ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»
Факультет компьютерных наук
Департамент программной инженерии

Отчет
по курсовой работе

на тему «Исследование алгоритмов приближенного поиска ближайших соседей»
по направлению подготовки «Программная инженерия»

Выполнил
студент группы 301 ПИ
образовательной программы
«Программная инженерия»
Е. А. Кудряшов

_____

Научный руководитель
проф, д-р техн. наук
М. В. Ульянов
Оценка: _____

_____

Москва
2015

Реферат

В соответствии с приказом Национального исследовательского университета «Высшая школа экономики» № 6.18.1-02/1912-09 от 19.12.2014 была произведена исследовательская работа на тему «Исследование алгоритмов приближенного поиска ближайших соседей». Данный отчет содержит итоги проведения данной курсовой работы.

В ходе проекта проводилось изучение алгоритмов приближенного поиска ближайших соседей, выбор входных данных, определение метрик, для сравнения алгоритмов, а также конечный анализ исследованных алгоритмов.

Были выбраны наиболее популярные типы алгоритмов для поиска. Входные данные выбирались исходя из разумных рассуждений об из разнородности и различности между собой.

В ходе испытаний алгоритмов были получены результаты их работы на различных входных данных, а также были построены графики по данным результатам (см. Приложение А, Приложение Б).

Был произведен сравнительный анализ данных, основанный на специфике алгоритмов, особенностей реализации и их работе на различных входных данных, что помогло объяснить получившиеся результаты.

В заключении приведены рекомендации по использованию алгоритмов поиска приближенных ближайших соседей на различных входных данных.

Содержание

Нормативные ссылки

В настоящем стандарте использованы ссылки на следующие стандарты:

ГОСТ 1.5–93 Государственная система стандартизации РФ. Общие требования к построению, изложению, оформление и содержанию стандартов.

ГОСТ 2.105–95 Единая система конструкторской документации. Общие требования к текстовым документам.

Определения

Введем базовые определения:

– Точка запроса - такая точка, для которой необходимо найти ближайшего соседа из множества точек S, будет обозначаться $\mathbf{q}$. $\mathbf{q} \in \mathbf{R}^d$.

– Истинный ближайший сосед или ближайший сосед. Пусть дано множество S из n точек, векторное пространство $V = \mathbb{R}^d$ с определенной нормой $||\cdot||$ и точка запроса $\mathbf{q} \in V$, тогда точка $\mathbf{p}$ истинный ближайший сосед точки $\mathbf{q}$ тогда и только тогда, когда

$$d(\mathbf{p}, \mathbf{q}) = ||\mathbf{p} - \mathbf{q}|| = \min\{||\mathbf{p}' - \mathbf{q}||; \mathbf{p}' \in S \setminus \mathbf{q}\}. \quad (1)$$

– Пусть дано множество S из n точек, векторное пространство $V = \mathbb{R}^d$ с определенной нормой $||\cdot||$ и точка запроса $\mathbf{q} \in V$, тогда точка $\mathbf{p}$ приближенный ближайший сосед точки $\mathbf{q}$, если

$$d(\mathbf{p}, \mathbf{q}) = (1 + \epsilon)||\mathbf{p}^* - \mathbf{q}||, \epsilon > 0, \quad (2)$$

где $\mathbf{p}^*$ истинный ближайший сосед точки $\mathbf{q}$.

– KD-дерево. Дерево представляющее такую структуру данных, которая разбивает пространство таким образом, что происходит упорядочивание точек в k-мерном пространстве.

– Хеш-функция. Функция, преобразующая определенным образом входной массив данных произвольной длины в выходную битовой строку фиксированной длины.

Обозначения и сокращения

NN — Nearest neighbor – ближайший сосед.

ANN — Approximate Nearest Neighbor – приближенный ближайший сосед.

LSH — Locality-Sensitive Hashing – хеширование чувствительное к расстоянию.

Введение

Задача эффективного по времени поиска ближайшего соседа состоит в создании такой структуры из множества точек S, в которой можно быстро найти ближайшего соседа из множества S для данной точки запроса q. Данная задача имеет множество применений, таких как: поиск последовательностей белка в молекулярной биологии [1], многомерные интерполяции [2], интеллектуальный анализ данных [3], в медицинском компьютерном диагностировании [4], построение NN-классификатора в распознавании образов [5, 6], фрактальное сжатие изображений [7]. Задача также имеет название задачи почтальона [8]. Данная проблема обычно описывается в векторном пространстве с определенной метрикой (Эвклидовой l_2 или Чебышёва l_∞), хотя в некоторых приложениях используется метрика Левенштейна, например, для расчета расстояния между строками.

Предполагается, что многочисленные запросы должны быть исполнены для одного и того же множества S, следовательно, сначала необходима предварительная обработка множества S, чтобы в дальнейшем быстро (имеется ввиду за время меньшее линейного) получить ответ для последовательности точек запроса.

Поскольку задача поиска истинного ближайшего соседа часто затратна по времени для многих приложений, особенно для высоко размерных пространств и большого количества точек, задача поиска приближенного ближайшего соседа, в большинстве случаев, является хорошей заменой. Она не даёт гарантий нахождения истинного ближайшего соседа, а пытается найти точку из множества S, которая будет «достаточно близка». Близость точек может быть описана в терминах расстояния между точками: необходимо найти точку, расстояние до которой от точки запроса не будет превышать $(1 + \epsilon)r$, где r расстояние до истинного ближайшего соседа. Здесь мы ослабим требования тем, что приближение является частью предметной области и зависимой от неё ошибки.

Задача поиска ближайшего соседа в малоразмерном векторном пространстве хорошо изучена. Для $d = 1$, асимптотически оптимальное решение состоит в сортировке исходного множества точек и применении бинарного поиска для поиска ближайшего соседа. Для $d = 2$, асимптотически оптимальное решение ведёт к диаграмме Вороного [9, 10], что на практике является медленным решением, так как имеет большую константу перед оценкой. Существует большое количество алгоритмов, которые основываются на обработке каждого измерения отдельно, в результате чего уменьшается множество потенциальных ближайших соседей и поиск заканчивается, когда остаётся один [11, 12]. К сожалению, данные стратегии применимы только к пространствам невысокой размерности и распределению точек близкому к равномерному.

В данной научно-исследовательской работе стоит цель исследовать характерных представителей двух основных типов алгоритмов приближенного поиска ближайших соседей (в сравнении с алгоритмом полного перебора): алгоритм, использующий деревья и алгоритм, использующий локальный поиск. Будет рассмотрен алгоритм, основанный на бинарном дереве. Способ использования деревьев в качестве разделение пространства хорошо зарекомендовал се-

бя в поиске ближайших соседей в многомерных пространствах [13, 14]. Вторым сравниваемым алгоритмом является алгоритм использующий хеш-функции. Алгоритм заключается в подборе таких хеш-функций, которые с определенной вероятностью помещают похожие объекты в одну корзину, под корзиной понимается некий список точек, для которых значение хеш-функции одинаково. Таким образом происходит некое понижение размерности пространства, которое позволяет эффективно искать приближенных ближайших соседей. В работе не рассматриваются алгоритмы, которые основаны на дискретном метрическом пространстве, а также алгоритмы, используемые для поиска среди очень большого количества данных, когда используется активность чтения и записи с жесткого диска, так как данный объём информации не удаётся хранить в оперативной памяти. Алгоритмы, использующие чтение и запись на жесткий диск опущены из-за отсутствия их эффективности на данных небольшого размера ($n \leqslant 10^6$).

Данная работа структурирована следующим образом: сначала будут описаны алгоритмы, используемые при сравнении, после этого следует описание входных данных, на которых тестировались алгоритмы, характеристики сравнения, используемые для сравнения алгоритмов, далее представлены результаты испытаний, сравнительный анализ алгоритмов и в заключении будут приведены выводы данной работы, основанные на результатах.

1 Описание алгоритмов

В данной главе будут приведены описания алгоритмов приближенного поиска ближайших соседей. Сначала последует описание алгоритма, основанного на KD-деревьях, а затем алгоритма, основанного на локальном хешировании.

1.1 Approximate Best Bin First *k*-d Tree

В качестве алгоритма, основанного на деревьях, был выбран алгоритм Approximate Best Bin First k-d Tree [15]. В его основе лежат k-d деревья, и состоит он из двух частей:

а) Функция BuildTree, которая на вход принимает множество S из n точек из $\mathbb{R}^d$ и создаёт расширенное k-d дерево T.

б) Функция ANN, которая на вход принимает дерево T и точку запроса $\mathbf{q}$ и возвращает приближенного ближайшего соседа.

