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 БИНАРНЫХ ВЕКТОРОВ

Аннотация. Дан обзор индексных структур для быстрого поиска по сходству объектов, представленных бинарными векторами (с компонентами 0 или 1). Рассмотрены структуры как для точного, так и для приближенного поиска по расстоянию Хэмминга и другим мерам сходства. Представлены, главным образом, индексные структуры на основе хэш-таблиц, сохраняющего сходство хэширования, а также древовидных структур, графов соседства и нейросетевой распределенной автоассоциативной памяти. Изложены идеи известных и предложенных в последнее время алгоритмов.

Ключевые слова: поиск по сходству, расстояние Хэмминга, ближайший сосед, ближний сосед, индексные структуры, мультииндексное хэширование, локально-чувствительное хэширование, древовидные структуры, граф соседства, нейросетевая автоассоциативная память.

ВВЕДЕНИЕ

Поиск сходных с объектом-запросом объектов базы по некоторой мере расстояния/сходства называют поиском по сходству. Найденные объекты (например, тексты или изображения [1–3]) могут являться конечным результатом поиска или содержать дополнительную ассоциированную информацию о сходном с ними входном объекте-запросе (см. ссылки в обзоре [4]).

В настоящем обзоре рассмотрено выполнение поиска по представлениям объектов бинарными векторами с компонентами из $\{0, 1\}$. Плотные бинарные векторы (с приблизительно равным количеством единичных и нулевых компонентов) можно компактно хранить (требуется всего лишь один бит памяти на компонент) и быстро обрабатывать, особенно выполнением покомпонентных операций, что используется при вычислении многих мер расстояния/сходства бинарных векторов (например, расстояния Хэмминга, подразд. 1.1). Разреженные бинарные векторы (с малой долей единичных компонентов) также эффективно хранятся и обрабатываются (подразд. 1.3).

Предложены представления объектов различного типа в виде бинарных векторов, сходство которых отражает сходство объектов. Такие бинарные векторы можно формировать непосредственно из исходных представлений объектов. Например, для представления изображений используют плотные бинарные векторы — локальные дескрипторы размерностью в сотни битов (BRIEF, ORB, BRISK, FREAK [5–7] или векторы из слоев глубоких нейронных сетей [8–10]), применяют бинаризацию дискретного косинусного преобразования DCT [11] и др. Такие бинарные векторы также объединяют в векторы размерностью в десятки тысяч битов. Разреженные бинарные векторы размерностью в сотни тысяч битов и более используют, например, для представления слов [12], изображений лиц [13], данных о покупках [14] и др. Бинарные векторы, отражающие сходства объектов, формируют из векторных или других представлений без обучения (например, см. обзоры в [15–17]) и с обучением [18, 19].

Вследствие эффективности оперирования бинарными векторами для не слишком больших баз может оказаться приемлемым даже линейный поиск по сходству, т.е. вычислением значений расстояний/сходств между вектором-запросом и всеми векторами базы. Скорость линейного поиска повышается для векторов малой размерности.

В обзоре [17] (см. также [20]) рассмотрены методы снижения размерности бинарных векторов без учителя (например, сэмплирование, случайное проецирование, использование LSH и др.) для получения векторов, по которым можно быстрее оценить сходство исходных. Так как оценка исходного сходства по векторам сниженной размерности (в большинстве случаев) неточная, результаты могут отличаться от результатов линейного поиска по исходной мере сходства. Однако во многих случаях отличия от результатов точного поиска приемлемы, если время поиска уменьшается.

Для очень больших баз линейный поиск недопустимо медленный. Возможность поиска по сходству с сублинейной относительно количества объектов в базе сложностью предоставляют специальные структуры данных — индексные структуры (ИС), обеспечивающие ускорение поиска. Хотя для бинарных векторов можно использовать ИС, работающие с информацией только о расстояниях [4], или ИС для вещественных векторов [21–23], ввиду специфики бинарных векторов специализированные для них ИС часто более эффективны. В настоящем обзоре рассмотрены такие ИС для точного и приближенного поиска по сходству. Структура обзора приведена в подразд. 1.6.

1. ОСНОВНЫЕ ПОНЯТИЯ

1.1. Меры расстояния и сходства бинарных векторов. 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 D — размерность векторов $\mathbf{a}$, $\mathbf{b}$, $\{.\}$ — индикаторная функция. Для бинарных векторов с компонентами из $\{0,1\}$ выполняется $\mathrm{dist}_{\mathrm{Ham}}(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}||_1 = ||\mathbf{a} - \mathbf{b}||_s^s$, где $||\mathbf{a} - \mathbf{b}||_s = \left(\sum_{i=1}^{D} |a_i - b_i|^s \right)^{1/s}$ — расстояния Минковского различного порядка s (например, для $s = 2$ получаем расстояние Евклида). Эти расстояния при $s \geq 1$ являются метриками, т.е. подчиняются метрическим аксиомам, таким как неравенство треугольника и др.

Для сходств бóльшие значения соответствуют более сходным объектам. Одной из мер сходства векторов является скалярное произведение $\mathrm{sim}_{\mathrm{dot}}(\mathbf{a}, \mathbf{b}) \equiv \equiv \langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=1}^{D} a_i b_i$. Косинус угла между векторами определяется как $\cos(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / (||\mathbf{a}||_2 ||\mathbf{b}||_2) \equiv \mathrm{sim}_{\cos}$, где $||\mathbf{z}||_2$ — евклидова норма $\mathbf{z}$. Угол между $\mathbf{a}$, $\mathbf{b}$ определяет расстояние $\mathrm{dist}_{\mathrm{ang}} = \arccos(\mathbf{a}, \mathbf{b})$.

