НОВІ ЗАСОБИ КІБЕРНЕТИКИ, ІНФОРМАТИКИ, ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА СИСТЕМНОГО АНАЛІЗУ

Д.А. РАЧКОВСКИЙ

УДК 004.22 + 004.93′11

ИНДЕКСНЫЕ СТРУКТУРЫ ДЛЯ БЫСТРОГО ПОИСКА ПО СХОДСТВУ ВЕЩЕСТВЕННЫХ ВЕКТОРОВ. I

Аннотация. Дан обзор индексных структур для быстрого поиска по сходству объектов, представленных вещественными векторами. Рассмотрены индексные структуры на основе локально-чувствительного хэширования и их модификации. Изложены идеи конкретных алгоритмов, включая недавно предложенные. Обсуждена их взаимосвязь и некоторые теоретические аспекты.

Ключевые слова: поиск по сходству, ближайший сосед, ближний сосед, индексные структуры, локально-чувствительное хэширование, локально-чувствительная фильтрация.

ВВЕДЕНИЕ

Поиск по сходству — это вариант информационного поиска, при котором объекты из некоторой базы (множества) объектов, релевантные объекту-запросу, определяются по значениям некоторых мер (функций) сходства или несходства (расстояния) между представлениями объектов.

Будем считать представления объектов и меру расстояния/сходства, по которой выполняется поиск, заданными. Проще всего выполнить поиск по сходству вычислением величин сходств объекта-запроса и всех N объектов базы и возвращением объектов, удовлетворяющих условиям запроса. Такой «линейный» (или «последовательный») поиск часто оказывается недопустимо медленным, особенно для больших баз, сложных многокомпонентных представлений объектов и вычислительно сложных мер сходства.

Один подход к ускорению линейного поиска по сходству — быстро (хотя и приближенно) оценить величины всех расстояний/сходств (см. обзоры [1, 2]). Другой подход заключается в построении по базе такой структуры данных (индексной структуры, ИС), использование которой при выполнении запроса поиска по сходству позволило бы сократить количество вычислений сходства объекта-запроса с другими объектами по сравнению с линейным поиском (сублинейный поиск относительно N). Отметим, что алгоритмы обоих подходов зачастую ускоряют поиск ценой получения результатов, не полностью совпадающих с результатами (точного) линейного поиска по сходству, т.е. обеспечивают приближенный поиск по сходству.

Обзор ИС, разрабатываемых в рамках второго подхода и не требующих явного доступа к представлениям объектов, кроме как для вычисления расстояний/сходств, представлен в [3]. В обзоре [4] рассмотрены ИС поиска по сходству для объектов — бинарных векторов. В настоящей статье дан обзор основных типов ИС для быстрого поиска по сходству объектов, представленных вещественными векторами (компоненты которых — вещественные числа). Такие векторы широко применяются, например, для представления текстов, изображений, и др. [5–9]. Хотя для векторов можно использовать ИС из [3], векторы предоставляют больше средств для индексирования за счет возможностей доступа к своим компонентам и их подмножествам, явного задания границ областей векторного пространства, применения специализированных операций и алгоритмов (усреднение, кластеризация) и др.

Обзор состоит из двух частей, настоящая статья является его первой частью. Рассмотрены, главным образом, практически реализованные или имеющие перспективу такой реализации ИС, конструируемые без информации от учителя. В данной части обзора обсуждаются ИС для приближенного поиска по сходству на основе сохраняющего сходства хэширования (разд. 2). Введение в проблематику поиска по сходству представлено в разд. 1.

1. ОСНОВНЫЕ ПОНЯТИЯ

Рассмотрим определения используемых в ИС расстояний и сходств, типы запросов поиска по сходству, проблемы точного поиска и необходимость перехода к приближенному поиску для быстрого выполнения запросов, а также принципы построения и применения ИС.

1.1. Расстояния и сходства. Для оценки сходства между объектами используют расстояние, т.е. «несходство». Большим значениям сходства соответствуют малые значения расстояния. Многие расстояния являются метриками, т.е. подчиняются метрическим аксиомам, таким как неравенство треугольника и др. Для вещественных векторов $\mathbf{a}$, $\mathbf{b}$ размерности D широко применяются расстояния Минковского L_s различного порядка s: $||\mathbf{a}-\mathbf{b}||_s = \left(\sum_{i=1}^{D} |a_i - b_i|^s\right)^{1/s}$. Наиболее распространены метрические расстояния L_2 (евклидово $||\mathbf{a}-\mathbf{b}||_2$), L_1 (манхэттеново $||\mathbf{a}-\mathbf{b}||_1$), L_∞ (расстояние Чебышева $||\mathbf{a}-\mathbf{b}||_\infty$). При $0 < s < 1$ получаем дробные расстояния, которые не являются метриками.

Для сходств большие значения соответствуют более сходным объектам. Одна из мер сходства векторов — скалярное произведение: $\mathrm{sim}_{\mathrm{dot}}(\mathbf{a}, \mathbf{b}) \equiv \langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{D} a_i b_i$. Отметим, что $||\mathbf{a}-\mathbf{b}||_2^2 = ||\mathbf{a}||_2^2 + ||\mathbf{b}||_2^2 - 2\langle \mathbf{a}, \mathbf{b} \rangle$, а также то, что величина $\mathrm{sim}_{\mathrm{dot}}(\mathbf{a}, \mathbf{a})$ не является максимально возможной. Косинус угла между векторами определяется как $\cos(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / (||\mathbf{a}||_2 ||\mathbf{b}||_2) \equiv \mathrm{sim}_{\cos}$. Угол между $\mathbf{a}$, $\mathbf{b}$ определяет расстояние $\mathrm{dist}_{\mathrm{ang}} = \arccos(\mathbf{a}, \mathbf{b})$. Векторы единичной евклидовой нормы ($||\mathbf{a}||_2 = 1$) еще называют векторами (евклидовой) единичной сферы, которую обозначают S^{D-1}. Упорядочение таких векторов по возрастанию евклидова расстояния (или угла) эквивалентно упорядочению по убыванию скалярного произведения или косинуса угла. Для бинарных векторов широко используют расстояние Хэмминга $\mathrm{dist}_{\mathrm{Ham}}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{D} \{a_i \neq b_i\}$, где $\{\cdot\}$ — индикаторная функция. Сложность (время) вычисления приведенных расстояний/сходств составляет $O(D)$.

1.2. Точный и приближенный поиск по сходству и типы запросов

1.2.1. Запросы точного поиска по сходству. Объекты базы, которые являются ответом на конкретный запрос поиска по сходству, определяются типом запроса и его параметрами (задаются объектом-запросом, мерой расстояния/сходства и др.). Объекты-запросы могут не принадлежать базе. В настоящем обзоре объекты являются вещественными векторами.

Диапазонный запрос (range query) обозначим rNN. Он возвращает объекты базы, расстояния которых от объекта-запроса (по мере расстояния, заданной в запросе) не превышают радиуса запроса r. При некоторых r результатом диапазонного запроса может являться пустое множество объектов или все объекты конкретной базы. В последнем случае ускорение выполнения запроса rNN по сравнению с линейным поиском невозможно в принципе. Любой объект, являющийся ответом на диапазонный запрос, называют r-ближним соседом (near neighbor), обозначим его rNN1. Запрос rNN1 возвращает такой объект, если он имеется в базе.

Запрос k ближайших соседей (k nearest neighbors, kNN) возвращает k объектов базы, ближайших к объекту-запросу. Запрос одного ближайшего соседа, а также объект, который является ближайшим соседом, обозначим NN.

Запросы rNN и kNN с приведенными определениями — запросы точного поиска по сходству. Определения типов запросов в терминах не расстояния, а сход-

ства аналогичны. Другие типы запросов точного поиска по сходству в настоящем обзоре не рассматриваются.

1.2.2. Проблемы точного поиска и приближенный поиск по сходству. Время выполнения запросов поиска по сходству линейным поиском в базе из N объектов-векторов для мер расстояния/сходства с временем вычисления $O(D)$ составляет $O(ND)$.

Для данных размерности $D=1$ (каждый объект представлен одним числом) несложно быстро выполнять точные запросы поиска по сходству. Сортируют и запоминают N чисел, представляющих N объектов базы. Запрос NN выполняют поиском дихотомией. Для такой ИС затраты памяти составляют $O(N)$, а время поиска $O(\log N)$ в худшем случае [3].

Аналогичная ИС для $D \geq 2$ строит диаграмму Вороного, т.е. разбивает пространства на N областей, каждая их которых соответствует объекту базы и состоит из точек пространства, которые ближе к нему, чем к другим объектам базы. К сожалению, затраты памяти на хранение диаграммы Вороного составляют $O(N^{\lceil D/2 \rceil})$ [10]. Лучший известный алгоритм этого класса для поиска NN, хотя и обеспечивает время выполнения запроса $O(D^{O(1)} \log N)$ [11, 12], требует $N^{O(D)}$ памяти, что нереально для баз умеренного размера N, даже при малых D. Для алгоритмов с затратами памяти, близкими к линейным, время для случайного объекта-запроса (в среднем) $\min(2^{O(D)}, DN)$ [12]. Поэтому для баз с достижимым N точный поиск вырождается в линейный, даже при не слишком больших D, что подтверждается и на практике [13].