1.1.1 Расширенное *k*-d дерево

K-d дерево хранит точки в своих листьях. Каждый узел дерева представляет из себя гиперинтервал (одна из осей прямоугольного гиперпараллелепипеда), который в дальнейшем будет называться внешней границей блока (LBB от англ. Loose Bounding Box). LBB может быть конечным, бесконечным или частично бесконечным. Внутри данного гиперпараллелепипеда хранятся все точки поддеревьев данного узла. LBB хранится явно в каждом узле дерева и строится рекурсивно: внешняя граница для корня дерева это $\mathbb{R}^d$. Далее, каждый узел дерева хранит информацию о том, по какому измерению m произошло разделение и значение ξ, которое определяет перпендикулярную гиперплоскость. Внешняя граница блока для левого и правого поддерева запишется следующим образом:

$$LBB_L = \{x_m < \xi; x \in LBB_P\}, \tag{3}$$

$$LBB_R = \{x_m \geqslant \xi; x \in LBB_P\}, \tag{4}$$

где LBB_P внешняя граница блока родителя узла, а x_m - m-компонента вектора $\mathbf{x}$. Также каждый узел хранит внутренний граничный блок (TBB от англ. Tight Bounding Box). Внутренний граничный блок представляет из себя наименьший гиперпараллелепипед содержащий все точки поддерева Q и определяется максимальными и минимальным координатами точек в данном поддереве (используя обозначения [15]):

$$TBB_Q = \prod_{i \in \{1 \ldots d\}} \left[l_i^Q; h_i^Q\right] = \prod_{i \in \{1 \ldots d\}} \left\{\mathbf{x}; \mathbf{x} \in \mathbb{R}^d, l_i^Q \leqslant x_i \leqslant h_i^Q\right\}, \tag{5}$$

$$l_i^Q = \min_{y \in Q} y_i, h_i^Q = \max_{y \in Q} y_i, \tag{6}$$

где $\mathbf{x} = (x_1, \ldots, x_d)$ координаты точки, $\prod$ означает декартово произведение, $\mathbf{y} \in Q$ все точки из поддерева Q. Цель данного шага повысить эффективность поиска, т.к. TBB всегда меньше чем соответствующий для этого же узла LBB.

1.1.2 BuildTree

Алгоритм построения дерева (см. Алгоритм 1) стандартный. Он основан на рекурсивном разделении данных. Построение начинается с полного множества точек S как корня дерева и пространством $\mathbb{R}^D$ в качестве внешнего граничного блока. Затем рекурсивно от корня, для каждого узла Q выбирается ось m по которой происходит разделение точек. Ось выбирается как самое длинное ребро внутреннего граничного блока.

$$m = \arg \max_{i \in \{1 \dots d\}} \left(l_i^Q - h_i^Q \right), \tag{7}$$

где границы внутреннего граничного блока определяются по формулам (5, 6). Разделяющая величина ξ является медианой $\{x_m; \mathbf{x} \in Q\}$. В таком случае построенное дерево будет сбалансированное с равным количеством точек в поддеревьях. Разделение точек останавливается, когда количество точек в текущем узле меньше чем заранее установленный параметр L (максимальное количество точек в листе дерева). Затем рассчитывается внутренний граничный блок для каждого нового узла, согласно формуле (5) проходя все точки в узле; внешний граничный блок создаётся за константное время согласно формулам (3, 4)

Наиболее асимптотически трудоёмкий шаг – это расчет внешнего граничного блока, который имеет оценку $O(md)$, где m количество точек в узле; медиана рассчитывается за $O(m)$. Итоговая асимптотическая сложность построения дерева будет равна $O\left(dn \log \frac{n}{L}\right)$, где $\log_2 \frac{n}{L}$ глубина дерева.

1.1.3 ANN

Алгоритм поиска приближенного поиска ближайшего соседа (см. Алгоритм 2) основывается обходе дерева снизу используя очередь с приоритетами.

Алгоритм начинает движение с того, что находит узел Q в которой была бы помещена точка $\mathbf{q}$, если бы она была в множестве S. Далее начинается обход дерева не посещенных узлов дерева и обновление кандидата на ближайшего соседа. Порядок обхода определяется стратегией best-bin-first, которая использует нижнюю границу расстояния η_X от точки $\mathbf{q}$ до ещё не пройденных точек из узла X. Её идея заключается в том, что ближайший сосед скорее всего находится в узле, который ближе к точке $\mathbf{q}$. η_X рассчитывается следующим образом: если X не предок Q, значит $\mathbf{q}$ лежит вне X, тогда η_X это расстояние от $\mathbf{q}$ до TBB_X, $\eta_X = d\left(\mathbf{q}, TBB_X\right)$. В другом случае, если X предок Q, значит $\mathbf{q}$ лежит внутри X. А так как к вершине X алгоритм пришёл из вершины Q, это значит, что X имеет поддерево Z, которое мы ещё не посетили, тогда η_X это расстояние от $\mathbf{q}$ до LBB_Z, $\eta_X = d\left(\mathbf{q}, LBB_Z\right)$.

Когда узел дерева посещен, если он ещё не был просмотрен, он помещается в очередь с приоритетами, в качестве приоритета используется η_X. Вершина для следующего посещения выбирается из вершины очереди η которой наименьшая. Если значение η узла больше чем расстояния до текущего кандидата в ближайшие соседи, то вершина не помещается в очередь.

Когда очередь пуста, алгоритм завершает свою работу.

Алгоритм 1: BuildTree, построение расширенного k-d дерева.

Исходные параметры: Множество точек S из $\mathbb{R}^d$.

Результат: k-d дерево T для множества S.

Function `BuildTree`(S)

создать корень дерева R с точками из S и $LBB = \mathbb{R}^d$

`SplitNode`(R)

возвратить *дерево T с корнем R*

Function `SplitNode`(*Узел дерева Q с множеством точек P и LBB_Q*)

вычислить TBB_Q следующим образом:

$$l_i^Q = \min\{x_i; \mathbf{x} \in P\}, h_i^Q = \max\{x_i; \mathbf{x} \in P\}$$

если *количество точек* $|P| > L$ **тогда**

$m \leftarrow \arg\max_{1 \leqslant i \leqslant D}(x_i^H - x_i^L)$

$\xi \leftarrow$ `Median` $\{x_i; \mathbf{x} \in P\}$

$P_L \leftarrow \{x_m < \xi; x \in P\}$

$P_R \leftarrow \{x_m \geqslant \xi; x \in P\}$

$LBB_L \leftarrow \{x_m < \xi; x \in LBB_Q\}$

$LBB_R \leftarrow \{x_m \geqslant \xi; x \in LBB_Q\}$

если $P_L \neq \varnothing$ *и* $P_R \neq \varnothing$ **тогда**

создать левое поддерево L с предком Q, точками P_L и LBB_L

создать правое поддерево R с предком Q, точками P_R и LBB_R

`SplitNode`(L)

`SplitNode`(R)

конец условия

конец условия

Алгоритм выше описывает поиск ближайшего соседа. Зачастую он оказывается медленным и не совсем эффективным. Для устранения данного недостатка используется следующая деталь. Алгоритму на вход подаётся параметр V, которые определяет, сколько точек алгоритму позволено посетить. В таком случае, алгоритм завершает свою работу, когда резерв посещения точек исчерпан. Тогда сложность в худшем случае будет $O(V)$.

Алгоритм 2: ANN, поиск приближенного ближайшего соседа.

Исходные параметры: Дерево T с точками из множества S, точка запроса $\mathbf{q}$, параметр V.

Результат: Приближенный ближайший сосед.

Function `ANN(`*дерево T, точка* $\mathbf{q}$*, ограничение V*`)`

- $Q \leftarrow$ `FindNode(q)`
- $\mathbf{p}_{\min} \leftarrow \arg\min_{\mathbf{x} \in Q} ||\mathbf{q} - \mathbf{x}||$
- $dist_{\min} \leftarrow ||\mathbf{q} - \mathbf{p}_{\min}||$
- **если** $dist_{\min} = 0$ **тогда**
  - **возвратить** $\mathbf{p}_{\min}$
- **конец условия**
- $v \leftarrow V - |Q|$
- $PQ \leftarrow$ пустая очередь с приоритетами пар (Z,Z') отсортированных по η_Z
- `PushIfBetter(`*Parent(Q),Q*`)`
- **до тех пор, пока** $PQ \neq \varnothing$ *и* $v > 0$ **выполнять**
  - $(Z, Z') \leftarrow$ получить верхний элемент очереди // элемент удаляется из очереди
  - **если** $\eta_Z \geqslant dist_{\min}$ **тогда**
    - **возвратить** $\mathbf{p}_{\min}$
  - **конец условия**
  - **если** Z *лист* **тогда**
    - $dist \leftarrow \min_{\mathbf{x} \in Z} ||\mathbf{q} - \mathbf{x}||$
    - $v \leftarrow v - |Z|$
    - **если** $dist < dist_{\min}$ **тогда**
      - $dist_{\min} \leftarrow dist$
      - $\mathbf{p}_{\min} \leftarrow \arg\min_{\mathbf{x} \in Z} ||\mathbf{q} - \mathbf{x}||$
    - **конец условия**
  - **иначе**
    - **цикл** $X \in \{Parent, LeftChild, RightChild\}(Z) \setminus \{Z'\}$ **выполнять**
      - `PushIfBetter(`X, Z`)`
    - **конец цикла**
  - **конец условия**
- **конец цикла**
- **возвратить** $\mathbf{p}_{\min}$

Function `PushIfBetter(`*новая вершина* X, *прошлая вершина* Z`)`

если $X = Parent(Z)$ **тогда**
  $\eta_X = d\,(\mathbf{q}, LBB_Z)$
иначе
  $\eta_X = d\,(\mathbf{q}, TBB_X)$
конец условия
если $\eta_x < dist_{\min}$ **тогда**
  `push(`X, Z, η_X`)`
конец условия

1.2 Locality-Sensitive Hashing

Алгоритм локально-чувствительного хеширования [16] использует хеш-функции для поиска ближайших соседей. В его основе используется следующая идея: необходимо найти такое семейство хеш-функций, которые бы имели одинаковое значение если точки в пространстве находятся в определенной близости.