Для бинарных векторов $\cos(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / (|\mathbf{a}||\mathbf{b}|)^{1/2}$, где $|\mathbf{z}|$ — количество ненулевых компонентов вектора, т.е. хэммингова норма. Выполняется $\mathrm{dist}_{\mathrm{Ham}}(\mathbf{a}, \mathbf{b}) = |\mathbf{a}| + |\mathbf{b}| - 2\langle \mathbf{a}, \mathbf{b} \rangle$. Сходство Жаккара определяется как $\mathrm{sim}_{\mathrm{Jac}}(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / (|\mathbf{a}| + |\mathbf{b}| - \langle \mathbf{a}, \mathbf{b} \rangle)$, а сходство Брауна–Бланке — как $\mathrm{sim}_{\mathrm{BB}}(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle / \max(|\mathbf{a}|, |\mathbf{b}|)$. Эти меры часто используют для разреженных векторов. Если упорядочить (ранжировать) базу бинарных векторов с фиксированной хэмминговой нормой по возрастанию $\mathrm{dist}_{\mathrm{Ham}}$ или $\mathrm{dist}_{\mathrm{ang}}$ относительно некоторого вектора, такой же порядок получим упорядочиванием по убыванию $\mathrm{sim}_{\mathrm{dot}}$, $\mathrm{sim}_{\cos}$, $\mathrm{sim}_{\mathrm{Jac}}$, $\mathrm{sim}_{\mathrm{BB}}$.

Отметим, что бинарные векторы можно рассматривать как характеристические векторы соответствующих невзвешенных множеств (1 соответствует наличию элемента множества, 0 — отсутствию) и вычислять по векторам меры сходства/различия таких множеств.

1.2. Типы запросов точного поиска по сходству. Объекты базы, которые являются ответом на конкретный запрос поиска по сходству, определяются ти-

пом запроса и его параметрами (объектом-запросом, мерой расстояния/сходства и др.). Объекты-запросы могут не принадлежать базе. В настоящем обзоре объекты являются бинарными 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 быть 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.

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.

Запрос kNN можно выполнить последовательностью запросов rNN или rNN1 с увеличением r до тех пор, пока не будет достигнуто количество k возвращенных объектов. Можно также вначале принять $r = D/2$ и выполнить запрос rNN1, а затем $r = D/4$ или $r = 3D/4$ (в зависимости от наличия или отсутствия объектов базы в результате запроса) и так далее, тогда после $\log D$ запросов получают NN. Отметим, что многие объекты базы могут иметь одинаковое расстояние до объекта-запроса, особенно если объекты — бинарные векторы.

Запросы rNN и kNN с приведенными определениями являются запросами точного поиска по сходству. Определения типов запросов в терминах не расстояния, а сходства аналогичны. Существуют и другие типы запросов точного поиска по сходству, однако в настоящем обзоре они не рассматриваются.

1.3. Линейный поиск. Время выполнения запросов поиска по сходству линейным поиском в базе из N объектов-векторов для мер расстояния/сходства с временем вычисления $O(D)$ (например, меры для плотных векторов, см. подразд. 1.1) составляет $O(ND)$.

Для бинарных векторов, компонент которых представлен одним битом (например, 64-разрядного слова), вычисление $\mathrm{dist}_{\mathrm{Ham}}$ выполняется быстрыми побитовыми операциями XOR (одновременно над 64 компонентами двух векторов) и подсчетом количества единичных битов в каждом получившемся слове. Для подсчета применяют либо быстрые специализированные команды процессора, либо таблицы (например, двоичному числу 11111110 соответствует ячейка таблицы, в которой хранится число 7). Используют и иные возможности современных процессоров [24]. Аналогично вычисляются другие меры расстояния/сходства бинарных векторов из подразд. 1.1. Время вычислений по сравнению с вещественными векторами может сократиться в десятки раз.

Для разреженных бинарных векторов используют представление в виде номеров ненулевых компонентов. Время вычисления меры расстояния/сходства пропорционально суммарному количеству ненулевых компонентов в двух векторах. Применяют также специализированные алгоритмы (подразд. 1.4.3) и средства ([25] и подразд. 5.1 в [17]).

1.4. Базовые индексные структуры для ускорения точного поиска по сходству.

1.4.1. Точный поиск в шаре Хэмминга модификацией вектора запроса. Особенностью бинарных векторов является возможность их непосредственного использования в качестве адресов ячеек (корзин) хэш-таблиц. Простой алгоритм выполнения запроса rNN по $\mathrm{dist}_{\mathrm{Ham}}$ (например, [26]) конструирует таблицу, в которой каждый бинарный вектор базы помещается в ячейку с номером, соответствующим его бинарному коду (так, вектору 1111 соответствует ячейка с номером 15). Наличие в базе вектора, совпадающего с запросом, определяется проверкой заполненности соответствующей ячейки, т.е. за постоянное время $O(1)$. Для запроса rNN выполняют систематическую модификацию вектора-запроса,

изменяя (0 на 1 и 1 на 0) значения r его компонентов так, чтобы получить все векторы, лежащие внутри шара Хэмминга с радиусом r и центром в векторе-запросе (т.е. лежащие на сферах Хэмминга с радиусами от 0 до r). Модификации используют в качестве запросов для поиска по совпадению. Например, для вектора 1111 модификациями при $r=1$ являются $\{0111, 1011, 1101, 1110\}$.

Преимущества такой ИС — простота конструирования, возможность выполнения запросов rNN с изменяемым радиусом и быстрый поиск для малых r, а один из недостатков — затраты памяти 2^D и ее нерациональное использование при $N << 2^D$ (большинство ячеек пустые). Устранить этот недостаток можно сортировкой векторов базы в лексикографическом порядке (0 предшествует 1, что эквивалентно сортировке по величине соответствующих целых чисел) и запоминанием в N ячейках. Совпадающий с запросом вектор находят поиском дихотомией, время которого в худшем случае $O(\log N)$ [4]. Время $O(D)$ получают при поиске совпадения покомпонентными сравнениями.