Анализ всех известных алгоритмов точного выполнения запроса NN с сублинейным от N временем показывает, что затраты времени или памяти растут экспоненциально от D [12]. Согласно предположению «проклятия размерности» [12] такая зависимость неизбежна при точном поиске по сходству для данных худшего случая (для объектов-запросов и объектов базы, которые требуют наибольшего времени выполнения запроса). Для некоторых алгоритмов экспоненциальная зависимость проявляется в терминах «внутренней размерности» (см. ссылки в [3]), которая может быть меньшей, чем «внешняя» размерность D.

Преодолеть (точнее, обойти) проклятие размерности удается переходом от точного к более быстрому, но приближенному поиску, допускающему отличие результатов от точного поиска. Редукция ряда других задач к задаче приближенного ближнего соседа описана в [12].

1.2.3. Запросы приближенного поиска по сходству. Приведем распространенные типы запросов приближенного поиска по сходству [14–16, 12].

Обозначим (c, δ)-rNN1 запрос вероятностного c-приближенного r-ближнего соседа (randomized c-approximate r-near neighbor); он с вероятностью, не меньшей $1-\delta$, возвращает любой объект базы с расстоянием от объекта-запроса, не превышающим cr ($c>1$), если существует объект базы с расстоянием от объекта-запроса, не превышающим r. Если такого объекта не существует, то возвращается объект с расстоянием, не превышающим cr, или ответ «нет».

Обозначим (c)-rNN1 запрос (c, δ)-rNN1 при $\delta = 0$ (детерминированный вариант).

Обозначим (δ)-rNN вероятностный диапазонный запрос (all randomized r-near neighbors); он возвращает все объекты базы с расстоянием от объекта-запроса, не большим r, с вероятностью, не меньшей $1-\delta$, для каждого такого объекта.

Обозначим c-NN запрос c-приближенного ближайшего соседа (approximate nearest neighbor); он возвращает объект базы, расстояние которого от объекта-запроса не более чем в $c>1$ раз превышает расстояние до точного ближайшего соседа.

Выполнение запроса $c(1+\gamma)$-NN (для некоторого $\gamma > 0$) можно редуцировать [16, 12] к последовательности запросов (c)-rNN1 с различными радиусами. Пусть Δ — конечное отношение наибольшего и наименьшего расстояний между объектами базы (включая и запрос), минимальное расстояние равно 1. Построим $O(\log_{1+\gamma} \Delta)$ штук ИС для (c)-rNN1 с радиусами r, принимающими значения

$0.5(1+\gamma)^i$ для $i = 0, 1, \ldots$ При поступлении запроса, выполнив поиск дихотомией в этих ИС (т.е. выполнив $O(\log \log_{1+\gamma} \Delta)$ запросов (c)-rNN1 с различными r), найдем ответ на запрос $c(1+\gamma)$-NN как объект, возвращенный при минимальном значении r. Для произвольных метрических пространств (с неограниченным Δ) в [16] предложена более эффективная (но непрактичная) редукция на основе ИС для запроса (c, δ)-rNN1. Чтобы избежать увеличения затрат памяти, для поиска приближенного NN часто используют другие ИС, без специфицированных теоретических гарантий (качество проверяют на базах).

Отметим, что обозначение NN в запросах (c)-rNN1, (c, δ)-rNN1, (δ)-rNN используется для ближнего соседа, а в запросах NN, kNN, c-NN, $c(1+\gamma)$-NN — для ближайшего соседа. Практические ИС, выполняющие некоторые вероятностные запросы, описанные в этом подразделе, с гарантиями на время выполнения для данных худшего случая, рассмотрены в разд. 2.

1.2.4. Меры качества поиска по сходству. Приближенный поиск по сходству востребован на практике, поскольку для многих применений достаточно получать приближенные результаты, но быстро.

Для алгоритмов приближенного поиска с количественными гарантиями рассматривают два типа отличий их результатов от результатов точного поиска. В случае детерминированных приближенных алгоритмов расстояние до найденных объектов не более чем на заданный множитель превышает расстояние до объектов, которые являются точным ответом на запрос. В случае рандомизированных алгоритмов Monte Carlo допускаются false negatives: алгоритм может не возвратить объекты, которые являются ответом на запрос (однако вероятность этого по реализациям алгоритма и время поиска при анализе ограничивают для данных худшего случая). Алгоритмы Monte Carlo, выполняющие вероятностные приближенные запросы, могут давать оба отличия. Рандомизированные алгоритмы Las Vegas работают без false negatives, при анализе ограничивают среднее по всем реализациям алгоритма время поиска для данных худшего случая (а не для среднего, как утверждалось в [4]).

На практике пользователей часто интересуют не теоретические гарантии, а время выполнения запросов в экспериментах на конкретных базах. Это время обычно зависит от выбора параметров ИС. Для точного поиска лучшей является ИС, обеспечивающая наименьшие затраты времени. Для приближенного поиска выбор параметров ИС обычно позволяет достичь некоторого компромисса между качеством результатов поиска (степенью их отличия от результатов точного поиска) и временем поиска. Поэтому ИС для приближенного поиска сравнивают по (усредненному) времени выполнения запроса при фиксированном качестве поиска или по значению меры качества поиска при одинаковом времени поиска. Важной характеристикой также являются затраты памяти ИС (квадратичные затраты неприемлемы).

Часто используют меры качества выполнения конкретного запроса, известные [17–19] как полнота (recall), равная $n1 / n2$, и точность (precision), равная $n1 / n3$, где $n1$ — количество возвращенных объектов, совпавших с релевантными запросу объектами в базе, $n2$ — количество релевантных запросу объектов в базе, $n3$ — количество возвращенных объектов базы. Для запроса kNN $n2 = n3 = k$, поэтому точность равна полноте. В качестве релевантных объектов используют объекты, возвращаемые запросами точного поиска либо указанные экспертом. «Хорошими» соседями могут быть не только точные соседи, но и объекты базы, близкие к точным соседям. Поэтому качество поиска также измеряют отношением суммы (или среднего) расстояний объекта-запроса до возвращенных объектов к соответствующей величине для эталонного результата (distance ratio metric) и др. [19].

При выполнении множества запросов полученные для индивидуальных запросов значения мер качества усредняют. Так, для запроса NN полноту (равную точности) измеряют как процент запросов, для которых был возвращен эталонный NN [20].

1.3. Общие принципы конструирования и применения индексных структур. В обеих частях обзора (часть II готовится к публикации) рассматриваются в основном ИС, при построении которых некоторым образом разбивают множество объектов базы на подмножества (в одних типах ИС они не пересека-

ются, а в других их пересечение возможно). Некоторые типы ИС хранят и используют информацию об областях пространства объектов, которым принадлежат эти подмножества. Иногда для одной базы конструируют несколько ИС с различными разбиениями на подмножества.

Различают статические и динамические ИС. В статических алгоритм конструирования ИС предполагает наличие всей базы объектов. В динамических конструирование осуществляется операциями вставки объектов по мере поступления данных.

При выполнении запроса применяют процедуры, которые можно считать вариантами двухэтапной стратегии фильтрации и уточнения (filter-and-refine, F&R). На первом этапе осуществляют быстрый отбор объектов-кандидатов. Результаты первого этапа уточняются на втором этапе (обычно с использованием линейного поиска среди объектов-кандидатов по мере расстояния/сходства, заданной в запросе). Стратегию F&R иногда применяют многократно при выполнении одного запроса. Если суммарное количество объектов–кандидатов меньше количества объектов базы N и их отбор выполняется достаточно быстро, можно получить ускорение выполнения запроса относительно линейного поиска (сублинейный поиск).

Для точного поиска при выполнении запроса кандидаты выбираются так, что они гарантированно содержат ответ на запрос точного поиска. Многие ИС точного поиска по сходству используют разновидности метода ветвей и границ (branch and bound, B&B) (см. обзор [3]). Однако сублинейное время выполнения запросов точного поиска не гарантируется и не наблюдается на практике (подразд. 1.2.2).

В базовом варианте ИС для приближенного поиска по сходству на основе сохраняющего сходство хэширования ставят в соответствие объектам такие хэш-значения, что вероятность совпадения хэшей сходных объектов выше, чем несходных. Объекты с одинаковым значением хэшей образуют подмножества в ИС. При выполнении запроса объектами-кандидатами являются подмножества с тем же значением хэша, что и объект-запрос. Это обеспечивает извлечение объектов-кандидатов, сходных с объектом-запросом, и гарантирует сублинейное время выполнения некоторых типов запросов вероятностного приближенного поиска (в частности (c, δ)-rNN1).

2. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ СОХРАНЯЮЩЕГО СХОДСТВО ХЭШИРОВАНИЯ

В локально-чувствительном хэшировании (LSH), в отличие от обычного, LSH-функции генерируют хэш-значения так, что вероятность их совпадения у сходных объектов выше, чем у несходных ([15, 16], обзоры [2] и [4]). Объекты с одинаковым хэшем образуют корзину. При поступлении объекта-запроса генерируют его LSH-хэш, извлекают из соответствующей корзины объекты базы и используют их в качестве кандидатов для линейного поиска по исходному расстоянию. Хэш-значения сходных объектов, сгенерированные одной LSH-функцией, могут оказаться различными, поэтому для повышения вероятности нахождения близких к запросу объектов объединяют списки кандидатов из корзин несколько независимых LSH-функций, аналогично лесам деревьев (см., например, [20]). Индексные структуры на основе LSH — одни из немногих реализованных практически для некоторых типов запросов приближенного вероятностного поиска по сходству с гарантиями сублинейного времени в худшем случае.

2.1. Индексная структура LSH для приближенного поиска по сходству

2.1.1. LSH-функции. Семейство F хэш-функций h является (r_1, r_2, p_1, p_2)-чувствительным для любых объектов a, b при выполнении условий: если $\text{dist}(a, b) \le r_1$, то $\Pr_F\{h(a) = h(b)\} \ge p_1$, а если $\text{dist}(a, b) \ge r_2$, то $\Pr_F\{h(a) = h(b)\} \le p_2$ [15, 16], где вероятность Pr соответствует случайному равномерному выбору хэш-функций h из семейства. Для полезного LSH-семейства $r_1 < r_2$ и $p_1 > p_2$. Это определение LSH — на основе зазора расстояний между сходными и несходными объектами. Аналогично можно дать определение LSH на основе зазора сходств [21].

В другом определении LSH на основе монотонности величины сходства [22] для LSH-функций должно выполняться $\Pr_F\{h(a) = h(b)\} = \text{sim}(a, b)$, где sim — мера сходства объектов, принимающая значения в $[0, 1]$. Отметим, что sim может также являться монотонно возрастающей функцией со значениями в $[0, 1]$ от некоторой другой функции сходства [21]. Аналогичное определение существует и

для расстояний. Из определения LSH на основе монотонности следует определение на основе зазора [21]. Отметим обобщение LSH вида $\Pr_F\{h_1(a) = h_2(b)\} = f(\mathrm{dist}(a, b))$, где $f(\cdot)$ может не являться убывающей функцией dist [23].

Пример LSH-функции. Для угла θ между векторами широко применяется семейство LSH-функций SimHash [22]: $h_{\mathbf{r}}(\mathbf{a}) = \mathrm{sign}(\langle \mathbf{r}, \mathbf{a}\rangle)$, $\mathbf{r}$ — случайный вектор размерности D с компонентами, которые являются независимо и одинаково распределенными (i.i.d.) случайными величинами (с.в.) из гауссова распределения ($\mathbf{r} \sim \mathrm{Norm}(\mathbf{0}, \mathbf{I}_D)$), $\mathrm{sign}(z) = 1$ при $z \geq 0$ и $\mathrm{sign}(z) = -1$ при $z < 0$. Для этой LSH-функции $\Pr\{h(\mathbf{a}) = h(\mathbf{b})\} = 1 - \theta(\mathbf{a}, \mathbf{b})/\pi$. Другие примеры LSH рассмотрены в подразд. 2.2.

2.1.2. Базовая индексная структура LSH. При конструировании ИС на основе LSH создают и запоминают L LSH-функций g, каждая из которых является конкатенацией m LSH-функций h, случайно выбранных из семейства LSH. Функция g является (r_1, r_2, p_1^m, p_2^m)-чувствительной. (При анализе запроса (c, δ)-rNN1 полагают $r_1 = r$, $r_2 = cr$.) Для каждой из L функций g конструируют свою хэш-таблицу. Каждый объект базы y помещают в одну корзину каждой из L хэш-таблиц (корзину, которая соответствует его хэшу $g(y)$ для этой таблицы). Таким образом, объекты с одинаковыми значениями g попадают в одну корзину. Так как при $N << 2^m$ большинство ячеек таблиц пусты, к LSH-хэшам дополнительно применяют обычное хэширование. Другие особенности реализации базовой ИС LSH описаны в [24].

Для выполнения запроса в ИС LSH используют стратегию F&R. Хэши объекта-запроса генерируют теми же LSH-функциями, что применялись для конструирования ИС. Объекты-кандидаты извлекают из корзин, хэши которых совпадают с хэшами объекта-запроса (коллизия). Для выполнения запроса (c, δ)-rNN1 выбирают фиксированное число объектов из этих корзин, например $3L$, включая дубликаты («Стратегия 1» [15]). Для запроса (δ)-rNN используют все объекты из этих корзин («Стратегия 2» [15]). Окончательный результат получают линейным поиском в списке кандидатов по мере расстояния/сходства запроса.

Используя $\rho = \log 1/p_1 / \log 1/p_2 = \log p_1 / \log p_2 \in (0, 1)$, можно записать $L = N^{\rho}$. Получаем затраты памяти на LSH-структуру из L таблиц и базу $O(DN + N^{1+\rho})$. Время поиска состоит из времени вычисления хэш-функций $O(Lm) = O(N^{\rho} \log N)$ и времени вычисления расстояний до кандидатов $O(LD) = O(N^{\rho} D)$. Стратегия 1 гарантирует время поиска, так как количество кандидатов ограничено. Таким образом, $\rho < 1$ (чувствительность) характеризует «качество» LSH-функций (насколько они позволяют ускорить поиск относительно линейного, каковы затраты памяти). Обычно $\rho = 1$ для $c = 1$ и $\rho \to 0$ для $c \to \infty$.

При расчете параметров для стратегии 2 (запрос (δ)-rNN) выбирают [24] $L = \lceil \log \delta / \log(1 - p_1^m) \rceil$, а значение m, минимизирующее время запроса, определяют в экспериментах. В [25] выбирают L, исходя из имеющейся памяти, а $m = \lceil \log(1 - \delta^{1/L}) / \log p_1 \rceil$.

2.2. Примеры LSH-функций. Расчет ИС LSH для конкретной меры расстояния/сходства требует наличия для нее LSH-семейства (с известными характеристиками, которые не зависят от конкретного набора объектов базы). Арсенал LSH-семейств см. в обзорах [15, 26, 2, 27]. Большинство LSH-семейств разработано для векторов (но см. [28, 29] для ядерных сходств).

Для пространств L_s, $s \in (0, 2]$, предложены семейства LSH [24] с использованием векторов $\mathbf{r}$ с компонентами — i.i.d. с.в. из s-устойчивых распределений: $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a}\rangle + U)/w \rfloor$, где w — параметр, определяющий размер хэш-корзины, U — i.i.d. с.в. из равномерного распределения в $[0, w]$: $U \sim \mathrm{Unif}(0, w)$. Примерами s-устойчивых распределений являются: распределение Коши для $s = 1$ и Гаусса для $s = 2$. Для этих семейств $\rho = \max\{1/c^s, 1/c\} + o(1)$. Обозначим LSH-функции этого семейства L2LSH для L_2 и L1LSH для L_1.

В [30] для L_2 получено $\rho = 1/c^2 + o(1)$, что является оптимальным значением. Вначале выполняют случайное проецирование (см. обзор [1]) исходных век-

торов в векторы сниженной размерности, затем выбирают множество случайных векторов. Значением LSH-хэша является идентификатор первого случайного вектора, шару вокруг которого с заданным радиусом принадлежит хэшируемый вектор. В более практических вариантах хэш-значением является ближайшая точка решетки [30, 31].

Чтобы расстояние имело семейство LSH-функций, необходимо (но не достаточно) чтобы оно было метрическим [22] и имело изометрическое вложение в L_1 [32]. Если каждое из двух сходств имеет LSH-функцию, их произведение также ее имеет. Необходимым и достаточным условием сохранения свойства LSH для функции преобразования сходства является ее принадлежность к классу (масштабированных) производящих функций вероятностей. В [33] рассматривают сходства, которые не имеют LSH, и вводят понятие искажения, чтобы охарактеризовать аппроксимацию таких сходств теми, что имеют LSH. Показаны плотные верхние и нижние границы искажения для известных сходств без LSH, подтвержденные экспериментами.

Отметим, что наличие LSH-семейств для L_1 в принципе позволяет использовать ИС LSH для приближенного поиска по представлениям объектов и их расстояниям, которые можно преобразовать в L_1 с некоторым искажением (например, расстояние редактирования, бульдозера и др. [34, 16, 35, 1]). Возможно применение LSH-функции и для формирования бинарных или целочисленных компонентов векторов (скетчей) для оценки соответствующих LSH-функциям расстояний/сходств между исходными объектами по эмпирической вероятности совпадения хэшей ($\mathrm{dist}_{\mathrm{Ham}} / D$ скетчей [2]) согласно определению LSH на основе монотонности.

