество точек в разбиении становится меньше заданного порогового значения. Внутри каждого листового кластера строится локальный приближенный граф ближайших соседей как минимальное остовное дерево с ограниченной степенью (MST), где каждая точка имеет максимальную степень K. Затем избыточные ребра отсекаются для уточнения графа.

Алгоритм PyNNDescent [54] сочетает кластерный подход с итеративным уточнением для построения эффективного графа ближайших соседей. Изначально он строит граф, используя кластеризацию на основе случайно выбранных гиперплоскостей. В каждом листовом кластере локальный граф индекса соединяет каждую точку с ее точными K ближайшими соседями.

Помимо кластеризации, PyNNDescent использует итеративный этап постобработки, известный как метод спуска по ближайшему соседу. Этот процесс начинается с деориентирования графа – добавления обратного ребра к каждому направленному ребру. Затем каждая точка вычисляет свое двухсекционное соседство Q и сохраняет K ближайших кандидатов из Q. Алгоритм сходится, когда на каждой итерации изменяется лишь небольшая доля ребер. Наконец,

применяется алгоритм обрезки для удаления длинных ребер из всех треугольников, что улучшает граф.

За последнее десятилетие разработка графовых алгоритмов и фреймворков резко возросла. Системы с общей памятью, обрабатывающие наборы данных в оперативной памяти в пределах одного вычислительного узла, включают в себя такие известные примеры, как Galois [55], Ligra [56], Polymer [57], GraphGrind [58], GraphIt [59] и Graptor [60]. Эти системы разработаны для использования многоядерных архитектур для повышения производительности. Распределенные системы, такие как Pregel [61], GraphLab [62] и PowerGraph [63], выполняют масштабируемую обработку графов большого размера на нескольких узлах, обеспечивая масштабируемость и надежность. Внеядерные решения, такие как GraphChi [64] и X-Stream [65], обрабатывают большие графы с поддержкой дисковой памяти, преодолевая ограничения памяти. Графовые фреймворки на базе графических процессоров, такие как CuSha [66], Gunrock [67], GraphReduce [68] и Graphie [69], используют параллельную вычислительную мощность графических процессоров для ускорения вычислений на графах. Эти системы обычно придерживаются модели, ориентированной на вершины [61], или ее вариантов, таких как реброориентированные модели [65] в рамках модели BulkSynchronousParallel (BSP) [70]. В отличие от систем, которые часто используют синхронную обработку, Speed-ANN использует отложенную синхронизацию, вдохновленную моделью запаздывающей синхронизации [71]. Такой подход позволяет рабочим потокам работать асинхронно до синхронизации, поддерживая высокий параллелизм и минимизируя вычислительные расстояния.

Универсальные схемы поиска предполагают одновременную параллельную работу набора разных алгоритмов поиска, таких как поиск в ширину (BFS) [56], поиск в глубину (DFS) [72] и поиск по лучу [73]. Оптимизация и ускорение процессов поиска основаны на следующих ключевых идеях [74]: признание того, что узкое место сходимости в поиске ANN возникает из-за необходимости идентификации множества целей, которые могут присутствовать или

отсутствовать в графе – проблема, которая принципиально отличается от многих традиционных задач поиска по графам; реализация методов оптимизации, специально разработанных для сокращения количества вычислений расстояния и накладных расходов на синхронизацию при параллельном расширении соседей. Эти оптимизации включают поэтапный поиск, который сокращает избыточные вычисления, и синхронизацию с учетом избыточного расширения, которая адаптивно корректирует частоту синхронизации для баланса между точностью и производительностью.

В следующих разделах рассматриваются несколько методов, основанных на квантовании. Среди них наиболее распространенными и популярными являются Product Quantization (PQ) и алгоритмы обратного (инвертированного) файлового индекса (IVF). Следующие разделы описывают эти методы, а также принципы векторного индексирования и алгоритмы сжатия данных, с помощью которых решается задача поиска соседей.

1.3 Методы приближенного поиска ближайших соседей, основанные на квантовании

Методы, основанные на квантовании, представляют собой один из подходов масштабируемого поиска максимальных внутренних произведений в базах данных большого объема. Традиционные подходы к квантованию основаны на минимизации ошибок при восстановлении объектов базы данных.

Поиск полным перебором требует чрезмерного количества как вычислительного времени, так и памяти. В этом разделе мы исследуем основные существующие подходы к поиску приближенных ближайших соседей (ANN), которые направлены на нахождение ближайшего вектора к заданному запросу при ограниченных ресурсах.

Векторное квантование (VQ) [30, 75] – это метод сжатия данных, используемый в различных областях, таких как сжатие изображений и видео [76], обработка сигналов и аудио [77, 78, 79], распознавание образов и анализ данных.

VQ также применяется в машинном обучении, особенно при классификации и кластеризации векторов данных для работы с большими наборами данных и извлечения признаков. Основная идея векторного квантования заключается в том, чтобы представлять многомерные данные с помощью конечного набора векторов, известных как кодовые векторы. По существу, VQ разбивает пространство признаков на группы (кластеры), при этом каждая группа представляется соответствующим кодовым вектором. Такой подход снижает объем хранимой информации.

Процесс векторного квантования (VQ) состоит из следующих этапов: обучение (кластеризация, генерация кодовой книги, присвоение векторов, квантование), кодирование и декодирование. На этапе обучения сначала применяется алгоритм кластеризации для нахождения центров кластеров (центроидов), после чего осуществляется генерация кодовой книги и квантование. Обычно этапы 0-3 можно считать частью процесса обучения. На этапе кодирования каждый оригинальный вектор данных сопоставляется с ближайшим (соответствующим) кодовым вектором. На этапе декодирования каждый оригинальный вектор восстанавливается из кодовых векторов.

Этап 0: Кластеризация. На этом этапе алгоритмы, такие как метод *k*-средних, используются для разделения исходного набора векторов на группы (кластеры) на основе мер сходства. Определяется центр каждого кластера, известный как центроид, и каждый центроид служит кодовым вектором для этого кластера.

Этап 1: Генерация кодовой книги. Кодовая книга состоит из набора определенных кодовых векторов и генерируется на основе исходных данных (исходного набора многомерных векторов) и желаемого уровня сжатия. Кодовая книга служит сжатым представлением исходных данных.

Этап 2: Назначение векторов. Каждый исходный вектор сопоставляется с кодовым вектором из кодовой книги.

Этап 3: Квантование. На этом этапе каждый исходный вектор данных заменяется индексом ближайшего кодового вектора в кодовой книге. Ошибка

квантования – это разница между исходным вектором и соответствующим ему вектором из кодовой книги. Минимизируя эту ошибку, процесс сжатия обеспечивает более точное представление данных.

Этап 4: Кодирование. В процессе кодирования каждый исходный вектор данных заменяется индексом соответствующего ему кодового вектора, что значительно сокращает объем хранимых данных.

Этап 5: Декодирование. На этом этапе индексы используются для извлечения кодовых векторов из кодовой книги. Затем исходные векторы восстанавливаются из этих кодовых векторов.

Целью векторного квантования является минимизация искажения между исходными входными векторами и векторами кодовой книги, достигая компактного представления данных и сохраняя при этом как можно больше информации.

В [30] авторы дали следующее определение цели векторного квантования: цель VQ состоит в уменьшении кардинальности представления пространства, в частности, когда входные данные представлены векторами действительных чисел. Они также дали определение квантизатору: «…квантизатор – это функция q, отображающая D-мерный вектор $x \in R^D$ в вектор $q(x) \in C = \{c_i, i \in I\}$, где индекс I считается конечным: $I = 0 \ldots k\text{-}1$. Значения c_i называются центроидами. Множество значений C представляет собой кодовую книгу размера k». Набор векторов, сопоставленных с заданным индексом i, называется ячейкой Вороного (диаграммой Вороного) [30] и определяется как

$$V_i \triangleq \{x \in R^D : q(x) = \mathrm{c}_i\}.$$

Другими словами, все векторы, сопоставленные с заданным индексом i, реконструируются с использованием одного и того же вектора из кодовой книги (центроида). Качество квантования можно оценить по среднеквадратичной ошибке между исходным вектором x и соответствующим ему вектором $q(x)$.

Авторы статьи [30] утверждают, что для того, чтобы квантизатор был оптимальным, он должен удовлетворять двум условиям оптимальности Ллойда:

1. Вектор x должен быть квантован по ближайшему центроиду кодовой книги, выраженному в евклидовом расстоянии. Таким образом ячейки разграничиваются гиперплоскостями.

2. Значение реконструкции должно быть математическим ожиданием векторов, лежащих в ячейке Вороного.

Квантизатор Ллойда, который использует алгоритм кластеризации k-средних [80], а также его модификации, [81], находит близкую к оптимальной кодовую книгу путем итеративного назначения векторов обучающего набора центроидами и уточнения этих центроидов.

Скалярное квантование (Scalar Quantization, SQ), также известное как целочисленное квантование, – это метод сжатия данных, преобразующий значения с плавающей запятой в целые числа. Основная идея SQ заключается в преобразовании каждого действительного значения z вектора в целочисленное значение [82] путем усечения и масштабирования каждого измерения z [83]. SQ использует n бит для представления каждого измерения вектора z в виде целого числа в диапазоне $[0, 2n-1]$ (как правило, n=8, n=6, или n=4) [84]. Каждое измерение преобразуется следующим образом:

$$z \rightarrow \left[\frac{(2^{b-1})min(max(z,a),b)}{b-a}\right],$$

где $[\cdot]$ обозначает округление до ближайшего целого числа. Обычно значения a и b выбираются на основе процентилей распределения. Например, при кодировании с n=4 битами каждое измерение z представляется как целое число в диапазоне [0,15].

Процесс SQ состоит из следующих этапов [85]:

Шаг 1: Построение индекса: Этот этап включает преобразование векторов с действительными значениями в целочисленные. Для каждого измерения вектора SQ использует минимальное значение a и максимальное значение b этого измерения и равномерно делит это измерение на уровни (ячейки) по всему диапазону:

Шаг 1.1: Определить минимальное значение a и максимальное значение *b* для каждого измерения вектора во всем наборе данных;

Шаг 1.2: Установить начальное значение и размер шага для каждого измерения. Начальное значение задается как минимальное, а размер шага определяется количеством дискретных ячеек в используемом целочисленном типе: диапазон значений (обычно между 1-м и 99-м процентилями) делится на конечное число уровней (ячеек), каждой ячейке присваивается идентификатор. Например, при использовании 8-битных беззнаковых целых чисел SQ создает всего 256 уровней (ячеек);

Шаг 2: Квантование: Квантование выполняется путем вычитания начальных значений для каждого измерения и деления полученного значения на размер шага. Каждое измерение *z* вектора сопоставляется с ближайшим репрезентативным значением в пределах данного набора уровней квантования путем округления действительного числа до ближайшего целого. В квантованном векторе номер идентификатора заменяет исходное значение;

Шаг 3: Кодирование: После квантования каждый вектор представлен набором идентификаторов, соответствующих уровням, к которым относятся его измерения. Эти квантованные векторы требуют значительно меньше памяти для хранения по сравнению с исходным вектором, что снижает требования к объему ОЗУ и дисковой памяти.

Основной недостаток метода SQ заключается в том, что он не учитывает распределение значений внутри каждого измерения. Альтернативный подход, метод Product Quantization (квантование произведений, PQ), также предназначенный для сжатия данных, может использоваться независимо от распределения векторных данных.

PQ был предложен в [30] и развит в работах [24, 30, 32-35]. Алгоритм значительно сокращает объем хранимой информации, одновременно ускоряя процесс поиска ближайшего соседа.

Product Quantization – это метод снижения размерности и ускорения поиска в больших наборах данных с пространствами высокой размерности. Этот подход

основан на сжатии при помощи кластеризации. Первоначально пространство данных разлагается в декартово произведение нескольких подпространств низкой размерности, а затем данные квантуются в каждом подпространстве отдельно [15, 30]. В отличие от VQ, PQ использует несколько кодовых векторов для каждой группы подвекторов, что повышает эффективность квантования и сокращает объем хранимой информации.

PQ состоит из следующих шагов [30]:

Шаг 1: Разложение пространства на подпространства.

Дан набор данных из N векторов: x= $\{x_1, x_2, \ldots, x_N\}$, где x_i – вектор данных размерности D, $x_i \in R^D$: xi= {xi1,xi2,…, xiD} [86]. Исходный вектор данных x_i размерности D делится на m подвекторов u_j, j=1..m. Размерность каждого подвектора u_j – D/m.

$$x_i = [x_i^1 \ldots x_i^m] = \left[\{x_{i1}, \ldots, x_{iD}\} \ldots \{x_{iD-iD+1}, \ldots, x_{iD}\}\right],$$

где $x_i^1, x_i^2, \ldots, x_i^m \in R^{\frac{D}{m}}$, $u^1 = \{x_i^1\}, \ldots, u^m = \{x_i^m\}$.

Каждый элемент исходного вектора может быть числом с плавающей точкой, а D должно делиться на m.

Шаг 2: Кластеризация подпространств. Для каждого подпространства применяется алгоритм кластеризации, такой как k-средних, для получения набора кодовых векторов и создания кодовой книги C_j для каждого подпространства.

Шаг 3: Квантование подвекторов. Каждый подвектор u_j квантуется отдельно с использованием m отдельных квантизаторов. Каждый подвектор u_j заменяется индексом ближайшего центроида из соответствующей кодовой книги C_j. Здесь q_j представляет субквантизатор, к которому относится подвектор j, j=1,…,m. Каждый j-й подвектор имеет свой собственный набор индексов I_j, кодовую книгу C_j и соответствующие необходимые для восстановления вектора значения c_{ji}.

Шаг 4: Кодирование. После квантования подвекторов каждый исходный вектор данных представляется в виде набора индексов из кодовых книг C_j j-х подвекторов. Таким образом, оригинальный вектор значений с плавающей точкой размерности D представлен набором целых чисел (индексов) размерности m:

Значения квантизатора, необходимые для восстановления вектора (значения воспроизведения), являются элементами множества $I=I_1\times\ I_2\times\ldots\times I_m$. Кодовая книга – это декартово произведение $C=C_1\times\ C_2\times\ldots\times\ C_m$.

Центроид этого множества представляет собой конкатенацию центроидов m субквантизаторов. Все субквантизаторы имеют одинаковое конечное число k^* значений воспроизведения. Общее число центроидов определяется формулой $k=(k^*)m$. Исследования [30] показывают, что использование $k^*=256$ и $m = 8$ часто является подходящим выбором.

Шаг 5: Хранение. Закодированные данные хранятся в виде индексов кодовых векторов, что значительно сокращает объем памяти, необходимый для хранения исходных векторов.

Шаг 6: Поиск и декодирование. На этом этапе индексы используются для извлечения кодовых векторов из кодовых книг каждого подвектора, что позволяет быстро вычислять приближенные значения.

Требования к памяти алгоритма PQ – $m\log 2k^*$ бит [87]. Параметры m и k^* таким образом определяют баланс между точностью восстановления и затратами по памяти.

В векторных наборах данных часто требуется приблизительное сопоставление запросов [41]: задача состоит в том, чтобы найти векторы данных, наиболее близкие к вектору запроса, используя алгоритмы приближенного поиска ближайшего соседа (ANN) [42]. Поиск ближайшего соседа зависит от расстояния между вектором запроса и векторами набора данных. В [73] представлены два подхода к сравнению векторов на основе их индексов квантизации: симметричное вычисление расстояний (SDC) и асимметричное вычисление расстояний (ADC). Авторы рекомендуют использовать ADC для поиска ближайшего соседа, поскольку он обеспечивает меньшие искажения при измерении расстояния. Поиск с использованием ADC быстрее по сравнению с прямым линейным (точным) поиском, но все же становится медленнее для очень больших наборов данных.

Авторы [87] отметили три положительных свойства PQ: PQ сжимает исходный вектор в короткий код; вычисление приближенного расстояния с

использованием ADC между исходным вектором и сжатым кодом PQ является эффективным; структура данных и алгоритмы кодирования просты, что позволяет осуществлять гибридизацию с другими структурами индексации.

1.4 Метод инвертированного индекса

Для ускорения поиска по большому набору векторов часто используется инвертированный индекс (inverted index) - структура данных, которая для каждого характерного вектора хранит список идентификаторов объектов, отнесенных к нему. В рассматриваемом подходе сначала выполняется векторная квантизация: по исходным данным строится кодовая книга - конечный набор характерных векторов (кодовых слов), получаемых методом кластеризации (в качестве характерных векторов выступают найденные центры или центроиды кластеров). Затем каждый вектор базы сопоставляется ближайшему характерному вектору и его идентификатор добавляется в соответствующий список инвертированного индекса. Тем самым пространство признаков разбивается на области, аналогичные ячейкам диаграммы Вороного (рисунок 1.1) [6], а поиск ограничивается просмотром объектов из нескольких наиболее близких областей, что существенно снижает вычислительные затраты.

Целью инвертированного файла, основанного на стандартном квантовании, является эффективная генерация списка векторов набора данных, близких к любому вектору запроса. При выполнении запроса определяется либо ближайшее кодовое слово, либо набор из нескольких ближайших кодовых слов. Затем соответствующие этим кодовым словам списки объединяются для получения ответа на запрос (рисунок 1.1).

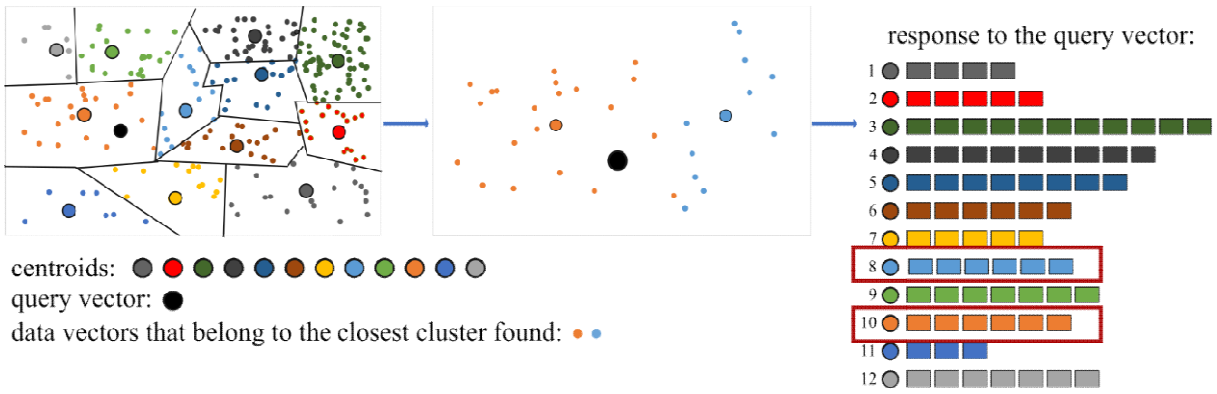


Рисунок 1.1 – Создание кодовой книги. Большая черная точка – вектор запроса, большие отмеченные цветом точки – построенные центроиды, маленькие отмеченные цветом точки – векторы данных

Однако в этом подходе, при работе с большим набором данных (от сотен миллионов до миллиардов векторов данных) существует высокая вероятность формирования чрезмерно большого (неэффективного) списка векторов набора данных, близких к вектору запроса. Это может быть следствием неточного начального разбиения исходного набора векторов по группам (кластерам), то есть кластеризация существенно влияет на точность поиска [25]. Если число центроидов (кодовых векторов) увеличить ради более точного распределения исходного набора данных, то возрастает как время выполнения запросов, так и время построения индекса [88] (алгоритм PQ решает эту проблему, однако, качество поиска может заметно снижаться). При необходимости вычисляя истинные расстояния между каждым вектором и запросом можно получить полные векторы для всех идентификаторов, хранящихся в инвертированном списке для заданного центроида. Этот процесс, известный как повторное ранжирование, может повысить производительность запросов [89].

Извлечение данных существенно ускоряется по сравнению с полным перебором, что решает одну из проблем при работе с большими наборами векторов, т.е. при запросе к базе данных с построенным индексом инвертированного файла не требуется оценивать расстояния между вектором запроса и каждым вектором в исходном наборе данных.

Концепция инвертированного файлового индекса (Inverted File Index, IVF) заключается в следующем: исходный набор векторов (векторное пространство) группируется в кластеры с использованием алгоритма кластеризации, например, метода k-средних, а затем в каждой группе найденный центр кластера (центроид) назначается кодовым вектором (рисунок 1.2). Каждый центроид (кодовый вектор) выражает информацию обо всех векторах, принадлежащих данному кластеру, формируя таким образом кодовую книгу (рисунок 1.3).

При поиске ближайших к вектору запроса соседей вместо поиска по всему набору векторов определяется ближайший центроид к вектору запроса (рисунок 1.4a)). Поиск выполняется путем нахождения минимального расстояния между вектором запроса и центроидами. Затем среди векторов в кластере, соответствующем заданному центроиду (рисунок 1.4b)), находится ближайший сосед или соседи, ограничивая таким образом область поиска. Стандартной практикой считается использование $k = \sqrt{N}$ [6, 90], где N – это число векторов в наборе данных.

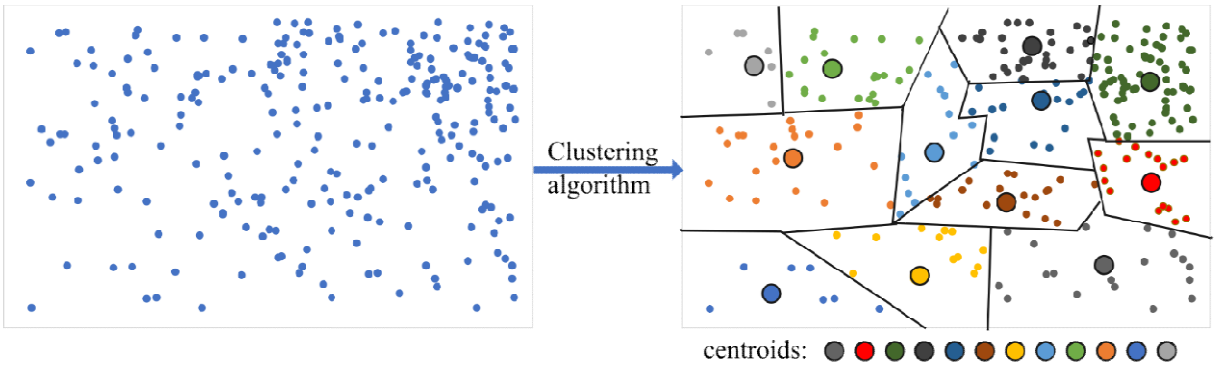


Рисунок 1.2 – Пространственное разложение векторов после применения алгоритма кластеризации. Большие отмеченные цветом точки – построенные центроиды, маленькие отмеченные цветом точки – векторы данных

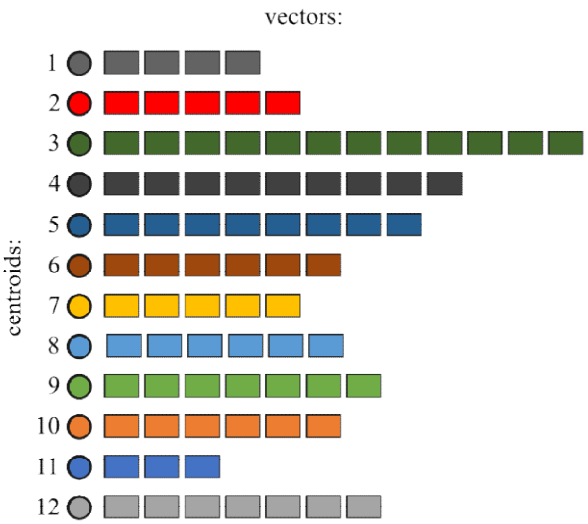


Рисунок 1.3 – Сопоставление векторов с центроидами (векторами кодовой книги). Большие отмеченные цветом точки – построенные центроиды, прямоугольники – соответствующие центроидам векторы данных

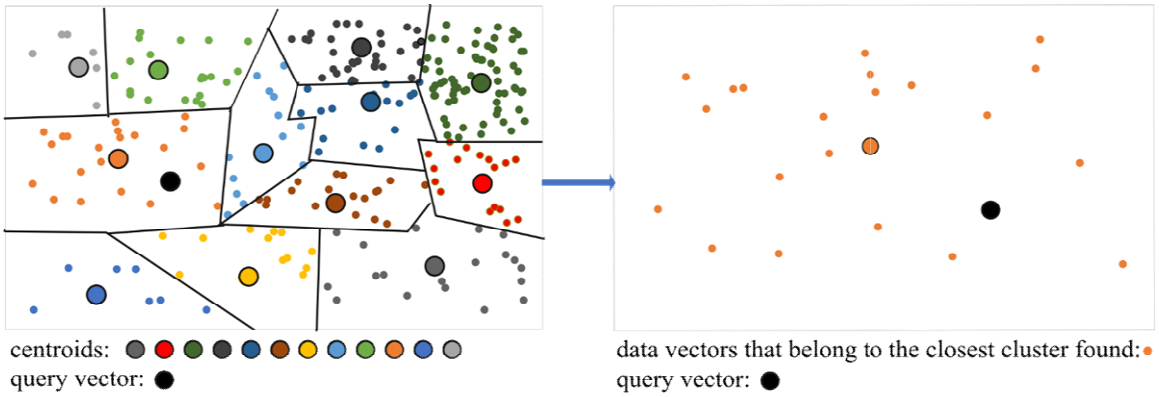


Рисунок 1.4 – Поиск ближайших соседей среди векторов данных, которые принадлежат ближайшему найденному кластеру. Большая черная точка – вектор запроса, большие отмеченные цветом точки – построенные центроиды, маленькие отмеченные цветом точки – векторы данных

Стоит отметить, что существует высокая вероятность того, что вектор запроса будет находиться вблизи «края» кластера, а его ближайший сосед будет находиться в смежном кластере; эта проблема известна как проблема края (the edge problem) [84].

Проще говоря, если в KNN точка запроса сопоставляется со всеми N точками данных (из размерности D, что дает временную сложность $O(ND)$), то в ANN с использованием IVF точка запроса сопоставляется только с подмножеством из N точек в ближайших кластерах, что дает временную сложность: $O(KD + ND/K)$ [6], где K – число кластеров (групп), которое может варьироваться. На первом шаге (сопоставление со всеми центроидами) временная сложность равна $O(KD)$, а на втором шаге (поиск ближайшего соседа в ближайшем кластере) – $O(ND/K)$.

Таким образом, для достижения компромисса между высоким качеством поиска и производительностью нужна не только качественная кластеризация, но и тщательная настройка параметра K.

Среди рассмотренных алгоритмов Product Quantization (PQ) обладает высокой эффективностью хранения данных и высокой скоростью поиска. Однако применение PQ может снизить точность поиска [6].

Основанные на графах алгоритмы поиска получили значительное внимание научного сообщества, демонстрируя многообещающие характеристики эффективности и производительности. Эти методы обеспечивают компромисс между высокой вычислительной эффективностью и требуемой точностью; вместе с тем их применение зачастую требует большого количества памяти.

Индекс обратного файла (IVF) позволяет построить такую структуру данных, которая обеспечивала бы точный результат и высокую скорость поиска, что важно для задач с повышенными требованиями к показателю полноты. Этот метод обладает широкими возможностями для улучшения как кластеризации при построении индекса, так и процедуры поиска. Таким образом, Инвертированный Индекс может быть использован для повышения качества поиска. IVF реализован во многих векторных базах данных, в том числе в приложениях с открытым исходным кодом.

1.5 Алгоритмы автоматической группировки данных

Для построения индекса обратного файла необходимо создание кодовых книг – групп векторов данных, где каждая группа характеризуется так называемым кодовым вектором, представляющим собой некое приближение или усреднение характеристик векторов данных внутри группы. Так возникает задача поиска похожих объектов и объединения их в группы – задача кластеризации. Задача автоматической группировки данных, также называемая задачей кластеризации, является распространенной задачей обучения без учителя. У этой задачи нет строгого математического определения, но неформально ее можно описать следующим образом: требуется разбить множество некоторых объектов на некоторые подмножества таким образом, чтобы внутри одного подмножества объекты были максимально похожи друг на друга и как можно больше отличались от объектов других подмножеств по некоторой мере [91].

На сегодняшний день существует огромное количество самых разных алгоритмов кластеризации [92 – 97], примерами которых являются алгоритмы *k*-средних [98], различные иерархический алгоритмы [99], EM-алгоритм [100], DBSCAN [101] и другие [91]. Также существуют эволюционные подходы, последовательно улучшающие существующее решение на каждой итерации. Примерами таких подходов служат генетические алгоритмы [98, 102, 103, 104], VNS-алгоритм [105] и (1+λ) алгоритмы [106].

На сегодняшний день существуют десятки методов кластерного анализа, позволяющих выявлять самые разные по совокупности параметров группы. Для построения кодовых книг IVF чаще всего используется алгоритм *k*-средних (англ. *k*-means), который решает задачу *k*-средних. Это алгоритм локальной оптимизации, который зависит от выбора начальных решений (начального положения центроидов). Алгоритм *k*-средних обладает воспроизводимостью: при выборе одинаковых начальных положений центроидов алгоритм будет давать один результат, что критически важно для некоторого класса задач, в том числе производственных [107, 108].

Задача *k*-средних является классической задачей размещения, в рамках которой требуется найти k таких центров кластеров X_1, …, X_k в d-мерном пространстве, чтобы минимизировать сумму квадратов расстояний от них до заданных точек A_i:

$$F(X_i, \dots, X_k) = \arg\min_{X \in \mathbb{R}^d} \sum_{i-1}^{N} \min_{X \in \{X_1, \dots, X_k\}} \|A_i - X\|^2 \quad (1.1)$$

Классическим методом решения задачи *k*-средних является одноименный алгоритм, также известный по фамилии автора [109] как процедура Ллойда и иногда называемый ALA-процедурой (англ. Alternating Location Allocation – чередующееся размещение и распределение). Алгоритм включает всего два чередующихся шага: определение каждого объекта к кластеру, характеризующемуся ближайшим из известных центров и пересчет положения центроидов.

Алгоритм ищет локальный минимум (формула 1.1), итеративно улучшая известное решение. Алгоритм имеет некоторые ограничения, в частности нужно заранее задать число групп k, на которые разбиваются объекты. Результат сильно зависит от начального решения, обычно выбираемого случайно или при помощи процедуры k-means++ [91]. Для поиска глобального минимума используется мультистарт алгоритма *k*-средних или различные методы глобальной оптимизации, использующие алгоритм *k*-средних (алгоритм 1.1) [106, 108, 109].

Алгоритм 1.1 - Алгоритм *k*-средних

Дано: векторы данных $A_1 \dots A_N$, k начальных центров кластеров $X_1 \dots X_k$.

Шаг 1. Составить кластер C_i векторов данных для каждого центра X_i так, чтобы для каждого вектора данных его центр был ближайшим;

Шаг 2. Рассчитать новое значение центра X_i для каждого кластера.

Шаг 3. Если Шаги 1-2 не привели к изменениям, то ОСТАНОВ, иначе переход к Шагу 1.

Идея алгоритма *k*-средних была предложена в 1956 году Штейнгаузом [110], а сам алгоритм был разработан Ллойдом год спустя. Так как алгоритм *k*-средних является алгоритмом локального поиска, для поиска глобального минимума целевой функции необходимо использовать глобальный поиск. Самым простым примером глобального поиска является многократный запуск со случайно сгенерированными начальными решениями.

