# Приближенное индексирование многомерных объектов ♣

© Е.Г. Михайлова, Б.А. Новиков

Санкт-Петербургский государственный университет
egmichailova@mail.ru, borisnov@acm.org

## Аннотация

Задачи поиска ближайших соседей в многомерном пространстве возникают во многих задачах информационного поиска, обработки текстов на естественном языке, в логическом анализе данных. Эффективное решение этой задачи возможно только с помощью специализированных индексных структур, однако точные методы неприменимы для пространств большой размерности. В работе предлагается и анализируется индексная структура для приближенного решения задачи поиска K ближайших соседей, основанная на использовании кластеризации для построения индексного дерева. Реализация построена над высокопроизводительной реляционной СУБД.

## 1 Введение

Любая система, для функционирования которой требуются хранение и обработка больших объемов информации, по необходимости должна так или иначе решать задачу поиска. Как правило, для выполнения отдельных функций требуется относительно небольшая часть информации, хранимой в системе, и поэтому эффективная работа системы обеспечивается с помощью индексных структур. Любой рассматриваемый процесс или объект, информация о котором хранится, может быть характеризован набором признаков, которые обычно называются атрибутами. Назначение любого индекса – быстрое получение доступа к объекту по значениям некоторых из его атрибутов. Иначе говоря, индексы обеспечивают эффективное выполнение ассоциативного поиска.

Если значения атрибута принадлежат некоторому линейно упорядоченному множеству, то задачи поиска решаются с помощью одномерных индексов. Одномерные индексные структуры (такие, как *B*-деревья и *hash*-файлы) хорошо исследованы в 1970 – 1980 гг. Во многих приложениях, однако, требуется поиск по значениям нескольких атрибутов или по атрибутам, которые не могут быть естественным образом линейно упорядочены, что приводит к необходимости решать задачу поиска в многомерном пространстве. Из многочисленных многомерных индексных структур, предложенных в 1980-е гг., проверку временем выдержали только R-деверья [1], широко применяемые в различных приложениях и реализованные в промышленных СУБД.

Задачи многомерного поиска связаны, в первую очередь, с обработкой пространственных и пространственно-временных данных. Другим источником многомерных данных является широкий класс приложений, в которых объекты моделируются векторами, составленными из значений некоторых вычисляемых признаков или характеристик объекта. К этому классу приложений относятся системы, основанные на различных вариантах векторной модели текстового информационного поиска, поиск изображений по содержанию, некоторые методы логического анализа данных и многие другие. Данные, полученные из приложений этого класса, обычно характеризуются большой размерностью.

Как правило, при обработке многомерных данных требуется поиск по признаку близости или подобия, а не на точное совпадение каких-либо атрибутов. Наиболее часто упоминаются задачи поиска K ближайших соседей, поисков заданной окрестности и операция соединения на основе подобия.

## 2 Обзор родственных работ

### 2.1 Размер страницы

В исследовательской литературе предложено большое количество индексных структур для многомерных данных, в том числе *R*-деревья [8], *X*-деревья [2], *AV*-файлы и многие другие [3]. В последнее десятилетие большое внимание привлекает схема пространственно-чувствительного хеширования (*LSH* – locality sensitive hashing). Рассматривались также *M*-деревья [4], предназначенные для индексирования точек метрического пространства.

Для пространственных данных малой размерности хорошо работают индексы на основе *R*-деревьев [8]. Задача многомерного поиска чрезвычайно сложна при большом размере векторов. Можно даже утверждать, что построение универсальных индексных структур для данных большой размерности невозможно. Этот факт, известный под названием «проклятие размерности», определяется топологией многомерного пространства и не зависит от метода построения индекса. Практическое следствие состоит в том, что любая индексная структура, начиная с

**Труды 12[й] Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» – RCDL'2010, Казань, Россия, 2010**

некоторой размерности, становится по производительности хуже, чем полный просмотр. Остроту этой проблемы можно несколько снизить, рассматривая структуры индексов и алгоритмы, дающие приближенный результат. Как правило, векторы только приблизительно отражают близость исходных объектов приложения (например, векторы в задачах информационного поиска не могут точно выражать смысл документов), поэтому использование приближенных методов поиска допустимо. Так, например, *LSH* [6] дает только вероятностные гарантии решения задачи *K-nn* – поиска ближайших соседей [9, 5].