Введём обозначения: S - конечное множество точек пространства $\mathbb{R}^d$, q будет обычно употребляться в качестве точки запроса, u,v - точки из множества S. Шар радиуса r, центр которого в точке v, обозначается за $B(v,r)$. Для точки q мы будем называть v приближенным ближайшим соседом, если $v \in B(q, R)$

1.2.1 Обобщенная схема локально-чувствительного хеширования

Обозначим семейство LSH следующим образом:

Определение 1 Семейство $\mathcal{H} = \{h : S \to U\}$ называется локально-чувствительным, если для любого вектора q функция $p(t) = \Pr_{\mathcal{H}}[h(q) = h(v) : ||q - v|| = t]$ строго убывающая. То есть вероятность коллизии точки q и v убывает с увеличением расстояния между ними.

Таким образом, чтобы достичь желаемого результата, необходимо увеличить разницу между вероятностью коллизий хеш-функций в диапазоне $[0, R]$ (где находятся приближенные ближайшие соседи) и в диапазоне (R, ∞). Для этого происходит объединение нескольких функций $h \in \mathcal{H}$. Таким образом определяем семейство функций $\mathcal{G} = \{g : S \to U^k\}$. Другими словами, $g(v) = (h_1(v), \ldots, h_k(v))$, где $h_i \in \mathcal{H}$. Для целого числа L алгоритм выбирает L функций $g_1, \ldots, g_L$ из $\mathcal{G}$ независимым и произвольным образом. Алгоритм хранит каждую точку в корзине $g_j(v)$ для всех $j = 1, \ldots, L$. Так как итоговое количество корзин может быть большим, алгоритм хранит только непустые, используя хеширование.

Чтобы обработать точку запроса q, алгоритм вычисляет значения во всех корзинах $g_1(q), \ldots, g_L(q)$. Для каждой точки v найденной в корзине, алгоритм вычисляет расстояние от q до v и если оно меньше чем текущее расстояние до текущего кандидата в ближайшие соседи (если текущего кандидата нет, то расстояния принимается равным бесконечности), то кандидат обновляется.

1.2.2 Хеш-функция

Математическое обоснования формул хеш-функции можно найти в [16]. Формально хеш-функция $h_{a,b}(v) : \mathbb{R}^d \to \mathbb{Z}$ переводит d-мерный вектор в целое число. Каждая функция определяется произвольными значениями a и b, где a это вектор размерности d, компоненты которого взяты из стандартного нормального распределения; b - число, выбранное из равномерного распределения в промежутке $[0, w]$. Для фиксированных a,b хеш-функция $h_{a,b}$ имеет

вид:

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{w} \right\rfloor. \tag{8}$$

1.2.3 Сложность

Сложность данного алгоритма можно разбить на две составляющие. Первая составляющая связана со сложностью вычисления хеш-функции: скалярное произведение вычисляется за $O(d)$. Всего k функций в L корзинах, итоговая сложность составляет – $O(dkL)$. Вторая компонента включает сложность вычисления расстояний до точек, с одинаковым значением хеш-функций. Количество таких точек находим через математическое ожидание и вероятность коллизий (см. определение 1). Получаем оценку $O\left(dL \sum_{v \in P} p^k(||q - v||)\right)$.

2 Входные данные

В данной главе будут описаны типы данных, на которых тестировались алгоритмы и соответствующие способы их генерации. Было решено проводить эксперименты на интервале $[0, 1)$; в размерностях $d = 2^i; i \in \{1, \ldots, 6\}$; в количествах $n = 1000, 5000, 10000, 50000, 100000$. В качестве тестов генерируется ещё $n_q = 1000$ точек того же типа входных данных, по которым снимаются показатели и находятся средние значения.

2.1 Равномерное распределение

Точки имеют равномерное распределение в полусегменте $[0, 1)$. Для измерения расстояния используется l_2-норма. Пример распределения точек приведен на рисунке 2.1.

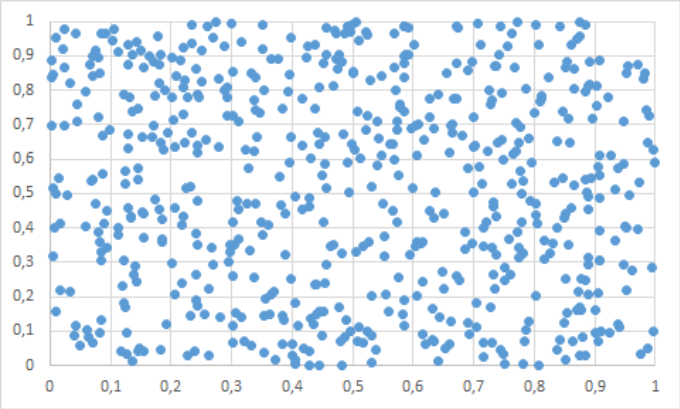

Рисунок 2.1 — Равномерное распределение для размерности $d = 2$.

2.2 Нормальное распределение

Точки имеют нормальное распределение по каждой из осей с параметрами $\mu = \frac{1}{2}, \sigma = \frac{1}{6}$. Данные параметры подобраны специально, используя правило 3σ. Для измерения расстояния используется l_2-норма. Пример распределения точек приведен на рисунке 2.2.

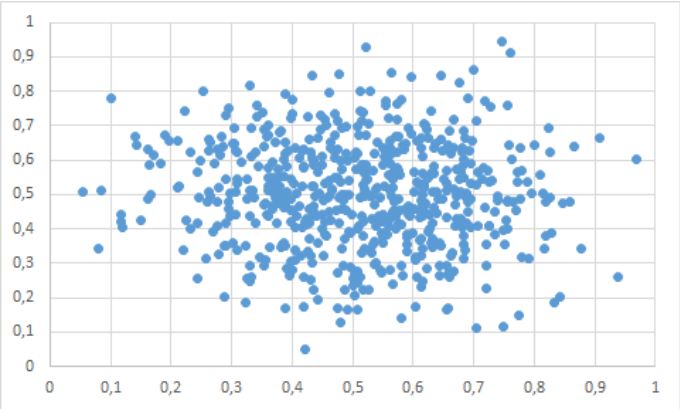

Рисунок 2.2 — Нормальное распределение $\mathcal{N}(\frac{1}{2}, \frac{1}{6})$, $d = 2$.

2.3 Обратное нормальное распределение

Под обратным или инверсным понимается нормальное распределение с обратной функцией распределения. Другими словами, вероятность появления в точке математического ожидания будет наименьшей, чем в любых других точках. Для измерения расстояния используется l_2-норма. Пример распределения точек приведен на рисунке 2.3.

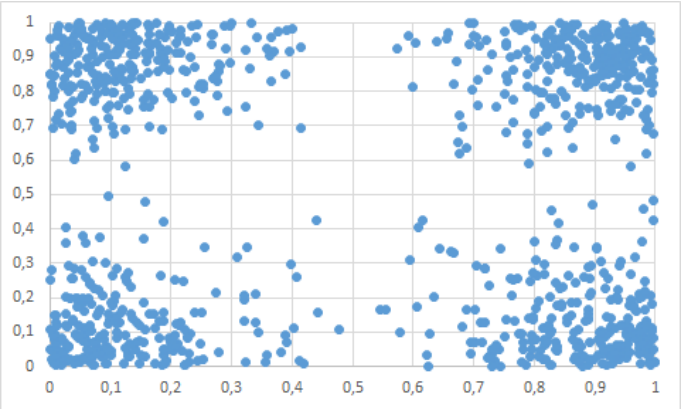

Рисунок 2.3 — Обратное нормальное распределение $d = 2$.

2.4 Равномерное распределение на гиперсфере

Данный вид входных данных предоставляет точки, равномерно распределенные на сфере, вписанной в единичный гиперкуб. Для измерения расстояния будет использоваться кратчайшая дуга между двумя точками. Для упрощения расчета расстояния переместим сферу в центр координат. Тогда точки на d-мерной сфере задаются следующим соотношением:

$$\begin{cases} x_1 = \frac{1}{2} \cdot \sin\alpha_1 \cdot \sin\alpha_2 \cdot \ldots \cdot \sin\alpha_{d-1} \\ x_2 = \frac{1}{2} \cdot \cos\alpha_1 \cdot \sin\alpha_2 \cdot \ldots \cdot \sin\alpha_{d-1} \\ x_3 = \frac{1}{2} \cdot \cos\alpha_2 \cdot \sin\alpha_3 \cdot \ldots \cdot \sin\alpha_{d-1} \\ \ldots \\ x_d = \frac{1}{2} \cdot \cos\alpha_{d-1} \end{cases} \tag{9}$$

где x_i - i-тая компонента вектора $\mathbf{x}$. Т.к. норма вектора до любой точки сферы одинакова и равна её радиусу $r = \frac{1}{2}$ вписанной окружности в единичный гиперкуб, получаем расстояние между двумя точками:

$$d(a,b) = r \cdot \alpha_{a,b} = \frac{1}{2} \arccos \frac{a \cdot b}{|a||b|} = \frac{1}{2} \arccos 4(a,b), \tag{10}$$

где (a,b) - скалярное произведение. Пример распределения точек приведен на рисунке 2.4.