Другой вариант усовершенствования такой ИС состоит в использовании универсального хэширования [27] и создании хэшей размерностью $\log N$. Это дает среднее время доступа $O(1)$. Идеальное хэширование [28] (хэши не имеют коллизий для различных объектов базы) исходных векторов дает время доступа $O(1)$ в худшем случае за счет затрат памяти $O(N)$ на хранение идеальной хэш-функции для конкретной базы. На практике идеальное хэширование приводит к значительным накладным расходам [29].

Основным недостатком рассматриваемой ИС является количество $\sum_{i=0}^{r} C_D^i$ модификаций вектора запроса, что превышает $(D/r)^r$ [30]. Это число может превысить размер базы N, что делает такую ИС непригодной для практического применения при больших D и r. Использование запросов поиска не по совпадению, а с единичным радиусом (подразд. 2.1) позволяет выполнять запрос в r раз быстрее при затратах памяти $O(ND)$, что не решает проблемы.

1.4.2. Индексные структуры с экспоненциальными затратами памяти. В ИС для выполнения запроса NN [31] для всех возможных 2^D векторов-запросов при конструировании запоминают точного NN. Для выполнения запроса rNN в ИС можно запомнить все векторы базы, находящиеся на расстоянии до r от исходных. Это увеличивает затраты памяти в $\sum_{i=0}^{r} C_D^i$ раз. Такие ИС непрактичны вследствие экспоненциальных от D затрат памяти.

1.4.3. Обратный индекс. Классическое обратное индексирование применяют для быстрого вычисления $\mathrm{sim}_{\mathrm{dot}}$ разреженных векторов базы и запроса [32, 33, 25, 34], в том числе и для бинарных или тернарных векторов [33, 25, 34]. Сходное решение используют в поисковых системах Интернета [32, 25].

Для каждого компонента вектора запоминают список объектов базы, для которых этот компонент имеет ненулевое значение, и значения компонента в этих объектах. Для получения $\mathrm{sim}_{\mathrm{dot}}$ вектора-запроса с векторами базы вычисляют произведения ненулевых компонентов вектора-запроса и запомненных значений в соответствующих им списках объектов и суммируют произведения, соответствующие каждому посещенному объекту базы. Если количество посещенных объектов меньше N, получим сублинейное время (лишь эти объекты подлежат ранжированию). Кроме того, сокращения времени вычислений достигают за счет обработки только ненулевых компонентов векторов. Поэтому обратный индекс особенно эффективен для векторов с малым количеством ненулевых компонентов при малом количестве посещенных объектов. Аналогично вычисляют другие меры, использующие $\mathrm{sim}_{\mathrm{dot}}$ (см. подразд. 1.1).

1.5. Переход от точного к приближенному поиску. Для алгоритмов, рассмотренных в подразд. 1.4.1 и 1.4.2, затраты памяти или времени растут экспоненциально от D или r. Более того, анализ всех известных алгоритмов точного выполнения запроса NN с сублинейным от N временем показывает, что затраты памяти

или времени растут экспоненциально от D [31]. Для алгоритмов с затратами памяти, близкими к линейным, лучшее известное время $\min(2^{O(D)}, DN)$ [31]. Поэтому для баз с достижимым N точный поиск вырождается в линейный с ростом D, даже при не слишком больших D, что подтверждается и на практике [35].

Согласно предположению «проклятия размерности» [31] такая зависимость неизбежна при точном поиске по сходству для данных худшего случая (для объекта-запроса и объектов базы, которые дают наибольшее время выполнения запроса). Для некоторых алгоритмов экспоненциальная зависимость проявляется в терминах «внутренней» размерности (см. ссылки в [4]), которая может быть меньше «внешней» размерности D [36]. Отметим исключение для специального случая $D = O(\log N)$ и обработки N запросов (худшего случая): для каждого запроса получают (рандомизированное, с высокой вероятностью) сублинейное время нахождения точного NN для ряда расстояний/сходств [37].

Преодолеть (точнее, обойти) проклятие размерности удается переходом от точного к более быстрому, но приближенному поиску, который допускает отличие результатов от точного поиска. 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 (точных или приближенных) типа Monte Carlo (т.е. с ограниченным временем работы, но с некоторой вероятностью некорректного ответа) для поиска по сходству допускаются false negatives (алгоритм может не возвратить объекты, которые являются ответом на запрос).

Приближенные рандомизированные алгоритмы (т.е. допускающие оба типа отличий: приближенность и вероятность ошибки) обычно имеют лучшие характеристики, чем алгоритмы, допускающие один тип отличий: либо приближение, либо вероятность ошибки [31]. В разд. 3 рассмотрены типы запросов для приближенного поиска со специфицированными приближением и вероятностью ошибки, а также алгоритмы Monte Carlo их выполнения с гарантиями сублинейного времени для данных худшего случая, включая практически реализуемые.

С помощью рандомизированных алгоритмов типа Las Vegas запросы поиска по сходству (точного или приближенного) выполняют без false negatives, однако сублинейное время обеспечивается лишь для среднего случая (векторы базы и запросов выбраны случайно и независимо из их распределений). Для данных худшего случая время запроса может становиться равным (и большим) времени линейного поиска. Алгоритмы Las Vegas для точного поиска приведены в разд. 2 и для приближенного поиска — в подразд. 3.6.

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 обычно неприемлемы).

Часто используют меры качества выполнения конкретного запроса, известные как [38] полнота (recall), равная $n1 / n2$, и точность (precision), равная $n1 / n3$, где $n1$ — количество возвращенных объектов, совпавших с релевантными, $n2$ —

количество релевантных запросу объектов в базе, $n3$ — количество возвращенных объектов базы. Для запроса kNN $n2 = n3 = k$, поэтому точность равна полноте. «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 точного результата.