Во многих случаях вычислительно трудоемкими оказываются не только поиск и подготовка векторов (например, извлечение признаков изображений), но даже вычисление функции подобия (или расстояния между векторами). Поиск мог бы быть существенно ускорен, если бы можно было хранить матрицу попарных расстояний между объектами. К сожалению, такое решение не является масштабируемым, поскольку размер индекса в этом случае квадратично зависел бы от количества объектов.

Цель данной работы – построение субоптимальной индексной структуры для приближенного поиска многомерных объектов на основе функции подобия (или функции расстояния), которая бы обеспечивала приемлемое время ответа в практических важных диапазонах различных параметров. Для того чтобы оценить характеристики и качество этой структуры, мы реализовали ее модель над структурами данных, предоставляемыми промышленной реляционной СУБД. Возможно, такое решение отрицательно сказывается на эффективности, но обеспечивает быструю реализацию, пригодную для использования в прототипах, например, в экспериментальной системе поиска изображений по содержанию, т. е. разработанная система позволяет находить изображения по ряду характеристик, близких к искомому.

В предположении, что имеется доступная матрица попарных расстояний, задачу поиска K ближайших соседей можно решать просмотром одномерного индекса по диапазону. Эта операция эффективно реализована в любой реляционной СУБД. Для того чтобы ограничить размер такого индекса, мы будем хранить не все попарные расстояния, а только расстояния до *K* ближайших точек. Как правило, в поисковых системах необходимо решать задачу *K-nn* только при небольших значениях *K*.

Для того чтобы обеспечить поиск ближайших соседей для произвольной точки в многомерном пространстве, мы строим древовидную структуру следующим образом: имеющийся набор векторов кластеризуется и выбираются центроиды каждого кластера. Эти центроиды образуют множество векторов меньшего размера, чем исходной, и образуют второй уровень нашей структуры. Далее кластеризация и выделение центроидов повторяются до тех пор, пока количество выделенных центроидов не окажется достаточно малым. Множество этих центроидов образует верхний уровень индексного дерева.

Кластеризация ранее использовалась для оптимальной реорганизации *R*-дерева [7].

Для решения задачи поиска ближайших соседей необходимо вычислить расстояния от центра, заданного поисковым запросом, до всех точек верхнего уровня и выбрать из них несколько ближайших. Далее с помощью предварительно построенной матрицы расстояний можно выбрать точки следующего уровня, ближайшие к выбранным, возможно, сократить это множество и перейти на следующий уровень.

Чтобы облегчить задачу многомерного поиска, необходимо использовать алгоритмы кластеризации, или деления на классы. Известные методы относят к одному классу объекты, многомерные векторы которых в пространстве признаков являются соседними.

## 3 Структуры данных и алгоритмы

### 3.1 Структура данных

Для представления объектов будем использовать реляционную модель. Рассматриваем объекты как многомерные векторы. Координаты многомерного вектора являются различными характеристиками изображений. Предполагается, что похожие изображения будут характеризоваться близкими векторами.

Для хранения будем использовать таблицы Координат, Векторов, Расстояний и Компонент.

В таблице Координат хранится информация о значениях каждой координаты вектора. Чтобы структура таблицы не зависела от размерности векторов, каждый вектор представляется в таблице набором строк – по одной строке для каждой координаты вектора.

Каждая строка таблицы имеет следующую структуру: номер вектора; номер координаты вектора; численное значение координаты вектора.

Между всеми парами векторов необходимо вычислить расстояние. В разных приложениях можно использовать для этого разные функции. В данном случае расстояние вычисляется с помощью метрики *L1*, но для метода это не принципиально.

Из полученных расстояний строится таблица Расстояний, в каждой строке которой хранится расстояние между двумя векторами.

Итак, получилась таблица с атрибутами:

1. Номер вектора_1,
2. Номер вектора_2,
3. Расстояние между векторами,
4. Номер компоненты связности, которой принадлежат оба вектора, если вектор является частью остовного дерева для этой компоненты.