В рамках развития алгоритмов кластеризации было разработано множество эвристических подходов, предназначенных для глобального поиска; классическим примером здесь служат эволюционные алгоритмы (ЭА). Их ключевое отличие от других методов оптимизации заключается в том, что ЭА работают не с одним решением, а с множеством решений – «популяцией», – и поэтапно улучшают заданную целевую функцию.

Ранние эволюционные стратегии применялись в гидродинамике для оптимизации формы изогнутой трубы и решения родственных задач. Поначалу использование ЭА было весьма ограниченным, однако сегодня эволюционные алгоритмы широко задействуются при построении различных моделей, таких как p-median, многочисленные модели кластеризации и нейронные сети [98, 106].

Другим подходом являются так называемые жадные агломеративные эвристики [103], которые могут быть использованы на некоторых этапах ЭА. Жадные агломеративные эвристики представляют собой набор алгоритмов, которые стартуют с набора отдельных элементов и итеративно объединяют их в более крупные структуры, неизменно выбирая на каждом шаге то объединение, которое, согласно заранее заданному критерию, обеспечивает наиболее благоприятный результат. Данный подход реализует агломеративную стратегию, продвигаясь от малых групп объектов к большим.

Такой агломеративный подход опирается на жадный принцип: на каждой стадии алгоритм оценивает возможный выигрыш от удаления каждой из групп объектов и перераспределения векторов из удаляемой группы по оставшимся группам векторов. Под выигрышем может пониматься, например, минимизация расстояний в задачах кластеризации либо улучшение значения целевой функции в

оптимизационных задачах. Агломеративная природа подхода предполагает поэтапное укрупнение – от избыточного числа малых групп объектов к полностью сформированным кластерам.

Жадную агломеративную процедуру можно описать следующим образом [106]:

Алгоритм 1.2 - Жадная агломеративная процедура

Дано: Набор начальных центров кластеров $S = \{X_1, \ldots, X_K\}$, $|S| = K > p$, где p – число искомых центроидов.

Шаг 1. Улучшить S с использованием процедуры Ллойда, если это возможно.

Шаг 2. $F_i \leftarrow F(S \backslash \{X_i\}), i = \overline{1, |S|}$.

Шаг 3. Выбрать подмножество $S' \subset S$ центроидов с наименьшим соответствующим значением Fi, $|S'| = \max\{1, \lceil (|S| - p) \cdot 0.2 \rceil\}$.

Шаг 4. Создать новое решение $S \leftarrow S \backslash S'$.

Шаг 5. Улучшить созданное решение при помощи процедуры Ллойда.

Шаг 6. Если $|S|<k$, то перейти к Шагу 2.

В целом, интеграция эволюционных алгоритмов и жадной агломеративной эвристики позволяет преодолеть ограничения традиционного алгоритма k-средних при построении высококачественных кодовых книг для IVF-индексов. Используя популяционную оптимизацию и итеративные стратегии объединения, эти методы усиливают возможности глобального поиска, снижая зависимость от начального расположения центроидов и повышая общую устойчивость кластеризации. Это не только повышает точность приближенного поиска ближайших соседей, но и позволяет создавать более масштабируемые информационные системы для обработки больших объемов данных, где вычислительная эффективность и надежность результатов имеют первостепенное значение.

1.6 Методы группировки мультимодальных данных

Современные системы хранения данных все чаще работают не только с «классическими» таблицами, но и с мультимодальными данными: текстами, изображениями, аудио, видео, геоданными и потоками от датчиков. Практическая потребность в этом возникает в задачах, где один объект имеет несколько представлений: например, карточка товара включает описание, фото и отзывы; медицинское исследование – снимки, заключение врача и показатели анализов; инцидент в службе поддержки – текст обращения, скриншоты и запись звонка. Одно представление называется модальностью.

Отдельное направление – поддержка смыслового поиска и сопоставления объектов разных модальностей. На практике это часто реализуется через векторные представления (эмбеддинги): текст, изображение или аудио преобразуются в числовой вектор, который затем индексируется в базе данных и позволяет выполнять поиск ближайших соседей. В результате пользователь может искать по картинке похожие товары, по описанию – близкие изображения, а по фрагменту аудио – похожие записи.

Ключевым вызовом в области анализа мультимодальных данных выступает разработка корректных методов оценки близости и вычисления расстояний между объектами. В то время как для однородных данных (в рамках одной модальности) задача сопоставления решается через классические метрики и заданные меры расстояния, взаимодействие различных типов информации требует принципиально иных подходов.

Сложность работы с многообразными представлениями данных обусловлена необходимостью эффективной агрегации и слияния (fusion) различных модальностей. На этапе совмещения признаков важно не только сохранить уникальные характеристики каждого источника, но и выявить скрытые корреляции между ними. В современной практике машинного обучения выделяют два основных архитектурных направления. К первому относятся комплексные end-to-end модели («сквозные»), которые способны обрабатывать сырые входные

сигналы – изображения, текстовые массивы, видеопотоки и аудиозаписи – извлекая признаки в едином цикле обучения. Второе направление представлено методами, ориентированными на работу с промежуточными представлениями. В этом случае данные сначала трансформируются в высокоуровневые векторы (эмбеддинги) с помощью специализированных энкодеров, и лишь затем подвергаются слиянию в общем пространстве для выполнения задач классификации, поиска или генерации.

Подавляющее большинство из существующих методов агрегации модальностей так или иначе используют методы машинного обучения с учителем, что не подходит задачам кластеризации, так как во входных данных нет «меток» кластеров.

Одним из базовых методов обработки мультимодальных данных является обучение совместного представления (joint representation learning). В рамках этого подхода данные разных модальностей сначала независимо преобразуются в векторные представления, а затем полученные векторы объединяются в общее семантическое подпространство с помощью модели слияния (например, EmbraceNet, EmbraceNet+, простая конкатенация векторов или BiT [111]). Однако у данного подхода есть и недостатки. Итоговое представление, как правило, сохраняет общую семантику между модальностями, но при этом может игнорировать специфическую информацию, присущую каждой отдельной модальности. Кроме того, после применения такого объединения обычно невозможно извлечь представление каждой модальности по отдельности.

Модель согласованного (coordinated) представления, напротив, строит отдельные представления для каждой модальности, а затем «сводит» их в единое согласованное пространство. Такое представление позволяет учитывать модально-специфические характеристики [112]. Отдельный класс решений составляют архитектуры encoder–decoder, которые переводят представление одной модальности в представление другой. В этой схеме энкодер преобразует первую модальность в скрытый вектор, а декодер по этому вектору восстанавливает представление второй модальности. Модель может включать

несколько энкодеров и декодеров, чтобы раздельно выделять несколько независимых характеристик одной модальности (например, несколько записанных музыкальных инструментов в аудио). Важно, что скрытый вектор формируется не только на основе входной модальности, как может показаться, но и с учетом выходной (целевой) модальности, поскольку ошибка «перевода» распространяется от декодера к энкодеру и тем самым учитывает обе модальности [113].

Еще один существующий подход к работе с мультимодальными данными называется «глубокие мультимодальные машины Больцмана» (deep multimodal Boltzmann machines, DBM). DBM представляют собой вероятностные графические модели, которые можно описать как две ограниченные машины Больцмана, имеющие общий слой представления. Связь между модальностями задается через функцию потерь, основанную на корреляции. Кроме того, такие сети являются двунаправленными, что позволяет выполнять «перевод» данных между модальностями. Существенным недостатком этих моделей выступает высокая потребность в вычислительных ресурсах [114].

Архитектуры автоэнкодеров позволяют восстанавливать любую модальность даже в случае, когда одна из модальностей отсутствует. Обучение автоэнкодеров обычно сводится к минимизации ошибки реконструкции модальностей. В работе [115] предложено регуляризовать веса функции потерь для разных модальностей, чтобы уменьшить избыточность в получаемом представлении. Отмечается также, что модель способна выделять модально-специфические признаки.

Также важным элементом теории обработки мультимодальных данных являются работы Г.С. Лбова и соавторов [116], которые также решали задачи автоматической группировки разнородных данных, однако те подходы опираются на построение логических правил и плохо применимы к современным наборам данных большого объема и большой размерности.

Несмотря на наличие методов, работающих с мультимодальными данными, ни один из рассмотренных подходов не является напрямую подходящим для задач

приближенного поиска ближайших соседей: возникают трудности с определением расстояний в пространстве признаков нескольких модальностей, а также при создании кодовых книг: в общем случае кластеризация не требует размеченных данных, поэтому обучать такие модели на стандартных наборах данных с учителем оказывается затруднительно.

По итогам первой главы сформулирована и уточнена задача поиска ближайших соседей для векторных представлений (эмбеддингов) при наличии фиксированного множества объектов и множества запросов. Показано, что точный поиск методом полного перебора в условиях высокой размерности и больших объемов данных часто оказывается вычислительно неприемлемым, вследствие чего на практике востребованы приближенные методы поиска ближайших соседей (ANN), допускающие контролируемую потерю точности при существенном выигрыше по времени. Обосновано, что ключевым способом достижения ускорения является построение индекса, то есть предварительная обработка набора векторов, позволяющая уменьшить число сравниваемых кандидатов на этапе обработки запросов по сравнению с линейным просмотром.

В главе рассмотрены основные направления индексации и ANN-поиска, включая методы на основе деревьев, локально-чувствительного хеширования и подходы, основанные на квантизации векторов, а также подчеркнута их практическая значимость для задач поиска по похожести в системах обработки разнородных, в том числе мультимодальных, данных, где объекты представлены унифицированными эмбеддингами. Детально описан подход на основе инвертированного файла, в рамках которого поиск организуется как двухэтапная процедура: сначала отбирается ограниченное число наиболее перспективных кластеров, а затем выполняется перебор кандидатов только внутри выбранных кластеров. Показано, что перспективность кластера естественным образом определяется расстоянием от запроса до характерного вектора кластера (центра/центроида) в выбранной метрике, что делает качество кластеризации критически важным для эффективности и точности поиска.

Также в главе представлена постановка задачи кластеризации и рассмотрены алгоритмы автоматической группировки данных, применимые при формировании кластерной структуры индекса. Дополнительно рассмотрены стратегии глобального поиска.

В завершение выполнен обзор современных подходов к работе с мультимодальными данными и обоснована необходимость разработки алгоритма автоматической группировки неразмеченных мультимодальных объектов, представленных в едином векторном пространстве. Тем самым первая глава задает теоретическую и методическую основу для дальнейшего исследования: выбор и построение индексных структур для ANN-поиска через кластеризацию, а также постановку и критерии качества автоматической группировки мультимодальных данных.

1.7 Выводы к главе 1

В первой главе рассмотрена задача приближенного поиска ближайших соседей (англ. Approximate Nearest Neighbour search, ANN), являющаяся важным компонентом современных систем обработки больших объемов многомерных данных. Проведен анализ основных классов алгоритмов ANN. Графовые методы, в частности Hierarchical Navigable Small World (HNSW) и Navigating Spreading-out Graph (NSG), обеспечивают высокую скорость поиска за счет эффективной навигации по структурам типа «малый мир». Методы, основанные на квантовании, такие как Product Quantization (PQ) и скалярное квантование (SQ), решают проблему хранения и обработки больших наборов векторов путем их сжатия в компактные коды, что существенно снижает требования к памяти и ускоряет вычисление расстояний. Отдельное внимание уделено методу инвертированного файлового индекса (Inverted File Index, IVF), который организует поиск как двухэтапную процедуру: выбор ограниченного числа перспективных кластеров на основе расстояния до их центроидов и последующий

перебор кандидатов внутри этих кластеров. Эффективность IVF в значительной степени определяется качеством предварительной кластеризации данных.

Среди рассмотренных алгоритмов ANN поиска IVF выделяется не только эффективностью, но и целым рядом возможностей для усовершенствования, так как скорость и качество поиска может зависеть не только от самой процедуры поиска ближайших соседей, но и от процедуры создания индекса – объединения групп векторов.

В связи с этим в первой главе также детально рассмотрена задача автоматической группировки (кластеризации) данных, являющаяся ключевой для построения кодовых книг в IVF и других методах, основанных на группировке векторов и сокращения области поиска. Проанализированы классические алгоритмы, такие как *k*-средних и его модификации, а также эволюционные и агломеративные эвристики, направленные на преодоление локальной оптимальности и поиск глобального минимума целевой функции.

Кроме того, обозначена актуальная проблема работы с мультимодальными данными, где объекты представлены разнородными модальностями (например, текстом, изображением, аудио). Отмечено, что существующие подходы к совместному обучению представлений (joint и coordinated representation learning, автоэнкодеры) часто требуют размеченных данных и не адаптированы напрямую для задач кластеризации и последующего ANN-поиска в едином пространстве признаков.

2 АЛГОРИТМ АДАПТИВНОГО ПОИСКА ПО ФАЙЛУ ОБРАТНОГО ИНДЕКСА

Метод IVF (Inverted File Index) представляет собой эффективный подход к приближенному поиску ближайших соседей в векторных базах данных, основанный на комбинации кластеризации и обратного (инвертированного) индекса. Векторная база данных - это специализированная система хранения и поиска, которая представляет данные в виде многомерных векторов (массивов чисел) и позволяет эффективно находить похожие элементы на основе их векторного представления, используя метрики расстояния или сходства. В основе IVF-метода лежит предварительная кластеризация всего набора векторов с использованием алгоритма *k*-средних, где векторное пространство разделяется на фиксированное число кластеров, каждый из которых ассоциируется с центроидом. Каждый вектор базы данных присваивается ближайшему центроиду, формируя инвертированный файл – список центроидов, каждому из которых ставится в соответствие список объектов, для которых этот центроид является ближайшим. Списки объектов в этом случае являются кластерами. Для повышения точности часто применяется дополнительная квантизация остатков векторов относительно центроидов, что позволяет компактно представлять данные и минимизировать объем памяти, обеспечивая при этом баланс между скоростью и точностью поиска.

Процесс поиска в IVF начинается с вычисления расстояний от вектора-запроса до всех центроидов, после чего выбираются наиболее близкие кластеры (обычно небольшое их количество), и детальный поиск ближайших соседей проводится только в пределах соответствующих инвертированных списков. Это значительно сокращает объем вычислений по сравнению с полным перебором, делая метод масштабируемым для больших объемов данных (до миллиарда объектов).

2.1 Исследование влияния характеристик запроса на вычислительную сложность его обработки

Настоящая глава посвящена процессу поиска приближенных ближайших соседей с использованием построенного обратного индекса. IVF-индекс является дополнительной структурой данных, списка k кластеров, где каждый кластер имеет свой центроид C_i, который является точкой (вектором) в том же пространстве, что и векторы данных и называется характерной точкой. Каждый из векторов данных присваивается кластеру, центроид которого ближайший к этому вектору данных. Таким образом, кластеры являются списками векторов данных, отнесенных к центроидам. Обработка запроса начинается с поиска центроидов, ближайших к вектору запроса q. Число таких центроидов (и соответствующих кластеров) ограничено натуральным числом $nprobe<k$. Затем все векторы данных в $nprobe$ кластерах просматриваются (сканируются) для нахождения ближайших K соседей вектора q. Основная идея предлагаемого подхода заключается в реализации итеративной процедуры поиска, которая инициируется с ограниченного набора ближайших центроидов и соответствующих им кластеров. После предварительной обработки и оценки полученных результатов, в случае их недостаточности, алгоритм расширяет область поиска, включая дополнительные кластеры для повышения полноты и точности конечного результата, при этом количество дополнительно сканируемых кластеров зависит от результатов предварительной обработки.

Стандартным подходом к оценке меры качества обработки запросов считается метрика полноты, также называемая Recall. Одной из задач, решение которой описано в данной главе, является разработка алгоритма поиска приближенных ближайших соседей, который позволял бы достичь значения Recall не менее 0,99. Полнота зависит от качества кластеризации, от числа построенных кластеров и от количества кластеров, просматриваемых в процессе обработки запроса K. В качестве примера использовался набор данных

SIFT1M [89, 117], содержащий 1 млн. объектов размерности 128 и разбитый на 4096 кластеров.

По результатам предварительных экспериментов, целью которых являлся анализ характеристик (признаков) запроса, для получения усредненного Recall=099 при поиске 100 ближайших соседей, необходимо просматривать порядка 5% кластеров, то есть примерно 200 кластеров. Это приводит к обработке примерно 50000 векторов данных в каждом запросе ANN. Если сократить число просматриваемых кластеров до 3% от общего (127 кластеров), разброс значений Recall при обработке запросов заметно увеличивается, не позволяя стабильно достигать высоких значений полноты. Запросы, для которых при обработке малого числа кластеров (127 кластеров в рамках этого эксперимента) Recall достигает 1.0 или 0.99, будем называть «простыми» (таблица 2.1). Запросы, для которых после обработки малого числа кластеров Recall не достигает этого порога, будем называть «сложными». Проще говоря, для простых запросов в наборе данных SIFT1M просмотр 127 кластеров является достаточным для достижения высоких значений Recall. В SIFT1M истинные соседи предварительно вычисляются для специального перечня векторов-запросов, так называемых «бенчмарков». В реальных наборах данных истинные соседи, как правило, неизвестны, вследствие чего фактические значения Recall не могут быть рассчитаны точно без использования полного перебора. Вместе с тем достаточно точный список ближайших соседей может быть получен приближенно за счет выполнения не полного, но расширенного поиска для ограниченного числа запросов (обучающей выборки запросов).

Идея предлагаемого метода заключается в классификации запросов, то есть в разделении запросов на «простые» и «сложные» после обработки ограниченного числа кластеров. Имея описанную выше обучающую выборку, можно обучить классификатор сложности запросов, который бы распознавал простые и сложные запросы и сигнализировал о необходимости продолжения поиска для сложных запросов.

Таблица 2.1 – Результат поиска ближайших соседей на наборе данных SIFT1M. Число просмотренных кластеров - 3% от общего числа кластеров в построенном индексе, l_2 –расстояние до выбранного ближайшим соседом вектора, l_c – расстояние до центра кластера ближайшего соседа, n_{res} – число кластеров, в которых был найден по крайней мере один ближайший сосед, Recall – доля верно определенных соседей среди 100 предсказанных

Номер запроса	$<l_2>$	$\sigma(l_2)$	$\max(l_2)$	$<l_c>$	$\sigma(l_c)$	$\max(l_c)$	n_{res}	Recall
0	78725.8	5723.66	83582	82588.8	10322.3	103515	41	0.97
1	62840.1	3453.5	66853	54277.1	10035.0	75566.6	28	0.96
2	41863.3	3374.59	45862	43334.7	8653.09	62068.9	23	0.99
3	41260.5	3038.36	44852	43877.7	4615.84	70784.3	14	1.00
4	62709.1	4470.44	67477	63630.6	19619.1	93600.6	29	0.97
5	81312.2	5371.47	87644	86521.2	12956.8	108471	42	0.88
6	61879.8	3354.74	66123	61033.3	9810.32	78267.6	43	0.94
7	36848.8	4143.73	41901	33443.5	12430.3	73763.8	16	1.00
8	60241.2	3521.91	64914	57454.6	12061.0	81352.5	30	0.94
9	58733.8	3965.1	63567	61073.5	11142.1	84609.7	32	1.00
10	79190.0	6635.88	86494	85952.2	8779.93	106106.0	44	0.93
11	44841.0	3336.09	48074	44711.4	6757.47	58723.0	26	0.97
12	53708.8	3670.09	57272	56956.6	7524.57	70652.4	32	0.98
13	57174.1	3635.02	61995	54672.5	13568.5	83061.7	30	0.95
14	69782.6	4423.02	74929	65107.2	15021.9	90432.0	36	0.94
15	55450.7	3565.02	59554	58087.5	7800.52	74834.5	30	1.00
16	51708.7	3701.75	55406	49710.7	7565.09	69743.6	38	0.99
17	57093.6	3028.29	60511	56592.3	6949.12	72847.6	36	0.94
18	62165.1	3509.82	66257	65344.1	9005.28	84700.5	44	0.98
19	46533.5	2720.02	49592	44272.4	6638.65	58191.4	32	0.97

Алгоритмы приближенного поиска ближайших соседей, основанные на IVF, подвержены «проблеме края» (англ. edge problem): существует высокая вероятность того, что вектор запроса расположен вблизи границы кластера, тогда как его ближайший сосед находится в соседнем кластере. Поскольку приближенный поиск ближайших соседей (ANN) с использованием IVF ограничивает поиск лишь подмножеством точек, принадлежащих ближайшим

кластерам, отсутствует гарантия того, что будут просканированы именно те кластеры, в которых находятся истинные ближайшие соседи. Однако, при использовании адаптивного алгоритма приближенного поиска ближайших соседей число кластеров, подлежащих сканированию, определяется по результатам предварительного поиска (по результатам сканирования минимального заданного числа кластеров). Для более сложных запросов проблема края нередко приводит к ошибочной идентификации кластеров. В рамках данного исследования принимается допущение, что чем выше доля ошибок при идентификации кластеров по расстояниям до их центроидов (то есть чем выше сложность запроса), тем больше разнообразие числа тех кластеров, которые на ранней стадии поиска обеспечивают попадание ближайших соседей. Указанный параметр (число эффективных кластеров на ранних стадиях поиска) используется в разработанном классификаторе сложности запросов и адаптивном алгоритме IVF-поиска.

Предлагаемый в настоящей главе алгоритм ANN поиска является адаптивным методом на основе IVF и, следовательно, также подвержен проблеме края. Предполагается, что эта проблема является более существенной именно для запросов, классифицируемых как сложные.

Результаты предварительной обработки запросов могут характеризоваться несколькими статистическими показателями. Например, характеристикой запроса может быть среднее расстояние от запроса до его ближайших соседей, а также дисперсия (отклонение) этих расстояний. Проведены предварительные эксперименты, направленные на построение классификатора, способного распознавать простые запросы на основе признаков, перечисленных в описании к таблице 2.1. Вычисление указанных показателей требует обработки некоторого начального числа кластеров.

Сложные запросы характеризуются необходимостью расширять поиск на большее число кластеров для достижения высоких значений Recall. Такие запросы представляют собой одну из наиболее трудных задач в приближенном поиске ближайших соседей. Основная цель при обработке таких запросов состоит

в максимизации Recall, то есть в верном определении как можно большего числа сложных запросов, при одновременном соблюдении жестких ограничений по точности (precision), чтобы избежать существенного роста числа ложноположительных срабатываний, приводящих к существенному снижению скорости поиска. Такой баланс критически важен для получения корректных и надежных результатов поиска за разумное время, особенно в прикладных сценариях, где сложные запросы часто имеют высокую значимость.

Для достижения такого баланса классификатор сложности запросов должен быть ориентирован на снижение вероятности пропуска релевантных результатов для сложных запросов. Это предполагает применение продвинутых методов, таких как динамическое расширение запроса (dynamic query expansion), адаптивное распределение ресурсов (adaptive resource allocation) или специфичные для запроса стратегии поиска (query-specific search strategies), обеспечивающих эффективную обработку подобных «трудных» запросов. Приведенное в этой главе исследование ориентировано на создание новой адаптивной стратегии поиска, при которой каждый запрос обрабатывается с учетом его сложности.

Далее приведены результаты серии предварительных экспериментов на сравнительно небольшом наборе данных (10^6 векторов данных), что позволяет выявить наиболее значимые признаки (характеристики) ANN-запроса и оценить уровень его сложности. На основе результатов данных предварительных экспериментов был разработан описанный в разделе 2.3 адаптивный алгоритм IVF-поиска, включающий классификатор сложности запросов и алгоритм обучения классификатора сложности запросов.

Для анализа зависимости сложности запроса и ожидаемого значения Recall от различных факторов была выполнена серия предварительных экспериментов, различавшихся числом обрабатываемых кластеров (*nprobe*) при обработке одних и тех же запросов. Когда алгоритм поиска получает на вход вектор запроса q, выполняется поиск K ближайших соседей в *nprobe* кластерах, ближайших к q. По итогам эксперимента были агрегированы данные, включающие характеристики

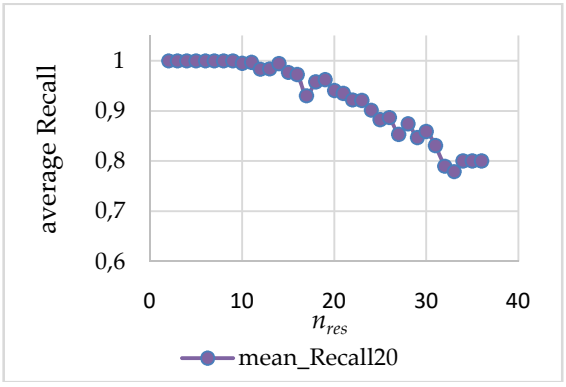
запроса, такие как среднее расстояние до вектора данных выбранного соседа, стандартное отклонение этих расстояний, максимальное значение этих расстояний, среднее расстояние до центроида кластера, которому принадлежит вектор данных выбранного соседа, стандартное отклонение этих расстояний, максимальное значение этих расстояний, число кластеров, в которых были найдены ближайшие соседи, а также значения Recall при поиске одного, пяти, десяти, двадцати, пятидесяти и ста ближайших соседей. Каждый запрос выполнялся при различных значениях *nprobe*.

После каждого эксперимента вычислялось достигнутое значение Recall. Для каждого запроса фиксировалось минимальное значение *nprobe*, необходимое для достижения требуемого значения Recall (Recall@100 = 0.99). Полученные результаты позволяют проанализировать, каким образом различные характеристики запроса влияют на число кластеров, которые необходимо обработать для достижения высоких значений Recall.

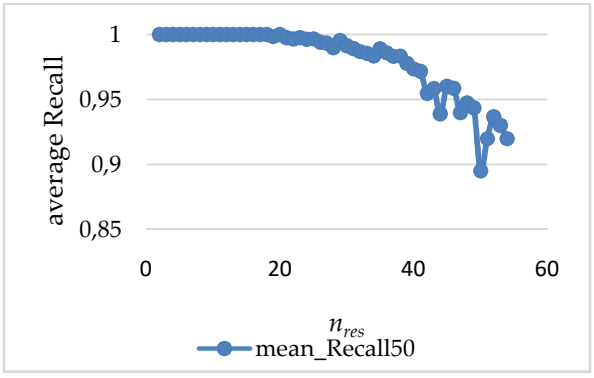
Накопленных данных достаточно для исследования зависимости сложности запроса от его характеристик, вычисляемых на ранних этапах обработки запроса (после поиска в сравнительно небольшом числе кластеров). Данный анализ позволил сформировать модель, способную классифицировать запросы на первых этапах поиска соседей и принимать решение о целесообразности продолжения поиска.

Одним из важных признаков в таких данных является число кластеров, содержащих хотя бы одного найденного ближайшего соседа. Этот признак обозначается как n_{res} (см. таблицу 2.1). Была исследована зависимость n_{res} от *nprobe*. Это дополнительное исследование на том же наборе данных SIFT1M и с IVF-индексом с теми же параметрами (4096 кластеров) потребовало еще одного предварительного эксперимента. Анализ опирается на результаты ANN-поиска при фиксированных значениях *nprobe* в диапазоне от 1% до 7% от общего числа кластеров ($40 < nprobe < 287$), где 1% обеспечивает минимально значимый результат, а 7% соответствует качеству расширенного поиска. На рисунке 2.1 представлены средние значения Recall (ось *y*) для подмножеств запросов с

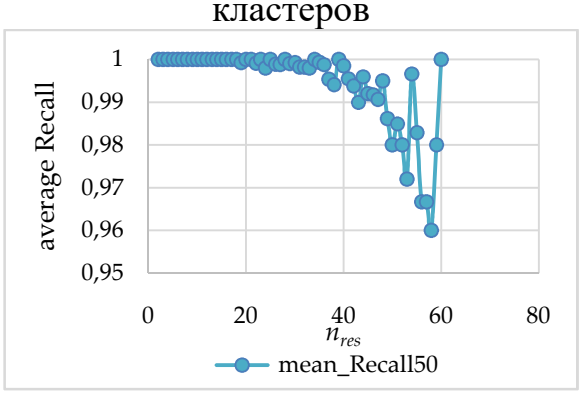
одинаковыми значениями n_{res} (ось x), измеренные при фиксированных значениях *nprobe*, равных приблизительно 1%, 3%, 5% и 7% от общего числа кластеров.



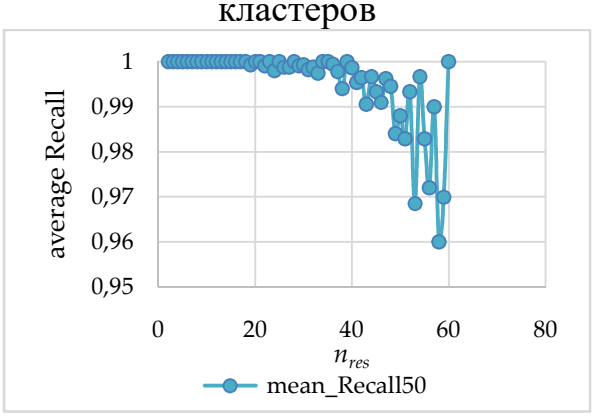
Просмотрено 1% от общего числа кластеров



Просмотрено 3% от общего числа кластеров



Просмотрено 5% от общего числа кластеров



Просмотрено 7% от общего числа кластеров

Рисунок 2.1 – Зависимость среднего достигаемого Recall от n_{res}

Рисунок 2.1 показывает, что чем меньше значение n_{res}, тем выше значение полноты. Это означает, что простые запросы (для которых достаточно поиска в небольшом количестве кластеров) обычно дают лучшие решения. Мы классифицируем запросы по сложности их обработки. Простые запросы – это запросы, результаты которых сосредоточены в небольшом количестве кластеров (низкие значения n_{res}) и при этом обеспечивают высокую полноту. С другой стороны, для сложных запросов результаты распределяются по большему количеству кластеров (высокие значения n_{res}). Для таких запросов полнота может снижаться из-за возросшей сложности поиска, хотя для некоторых запросов высокая полнота все же может сохраняться.

Такая классификация сложности запроса помогает оптимизировать производительность запросов путем разделения запросов на простые и сложные.

На рисунке 2.2 значения запроса представлены следующим образом. Эта диаграмма рассеяния демонстрирует связь между *nprobe* и n_{res}.