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 точный NN [39].

1.6. Типы рассмотренных в обзоре индексных структур. Многие алгоритмы ускорения поиска по сходству, приведенные в обзоре, применяют двухэтапную стратегию фильтрации и уточнения (filter-and-refine, F&R). На первом этапе осуществляется 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. Отметим, что для запроса rNN второй этап устраняет false positives.

Большинство ИС, рассмотренных в разд. 2 и 3, используют множество хэш-таблиц. В разд. 2 приведены ИС для точного поиска по $\mathrm{dist}_{\mathrm{Ham}}$ с гарантиями сублинейного времени для худшего случая при очень малых радиусах, а в основном с гарантиями для среднего случая. В разд. 3 рассмотрены ИС на основе LSH-хэширования для $\mathrm{dist}_{\mathrm{Ham}}$ и других мер расстояния/сходства, для которых существуют LSH-функции. Эти ИС обеспечивают гарантии худшего случая для некоторых типов запросов приближенного вероятностного поиска. В разд. 4 и 5 представлены соответственно ИС на основе деревьев и графов соседства в основном для приближенного поиска без строгих гарантий. В разд. 6. рассмотрены ИС на основе распределенной нейросетевой ассоциативной памяти, главным образом, с асимптотическими гарантиями (при размерности, стремящейся к бесконечности) для случайных разреженных векторов.

2. ИНДЕКСНЫЕ СТРУКТУРЫ ДЛЯ ТОЧНОГО ПОИСКА ПО РАССТОЯНИЮ ХЭММИНГА

Для выполнения точного запроса rNN по $\mathrm{dist}_{\mathrm{Ham}}$ применяют ИС в основном с использованием хэш-таблиц. Далее приведены ИС с гарантиями худшего случая для единичного и других малых фиксированных радиусов запроса (подразд. 2.1), а также с гарантиями среднего случая для фиксированного и изменяемого радиуса (подразд. 2.2).

2.1. Индексные структуры для точного поиска с гарантиями худшего случая.

2.1.1. Единичный радиус. Выполнение запроса rNN по $\mathrm{dist}_{\mathrm{Ham}}$ для $r = 1$ модификацией запроса (см. подразд. 1.4.1) требует выполнения $D+1$ запросов на совпадение (в терминах D-компонентных векторов). Разработаны усовершенствования этого алгоритма.

В [40] решают следующую задачу: определить наличие в базе вектора на расстоянии $r \le 1$ от вектора-запроса, т.е. ближнего соседа rNN1. Выявление совпадения ($r = 0$) и обработка «плохих» векторов-запросов выполняются с использованием хэширования векторов полной размерности. Для $r = 1$, если разбить векторы на две части, то в одной вектор-запрос совпадет с векторами-кандидатами базы, среди которых находятся векторы, являющиеся ответом на запрос, а в другой — будет отличаться в одном компоненте. Для векторов-кандидатов с совпавшей частью несовпавшую рекурсивно разбивают на две части, пока одна из них не станет компонентом или количество кандидатов не уменьшится до одного. Для выявления совпадающих частей векторов убывающей размерности и соответствующих подмножеств кандидатов при выполнении запроса для векторов базы предварительно конструируют ИС на основе хэш-таблиц (для частей вдвое убывающей размерности и множеств векторов базы убывающей мощности). Для модели bit probe, в которой сложность измеряется только количеством битовых доступов к памяти ИС

(доступ возвращает один бит), затраты памяти составляют $O(DN \log D)$, время решения задачи $O(D \log \log N)$. Модификация этой ИС выполняет и запрос rNN.

В [26] применяют модель RAM (random access machine), в которой время запроса включает как операции доступа, так и требующиеся арифметические операции. Использование префиксных деревьев с лексикографически упорядоченными векторами позволило при затратах памяти $O(DN)$ достичь времени $O(D)$ получения решения о существовании rNN1 в базе и возвращения всех таких векторов (количество которых не превышает $D+1$).

В работе [41] для всех D однокомпонентных модификаций каждого вектора базы конструируют идеальную функцию хэширования [28]. В каждой из $O(DN)$ ячеек таблицы запоминают номер модифицированного компонента, что требует $\log D$ битов, общие затраты памяти $O(DN \log D)$ битов. При выполнении запроса вычисляют хэш вектора-запроса, извлекают из таблицы номер искаженного компонента, инвертируют его в векторе-запросе и проверяют совпадение с вектором базы. Если совпадение существует, то получают ответ, в противном случае $\text{dist}_{\text{Ham}} > 1$. Для модели cell probe (аналог bit probe с доступом к машинным словам размерностью D) затраты памяти составляют $O(N \log D)$ при времени $O(1)$ выполнения запроса rNN1.

2.1.2. Индексные структуры с фиксированным радиусом запроса $r > 1$. Задача создания ИС для выполнения точных запросов rNN при $r > 1$ с гарантиями для данных худшего случая очень сложна. Простая ИС с модификацией запроса (см. подразд. 1.4.1) позволяет работать и с изменяемым r, но ускорение по сравнению с линейным поиском на практике возможно только при малых D и r. В [42] для запроса rNN с фиксированным r предложена ИС k-errata trie, которая включает суффиксное дерево и структуру наибольшего общего префикса. Затраты памяти $O(DN + N(c_1 \log N)^r / r!)$, время запроса $O(D + ((c_2 \log N)^r / r!) \log\log(DN) + \text{occ})$, где c_1, $c_2 > 0$ — константы, occ — количество векторов базы, являющихся ответом на запрос. Для линейных затрат памяти получено время $O(D + \text{occ} + (\log N)^{r(r+1)} \text{loglog} N)$ [43] и $O(D^{r-1} \log N \log\log N + \text{occ})$ [44]. Такие ИС [42–44] нельзя использовать на практике ввиду малого r, для которого достигается сублинейное время запроса и умеренные затраты памяти.