Если вектора относятся к разным компонентам связности, то номер компоненты в соответствующей строке будет равен нулю.

Эта таблица будет очень большой. Она необходима только на этапе построения индекса. В даль-

нейшем можно хранить не все попарные расстояния между векторами, а только несколько ближайших векторов к каждому и расстояние до него.

Таблица Векторов имеет следующую структуру:

1. номер вектора,
2. номер компоненты связности, которой этот вектор принадлежит,
3. количество векторов нижнего уровня, для которых данный вектор является центром компоненты связности,
4. расстояние до центра связности текущего уровня.

Структура таблицы Компонент:

1. номер компоненты связности,
2. уровень,
3. количество векторов, относящихся к этой компоненте,
4. номер вектора-центра этой компоненты.

### 3.2 Построение индексной структуры

Чтобы при поиске избежать необходимости полного просмотра, необходимо построить иерархическую поисковую структуру. Для построения этой древовидной структуры будем использовать метод кластеризации. Для кластеризации был реализован метод, основанный на построении остовного дерева минимальной длины.

Для построения остовного дерева используется следующий алгоритм:

1. полагаем многомерные вектора вершинами дерева, а расстояние между векторами – дугами;
2. на нижнем уровне полагаем каждую вершину отдельной компонентой связности;
3. находим две компоненты связности, расстояние между которыми минимально; за расстояние между компонентами связности полагаем минимальное расстояние между вершинами, принадлежащим соответствующим компонентам;
4. добавляем в остовное дерево дугу между этими двумя вершинами;
5. соединяем эти компоненты связности в одну;
6. продолжаем п.2 – п.4 до тех пор, пока все вершины не войдут в одну компоненту связности.

При построении дерева для наибольшей равномерности вводилось ограничение на число входящих дуг для каждой вершины – не больше $N_{in}$.

Далее разбиваем построенное остовное дерево на несколько компонент связности. В каждой компоненте связности должно быть от $N_{min}$ до $N_{max}$ вершин. Количество компонент зависит от количества векторов, хранящихся в таблице, и определяется.

При разбиении остовного дерева на компоненты связности из дуг, входящих в остовное дерево, выбираем дугу, имеющую максимальную длину. Эту дугу исключаем из остовного дерева, при этом образуются 2 компонент связности. Проверяем, что количество векторов в образовавшихся компонентах больше $N_{min}$. Если нет, то возвращаем удаленную дугу обратно, пометив ее. Таким образом перебираем вектора остовного дерева, пока не удастся разбить компоненту связности на две части. Затем разбиваем каждую новую часть, пока размер компонент не окажется между $N_{min}$ до $N_{max}$.

В каждой компоненте связности находим центроиды. Для нахождения *i*-ой компоненты центроида находим усредненного значение *i*-х компонент векторов, входящих в заданную компоненту связности.

Далее рассматриваем найденные центроиды в качестве многомерных векторов.

Далее повторяем наш алгоритм: вычисляем попарные расстояния между векторами, строим остовное дерево, вновь разбиваем его на компоненты связности. Со второй итерации при нахождении центроидов новых компонент связности учитываем «вес» центроидов предыдущего уровня – количество векторов следующего уровня, объединенных этим центроидом.

Так мы повторяем все шаги алгоритма, на каждой итерации выбирая в качестве новых векторов центры полученных компонент связности, пока количество этих компонент не станет равным или меньшим заданного *M*. В результате получается некая многоуровневая иерархическая структура, которую можно использовать для поиска n векторов, ближайших к искомому вектору $v_1$.

### 3.3 Поиск

Для организации поиска надо начать с верхнего уровня, определить вектор $v_2$, ближайший к искомому $v_1$. Затем рассмотреть компоненту связности, для которой вектор $v_2$ является центром. Если количество векторов в этой компоненте связности больше искомого *n*, то можно рассматривать только эту компоненту, если меньше *n*, то ищем следующую ближайшую компоненту связности. От центра спускаемся к векторам данной компоненты следующего уровня и так далее, пока не доходим до нижнего уровня.