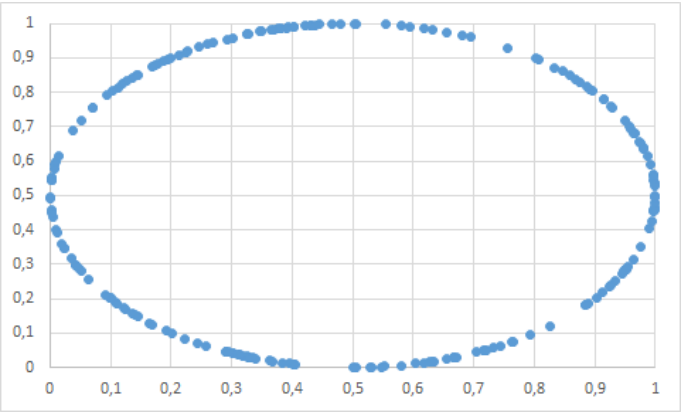

Рисунок 2.4 — Равномерное распределение на гиперсфере $d = 2$.

3 Характеристики для оценки качества алгоритмов

Для сравнения алгоритмов было решено ввести следующие характеристики:

а) Среднее время поиска приближенного ближайшего соседа. Основной критерий при выборе алгоритма. Насколько быстро он справляется с поставленной задачи.

б) Среднее отношение расстояния до приближенного соседа к расстоянию до истинного ближайшего соседа. Основная характеристика алгоритмов приближенного поиска ближайших соседей.

в) Доля найденных истинных ближайших соседей. При работе с приближенными алгоритмами хочется знать, насколько сильно часто они дают результат точных алгоритмов.

г) Доля не найденных соседей. Данная характеристика показывает процент точек, для которых алгоритм не нашёл решение. Больше всего она актуальна к локально-чувствительному поиску, так как при определенных данных может не существовать точек в корзинах с таким же значением хеш-функции как у точки запроса, следовательно, соседи найдены не будут.

4 Результаты тестов

Приложение А содержит результаты тестов в табличном виде.

Приложение Б содержит визуализированные данные в графиках.

5 Анализ результатов

В данной главе будет произведен анализ результатов. Будут рассмотрены особенности каждого алгоритма в отдельности и общее поведение данных алгоритмов.

5.1 Approximate Best Bin First *k*-d Tree

Алгоритмы поиска ближайших соседей, основанные на деревьях, страдают от «проклятья размерности». Проклятье размерности подразумевает резкое ухудшение эффективности алгоритмов в виду повышения размерности. Размерность пространства сказывается следующим образом: деревья обычно разделяют какими-либо способами пространство точек; когда размерность становится достаточно большой и превышает высоту дерева $h = \log_2(n)$, то алгоритм, можно сказать, голодает от недостатка данных, он не может в полной мере оценить, что две точки действительно находятся близко. Например, у нас количество точек $n = 100000$, тогда высота дерева $h = 17$, пусть размерность измерения 32. Таким образом находясь в листе, мы знаем, что в нём находятся близкие точки только по 17 измерениям. Что происходит с другими 15 алгоритму неизвестно. Поэтому приходится делать обход по большому количество точек и обходить полное дерево. Что нам показывает и алгоритм Best Bin First. Начиная с размерности $d = 16$, время его работы начинает превышать линейное время. Связано это с издержками на хождение по дереву. Даже такие оптимизации как разделение пространства по измерению, имеющему наибольшую ширину (речь идёт об интервале разброса точек по данной оси), очередь приоритетов и обход ограниченного числа вершин не спасают от данного проклятья.

Однако, ситуация резко меняется, если рассматривать данный алгоритм на точках, размещенных на сферах. Ответ на самом деле прост. В отличии от точек «раскиданных» в кубе, на сфере не происходит сильного разброса. Рассмотрим объём гиперсферы при $d = 16$. Согласно [17], объём сферы единичного радиуса $V(d) = \frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}$. При $d = 64, V(d) < 10^{-19}$, когда объём куба остаётся равным единице. Другими словами, на поверхности гиперсферы концентрация точек выше, чем в объёме гиперкуба, при одинаковых размерностях. Таким образом, алгоритм лучше работает, потому что данные высоко сконцентрированы и ближайший сосед находится «легче».

5.2 Locality-Sensitive Hashing

Локально-чувствительное хеширование хорошо себя показывает при высоких размерностях данных, так сказать, является фаворитом. Но на низких размерностях работает практически наравне с линейным. Также на графиках выделяется «нездоровое» замедление работы при $d = 8$. Связано это с ростом количества корзин, необходимых для данной размерности. Практически на всех данных алгоритм показывает линейное время, которое меньше линейного перебора, однако особо сильное преимущество достигается только на высоких размерностях.

На гиперсфере ситуация обратная деревьям, связано это с попаданием большого количество точек под одно значение хеш-функции, таким образом алгоритму приходится обходить много одинаковых точек, которые попали по нескольку раз в разные корзины.

5.3 Истинные ближайшие соседи

Зависимость доли истинных ближайших соседей среди приближенных от размерности пространства качественно совпадает у обоих алгоритмов. Единственное отличие заключается в оценке производной. У LSH производная имеет большее значение чем у BBF.

Значение ϵ в расстоянии до приближенного соседа, у локально-чувствительного поиска стабильно растёт (за исключением случая распределения на сфере, где при росте измерения он становится почти константным). У BBF имеются некие пики, когда ϵ достигает максимума. Эти пики приходятся на $d = 8$ и $d = 16$. Именно в этом промежутке происходит некий переломный момент, когда дерево начинает работать медленнее линейного перебора.

Заключение

В ходе научно-исследовательской работы были изучены алгоритмы приближенного поиска ближайших соседей для которых было проведено сравнение на различных входных данных, различных размерностях и объёмах выборок. Результаты полученные в ходе исследования оказались неоднозначны. Алгоритм поиска соседей, основанный на деревьях проявил свою подверженность «проклятью размерности», однако, наравне с этим он показал хорошее время работы на специфичных входных данных в больших размерностях, таких как равномерно распределенные точки на поверхности гиперсферы. Алгоритм локально-чувствительного хеширования хорошо показал себя на данных высокой размерности, но так как хеширование основано на сферических областях, а в работе было показано, что отношение объёма сферы к объёму куба, описанного вокруг сферы, стремится к нулю при стремлении размерности к бесконечности, то коллизий на высоко размерных данных возникает меньше и количество найденных истинных ближайших соседей падает, также растёт и расстояние до приближенных ближайших соседей.

В момент завершения исследования был опубликован алгоритм [18], который способен с большей эффективностью искать приближенных ближайших соседей, но только на сферических плоскостях, который, к сожалению, не вошёл в работу.

На основе полученных результатов были сформулированы рекомендации. В общем случае (равномерное, нормальное и инверсное нормальное распределения) до размерности пространства $d = 8$ включительно, рекомендуется использовать алгоритмы, использующие деревья, они дают наилучшее время поиска приближенного ближайшего соседа. При размерностях выше, рекомендуется использовать алгоритмы, использующие хеш-функции. Также в случае поиска ближайшего соседа на сферических плоскостях, лучшее время показывает алгоритм, основанный на дереве, до размерности $d = 64$ включительно.

Список использованных источников

1 M. S. Waterman, Introduction to Computational Biology, Chapman and Hall, 2000.

2 S. W. Park, L. Linsen, O. Kreylos, J. D. Owens, B. Hamann, Discrete sibson interpolation, IEEE Transactions on Visualization and Computer Graphics 12 (2) (2006) 243–253.

3 U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, From data mining to knowledge discovery in databases, AI Magazine 17 (1996) 37–54.

4 F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, Z. Protopapas, Fast nearest neighbor search in medical image databases, in: The VLDB Journal, 1996, pp. 215–226

5 R. O. Duda, P. E. Hart, D. G. Stork, Pattern classification, 2nd Edition, Wiley Interscience Publication, John Wiley, New York, 2001.

6 T. M. Cover, P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory 13 (1967) 21–27.

7 Винокуров Станислав Владимирович. Математическое и программное обеспечение методов повышения временной эффективности фрактального сжатия изображений : диссертация ... кандидата технических наук : 05.13.11 / Винокуров Станислав Владимирович; [Место защиты: Моск. гос. ун-т приборостроения и информатики]. - Москва, 2007. - 126 с. : ил. РГБ ОД.

8 D. E. Knuth, The Art of Computer Programming, Vol 3: Sorting and Searching, 2nd Edition, Addison-Wesley, Reading, Mass., 1998.

9 S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, Journal of the ACM 45 (6) (1998) 891–923.

10 P. Tsaparas, Nearest neighbor search in multidimensional spaces, Tech. Rep. 31902, Dept. of Computer Science, University of Toronto, Canada, qualifying Depth Oral Report (1999).

11 J. H. Friedman, J. L. Baskett, L. J. Shustek, An algorithm for finding nearest neighbors, IEEE Trans. Comput. 24 (1975) 1000–1006.

12 S. A. Nene, S. K. Nayar, A simple algorithm for nearest neighbor search in high dimensions, IEEE Trans. Pattern Anal. Mach. Intell. 19 (9) (1997) 989–1003.

13 H. Samet, The quadtree and related hierarchical data structures, ACM Comput. Surv. 16 (2) (1984) 187–260.

14 J. S. Beis, D. G. Lowe, Shape indexing using approximate nearestneighbour search in high-dimensional spaces, in: Proceedings of Conference on Computer Vision and Pattern Recognition, 1997, pp. 1000–1006.

15 Jan Kybic, Ivan Vnucko, Approximate all nearest neighbor search for high dimensional entropy estimation for image registration, Signal Processing, 2011.

16 M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-Sensitive hashing scheme based on p-stable distributions. DIMACS Workshop on Streaming Data Analysis and Mining, 2003.