2.2. Многотабличные индексные структуры для точного поиска с гарантиями среднего случая. Рассмотрим более практичные решения для точного сублинейного поиска по dist_{Ham} с гарантиями среднего случая (Las Vegas).

Приведенные в подразд. 2.2 ИС относятся к классу мультииндексного хэширования. Идея состоит в разбиении векторов на несколько непересекающихся частей; для каждой части вектора строится частичная ИС (обычно табличная), содержащая все векторы базы и позволяющая проводить эффективный поиск по части вектора. При выполнении запроса из каждой ИС извлекают векторы базы, части которых совпадают или находятся в пределах некоторого dist_{Ham} от соответствующей части вектора-запроса. Эти кандидаты уточняются на втором этапе стратегии F&R. В подразд. 2.2.1 рассмотрены ИС поиска с фиксированным радиусом запроса, а в подразд 2.2.2 — с изменяемым. Для ИС с фиксированным r необходима отдельная ИС для каждого r.

2.2.1. Индексные структуры с фиксированным радиусом запроса. В [45] векторы разбивают на $M = r+1$ частей. Такое разбиение гарантирует совпадение с частью вектора-запроса по крайней мере одной части искомых векторов базы (см. также [46]). При равномерном распределении данных и размерности части $d = D / M$ среднее количество векторов базы в корзине $N / 2^d = N / 2^{D/M} = N / 2^{D/(r+1)}$, что очень велико для больших r. Для уменьшения количества кандидатов в [45] предлагают разбивать вектор на большее количество $M > r+1$ частей, но объединять эти части. Тогда не менее $M - r$ малых (до объединения) частей векторов, соответствующих запросу rNN, совпадут с соответствующими частями вектора-запроса. Количество различных комбинаций $M - r$ малых частей без учета порядка их следования составляет $L = C_M^{M-r} = C_M^r$.

Если построить L соответствующих частичных ИС, извлеченные из них (по объединенным частям) кандидаты будут гарантированно включать все векторы базы, являющиеся ответом на запрос rNN. Среднее количество кандидатов в частичной ИС уменьшается до $N/2^{d(M-r)}$, однако количество ИС возрастает с M до L, увеличивая затраты памяти. Компромисс исследован в [47].

В ИС MH [47] $M-r$ малых частей располагают в начале вектора, остальные части — в естественном порядке. В каждой из L частичных ИС векторы базы сортируют в лексикографическом порядке, чтобы быстрым поиском дихотомией находить совпадающие с запросом объединенные части. Недостатки этой ИС — большие затраты памяти $O(LND)$, малые значения r (на практике $r=2,\dots,7$ при $D=64$).

Если не требовать полного совпадения частей векторов, количество частичных ИС можно уменьшить: при $M(1)=\lfloor r/2\rfloor+1$ не менее чем для одной части векторов $\mathrm{dist}_{\mathrm{Ham}}\le 1$ [48], а при $M(r_p)=\lfloor r/(r_p+1)\rfloor+1$ не менее чем для одной части векторов $\mathrm{dist}_{\mathrm{Ham}}\le r_p$ [49]. Кандидатов в частичной ИС в пределах радиуса r_p находят модификацией части вектора-запроса и поиском совпадений. Количество запросов увеличивается с ростом r_p, например, с $r+1$ для $r_p=0$ до $M+M(D/M)=\lfloor r/2\rfloor+1+D$ для $r_p=1$. Однако уменьшение M с ростом r_p приводит к уменьшению количества частичных ИС (и затрат памяти), количества кандидатов в каждой корзине, а также общего количества кандидатов во всех извлеченных корзинах [49].

В ИС HEngine [48] (вариант для фиксированного r) рассматривают разбиение на $M\ge\lfloor r/2\rfloor+1$ частей, при этом $\mathrm{dist}_{\mathrm{Ham}}\le 1$ для не менее $M-\lfloor r/2\rfloor$ частей. Аналогично [47] существует $L=C_M^{M-\lfloor r/2\rfloor}$ способов расположить $M-\lfloor r/2\rfloor$ различных частей первыми. Формируют L таблиц, в которых векторы упорядочивают лексикографически. В отличие от [47] в L таблицах находят не только векторы базы с полным совпадением $M-\lfloor r/2\rfloor$ частей, но и с $\mathrm{dist}_{\mathrm{Ham}}=1$. В HEngine количество таблиц уменьшено в несколько раз по сравнению с [47] при некотором уменьшении времени запроса для $r\le 10$.

В ИС HmSearch [50] используют разбиение на M частей, в частичной ИС для каждой части применяют поиск с $\mathrm{dist}_{\mathrm{Ham}}\le 1$. Дополнительные условия фильтрации позволяют увеличить практически достижимые D и r. Для более равномерного распределения векторов по корзинам (подразд. 2.2.2) используют жадное формирование разбиений для минимизации максимальной частоты встречаемости любого подвектора базы.

Для уменьшения количества кандидатов HEngine в [49] применяют разбиения объектов на подмножества на основе расстояний до опорных объектов и неравенство треугольника (такие подходы приведены в [4]). Кроме того, за счет использования компактных структур данных и других оптимизаций гарантируется уменьшение памяти от 18 до 40 % при сохранении времени запроса и даже ускорение поиска при $r=\{2,3,4\}$. Общее время запроса сокращается в три-четыре раза.