При вставке нового вектора сначала производится поиск одного ближайшего, затем новый вектор добавляют в ту же компоненту, где находится ближайший вектор. Если количество векторов в этой компоненте связности окажется больше максимально допустимого, то компонента делится на две. Так же для каждого следующего уровня.

## 4 Эксперимент и анализ результатов

Описанная модель была реализована на векторах, координаты которых описывают различные характеристики изображений: яркость, контрастность, насыщенность, и т. д.

Ставилась задача быстрого поиска векторов, наиболее близких к искомому вектору.

Задачей экспериментального исследования было также определение параметров, при которых данный метод может давать наилучшие результаты. Параметрами данного метода, которые можно изменять, являются:

– максимальное количество векторов, входящих в каждую вершину остовного дерева;

– минимальное и максимальное количество векторов, входящих в каждую компоненту связности, на которые на каждом шаге построения иерархической структуры делится остовное дерево;

– количество найденных векторов, ближайших к искомому, которые мы ищем;

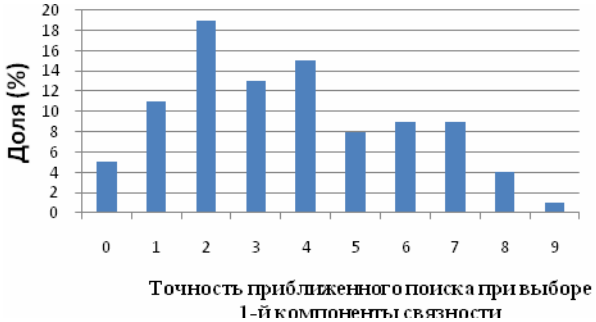
– количество компонент, центры которых наиболее близки к искомому вектору, которые просматриваются на каждом шаге.

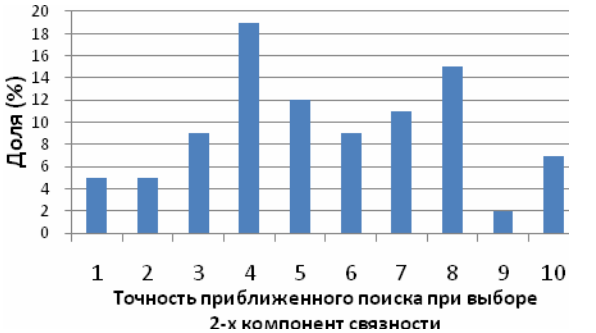
Индексная структура была построена и протестирована над несколькими тысячами векторов, характеризующих изображения. Вектора имеют 41 координату.

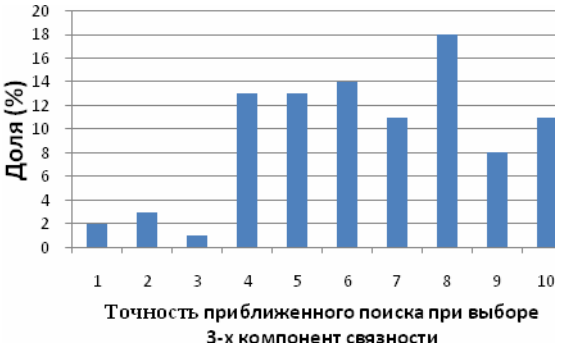
Скорость поиска даже при таком небольшом количестве векторов оказалась в десятки раз меньше, чем при линейном просмотре.

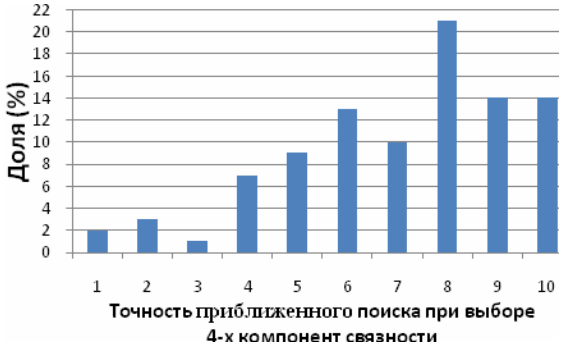
При разбиении дерева на компоненты связности максимальное количество векторов подбиралось так, чтобы оно превосходило минимальное количество векторов в компоненте в 3 – 4 раза. А минимальное количество векторов в компоненте, должно, в свою очередь, как минимум в 2 раза превосходить максимальное количество дуг, входящих в каждую вершину остовного дерева.