17 Equation 5.19.4, NIST Digital Library of Mathematical Functions. http://dlmf.nist.gov/, Release 1.0.6 of 2013-05-06.

18 Alexandr Andoni, Ilya Razenshteyn. Optimal Data-Dependent Hashing for Approximate Near Neighbors, 2015.

Приложение А

(обязательное)

Результаты тестов

Таблица 1 — Равномерное распределение в единичном гиперкубе

		Полный перебор	Best Bin First				Local Sensitive Hashing			
Размерность	Количество точек в структуре	Среднее время поиска	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов
2	1000	0,000031	0,000001	1,000	1,000	0,000	0,000031	1,000	1,000	0,000
2	5000	0,000187	0,000002	1,000	1,000	0,000	0,000156	1,000	1,000	0,000
2	10000	0,000609	0,000009	1,000	1,000	0,000	0,000546	1,000	1,000	0,000
2	50000	0,002650	0,000016	1,000	1,000	0,000	0,002340	1,000	1,000	0,000
2	100000	0,007707	0,000015	1,000	1,000	0,000	0,007038	1,000	1,000	0,000
4	1000	0,000063	0,000016	1,001	0,999	0,000	0,000062	1,000	1,000	0,000
4	5000	0,001345	0,000094	1,000	1,000	0,000	0,000639	1,014	0,966	0,000
4	10000	0,001919	0,000156	1,000	1,000	0,000	0,000920	1,019	0,962	0,000
4	50000	0,007832	0,000422	1,000	1,000	0,000	0,005758	1,006	0,984	0,000
4	100000	0,013734	0,000561	1,000	1,000	0,000	0,011205	1,008	0,989	0,000
8	1000	0,000094	0,000062	1,073	0,681	0,000	0,000062	1,012	0,944	0,000
8	5000	0,001312	0,000702	1,050	0,779	0,000	0,000858	1,003	0,965	0,000
8	10000	0,002810	0,001404	1,044	0,803	0,000	0,003170	1,001	0,994	0,000
8	50000	0,010178	0,005569	1,007	0,961	0,000	0,007476	1,005	0,969	0,000
8	100000	0,017450	0,009222	1,004	0,989	0,000	0,019786	1,001	0,996	0,000
16	1000	0,000125	0,000125	1,057	0,568	0,000	0,000171	1,004	0,958	0,000
16	5000	0,001825	0,001264	1,060	0,547	0,000	0,001452	1,012	0,892	0,000
16	10000	0,003635	0,002605	1,056	0,563	0,000	0,001499	1,021	0,791	0,000
16	50000	0,011396	0,013932	1,068	0,517	0,000	0,007659	1,011	0,865	0,000
16	100000	0,019677	0,027747	1,077	0,496	0,000	0,011332	1,017	0,820	0,000
32	1000	0,000359	0,000234	1,030	0,533	0,000	0,000062	1,051	0,395	0,001
32	5000	0,002068	0,001498	1,032	0,553	0,000	0,000421	1,046	0,459	0,000
32	10000	0,003604	0,003028	1,032	0,525	0,000	0,000702	1,045	0,438	0,000
32	50000	0,012716	0,015962	1,033	0,541	0,000	0,004073	1,033	0,527	0,000
32	100000	0,023750	0,032321	1,037	0,513	0,000	0,007695	1,030	0,553	0,000
64	1000	0,000546	0,000343	1,021	0,507	0,000	0,000063	1,098	0,081	0,004
64	5000	0,002449	0,001840	1,020	0,513	0,000	0,000189	1,063	0,136	0,001
64	10000	0,004371	0,003619	1,018	0,525	0,000	0,000499	1,046	0,223	0,000
64	50000	0,018133	0,018991	1,017	0,531	0,000	0,003214	1,037	0,300	0,000
64	100000	0,032422	0,038461	1,018	0,522	0,000	0,007386	1,033	0,329	0,000

Таблица 2 — Нормальное распределение в единичном гиперкубе

		Полный перебор	Best Bin First				Local Sensitive Hashing			
Размерность	Количество точек в структуре	Среднее время поиска	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов
2	1000	0,000036	0,000000	1,000	1,000	0,000	0,000031	1,000	1,000	0,000
2	5000	0,000187	0,000000	1,000	1,000	0,000	0,000156	1,000	1,000	0,000
2	10000	0,000609	0,000015	1,000	1,000	0,000	0,000531	1,000	1,000	0,000
2	50000	0,003473	0,000015	1,000	1,000	0,000	0,002428	1,000	1,000	0,000
2	100000	0,008011	0,000028	1,000	1,000	0,000	0,007105	1,000	1,000	0,000
4	1000	0,000078	0,000031	1,004	0,982	0,000	0,000078	1,000	1,000	0,000
4	5000	0,001079	0,000168	1,001	0,998	0,000	0,000614	1,013	0,960	0,000
4	10000	0,001952	0,000297	1,000	1,000	0,000	0,001117	1,012	0,963	0,000
4	50000	0,007860	0,000752	1,000	1,000	0,000	0,007181	1,004	0,985	0,000
4	100000	0,013964	0,001033	1,000	1,000	0,000	0,013143	1,001	0,998	0,000
8	1000	0,000094	0,000078	1,096	0,541	0,000	0,000093	1,003	0,975	0,000
8	5000	0,001260	0,000780	1,100	0,581	0,000	0,001368	1,002	0,982	0,000
8	10000	0,002480	0,001658	1,095	0,613	0,000	0,004469	1,000	1,000	0,000
8	50000	0,010328	0,008334	1,052	0,771	0,000	0,012938	1,000	0,995	0,000
8	100000	0,017951	0,014405	1,040	0,867	0,000	0,026064	1,000	0,998	0,000
16	1000	0,000124	0,000141	1,062	0,463	0,000	0,000249	1,000	1,000	0,000
16	5000	0,001827	0,001263	1,066	0,463	0,000	0,002195	1,000	0,994	0,000
16	10000	0,003645	0,002606	1,071	0,435	0,000	0,002598	1,009	0,879	0,000
16	50000	0,011462	0,013835	1,073	0,450	0,000	0,012566	1,005	0,940	0,000
16	100000	0,019755	0,027447	1,080	0,408	0,000	0,022379	1,005	0,936	0,000
32	1000	0,000349	0,000239	1,037	0,475	0,000	0,000172	1,017	0,731	0,000
32	5000	0,002078	0,001514	1,035	0,450	0,000	0,001237	1,018	0,701	0,000
32	10000	0,003621	0,003042	1,036	0,450	0,000	0,001528	1,020	0,632	0,000
32	50000	0,012828	0,015823	1,033	0,483	0,000	0,012093	1,010	0,822	0,000
32	100000	0,023302	0,032754	1,036	0,450	0,000	0,021324	1,008	0,837	0,000
64	1000	0,000547	0,000362	1,023	0,480	0,000	0,000137	1,036	0,346	0,000
64	5000	0,002456	0,001843	1,021	0,469	0,000	0,000994	1,020	0,530	0,000
64	10000	0,004369	0,003610	1,021	0,465	0,000	0,002048	1,016	0,551	0,000
64	50000	0,018333	0,018723	1,020	0,443	0,000	0,011907	1,011	0,673	0,000
64	100000	0,032084	0,037908	1,019	0,461	0,000	0,029642	1,006	0,808	0,000

Таблица 3 — Обратное нормальное распределение в единичном гиперкубе

		Полный перебор	Best Bin First				Local Sensitive Hashing			
Размерность	Количество точек в структуре	Среднее время поиска	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов
2	1000	0,000036	0,000002	1,000	1,000	0,000	0,000030	1,000	1,000	0,000
2	5000	0,000192	0,000000	1,000	1,000	0,000	0,000182	1,000	1,000	0,000
2	10000	0,000657	0,000006	1,000	1,000	0,000	0,000593	1,000	1,000	0,000
2	50000	0,002662	0,000015	1,000	1,000	0,000	0,002354	1,000	1,000	0,000
2	100000	0,014532	0,000000	1,000	1,000	0,000	0,007048	1,000	1,000	0,000
4	1000	0,000046	0,000016	1,000	0,999	0,000	0,000062	1,000	1,000	0,000
4	5000	0,001158	0,000031	1,000	1,000	0,000	0,000602	1,008	0,979	0,000
4	10000	0,001985	0,000046	1,000	1,000	0,000	0,000818	1,004	0,984	0,000
4	50000	0,007839	0,000140	1,000	1,000	0,000	0,004782	1,005	0,991	0,000
4	100000	0,014276	0,000207	1,000	1,000	0,000	0,009313	1,002	0,991	0,000
8	1000	0,000093	0,000031	1,010	0,981	0,000	0,000047	1,012	0,970	0,000
8	5000	0,001264	0,000100	1,002	0,996	0,000	0,000555	1,003	0,982	0,000
8	10000	0,002492	0,000156	1,001	0,998	0,000	0,002464	1,002	0,991	0,000
8	50000	0,010258	0,000343	1,000	1,000	0,000	0,004930	1,003	0,979	0,000
8	100000	0,017540	0,000499	1,000	1,000	0,000	0,017096	1,001	0,993	0,000
16	1000	0,000141	0,000109	1,075	0,606	0,000	0,000109	1,018	0,874	0,000
16	5000	0,001826	0,001217	1,070	0,645	0,000	0,000998	1,028	0,838	0,000
16	10000	0,003651	0,002522	1,070	0,697	0,000	0,001171	1,045	0,750	0,000
16	50000	0,011398	0,012744	1,064	0,790	0,000	0,005631	1,022	0,886	0,000
16	100000	0,020671	0,021707	1,038	0,883	0,000	0,005923	1,045	0,804	0,000
32	1000	0,000355	0,000233	1,031	0,584	0,000	0,000026	1,099	0,211	0,003
32	5000	0,002078	0,001612	1,033	0,573	0,000	0,000191	1,083	0,293	0,000
32	10000	0,003716	0,003129	1,037	0,574	0,000	0,000349	1,076	0,304	0,000
32	50000	0,013339	0,016622	1,043	0,549	0,000	0,001625	1,070	0,331	0,000
32	100000	0,023948	0,032266	1,047	0,544	0,000	0,003358	1,066	0,362	0,000
64	1000	0,000549	0,000350	1,018	0,529	0,000	0,000043	1,140	0,032	0,031
64	5000	0,002449	0,001936	1,018	0,557	0,000	0,000078	1,107	0,057	0,001
64	10000	0,004654	0,003853	1,019	0,526	0,000	0,000200	1,081	0,099	0,000
64	50000	0,018952	0,018819	1,020	0,504	0,000	0,001098	1,070	0,108	0,000
64	100000	0,032365	0,038326	1,017	0,562	0,000	0,002275	1,062	0,125	0,000