2.2.2. Индексные структуры с переменным радиусом запроса. В ИС HEngine [48] (вариант для изменяемого r) при построении векторы базы рекурсивно делят на две части, пока их размерности не достигнут заданного порогового значения. Векторы базы сортируют в лексикографическом порядке в таблице для каждой части. При поиске учитывают, что для векторов с $\mathrm{dist}_{\mathrm{Ham}}\le r$ при разбиении на две части $\mathrm{dist}_{\mathrm{Ham}}\le\lfloor r/2\rfloor$ для одной или другой части. При выполнении запроса, если r превышает порог, вектор-запрос делят на две части, вдвое уменьшают r и рекурсивно решают получившиеся задачи. Одна из частей вектора базы, который является ответом на запрос, при таком рекурсивном разбиении должна совпасть с соответствующей частью вектора-запроса. Если достигнуто пороговое значение размерности частей вектора, выполняется линейный поиск

по текущим таблицам. Относительно линейного поиска достигается ускорение выполнения запроса до 46 раз ($D = 64$, $r = 2, \ldots, 7$) при увеличении затрат памяти в 1.7 раз.

В ИС Multi-index hashing (MIH) [51] строят таблицу для каждой из M частей, при этом для запроса rNN не менее чем в одной части $\text{dist}_{\text{Ham}} \le r_p = \lfloor r / M \rfloor$. Поэтому при выполнении запроса в каждой из M таблиц модификацией соответствующей части вектора-запроса находят в качестве кандидатов векторы базы с расстоянием $\text{dist}_{\text{Ham}} \le \lfloor r / M \rfloor$ от соответствующей части вектора-запроса. Используют $\log N$ компонентов в части вектора. Показана работа ИС при D до 256; в отличие от рассмотренных ранее алгоритмов r составляет значительную долю D. Реализовано также выполнение запроса kNN увеличением радиуса запроса rNN.

При случайном выборе компонентов частей корреляция между ними (для реальных баз с неравномерным распределением данных) может приводить к неравномерному распределению векторов базы по корзинам таблиц. Это увеличивает количество кандидатов и замедляет выполнение запроса (например, если некоторая часть векторов одинакова у всех векторов базы и у вектора-запроса, поиск вырождается в линейный).

Декорреляция компонентов в частях на основе анализа базы приводит к 20–50 % ускорению поиска [52, 51]. В [53] в целях лучшей балансировки корзин для каждой части вектора генерируют новый хэш с использованием проецирования на главную ось векторов этой части. В [54] компоненты вектора взвешивают таким образом, что компоненты с более близкой к 0.5 частоте 1 (или 0) получают больший вес, и разбивают векторы на части (причем неодинаковой размерности) так, что в каждой части суммарный вес одинаков. В [55] разбиение проводят решением оптимизационной задачи, минимизирующей сумму квадратов разности энтропий векторов каждой части и ее среднего значения. В [56] для ускорения поиска за счет сокращения количества кандидатов предлагают использовать части различной размерности, при выборе которых не учитываются особенности базы, рассматривается также приближенный поиск.

В работе [53] для повышения качества ранжирования кандидатов используют взвешивание компонентов вектора. В [55] удается избежать этапа уточнения кандидатов стратегией F&R. Для каждой части известны корзины (и векторы в них) для всех значений $\text{dist}_{\text{Ham}} \le r_p$ до соответствующей части вектора-запроса. Все радиусы запросов от 0 до r можно представить в виде суммы подходящих радиусов для каждой части. Для каждой такой комбинации радиусов пересечение множеств кандидатов, полученных для каждой части, дает множество векторов, которые являются ответом на запрос. Все эти множества векторов объединяют в окончательный ответ. Для малых r этот метод работает быстрее MIH и HmSearch. Разновидность MIH применяется в [57] для точного поиска kNN по $\text{sim}_{\cos}$ между бинарными векторами с использованием соотношений между $\text{sim}_{\cos}$ и dist_{Ham}.

3. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ ЛОКАЛЬНО-ЧУВСТВИТЕЛЬНОГО ХЭШИРОВАНИЯ

В отличие от обычного хэширования в локально-чувствительном (locality-sensitive hashing, LSH) LSH-функции генерируют хэш-значения так, что вероятность их совпадения у сходных объектов выше, чем у несходных (см. обзоры в [58, 59, 17, 31]). Индексные структуры на основе LSH — одни из немногих с гарантиями сублинейного времени в худшем случае для некоторых типов запросов приближенного вероятностного поиска по сходству, которые реализованы на практике. Большинство LSH-функций разработано для определенных мер расстояния/сходства вещественных векторов и могут применяться для таких же мер бинарных векторов, как для частного случая. Ряд LSH-функций предназначен непосредственно для различных мер сходства бинарных векторов, включая разреженные большой размерности.

3.1. 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 [58, 59]: 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 $\Pr_F\{h(a) = h(b)\} \le p_2$, 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 дано на основе зазора расстояний между сходными и несходными объектами, аналогичное определение LSH существует [60] на основе зазора сходств.

В другом определении LSH на основе монотонности величины сходства [61] 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 [62]. Аналогичное определение существует и для расстояний. Из определения LSH на основе монотонности следует определение на основе зазора [62].

Простым примером LSH-функции для dist_{Ham} является функция h, которая случайно выбирает один из компонентов вектора $\mathbf{a}$: $h(\mathbf{a}) = a_i$ [59], т.е. выполняет сэмплирование с замещением [17, 20]. Для этой LSH-функции $\Pr\{h(\mathbf{a}) = h(\mathbf{b})\} = \Pr\{a_i = b_i\} = 1 - \text{dist}_{\text{Ham}}(\mathbf{a}, \mathbf{b})/D$.

3.2. Запросы приближенного поиска по сходству. Приведем типы запросов приближенного поиска по сходству, которые выполняют различные ИС LSH.

Обозначим (c)-rNN1 запрос c-приближенного r-ближнего соседа (c-approximate r-near neighbor); он возвращает любой объект базы с расстоянием, не превышающим cr ($c > 1$), от объекта-запроса, если существует объект базы с расстоянием, не большим r, от объекта-запроса.