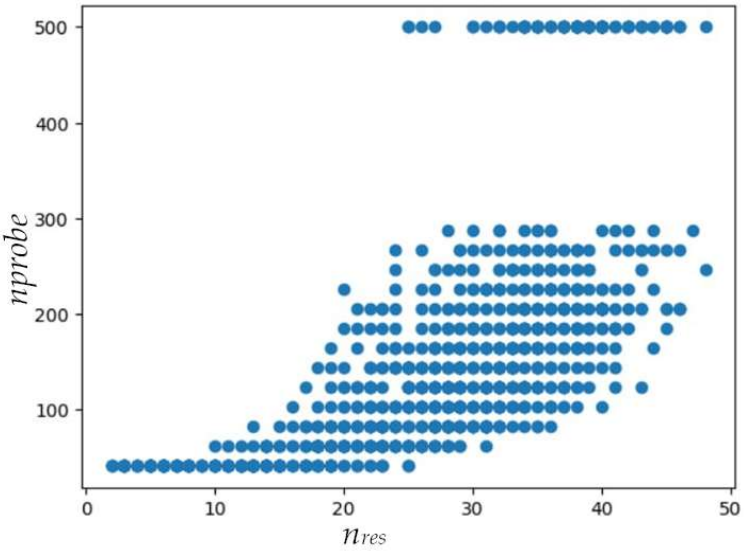
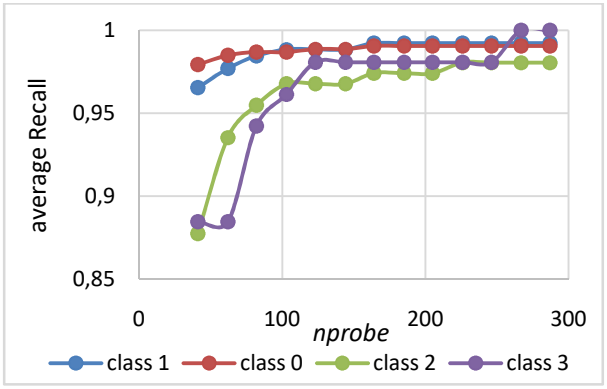
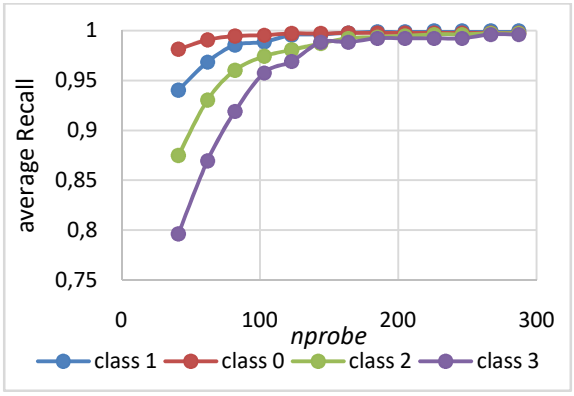


Рисунок 2.2 – Соотношение между числом эффективных кластеров n_{res} и числом обработанных кластеров *nprobe*. По оси *Y*: значение *nprobe*, необходимое для достижения заданного уровня полноты. По оси *X*: соответствующее значение n_{res}

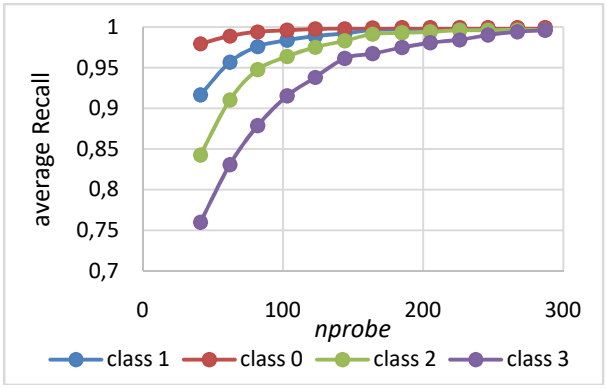
Рисунок 2.2 показывает, что сложность запроса может рассматриваться как непрерывное значение или класс. Для запросов, сгруппированных в четыре равномощных класса сложности, были также построены графики зависимости средних достигнутых значений полноты от значения *nprobe* (рисунки 2.3, 2.4).



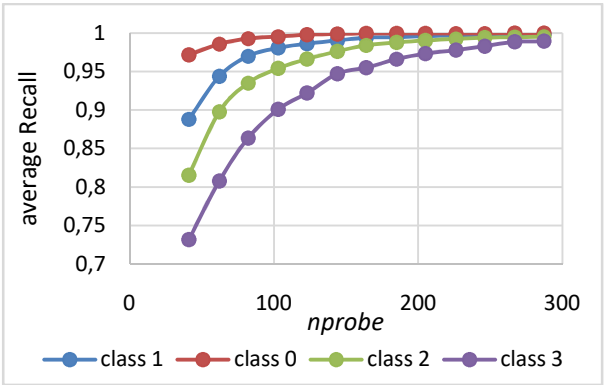
Recall@1



Recall@5

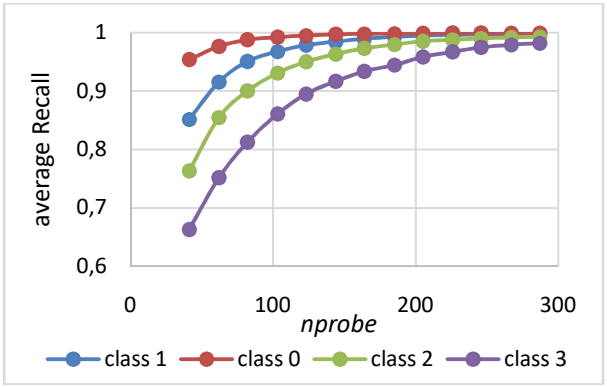


Recall@10

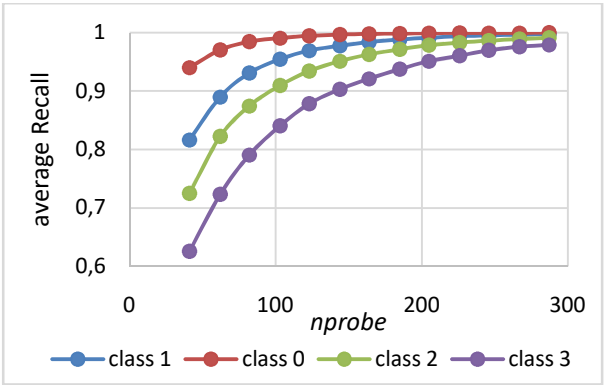


Recall@20

Рисунок 2.3 – Средние значения Recall для четырех классов. Recall@1-Recall@20



Recall@50



Recall@100

Рисунок 2.4 – Средние значения Recall для четырех классов. Recall@50-Recall@100

Также была построена аналитическая модель, аппроксимирующая полученные зависимости для четырех классов сложности (рисунок 2.5). Аналитическая форма модели задается экспоненциальным выражением $c - a \cdot e^{\frac{1-x}{b}}$. Для каждого класса сложности были определены соответствующие

коэффициенты модели, а также среднеквадратичное отклонение (RMSD); указанные значения приведены в таблице 2.2.

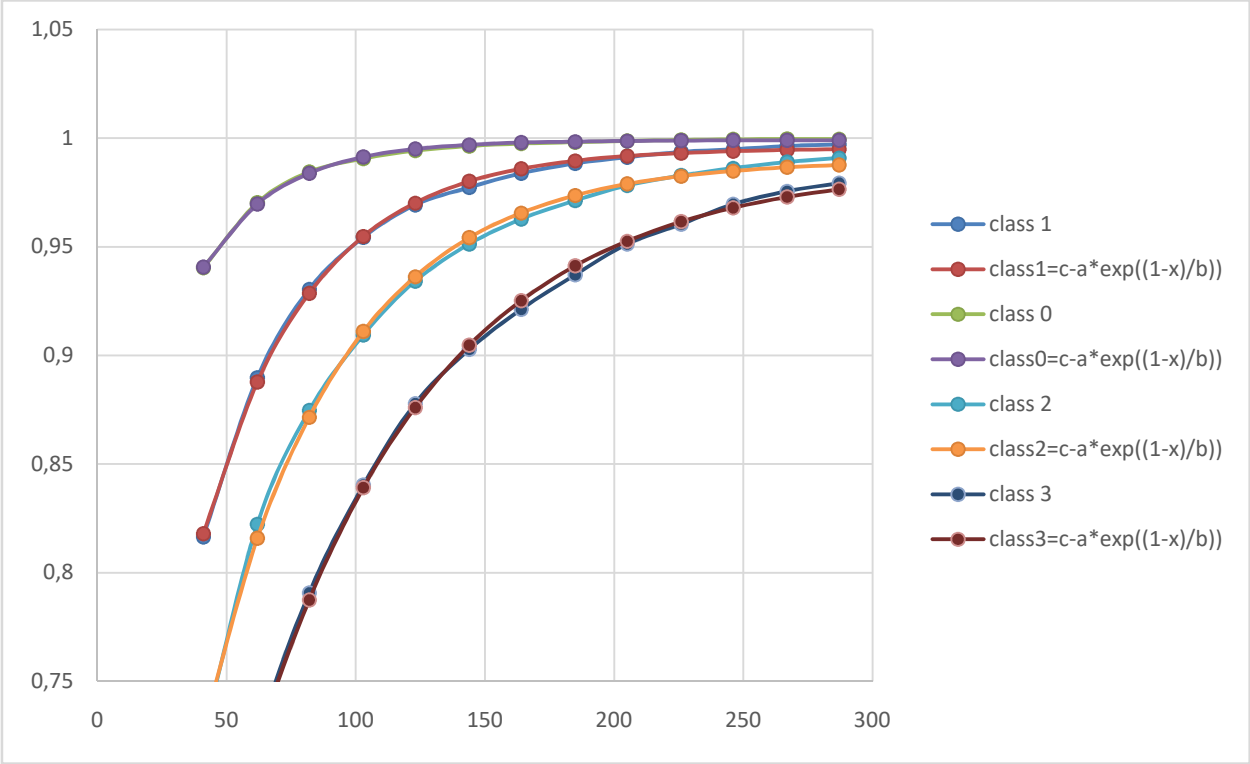


Рисунок 2.5 – Аппроксимация Recall@100 показательной функцией. По оси *X*: значение параметра *nprobe*, по оси *Y*: Recall@100

Таблица 2.2 – Параметры аппроксимирующих функций (рисунок 2.5)

Номер класса сложности	a	b	c	RMSD
class 0	0.216248	30.53076	0.998972	3.68×10^{-6}
class 1	0.460257	41.96785	0.995304	3.48×10^{-6}
class 2	0.565668	51.76675	0.989846	1.19×10^{-6}
class 3	0.635409	70.00414	0.987077	8.45×10^{-6}

Рисунки 2.3-2.5 показывают, что значения *nprobe* могут быть получены аппроксимацией. Были рассмотрены четыре модели обучения с учителем: решающее дерево, случайный лес, SVM (от англ. Support Vector Machine, метод опорных векторов) и линейную регрессию. Методы случайного леса и SVM не дали сравнительно хороших результатов для этой конкретной задачи. Более того,

они потребляют значительно больше вычислительных ресурсов на обучение и обработку входных данных. Помимо решаемой задачи аппроксимации, построенное дерево решений позволяет определить важность факторов (признаков), влияющих на прогноз модели, по формуле

$$\text{Importance}(f) = \frac{1}{N} \sum_{t \in T(f)} N(t) \Delta\phi(t), \quad (2.1)$$

где T(f) – узлы дерева, использующие признак f, N(t) – число объектов обучающей выборки, проходящих через t, $\Delta\phi(t)$ - изменение функции неопределенности (мера равномерности предсказаний классификатора) после прохождения узла, N – число объектов в обучающей выборке [6].

Полученные значения важности признаков представлены в таблице 2.3.

Таблица 2.3 – Важность факторов (формула 2.1) в процессе принятия решений

Фактор	Важность, доля
n_{res}	0.68
$\sigma(l_2)$	0.12
$<l_c>$	0.09
Big5	0.05
$<l_2>$	0.04
$\max(l_2)$	0.02

Согласно данным, приведенным в таблице 2.3, наиболее значимым фактором при классификации запроса является число эффективных кластеров n_{res}. Это позволяет использовать характеристику n_{res} как основу для классификации сложности запроса.

Как отмечалось выше, методы ANN-поиска на основе IVF-индекса подвержены проблеме края: если центроид c_1 является ближайшим центроидом к вектору запроса q, то вектор данных из другого кластера может располагаться на периферии своего кластера и оказаться ближе к вектору запроса, чем ближайшие векторы данных кластера, соответствующего ближайшему центроиду c_1.

Классический подход к «проблеме края» заключается в разделении кластера на «ядро» и «край» (периферию) [6]. Однако данная идея дает лишь минимальный практический эффект.

На рисунке 2.6 q обозначает вектор запроса, c_1 - ближайший к нему центроид, а c_2 – один из прочих построенных центроидов. Пусть расстояния определены как $D_1 = \|q - c_1\|$ и $D_2 = \|q - c_2\|$. Тогда внутри радиуса D_2–D_1 вокруг центроида c_2 не может находиться ни один ближайший сосед для q. Соответствующая область вокруг c_2 может быть определена как область «ядра» кластера, с центроидом c_2.

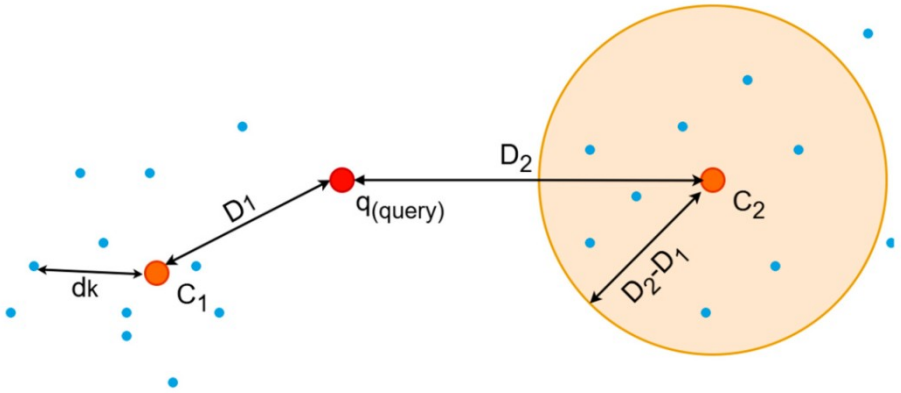


Рисунок 2.6 – Иллюстрация к «проблеме края»

Для ускорения вычислений для векторов данных, принадлежащих ядерным областям кластеров, в IVF-индексе может быть выполнено предварительное вычисление расстояний от каждого вектора данных до его ближайшего центроида d_k. Далее, после вычисления расстояния D^* от поступающего запроса q до его ближайшего центроида, может быть выполнено частичное исключение из рассмотрения векторов данных, относящихся к ядрам других кластеров. Для j-го кластера при заданном расстоянии D_j от вектора запроса до центроида c_j из рассмотрения исключаются те элементы j-го кластера, для которых $d_k \leq D_j - D^*$. Радиус D_j-D^* очерчивает области ядер кластеров (относительно конкретного запроса q), и внутри этого радиуса эффект края не возникает. Очевидно, кластеры,

для которых D_j>2D^*, также могут быть безопасно исключены из обработки запроса при поиске единственного ближайшего соседа.

Кроме того, если запрос попадает внутрь гиперсферы с центром в ближайшем центроиде и радиусом $\frac{1}{4}D'$, где D' - это расстояние от ближайшего центроида до его ближайшего соседнего центроида (указанные расстояния также могут быть вычислены на этапе построения IVF-индекса), то поиск единственного ближайшего соседа для q может быть ограничен внутренней областью данной гиперсферы. То есть, если $D^* < \frac{1}{4}D'$, то ближайший сосед гарантированно находится в кластере, ассоциированном с ближайшим центроидом.

Однако применение данного принципа при выполнении ANN-поиска в многомерных эмбеддингах практически не дает полезного эффекта, увеличивая при этом вычислительную сложность построения индекса. На рисунке 2.7 приведены графики плотности распределений расстояний d_k и D'. Рисунок 2.7 демонстрирует, что как внутрикластерные расстояния d_k, так и расстояния между центроидами D' сосредоточены в узком диапазоне (распределение имеет малую дисперсию). Фактически в пространстве высокой размерности, таком как рассматриваемый 128-мерный набор эмбеддингов (SIFT), почти все попарные расстояния оказываются приблизительно равными. В результате условие $d_k \leq D_j - D^*$ задает гиперсферу пренебрежимо малого объема, аналогично гиперсферам, определяемым условием $D^* < \frac{1}{4}D'$.

Предварительные эксперименты на наборе данных SIFT1M, ориентированные на ускорение поиска ближайших соседей за счет предварительного вычисления расстояний и проверки условий $d_k \leq D_j - D^*$ и $D^* < \frac{1}{4}D'$ показали, что указанные условия выполняются менее чем для 0,35% тестовых запросов (бенчмарков). Следовательно, попытки выделить «ядерные» и «пограничные» области кластеров, внутри которых эффект края не проявляется, и фильтрация данных на основе этих условий лишь замедляют поиск (вследствие накладных затрат на проверку условий).

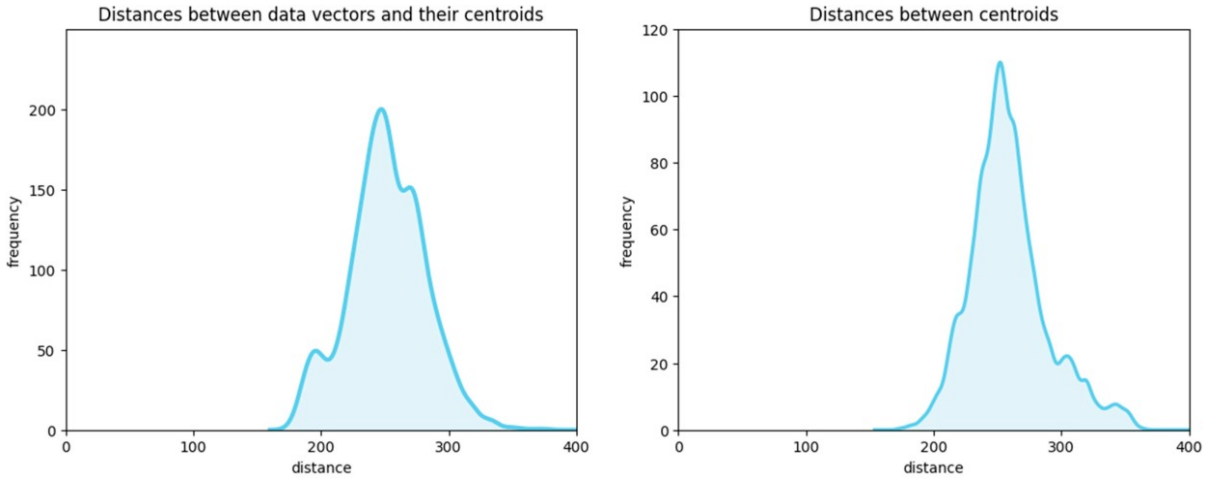


Рисунок 2.7 – Плотность распределения расстояний до ближайшего центроида (слева) и между центроидами (справа)

Данный факт также косвенно объясняет ограниченную информативность тех характеристик запросов (таблица 2.3), которые основаны на агрегации межкластерных или внутрикластерных расстояний.

Во многих задачах кластеризации также широко применяются методы, основанные на относительных расстояниях. Например, алгоритм Элкана [118] и соответствующие подходы используют так называемые «ядра кластеров» (cluster cores). Эти методы опираются на сравнение расстояний между векторами запросов, векторами данных и центроидами. Однако результаты, приведенные в таблице 2.3, показывают, что признаки, основанные на расстояниях, играют лишь второстепенную роль. Результаты, полученные при использовании алгоритма Элкана для построения индекса, не демонстрируют улучшений относительно результатов, полученных с использованием алгоритма *k*-средних.

В то же время влияние эффекта края, проявляющегося в разнообразии кластеров, в которых обнаруживаются ближайшие соседи, может быть оценено по результатам ранних стадий поиска. Соответственно, признак n_{res} является существенно более информативным по сравнению с «геометрическими» признаками, такими как $<l_2>$, $\sigma(l_2)$, $\max(l_2)$, $<l_c>$, $\sigma(l_c)$ и $\max(l_c)$.

2.2 Выбор числа обрабатываемых кластеров в зависимости от числа результативных кластеров на начальной стадии поиска

Для построения эффективного классификатора сложности запроса важно понимать природу зависимости между n_{res} и *nprobe*. На рисунке 2.8 показано, как в среднем растет количество эффективных кластеров по мере роста количества обработанных кластеров. Эта зависимость позволяет оценить количество кластеров, необходимое для достижения высокого значения полноты. Изучение этой тенденции поможет понять, как выбор *nprobe* влияет на эффективность модели.

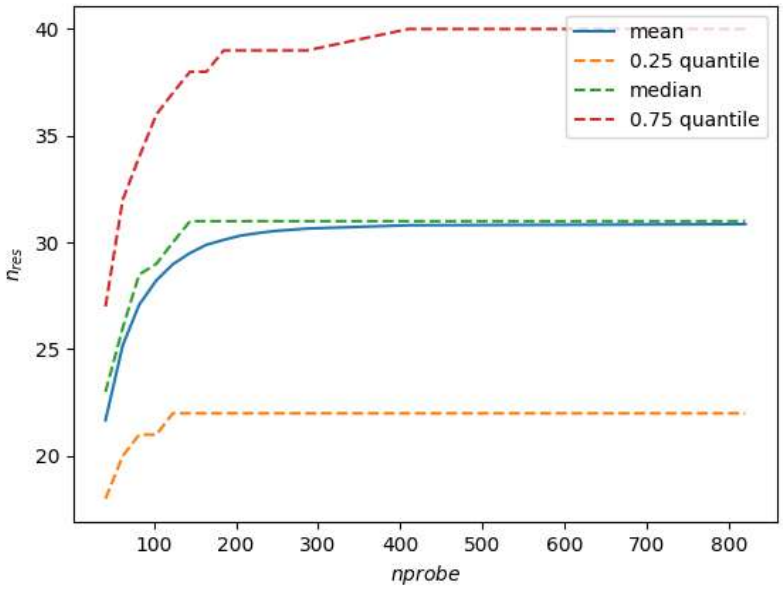


Рисунок 2.8 – Среднее значение n_{res}, первый, второй и третий квартили значений n_{res} в зависимости от *nprobe*

Рисунок 2.8 иллюстрирует, что зависимость имеет нелинейный характер. Это означает, что, как правило, рассмотрения лишь небольшого количества кластеров достаточно для достижения требуемого значения полноты.

На рисунке 2.9 показано, что при увеличении параметра *nprobe* дисперсия значений Recall уменьшается. При малых значениях *nprobe* увеличение *K*

приводит к снижению среднего значения Recall. Данное наблюдение подчеркивает значимость адаптивной реализации алгоритма, поскольку различным запросам требуется неодинаковый объем вычислительных ресурсов для достижения высокого показателя полноты. Характер распределений на гистограммах указывает на возможность эффективной сегментации набора данных на несколько сбалансированных классов сложности. Границы указанных классов могут быть определены путем задания пороговых значений n_{res}, выступающих маркерами различных уровней сложности. Предложенный подход обеспечивает более рациональное распределение ресурсов с учетом трудоемкости обработки запроса.

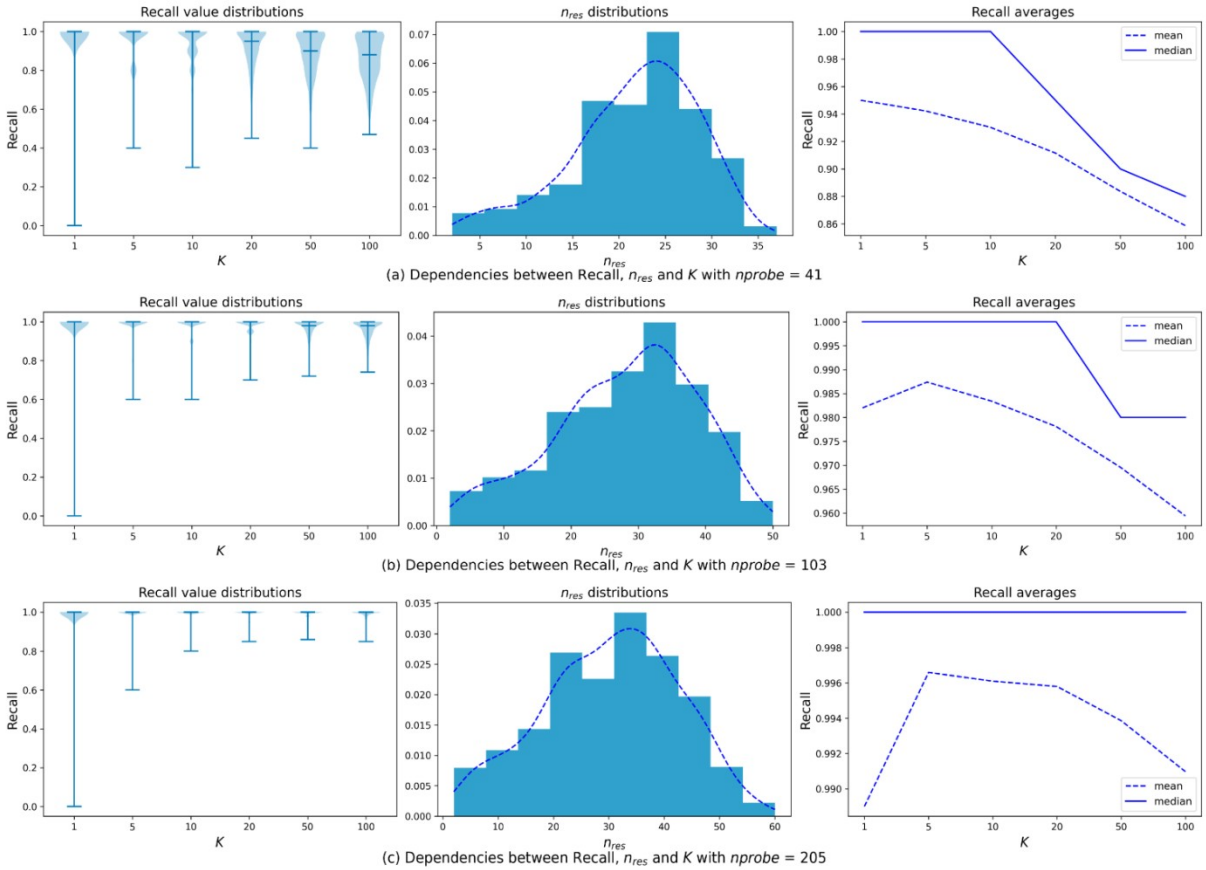
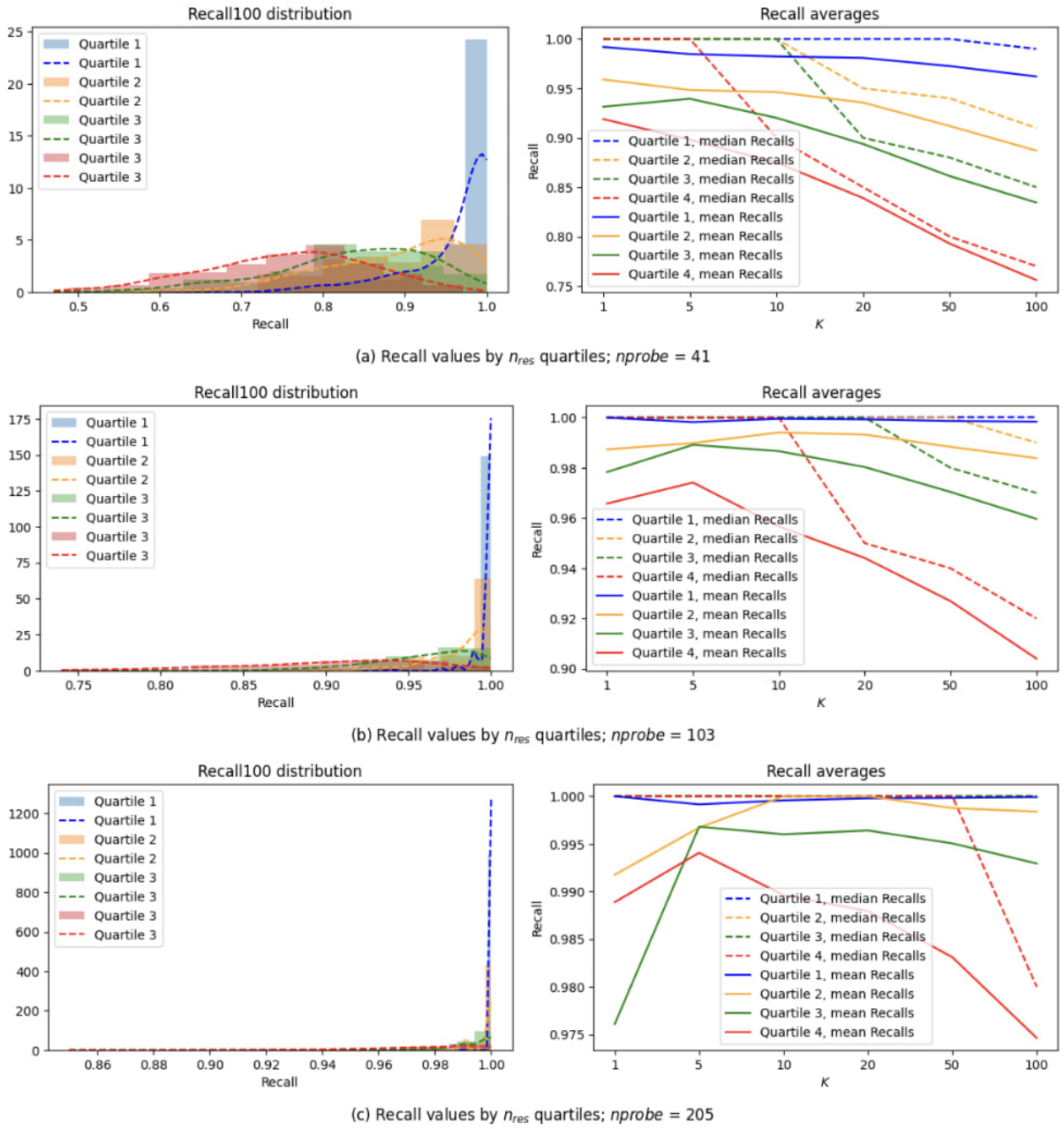


Рисунок 2.9 – Распределение значений достигнутой полноты и числа результативных кластеров для различных значений *nprobe*

На рисунке 2.10 изображено распределение значений Recall по классам сложности, определяемых как квартили значений n_{res}.



(a) Recall values by n_{res} quartiles; *nprobe* = 41

(b) Recall values by n_{res} quartiles; *nprobe* = 103

(c) Recall values by n_{res} quartiles; *nprobe* = 205

Рисунок 2.10 – Распределения значений n_{res} и ожидаемого Recall в зависимости от запроса

После проведенного анализа был построен простой классификатор на основе логических правил – не глубокое и очень простое решающее дерево. Сложность правил, которые получались в ходе обучения классификатора, зависит от объема обучающей выборки. Правила могут включать средние расстояния, их среднеквадратичные отклонения, но самой важной характеристикой остается число кластеров, в которых найден хотя бы один ближайший сосед. Было

просканировано 127 кластеров в наборе данных SIFT1M, и количество кластеров, в которых был найден по крайней мере один ближайший сосед, варьировалось от 13 до 70.

Логические правила, определяющие сложность запросов (пример классификатора №1):

- если $n_{res} \leq 19$, то q – простой запрос;
- если $n_{res} \leq 21$ и $\sigma(l_2) \geq 4980.56$, то q – простой запрос;
- если $n_{res} \leq 28$ и $\sigma(l_c) \geq 9593.3$, то q – простой запрос;
- если $n_{res} \leq 33$, $<l_2> \leq 48853.7$ и $\max(l_c) \geq 77234.4$, то q – простой запрос;
- если $n_{res} \leq 34$, $<l_2> \leq 37432.2$ и $<l_c> \leq 36761.1$, то q – простой запрос;
- иначе q считается сложным запросом.