В качестве примера LSH рассмотрим недавно появившиеся LSH-функции для единичной сферы.

2.2.1. LSH-функции и реализация поиска по евклидову расстоянию единичных векторов. Оптимальное и эффективно реализуемое семейство LSH-функций для евклидова расстояния векторов на единичной сфере проанализировано в [36]. Из соотношений, приведенных в подразд. 1.1, следует его применимость для поиска по некоторым другим мерам. Семейство использует случайно повернутые гипероктаэдры (единичные шары в L_1) [37, 38]. Для получения значения LSH-хэша вектора его умножают на случайную i.i.d. гауссову матрицу $R\,(D\times D)$, свою для каждой LSH-функции, результат нормируют к единичной норме. Затем находят ближайшую к полученному вектору вершину гипероктаэдра (максимальный компонент вектора), номер которой и является хэш-значением. Это вычислительно эффективно, так как расстояние эквивалентно скалярному произведению, а в векторах вершин ненулевой только один компонент.

Для полного диапазона расстояний $0< r< cr< 2$ получена зависимость $\rho = (1/c^2)(4-c^2r^2)/(4-r^2)+o(1)$, где $o(1)\to 0$ при $D\to\infty$. При $r=\sqrt{2}/c$, $rc=\sqrt{2}$ эта LSH достигает оптимального значения $\rho = 1/(2c^2-1)$, как и зависящие от данных LSH для L_2 [39] (которые, однако, практически не реализуемы вследствие большого времени вычисления, подразд. 2.6.3). Дана также нижняя граница ρ для различных соотношений p_1, p_2, которая показывает, что гипероктаэдрный LSH близок к оптимальному.

Для ускорения хэширования (без теоретических гарантий) [36] используют hashing trick снижения размерности и быстрое умножение на ортогональный матричный конвейер (см. обзор [1]), а также специализированный мультипробный LSH (подразд. 2.3.1). Программная реализация FALCONN ускоряет запрос NN на ряде баз по сравнению с SimHash в 4–10 раз для больших N [36] (хотя получение значения хэша требует большего времени). Отметим, однако, что для SimHash также возможно ускорение применением быстрых матричных конвейеров [2].

В [40] исследован гиперкубический LSH, где разбиение на области производится ортогональными гиперплоскостями (областями Вороного вокруг вершин гиперкуба). Предварительно выполняется случайное вращение хэшируемого вектора (для быстрого псевдослучайного вращения также можно получить теоретические гарантии [41]). Для гиперкубического LSH ρ находится между SimHash и гипероктаэдрным, однако некоторые применения работают быстрее с гиперкубическим LSH [40].

2.3. Улучшения базовой ИС LSH

2.3.1. Мультипробный LSH. Большое количество L таблиц LSH требует значительных затрат памяти. Для уменьшения L (с сохранением качества поиска) в [42, 43] предложено изменить процедуру выполнения запроса так, чтобы извлекать кандидатов не только из одной корзины таблицы, соответствующей LSH-хэшу запроса, но и из других корзин, «близких» к объекту-запросу. В таком «мультипробном» (multi-probe) LSH используют как модификации вектора запроса [42], так и (более быстрые) модификации LSH-хэша запроса [43]. Схему генерации модификаций и порядка обхода корзин конструируют так, чтобы она была вычислительно эффективной и в первую очередь обеспечивала посещение наиболее «перспективных» корзин для конкретного объекта-запроса. В [43] получены такие же точность и время запроса kNN, как и у базового LSH, при уменьшении L более чем в 10 раз.

Экспериментальные исследования вариантов модификации хэш-кода запроса [44, 45] показали хорошие результаты. В экспериментах [36] по поиску (точного) NN при одинаковых $1-\delta$ и L время запроса для мультипробного варианта уменьшилось более чем в 10 раз (за счет большего оптимального значения m, обеспечивающего меньшее количество кандидатов).

Мультипробный LSH используется также в теоретических работах. В [46] его применяют для получения плавного компромисса между затратами памяти и временем поиска. Для мультипробного варианта адаптивной ИС LSH [47] (и подразд. 2.3.3) для запроса (δ)-rNN теоретически обоснована не только экономия памяти, но и ускорение выполнения запроса.

Для запроса rNN (без false negatives) в L_s, $s \in [1, \infty]$, предложено [48] семейство LSH-функций вида $h_s(\mathbf{x}) = \lfloor \langle \mathbf{x}, \mathbf{r} \rangle / (rD^{1-1/s}) \rfloor$, где r — радиус запроса, $\mathbf{r}$ — вектор i.i.d. с.в. из ограниченного центрированного распределения: $-1 \le r_i \le 1$, $\mathrm{E}\{r_i\} = 0$. В качестве кандидатов здесь требуется возвращать по запросу $\mathbf{x}$ все объекты $\mathbf{y}$ базы, для которых $||g(\mathbf{x}) - g(\mathbf{y})||_\infty \le 1$. Для этого предлагается использовать 3^m модификаций хэша запроса или 3^m модификаций хэша каждого объекта базы. К сожалению, кандидатами могут являться объекты с большим расстоянием до запроса (до cr, $c \ge \max\{D^{1/2}, D^{1-1/s}\}$). Ускорение запроса рассмотрено в [49].

2.3.2. Модели и выбор параметров ИС LSH для конкретной базы. Учитывая особенности данных конкретной базы, можно улучшить характеристики поиска в ИС LSH (см. также подразд. 2.6.3). В [24] выбор параметров проводят перебором их комбинаций на программной реализации реальной LSH-структуры. Это вычислительно сложно, даже если работать с подмножеством базы. Для решения этой проблемы предложены [50, 45] статистические модели ИС LSH для L2LSH и вероятностного запроса (k)NN. Модели используют распределения расстояний между запросом и (k)NN, а также между объектами базы.

В модели мультипробного LSH [50] для максимизации полноты поиска выбирается максимальное L, исходя из имеющейся памяти. При заданном среднем значении полноты поиска на базе офлайн настраиваются w в $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a} \rangle + U) / w \rfloor$ и количество конкатенируемых функций m для минимизации времени запроса (средней доли объектов базы, которую составляют кандидаты). Для конкретного объекта-запроса на основе расстояний до текущих kNN настраивается количество модификаций запроса для достижения заданной полноты поиска. Детальная модель [45] минимизирует среднее время поиска для заданной вероятности ошибки, меньшей δ (с упрощенной модификацией запроса, без адаптации к запросу), и может использоваться для настройки параметров для различных типов запросов.

2.3.3. Адаптация к конкретному запросу. Адаптация к конкретному вектору-запросу рассматривалась в модели мультипробного LSH [50]. В [31] предложено улучшать соотношение скорость–точность поиска (в L_2) выбором при выполнении запроса тех LSH-таблиц из их имеющегося набора, в которых вектор-запрос с наибольшей вероятностью попадет в одну корзину с ближайшим соседом. Используется критерий релевантности хэш-корзины на основе расстояния запроса до центра корзины.

В работе [47] рассматривают запрос (δ)-rNN. Для базовой ИС LSH и «трудных» запросов среднее количество кандидатов может составлять до $O(nL)$, где n — количество объектов базы, являющихся ответами на запрос. Для решения этой проблемы в многоуровневом адаптивном LSH [47] строят наборы (уровни) LSH-таблиц для различных сочетаний m и L (L растет с увеличением m). При выполнении запроса в этой ИС среднее количество кандидатов $\Theta(n(N/n)^{\rho})$ всегда меньше N, а вариант ИС с модификацией запроса приближает его к нижней границе $O(N^{\rho}+n)$. Для реализации подхода необходимы большие затраты памяти.

В практическом алгоритме [25] для запроса (δ)-rNN конструируют ИС LSH для фиксированного L, выбирая $m=\lceil \log(1-\delta^{1/L})/\log p_1 \rceil$ (см. подразд. 2.1.2, стратегия 2). Чтобы при выполнении конкретного запроса выбрать использование базовой ИС LSH либо линейного поиска, предлагается быстро оценивать время запроса для LSH. Для этого в HybridLSH [25] с каждой хэш-корзиной ассоциирована структура Hyperloglog [51], которая позволяет аппроксимировать число объектов без дубликатов в выбранных хэшем запроса L корзинах.

2.3.4. Исключение кандидатов по LSH-хэшам. Оценки вероятности совпадения LSH-хэшей объектов могут использоваться для быстрой обработки кандидатов в целях исключения тех из них, для которых мала вероятность превышения сходством порогового значения [52]. Принимать решение об исключении можно после оценки малой части хэшей из их общего количества. Например, если всего 1000 хэшей и из первых 100 только 10 совпадают, то очень маловероятно, что сходство объектов превышает 0.8. Более того, можно оценить и саму величину сходства с заданной пользователем точностью. Требующиеся для этого функции распределения вероятности величины сходства при условии совпадения заданной доли хэш-значений получены в [52] для LSH MinHash [27] (сходство Жаккара) и SimHash [22] (с модификацией для косинусного сходства). Рассмотрено также применение для (нормированных) ядерных функций сходства [52].