Таблица 4 — Равномерное распределение на сфере

		Полный перебор	Best Bin First				Local Sensitive Hashing			
Размерность	Количество точек в структуре	Среднее время поиска	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов	Среднее время поиска	E[(1+eps)R]	Доля найденных ближайших соседей	Доля невыполненных запросов
2	1000	0,000084	0,000002	1,000	1,000	0,000	0,000070	1,000	0,999	0,000
2	5000	0,000546	0,000000	1,004	0,997	0,000	0,000430	1,000	1,000	0,000
2	10000	0,001086	0,000000	1,011	0,991	0,000	0,000696	1,000	1,000	0,000
2	50000	0,005758	0,000006	1,131	0,920	0,000	0,006282	1,000	1,000	0,000
2	100000	0,012095	0,000016	1,478	0,813	0,000	0,012019	1,000	1,000	0,000
4	1000	0,000109	0,000016	1,000	1,000	0,000	0,000109	1,000	0,998	0,000
4	5000	0,001233	0,000078	1,000	1,000	0,000	0,000312	1,024	0,948	0,000
4	10000	0,002327	0,000124	1,000	1,000	0,000	0,001170	1,013	0,969	0,000
4	50000	0,010585	0,000234	1,000	1,000	0,000	0,005136	1,021	0,978	0,000
4	100000	0,017664	0,000297	1,000	1,000	0,000	0,005735	1,011	0,977	0,000
8	1000	0,000129	0,000054	1,012	0,973	0,000	0,000064	1,009	0,956	0,000
8	5000	0,001467	0,000311	1,031	0,969	0,000	0,001254	1,002	0,989	0,000
8	10000	0,002919	0,000659	1,015	0,984	0,000	0,001658	1,002	0,989	0,000
8	50000	0,012613	0,003108	1,004	0,997	0,000	0,010393	1,003	0,981	0,000
8	100000	0,021399	0,004303	1,000	1,000	0,000	0,032093	1,000	1,000	0,000
16	1000	0,000158	0,000062	1,017	0,960	0,000	0,000109	1,002	0,977	0,000
16	5000	0,001806	0,000425	1,008	0,987	0,000	0,001717	1,001	0,994	0,000
16	10000	0,003924	0,000783	1,010	0,988	0,000	0,002905	1,001	0,994	0,000
16	50000	0,012253	0,003041	1,002	0,996	0,000	0,010558	1,001	0,993	0,000
16	100000	0,022698	0,005334	1,002	0,996	0,000	0,018028	1,005	0,967	0,000
32	1000	0,000530	0,000110	1,014	0,975	0,000	0,000353	1,006	0,968	0,000
32	5000	0,002134	0,000503	1,011	0,984	0,000	0,001643	1,003	0,983	0,000
32	10000	0,003822	0,000884	1,010	0,984	0,000	0,002134	1,003	0,980	0,000
32	50000	0,013998	0,003255	1,006	0,989	0,000	0,010739	1,003	0,980	0,000
32	100000	0,026209	0,005885	1,014	0,987	0,000	0,021405	1,000	0,997	0,000
64	1000	0,000547	0,000130	1,025	0,960	0,000	0,000652	1,001	0,996	0,000
64	5000	0,002484	0,000620	1,030	0,964	0,000	0,002668	1,000	0,999	0,000
64	10000	0,004471	0,001081	1,014	0,981	0,000	0,005355	1,001	0,997	0,000
64	50000	0,018874	0,003858	1,009	0,994	0,000	0,023724	1,000	0,999	0,000
64	100000	0,034880	0,007039	1,000	1,000	0,000	0,073687	1,000	1,000	0,000

Приложение Б

(обязательное)