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, не превышающим cr, от объекта-запроса, если существует объект базы с расстоянием, не большим r, от объекта-запроса. Если такого объекта не существует, то следует ответ none или возвращается объект с расстоянием, не превышающим cr.

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

Редукция запросов приближенного ближайшего соседа различного типа к запросам ближнего соседа рассмотрена в [30, 59, 31] (см. также подразд. 1.2). Однако эта редукция требует построения отдельных ИС для многих r, что значительно увеличивает затраты памяти и поэтому не применяется на практике. Для выполнения запросов приближенного NN часто используют ИС без специфицированных гарантий (качество поиска 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 — для ближайшего соседа.

3.3. 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. (Таким образом, в LSH сэмплированием для dist_{Ham} таблицам соответствуют различные части векторов, которые могут пересекаться отдельными компонентами в отличие от ИС, рассмотренных в разд. 2.) 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 [63].

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» [58]). Для запроса (δ)-rNN используют все объекты из этих корзин («Стратегия 2» [58]). Окончательный результат получают линейным поиском в списке кандидатов по мере сходства/расстояния запроса.

Для стратегии 1 выбирают $m = \log_{1/p_2} N = \log N / \log 1/p_2$, $L = p_1^{-m}$ [59]. Результат поиска получают с постоянной вероятностью $1-\delta$. Уменьшить δ до произвольно малого значения можно применением нужного количества (L-табличных) ИС LSH. 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 $c \to \infty$. Например, LSH сэмплированием для бинарных векторов при $c > 1$ имеет $\rho \le 1/c$ [58, 59]. Это плотная граница, так как нижняя граница $\rho \ge 1/c - o_D(1)$ [64]. Поэтому для $c = 2$ получаем $\rho = 1/2$, т.е. находим r-ближнего соседа с аппроксимацией 2 за время $O(\sqrt{N})$. Отметим, что логарифмическое время не обеспечивается.

При расчете параметров для стратегии 2 (запрос (δ)-rNN) выбирают [63] $L = \lceil \log \delta / \log(1 - p_1^m) \rceil$, а значение m, минимизирующее время запроса, определяют экспериментально для базы и запросов. Также L выбирают (подразд. 3.6.1), исходя из имеющейся памяти, а $m = \lceil \log(1 - \delta^{1/L}) / \log p_1 \rceil$.

3.4. Примеры LSH-функций для бинарных векторов.

3.4.1. Сходство Жаккара. Для $\mathrm{sim}_{\mathrm{Jac}}$ LSH-хэширование MinHash на основе случайных перестановок предложено в [65–67]. Целочисленное хэш-значение MinHash есть минимальный номер ненулевого компонента вектора, полученного случайной перестановкой исходного. Для различных перестановок получают различные значения MinHash. Показано, что $\Pr\{\min h(\mathbf{a}) = \min h(\mathbf{b})\} = \mathrm{sim}_{\mathrm{Jac}}(\mathbf{a}, \mathbf{b})$. Поэтому MinHash является LSH.

Для снижения затрат памяти и вычислительной сложности получения множества хэш-значений MinHash предложены модификации (см. обзоры в [17, 68, 69] и ссылки к ним). Одно из направлений — сокращение количества битов в хэш-значении (вплоть до одного) в результате отбрасывания старших битов [70], что несколько увеличивает дисперсию оценки сходства. Другое направление — уменьшение времени формирования скетчей. Использование одной перестановки (или ее эмуляции) с последующим разбиением вектора на части и извлечением хэш-значения из каждой части требует решения проблемы пустых частей [71, 17]. Различные варианты заимствования хэш-значений из соседних корзин [17] увеличивают дисперсию оценки сходства. Вариант [71] приближает дисперсию к MinHash с помощью выбора части, из которой заимствуют хэш-значение, посредством 2-Universal Hashing с использованием номеров текущей кор-

зины и попытки хэширования. Предложенный в [69] метод получения хэш-значений без перестановки и разбиения на основе mixed tabulation hash function уменьшает дисперсию оценки сходства по сравнению с одной перестановкой и даже с MinHash. Для построения LSH-таблиц создают скетч и из него определенным образом сэмплируют хэш-значения.

В работе [72] показано, что MinHash может использоваться и как LSH для $\mathrm{sim}_{\cos}$ и имеет значение ρ меньшее, чем для LSH SimHash [61] (которое предложено для $\mathrm{dist}_{\mathrm{ang}}$ между векторами и может применяться для $\mathrm{sim}_{\cos}$). Это справедливо и для 1-битового варианта MinHash. Последние улучшения для $\mathrm{sim}_{\mathrm{Jac}}$ рассмотрены в подразд. 3.6.3.

3.4.2. Скалярное произведение. Для $\mathrm{sim}_{\mathrm{dot}}$ в общем случае LSH не существует ни для бинарных векторов, ни для вещественных [72, 73]. Асимметричное LSH для $\mathrm{sim}_{\mathrm{dot}}$ бинарных векторов предложено в [62]. Перед применением MinHash векторы-запросы и векторы базы преобразуют по-разному. Пусть A — максимальное количество ненулевых компонентов в векторах базы. К векторам базы $\mathbf{y}$ добавляют $A-|\mathbf{y}|$ единичных компонентов и $A+|\mathbf{y}|$ нулевых; к векторам-запросам $\mathbf{x}$ добавляют A нулевых компонентов, $A-|\mathbf{x}|$ единичных и $|\mathbf{x}|$ нулевых. Для полученных представлений поиск по $\mathrm{sim}_{\mathrm{Jac}}$ дает такой же результат, как поиск по $\mathrm{sim}_{\mathrm{dot}}$ или $\mathrm{sim}_{\cos}$. Результаты поиска лучшие, чем при использовании асимметричных LSH-функций для вещественных векторов, особенно для разреженных векторов большой размерности. Такое же преобразование позволяет использовать LSH для $\mathrm{dist}_{\mathrm{Ham}}$ при поиске по $\mathrm{sim}_{\mathrm{dot}}$ [62, 74].