Если индекс для приближенного поиска ближайших соседей строится по исходным векторам данных и не требует проведения никакой вычислительно сложной предварительной подготовки данных, то для обучения классификатора сложности запросов необходимо знать истинные значения Recall. Вычисление истинных ближайших соседей вычислительно затратно, так как нужно найти множество истинных ближайших соседей при помощи точного поиска. Таким образом, классификатор будет обучен на минимальном числе запросов.

Логические правила, определяющие сложность запросов, основанные на обучении на 100 векторах-запросах (пример классификатора №2):

- если $n_{res} \leq 21$, то q – простой запрос;
- если $n_{res} \leq 30$) и число просмотренных векторов $\geq 31{,}676$, то q – простой запрос;
- иначе q считается сложным запросом.

Если классификатор обучается всего на 10 запросах, то остается лишь одно логическое правило. В приведенном ниже случае число кластеров, давших по крайней мере одного ближайшего соседа, должно быть меньше 28. В противном случае запрос считается сложным.

Простейшее логическое правило, определяющее сложность запросов, основанное на обучении на 10 векторах-запросах (пример классификатора №3):

- Если $n_{res} \leq 28$, то q считается простым запросом.

Точность такого классификатора составляет 0,85 для набора данных SIFT1M. Таким образом, классификатор всего с одним логическим правилом способен достаточно точно определять сложные запросы. Точности для всех трех классификаторов приведены в таблице 2.4.

Таблица 2.4 – Сравнение точности классификаторов с разным количеством логических правил

Классификатор	Точность
Пример классификатора №1, 6 логических правил	0.91
Пример классификатора №2, 3 логических правил	0.83
Пример классификатора №3, 1 логическое правило	0.85

Основываясь на приведенных выше результатах предварительных экспериментов для проверки гипотезы об эффективности аппроксимации *nprobe* как функции от n_{res}, была разработана простая модель бинарного классификатора. Разработанный алгоритм бинарной классификации работает в два этапа: на первом этапе для каждого запроса обрабатывается некоторое не слишком большое число кластеров (в данном примере 127 кластеров), а затем классификатор применяется к каждому запросу. Если запрос классифицируется как простой, алгоритм заканчивает свою работу и возвращает ближайших соседей, полученных из 127 кластеров. В противном случае просматриваются еще 127 кластеров. В таблице 2.5 показано сравнение двух алгоритмов: первый алгоритм сразу обрабатывает 184 кластера для каждого запроса (Стандартный IVF-поиск), а второй сначала обрабатывает 127 кластеров, затем классифицирует запрос и при необходимости обрабатывает еще 127 кластеров (IVF поиск с бинарной классификацией сложности запроса).

Таблица 2.5 – Результат работы двухэтапного бинарного классификатора в сравнении со стандартным алгоритмом поиска по файлу обратного индекса

Метод	Количество обработанных кластеров (в среднем)	Количество обработанных векторов (в среднем)	Recall@100 (в среднем, 10000 запросов)
Стандартный IVF-поиск	184	45519.10	98.84%
IVF-поиск с бинарной классификацией сложности запроса	182.96	45498.71	99.15%

В обоих алгоритмах (таблица 2.5) обрабатывается примерно одинаковый объем данных в каждом запросе. Второй алгоритм в среднем обрабатывает даже меньше данных. Для простых запросов он обрабатывает 127 кластеров, а для сложных – вдвое больше. Средняя вычислительная сложность одинаковая; однако, среднее достигаемое значение полноты у второго подхода значительно выше. Снижение уровня ошибок составляет около 26%. Этот подход может повысить среднее достигаемое значение Recall или существенно сократить время обработки запросов. Результаты показали, что даже этот простой классификатор дает определенные улучшения, а концепция классификации сложности запросов выглядит многообещающей.

2.3 Классификатор сложности запросов

На основе результатов, представленных в разделе 2.2, был создан классификатор, который относит запросы к одному из четырех классов сложности в зависимости от числа результативных кластеров n_{res} после обработки

некоторого минимального числа кластеров $n_{minchecked}$. Первый класс представляет запросы, для которых сканирования $n_{minchecked}$ кластеров достаточно, чтобы достичь целевого значения полноты, расширенный поиск для первого класса сложности не требуется; остальные три класса формируются путем ранжирования оставшихся запросов по значению n_{res} и последующего разбиения (согласно числу запросов) упорядоченного множества на три равномощных подмножества. Метка класса используется для определения объема дополнительных вычислений: она указывает, сколько дополнительных кластеров следует просканировать для конкретного запроса, то есть на сколько требуется увеличить параметр *nprobe*, чтобы обеспечить достижение требуемого уровня полноты.

Алгоритм классификации сложности запросов (алгоритм 2.1) предназначен для определения числа кластеров для сканирования (*nprobe*), обеспечивающего достижение требуемого уровня полноты при ANN поиске, представляя собой метод динамического определения глубины поиска в IVF-индексе на основе оценки результативности кластеров.

Для обучения классификатора используется выборка, состоящая из случайно отобранных векторов данных, содержащихся в базе данных. Как показано в предыдущем разделе, для обучения классификатора достаточно небольшой выборки, хотя ее объем, безусловно, влияет на точность классификатора и, как следствие, на скорость и точность ANN поиска. Для каждого вектора данных, включенного в такую выборку, необходимо с высокой точностью определить значение n_{res}. В ходе определения n_{res} алгоритмом многократно вычисляются получаемые значения Recall. Это требует определения истинных ближайших соседей для каждого вектора данных, включенного в обучающую выборку классификатора, что вычислительно сложно. Таким образом, должен быть обеспечен компромисс между объемом обучающей выборки и точностью обученного классификатора. Вопрос требуемого объема выборки в рамках настоящего исследования подробно не рассматривался. В приведенных ниже экспериментах использовались выборки из 200 векторов данных, имитирующих запросы. Как показывают результаты исследования,

такого объема выборки достаточно для обучения классификатора сложности запросов и существенного повышения производительности алгоритма приближенного поиска ближайших соседей. Алгоритм обучения классификатора в качестве гиперпараметров требует ожидаемого значения Recall@K, числа ближайших соседей *K*, объема обучающей выборки и переменной $n_{minchecked}$.

Затем для каждого запроса из множества Q определяются истинные ближайшие соседи с использованием точного поиска. Для каждого запроса $q_i \in Q$ счетчик *nprobe* инициализируется нулем. Затем алгоритм начинает итеративно увеличивать значение *nprobe*, пока для q_i не будет достигнут целевой показатель Recall@K.

Затем обучающая выборка запросов разделяется на четыре группы (класса) приблизительно одинакового размера на основе значений n_{res}. Класс 1 определяется значением $n_{minchecked}$, которое представляет собой минимальное количество кластеров для обработки. Классы 2-4 должны быть приблизительно одинакового размера, поэтому для определения границ классов сложности используются 33-й и 66-й процентили.

Наконец, вычисляются средние значения *nprobe* для каждого из четырех классов сложности. Эти значения *nprobe* используются в алгоритме адаптивного ANN.

Алгоритм 2.1 - Алгоритм обучения классификатора сложности запросов

Дано: Обучающая выборка запросов *Q*, представленная в виде случайной выборки векторов данных; ожидаемое значение полноты Recall@K; количество ближайших соседей *K*; минимальное количество проверяемых кластеров $n_{minchecked}$.

Шаг 1. Для каждого запроса *q* из набора *Q* выполнить исчерпывающий поиск для определения истинных ближайших соседей;

Шаг 2. Для каждого запроса q_i в наборе *Q* выполнить поиск в $n_{minchecked}$ кластерах и рассчитать количество найденных результатов n_{res};

Шаг 3. Для каждого запроса q_i в наборе *Q* выполнять:

Шаг 4. Установить начальное количество проверяемых кластеров *nprobe*=0;

Шаг 5. Увеличить значение *nprobe* на единицу (*nprobe*←*nprobe*+1);

Шаг 6. Оценить значение полноты Recall(q_i)@K, полученное после сканирования *nprobe* кластеров;

Шаг 7. Если Recall(q_i)@K ≥ Recall@K, то вернуться к шагу 5;

Шаг 8. Зафиксировать необходимое количество кластеров для данного запроса: $nprobe@q_i=nprobe$;

Шаг 9. Завершить цикл для текущего запроса;

Шаг 10. Разделить набор Q на четыре подмножества на основе $nprobe@q_i$ и определить границы классов сложности следующим образом:

Шаг 10.1. Установить $M1=n_{minchecked}$;

Шаг 10.2. Сформировать множество *Class*1, включающее запросы, где $nprobe@q_i \leq n_{minchecked}$;

Шаг 10.3. Определить остаточное множество *Classes*234=$Q \setminus$*Class*1;

Шаг 10.4. Рассчитать $M2$ как 33-й процентиль значений *nprobe* среди запросов из *Classes*234;

Шаг 10.5. Рассчитать $M3$ как 66-й процентиль тех же значений;

Шаг 10.6. Сформировать множества классов сложности: *Class*2 ← $M1 < nprobe@q_i \leq M2$; Class3 ← $M2 <$ *nprobe*@$q_i \leq$ M3; Class4 ← $nprobe@q_i >$ M4;

Шаг 11. Для каждого класса сложности $i \in \{1,2,3,4\}$ рассчитать среднее значение $nprobe@q_i$, при котором достигается требуемая полнота, и сохранить их как параметры $S1,S2,S3,S4$.

Шаг 12. Вернуть полученные значения порогов классов и значения *nprobe* для каждого класса: $M1,M2,M3,S1,S2,S3,S4$.

Значения *expected_K*, *expected_recall@expected_K*, $n_{minchecked}$, $M1$, $M2$, $M3$, $S1$-$S4$. составляют дополнение к индексу. Значение *expected_K* хранит количество соседей, для которого построен индекс. Значение *expected_recall@expected_K* показывает ожидаемое значение полноты для этих K соседей. Значение $n_{minchecked}$ показывает количество кластеров, которые необходимо обработать для вычисления значений n_{res}. Значения *M1*, *M2*, *M3* служат пограничными

значениями между классами. Значения $S1$-$S4$ являются средними значениями *nprobe*, при которых достигается *expected_recall* на *expected_K* соседях.

Алгоритм 2.2 описывает процесс определения ближайших соседей для вектора запроса на основе заданных параметров. Процесс начинается с получения вектора запроса q, указанного количества ближайших соседей K и ожидаемого уровня полноты.

Затем алгоритм проверяет наличие дополнения к построенному индексу. Если для построенного индекса нет дополнения (классификатор не обучен), то запускается стандартная процедура ANN. В обратном случае алгоритм ищет K ближайших соседей в заданном минимальном количестве кластеров, обозначенном как $n_{minchecked}$. Значение *nprobe* уточняется по результатам этого поиска.

Алгоритм 2.2 - Адаптивный алгоритм поиска приближенных ближайших соседей

Дано: Вектор запроса q, количество искомых ближайших соседей K, минимальное количество проверяемых кластеров $n_{minchecked}$ границы классов сложности $M1$,$M2$,$M3$, значения количества проверяемых кластеров *nprobes* для каждого класса сложности $S1$,$S2$,$S3$,$S4$, обученный классификатор для K, $n_{minchecked}$ и ожидаемой Recall@K (см. алгоритм 2.1).

Шаг 1. Проверить наличие дополнения к индексу (обученного классификатора) для заданных K, $n_{minchecked}$ и ожидаемой Recall@K. Если классификатор не обучен: выполнить стандартную процедуру поиска ближайших соседей (ANN) и завершить алгоритм;

Шаг 2. Выполнить поиск K ближайших соседей в фиксированном минимальном количестве кластеров $n_{minchecked}$ алгоритмом IVF-поиска;

Шаг 3. Рассчитать количество найденных результатов n_{res}.

Шаг 4. Классифицировать сложность запроса по количеству результатов n_{res} и выбрать соответствующее значение *nprobes*:

Если $n_{res} \leq M1$, то $nprobes \leftarrow S1$

Если $M1 < n_{res} \leq M2$, то $nprobes \leftarrow S2$

Если $M2 < n_{res} \leq M3$, то $nprobes \leftarrow S3$

Если $n_{res} > M3$, то $nprobes \leftarrow S4$

Шаг 5. Выполнить дополнительный поиск алгоритмом IVF-поиска в ($nprobes$ - $n_{minchecked}$) дополнительных кластерах.

Шаг 6. Вернуть найденных K ближайших соседей.

Алгоритм 2.1, который должен запускается после построения IVF-индекса, предоставляет почти все параметры для алгоритма 2.2, выполняющего адаптивную обработку запросов IVF. Однако параметр $n_{minchecked}$ (минимальное число кластеров, подлежащих сканированию) должен быть задан заранее.

С одной стороны, данный параметр определяет относительный размер класса сложности Class1 (наиболее простые запросы). Таким образом, значение $n_{minchecked}$ должно быть выбрано так, чтобы доля данного класса была приблизительно равна долям остальных классов, то есть сканирования $n_{minchecked}$ кластеров должно быть достаточно для достижения требуемого значения Recall примерно для четверти запросов.

С другой стороны, выбор значения $n_{minchecked}$ критически важен для корректной работы классификатора, основанного на значении n_{res}. Если общее число кластеров в IVF-индексе невелико, то сканирование малого числа кластеров обеспечивает требуемое значение Recall для 25% запросов. В этом случае, поскольку значения n_{res} являются целыми числами, их дискретность может приводить к получению одинаковых значений $M1$–$M3$ для различных классов запросов.

Однако, поскольку диссертационная работа ориентирована на достижение высоких значений Recall для достаточно крупных наборов данных, предполагается, что IVF-индекс содержит как минимум несколько тысяч кластеров, и, следовательно, число сканируемых кластеров составляет, по меньшей мере, несколько десятков, чего достаточно для того, чтобы алгоритм 2.1 мог корректно установить границы между классами сложности.

2.4 Вычислительный эксперимент

В алгоритме 2.2 запросы разделяются на четыре группы в зависимости от их сложности. Классификация запросов по сложности позволяет оптимизировать процесс поиска: более сложным запросам назначается большее значение *nprobe*, тогда как для более простых запросов применяется меньшее значение *nprobe*. Такой подход позволяет сохранять требуемый уровень Recall.

Алгоритмы 2.1 и 2.2 были реализованы для крупных наборов данных различных размеров. Для всех использованных в экспериментах наборов данных после построения IVF-индекса выполнялся алгоритм 2.1, чтобы обучить классификатор сложности запросов и создать дополнение к индексу.

Для экспериментов, приведенных ниже, использовался набор данных SIFT10M - подвыборка из 10 миллионов записей 128-мерного набора данных SIFT1B [33] (также известного как BIGANN). SIFT1B состоит из 10^9 объектов размерности 128. Набор данных SIFT1B, а также его поднаборы традиционно применяются для оценки эффективности приближенного поиска ближайших соседей. Все эти наборы данных поставляются вместе со стандартными наборами тестовых запросов [33], которые использовались в экспериментах. Во всех экспериментах использовалось евклидово расстояние.

Эксперименты проводились с использованием евклидова расстояния и обладали следующими характеристиками: число параллельно исполняемых процессов (режим Parallel on) для одновременной обработки запросов СУБД; число кластеров (размер кодовой книги) *nlist*; число искомых ближайших соседей K=100; параметр $n_{minchecked} = 40$.

Технические характеристики вычислительного оборудования: процессор 8×Ascend 910B4, 1.5 ТБ DDR4 ОЗУ, накопитель 7 ТБ NVMI.

Результаты вычислительных экспериментов, направленных на оценку стабильности и эффективности реализованного адаптивного алгоритма (алгоритм 2.2) при различных значениях параметров (*M*1, *M*2, *M*3, *S*1, *S*2, *S*3 и *S*4), представлены в таблицах 2.6–2.10, где QPS – среднее количество обработанных

запросов в секунду, а время отклика – среднее время между подачей запроса и получением ответа.

Таблица 2.6 - Результаты с перестроением индекса: алгоритм с адаптивным *nprobe* (алгоритм 2.2). Набор данных SIFT10M; размер выборки для построения индекса: 10^4

	Индекс №1		Индекс №2		Индекс №3		Индекс №4	
Номер запуска	1	2	1	2	1	2	1	2
K	100	100	100	100	100	100	100	100
Parallel processes	100	100	100	100	100	100	100	100
$M1$	17	17	17	17	17	17	17	17
$M2$	22	22	22	22	22	22	22	22
$M3$	29	29	29	29	29	29	29	29
$M1/n_{minchecked}$	0.43	0.43	0.43	0.43	0.43	0.43	0.43	0.43
$M2/n_{minchecked}$	0.55	0.55	0.55	0.55	0.55	0.55	0.55	0.55
$M3/n_{minchecked}$	0.73	0.73	0.73	0.73	0.73	0.73	0.73	0.73
$S1$	40	40	40	40	40	40	40	40
$S2$	100	100	100	100	100	100	100	100
$S3$	141	141	141	141	141	141	141	141
$S4$	185	185	185	185	185	185	185	185
Recall@100	0.9913	0.9913	0.9912	0.9912	0.9913	0.9913	0.9910	0.9910
Время отклика, мс	227.77	222.5	235.17	232.34	224.73	222.63	225.52	226
QPS, c^{-1}	436.59	446.87	422.13	427.46	442.22	446.85	437.02	440.29
nlist	3150	3150	3150	3150	3150	3150	3150	3150

Таблица 2.7 - Результаты эксперимента со стандартным алгоритмом IVF-поиска, фиксированное значение *nprobe* = 125. Набор данных: SIFT10M; размер выборки для построения индекса: 10^4

	Индекс №1		Индекс №2		Индекс №3		Индекс №4	
Номер запуска	1	2	1	2	1	2	1	2
nprobe	125	125	125	125	125	125	125	125
K	100	100	100	100	100	100	100	100
Parallel processes	100	100	100	100	100	100	100	100
Recall@100	0.9912	0.9912	0.9911	0.9911	0.9913	0.9913	0.9912	0.9912
Время отклика, мс	296.53	288.4	294.87	315.64	319.58	271.42	297.33	286.06
QPS, c^{-1}	335.61	344.78	337.15	314.85	310.92	366.25	334.37	347.47
nlist	3150	3150	3150	3150	3150	3150	3150	3150

Таблица 2.8 - Усредненные значения параметров $M1$, $M2$, $M3$, $S1$, $S2$, $S3$, $S4$. Набор данных: SIFT10M. σ – среднеквадратичное отклонение

Номер эксперимента	1	2
K	100	100
Parallel processes	100	100
$M1$	10	10
$M2$	18	17
$M3$	23	23
$S1$	12	9
$S2$	47	37
$S3$	82	88
$S4$	158	163
Recall@100	0.99004 (средн.), σ = 0.0002	0.99072 (средн.), σ = 0.0002
Время отклика, мс	227.428 (средн.), σ = 14.135	246.847 (средн.), σ = 14.623
QPS, c^{-1}	437.622	404.823
nlist	3150	3150

Таблица 2.9 – Сравнение результатов работы алгоритма полного перебора, стандартного алгоритма IVF-поиска и алгоритма 2.2. Набор данных: SIFT100M (10^8 векторов данных); размер выборки для построения индекса: 10^5

Алгоритм поиска	Полный перебор	Стандартный IVF-поиск, Средние значения по результатам 6 запусков	Адаптивный IVF, Индекс 1		Адаптивный IVF, Индекс 2		Адаптивный IVF, Индекс 3	Адаптивный IVF, Индекс 4
K	100	100	100	100	100	100	100	100
Parallel processes	100	100	100	100	100	100	100	100
M1	-	-	15	15	15	15	15	15
M2	-	-	23	23	23	23	23	23
M3	-	-	27	27	27	27	27	27
$M1/n_{minchecked}$	-	-	0.38	0.38	0.38	0.38	0.38	0.38
$M2/n_{minchecked}$	-	-	0.58	0.58	0.58	0.58	0.58	0.58
$M3/n_{minchecked}$	-	-	0.68	0.68	0.68	0.68	0.68	0.68
$S1$	-	-	40	40	40	40	40	40
$S2$	-	-	85	85	85	85	85	85
$S3$	-	-	140	140	140	140	140	140
$S4$	-	-	255	255	255	255	255	255
Recall@100	1.00	0.9863	0.9904	0.9902	0.9902	0.9901	0.9901	0.9906
Время отклика, мс	-	719.8	660.54	602.82	572.32	670.82	673.5	679.2
QPS, c^{-1}	19.61	136.21	147.14	158.3	167.32	145.21	145.02	144.88
nlist	-	10,000	10,000	10,000	10,000	10,000	10,000	10,000
nprobe	-	120	средн. 106.4	средн. 106.4	средн. 106.4	средн. 106.4	средн. 106.4	средн. 106.4

Таблица 2.10 - Сравнение результатов работы стандартного алгоритма IVF-поиска и алгоритма 2.2. Набор данных: SIFT1B (10^9 векторов данных); размер выборки для построения индекса: 10^6. σ – среднеквадратичное отклонение

	Recall100@100	Время отклика, мс	QPS, с-1
Стандартный IVF	0.9904 (средн.), σ = 0.0002	7959	6.08
Адаптивный IVF (алгоритм 2.2)	99.05 (средн.), σ = 0.0002	5607	8.47
Прирост, %		-29.6%	39.3%

Для оценки сравнительной эффективности предложенных алгоритмов (алгоритмы 2.1-2.2) были использованы пять версий IVF-индекса, поскольку эффективность поиска зависит в том числе и от качества построенного IVF-индекса.

Для оценки эффективности алгоритма приближенного поиска ближайших соседей наиболее значимыми метриками являются средняя задержка (время от момента поступления ANN-запроса до завершения его обработки) и QPS (англ. queries per second, число запросов в секунду).

Усредненные ключевые метрики первого эксперимента (таблица 2.6) составили: Recall@100 = 0.9912, QPS = 434.84.

Для получения центроидов кластеров алгоритмы построения IVF-индекса могут использовать как стандартный алгоритм кластеризации *k*-средних (индексы 3-5 в таблицах 2.6-2.7), так и более сложные алгоритмы, такие как жадная агломеративная кластеризация (алгоритм 3.5) (индексы 1 и 2 в таблицах 2.6-2.7).

Алгоритмы кластеризации характеризуются высокой вычислительной стоимостью, особенно агломеративные методы. В связи с этим при построении IVF-индексов на больших наборах данных традиционно используется не весь набор данных, а его выборка [6]. В представленных экспериментах для создания кодовых книг также использовалась случайная выборка. После определения положений всех центроидов для каждого вектора данных из полного набора был найден ближайший центроид. Это могло незначительно снизить итоговое

качество индекса; однако использование выборки позволило сократить время построения индекса.

В таблице 2.7 приведены результаты эксперимента на том же наборе данных для стандартного алгоритма IVF-поиска при фиксированном значении *nprobe* = 125. Агрегированные ключевые метрики данного эксперимента составили: Recall со средним значением 99.12 и QPS со средним значением 337.26. Во всех экспериментах со стандартным алгоритмом поиска IVF выбиралось минимальное значение *nprobe*, обеспечивающее требуемое значение Recall (для набора данных SIFT10M при *nlist* = 3150 такое минимальное значение составило 125).

Экспериментальные результаты, представленные в таблицах 2.6-2.10, демонстрируют преимущество предложенного адаптивного алгоритма приближенного поиска ближайших соседей по сравнению со стандартным поиска IVF-поиска с фиксированным *nprobe*: Время отклика уменьшается на 23%, тогда как QPS увеличивается на 25% при сохранении требуемого уровня Recall выше 0.99. Статистическая значимость преимущества алгоритма 2.2 подтверждается критерием Манна–Уитни–Уилкоксона.

Алгоритм 2.1 возвращает семь параметров поиска (*M*1-*M*3 и *S*1-*S*4), однако значения границ классов *M*1-*M*3 можно настроить вручную. В таблице 2.8 показано, как точная настройка этих параметров может повлиять на качество и производительность поиска. Представленные результаты - это усредненные метрики, полученные при 10 запусках на 10 различных индексах. Приведенные в таблице 2.8 результаты показывают, что экспертное уточнение параметров *M*1-*M*3 и *S*1-*S*4 может привести к повышению эффективности.

Результаты, представленные в таблице 2.10, демонстрируют существенное повышение производительности поиска, достигаемое предложенным адаптивным алгоритмом при применении к набору данных объемом 10^9 объектов. Тем самым подтверждается способность алгоритма 2.2 эффективно работать с данными большого масштаба, обеспечивая при этом высокие значения полноты.

2.5 Результаты главы 2

В настоящей главе предложен новый адаптивный подход к приближенному поиску ближайших соседей (ANNS) с использованием инвертированного файлового индекса (IVF), в котором вычислительные ресурсы выделяются в зависимости от сложности каждого конкретного запроса. Ключевой идеей подхода является классификация сложности запроса по результатам предварительного поиска и определения количества просматриваемых кластеров *nprobe*. Эксперименты подтвердили эффективность алгоритма: при требуемом уровне полноты (Recall) выше 0,99 достигнуто увеличение скорости обработки запросов (QPS) до 28,93% и снижение средней задержки до 22,7% по сравнению со стандартным алгоритмом IVF-поиска. Алгоритм адаптивного ANN поиска демонстрирует высокую стабильность и масштабируемость на наборах данных до 10^9 объектов, не требуя модификации процесса построения индекса. Предложенное решение позволяет повысить производительность векторных баз данных в реальных системах, где критичны баланс между точностью, скоростью и ресурсозатратностью.

Однако потенциал усовершенствования ANN поиска не ограничивается улучшением процедуры поиска. Другой важной составляющей IVF является создание индекса, которое использует алгоритмы кластеризации.

Результаты главы опубликованы в [6], внедрена в эксплуатацию (Приложение А) программа для ЭВМ.

3 ЭВОЛЮЦИОННЫЕ АЛГОРИТМЫ КЛАСТЕРИЗАЦИИ ДЛЯ СОЗДАНИЯ IVF-ИНДЕКСА

Помимо свойств самого алгоритма приближенного поиска ближайших соседей по IVF-индексу, на скорость и точность поиска влияет качество построения индекса. Эффективность такого индекса определяется тем, насколько удачно исходные данные разделены на группы (кластеры), поскольку именно кластерная структура задает пространство поиска. Если объекты распределены по группам «неудачно», возрастает вероятность пропуска действительно близких соседей либо увеличивается число кластеров, которые необходимо просматривать, что приводит к снижению точности или росту времени поиска.

Помимо задачи приближенного поиска ближайших соседей задачи размещения и кластеризации возникают во многих областях науки и промышленности, таких как здравоохранение [119], промышленное производство [103, 120], сегментация рынка [121], социологические исследования [122] и во многих других областях науки и техники. Целью многих задач размещения и кластеризации является минимизация суммарного или максимального расстояния между заданными объектами и объектами, которые требуется найти.

Существует множество эвристических методов, разработанных для решения таких задач, и одним из классических примеров эвристических методов является использование эволюционных алгоритмов (ЭА) [123]. Принципиальное отличие таких методов от других алгоритмов оптимизации состоит в том, что ЭА работают с целым набором решений, называемым популяцией, последовательно улучшая заданную целевую функцию.

Первые эволюционные стратегии применялись в гидродинамике для оптимизации формы изогнутой трубы и аналогичных задач. Первоначально применение ЭА было весьма ограниченным [123], однако в настоящее время эволюционные алгоритмы наиболее часто используются для настройки других моделей, включая алгоритмы решения задачи p-медиан [124], многих других моделей кластеризации [125] и нейронных сетей [126]. Это позволяет не только

повышать качество модели, но и автоматически настраивать модели и сокращать временные затраты, которые в противном случае ушли бы на ручную настройку.

В настоящей главе рассматривается не только задача *k*-средних, но и другая задача, которая может быть использована для построения IVF-индекса: задача *p*-медиан. Задача *p*-медиан является одной из наиболее популярных и широко используемых моделей в теории размещения. В отличие от задачи *k*-средних, она минимизирует сумму расстояний от известных точек до искомых точек, называемых центрами или медианами. Эта NP-трудная задача размещения также используется при решении задач кластеризации (автоматической группировки данных). В этом случае искомые точки рассматриваются как центры кластеров. В отличие от близкой модели *k*-средних, кластеризация в модели *p*-медиан менее чувствительна к зашумленным данным и появлению выбросов (отдельно расположенных точек, не относящихся ни к одному кластеру) [124]. Задача *p*-медиан может быть решена специализированным алгоритмом обучения без учителя, таким как алгоритм *k*-средних (алгоритм 1.1).

Выбор алгоритма вручную и настройка его параметров являются сложными и иногда неэффективными. Это ведет к развитию методологий самонастраивающихся алгоритмов машинного обучения, таких как автоматическое машинное обучение (AutoML) [127, 128], а также методов глобального поиска [103, 129]. Принципиальная идея таких методов заключается в отсутствии необходимости в применении экспертных (выполненных человеком) оценок: машина самостоятельно настраивает модель или алгоритм и выбирает его гиперпараметры. Первая группа методов применяется тогда, когда неизвестно, какой алгоритм можно применить к конкретной задаче, например, когда требуется найти решение ранее не исследованной задачи или для неизученных данных [130]. С другой стороны, когда наиболее подходящий алгоритм известен, AutoML-система затрачивает дополнительное время на настройку и оценку других алгоритмов.

Формальное определение задачи *p*-медиан имеет следующий вид. Для заданного множества из n точек в непрерывном пространстве: $x_1, \dots, x_n$, где

$x_i = (x_{i,1}, \dots, x_{i,d})$, $x_i \in \mathbb{R}^d$, требуется найти множество $Y = \{y_1, \dots, y_p\} \subset \mathbb{R}^d$, называемое множеством медиан. Целевая функция имеет вид [131]:

$$f(y_1, \dots, y_p) = \sum_{i=1}^{n} \min_{j=1,\dots,p} d(x_i, y_j) \rightarrow \min_{y_1,\dots,y_p \in \mathbb{R}^d}.$$

Здесь p (целое число) и метрика расстояния $d(\cdot,\cdot)$ должны быть известны заранее. Наиболее широко используемой метрикой является расстояние Минковского:

$$d_q(y_j, x_i) = \left(\sum_{k=1}^{d} \left| y_{j,k} - x_{i,k} \right|^q \right)^{1/q}.$$

Значение $q = 1$ соответствует метрике городских кварталов (манхэттенской), а $q = 2$ - евклидовой метрике.

Процедура Ллойда является алгоритмом локального поиска, также известным как алгоритм k-средних, и используется как часть различных эволюционных алгоритмов. Она требует множества из p начальных медиан $Y = \{y_1, \dots, y_p\}$ и состоит из двух чередующихся шагов:

- Для каждого центра y_j, $j = 1, \dots, p$, найти подмножество $C_j \subset \{x_1, \dots, x_n\}$ точек спроса (кластеров), ближайший центр для которых совпадает и равен y_j.

- Для каждого кластера C_j, $j = 1, \dots, p$, заново вычислить положение его центра y_j:

$$y_j \leftarrow \operatorname*{argmin}_y \sum_{x \in C_j} d(y, x).$$

Эти два шага повторяются до тех пор, пока происходят какие-либо изменения [124]. Начальные центры часто выбираются случайно либо по специальной процедуре k-means++ [132].