2.4. Поиск приближенных ближайших соседей в индексных структурах LSH. Теоретически обоснованная базовая ИС LSH для запроса (c,δ)-rNN1 требует конструирования набора таблиц со своими параметрами для каждой совокупности r, c. При выполнении запроса вероятностного c-NN последовательностью запросов (c,δ)-rNN1 с различными радиусами (см. подразд. 1.2.3) это приводит к большим затратам памяти. Для экономии предложены алгоритмы, эмулирующие запросы с различными радиусами в одной ИС на основе LSH.

В LSH-forest [53] количество m конкатенируемых функций h переменное — такое, чтобы объекты базы получили различные значения LSH-хэшей. Вместо хэш-таблиц используют экспериментально выбираемое количество L префиксных деревьев, в которых объектам соответствуют листовые узлы. Увеличение радиуса запроса эмулируют выбором объектов-кандидатов, хэши которых имеют частичное совпадение с хэшем запроса. Вначале в каждом дереве находят лист, имеющий наибольший совпадающий префикс с запросом. Затем осуществляют синхронный поуровневый обход L деревьев с самого нижнего уровня к корню, извлекая листья-потомки родительских узлов, пока число кандидатов не достигнет заданного. Из кандидатов выбирают k ближайших к запросу. Теоретический анализ [54] рассматривает разновидность LSH-forest как практический вариант зависящего от данных LSH (подразд. 2.6.3) для запроса (c,δ)-rNN1 (с меньшим значением ρ, чем для не зависящего от данных LSH).

В LSB-tree [55] для поиска в L_2 вначале снижают размерность случайным проецированием гауссовой i.i.d. матрицей. Векторы преобразуют в значения Z-порядка [56], векторы базы индексируют B-tree [57]. Используют одиночное дерево или их лес. Выполнение серии запросов с возрастающим радиусом эмулируют извлечением векторов из листовых узлов в порядке убывания количества совпадающих старших битов Z-значений.

В SKLSH [58] используют ключи, являющиеся конкатенацией значений хэшей L2LSH. Предложено метрическое расстояние между ключами (на основе совпаде-

ния префиксов и отличия в следующем компоненте), величина которого отражает расстояние L_2 между векторами. Это позволяет упорядочить ключи при конструировании ИС так, что ключи векторов базы, близких к вектору запроса, хранятся в памяти рядом. Использование B+tree [57] позволяет при поиске извлекать в первую очередь объекты с ключами, более близкими к ключу объекта-запроса. Преимущество над LSB-tree и C2LSH (см. далее) достигается за счет извлечения гораздо большего числа объектов-кандидатов при значительно меньшем числе операций случайного ввода-вывода. В [59] для ускорения применяют ранний останов SKLSH.

Ранжировать кандидатов по количеству их встречаемости в списке кандидатов предложено в [60]. В FBLSH [61] и C2LSH [62] в качестве кандидатов выбирают объекты базы, для которых количество совпадений индивидуальных LSH-хэшей h с LSH-хэшами запроса превышает пороговое значение. Аналогичный отбор используется в работах [63, 64].

В C2LSH [62] используют увеличение радиусов поиска $\{1, c, c^2, \dots\}$ с помощью модификации хэш-функций, в качестве которых применяют вариант L2LSH. Модификация осуществляется как $\lfloor h(\cdot)/c \rfloor$, где $h(\cdot)$ — значение хэш-функции для предшествующего радиуса.

В LazyLSH [63] выполняют запросы для расстояний L_s, $s \in [1/2, 1]$, с использованием варианта C2LSH для L1LSH. Радиус r для L_s, $s \in [1/2, 1]$, аппроксимируют через радиус r для L_1. В отличие от C2LSH, где увеличение r приводит к увеличению размера корзин, которое не зависит от объекта-запроса, в LazyLSH объединяют корзины, соседние с корзиной объекта-запроса, что повышает вероятность совпадения хэшей для близких к запросу векторов базы. При одновременном выполнении запроса для L_s с различными s удается оптимизировать общее время извлечения кандидатов. Подход можно использовать и с другими ИС для L_1.

В QALSH [64] при конструировании ИС для L_2 не используют разбиения на корзины, а индексируют с помощью B+tree значения проекции $\langle \mathbf{y}, \mathbf{r} \rangle$ векторов базы $\mathbf{y}$ на случайные направления $\mathbf{r}$. При поступлении запроса $\mathbf{x}$ объекты базы в виртуальной корзине находят поиском в B+tree в диапазоне $[\langle \mathbf{x}, \mathbf{r} \rangle - wr/2, \langle \mathbf{x}, \mathbf{r} \rangle + wr/2]$. Благодаря отсутствию фиксированного разбиения на корзины, запросы с радиусами r из $\{1, c, c^2, \dots\}$ могут выполняться с любым $c > 1$. В экспериментах QALSH превосходит C2LSH и LSB-forest.

В [65] извлекают объекты-кандидаты в порядке увеличения $\mathrm{dist}_{\mathrm{Ham}}$ их бинарных LSH-хэшей от хэшей объекта-запроса (до достижения количества кандидатов, эмпирически определяемого для задачи и для k). Для поиска по $\mathrm{dist}_{\mathrm{Ham}}$ используют структуру MIH [66] (выполняя запросы с увеличивающимся радиусом). Бинарные значения хэшей L2LSH получают выбором параметра w в $h(\mathbf{a}) = \lfloor (\langle \mathbf{r}, \mathbf{a} \rangle + U)/w \rfloor$. Этот BLSH возвращает более точных kNN за меньшее время, чем LSB, C2LSH, SKLSH, особенно для данных более высокой размерности ($D = 192$).

В отличие от рассмотренных подходов в Selective Hashing [67] предлагается строить набор ИС LSH для каждого радиуса поиска из $\{r, cr, c^2 r, \dots\}$, однако помещать объект только в корзину таблицы для одного из радиусов. Объекты базы из областей пространства с большой плотностью помещают в корзины с меньшим размером (соответствующим меньшему радиусу запроса). Это можно рассматривать как подстройку ИС локально к каждому объекту базы. При запросе кандидаты извлекают из таблиц, начиная с малого радиуса, пока в оставшихся ИС еще могут содержаться потенциальные kNN. Так как условие останова учитывает положение объекта-запроса относительно объектов базы, происходит адаптация к запросу. Этот практический подход обеспечивает затраты памяти на уровне базовой ИС LSH для одного радиуса и полноту поиска kNN на уровне ИС LSH для множества (увеличивающихся) радиусов.

2.5. Локально-чувствительная фильтрация и компромисс время/память. Базовая ИС LSH работает в симметричном режиме: значение ρ (см. подразд. 2.1.2) одинаково как для времени запроса, так и для затрат памяти (и времени вставки/удаления объектов). Однако для некоторых применений требуется асимметричный режим с различными компромиссами между увеличением затрат памяти и уменьше-

нием времени выполнения запроса. Плавный компромисс время/память, обеспечивающий сублинейное время поиска в L_2 для худшего случая для любых $c>1$, улучшающий результаты [46] для всех c, получен в [68] на основе подхода локально-чувствительной фильтрации (locality-sensitive filtering, LSF) [69].

ИС LSH ассоциируют корзину с каждым значением LSH-функции, обеспечивая исчерпывающее разбиение пространства объектов: каждый объект попадает в одну из корзин. LSF-фильтр отличается тем, что пропускает только часть входных объектов. Сходные объекты имеют большую вероятность преодолеть LSF-фильтр, чем несходные. LSF-фильтры могут использоваться индивидуально или комбинироваться последовательно и параллельно. В [70] для быстрой эмуляции большого количества LSF-фильтров предлагается получить объекты на выходе некоторого множества базовых фильтров, а затем результат на выходе одного эмулируемого фильтра находить пересечением объектов соответствующего ему подмножества базовых.

В корзине запоминают объекты, которые остались после прохождения LSF-фильтра или их совокупности на пути к ней. Один и тот же объект может принадлежать различным корзинам. При выполнении запроса кандидатами становятся объекты из корзин, в которые прошел объект-запрос. Параметры фильтров при конструировании ИС и при выполнении запроса могут отличаться, что и обеспечивает упомянутый компромисс.

При создании LSF-фильтров для поиска на единичной сфере S^{D-1} в [69] и [68] используют идею разбиения сферы на области большим количеством малых сферических сегментов (spherical cap, SC). Вектор **a** принадлежит к SC (r, t), если $\langle \mathbf{r}, \mathbf{a} \rangle \geq t$, **r** — случайный гауссов вектор или случайный вектор с S^{D-1}. (Отметим, что аналогичное преобразование применяется для получения разреженных бинарных/тернарных векторов в [71–74].) Порог t_u, используемый при построении ИС, может отличаться от t_q при обработке запроса.