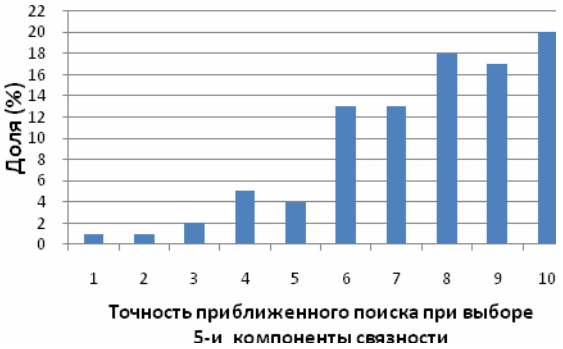
Следующие графики показывают полученную точность результатов приближенного поиска в зависимости от количества компонент связности, просматриваемых на каждом шаге.




Приведенные гистограммы показывают, что оптимально на каждом шаге просматривать 3 – 4 компоненты.

Скорость поиска с помощью построенной индексной структуры в десятки раз превышала скорость линейного поиска.

## 5 Заключение

Индексирование многомерных объектов является трудной задачей, которая привлекает внимание исследователей на протяжении нескольких десятилетий.

В этой работы мы предлагаем практическую реализацию индексной структуры для многомерных объектов в решении задачи поиска *k* ближайших соседей. Мы предлагаем критерии оценки эффективности и предлагаем параметры, от которых может зависеть производительность предложенной индексной структуры.

## Литература


[1] Arge L., de Berg M., Haverkort H.J., Yi Ke. The priority RTree: a practically efficient and WorstCase optimal RTree.

[2] Berchtold S., Keim D.A., Kriegei H.-P. The X-tree: an index structure for high-dimensional data// Proc. of the 22nd Int. Conf. on Very Large Databases, 1996. – P. 28-39.

[3] Bremner D., Demaine E., Erickson J., Iacono J., Langerman S., Morin P., Toussaint G. Output-sensitive algorithms for computing nearest-neighbor decision boundaries// Discrete and Computational Geometry. – 2005. – V. 33, No 4. – P. 593-604.

[4] Ciaccia et al. M-tree: an efficient access method for similarity search in metric spaces// VLDB-1997, 1997.

[5] Cover Th.M., Hart P.E. Nearest neighbor pattern classification// IEEE Transactions on Information Theory. – 1967. – V. 13, No 1. – P. 21-27.

[6] Gionis A., Indyk P., Motwani R. Similarity search in high dimensions via hashing// Proc. of the 25th Very Large Database (VLDB) Conf., 1999.

[7] Guttman A. R-Trees: a dynamic index structure for spatial searching// SIGMOD Conference, 1984. – P. 47-57.

[8] Manolopoulos Y., Nanopoulos A., Papadopoulos A.N., Theodoridis Y. R-Trees: theory and applications. – Springer, 2005.

[9] Nigsch F.A., Bender A., van Buuren B., Tissen J., Nigsch E., Mitchell J.B.O. Melting point prediction employing k-nearest neighbor algorithms and genetic parameter optimization// J. of Chemical Information and Modeling. – 2006. – V. 46, No 6. – P. 2412–2422. – doi:10.1021/ci060149f.


## Approximate indexing for multidimensional objects


E. Mikhaylova, B. Nivikov



The multidimensional k nearest neighbor problem is essential for several application areas, including information retrieval, natural language processing, and data mining. To solve this problem efficiently, one needs an index structure supporting this kind of search. However, exact solution is hardly feasible in multidimensional space.

In this paper we describe and analyze an indexing technique for approximate solution of the k-nn problem. The construction of the index tree is based on clustering. The index is implemented on top of high-performance industrial DBMS.



♣Работа выполнена при финансовой поддержке РФФИ (проект 10-07-00156)