Алгоритм $(1 + \lambda)$ является эволюционным алгоритмом оптимизации. Он минимизирует заданную целевую функцию φ: $\mathbb{R}^d \rightarrow \mathbb{R}$. На каждой итерации одно родительское решение порождает λ решений-потомков $\{z_1, \dots, z_\lambda\}$ посредством заданной процедуры мутации и выбирает наилучшее из них.

Жадная агломеративная эвристика также является эволюционным алгоритмом оптимизации [98]. В контексте задачи кластеризации идея жадного подхода состоит в последовательном исключении тех кластеров, исключение которых с последующим перераспределением объектов кластера по оставшимся кластерам приводит к наименьшему приросту целевой функции.

В Главе 3 предлагается новая эволюционная стратегия $(1 + \lambda)$ со специальным оператором мутации, аналогичным процедуре кроссовера (скрещивания), включающим жадную агломеративную процедуру [98] для задач кластеризации и размещения. Более простой эволюционный алгоритм $(1 + 1)$ с той же процедурой мутации был представлен в [124]. Такая процедура весьма чувствительна к параметру r, который определяет число центров, добавляемых процедурой мутации к промежуточному решению. В [124] этот параметр предварительно настраивался в ходе предварительного поиска. В ходе работы предлагаемого нового эволюционного алгоритма $(1 + \lambda)$ такой параметр динамически корректируется в соответствии с изменениями значения этого параметра, обеспечивающего наилучший результат на текущей итерации. Несмотря на применимость представленного эволюционного алгоритма для построения IVF-индекса для специфических задач ANN поиска среди эмбеддингов могут потребоваться более специализированные алгоритмы, поэтому в настоящей главе также предлагается особая ускоренная агломеративная процедура для кластеризации эмбеддингов, использующая так называемую «проблему края» для аппроксимации прироста целевой функции после исключения кластера.

3.1 Эволюционные алгоритмы оптимизации

Эволюционные алгоритмы оптимизации - это современные и активно исследуемые и развивающиеся алгоритмы глобального поиска, которые имитируют принципы естественного отбора и работают с целым набором (популяцией) решений-кандидатов (особей), последовательно улучшая

популяцию по значению целевой функции. Ключевую роль в создании новых решений играют две процедуры: мутация - случайное изменение параметров (генов) отдельного кандидата для внесения разнообразия и исследования новых областей пространства поиска, и кроссовер (скрещивание) - комбинирование частей двух (или более) кандидатов-родителей с целью получить потомка, наследующего признаки обоих и потенциально обладающего лучшими свойствами. В данном разделе кратко описываются современные исследования в этой области.

В [133] авторами предложен новый способ самонастройки интенсивности мутации в популяционных эволюционных алгоритмах для дискретных пространств поиска. Доэрр и соавторы анализируют, как алгоритм $(1+\lambda)$ с предложенной самонастраиваемой интенсивностью мутации оптимизирует специальную тестовую функцию. Идея метода заключается в том, чтобы формировать половину потомков с удвоенной текущей интенсивностью мутации, а вторую половину - с половинной текущей интенсивности. На следующей итерации используется то значение, которое использовалась при порождении лучшего из полученных потомков.

Оптимальная интенсивность мутации для специальной задачи OneMax исследуется в [134]. Задача OneMax (максимизация расстояния Хэмминга) является классической задачей для исследований эволюционных стратегий. Задача OneMax состоит в максимизации линейной функции, подсчитывающей число единиц в битовой строке:

$$\mathrm{OneMax}(x) = \sum_{i=1}^{n} x_i.$$

Использование этой задачи является стандартной практикой для оценки ЭА с бинарным кодированием решения. В работе [134] расширен анализ оптимальной интенсивности мутации для эволюционных алгоритмов $(1+\lambda)$, а также для $(1+\lambda)$ случайного локального поиска и соответствующих им интенсивностей мутации, максимизирующих различия между особями в популяции.

В [125] предложен эволюционный алгоритм для задачи кластеризации на основе мета-обучения. На каждом шаге тип мутации для алгоритма $(1+1)$ выбирается случайным образом из выбранного подмножества посредством алгоритма мета-обучения согласно динамически настраиваемому вектору вероятностей, который изменяется на каждой итерации. Успех каждой операции мутации определяется значением выбранной функции приспособленности: если качество потомка выше качества родительского решения, то мутация считается успешной. Авторы рассматривают 32 различных операции мутации, включая мутацию на основе прототипов, мутацию на основе минимального остовного дерева, мутации, не ориентированные на кластеры, и др.

В [98] не только предложена новая имитирующая кроссовер со случайно сгенерированной особью мутация для генетических алгоритмов, но и приведен краткий обзор операторов мутации для задачи p-медиан. Все особи кодируются хромосомой на основе центроидов. Классическая процедура мутации задается следующим образом: каждая хромосома мутирует с вероятностью мутации μ по правилу $v \leftarrow v \pm 2 \times b \times v,$ где $b \sim U(0,1)$ то есть мутация случайным образом смещает координаты центров кластеров.

Используемая в предложенном (1+λ) алгоритме процедура мутации устроена иначе. Так называемая «кроссовероподобная» мутация по сути является процедурой скрещивания со случайно сгенерированной особью и может быть формализована следующим образом:

Алгоритм 3.1 - Кроссовероподобная мутация

Шаг 1. Сгенерировать случайное начальное решение $S = \{c_1, \dots, c_N\}$;

Шаг 2. Запустить алгоритм k-средних для получения локально оптимального значения S;

Шаг 3. Применить выбранную процедуру скрещивания к мутирующей особи S' для получения нового решения S'';

Шаг 4. Использовать алгоритм k-средних для оптимизации сгенерированного решения S'';

Шаг 5. Если $F(S'') < F(S')$, то $S' \leftarrow S''$, где F — целевая функция.

Алгоритм 3.1 является весьма эффективным инструментом для генетических алгоритмов [98], ускоряющим оптимизацию целевой функции. Идея кроссовероподобной мутации предельно проста: для заданного (мутируемого) решения генерируется новое случайное решение, после чего выполняется операция скрещивания между исходным и новым решениями. Данная мутация оказывается эффективной при использовании жадного агломеративного оператора скрещивания. Идея такого скрещивания была предложена для дискретной задачи p-медиан в [102]. В [124, 135, 136] она была адаптирована для непрерывных задач k-средних и p-медиан. Жадная агломеративная процедура может быть описана следующим образом [124].

Алгоритм 3.2 - Классическая агломеративная эвристика (BasicAggl)

Дано: множество начальных центров $S = \{X_1, \dots, X_K\}$, $|S| = K > p$, требуемое конечное число центров p.

Шаг 1. Улучшить S с помощью алгоритма k-средних, если это возможно;

Шаг 2. Пока $|S| > p$, выполнять:

Шаг 2.1. Для $i = 1, \dots, |S|$ вычислить $F_i \leftarrow F(S \backslash \{X_i\})$;

Шаг 2.2. Выбрать подмножество центров $S' \subset S$ с минимальными значениями соответствующих величин F_i так, чтобы $|S'| = \max\{1, \lceil (|S| - p) \cdot 0.2 \rceil\}$;

Шаг 2.3. Улучшить новое решение S алгоритмом k-средних $S \leftarrow S \backslash S'$;

Шаг 3. Вернуть S.

Агломеративная процедура скрещивания для двух «родительских» решений описана в алгоритме 3.3.

Алгоритм 3.3 - $AGGL_r(S,S_2)$

Дано: два решения (множества центров) S и S_2, $|S| = |S_2| = p$; число центров r из решения S_2, используемых для получения результирующего решения, $r \in \{1, \dots, p\}$.

Шаг 1. Для $i = 1, \dots, \max\{1, \lfloor p/r \rfloor\}$ выполнять:

Шаг 1.1. Выбрать подмножество $S' \subset S_2$ такое, что $|S'| = r$;

Шаг 1.2. $S' \leftarrow \text{BasicAggl}(S \cup S')$;

Шаг 1.3. Если $F(S') < F(S)$, то $S \leftarrow S'$;

Шаг 2. Вернуть S.

Эффективность эволюционных алгоритмов, использующих подобную процедуру скрещивания, в существенной степени зависит от параметра r [124], причем данная зависимость носит непредсказуемый характер. Для настройки этого параметра в [124] авторы используют процедуру предварительного поиска в эволюционном алгоритме (1+1) и в результате получают более универсальный вычислительный инструмент для задачи p-медиан.

Однако значение параметра r, обеспечивающее наиболее быструю сходимость, изменяется в ходе процесса оптимизации. Для динамической настройки параметра r используется подход, близкий к идее самонастраивающегося эволюционного алгоритма $(1 + \lambda)$ с традиционным двоичным кодированием решений и инверсной мутацией, предложенного в [133]. В отличие от настройки интенсивности мутации, в предложенном в следующем разделе $(1 + \lambda)$ эволюционном алгоритме настраивается параметр r.

3.2 Новый (1+λ) эволюционный алгоритм

В $(1 + \lambda)$ алгоритмах на каждой итерации одно текущее решение используется для порождения λ новых решений посредством процедуры мутации. Из полученной популяции выбирается наилучшее решение, которое затем принимается в качестве нового текущего решения. Как правило, наиболее важным

параметром таких алгоритмов при двоичном кодировании решений является вероятность (скорость) мутации. В самонастраивающемся эволюционном алгоритме $(1+\lambda)$ [124] половина популяции (то есть $\lambda/2$ решений) генерируется с уменьшенной вероятностью мутации, а другая половина - с удвоенной вероятностью мутации.

Алгоритм 3.4 - $(1+\lambda)$ эволюционный алгоритм с жадной агломеративной кроссовероподобной мутацией

Дано: Набор объектов A, число центров k, количество создаваемых потомков на каждой итерации λ.

Шаг 1. Случайным образом создать подмножество $S \subset \{A_1,\ldots,A_N\}$, $A_i \in A$; $r \leftarrow [p/2]$; $S \leftarrow ALA(S)$.

Шаг 2. Если $r<p/4$, то генерируется λ решений потомков и переход к шагу 3, иначе генерируется только одно решение и переход к шагу 12.

Шаг 3. Для каждого i из {1,...,λ} выбрать случайное подмножество $i' \subset \{A_1,\ldots,A_N\}$; $S_{i'} \leftarrow ALA(S_{i'})$, если i≤[λ/2], то r' выбрать из $\overline{\left\{\max\left(1,\frac{r-1}{2}\right),r\right\}}$, иначе из $\overline{\{r+1,[2(r+1)]\}}$; $S_i' \leftarrow AGGLr'(S,S_{i'})$.

Шаг 4. Определить i, обеспечивающее минимальное значение $F(S_i')$.

Шаг 5. Если $F(S_i') < F(S)$ и i≤[λ/2], то $r \leftarrow \max\left\{1,\frac{r}{1.5}\right\}$.

Шаг 6. Если $F(S_i') < F(S)$и i>[λ/2], то $r \leftarrow \min\left\{\left[\frac{k}{2}\right],1.5r\right\}$.

Шаг 7. Если $F(S_i') > F(S)$ и i≤[λ/2], то $r \leftarrow \max\left\{1,\frac{r}{1.25}\right\}$.

Шаг 8. Если $F(S_i') > F(S)$ и i>[λ/2], то $r \leftarrow \max\{1,1.25r\}$.

Шаг 9. Если r=1, то с вероятностью 0.1 выполнить $r \leftarrow \left[\frac{p}{2}\right]$.

Шаг 10. Если $r=[p/2]$, то с вероятностью 0.1 выполнить $r \leftarrow 1$.

Шаг 11. Перейти к Шагу 15.

Шаг 12. Случайным образом определить $S \subset \{A_1,\ldots,A_N\}$; $S' \leftarrow ALA(S')$.

Шаг 13. Случайным образом определить r' из $\overline{\left\{\max\{1, \left[\frac{r-1}{1.5}\right], [1.5(r+1)]\}\right\}}$; $S \leftarrow AGGLr'(S, S')$.

Шаг 14. Если $F(S') < F(S)$, то $S \leftarrow S'$.

Шаг 15. Если не исчерпано доступное на выполнение время, то вернуться к Шагу 2.

Шаг 16. ОСТАНОВ.

Новый коэффициент мутации выбирается в зависимости от подмножества потомков (субпопуляции), в котором находится наилучшее найденное решение. Аналогичная идея используется и для эволюционного алгоритма с вещественным кодированием (при этом решения представляют собой множества центров) и кроссовероподобной мутацией, реализуемой жадным агломеративным оператором скрещивания (см. алгоритм 3.1).

Динамически настраиваемым параметром в алгоритме 3.4 схеме является параметр *r*. На каждой итерации алгоритм формирует две субпопуляции потомков: одну - с уменьшенным значением *r*, и вторую - с увеличенным значением *r*. Затем значение *r* корректируется в соответствии с тем, в какой из субпопуляций расположен наилучший потомок. Если *r*=1, то с некоторой малой вероятностью его значение может быть заменено на новое значение [*p*/2]. Аналогично, если *r*=[*p*/2], то с некоторой малой вероятностью вместо него может быть установлено значение 1. Если же выполняется неравенство 1<*r*<*p*/2 то изменение значения *r* носит менее существенный характер. Итерации, на которых производится изменение значения *r*, комбинируются с более простыми итерациями, в рамках которых порождается единственный потомок.

Проведенные вычислительные эксперименты демонстрируют преимущество предложенного нового алгоритма.

3.3 Экспериментальная оценка эффективности (1+λ) эволюционного алгоритма кластеризации

В экспериментах использовались наборы данных из репозиториев UCI Machine Learning и Clustering basic benchmark repositories [137–139]: (a) *BIRCH3*: искусственно сгенерированный набор данных, содержащий 100 групп точек случайного размера, всего 100,000 точек в $\mathbb{R}^2$; (b) *Mopsi-Joensuu*: географические координаты пользователей (6014 точек, 2 измерения) в г. Йоэнсуу; (c) *Mopsi-Finland*: географические координаты пользователей (13467 точек, 2 измерения) на территории Финляндии; (d) крупный набор данных *Individual Household Electric Power Consumption (IHEPC)*: данные энергопотребления домохозяйств, более 2 млн. точек спроса (векторов данных) в $\mathbb{R}^7$, при этом данные нормированы и все измерения находятся в диапазоне [0,1].

Для вычислительных экспериментов использовалась следующая конфигурация ЭВМ: процессор Intel Core 2 Duo E8400, 16 ГБ ОЗУ, графический процессор NVIDIA GeForce GTX1050ti с 4096 MB ОЗУ, производительность операций с плавающей точкой 2138 GFLOPS. Программный код реализован на языке C++. Использовались компилятор Visual C++ 2017, встроенный в Visual Studio v. 15.9.5, NVIDIA CUDA 10.0 Wizards, а также NVIDIA Nsight Visual Studio Edition CUDA Support v. 6.0.0. Для всех наборов данных для каждого из алгоритмов выполнялось по 30 запусков.

Выполнено по 30 запусков для следующих алгоритмов (см. таблицы 3.1–3.8): (a) *Lloyd*: мультистарт алгоритма *k*-средних; (b) *j-means*: мультистарт *j*-means алгоритма (регулярный поиск в окрестности SWAP1 в сочетании с алгоритмом *k*-средних); (c) $AGGL_r$: рандомизированный поиск в окрестности $\mathrm{AGGL_r}$ (см. [124]), $r = \overline{1,p}$; (d) $SWAP_r$: локальный поиск в окрестностях $\mathrm{SWAP_r}$, $r = \overline{1,p}$ (приведен только лучший результат для $\overline{1,p}$); (e–g) *GH-VNS1*, *GH-VNS2*, *GH-VNS3*: алгоритмы Variable Neighborhood Search, в которых окрестности формируются применением процедуры $\mathrm{AGGL}_r(\cdot)$ [140, 141]; (h) *GA-1POINT*: генетический алгоритм с решениями в вещественном кодировании и стандартным

одноточечным кроссовером; (i) *GA-UNIFORM*: то же, что *GA-1POINT*, но с равномерной случайной мутацией [142, 143]; (j) *Aggl* $(1+1)$ -ЭА: самонастраивающийся эволюционный алгоритм $(1+1)$ с кроссовероподобной мутацией на основе жадного агломеративного оператора [124]; (k) $(1+\lambda)$-ЭА$_k$: предложенный новый алгоритм, $k=\lambda \in \{2,4,8\}$.

Таблица 3.1 - Сравнительные результаты для набора данных BIRCH3: 10^5 векторов данных в $\mathbb{R}^2$, $k=100$ центров, ограничение по времени 10 с

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	2.44280×10^9	2.46163×10^9	7.42330×10^7
j-means (SWAP1 + Lloyd)	1.69226×10^9	1.67712×10^9	8.03684×10^7
AGGL$_2$	1.49822×10^9	1.49445×10^9	1.51721×10^7
AGGL$_5$	1.50433×10^9	1.49439×10^9	2.44200×10^7
GH-VNS1	1.54540×10^9	1.51469×10^9	5.07461×10^7
GH-VNS2	1.50692×10^9	1.50235×10^9	1.87495×10^7
GH-VNS3	1.50561×10^9	1.49497×10^9	1.84421×10^7
SWAP$_1$	1.83218×10^9	1.83832×10^9	6.04418×10^7
GA-1POINT	1.69042×10^9	1.65346×10^9	7.23930×10^7
GA-UNIFORM	1.77471×10^9	1.76783×10^9	6.95451×10^7
Aggl$(1+1)$-ЭА	1.50670×10^9	1.49495×10^9	2.46374×10^7
$(1+\lambda)$-ЭА, $\lambda=2 \leftrightarrow \Uparrow$	$\underline{1.49353 \times 10^9}$	$\underline{1.49344 \times 10^9}$	$\underline{2.17671 \times 10^5}$
$(1+\lambda)$-ЭА, $\lambda=4$	1.49366×10^9	1.49346×10^9	5.45692×10^5
$(1+\lambda)$-ЭА, $\lambda=8$	1.49376×10^9	1.49355×10^9	7.49686×10^5

Наилучшие результаты во всех таблицах подчеркнуты. Статистическая значимость преимущества нового алгоритма оценивается с использованием *t*-критерия (результаты помечены стрелками) и критерия Манна–Уитни–Уилкоксона (двойные стрелки) при уровне значимости 0.05 для обоих тестов. $\uparrow/\Uparrow$: преимущество нового алгоритма статистически значимо, $\downarrow/\Downarrow$: статистически значим недостаток нового алгоритма, $\leftrightarrow/\Leftrightarrow$: преимущество или недостаток статистически незначимы.

Таблица 3.2 - Сравнительные результаты для набора данных BIRCH3: 10^5 векторов данных в $\mathbb{R}^2$, $k = 300$ центров, ограничение по времени 10 с

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	1.59859×10^9	1.60045×10^9	2.62443×10^7
j-means ($SWAP_1$ + Lloyd)	1.05927×10^9	1.04561×10^9	4.63788×10^7
$AGGL_{250}$	9.36947×10^8	9.29497×10^8	1.97893×10^7
$AGGL_{300}$	9.39030×10^8	9.39434×10^8	1.27907×10^7
GH-VNS1	1.03856×10^9	1.03485×10^9	2.20294×10^7
GH-VNS2	1.01461×10^9	1.00404×10^9	3.80607×10^7
GH-VNS3	9.31225×10^8	9.32001×10^8	8.25653×10^6
$SWAP_2$	1.33987×10^9	1.35305×10^9	5.64094×10^7
$SWAP_3$	1.34388×10^9	1.34424×10^9	5.01104×10^7
GA-1POINT	1.23404×10^9	1.22828×10^9	7.05936×10^7
GA-UNIFORM	1.26758×10^9	1.25424×10^9	5.20153×10^7
Aggl $(1+1)$-ЭА	9.34535×10^8	9.33403×10^8	1.51920×10^7
$(1+\lambda)$-ЭА, $\lambda = 2$ ↑⇑	$\underline{9.15236 \times 10^8}$	$\underline{9.13237 \times 10^8}$	7.53982×10^6
$(1+\lambda)$-ЭА, $\lambda = 4$	9.16937×10^8	9.14678×10^8	$\underline{7.51483 \times 10^6}$
$(1+\lambda)$-ЭА, $\lambda = 8$	9.23190×10^8	9.23817×10^8	7.64460×10^6

Таблица 3.3 - Сравнительные результаты для набора данных Mopsi-Joensuu: 6014 векторов данных в $\mathbb{R}^2$, $k = 100$ центров, ограничение по времени 5 с

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	110.4963	109.9706	5.6551
j-means ($SWAP_1$ + Lloyd)	51.6673	51.1146	2.0749
$AGGL_{75}$	44.1450	43.9860	0.4683
$AGGL_{100}$	44.0472	43.9875	0.3061
GH-VNS1	53.4896	53.1948	3.4123
GH-VNS2	44.8602	44.9149	0.5772
GH-VNS3	44.7263	44.6593	0.6018
$SWAP_1$	49.8287	49.5523	1.1239
GA-1POINT	67.2297	66.9182	3.8341
GA-UNIFORM	84.8705	83.8152	7.4129
Aggl $(1+1)$-ЭА	43.7486	43.7560	0.1624
$(1+\lambda)$-ЭА, $\lambda = 2$ ↑⇑	<u>43.6586</u>	<u>43.6481</u>	<u>0.0673</u>
$(1+\lambda)$-ЭА, $\lambda = 4$	43.7481	43.6724	0.1782
$(1+\lambda)$-ЭА, $\lambda = 8$	43.7641	43.7182	0.1856

Таблица 3.4 - Сравнительные результаты для набора данных Mopsi-Joensuu: 6014 векторов данных в $\mathbb{R}^2$, $k = 300$ центров, ограничение по времени 5 с

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	51.6395	51.1162	1.6453
j-means ($SWAP_1$ + Lloyd)	31.2554	31.8670	2.7425
$AGGL_{100}$	15.6951	15.6738	0.3081
GH-VNS1	27.7853	27.9011	2.5876
GH-VNS2	16.6424	16.6345	0.5166
GH-VNS3	15.9644	15.8850	0.4229
$SWAP_5$	27.5174	27.9711	1.9760
$SWAP_7$	27.6329	27.8934	1.9714
GA-1POINT	40.1539	39.6896	2.3970
GA-UNIFORM	43.8100	43.7275	3.7585
Aggl $(1+1)$-ЭА	15.4576	15.3310	0.8131
$(1+\lambda)$-ЭА, $\lambda = 2$	15.0468	15.0868	<u>0.1142</u>
$(1+\lambda)$-ЭА, $\lambda = 4$	14.9670	14.9857	0.1248
$(1+\lambda)$-ЭА, $\lambda = 8$ ↑⇑	<u>14.9530</u>	<u>14.8999</u>	0.1753

Таблица 3.5 - Сравнительные результаты для набора данных IHEPC: 2,075,259 векторов данных в $\mathbb{R}^7$, $k = 100$ центров, ограничение по времени 5 мин

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	64612.3371	65037.8281	1431.8760
j-means ($SWAP_1$ + Lloyd)	63433.8962	62227.1172	3308.9649
$AGGL_5$	69386.2210	59046.0703	13415.1144
GH-VNS1	73899.8482	83708.7031	13239.8092
GH-VNS2	83061.5056	83033.9844	<u>128.9717</u>
GH-VNS3	78905.5815	82124.6094	8593.7433
$SWAP_{10}$	70051.5938	62676.3125	10652.0083
GA-1POINT	<u>63051.7500</u>	63028.9922	733.3966
GA-UNIFORM	63656.8555	64155.6875	2084.6202
Aggl $(1+1)$-ЭА↓⇓	65782.6094	<u>58506.6484</u>	3208.6879
$(1+\lambda)$-ЭА, $\lambda = 2$	72226.2400	82464.4844	12905.4011
$(1+\lambda)$-ЭА, $\lambda = 4$	72327.9403	82547.3203	12849.0133
$(1+\lambda)$-ЭА, $\lambda = 8$	72455.6819	82540.6641	12679.5718

Таблица 3.6 - Сравнительные результаты для набора данных Individual Household Electric Power Consumption (IHEPC). 2 075 259 векторов данных в $\mathbb{R}^7$, $k = 300$ центров, ограничение времени 5 мин

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	53113.4275	52599.0273	2781.6693
j-means ($SWAP_1$ + Lloyd)	<u>45375.1657</u>	44608.1953	1505.4071
$AGGL_{15}$	57406.8198	66625.8047	12069.9982
$AGGL_{300}$	61283.1027	64492.8828	8781.8445
GH-VNS1	67645.6563	67467.4375	478.1909
GH-VNS2	63081.6233	66222.0313	8689.9374
GH-VNS3	61204.4487	64358.4844	8413.3478
$SWAP_3$	63618.6362	68361.8672	9467.2699
$SWAP_{100}$	65787.6618	67617.5781	6667.2423
GA-1POINT	49198.6964	47304.4453	3129.2994
GA-UNIFORM	52218.1077	50642.4961	5712.0544
Aggl $(1+1)$-ЭА	51888.0642	<u>42525.5000</u>	12122.2332
$(1+\lambda)$-ЭА, $\lambda = 2$ $\leftrightarrow\Leftrightarrow$	52601.9637	43008.0117	12322.3127
$(1+\lambda)$-ЭА, $\lambda = 4$	64539.6836	64481.6563	<u>147.6363</u>
$(1+\lambda)$-ЭА, $\lambda = 8$	61332.9838	64502.6328	8572.1015

Таблица 3.7 - Сравнительные результаты для набора данных Mopsi-Finland. 13 467 векторов данных в $\mathbb{R}^2$, $k = 100$ кластеров, ограничение времени 5 с

Алгоритм или окрестность	Ср. ЦФ	Медиан. ЦФ	Ср.кв.откл.ЦФ
Lloyd (multistart)	6.92349×10^6	6.95864×10^6	198384
j-means (SWAP1 + Lloyd)	3.85266×10^6	3.84859×10^6	59481.8
$AGGL_{25}$	3.64913×10^6	3.64784×10^6	5184.35
$AGGL_{30}$	3.65106×10^6	3.64702×10^6	7780.79
GH-VNS1	4.05038×10^6	3.93463×10^6	336130
GH-VNS2	3.65123×10^6	3.65015×10^6	6284.54
GH-VNS3	3.65735×10^6	3.65370×10^6	9247.64
$SWAP_1$	3.74321×10^6	3.74447×10^6	25846.1
GA-1POINT	4.83383×10^6	4.82569×10^6	198278
GA-UNIFORM	5.62017×10^6	5.60439×10^6	347120
Aggl $(1+1)$-ЭА	3.64473×10^6	3.64502×10^6	1467.87
$(1+\lambda)$-ЭА, $\lambda = 2$	3.64585×10^6	3.64538×10^6	2752.36
$(1+\lambda)$-ЭА, $\lambda = 4$ $\leftrightarrow\Leftrightarrow$	3.64474×10^6	3.64501×10^6	751.420
$(1+\lambda)$-ЭА, $\lambda = 8$	3.64505×10^6	3.64507×10^6	468.047

Таблица 3.8 - Сравнительные результаты для набора данных Mopsi-Finland. 13 467 векторов данных в $\mathbb{R}^2$, $k = 300$ кластеров, ограничение времени 5 с

Алгоритм или окрестность	Среднее	Медиана	Стд. откл.
Lloyd (multistart)	3.46203×10^6	3.47443×10^6	80996.0
j-means ($SWAP_1$ + Lloyd)	1.74473×10^6	1.71790×10^6	120717
$AGGL_{300}$	1.38198×10^6	1.37998×10^6	15384.2
GH-VNS1	1.87817×10^6	1.88773×10^6	101777
GH-VNS2	1.51093×10^6	1.49528×10^6	70431.8
GH-VNS3	1.38037×10^6	1.37633×10^6	14603.0
$SWAP_1$	1.57134×10^6	1.56292×10^6	47997.3
GA-1POINT	2.91834×10^6	2.88716×10^6	144687
GA-UNIFORM	2.91604×10^6	2.89257×10^6	205946
Aggl $(1+1)$-ЭА	1.38311×10^6	1.38149×10^6	12302.2
$(1+\lambda)$-ЭА, $\lambda = 2$	1.37152×10^6	1.36742×10^6	13363.3
$(1+\lambda)$-ЭА, $\lambda = 4$ ↑⇑	1.36187×10^6	$\underline{1.35671 \times 10^6}$	12143.7
$(1+\lambda)$-ЭА, $\lambda = 8$ ↑⇑	$\underline{1.35982 \times 10^6}$	1.35954×10^6	$\underline{7635.7}$

Из представленных таблиц следует, что новые алгоритмы обеспечивают существенно более качественные решения в большинстве рассмотренных случаев, но для наибольшего набора данных IHEPC указанные преимущества новых алгоритмов не проявились. Возможно, это связано с высокой вычислительной сложностью алгоритма (1+λ).

Выбор временного ограничения может играть существенную роль: алгоритм, демонстрирующий наилучшие результаты на начальных этапах поиска, впоследствии может прекратить улучшение значения целевой функции, тогда как другие алгоритмы продолжают поиск и достигают дальнейшего снижения целевой функции. Вместе с тем, алгоритм 3.4 демонстрирует сравнительное преимущество в широких интервалах времени (рисунки 3.1–3.4) на различных задачах, в частности на задачах географического размещения (Mopsi-Joensuu и Mopsi-Finland, рисунки 3.2–3.3), а также на широко используемой тестовой задаче BIRCH3 (рисунок 3.4).

Для наиболее крупной задачи IHEPC (рисунок 3.1) при числе точек данных 2075259 преимущество нового алгоритма обнаружено не было. В случае поиска 300 центров различие между новым алгоритмом и более простым эволюционным алгоритмом $(1+1)$ является несущественным. Для той же задачи при поиске 100

центров более простой алгоритм Aggl $(1+1)$-ЭА, использующий иной метод настройки параметров, превосходит новый алгоритм.

Тем не менее, несмотря на то, что в ряде случаев известные алгоритмы превосходят новый, предложенный подход остается конкурентоспособным при решении практических задач.

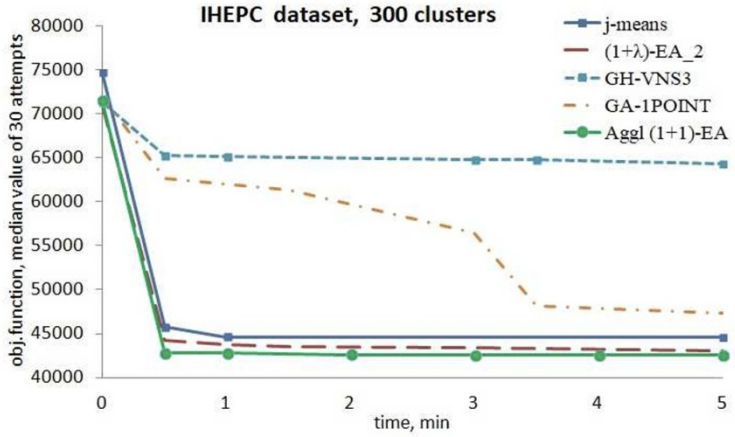


Рисунок 3.1 – Сравнительный анализ скорости сходимости: зависимость медианного результата от времени вычислений для набора данных IHEPC, 300 кластеров, 2 075 259 векторов данных, ограничение времени 300 с

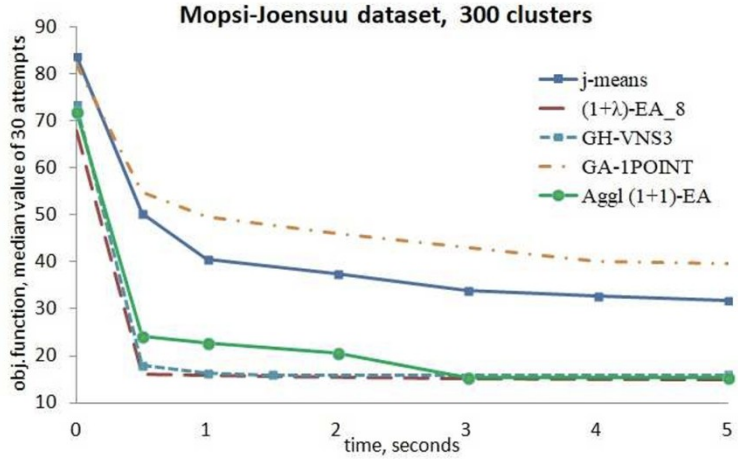


Рисунок 3.2 – Сравнительный анализ скорости сходимости: зависимость медианного результата от времени вычислений для набора данных Mopsi-Joensuu, поиск 300 центров, 6014 векторов данных, ограничение времени 5 с

Пример динамики изменения параметра r в процессе работы алгоритма представлен на рисунке 3.5. Видно, что на начальном этапе значение параметра постепенно уменьшается; при возникновении стагнации алгоритма параметр резко изменяется от 1 до $p/2$=150, что приводит к дальнейшему улучшению значений целевой функции $F(S)$.