При большом количестве SC определить, к каким из них принадлежит вектор, вычислительно трудоемко. Для решения этой проблемы в [68] каждой области-корзине соответствует пересечение нескольких SC. Для определения, к какой корзине принадлежит объект, используют дерево с $A+1$ уровнями, где каждый узел соответствует SC. Коэффициент ветвления не превышает M, поэтому на уровне A не более M^A листьев. Узлы хранят по одному i.i.d. случайному вектору из Norm $(\mathbf{0}, \mathbf{I})$. При построении ИС множество векторов-объектов некоторого узла разбивается его сыновьями на M (возможно пересекающихся) подмножеств, каждое определяется по $\langle \mathbf{r}, \mathbf{y} \rangle \geq t_u$. Для узлов с непустым множеством объектов разбиение повторяют, пока не достигают A-го уровня дерева.

При обработке вектора-запроса стартуют с корня и переходят на сыновей, для которых $\langle \mathbf{r}, \mathbf{x} \rangle \geq t_q$. Кандидатами являются векторы из узлов A-го уровня, до которых дошел вектор-запрос. Выбор параметров A, M, t_u и t_q зависит от r, c, а также ρ_u и ρ_q. Если $t_u = t_q$ то $\rho_u = \rho_q$ и получают режим LSH. Эта ИС практически не реализована.

В [69] предложен реализованный алгоритм. Листу соответствует сферический сегмент, однако вектор, его определяющий, состоит из $A = \Theta(\log D)$ векторов-частей размерности D / A. Для каждой части случайно генерируют M векторов с S^{D-1} и нормируют их, чтобы обеспечить единичную норму полного вектора. Из векторов-частей можно составить M^A различных векторов **r** размерности D, соответствующих корзинам. Показано, что вероятность попадания векторов в корзину с составным **r** близка к вероятности для случайных **r** с S^{D-1}.

Для быстрого определения, к какой корзине принадлежит входной вектор, строят ИС в виде A-уровневого дерева с коэффициентом ветвления M, где каждому узлу соответствует случайный вектор-часть. Для каждой из A частей входного вектора вычисляют M скалярных произведений с векторами узлов, что позволяет быстро определить (соответствующие листьям) корзины, к которым принадлежит входной вектор (сложность вычислений в A раз превышает количество корзин, к которым принадлежит вектор). Значения t_u и t_q могут отличаться.

В симметричном режиме $\rho_u = \rho_q$ такая ИС позволяет получить меньшие значения ρ, чем сферический и гипероктаэдрный LSH (см. подразд. 2.2.1) для данных с $D = O(\log N)$. Для худшего случая эти независимые от данных ИС позволяют достичь гибкого компромисса время/память $(c^2+1)\sqrt{\rho_q} + (c^2-1)\sqrt{\rho_u} \geq 2c$. Расширение с S^{D-1} на все L_2 (подразд. 2.6.1) позволяет для всех связанных этим выражением $c>1$, $r>0$ и ρ_u, ρ_q получить ИС для запроса (c,δ)-rNN1 в L_2 с затратами памяти $N^{1+\rho_u+o(1)}$ и временем запроса $N^{\rho_q+o(1)} + DN^{o(1)}$. Для $\rho_u = \rho_q$ получают $\rho \geq 1/c^2$. Расширение на другие L_s рассмотрено в подразд 2.6.1. Зависимые от данных ИС позволяют улучшить характеристики (подразд. 2.6.3).

2.6. Некоторые теоретические аспекты и адаптация к данным

2.6.1. Расширения на L_s. Результаты анализа ИС для поиска в S^{D-1} распространяют на все евклидово пространство, используя редукцию L_2 к S^{D-1} [75], сохраняющую отношение расстояний L_2 (с мультипликативным искажением).

Чтобы расширить результаты для L_2 на L_s, используют детерминированное покомпонентное вложение [76] (с мультипликативным и аддитивным искажением) из L_s ($1 \leq s < 2$) в $L_2^{2/s}$ (с соответствующим изменением r). Расширения на L_s базовой ИС LSH в L_2, для которой $\rho = 1/c^2 + o(1)$ (см. [30] и подразд. 2.2), дают для L_s значение $\rho = 1/c^s + o(1)$ [12].

Чтобы расширить результаты анализа ИС для приближенного поиска по скалярному произведению векторов с S^{D-1} (что эквивалентно поиску по евклидову расстоянию, см. подразд. 1.1) на L_s ($0 < s \leq 2$), в [70] предлагается использовать приближенное вложение ядерного сходства D-мерных векторов $\kappa(\mathbf{a}-\mathbf{b}) = \exp(-||\mathbf{a}-\mathbf{b}||_s^s)$, $0 < s \leq 2$, в скалярное произведение векторов на S^{D-1} ([77] и обзоры [1, 2]). Тогда выполнение запроса (c,δ)-rNN1 в L_s эквивалентно выполнению аналогичного запроса по $\mathrm{sim}_{\mathrm{dot}}$ векторов с S^{D-1} со значением сходства $\exp(-r^s)$ (с поправкой на искажение вложения).

Отметим, что в практических ИС редукции этого подраздела не встречаются.

2.6.2. Границы. Анализ предложенных алгоритмов поиска по сходству (не всегда практически реализуемых) позволяет получить верхние границы затрат памяти и времени запроса. Нижние границы для вещественных векторов с расстоянием L_s обычно получают через доказанные (для некоторых «трудных» распределений данных) нижние границы для поиска бинарных векторов $\{0,1\}^D$ по $\mathrm{dist}_{\mathrm{Ham}}$, используя $\mathrm{dist}_{\mathrm{Ham}}(\mathbf{a},\mathbf{b}) \equiv ||\mathbf{a}-\mathbf{b}||_s^s$. Применение L_s^s вместо $\mathrm{dist}_{\mathrm{Ham}}$ трансформирует c в c^s в выражении для ρ [12]. Для более общего случая вещественных векторов нижняя граница не может быть меньшей, чем для бинарных.

Для базового LSH из нижней границы для расстояния Хэмминга $\rho \geq 1/c - o_D(1)$ [78] следует $\rho \geq 1/c^s - o_D(1)$ для L_s. Для широкого класса независимых от данных ИС list-of-points (который включает и LSH, и LSF) для бинарных векторов с $D = \omega(\log N)$ показана [68] нижняя граница для всего диапазона ρ_u, ρ_q: $c\sqrt{\rho_q} + (c-1)\sqrt{\rho_u} \geq \sqrt{2c-1}$ с заменой c на c^s для $1 \leq s \leq 2$. Эта нижняя граница ρ с точностью до слагаемых $o(1)$ соответствует верхней границе для разбиений, зависимых от данных (подразд. 2.6.3). Для $\rho_u = \rho_q$ и L_1 получаем $\rho = 1/(2c-1)$, а для L_2 имеем $\rho = 1/(2c^2-1)$, что соответствует нижним границам и для зависимого от данных LSH. Анализ нижней границы для $s > 2$ в настоящее время не проведен.

2.6.3. Адаптация к данным. Индексные структуры, которые адаптируются к данным базы, на практике зачастую превосходят ИС, конструируемые независимо от данных, хотя в большинстве и не дают гарантий худшего случая. Это справедливо также для поиска по сходству по представлениям, сформированным

обучением. Обзоры ИС и формирования представлений с обучением представлены в [26, 79–81].

В [82] приведен теоретический анализ уменьшения времени запроса по сравнению с худшим случаем и выбор параметров для независимого от данных LSH, который работает с хорошо рассредоточенными данными или с данными низкой внутренней размерности.

Адаптация ИС к данным базы, позволяющая улучшить характеристики ИС для данных худшего случая, рассмотрена в [83, 39, 68, 54]. Оказывается, для некоторых («псевдослучайных») конфигураций (расположений) объектов базы можно сконструировать независимые от данных ИС с меньшим ρ, чем для данных худшего случая. Объекты базы худшего случая разбивают на подмножества, которые сводят к таким конфигурациям.

В [83] улучшение ρ для LSH для L_2 получено при достаточно больших c, а в [39] результаты улучшены до теоретически оптимальных [84] $\rho = 1/(2c^2 - 1) + o(1)$ для всех c. В [68] получен (оптимальный для широкого класса ИС) компромисс между ρ_u, ρ_q: $c^2\sqrt{\rho_q} + (c^2 - 1)\sqrt{\rho_u} \geq \sqrt{2c^2 - 1}$. К сожалению, вследствие сложности редукции данных худшего случая к «псевдослучайным» конфигурациям методы [83, 39, 68] практически не реализованы. В [54] приведен анализ практической схемы (модификации LSH-forest), в которой характеристики для данных худшего случая превосходят независимый от данных LSH (хотя и уступают оптимальным границам [39]).