Визуализация результатов

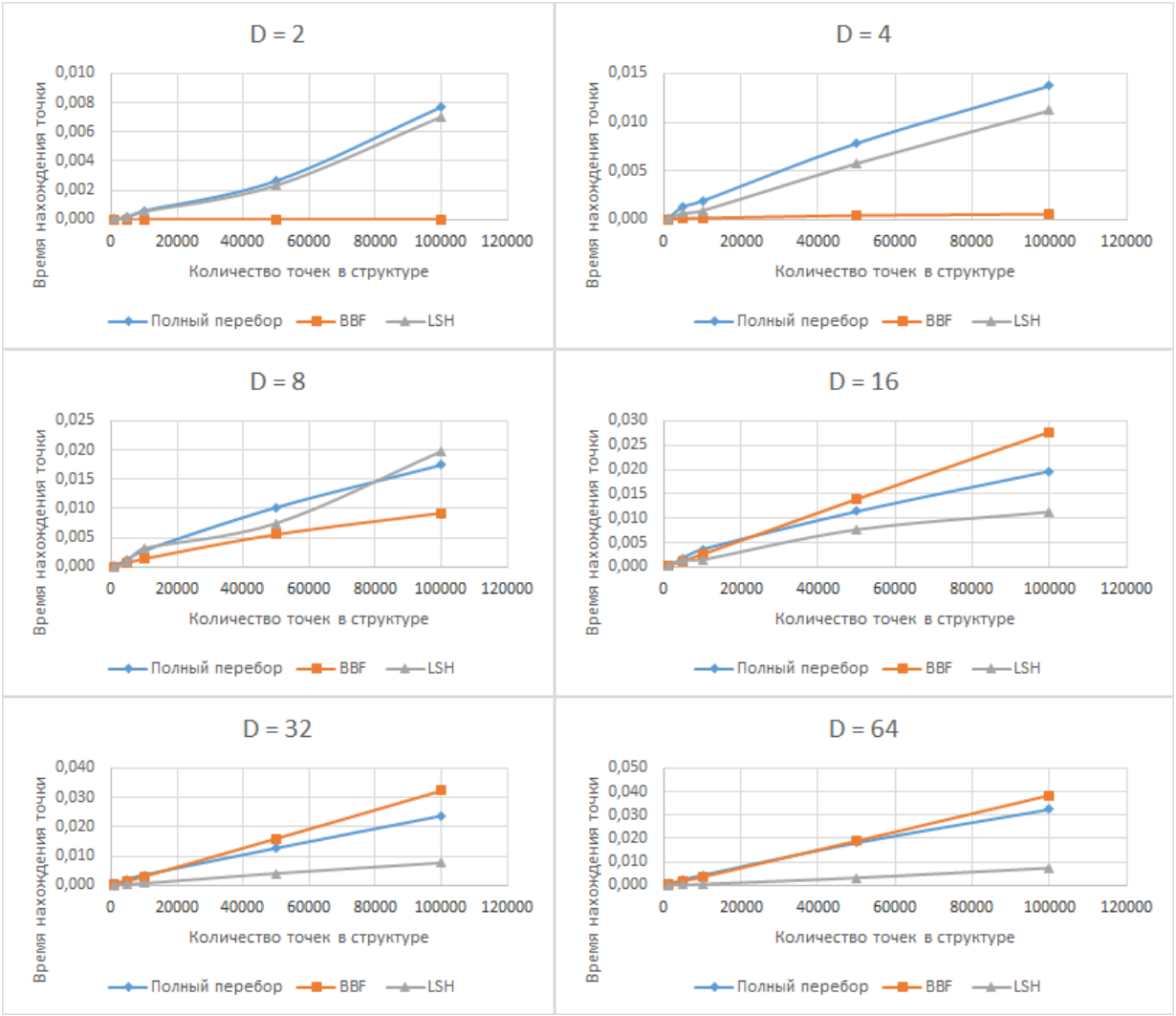

Рисунок 1 — Равномерное распределение в единичном гиперкубе. Время исполнения

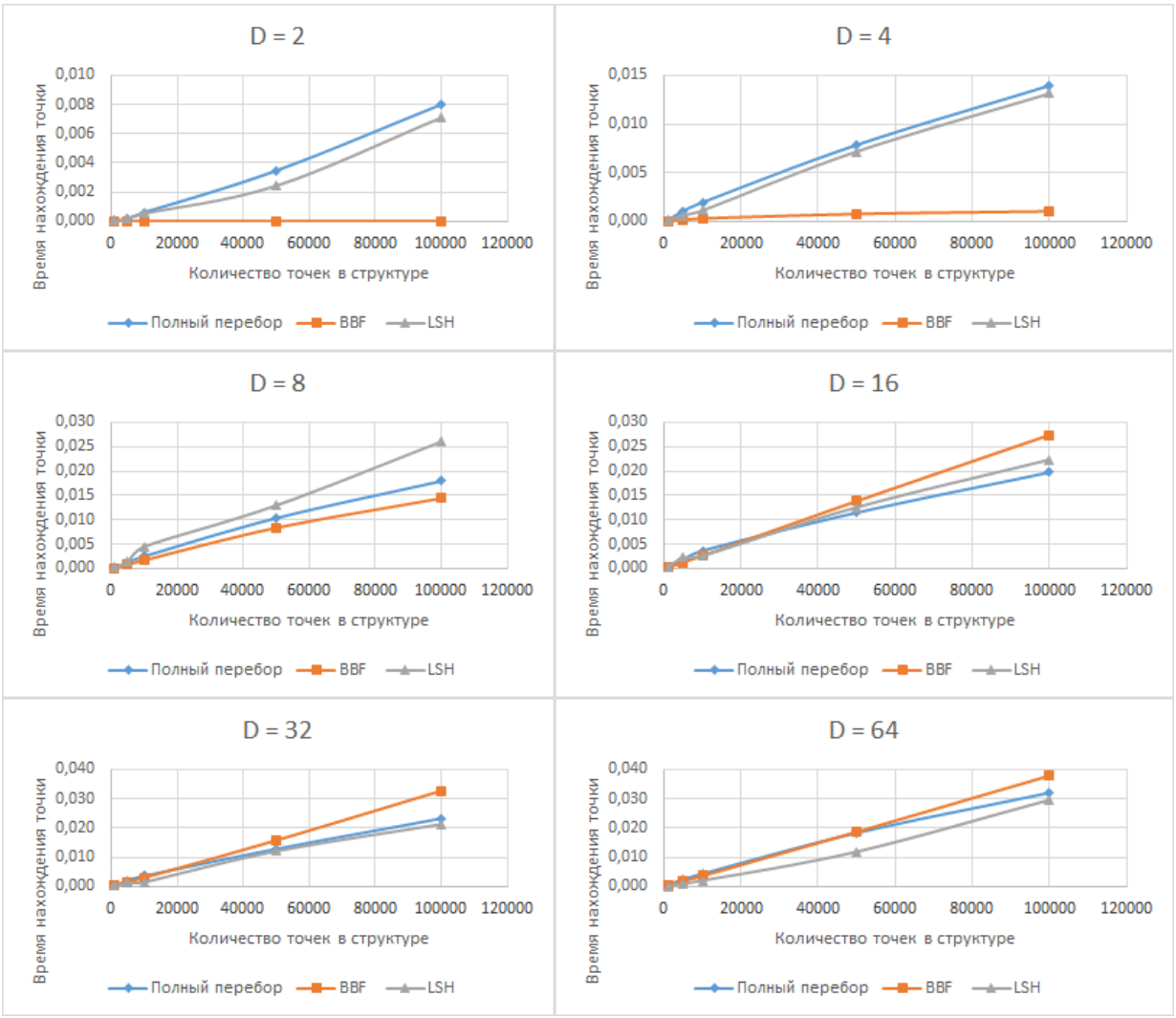

Рисунок 2 — Нормальное распределение в единичном гиперкубе. Время исполнения

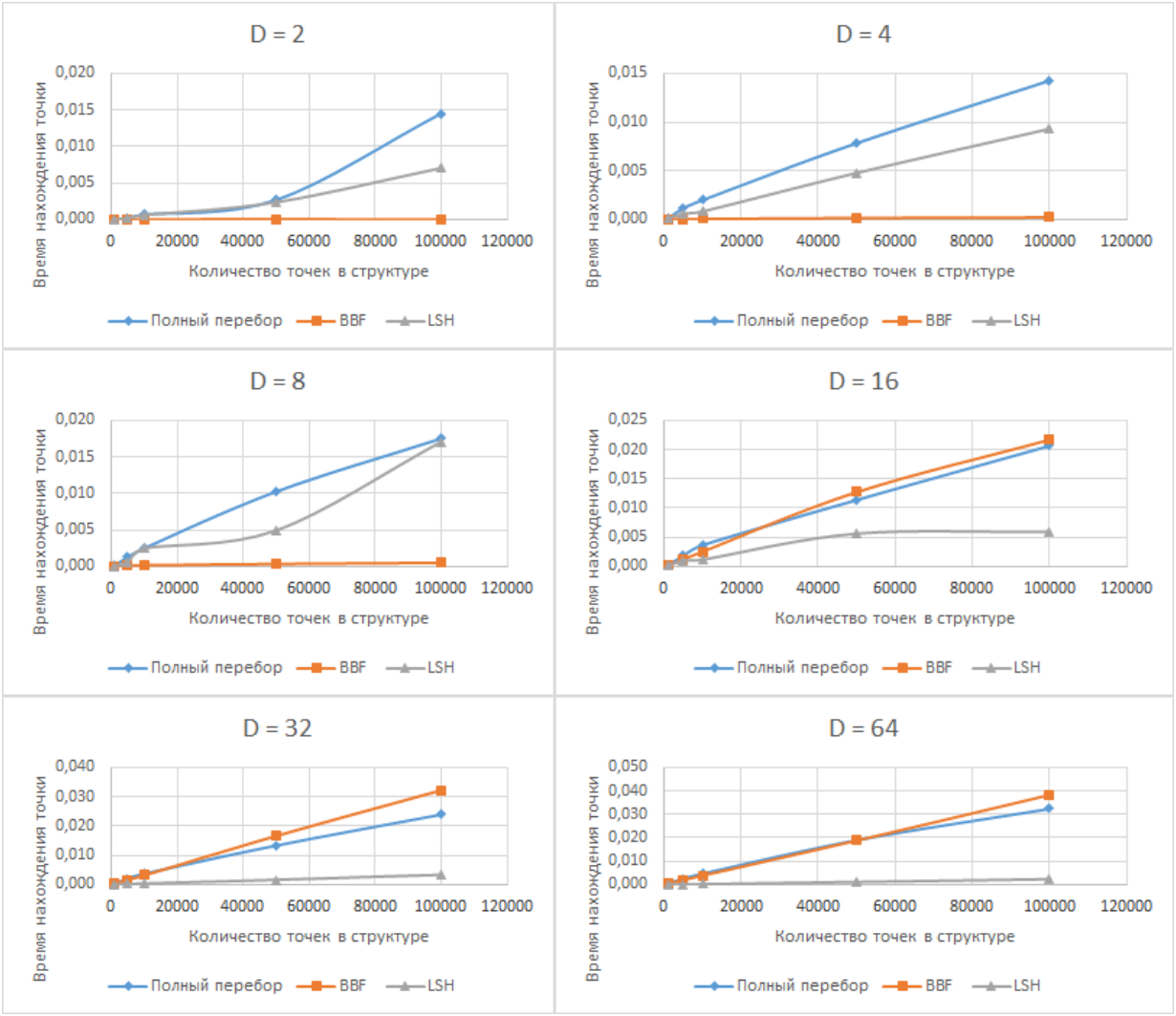

Рисунок 3 — Обратное нормальное распределение в единичном гиперкубе. Время исполнения