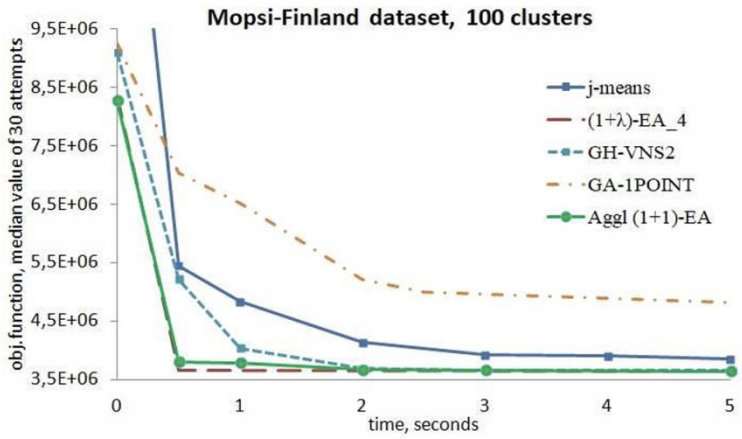


Рисунок 3.3 – Сравнительный анализ скорости сходимости: зависимость медианного результата от времени вычислений для набора данных Mopsi-Finland, поиск 100 центров, 13 467 векторов данных, ограничение времени 5 с

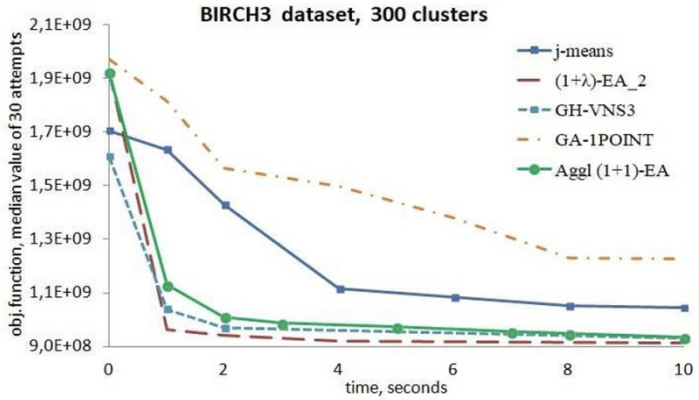


Рисунок 3.4 – Сравнительный анализ скорости сходимости: зависимость медианного результата от времени вычислений для набора данных BIRCH3, поиск 300 центров, 100 000 векторов данных, ограничение времени 10 с

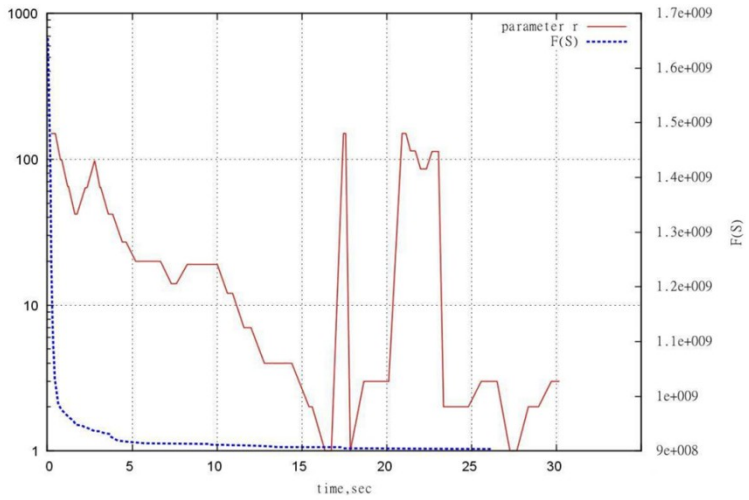


Рисунок 3.5 – Зависимость значения параметра r и результата $F(S)$ от времени вычислений для набора данных BIRCH3, поиск 300 центров, 100 000 векторов данных, $\lambda = 4$

Эксперименты подтверждают преимущество алгоритма 3.4 на большинстве тестов. Полученные результаты указывают на потенциальную применимость подхода к задачам типа k-средних и, следовательно, к задаче построения IVF-индекса. Однако результаты экспериментов, представленные в таблицах 3.1–3.8, также указывают на существенный недостаток эволюционных алгоритмов со сложными процедурами мутации и скрещивания: они вычислительно дороги.

3.5 Быстрая агломеративная процедура для создания IVF-индекса на наборе эмбеддингов

Несмотря на то, что предложенный в предыдущих разделах главы алгоритм показал сравнительно хорошие результаты, его использование может быть нецелесообразным для определенного класса задач. В пространствах данных высокой размерности (в частности, при работе с эмбеддингами) как внутрикластерные, так и межкластерные расстояния демонстрируют низкую дисперсию (рисунок 2.6). Это явление, следующее из проклятия размерности, приводит к уплощению распределений расстояний: различие между ближайшими и наиболее удаленными соседями становится статистически незначимым, а

подавляющее большинство векторов (до 99 % всего кластера) располагается на границах кластеров. Вследствие этого можно предположить, что приращение целевой функции алгоритма *k*-средних при удалении кластера и перераспределении его объектов к другим центроидам, в значительной степени определяется мощностью (размером) удаляемого кластера.

На основе данной гипотезы была разработана специализированная агломеративная процедура, предназначенная для автоматической группировки объектов в пространствах высокой размерности. Данная процедура существенно ускоряет построение IVF-индексов по сравнению с классическими агломеративными подходами: вместо непосредственного вычисления приращения целевой функции предложенный метод исходит из предположения, что прирост целевой функции пропорционален мощности удаляемого кластера (см. алгоритм 3.5). Такая аппроксимация позволяет применять агломеративные подходы к большим наборам данных, для которых указанные методы обычно неприменимы вследствие их высокой вычислительной сложности.

Алгоритм 3.5 - Ускоренная агломеративная процедура (AGGL) для кластеризации эмбеддингов

Дано: Исходное решение задачи кластеризации S с начальным числом центроидов k_0, требуемое число кластеров и центроидов k, $k_0>k$, число кластеров, удаляемое за одну итерацию k_e.

Шаг 1. Выполнить алгоритм Ллойда с k_0 кластерами s: $S \leftarrow k$-means (S).

Шаг 2. $k' \leftarrow k_0$.

Шаг 3. $k' \leftarrow \begin{cases} k' - k_e, & k - k_e \geq k \\ k, & k - k_e < k \end{cases}$;

Шаг 4. Удалить k' центроидов, соответствующих кластерам наименьшей мощности, перераспределить векторы данных по центроидам (переформировать кластеры).

Шаг 5.Повторять Шаги 3-4 пока $k' \neq k$.

Несмотря на концептуальную простоту предложенной процедуры, данный подход повышает качество построенного инвертированного файлового (IVF) индекса, что непосредственно приводит к улучшению эффективности поиска.

Очевидно, что применение данного подхода в сравнении с простейшей процедурой Ллойда существенно увеличивает время, необходимое для построения индекса, из-за увеличения времени работы алгоритма кластеризации. алгоритм 3.5 начинает работу с избыточным числом кластеров, которое постепенно уменьшается до достижения требуемого значения k. Такое итеративное сокращение непосредственно влияет на общую продолжительность построения индекса, делая его создание более трудоемким по сравнению с более простыми альтернативами.

Важно подчеркнуть, что время построения IVF-индекса не влияет на скорость или точность поиска ближайших соседей, так как весь индекс вместе со всеми связанными кодовыми книгами формируется единожды, их формирование происходит до выполнения любых поисковых операций. После построения эти компоненты остаются статичными, что позволяет быстро выполнять запросы без обращения к фазе построения.

3.6 Экспериментальное исследование эффективности разработанной быстрой агломеративной процедуры

Для описанных в настоящем разделе экспериментов использовался тот же набор данных SIFT1B, что и в разделе 2.4, а также выборки объектов SIFT1B, такие как SIFT100K и SIFT10M. Конфигурация вычислительной системы также аналогична ЭВМ, использовавшейся для проведения экспериментов из раздела 2.4.

В таблице 3.9 представлен набор экспериментов, проведенных с целью оценки метрик производительности. В ходе экспериментов было обеспечено получение сопоставимых значений Recall@K для различных конфигураций, что позволило сравнить производительности алгоритмов приближенного поиска

ближайших соседей. Параметр *nprobe* задает число ближайших кластеров, в пределах которых выполняется поиск ближайших соседей.

Таблица 3.9 - Результаты ANN поиска на основе IVF-индекса, построенного с использованием алгоритмов *k*-средних с многократным запуском (k-means) и алгоритма 3.5 (AgglFast). Набор данных SIFT100K, 10^5 векторов, 1024 кластера

№	k-means, *nprobe*	k-means, Recall	k-means, QPS	AgglFast, *nprobe*	AgglFast, Recall	AgglFast, QPS
1	80	98.50	788.13	95	99.01	648.78
2	100	99.15	608.88	95	98.08	673.87
3	95	99.07	637.05	98	99.09	658.08
4	97	99.08	623.13	98	99.09	634.91
5	100	99.17	532.84	98	99.09	666.27
6	100	99.17	549.07	98	99.09	666.47
7	98	99.09	634.87	98	99.09	656.05
8	98	99.09	685.52	98	99.09	667.73
9	98	99.10	582.58	98	99.10	648.40
10	98	99.10	652.46	98	99.10	666.62

В целом, результаты показывают, что агломеративный подход обеспечивает более стабильную и быструю работу алгоритма ANN поиска, в то время как многократный запуск алгоритма *k*-средних имеет более низкие значения QPS при аналогичных значениях *nprobe*. Это говорит о том, что жадная агломеративная процедура более эффективна для решения задач ANN поиска, особенно в задачах с высоким требуемым значением Recall.

В таблице 3.10 представлены результаты экспериментов с более высоким значением *nprobe*.

Как показано в таблице 3.10, новый агломеративный метод обеспечивает стабильные и адекватные результаты при большем количестве кластеров, чем в таблице 3.9. Это улучшение является значительным, поскольку оно решает проблемы масштабируемости, что приводит к более надежной кластеризации в больших наборах данных.

Таблица 3.10 - Результаты запуска алгоритма 3.5

№	*nprobe*	Recall	QPS	Время отклика	Значение целевой функции *k*-средних
1	125	99.23	340.50	290.62	13676465345
2	125	99.23	347.93	291.97	13676465345
3	125	99.21	360.25	285.36	13657293658
4	125	99.21	338.36	276.24	13657293658
5	125	99.20	267.98	294.07	13657414617
6	125	99.21	259.40	371.36	13655503947
7	125	99.21	283.83	383.09	13655503947
8	125	99.21	280.64	349.78	13668387005
9	125	99.21	273.15	353.68	13668387005
10	125	99.21	326.34	304.62	13673237378

В таблице 3.11 представлены сводные данные о полученных результатах в виде набора статистических показателей.

Таблица 3.11 – Усредненные значения полученных результатов поиска по IVF-индексу, построенному при помощи алгоритмов *k*-средних с многократным запуском (k-means) и алгоритма 3.5 (AgglFast)

	k-means, SIFT100K, 1024 clusters, *nprobe* = 98	AgglFast, SIFT100K, 1024 clusters, *nprobe* = 98	AgglFast, SIFT10M, 3150 clusters, *nprobe* = 125
Recall@100, среднее	99.08	99.09	99.21
Recall@100, медианное	99.09	99.09	99.21
Recall@100, ср.кв.откл.	0.0196	0.0042	0.0095
QPS, среднее	640.11	658.95	307.84
QPS, медианное	641.59	661.99	305.09
QPS, ср.кв.откл.	27.12	10.44	38.22
Время отклика, среднее	155.03	151.03	327.36
Время отклика, медианное	155.12	150.32	327.2
Время отклика, ср.кв.откл.	5.40	2.46	40.55

Как показывают представленные в таблице 3.11 результаты, новая жадная агломеративная процедура позволяет повысить точность ANN поиска. В частности, при поиске в одинаковом количестве кластеров (*nprobe*) этот подход дает более высокое среднее значение Recall, что указывает на большую долю релевантных результатов по сравнению с традиционным методом (многократный

запуск процедуры *k*-средних). Это улучшение обусловлено способностью процедуры более эффективно оптимизировать назначение кластеров, уменьшая ошибки в приближениях ближайших соседей и обеспечивая более точные совпадения запросов без чрезмерных вычислительных ресурсов.

Кроме того, процедура не только повышает точность, но и ускоряет общий процесс поиска, что подтверждается увеличением количества запросов в секунду (QPS) и уменьшением времени отклика. Помимо этих преимуществ в скорости и точности, метод обеспечивает значительно большую стабильность результатов производительности, проявляющуюся в стабильном качестве поиска и времени выполнения на различных наборах данных и при различной нагрузке запросов. Такая надежность особенно ценна в реальных условиях, где колебания производительности могут подорвать доверие пользователей и эффективность системы.

3.7 Результаты главы 3

В данной главе исследованы эволюционные алгоритмы оптимизации для задач размещения и кластеризации, в частности для NP-трудных задач *p*-медиан и *k*-средних. Предложен новый самонастраивающийся эволюционный алгоритм (1+λ) с динамической адаптацией параметра *r*, определяющего количество центров, добавляемых в ходе кроссовероподобной мутации на основе жадной агломеративной процедуры. Алгоритм демонстрирует статистически значимое преимущество по качеству решений на большинстве тестовых наборов данных по сравнению с классическими методами и ранее предложенным эволюционным алгоритмом Aggl (1+1)-ЭА. Динамическая настройка параметра *r* позволяет эффективно преодолевать стагнацию поиска, что подтверждается анализом сходимости.

Для пространств высокой размерности (эмбеддингов) разработана ускоренная агломеративная процедура, которая аппроксимирует прирост целевой функции при удалении кластера его мощностью, что существенно снижает

вычислительную сложность. Эксперименты на наборе данных SIFT показывают, что построение IVF-индекса с использованием данной процедуры обеспечивает более высокую точность поиска (Recall@100) и производительность (QPS) по сравнению с классическим многократным запуском *k*-средних при сопоставимых значениях *nprobe*. Метод демонстрирует лучшую стабильность и масштабируемость, что делает его перспективным для построения эффективных индексов в векторных базах данных.

Результаты главы опубликованы в [106], зарегистрирована [144] и внедрена в эксплуатацию (Приложение Б) программа для ЭВМ.

Представленные алгоритмы расширяют инструментарий методов для решения задач кластеризации и оптимизации размещения, предлагая эффективные механизмы адаптации параметров и специализированные процедуры для работы с данными высокой размерности, но область их применимости ограничена унимодальными данными, то есть такими данными, где каждый объект представлен только одной модальностью, например, только изображением или только текстом. Для расширения области применимости разработанных в этой и предыдущих главах алгоритмов в следующей главе представлена мера расстояния, агрегирующая расстояния между объектами, рассчитанные по отдельным модальностям и позволяющая применять алгоритмы кластеризации и приближенного поиска ближайших соседей к мультимодальным данным.

4 МОДЕЛЬ КЛАСТЕРИЗАЦИИ МУЛЬТИМОДАЛЬНЫХ ДАННЫХ

Решение задач кластеризации, используемой для построения IVF-индекса, нередко сопряжено с исследовательскими задачами. Даже при наличии достаточных вычислительных ресурсов для перебора всех возможных конфигураций алгоритмов кластеризации сохраняется неопределенность в части оценки получаемых результатов. Выбор подходящего алгоритма кластеризации в существенной степени требует экспертных знаний и зависит от множества факторов. Поскольку данный процесс преимущественно выполняется вручную, часто допускаются упущения, что делает принятие решений трудоемким и затратным по времени [145]. Существующие модели кластеризации зачастую требуют времени на обучение, на настройку параметров модели, а исследование, направленное на выбор и настройку сложной модели, может занимать значительное время.

В настоящей главе предлагается модель кластеризации мультимодальных данных, основанная на специальной мере расстояния, агрегирующей расстояния между объектами, рассчитанные по отдельным модальностям. Отличительной особенностью предлагаемой модели кластеризации является то, что она не требует создания единого векторного пространства, то есть объединения представлений модальностей в общее семантическое подпространство. Были модифицированы несколько алгоритмов кластеризации и выполнена их реализация на Apache Spark для запуска алгоритмов на распределенных вычислительных системах (кластерах компьютеров). Для оценки применимости представленной меры расстояния на Apache Spark были реализованы методы автоматического машинного обучения (AutoML), предназначенные для выбора и настройки наиболее подходящего алгоритма кластеризации и выбора меры качества кластеризации.

4.1 Используемые AutoML-методы для моделей обучения без учителя

В работе [128] представлен алгоритм, предназначенный для выбора и настройки алгоритмов обучения без учителя. Алгоритм MASSCAH функционирует следующим образом.

Задан набор алгоритмов машинного обучения, при этом каждому алгоритму сопоставлено пространство его гиперпараметров. Требуется за фиксированное время найти алгоритм, оптимальный с точки зрения заданной меры качества кластеризации. Если распределять бюджет времени настройки между различными алгоритмами кластеризации, то существенная доля времени может быть израсходована на настройку потенциально неэффективных алгоритмов. С другой стороны, приоритетная настройка одного алгоритма без учета остальных может привести к потере информации о качестве работы других алгоритмов кластеризации, которые потенциально способны обеспечить более качественное разбиение. Следовательно, требовалось разработать алгоритм, учитывающий компромисс между двумя описанными крайностями. Такой компромисс называется «компромиссом между исследованием и эксплуатацией» (англ. exploration–exploitation tradeoff).

Для достижения этого компромисса в [128] был предложен алгоритм, основанный на методах обучения с подкреплением, а именно на решении задачи о многоруком бандите. В рамках этой задачи агент итеративно активирует различные так называемые «ручки» и после каждой итерации получает вознаграждение. Цель агента заключается в разработке стратегии последовательности активации ручек, максимизирующей вознаграждение (оптимизирующей целевую функцию). «Ручки» в такой постановке задачи соответствуют алгоритмам оптимизации, настраивающим гиперпараметры каждого алгоритма кластеризации. Важно отметить, что используемый алгоритм должен быть итеративным, чтобы его можно было приостанавливать, сохранять текущее состояние и продолжать выполнение позже в случае повторного выбора

соответствующей ручки. Вознаграждение при этом задается улучшением выбранной меры качества кластеризации.

В [145] авторами предложен метод выбора меры качества кластеризации (англ. Cluster Validity Index, CVI), основанный на мета-обучении. В широком смысле мета-обучение представляет собой совокупность подходов, методов и техник, ориентированных на перенос знаний о решении одного множества задач для ускорения поиска при решении других задач. В узком смысле мета-обучение подразумевает использование мета-модели (мета-классификаторов) как подхода к решению задачи выбора алгоритма, при котором алгоритмы машинного обучения, а именно алгоритмы классификации, применяются к мета-данным о предыдущих экспериментах. В подобной задаче наборы данных выступают объектами, поэтому в [146] мета-классификатор обучается на наборе наборов данных. Характеристики наборов данных называются мета-признаками.

4.2 Существующие методы машинного обучения для мультимодальных данных

Одним из базовых подходов является создание совместного представления (joint representation). Данный подход заключается в независимом преобразовании данных различных модальностей в векторные представления с последующим объединением векторов в общее семантическое подпространство с использованием модели слияния (например, EmbraceNet, EmbraceNet+, простая конкатенация векторов или BiT [111]). Однако у данного подхода имеются недостатки. Получаемое представление, как правило, сохраняет общую семантику между модальностями, но при этом игнорирует специфическую информацию, присущую каждой модальности. Кроме того, после применения данного подхода невозможно отдельно извлечь представление каждой модальности.

Модель согласованного представления (coordinated representation) использует раздельные представления для каждой модальности, после чего объединяет их в единое согласованное пространство. Такое представление

позволяет учитывать специфические для конкретной модальности характеристики [124].

Энкодер-декодерные архитектуры переводят представление одной модальности в другую. При этом энкодер отображает представление первой модальности в скрытый вектор, тогда как декодер восстанавливает (конструирует) представление второй модальности на основе указанного вектора. Такая модель может включать несколько энкодеров и декодеров, что позволяет раздельно выделять несколько независимых характеристик одной модальности (например, несколько записанных музыкальных инструментов в аудио). Важно отметить, что скрытый вектор формируется не только с учетом входной модальности, как могло бы показаться, но и с учетом выходной (целевой) модальности, поскольку ошибка преобразования распространяется от декодера к энкодеру, тем самым учитывая обе модальности [113].

Еще одним существенным подходом к работе с мультимодальными данными являются глубокие мультимодальные машины Больцмана (Deep Multimodal Boltzmann Machines, DBM). DBM представляют собой вероятностные графовые модели, состоящие из двух ограниченных машин Больцмана, имеющих общий слой представления. Связь модальностей обеспечивается с использованием функции потерь, основанной на корреляции. Кроме того, сети являются двунаправленными, что позволяет выполнять преобразование данных между модальностями. Существенным недостатком таких моделей является высокая потребность в вычислительных ресурсах [114].

Архитектуры автоэнкодеров позволяют реконструировать любую модальность даже в ситуации отсутствия одной из модальностей. Процесс обучения автоэнкодера заключается в минимизации функции потерь, вычисляемой после реконструкции модальностей. В [115] предложена регуляризация весов слагаемых функции потерь, соответствующих различным модальностям, с целью уменьшения избыточности (дублирующейся информации) в формируемом представлении. Отмечается, что модель также способна выявлять модально-специфические признаки.

Принципиально другой подход был разработан в 1980-х гг. авторами [116]. Метод ADAPT работает с разнотипными переменными (смешанными непрерывными и бинарными) путем представления всего пространства поиска как дискретного множества D, где каждая переменная имеет конечный набор значений, независимо от типа. Непрерывные переменные не подвергаются явной дискретизации заранее - они интегрируются в дискретную структуру как упорядоченные множества возможных значений, что позволяет унифицировать обработку с бинарными переменными (которые, очевидно, дискретны, например, {0,1}). Такой подход применим к широкому классу задач, но не может быть универсальным. Более того, метод [116] не рассчитан на работу с эмбеддингами, так как был разработан до широкого внедрения нейронных сетей.

Несмотря на наличие методов, ориентированных на мультимодальные данные, ни один из рассмотренных подходов не является непосредственно применимым к современным задачам кластеризации. Это связано с тем, что в общем случае постановка кластеризации не предполагает наличия разметки, следовательно, отсутствует возможность обучения моделей на существующих размеченных наборах данных. Более того, методы, основанные на глубоком обучении, плохо реализуются на распределенных вычислительных системах, так как требуют значительного количества вычислительных ресурсов, и, таким образом, их применение затруднено при обработке больших данных.

4.3 Модель кластеризации мультимодальных данных без использования методов обучения с учителем

Пусть $X = (X_1, X_2, \dots, X_n)$ - мультимодальный набор данных, где через $X_i = (X_i^1, X_i^2, \dots, X_i^m)$ обозначен один мультимодальный объект, а X_i^k - представление k-й модальности объекта X_i. Разбиение набора данных X на попарно непересекающиеся множества объектов обозначено как $G(X)$, а целевая мера качества кластеризации задана отображением $Q: G(X) \to \mathbb{R}$. Требуется найти

такое разбиение $\hat{G}(X)$, которое максимизирует целевую меру качества кластеризации:

$$\hat{G}(X) = \underset{G(X)}{\operatorname{argmax}} Q(G(X)).$$

Для автоматизации процедуры выбора меры качества использовался распределенный алгоритм [91], основанный на мета-обучении. В широком смысле мета-обучение включает совокупность методов, направленных на перенос знаний о решении одной задачи с целью ускорения поиска решений для других задач. В узком смысле мета-обучение предполагает применение мета-моделей (мета-классификаторов) для выбора алгоритма, когда методы машинного обучения, в частности алгоритмы классификации, применяются к мета-данным о предыдущих экспериментах по машинному обучению.

В контексте узкой постановки задачи мета-обучения характеристики качества работы алгоритмов предсказываются по характеристикам данных (мета-признакам). В подобной задаче наборы данных выступают объектами. Следовательно, мета-классификаторы обучаются на наборах данных.

Используемая система рекомендации меры качества кластеризации опирается на гипотезу [146], согласно которой наилучшим образом подходящую внутреннюю меру качества кластеризации (CVI) можно определить по агрегированным характеристикам набора данных. Тем самым появляется возможность вычислить ряд статистических показателей и по ним предсказать рекомендуемую меру качества. Для обучения модели-предсказателя использовался искусственный набор данных OpenML [147]. Для каждого набора данных наиболее подходящая CVI определялась экспертами визуально [147], что позволило сформировать размеченную выборку, в которой метками выступали CVI. В число доступных мер входили мера Калински–Харабаша, индекс силуэта, обобщенный индекс Данна и Score-Function.

Для корректной обработки мультимодальных объектов при сохранении семантики предлагается модель кластеризации, состоящая из четырех основных компонентов (рисунок 4.1).

Рисунок 4.1 – Основные компоненты модели кластеризации мультимодальных данных

Веса модальностей могут быть определены как вручную исследователем (или пользователем), так и исходя из какой-либо эвристики. Вычисление расстояний между объектами производится путем агрегации взвешенных нормализованных внутримодальных расстояний.

В случае, если коэффициенты весов модальностей не заданы пользователем и не определен способ их вычисления, а также специфический способ их нормализации, предлагается использовать следующие формулы:

$$\alpha_k = \frac{\sum_{t=1}^{m} \dim(t)}{\dim(k)}, \tag{4.1}$$

$$\hat{d}_k\left(X_i^k, X_j^k\right) = \frac{d_k\left(X_i^k, X_j^k\right)}{\max\limits_{p,q \in \{1,2,\ldots,n\}} d_k\left(X_p^k, X_q^k\right)}, \tag{4.2}$$

$$D(X_i, X_j) = \sqrt{\sum_{k=1}^{m} \alpha_k \ \hat{d}_k^{\,2}(X_i^k, X_j^k)}. \tag{4.3}$$

Здесь $D(X_i, X_j)$ - расстояние между мультимодальными объектами, α_k - весовой коэффициент k-й модальности, $\dim(k)$ - размерность векторного представления k-й модальности, $\hat{d}_k$ - нормированное внутримодальное расстояние, d_k - внутримодальное расстояние между векторными представлениями для k-й модальности.

Коэффициент α_k (формула 4.1) в представленном подходе отвечает за вклад модальности k в расстояние. Эвристика $\alpha_k = \frac{\sum_{t=1}^{m} \dim(t)}{\dim(k)}$ позволяет «уравновесить» вклад каждой из модальностей, но предполагается, что весовые коэффициенты определяются экспертно (например, равными). Предлагаемая эвристика позволяет придать модальностям меньшей размерности больший вес.

Нормализация (формула 4.2), которая использует принцип, схожий с Min-Max Scaling, приводит все расстояния к диапазону $(0,1]$. Такая нормализация является стандартной практикой для эмбеддингов и очищенных от шума данных, но может быть неэффективна, если в данных присутствуют аномалии (выбросы). В этом случае рекомендуется использовать другие методы нормализации, например, нормализацию по среднеквадратическому отклонению.

Для агрегации модальностей (формула 4.3) также существуют альтернативы, такие как простая конкатенация модальностей или метод ADAPT, которые приводят все модальности к единому векторному пространству без использования методов глубокого обучения, но обладают рядом недостатков. Метод конкатенации векторов приводит к тому, что модальности высокой размерности слишком сильно влияют на конечное расстояние между объектами. Метод ADAPT позволяет обрабатывать мультимодальные данные, не прибегая к глубокому обучению, но он также имеет ряд недостатков: в современных задачах зачастую требуется обработка эмбеддингов, то есть векторов действительных чисел высокой размерности. Метод ADAPT приводит все действительные числа к конечному множеству дискретных значений (фактически осуществляя квантизацию), не уменьшая размерность данных, но приводя к потере информации, что приводит к потере точности без повышения вычислительной эффективности.

Очевидно, евклидова норма не является единственно возможной и может быть заменена на альтернативную. Более того, предложенный подход позволяет вычислять внутримодальные расстояния способом, специфичным для каждой модальности.

Такая модель кластеризации позволяет вычислять расстояния между модальностями по отдельности, а итоговое расстояние вычисляется по ним. Это позволяет избежать создания единого вектора (эмбеддинга), отражающего все модальности, для каждого объекта. Создание единого эмбеддинга для каждого объекта может приводить как к возникновению эффектов «проклятия размерности», если размерность результирующего вектора слишком высока, так и

к потере информации, если размерность мала. Также такой подход требует дополнительных вычислительных затрат, а для некоторых подходов и обучения модели глубокого обучения на размеченных данных.

Предложенная модель обладает следующими преимуществами:

- вычислительная простота;
- отсутствие необходимости обучения сложной модели с учителем;
- широкие возможности модификации и персонализации.

В рамках предлагаемой модели, основанной на особом вычислении расстояния между мультимодальными объектами, алгоритмы кластеризации могут быть применены и к мультимодальным данным: изменения затрагивают только способ оценки близости объектов в пространстве. Примерами таких алгоритмов кластеризации являются алгоритмы k-средних, MeanShift или DBSCAN.

4.4 Оптимизация гиперпараметров алгоритмов кластеризации

После определения способа оценки расстояний между мультимодальными объектами становится возможным использовать существующие алгоритмы кластеризации для получения разбиений $G(X)$.

Пусть $C(p_1, p_2, \dots, p_h): X \to G(X)$ - алгоритм кластеризации с гиперпараметрами $p_1, p_2, \dots, p_h$. Множество всех допустимых значений гиперпараметра p_i обозначим P_i , а пространство поиска для алгоритма C определим как

$$P(C) = P_1 \times P_2 \times \cdots \times P_h.$$

При этом P_i может быть как дискретным (целочисленным или категориальным), так и непрерывным (вещественным числом).