ЗАКЛЮЧЕНИЕ

Преимущество рассмотренных в данной статье ИС на основе локально-чувствительного хэширования и их модификаций заключается в теоретически обоснованном выборе параметров ИС, позволяющем гарантировать сублинейное относительно количества объектов в базе время выполнения некоторых типов запросов приближенного поиска по сходству для худшего случая. Во второй части обзора будет представлено обсуждение и сравнение этого подхода с ИС на основе: явного использования областей (древовидные и др.), графов соседства [3], преобразования исходных (вещественных векторных) представлений объектов, автоассоциативной памяти [85]. Отметим, что специализированный для сходства Жаккара между бинарными разреженными векторами очень большой размерности хорошо параллелизуемый алгоритм FLASH [86] (модифицированный вариант ИС LSH, использующий MinHash с одной перестановкой для получения большого количества хэш-значений [87], корзины ограниченного размера за счет сэмплирования, общие корзины для различных LSH-таблиц, ранжирование объектов-кандидатов по частоте их встречаемости без вычисления исходных расстояний/сходств до объекта-запроса) продемонстрировал преимущество над всеми другими ИС.

Автор благодарен Alex Andoni за разъяснения некоторых аспектов его исследований.

СПИСОК ЛИТЕРАТУРЫ

1. Rachkovskij D.A. Real-valued embeddings and sketches for fast distance and similarity estimation. *Cybernetics and Systems Analysis.* 2016. Vol. 52, N. 6. P. 967–988.
2. Rachkovskij D.A. Binary vectors for fast distance and similarity estimation. C*ybernetics and Systems Analysis*. 2017. Vol. 53, N 1. P. 138–156.
3. Rachkovskij D.A. Distance-based index structures for fast similarity search. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 4. P. 636–658.
4. Rachkovskij D.A. Index structures for fast similarity search for binary vectors. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 5. P. 799–820.
5. Manning C., Raghavan P., Schütze H. Introduction to information retrieval. New York: Cambridge University Press, 2008. 506 p.
6. Datta R., Joshi D., Li J., Wang J. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*. 2008. Vol. 40, N 2. P. 1–60.
7. Fouad M.M. Content-based search for image retrieval. *Int. J. Image, Graphics and Signal Processing*. 2013. Vol. 5, N 11. P. 46–52.

8. Khalifa F.A., Semary N.A., El-Sayed H.M., Hadhoud M.M. Local detectors and descriptors for object class recognition. *Int. J. of Intelligent Systems and Applications*. 2015. Vol. 7, N 10. P. 12–18.
9. Ziomek A., Oszust M. Evaluation of interest point detectors in presence of noise. *Int. J. Intelligent Systems and Applications*. 2016. Vol. 8, N 3. P. 26–33.
10. Fortune S. Voronoi diagrams and Delaunay triangulations. Chap. 27. *Handbook of Discrete and Computational Geometry*, 3rd edition. Boca Raton, USA: CRC Press. 2017. P. 705–721.
11. Meiser S. Point location in arrangements of hyperplanes. *Inform. and Comput*. 1993. Vol. 106, N 2. P. 286–303.
12. Andoni A., Indyk P. Nearest neighbors in high-dimensional spaces.Chap. 43. *Handbook of Discrete and Computational Geometry*, 3rd edition. Boca Raton, USA: CRC Press. 2017. P. 1133–1153.
13. Weber R., Schek H., Blott S. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. *Proc. VLDB'98*. 1998. P. 194–205.
14. Arya S., Mount D., Netanyahu N., Silverman R., Wu A. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*. 1998. Vol. 45, N 6. P. 891–923.
15. Andoni A., Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*. 2008. Vol. 51. N 1. P. 117–122.
16. Har-Peled S., Indyk P., Motwani R. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput.* 2012. Vol. 8. P. 321–350.
17. Powers D.M.W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *J. of Machine Learning Tech.* 2011. Vol. 2, N 1. P. 37–63.
18. Das R., Thepade S., Ghosh S. Content based image recognition by information fusion with multiview features. I. *J. Information Technology and Computer Science*. 2015. Vol. 7, N 10. P. 61–73.
19. Ramaswamy S., Rose K. Adaptive cluster distance bounding for high-dimensional indexing. *IEEE Trans. on KDE*. 2011. Vol. 23, N 6. P. 815–830.
20. Muja M., Lowe D.G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI*. 2014. Vol. 36, N 11. P. 2227–2240.
21. Shrivastava A., Li P. Asymmetric minwise hashing for indexing binary inner products and set containment. *Proc. WWW'15*. 2015. P. 981–991.
22. Charikar M. Similarity estimation techniques from rounding algorithms. *Proc. STOC'02*. 2002. P. 380–388.
23. Aumuller M., Christiani T., Pagh R., Silvestr F. Distance sensitive hashing. arXiv:1703.07867. 22 Mar 2017.
24. Andoni A., Datar M., Immorlica N., Indyk P., Mirrokni V.S. Locality-sensitive hashing using stable distributions. *Nearest Neighbor Methods for Learning and Vision: Theory and Practice*. Cambridge: MIT Press, 2006. P. 61–72.
25. Pham N. Hybrid LSH: Faster near neighbors reporting in high-dimensional space. *Proc. EDBT'17.* 2017. P. 454–457.
26. Wang J., Shen H.T., Song J., Ji J. Hashing for similarity search: A survey. arXiv:1408.2927. 13 Aug 2014.
27. Tang J., Tian Y. A systematic review on minwise hashing algorithms. *Annals of Data Science*. 2016. Vol. 3, N 4. P. 445–468.
28. Kulis B., Grauman K. Kernelized locality-sensitive hashing. *IEEE Trans. PAMI*. 2012. Vol. 34, N 6. P. 1092–1104.
29. Mu Y., Yan S. Non-metric locality sensitive hashing. *Proc. AAAI'10.* 2010. P. 539–544.
30. Andoni A., Indyk P. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Proc. FOCS'06.* 2006. P. 459-468.
31. Jegou H., Amsaleg L., Schmid C., Gros P. Query-adaptive locality sensitive hashing. *Proc. ICASSP'08.* 2008. P. 825–828.
32. Chierichetti F., Kumar R. Lsh-preserving functions and their applications. *J. ACM.* 2015. Vol. 62, N 5. P. 33:1–33:25.
33. Chierichetti F., Kumar R., Panconesi A., Terolli E. The distortion of locality sensitive hashing. *Proc. ITCS'17*. 2017. 23 p.
34. Sokolov A. Investigation of accelerated search for close text sequences with the help of vector representations. *Cybernetics and Systems Analysis.* 2008. Vol. 44, N 4. P. 493–506.
35. Andoni A., Krauthgamer R., Razenshteyn I.P. Sketching and embedding are equivalent for norms. *Proc. STOC'15*. 2015. P. 479–488.
36. Andoni A., Indyk P., Laarhoven T., Razenshteyn I., Schmidt L. Practical and optimal LSH for angular distance. *Proc. NIPS'15*. 2015. P. 1225-1233.
37. Terasawa K., Tanaka Y. Spherical lsh for approximate nearest neighbor search on unit hypersphere. *Proc. WADS'07*. 2007. P. 27–38.