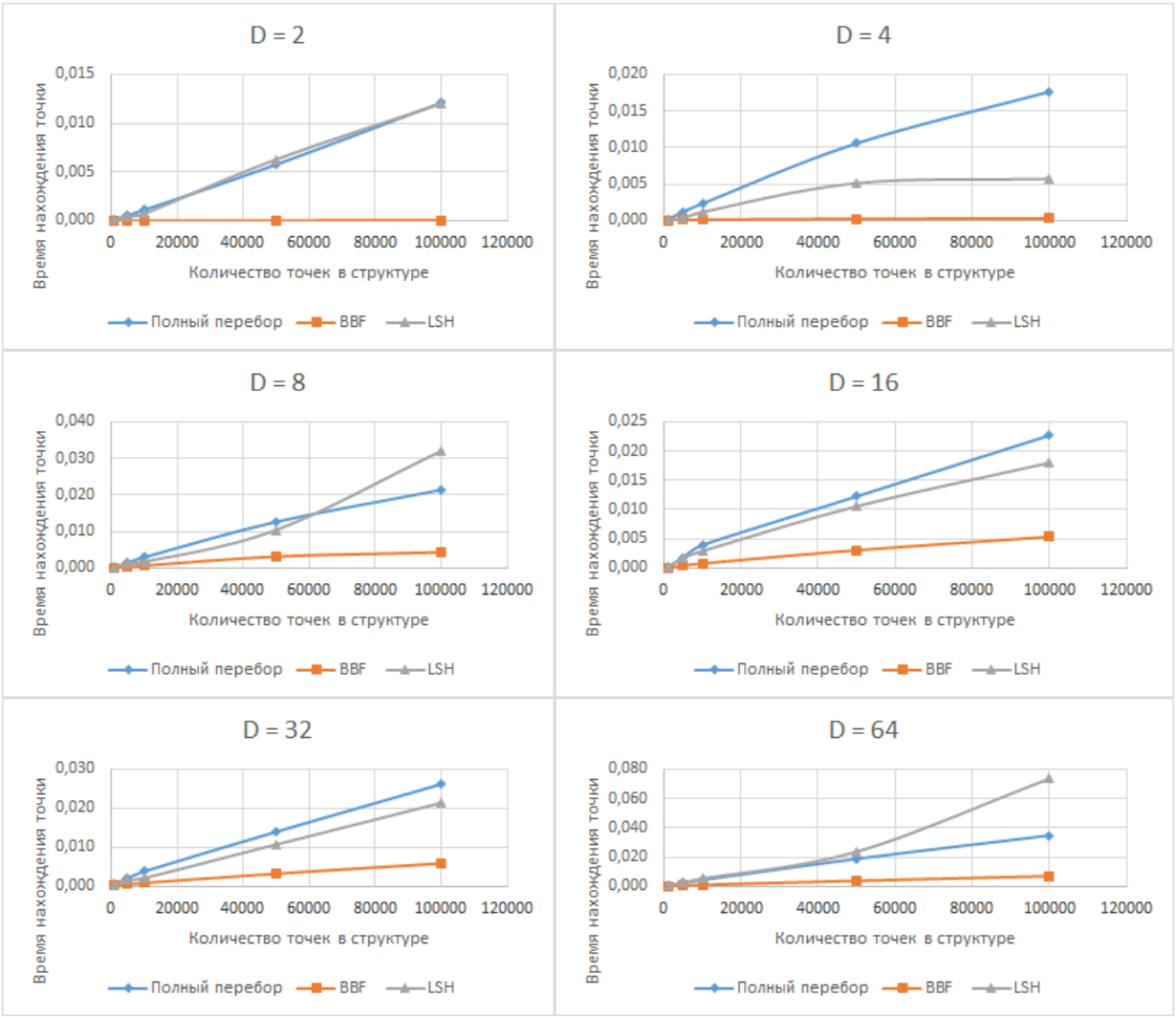

Рисунок 4 — Равномерное распределение на гиперсфере. Время исполнения

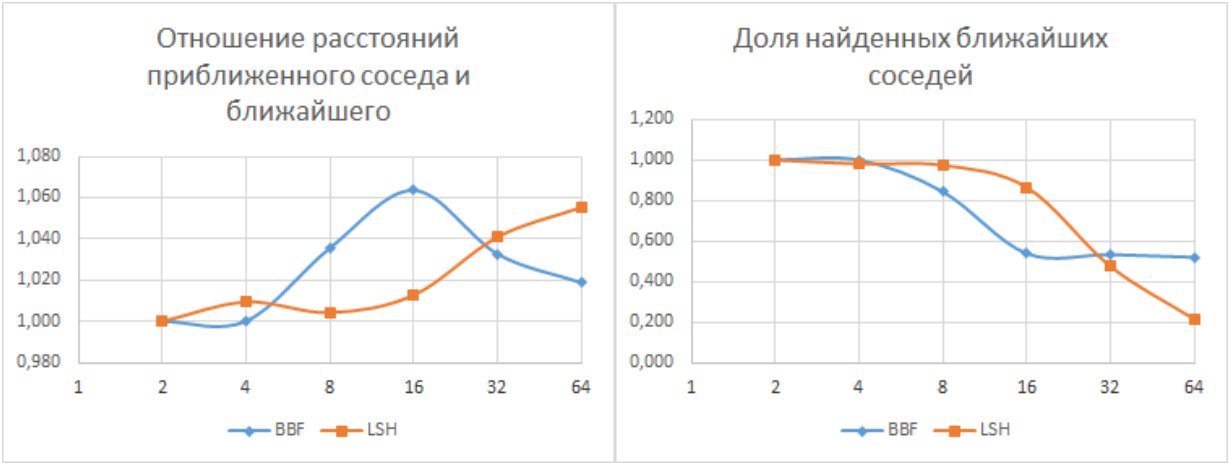

Рисунок 5 — Равномерное распределение в единичном гиперкубе. Ближайшие соседи

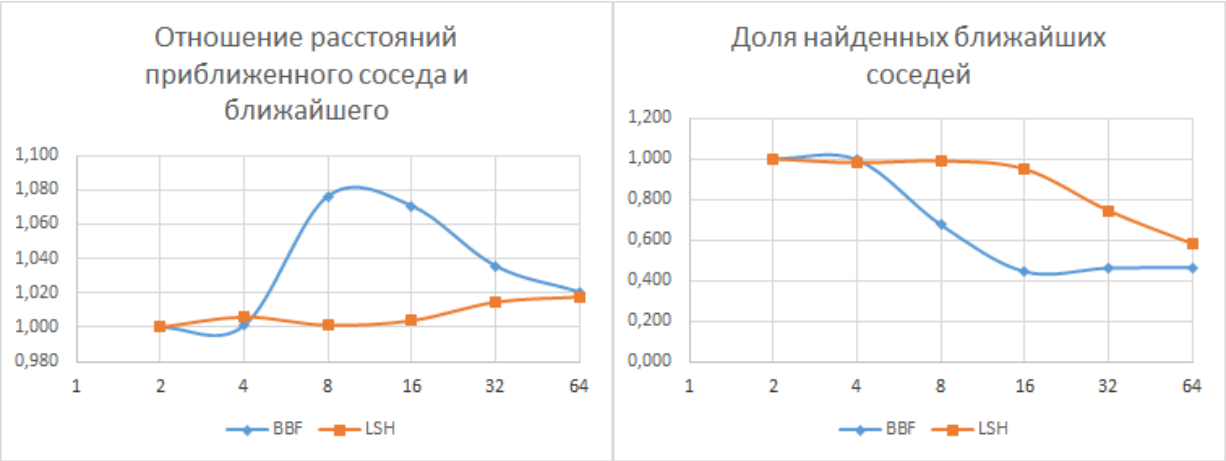

Рисунок 6 — Нормальное распределение в единичном гиперкубе. Ближайшие соседи

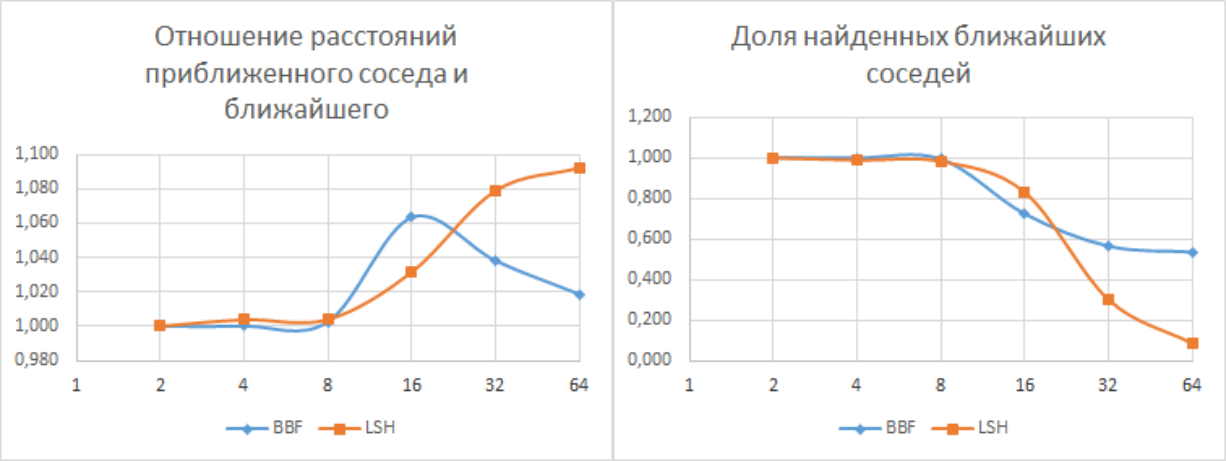

Рисунок 7 — Обратное нормальное распределение в единичном гиперкубе. Ближайшие соседи

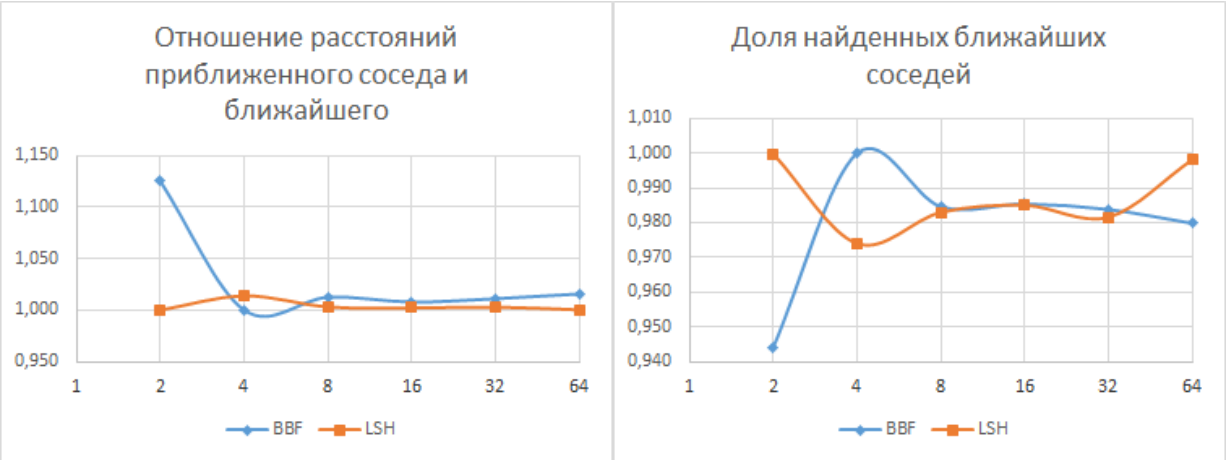

Рисунок 8 — Равномерное распределение на гиперсфере. Ближайшие соседи