В качестве подзадачи требуется найти оптимальную конфигурацию гиперпараметров:

$$(\hat{p}_1, \hat{p}_2, \dots, \hat{p}_h) = \arg \max_{(p_1, p_2, \dots, p_h)} Q(C(p_1, p_2, \dots, p_h)).$$

Пусть S_C: () $\rightarrow P(C)$ - генератор конфигураций (sampler), который на каждой итерации выдает алгоритму новый набор гиперпараметров. Существует множество подходов к генерации конфигураций (случайный поиск, перебор по сетке, байесовская оптимизация и др.); в настоящей работе используется метод Tree Parzen Estimator, использованный во фреймворке Optuna [148]. Кроме того, фреймворк Optuna прост в установке и интеграции в программный продукт.

Данный подход позволяет приостанавливать процесс оптимизации гиперпараметров конкретного алгоритма кластеризации и переключаться на оптимизацию гиперпараметров других алгоритмов кластеризации [128].

4.5 Выбор алгоритма кластеризации

Задача выбора алгоритма кластеризации в условиях ограниченного времени является задачей поиска компромисса между исследованием и эксплуатацией. Для поиска этого компромисса предлагается использовать стратегию многорукого бандита для выбора следующего алгоритма кластеризации.

Задача о многоруком бандите (multi-armed bandit) представляет собой формальную модель последовательного принятия решений, в рамках которой на каждом шаге выбирается одно из конечного множества действий («ручек»), после чего наблюдается случайная награда. Распределение наград неизвестно. Целью является максимизация накопленной ожидаемой награды на горизонте наблюдений. Принципиальная особенность данной задачи заключается в необходимости согласования двух взаимно конкурирующих требований: исследования (сбора информации о слабо изученных действиях) и эксплуатации (предпочтения действий с наилучшей текущей оценкой). Ниже приведена более формальная постановка задачи о многоруком бандите в контексте выбора и настройки алгоритмов кластеризации, где выбор «ручки» означает выбор алгоритма кластеризации, которому следует на следующей итерации выделить ресурс времени для оптимизации его параметров, а «наградой» является прирост меры качества кластеризации за отведенное время.

Пусть $C^1, C^2, \dots, C^a$ - различные алгоритмы кластеризации с соответствующими генераторами конфигураций $S_{C^1}, S_{C^2}, \dots, S_{C^a}$. На каждой итерации выбирается один из алгоритмов, затем генерируется его конфигурация, после чего возвращается разбиение набора данных $G(X)$. Итерацию обозначим как C_i^j - j-я конфигурация для алгоритма кластеризации C_i. Затраченный бюджет времени на оценку конфигурации обозначим $\mathrm{time}(C_i^j)$, а число выполненных запусков (конфигураций) для алгоритма C_i - как $\mathrm{runs}(C_i)$.

Для каждого алгоритма кластеризации вводится следующая функция вознаграждения:

$$R(C_i) = Q_r(C_i) + T_r(C_i),$$

$$Q_r(C_i) = \frac{Q_a(C_i)}{\max\limits_{p=1,2,\dots,a} Q_a(C_p)},$$

$$Q_a(C_i) = \max_{j=1,2,\dots,\mathrm{runs}(C_i)} \left(Q(C_i^j)\right) - \hat{Q},$$

$$\hat{Q} = \operatorname{argmin}_{G(X)} Q(G(X)),$$

$$T_r(C_i) = 1 - \frac{T_a(C_i)}{\sum_{p=1}^{a} T_a\,(C_p)},$$

$$T_a(C_i) = \sum_{j=1}^{\mathrm{runs}(C_i)} \mathrm{time}\left(C_i^j\right).$$

Здесь $R(C_i)$ - вознаграждение для алгоритма кластеризации C_i на текущей итерации; $Q_r(C_i)$ - компонент вознаграждения, отвечающий за качество; $Q_a(C_i)$ - оценка целевого показателя качества для наилучшей (среди уже выполненных) конфигурации алгоритма кластеризации; $\hat{Q}$ - оценка наихудшего возможного разбиения для набора данных X, приближенно получаемая посредством вычисления Q на случайном разбиении $G_{\mathrm{random}}(X)$; $T_r(C_i)$ - временной компонент вознаграждения; $T_a(C_i)$ - суммарные временные затраты на выполнение всех конфигураций алгоритма кластеризации C_i.

Иными словами, на каждой итерации алгоритм решения задачи о многоруком бандите стремится предсказать наиболее перспективный алгоритм с точки зрения улучшения значения целевой функции.

Алгоритм решения задачи о многоруком бандите лег в основу фреймворка, разработанного для оценки применимости разработанной мультимодальной меры расстояния.

На первом этапе работы разработанного фреймворка «Sparkling» необходимо загрузить набор данных в процедуру предварительной обработки. Далее, при необходимости, запускается система рекомендации меры качества кластеризации. После этого задается ограничение по времени и инициируется процесс оптимизации. На каждом шаге оптимизатор выбирает алгоритм, выполняет настройку его гиперпараметров в рамках ограниченного временного бюджета, после чего MASSCAH пересчитывает величину вознаграждения и повторяет цикл до тех пор, пока заданный лимит времени не будет исчерпан. Рисунок 4.2 схематически иллюстрирует этапы процесса построения модели кластеризации.

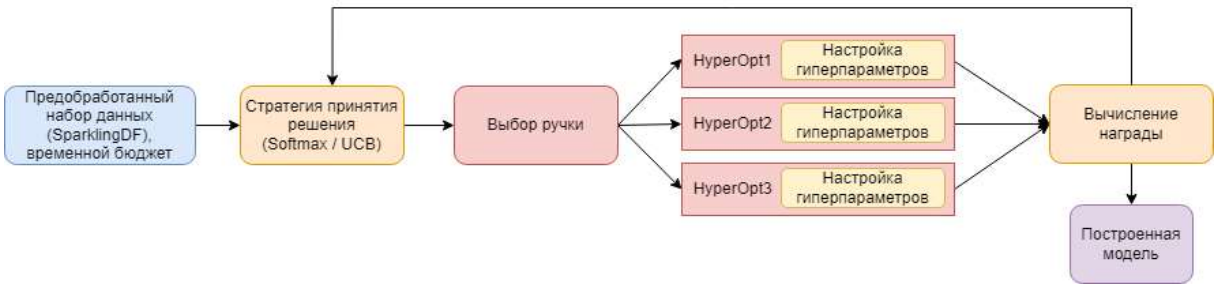


Рисунок 4.2 – Общая последовательность действий (pipeline) процесса выбора и оптимизации алгоритма кластеризации

Для работы с наборами данных большого объема использовался фреймворк Apache Spark [149, 150]. Этот фреймворк является эффективным инструментом для кластеризации данных на вычислительном кластере за счет высокой производительности (в частности, благодаря обработке в оперативной памяти), простоты использования API, как вертикальной, так и горизонтальной

масштабируемости, а также наличия встроенных инструментов машинного обучения. Указанные свойства делают Apache Spark пригодным для обработки больших объемов данных и решения задач кластеризации.

Нестандартные представления наборов данных (обусловленные мультимодальностью), а также предложенная мера расстояния не удовлетворяют API Spark MLLib. Для поддержания мультимодальных наборов данных были реализованы собственные алгоритмы кластеризации и меры качества кластеризации. Предварительные эксперименты показали, что выполнение собственных реализаций на основе PySpark API не позволяют создать высокопроизводительную реализацию алгоритмов кластеризации. Для ускорения как алгоритмов кластеризации, так и вычисления CVI они были реализованы на основе Scala Spark API, поскольку Apache Spark разработан для Scala. Кроме того, Apache Spark инициализирует JVM и поддерживает соединение Py4J между Python API и Java-процессом. Такая архитектура PySpark позволяет (прозрачно с точки зрения пользователя) подключать пользовательские реализации на Scala, используя тот же экземпляр JVM и мост Py4J.

4.6Приведение данных к векторным представлениям

Для преобразования текстов или изображений в векторные представления (эмбеддинги) разработанный фреймворк использует предварительно обученные модели глубокого обучения, представленные в репозитории HuggingFace. Вместе с тем вычисление эмбеддингов, вероятно, является наиболее узким местом в разработанном фреймворке по следующим причинам:

- выбранная модель должна быть распространена на каждый узел кластера, что создает повышенную нагрузку на сеть кластера, особенно для моделей с большим числом параметров;

- предварительно обученные модели требуют среды выполнения Python и не могут быть исполнены на Scala; это приводит к существенным накладным расходам при выполнении PySpark UDF (user-defined function);

- изображения хранятся в файловой системе кластера, а не непосредственно в Spark-датафрейме (пользователь должен задавать лишь путь к изображению для каждого объекта); это требует интенсивных операций ввода-вывода и создает дополнительную нагрузку на сеть кластера; хотя Sparkling предпринимает попытки оптимизировать этап преобразования, подгружая пакет изображений в память узла по требованию, это уменьшает потребление памяти, но не снижает количество операций ввода-вывода;

- аппаратная конфигурация кластера Apache Spark с GPU на каждом узле является дорогостоящей и потому не применялась, в то время как вычисление эмбеддингов на CPU резко замедляет предварительную обработку набора данных.

4.7 Экспериментальное исследование применимости разработанной модели кластеризации

Фреймворк «Sparkling» поддерживает запуск Apache Spark как локально, так и на YARN-кластере. Экспериментальные результаты с использованием (4.1)-(4.3) приведены для кластера Yandex Cloud, включающего 5 машин, каждая из которых оснащена 4 vCPU и 16 ГБ ОЗУ, под управлением ОС Ubuntu 18.04.

Поскольку предварительная обработка данных не входит в задачи работы, для модульного тестирования были выбраны 9 реальных табличных унимодальных наборов данных, а также несколько мультимодальных наборов данных, сгенерированных с различным числом объектов (в диапазоне от 100 тысяч до 2,5 миллионов). Для тестирования алгоритмов кластеризации для каждого алгоритма была задана сетка гиперпараметров, и каждая конфигурация оценивалась на описанных выше наборах данных.

Ниже приведены сетки гиперпараметров, заданные для каждого алгоритма [91]:

- алгоритм k-средних с параметром $k \in \{2,3,5,7,11\}$;

- алгоритм Birch с параметрами $\text{maxBranches} \in \{5,12,25\}$, $\text{threshold} \in \{0.1,0.3,0.7\}$, $k \in \{2,5,14\}$;

- алгоритм Bisecting K-means с параметрами $k \in \{2,3,5,7,11\}$, $\text{minClusterSize} \in \{0.2,0.5,1.0\}$;

- алгоритм DBSCAN с параметрами $\text{eps} \in \{0.02,0.06,0.11,0.17\}$, $\text{borderNoise} \in \{\text{True}, \text{False}\}$;

- алгоритм Mean Shift с параметром $\text{Radius} \in \{0.03,0.07,0.12,0.18,0.25\}$;

- спектральный алгоритм кластеризации с матрицей сходства с параметрами $\text{eigens} \in \{7,13,19\}$, $\gamma \in \{0.1,0.4,1.0\}$, $k \in \{2,5,9\}$;

- спектральный алгоритм кластеризации с матрицей смежности с параметрами $\text{eigens} \in \{7,13,19\}$, $\text{neighbours} \in \{5,11,20\}$, $k \in \{2,5,9\}$.

Меры качества кластеризации тестировались на тех же наборах данных, при этом результаты кластеризации в различных конфигурациях гиперпараметров использовались в качестве меток. Важно подчеркнуть, что сопоставить вычисленные результаты с существующими реализациями мер качества не представляется возможным. Это обусловлено использованием новой мультимодальной метрики расстояния, а также отсутствием альтернатив для ряда разработанных распределенных реализаций мер качества в рамках области охвата проекта.

Для каждого набора данных требуется настройка процесса предобработки данных. В таблице 4.1 приведено время, затрачиваемое на предварительную обработку данных различными моделями для разных модальностей данных.

К разработанному фреймворку выдвигались следующие требования: архитектура модели должна быть ориентирована на выполнение на центральных процессорах (CPU), объема оперативной памяти 12 ГБ должно быть достаточно для вычисления эмбеддингов и запуска алгоритмов кластеризации, размерность эмбеддингов должна находиться в диапазоне от 512 до 1024, а скорость и точность алгоритмов кластеризации должны быть приемлемыми в рамках поставленных задач.

Таблица 4.1 - Время, затрачиваемое на преобразование модальных данных в векторные представления, и используемые технологии

Набор данных	Модель для графических данных	Модель для текстовых данных	Время, мин
100BirdsSpecies	Swin Transformer	—	47
Flick dataset	Swin Transformer	BERT	45
DiffusionDB	Swin Transformer	ALBERT	65
Wikipedia dataset	Swin Transformer	ALBERT	10
Houses dataset	Swin Transformer; EfficientNet; ConvNeXT	—	6.50; 5.00; 8.50
Amazon	—	ALBERT	280

Этап автоматического выбора меры качества кластеризации является необязательным, если пользователь явно задает целевую метрику качества. В противном случае основная доля времени будет затрачиваться на вычисление мета-признаков набора данных, что эквивалентно суммарному времени формирования рекомендации. В таблицах 4.2 и 4.3 приведены результаты экспериментов, в рамках которых рекомендательная система предсказывает наиболее подходящую меру для различных наборов данных.

Таблица 4.2 - Рекомендуемые меры качества для синтетических наборов данных

Набор данных	Рекомендуемая мера	Время, с
target	GD41_APPROX	18
tetra	CALINSKI_HARABASZ	9
threenorm	SCORE	24
triangle1	CALINSKI_HARABASZ	24
triangle2	SCORE	25
twenty	CALINSKI_HARABASZ	24
twodiamonds	SCORE	19
wingnut	SCORE	25
xclara	SCORE	37
xor	GD41_APPROX	24

Таблица 4.3 - Рекомендуемые меры качества для реальных наборов данных

Набор данных	Рекомендуемая мера	Время, с
arrhythmia	GD41_APPROX	26
balance-scale	CALINSKI_HARABASZ	21
cpu	GD41_APPROX	7
dermatology	CALINSKI_HARABASZ	17
ecoli	SCORE	10
german	GD41_APPROX	33
glass	SCORE	7
haberman	SCORE	8
heart-statlog	CALINSKI_HARABASZ	7
iono	SCORE	10

После завершения предварительной обработки данных фреймворк Sparkling инициирует поиск наилучшего алгоритма кластеризации и его оптимальной конфигурации. Для идентификации оптимального алгоритма и решения задачи многорукого бандита могут применяться алгоритмы Softmax и UCB (Upper-Confidence Bound). Для настройки гиперпараметров рекомендуется использовать Optuna.

В таблице 4.4 представлена информация о результатах, полученных на мультимодальных наборах данных: время выполнения, какая внутренняя мера качества (CVI) оптимизировалась в рамках фреймворка, а также какой алгоритм оказался оптимальным.

Помимо реальных мультимодальных наборов данных, эксперименты проводились на двух коллекциях размеченных табличных данных (OpenML artificial и OpenML real-world). Для каждого файла из набора данных выполнялся запуск с оптимизацией таких метрик, как индекс силуэта, мера Калинского–Харабаза, обобщенный индекс Данна и Score-Function. В таблице 4.5 приведено сводное описание результатов, полученных при запуске Sparkling на указанных наборах данных и сравнение разработанной модели кластеризации с методом ADAPT.

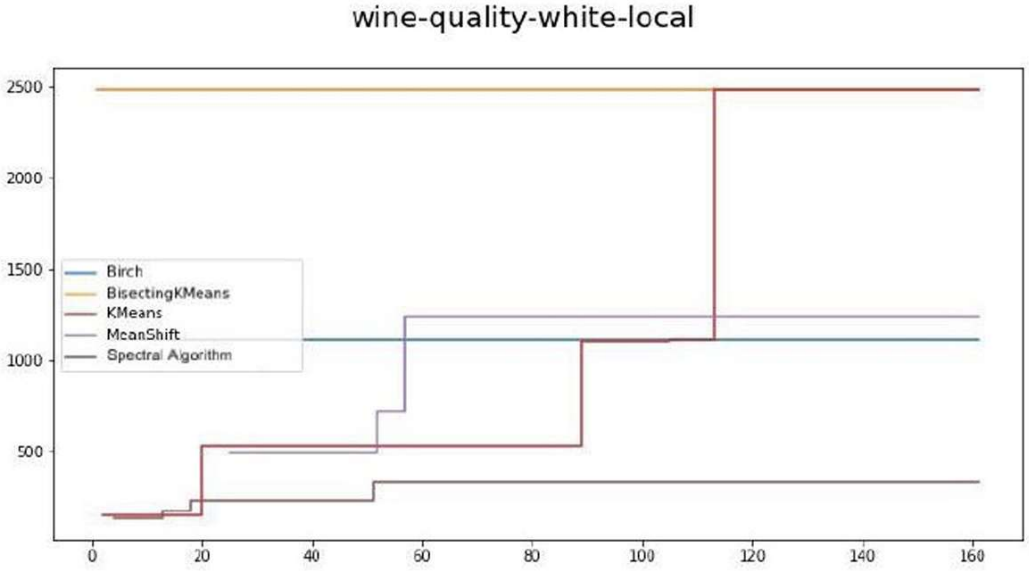


Рисунок 4.3 – Производительность алгоритма на наборе данных, описывающем белое вино; кумулятивный минимум меры Калински–Харабаша; временной ресурс 20 минут; оптимизатор Optuna; стратегия выбора алгоритма Softmax. По оси x – номер итерации алгоритма решения задачи о многоруком бандите, по оси y – достигнутое значение меры Калински–Харабаша

Задачей эксперимента, результаты которого представлены на рисунке 4.3, было выявление алгоритмов, которые могут обеспечивать достижение приемлемой меры качества кластеризации без необходимости ручной корректировки конфигурационных параметров или диапазонов оптимизируемых гиперпараметров. Для достижения этой цели была проведена серия экспериментов, результаты одного из которых представлен на рисунке 4.4.

Таблица 4.4 - Рекомендуемые меры качества для мультимодальных наборов данных

Набор данных	Рекомендуемая мера	Время, с	Оптимальный алгоритм
100BirdsSpecies	Davies-Bouldin Score	32	KMeans
100BirdsSpecies	Calinski-Harabasz Score	31	BisectingKMeans
100BirdsSpecies	Dunn Approximation	30	MeanShift
100BirdsSpecies	Silhouette Index Approx	150	KMeans
100BirdsSpecies	Score Function	77	BisectingKMeans
Flick dataset	Davies-Bouldin Score	33	Birch
Flick dataset	Calinski-Harabasz Score	35	Birch
Flick dataset	Dunn Approximation	66	MeanShift
Flick dataset	Silhouette Index Approx	34	Birch
Flick dataset	Score Function	36	BisectingKMeans
DiffusionDB	Davies-Bouldin Score	100	MeanShift
DiffusionDB	Calinski-Harabasz Score	46	Birch
DiffusionDB	Dunn Approximation	100	Birch
DiffusionDB	Silhouette Index Approx	46	KMeans
DiffusionDB	Score Function	30	BisectingKMeans
Houses dataset	Davies-Bouldin Score	40	MeanShift
Houses dataset	Calinski-Harabasz Score	32	BisectingKMeans
Houses dataset	Dunn Approximation	32	BisectingKMeans
Houses dataset	Silhouette Index Approx	27	MeanShift
Houses dataset	Score Function	32	BisectingKMeans
Hand Gestures	Davies-Bouldin Score	41	MeanShift
Hand Gestures	Calinski-Harabasz Score	53	MeanShift
Hand Gestures	Dunn Approximation	38	Birch
Hand Gestures	Silhouette Index Approx	38	KMeans
Hand Gestures	Score Function	41	Birch
Wikipedia dataset	Davies-Bouldin Score	28	MeanShift
Wikipedia dataset	Calinski-Harabasz Score	28	BisectingKMeans
Wikipedia dataset	Dunn Approximation	32	MeanShift
Wikipedia dataset	Silhouette Index Approx	29	MeanShift
Wikipedia dataset	Score Function	8	BisectingKMeans

Временная динамика на графике (рисунок 4.3) указывает на улучшение качества кластеризации по мере выполнения оптимизации. Итоговые значения основных внешних мер качества приведены в таблице 4.5.

Таблица 4.5 - Сводка достигнутых значений внешних мер качества на размеченных данных

Мера качества	Агрегация модальностей, среднее значение меры качества на сгенерированных данных	Агрегация модальностей, среднее значение меры качества на реальных данных	ADAPT, среднее значение меры качества на сгенерированных данных	ADAPT, среднее значение меры качества на реальных данных
Rand Index, среднее	0.78	0.73	0.68	0.66
Jaccard Index, среднее	0.69	0.71	0.63	0.63
F-мера, среднее	0.72	0.75	0.65	0.66

Представленные в таблице 4.5 результаты демонстрируют, что разработанная модель кластеризации способна определять группы схожих объектов на неразмеченных данных лучше, чем известный в литературе метод ADAPT.

Заголовок графика (рисунок 4.4) содержит соответствующие сведения. Представленные результаты были получены с использованием оптимизатора гиперпараметров Optuna и стратегии выбора «ручки» многорукого бандита UCB. Результаты тестирования подтверждают эффективность разработанной библиотеки на наборах данных малого масштаба и подчеркивают значимость корректного выбора внутренней меры качества для оптимизации с целью получения удовлетворительных результатов.

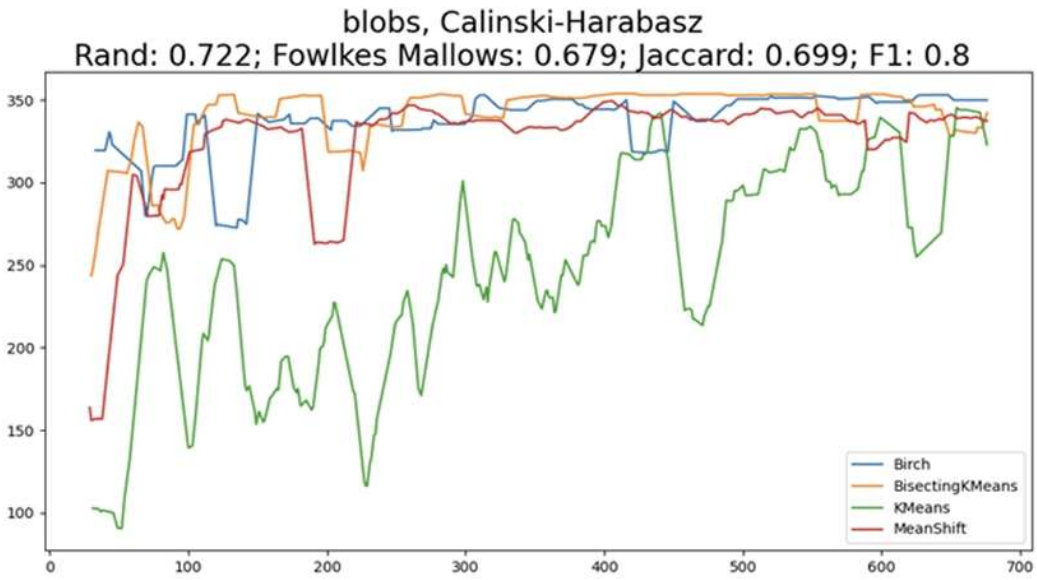


Рисунок 4.4 – Значения меры качества во времени, представленные со скользящим средним по 30 тикам для улучшения визуального восприятия. По оси x – номер итерации алгоритма решения задачи о многоруком бандите, по оси y – достигнутое значение меры Калински–Харабаша

Рисунок 4.4 иллюстрирует полученные значения внутренней меры Калински–Харабаша для стандартного сгенерированного набора данных blobs.

4.8 Результаты главы 4

В настоящей главе представлена модель кластеризации мультимодальных данных на основе меры расстояния, позволяющей вычислять меру близости мультимодальных объектов без создания единого векторного пространства. Данный подход устраняет необходимость в предварительном сведении разнородных данных к общему векторному представлению, что является ключевой особенностью предложенной модели.

Разработаны распределенные версии алгоритмов, вычисляющих внутренние и внешние меры качества кластеризации, проведены эксперименты с различными алгоритмами оптимизации гиперпараметров, а также реализован ряд алгоритмов кластеризации, специально адаптированных для работы с мультимодальными данными в рамках предложенной модели.

Проведенные экспериментальные исследования на различных наборах мультимодальных данных демонстрируют не только применимость предложенной модели кластеризации, но и ее преимущество по сравнению с известными моделями.

ЗАКЛЮЧЕНИЕ

В диссертационной работе предложены алгоритмы автоматической группировки данных для построения IVF-индекса и адаптивный алгоритм поиска ближайших соседей, позволяющие повысить точность и скорость работы алгоритмов поиска ближайших соседей на основе IVF-индекса на 10-39%. Были решены следующие задачи:

1. Представлен алгоритм классификации сложности запросов приближенного поиска ближайших соседей с использованием IVF-индекса, позволяющий определить сложность запроса и предсказать количество кластеров, в которых находятся ближайшие соседи, а также алгоритм для обучения этого классификатора. Продемонстрировано значительное повышение производительности поиска. Продемонстрирована применимость такого алгоритма к большим объемам данных.

2. Разработан адаптивный алгоритм поиска в векторной базе данных на основе IVF-индекса, который по результатам предварительного поиска и классификации сложности запроса динамически определяет ограничения пространства поиска и повышает вычислительную эффективность при сохранении требуемой точности, прирост составил 10-39% на наборах данных объемом до 10^9 объектов при значении полноты 0,99.

3. Разработаны новые эволюционные алгоритмы решения задачи *k*-средних, отличающиеся от известных применением оператора мутации, основанном на ускоренной жадной агломеративной процедуре. Результаты вычислительных экспериментов подтверждают стабильность и качество кластеризации, достигаемое новым эволюционным алгоритмом, а также прирост вычислительной эффективности процедуры поиска приближенных ближайших соседей по IVF-индексу.

4. Представлена новая модель кластеризации мультимодальных данных, позволяющая применять алгоритмы кластеризации напрямую, без приведения

модальностей к единому векторному пространству и повысить точность решения задачи кластеризации.

В настоящем диссертационном исследовании была разработана методология ускорения поиска в векторных базах данных, включающая определение глубины поиска, оптимизацию построения индекса методами кластеризации и обработку сложных типов данных (мультимодальных данных). Предложенные алгоритмы обеспечивают практическое повышение производительности при сохранении заданного качества поиска и демонстрируют устойчивость на больших объемах данных. Сформулированные подходы могут быть использованы при проектировании и модернизации высокопроизводительных систем приближенного поиска.

Также представленные в настоящей работе исследования открывают несколько направлений для дальнейших исследований. Оценка доли результативных кластеров на раннем этапе поиска ближайших соседей для определения глубины поиска может быть развита в дальнейшем, как для адаптивных методов поиска, так и для других задач. Принцип аппроксимации прироста целевой функции задачи k-средних, использованный в ускоренной агломеративной процедуре кластеризации эмбеддингов, может найти применение и в других практических задачах, включающих кластеризацию. Новая модель кластеризации мультимодальных данных, не использующая моделей глубокого обучения, также демонстрирует многообещающие перспективы практического применения.

СПИСОК ЛИТЕРАТУРЫ

1. Abbasifard M.R., Ghahremani B., Naderi H. A Survey on Nearest Neighbor Search Methods // International Journal of Computer Applications. 2014. Vol. 95. P. 39–52.

2. McLachlan G.J. Mahalanobis Distance // Resonance. 1999. № 4. P. 20–26.

3. Ponomarenko A., Malkov Yu., Logvinov A., Krylov V. Approximate Nearest Neighbor Search Small World Approach // International Conference on Information and Communication Technologies & Applications 2011. Vol. 17. P. 1-6. DOI: 10.13140/2.1.2152.8964.

4. Bhatia N., Ashev V. Survey of Nearest Neighbor Techniques // International Journal of Computer Science and Information Security. 2010. Vol. 8, no. 2. P. 1–4.

5. Hwang Y., Han B., Ahn H.-K. A fast nearest neighbor search algorithm by nonlinear embedding // 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Providence, RI, USA, 2012. Vol. 2012 P. 3053–3060. DOI: 10.1109/CVPR.2012.6248036.

6. Kazakovtsev V., Plekhanov M., Naumchev A., Shkaberina G., Masich I., Egorova L., Stupina A., Popov A., Kazakovtsev L. Fast Adaptive Approximate Nearest Neighbor Search with Cluster-Shaped Indices // Big Data and Cognitive Computing. 2025. Vol. 9. P. 254. DOI: 10.3390/bdcc9100254.

7. Weber R., Blott S. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces // VLDB. 2000. Vol. 98. P.194-205

8. Heneghan C. A method for initialising the K-means clustering algorithm using kd-trees // Pattern Recognition Letters. 2007. Vol. 28. P. 965–973. DOI: 10.1016/j.patrec.2007.01.001.

9. Kraus P., Dzwinel W. Nearest neighbor search by using Partial KD-tree method // TACS. 2008. Vol. 20. P. 149–165.

10. Yen Sh.-h., Shih Ch.-Y., Chang H.-W., Li T.-K. Nearest neighbor searching in high dimensions using multiple KD-trees // Proceedings of the 10th WSEAS international conference on Signal processing, computational geometry and artificial vision. 2010. Vol. 2010. P. 40-45.

11. Guttman A. R-Trees: A Dynamic Index Structure for Spatial Searching // Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data. 1984. Vol. 1984. P. 47–57.

12. Papadopoulos A., Manolopoulos Y. Performance of Nearest Neighbor Queries in R-Trees // ICDT 1997: Database Theory — ICDT'97. 1997. Vol. 1997. P. 394–408. DOI: 10.1007/3-540-62222-5_59.

13. Cheung K.L., Fu A.W.-C. Enhanced nearest neighbour search on the R-tree // SIGMOD Record. 1998. Vol. 27. P. 16–21.

14. Jagadish H., Ooi B.C., Tan K.-L., Zhang R. iDistance: An Adaptive B+-tree Based Indexing Method for Nearest Neighbor Search // ACM Transactions on Database Systems. 2005. Vol. 30. P. 364–397.

15. Song Z., Qin Zh., Deng W., Zhao Y. The B+-tree-based Method for Nearest Neighbor Queries in Traffic Simulation Systems // TELKOMNIKA Indonesian Journal of Electrical Engineering. 2014. Vol. 12. P. 8175-8192 DOI: 10.11591/telkomnika.v12i12.6332.

16. Jafari O., Maurya P., Islam K.M., Nagarkar P. Optimizing Fair Approximate Nearest Neighbor Searches Using Threaded B+-Trees // Similarity Search and Applications. SISAP 2021. Lecture Notes in Computer Science. Vol. 13058. Springer, Cham, 2021. P. 133–147. DOI: 10.1007/978-3-030-89657-7_11.

17. Beygelzimer A., Kakade S., Langford J. Cover Trees for Nearest Neighbor // Proceedings of the 23rd International Conference on Machine Learning (ICML 2006). 2006. Vol. 2006. P. 97–104. DOI: 10.1145/1143844.1143857.

18. Elkin Yu. New compressed cover tree for k-nearest neighbor search [Электронный ресурс] // CoRR 2022. arXiv:2205.10194. (дата обращения: 30.06.2025) DOI: 10.48550/arXiv.2205.10194.

19. Beyer K., Goldstein J., Ramakrishnan R., Shaft U. When Is “Nearest Neighbor” Meaningful? // Database Theory — ICDT’99. Lecture Notes in Computer Science. Vol. 1540. Springer, Berlin, Heidelberg, 1999. Vol. 1999. P. 217–235. DOI: 10.1007/3-540-49257-7_15.