38. Eshghi K., Rajaram S. Locality sensitive hash functions based on concomitant rank order statistics. *Proc. KDD'08*. 2008. P. 221–229.
39. Andoni A., Razenshteyn I. Optimal data-dependent hashing for approximate near neighbors. *Proc. STOC'15*. 2015. P. 793–801.
40. Laarhoven T. Hypercube LSH for approximate near neighbors. *Proc. MFCS'17*. 2017.
41. Kennedy C., Ward R. Fast cross-polytope locality-sensitive hashing. *Proc. ITCS'17*. 2017.
42. Panigrahy R. Entropy based nearest neighbor search in high dimensions. *Proc. SODA'06*. 2006. P. 1186–1195.
43. Lv Q., Josephson W., Wang Z., Charikar M., Li K. Multi-probe lsh: efficient indexing for high-dimensional similarity search. *Proc. VLDB'07*. 2007. P. 950–961.
44. Joly A., Buisson O. A posteriori multi-probe locality sensitive hashing. *Proc. MM'08*. 2008. P. 209–218.
45. Slaney M., Lifshits Y., He J. Optimal parameters for locality-sensitive hashing. *Proc. of the IEEE*. 2012. Vol. 100, N 9. P. 2604–2623.
46. Kapralov M. Smooth tradeoffs between insert and query complexity in nearest neighbor search. *Proc. PODS'15*. 2015. P. 329–342.
47. Ahle T.D., Aumuller M., Pagh R. Parameter-free locality sensitive hashing for spherical range reporting. *Proc. SODA'17*. 2017. P. 239–256.
48. Pacuk A., Sankowski P., Wegrzycki K., Wygocki P. Locality-sensitive hashing without false negatives for lp. *Proc. COCOON'16*. 2016. P. 105–118.
49. Wygocki P. On fast bounded locality sensitive hashing. arXiv:1704.05902. 19 Apr 2017.
50. Dong W., Wang Z., Josephson W., Charikar M., Li K. Modeling lsh for performance tuning. *Proc. CIKM'08*. 2008. P. 669–678.
51. Flajolet P., Fusy E., Gandouet O., Meunier F. Hyperloglog: The analysis of a near-optimal cardinality estimation algorithm. *Proc. AofA'07*. 2007. P. 127–146.
52. Chakrabarti A., Satuluri V., Srivathsan A., Parthasarathy S. A Bayesian perspective on locality sensitive hashing with extensions for kernel methods. *ACM TKDD*. 2015. Vol. 10, N 2. P. 19:1–19:32.
53. Bawa M., Condie T., Ganesan P. Lsh forest: self-tuning indexes for similarity search. *Proc. WWW'05*. 2005. P. 651–660.
54. Andoni A., Razenshteyn I., Shekel Nosatzki N. Lsh forest: Practical algorithms made theoretical. *Proc. SODA'17*. 2017. P. 67–78.
55. Tao Y., Yi K., Sheng C., Kalnis P. Efficient and accurate nearest neighbor and closest pair search in high dimensional space. *ACM TODS*. 2010. Vol. 35, N 3. P. 20:1–20:46.
56. Lawder J.K., King P.J.H. Querying multi-dimensional data indexed using the hilbert space filling curve. *ACM SIGMOD Record*. 2001. Vol. 30, N 1. P. 19–24.
57. Comer D. The ubiquitous B-tree. *ACM Comput. Surv*. 1979. Vol. 11. P. 121–138.
58. Liu Y., Cui J., Huang Z., Li H., Shen H.T. Sk-lsh: An efficient index structure for approximate nearest neighbor search. *Proc. VLDB Endowment.* 2014. Vol. 7, N 9. P. 745-756.
59. Chen J., He C., Hu G., Shao J. SELSH: a hashing scheme for approximate similarity search with early stop condition. *Proc. MMM'16.* 2016. Vol. 2. P. 104–115.
60. Hao F., Daugman J., Zielinski P. A fast search algorithm for a large fuzzy database. *IEEE Trans. Information Forensics and Security*. 2008. Vol. 3, N 2. P. 203–212.
61. Ling K., Wu G. Frequency based locality sensitive hashing. *Proc. ICMT'11*. 2011. P. 4929–4932.
62. Gan J., Feng J., Fang Q., Ng W. Locality-sensitive hashing scheme based on dynamic collision counting. *Proc. SIGMOD'12*. 2012. P. 541–552.
63. Zheng Y., Guo Q., Tung A.K.H., Wu S. LazyLSH: Approximate nearest neighbor search for multiple distance functions with a single index. *Proc. SIGMOD'16*. 2016. P. 2023–2037.
64. Huang Q., Feng J., Zhang Y., Fang Q., Ng W. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proc. VLDB Endowment*. 2015. Vol 9, N 1. P. 1–12.
65. Zhang X., Wang M., Cui J. Efficient indexing of binary LSH for high dimensional nearest neighbor. *Neurocomputing*. 2016. Vol. 213. P. 24–33.
66. Norouzi M., Punjani A., Fleet D.J. Fast exact search in Hamming space with multi-index hashing. *IEEE Trans. PAMI*. 2014. Vol. 36, N 6. P. 1107–1119.
67. Gao J., Jagadish, H.V., Ooi B.C., Wang S. Selective hashing: Closing the gap between radius search and k-NN search. *Proc. SIGKDD'15*. 2015. P. 349–358.
68. Andoni A., Laarhoven T., Razenshteyn I., Waingarten E. Optimal hashing-based time-space trade-offs for approximate near neighbors. *Proc. SODA'17*. 2017. P. 47–66.
69. Becker A., Ducas L., Gama N., Laarhoven T. New directions in nearest neighbor searching with applications to lattice sieving. *Proc. SODA'16*. 2016. P. 10–24.
70. Christiani T. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. *Proc. SODA'17*. 2017. P. 31–46.

71. Rachkovskij D.A., Misuno I.S., Slipchenko S.V. Randomized projective methods for construction of binary sparse vector representations. *Cybernetics and Systems Analysis*. 2012. Vol. 48, N 1. P. 146–156.
72. Rachkovskij D.A. Formation of similarity-reflecting binary vectors with random binary projections. *Cybernetics and Systems Analysis.* 2015. Vol. 51, N 2. P. 313–323.
73. Donaldson R., Gupta A., Plan Y., Reimer T. Random mappings designed for commercial search engines. arXiv:1507.05929. 21 Jul 2015.
74. Ferdowsi S., Voloshynovskiy S., Kostadinov D., Holotyak T. Fast content identification in highdimensional feature spaces using sparse ternary codes. *Proc. WIFS'16*. 2016. P. 1–6.
75. Valiant G. Finding correlations in subquadratic time, with applications to learning parities and the closest pair problem. *J. ACM.* 2015. Vol. 62, N 2. P. 13:1–13:45.
76. Nguyen H.L. Algorithms for high dimensional data. PhD thesis. Princeton University, 2014. URL: http://arks.princeton.edu/ark:/88435/dsp01b8515q61f.
77. Rahimi A., Recht B. Random features for large-scale kernel machines. *Proc. NIPS'07.* 2007. P. 1177–1184.
78. O'Donnell R., Wu Y., Zhou Y. Optimal lower bounds for locality sensitive hashing (except when *q* is tiny). *ACM TOCS*. 2014. Vol. 6, N 1. P. 5.1–5.13.
79. Wang J., Liu W., Kumar S., Chang S.-F. Learning to hash for indexing big data: A survey. *Proc. of the IEEE.* 2016. Vol. 104, N 1. P. 34–57.
80. Wang J., Zhang T., Song J., Sebe N., Shen H.T. A survey on learning to hash. *IEEE Trans. PAMI.* doi:10.1109/TPAMI.2017.2699960.
81. Gao L., Song J., Liu X., Shao J., Liu J., Shao J. Learning in high-dimensional multimedia data: the state of the art. *Multimedia Systems.* 2017. Vol. 23, N 3. P. 303–313.
82. Mou W., Wang L. A refined analysis of lsh for well-dispersed data points. *Proc. ANALCO'17*. 2017. P. 174–182.
83. Andoni A., Indyk P., Nguyen H.L., Razenshteyn I. Beyond locality-sensitive hashing. *Proc. SODA'14.* 2014. P. 1018–1028.
84. Andoni A., Razenshteyn I. Tight lower bounds for data-dependent locality-sensitive hashing. *Proc. SoCG'16*. 2016. P. 9:1–9:11.
85. Gritsenko V.I., Rachkovskij D.A., Frolov A.A., Gayler R., Kleyko D., Osipov E. Neural distributed autoassociative memories: A survey. *Cybernetics and Computer Engineering*. 2017. N 2 (188). P. 5–35.
86. Wang Y., Shrivastava A., Ryu J. FLASH: randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. arXiv:1709.01190. 4 Sep 2017.
87. Shrivastava A. Optimal densification for fast and accurate minwise hashing. *Proc. ICML'17*. 2017. P. 3154–3163.

Надійшла до редакції 23.08.2017

Д.А. Рачковський

ІНДЕКСНІ СТРУКТУРИ ДЛЯ ШВИДКОГО ПОШУКУ ЗА СХОЖІСТЮ ДІЙСНИХ ВЕКТОРІВ. I

Анотація. Наведено огляд індексних структур для швидкого пошуку за схожістю об'єктів, що представлені дійсними векторами. Розглянуто індексні структури на основі локально-чутливого хешування та їхні модифікації. Викладено ідеї конкретних алгоритмів (відомих та нещодавно запропонованих). Обговорено їхній взаємозв'язок і деякі теоретичні аспекти.

Ключові слова: пошук за схожістю, найближчий сусід, ближній сусід, індексні структури, локально-чутливе хешування, локально-чутлива фільтрація.

D.A. Rachkovskij

INDEX STRUCTURES FOR FAST SIMILARITY SEARCH OF REAL-VALUED VECTORS. I

Abstract. In this survey paper, we consider index structures for fast similarity search of objects represented by real-valued vectors. Index structures based on locality-sensitive hashing and their modifications are considered. The ideas of specific algorithms, including the recently proposed ones, are outlined. Their interrelations and some theoretical aspects are discussed.

Keywords: similarity search, nearest neighbor, near neighbor, index structures, locality-sensitive hashing, locality-sensitive filtering.

Рачковский Дмитрий Андреевич,
доктор техн. наук, ведущий научный сотрудник Международного научно-учебного центра информационных технологий и систем НАН Украины и МОН Украины, Киев, e-mail: dar@infrm.kiev.ua.