20. Ponomarenko A., Malkov Y., Logvinov A., Krylov V. Approximate Nearest Neighbor Search Small World Approach // International Conference on Information and Communication Technologies & Applications. 2011. Vol. 17. P. 1-6.

21. Malkov Y., Ponomarenko A., Logvinov A., Krylov V. Scalable Distributed Algorithm for Approximate Nearest Neighbor Search Problem in High Dimensional General Metric Spaces // Similarity Search and Applications. 2012. Vol. 2012. P. 132–147.

22. Malkov Y., Ponomarenko A., Logvinov A., Krylov V. Approximate nearest neighbor algorithm based on navigable small world graphs // Information Systems. 2014. Vol. 45. P. 61–68.

23. Zhao K., Lu H., Mei J. Locality Preserving Hashing // AAAI Conference on Artificial Intelligence. 2014. Vol. 28. P. 2874–2880.

24. Cover T., Hart P. Nearest neighbor pattern classification // IEEE Transactions on Information Theory. 1967. Vol. 13, no. 1. P. 21–27. DOI: 10.1109/TIT.1967.1053964.

25. Ge T., He K., Ke Q., Sun J. Optimized product quantization //IEEE transactions on pattern analysis and machine intelligence. 2013. Vol. 36. no. 4. P. 744-755.

26. Slaney M., Casey M. Locality-Sensitive Hashing for Finding Nearest Neighbors [Lecture Notes] // IEEE Signal Processing Magazine. 2008. Vol. 25. P. 128–131. DOI: 10.1109/MSP.2007.914237.

27. Blott S., Weber R. A Simple Vector-Approximation File for Similarity Search in High-Dimensional Vector Spaces. 1998. P. 1-20.

28. Yerpude P. Vector Approximation File: Cluster Bounding in High-Dimension Data Set // International Journal of Engineering and Advanced Technology (IJEAT). 2011. Vol. 1, no. 2. P. 2249–8958.

29. Kröger P., Schubert M., Zhu Z. Efficient Query Processing in Arbitrary Subspaces Using Vector Approximations // Proceedings of the International Conference on Scientific and Statistical Database Management, SSDBM. 2006. Vol. 2006. P. 184–190. DOI: 10.1109/SSDBM.2006.23.

30. Jegou H., Douze M., Schmid C. Product quantization for nearest neighbor search // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2011. Vol. 33, no. 1. P. 117–128.

31. Geist M., Pietquin O., Fricout G. Kernelizing vector quantization algorithms //ESANN'2009. 2009. Vol. 2009. P. 541-546.

32. Kalantidis Y., Avrithis Y. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search // Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 2014. Vol. 2014. P. 2321-2328. DOI: 10.1109/CVPR.2014.298.

33. Yu T., Meng J., Fang C. et al. Product Quantization Network for Fast Visual Search // International Journal of Computer Vision. 2020. Vol. 128. P. 2325–2343.

34. Zhang M., Zhe X., Yan H. Orthonormal Product Quantization Network for Scalable Face Image Retrieval // Pattern Recognition. 2021. Vol. 141. P. 109671.

35. Gu L., Liu J., Liu X. et al. Entropy-Optimized Deep Weighted Product Quantization for Image Retrieval // IEEE Transactions on Image Processing. 2024. Vol. 33. P. 1162–1174.

36. Wang W., Wang H., Xu B. et al.: A Learning Framework for Adaptive Nearest Neighbor Search using Static Features Only [Электронный ресурс] // CoRR. 2021. arXiv:2110.00696 (дата обращения: 13.09.2025).

37. Li C., Zhang M., Andersen D.G., He Y. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data. New York, NY, USA: Association for Computing Machinery, 2020. Vol. 2020 P. 2539–2554. DOI: 10.1145/3318464.3380600.

38. Babenko A., Lempitsky V. The Inverted Multi-Index // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2014. Vol. 37, no. 6. P. 1247–1260.

39. Malkov Y.A., Yashunin D.A. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2020. Vol. 42, no. 4. P. 824–836. DOI: 10.1109/TPAMI.2018.2889473.

40. pgvector. Open-source vector similarity search for Postgres. [Электронный ресурс] URL: https://github.com/pgvector/pgvector (дата обращения: 04.03.2026).

41. Wang J., Yi X., Guo R. et al. Milvus: A purpose-built vector data management system // Proceedings of the 2021 International Conference on Management of Data. 2021. Vol. 2021. P. 2614–2627.

42. Jin Y., Wu Y., Hu W. et al. urator: Efficient Indexing for Multi-Tenant Vector Databases // CoRR. 2024. arXiv: 2401.07119. (дата обращения: 18.01.2026) DOI: 10.48550/arXiv.2401.07119.

43. Bruch S., Nardini F. M., Rulli C., Venturini, R. Efficient inverted indexes for approximate retrieval over learned sparse representations // Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2024. Vol. 2024. P. 152-162.

44. ANN Benchmarks. [Электронный ресурс] URL: https://ann-benchmarks.com/index.html (дата обращения: 04.03.2026).

45. Fu C., Xiang C., Wang C., Cai D. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph // Proceedings of the VLDB Endowment. 2019. Vol. 12, no. 5. P. 461–474. DOI: 10.14778/3303753.3303754.

46. Harwood B., Drummond T. FANNG: Fast Approximate Nearest Neighbour Graphs // Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). 2016. Vol. 2016. P. 5713–5722.

47. Li W., Zhang Y., Sun Y. et al. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement // IEEE

Transactions on Knowledge and Data Engineering. 2020. Vol. 32, no. 8. P. 1475–1488. DOI: 10.1109/TKDE.2019.2909204.

48. Baranchuk D., Persiyanov D., Sinitsin A., Babenko A. Learning to Route in Similarity Graphs // Proceedings of the 36th International Conference on Machine Learning. 2019. Vol. 2019. P. 475–484. URL: http://proceedings.mlr.press/v97/baranchuk19a.html.

49. Bashyam K.G.R., Vadhiyar S. Fast Scalable Approximate Nearest Neighbor Search for High-dimensional Data // 2020 IEEE International Conference on Cluster Computing (CLUSTER). 2020. Vol. 2020. P. 294–302. DOI: 10.1109/CLUSTER49012.2020.00040.

50. Deng S., Yan X., Ng K.W.K. et al. Pyramid: A General Framework for Distributed Similarity Search on Large-scale Datasets // 2019 IEEE International Conference on Big Data (Big Data). 2019. Vol. 2019. P. 1066–1071. DOI: 10.1109/BigData47090.2019.9006219.

51. Subramanya S.J., Devvrit F., Simhadri H.V. et al. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node // Advances in Neural Information Processing Systems. 2019. Vol. 32. P. 13771–13781.

52. Lin P.-C., Zhao W.-L. Graph based Nearest Neighbor Search: Promises and Failures [Электронный ресурс] // CoRR. 2019. arXiv: 1904.02077 (дата обращения: 02.11.2025)

53. Muñoz J.A.V., Gonçalves M.A., Dias Z., Torres R.S. Hierarchical Clustering-Based Graphs for Large Scale Approximate Nearest Neighbor Search // Pattern Recognition. 2019. Vol. 96. P. 106970.

54. McInnes L. PyNNDescent for Fast Approximate Nearest Neighbors [Электронный ресурс] URL: https://pynndescent.readthedocs.io/en/latest/ (дата обращения: 04.03.2026).

55. Nguyen D., Lenharth A., Pingali K. A Lightweight Infrastructure for Graph Analytics // Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. 2013. Vol. 2013. P. 456–471. DOI: 10.1145/2517349.2522739.

56. Shun J., Blelloch G.E. Ligra: A Lightweight Graph Processing Framework for Shared Memory // Proceedings of the 18th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2013. Vol. 2013. P. 135–146. DOI: 10.1145/2442516.2442530.

57. Zhang K., Chen R., Chen H. NUMA-Aware Graph-Structured Analytics // Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2015. Vol. 2015. P. 183–193. DOI: 10.1145/2688500.2688507.

58. Sun J., Vandierendonck H., Nikolopoulos D.S. GraphGrind: Addressing Load Imbalance of Graph Partitioning // Proceedings of the International Conference on Supercomputing. 2017. Vol. 2017. P. 1–10. DOI: 10.1145/3079079.3079097.

59. Zhang Y., Brahmakshatriya A., Chen X. et al. Optimizing Ordered Graph Algorithms with GraphIt // Proceedings of the 18th ACM/IEEE International Symposium on Code Generation and Optimization. 2020. Vol. 2020. P. 158–170. DOI: 10.1145/3368826.3377909.

60. Vandierendonck H. Graptor: Efficient Pull and Push Style Vectorized Graph Processing // Proceedings of the 34th ACM International Conference on Supercomputing. 2020. Vol. 2020. P. 1–13. DOI: 10.1145/3392717.3392753.

61. Malewicz G., Austern M.H., Bik A.J.C. et al. Pregel: A System for Large-Scale Graph Processing // Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. 2010. Vol. 2010. P. 135–146. DOI: 10.1145/1807167.1807184.

62. Low Y., Gonzalez J.E., Kyrola A. et al. GraphLab: A New Framework for Parallel Machine Learning [Электронный ресурс] // CoRR. 2014. arXiv:1408.2041 (дата обращения: 17.06.2025).

63. Gonzalez J.E., Low Y., Gu H. et al. PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs // 10th USENIX Symposium on Operating Systems Design and Implementation. 2012. Vol. 2012. P. 17–30.

64. Kyrola A., Blelloch G., Guestrin C. GraphChi: Large-Scale Graph Computation on Just a PC // 10th USENIX Symposium on Operating Systems Design and Implementation. 2012. Vol. 2012. P. 31–46.

65. Roy A., Mihailovic I., Zwaenepoel W. X-Stream: Edge-Centric Graph Processing Using Streaming Partitions // Proceedings of the 24th ACM Symposium on Operating Systems Principles. 2013. Vol. 2013. P. 472–488. DOI: 10.1145/2517349.2522740.

66. Khorasani F., Vora K., Gupta R., Bhuyan L.N. CuSha: Vertex-Centric Graph Processing on GPUs // Proceedings of the 23rd International Symposium on High-Performance Parallel and Distributed Computing. 2014. Vol. 2014. P. 239–252. DOI: 10.1145/2600212.2600227.

67. Wang Y., Davidson A., Pan Y. et al. Gunrock: A High-Performance Graph Processing Library on the GPU // Proceedings of the 21st ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2016. Vol. 2016. P. 1–12. DOI: 10.1145/2851141.2851145.

68. Sengupta D., Song S.L., Agarwal K., Schwan K. GraphReduce: Processing Large-Scale Graphs on Accelerator-Based Systems // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2015. Vol. 2015. P. 1–12.

69. Han W., Mawhirter D., Wu B., Buland M. Graphie: Large-Scale Asynchronous Graph Traversals on Just a GPU // 26th International Conference on Parallel Architectures and Compilation Techniques. 2017. Vol. 2017. P. 233–245.

70. Valiant L.G. A Bridging Model for Parallel Computation // Communications of the ACM. 1990. Vol. 33, no. 8. P. 103–111. DOI: 10.1145/79173.79181.

71. Ho Q., Cipar J., Cui H. et al. More Effective Distributed ML via a Stale Synchronous Parallel Parameter Server // Advances in Neural Information Processing Systems. 2013. Vol. 2013. P. 1223–1231.

72. Naumov M., Vrielink A., Garland M. Parallel Depth-First Search for Directed Acyclic Graphs // Proceedings of the Seventh Workshop on Irregular Applications: Architectures and Algorithms. 2017. Vol. 2017. P. 1–8.

73. Meister C., Vieira T., Cotterell R. Best-First Beam Search // Transactions of the Association for Computational Linguistics. 2020. Vol. 8. P. 795–809.

74. Peng Z., Zhang M., Li K. et al. Speed-ann: Low-latency and high-accuracy nearest neighbor search via intra-query parallelism [Электронный ресурс] // CoRR. 2022. arXiv preprint arXiv:2201.13007 (дата обращения: 30.06.2025).

75. Linde Y., Buzo A., Gray R.M. An Algorithm for Vector Quantizer Design // IEEE Transactions on Communications. 1980. Vol. 28, no. 1. P. 84–95.

76. Zhang Z., Wu J. A Survey of Vector Quantization Techniques for Image Compression // Journal of Visual Communication and Image Representation. 2015. Vol. 33. P. 1–12.

77. Gersho A., Gray R.M. Vector Quantization and Signal Compression. Springer, 1992. Vol. 159. P. 720.

78. Makhoul J. Linear Prediction: A Tutorial Review // Proceedings of the IEEE. 1975. Vol. 63, no. 4. P. 561–580.

79. MacQueen J.B. Some Methods for Classification and Analysis of Multivariate Observations // Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability. 1967. Vol. 1. P. 281–297.

80. Ahmed M., Seraj R., Islam S.M.S. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation // Electronics. 2020. Vol. 9, no. 8. P. 1295. DOI: 10.3390/electronics9081295.

81. Golasowski M., Martinovič J., Slaninová K. Comparison of k-means clustering initialization approaches with brute-force initialization // Advanced Computing and Systems for Security. Advances in Intelligent Systems and Computing. Singapore: Springer, 2017. Vol. 567. P. 103–114.

82. Scalar Quantization-Based Text Encoding for Large Scale Image Retrieval // CEUR Workshop Proceedings. 2020. Vol. 2646. P. 258–265.

83. Veasey T., Trent B. Scalar quantization optimized for vector databases // Elastic Search Labs Blog [Электронный ресурс] URL: https://www.elastic.co/search-labs/blog/vector-db-optimized-scalar-quantization (дата обращения: 01.07.2025).

84. PINECONE. Faiss: The Missing Manual // Pinecone Learning Series [электронный ресурс] URL: https://www.pinecone.io/learn/series/faiss/vector-indexes/ (дата обращения: 01.07.2025).

85. Zilliz. Scalar Quantization and Product Quantization // Zilliz Learning [Электронный ресурс] URL: https://zilliz.com/learn/scalar-quantization-and-product-quantization (дата обращения: 01.07.2025).

86. Amato G., Carrara F., Falchi F., Gennaro C., Rabitti F., Vadicamo L. Scalar Quantization-Based Text Encoding for Large Scale Image Retrieval // Proceedings of the 2020 Italian Symposium on Advanced Database Systems (SEBD 2020). Online, 21–24 June 2020. Vol. 2646. P. 258–265.

87. Matsui Y., Uchida Y., Jégou H., Satoh S. [Invited Paper] A Survey of Product Quantization // ITE Transactions on Media Technology and Applications. 2018. Vol. 6, no. 1. P. 2–10. DOI: 10.3169/mta.6.2.

88. Babenko A., Lempitsky V. The inverted multi-index // IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Providence, RI, USA, 2012. P. 3069–3076. DOI: 10.1109/CVPR.2012.6248038.

89. Jordan J. Scaling nearest neighbors search with approximate methods // Jeremy Jordan Blog [Электронный ресурс] URL:https://www.jeremyjordan.me/scaling-nearest-neighbors-search-with-approximate-methods/ (дата обращения: 01.07.2025).

90. Microsoft. Choose an approach for optimizing vector storage and processing // Microsoft Learn [Электронный ресурс] URL: https://learn.microsoft.com/en-us/azure/search/vector-search-how-to-configure-compression-storage (дата обращения: 01.07.2025).

91. Муравьев С.Б., Казаковцев В.Л., Усов И.С. и др. An opensource library for AutoML multimodal clustering on Apache Spark // Записки научных семинаров Санкт-Петербургского отделения математического института им. В.А. Стеклова РАН. 2024. Т. 540. С. 178–193.

92. Казаковцев В.Л. Об операторе мутации в эволюционном алгоритме автоматической группировки // Системы управления и информационные технологии. 2022. Т. 88, № 2. С. 96–100.

93. Казаковцев В.Л. Комбинация жадной агломеративной эвристики и (1 + λ) алгоритма для задачи размещения // Системы управления и информационные технологии. 2024. № 1. С. 228–282.

94. Ахматшин Ф.Г., Насыров И.Р., Казаковцев В.Л., Казаковцев Л.А. О нормализации данных в задаче автоматической группировки промышленной продукции по однородным производственным партиям // Системы управления и информационные технологии. 2020. № 2. С. 86–89.

95. Рожнов И.П., Казаковцев В.Л. Реализация жадных эвристических алгоритмов кластеризации для массивно-параллельных систем // Системы управления и информационные технологии. 2019. № 2. С. 36–40.

96. Рожнов И.П., Казаковцев Л.А., Гудыма М.Н., Казаковцев В.Л. Алгоритм для задачи к-средних с рандомизированными чередующимися окрестностями // Системы управления и информационные технологии. 2018. № 3. С. 46–51.

97. Рожнов И.П., Орлов В.И., Гудыма М.Н., Казаковцев В.Л. Составление оптимальных ансамблей алгоритмов кластеризации // Системы управления и информационные технологии. 2018. № 2. С. 31–35.

98. Kazakovtsev L., Shkaberina G., Rozhnov I., Li R., Kazakovtsev V. Genetic algorithms with the crossover-like mutation operator for the k-means problem // Communications in Computer and Information Science (CCIS). 2020. P. 350–362.

99. Akiki T.J., Abdallah C.G. Determining the hierarchical architecture of the human brain using subject-level clustering of functional networks // Scientific Reports. 2019. Vol. 9, no. 1. Article number: 8245.

100. Celeux G., Govaert G. A classification EM algorithm for clustering and two stochastic versions // Computational Statistics & Data Analysis. 1992. Vol. 14, no. 3. P. 315–332.

101. Sheridan K., Puranik T. G., Mangortey E., et al. An application of dbscan clustering for flight anomaly detection during the approach phase // AIAA Scitech 2020 Forum. 2020. P. 1851.

102. Alp O., Erkut E., Drezner Z. An efficient genetic algorithm for the p-median problem // Annals of Operations Research. 2003. Vol. 122. P. 21–42.

103. Kazakovtsev L., Stashkov D., Gudyma M., Kazakovtsev V. Algorithms with greedy heuristic procedures for mixture probability distributions separation // Yugoslav Journal of Operations Research. 2019. Vol. 29, no. 1. P. 51–67.

104. Kazakovtsev V., Markushin E. Data segmentation through two-level clustering with greedy approach // ITM Web of Conferences : III International Workshop, Krasnoyarsk, 02–04 декабря 2024 года. Krasnoyarsk : EDP Sciences, 2025. P. 4007. DOI 10.1051/itmconf/20257204007.

105. Hansen P., Mladenović N. Variable neighborhood search // Search Methodologies. 2013. Vol. 2013. P. 313–337.

106. Kazakovtsev L., Rozhnov I., Kazakovtsev V. A (1 + λ) evolutionary algorithm with the greedy agglomerative mutation for p-median problems // AIP Conference Proceedings : Proceedings of the IV international scientific conference on advanced technologies in aerospace, mechanical and automation engineering: (MIST: Aerospace-IV 2021), Krasnoyarsk, December 10–11, 2021. Vol. 2700. AIP Publishing, 2023. P. 040003. DOI 10.1063/5.0124952.

107. Рожнов И. П., Орлов В. И., Гудыма М. Н., Казаковцев В. Л. Составление оптимальных ансамблей алгоритмов кластеризации // Системы управления и информационные технологии. 2018. № 2. С. 31–35.

108. Рожнов И. П., Казаковцев В. Л. Реализация жадных эвристических алгоритмов кластеризации для массивно-параллельных систем // Системы управления и информационные технологии. 2019. № 2. С. 36–40.

109. Насыров Р., Гудыма М. Н., Казаковцева О. Б., Казаковцев В. Л. Генетический алгоритм для кластеризации двумерных данных // Экономика и менеджмент систем управления. 2018. Т. 28, № 2-3. С. 399–408.

110. Steinhaus H. Sur la division des corps matériels en parties // Bulletin de l'Académie Polonaise des Sciences. 1956. Vol. 1, no. 804. P. 801.

111. Al Rahhal M. M., Bazi Y., Abdullah T. et al. Deep unsupervised embedding for remote sensing image retrieval using textual cues // Applied Sciences. 2020. Vol. 10, no. 24. P. 8931.

112. Baltrušaitis T., Ahuja C., Morency L.-P. Multimodal machine learning: A survey and taxonomy // IEEE Transactions on Pattern Analysis and Machine Intelligence. 2018. Vol. 41, no. 2. P. 423–443.

113. Chen C., Han D., Wang J. Multimodal encoder-decoder attention networks for visual question answering // IEEE Access. 2020. Vol. 8. P. 35662–35671.

114. Srivastava N., Salakhutdinov R. R. Multimodal learning with deep boltzmann machines //Advances in neural information processing systems. – 2012. – Vol. 25. P. 1-9.

115. Wang W., Ooi B. C., Yang X. et al. Effective multi-modal retrieval based on stacked auto-encoders // Proceedings of the VLDB Endowment. 2014. Vol. 7, no. 8. P. 649–660.

116. Лбов Г. С., Пестунова Т. М. Группировка объектов в пространстве разнотипных признаков // Анализ нечисловой информации в социологических исследованиях / под ред. В. Г. Андреенкова, А. И. Орлова, Ю. Н. Толстовой. Москва : Наука, 1985. С. 141–149.

117. Linde Y., Buzo A., Gray R. An Algorithm for Vector Quantizer Design // IEEE Transactions on Communications. 1980. Vol. 28. P. 84–95.

118. Elkan C. Using the triangle inequality to accelerate k-means // Proceedings of the 20th International Conference on Machine Learning (ICML-03), Washington, DC, USA, 21–24 August 2003. P. 147-153.

119. Almeida J. F. F., Campos F. C. C. A two-step planning method to increase accessibility to medium complexity procedures for public secondary healthcare // Ciência & Saúde Coletiva. 2021. Vol. 26. P. 4287–4298.

120. Ramadhanti N. S., Ridwan A. Y., Pambudi H. K. Feasibility study of determination a new distribution warehouse location using p-median and analytical network process methods in one of the cement industries // IOP Conference Series: Materials Science and Engineering. 2020. Vol. 982, no. 1. P. 012057.

121. Punj G., Stewart D. W. Cluster analysis in marketing research: Review and suggestions for application // Journal of Marketing Research. 1983. Vol. 20, no. 2. P. 134–148.

122. Farseev A., Samborskii I., Filchenkov A., Chua, T. S. Cross-domain recommendation via clustering on multi-layer graphs // Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval. 2017. Vol. 2017. P. 195–204.

123. Yin G., Rudolph G., Schwefel H. P. Analyzing the (1, λ) evolution strategy via stochastic approximation methods // Evolutionary Computation. 1995. Vol. 3, no. 4. P. 473–489.

124. Kazakovtsev L., Rozhnov I., Shkaberina G. Self-configuring (1+1)-evolutionary algorithm for the continuous p-median problem with agglomerative mutation // Algorithms. 2021. Vol. 14, no. 5. P. 130.

125. Tomp D., Muravyov S., Filchenkov A., Parfenov V. Meta-learning based evolutionary clustering algorithm // International Conference on Intelligent Data Engineering and Automated Learning. Cham : Springer International Publishing, 2019. Vol. 2019. P. 502–513.

126. Leung F. H. F., Lam H. K., Ling S. H., Tam P. K. S. Tuning of the structure and parameters of a neural network using an improved genetic algorithm // IEEE Transactions on Neural Networks. 2003. Vol. 14, no. 1. P. 79–88.

127. Cohen-Shapira N., Rokach L. Automatic selection of clustering algorithms using supervised graph embedding // Information Sciences. 2021. Vol. 577. P. 824–851.

128. Shalamov V., Efimova V., Muravyov S., Filchenkov A. Reinforcement-based method for simultaneous clustering algorithm selection and its hyperparameters optimization // Procedia Computer Science. 2018. Vol. 136. P. 144–153.

129. Kazakovtsev L., Rozhnov I., Shkaberina G., Orlov V. K-Means Genetic Algorithms with Greedy Genetic Operators // Mathematical Problems in Engineering. 2020. Vol. 2020, no. 1. P. 8839763. DOI: 10.1155/2020/8839763.

130. Kazakovtsev V., Oreshin S., Serdyukov A. et al. Recommender system for an academic supervisor with a matrix normalization approach // Proceedings of the

2020 1st International Conference on Control, Robotics and Intelligent System. 2020. Vol. 2020. P. 84–87.

131. Ghazal T. M. Performances of k-means clustering algorithm with different distance metrics //Intelligent Automation & Soft Computing. – 2021. – Vol. 30. no. 2. – P. 735-742.

132. Kapoor A., Singhal A. A comparative study of K-Means, K-Means++ and Fuzzy C-Means clustering algorithms // 2017 3rd International Conference on Computational Intelligence & Communication Technology (CICT). IEEE, 2017. Vol. 2017. P. 1–6.

133. Doerr B., Gießen C., Witt C., Yang J. The (1+λ) evolutionary algorithm with self-adjusting mutation rate // Proceedings of the Genetic and Evolutionary Computation Conference. 2017. Vol. 2017. P. 1351–1358.

134. Buzdalov M., Doerr C. Optimal Mutation Rates for the EA on OneMax // International Conference on Parallel Problem Solving from Nature. Cham : Springer International Publishing, 2020. Vol. 2020. P. 574–587.

135. Neema M. N., Maniruzzaman K. M., Ohgai A. New genetic algorithms based approaches to continuous p-median problem // Networks and Spatial Economics. 2011. Vol. 11, no. 1. P. 83–99.

136. Kazakovtsev L. A., Antamoshkin A. N. Genetic algorithm with fast greedy heuristic for clustering and location problems // Informatica. 2014. Vol. 38, no. 3. P. 229-240.

137. Clustering Basic Benchmark [Электронный ресурс]. URL: http://cs.joensuu.fi/sipu/datasets/ (дата обращения: 25.09.2021).

138. Dua D., Graff C. UCI Machine Learning Repository. 2019. [Электронный ресурс] URL: http://archive.ics.uci.edu/ml (дата обращения: 30.09.2025).

139. Fränti P., Sieranoja S. K-means properties on six clustering benchmark datasets // Applied Intelligence. 2018. Vol. 48, no. 12. P. 4743–4759.

140. Rozhnov I. P., Orlov V. I., Kazakovtsev L. A. VNS-based algorithms for the centroid-based clustering problem // Facta Universitatis, Series: Mathematics and Informatics. 2019. P. 957–972.

141. Maulik U., Bandyopadhyay S. Genetic algorithm-based clustering technique // Pattern Recognition. 2000. Vol. 33, no. 9. P. 1455–1465.

142. Page E. S. On Monte Carlo Methods in Congestion Problems: I. Searching for an Optimum in Discrete Situations // Operations Research. 1965. Vol. 13, no. 2. P. 291–299.

143. Reese J. Solution methods for the p-median problem: An annotated bibliography // NETWORKS: an International Journal. 2006. Vol. 48, no. 3. P. 125–142.

144. Казаковцев В.Л. Multi Group Encoding. Свидетельство о государственной регистрации программы для ЭВМ / Казаковцев В.Л., Ступина А.А., Казаковцев Л.А. // Роспатент. рег. No 2026618245 от 24.03.2026. заявл. 17.02.2026.

145. Kazakovtsev V., Muravyov S. Application of the automatic selection and configuration of clustering algorithms method for the Apache Spark framework // ACM International Conference Proceeding Series. 2021. Vol. 2021 P. 1-5.

146. Filchenkov A., Muravyov S., Parfenov V. Towards cluster validity index evaluation and selection // Proceedings of the 2016 IEEE Artificial Intelligence and Natural Language Conference (AINL). IEEE, 2016. Vol. 2015. P. 1–8.

147. Vanschoren J., van Rijn J. N., Bischl B., Torgo L. OpenML: Networked science in machine learning [Электронный ресурс] //CoRR. 2014. arXiv:1407.7722 (дата обращения: 30.06.2025).

148. Akiba T., Sano S., Yanase T. et al. Optuna: A next-generation hyperparameter optimization framework // Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. 2019. Vol. 2019 P. 2623-2631.

149. Salloum S., Dautov R., Chen X. et al. Big data analytics on Apache Spark //International Journal of Data Science and Analytics. 2016. Vol. 1. no. 3. P. 145-164.

150. Theodorakopoulos L., Karras A., Krimpas G. A. Optimizing apache spark MLlib: Predictive performance of large-scale models for big data analytics //Algorithms. 2025. Vol. 18. no. 2. P. 74.

ПРИЛОЖЕНИЕ А

АКТ О ВНЕДРЕНИИ РЕЗУЛЬТАТОВ ДИССЕРТАЦИИ

КРАСЭКО
АКЦИОНЕРНОЕ ОБЩЕСТВО
«КРАСНОЯРСКАЯ РЕГИОНАЛЬНАЯ
ЭНЕРГЕТИЧЕСКАЯ КОМПАНИЯ»

Мира пр., д. 10, Красноярск, 660049
телефон (391) 228-62-07, 228-62-24
e-mail: mail@kraseco24.ru
сайт: www.красэко24.рф

ОГРН 1152468001773 / ОКПО 75795891
ИНН 2460087269 / КПП 246601001

8 апреля 2026 год.

АКТ

о внедрении результатов диссертационного исследования

Казаковцева В.Л.

Настоящим подтверждается, что разработанный Казаковцевым В.Л. алгоритм адаптивного поиска ближайших соседей используется в деятельности АО «КрасЭКо» в составе системы управления расчетами с потребителями. Применение алгоритма адаптивного поиска по файлу обратного индекса, предложенного в рамках диссертационного исследования на соискание ученой степени кандидата технических наук на тему: «Алгоритмы ускоренного поиска в векторных базах данных» Казаковцева Владимира Львовича, позволило выявлять потенциальных должников на основе собранных данных об уже имеющихся задолженностях у других потребмтелей и рекомендовать частоту и содержание оповещений о задолженностях. Полученные результаты при практическом применении в АО «КрасЭКо» подтверждает значимость разработанных алгоритмов ускоренного поиска в векторных базах данных, предложенных в рамках диссертационного исследования Казаковцева В.Л.

Директор Цепков А.В.

ПРИЛОЖЕНИЕ Б

АКТ О ВНЕДРЕНИИ РЕЗУЛЬТАТОВ ДИССЕРТАЦИИ

Актив Туим

unaaktiv@gmail.com

Сокращенное наименование	ООО «Актив Туим»
ИНН	2460123911
КПП	246001001
ОГРН	1232400018971
Адрес места нахождения	660021, Красноярский край, город Красноярск, улица Республики, д. 51, стр. 1, помещ. 4, ком. 37
Банковские реквизиты	р/с 40702810401100006606 ББР Банк (АО) к/с 30101810745250000769 в ГУ Банка России по ЦФО БИК 044525769

АКТ
об внедрении результатов диссертационного исследования
Казаковцева Владимира Львовича

Настоящим подтверждаем, что предложенная Казаковцевым В.Л. модель автоматической группировки (кластеризации) мультимодальных данных была успешно внедрена в составе рекомендательной системы для выбора проектов малых архитектурных форм, описываемых текстовой информацией и числовыми характеристиками. Анализ действий пользователей рекомендательной системы показал, что пользователи в среднем на 19% переходят по предлагаемым новой рекомендательной системой ссылкам в сравнении с действовавшей до ее внедрения системой, рассматривавшей числовые данные в качестве составной части текстового описания.

Директор ООО «УНА»
управляющей организации
ООО «Актив Туим» Переверзев Д.В.