3.5. Поиск приближенного ближайшего соседа. Для поиска ближайшего соседа редукцией к поиску ближнего (см. подразд. 1.2 и 3.2) необходимо построение ИС для многих радиусов запроса, причем каждая ИС LSH должна состоять из L таблиц, что требует очень больших затрат памяти.

В работе [63] базовую ИС LSH для запроса (c,δ)-rNN1 эвристически используют для поиска приближенного ближайшего соседа, полагая расстояние до него известным. В [75] для LSH сэмплированием более равномерное распределение компонентов вектора по таблицам (за счет жадного выбора редко встречающихся компонентов) повысило точность поиска NN.

В работе [61] предложено генерировать $L=O(N^{1/c})$ случайных перестановок компонентов векторов базы, для каждой перестановки N векторов базы заносят в свою таблицу, упорядочив лексикографически. При поступлении вектора-запроса его соответствующие перестановки используются для поиска дихотомией двух ближайших векторов в каждой из L таблиц. Эти $2L$ векторов базы являются кандидатами, с которыми вычисляется $\mathrm{dist}_{\mathrm{Ham}}$ вектора-запроса для получения приближенного NN (с неизвестным расстоянием) с постоянной вероятностью. Выбор большего количества ближайших к запросу векторов в каждой таблице (вместо увеличения количества ИС) на практике уменьшает вероятность false negative.

В EWH [76] на этапе выполнения запроса генерируют модификации хэшей запроса (до некоторого $\mathrm{dist}_{\mathrm{Ham}}=R$ аналогично модификации запроса в подразд. 1.4.1) и используют в качестве начальных кандидатов все векторы из соответствующих корзин. Им присваивают рейтинг $\sum_{i=0}^{R} B_i C_i$, где C_i — количество хэшей, соответствующих кандидату, с $\mathrm{dist}_{\mathrm{Ham}}=i$ от хэша вектора-запроса, B_i — вес (например, $B_i=(m-i)/(mL)$). Для уточнения отбирают кандидатов с рейтингом, превышающим пороговый. Параметры определяют решением оптимизационной задачи для конкретной базы. Вследствие взвешивания кандидатов размер их списка в EWH гораздо меньший, чем в LSH при том же качестве. В экспериментах с реальными данными при D, равной до 1024, с расстоянием до NN, равным до $D/4$, при том же времени запроса ошибка EWH до 15 раз меньше LSH, а при той же точности время запроса до 10 раз меньше.

3.6. Развития подхода LSH.

3.6.1. Чувствительность к векторам базы и запросов. Для базовой ИС LSH, настроенной на выполнение запроса (c,δ)-rNN1, реальные времена выполнения запросов зависят от распределения расстояний от конкретного вектора-запроса до векторов конкретной базы и могут значительно отличаться. В [77, 78] предлагают модификации ИС LSH, позволяющие в процессе выполнения запроса получить время, которое дала бы ИС LSH с параметрами, оптимально выбранными для конкретных вектора-запроса и векторов базы (что требует знания конкретного распределения расстояний между ними).

В работе [78] рассматривают запрос (δ)-rNN. Для базовой ИС LSH и «трудных» запросов среднее количество кандидатов может составлять до $O(\text{occ}L)$. Для решения этой проблемы в многоуровневом адаптивном LSH [78] строят наборы (уровни) LSH-таблиц для различных сочетаний m и L (L увеличивается от m). Для всех корзин помимо ссылок на их объекты хранят количество объектов. При выполнении запроса для наборов (уровней) в порядке возрастания m определяют количество кандидатов (суммированием количества объектов в корзинах таблиц уровня, адресуемых хэшами запроса) и прекращают поиск (по критерию останова) на некотором уровне. При этом среднее количество кандидатов $\Theta(\text{occ}\,(N/\text{occ})^{\rho})$ всегда меньше N, а вариант ИС с модификацией запроса приближает его к нижней границе $O(N^{\rho}+\text{occ})$. При малой внутренней размерности части объектов базы, близких к запросу, верхние границы на среднее время запроса могут быть улучшены до $O(\log N)$. Практической реализации подхода препятствуют большие затраты памяти.

В алгоритме [79], который может применяться на практике, для запроса (δ)-rNN конструируют ИС LSH для фиксированного L, выбирая $m=\lceil \log(1-\delta^{1/L})/\log p_1 \rceil$ (см. подразд. 3.3, 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 Hybrid LSH [79] с каждой хэш-корзиной ассоциирована структура Hyperloglog [80], которая позволяет аппроксимировать число объектов без дубликатов в выбранных хэшем запроса L корзинах.

Отметим, что подходы этого подраздела применимы также и к вещественным векторам.

3.6.2. Точный поиск бинарных векторов по расстоянию Хэмминга. Индексная структура covering LSH (cLSH) [30] в отличие от базовой ИС LSH позволяет выполнять запросы по dist_{Ham} без возможности потери некоторых объектов-ответов (т.е. без false negatives). Вместо случайного независимого выбора компонентов g семейство cLSH-функций конструируют так, что для любой пары бинарных векторов с $\text{dist}_{\text{Ham}} \le r$ существует функция семейства, для которой достигается коллизия их cLSH-хэшей.

Для $cr=\log N$ получают $L=2^{r+1}-1\approx 2N^{1/c}$ и $\rho=1/c$, т.е. оптимальное для базового LSH. Для произвольных r, c, N значение $\rho=1.38/c$. При выполнении запроса ближнего соседа возвращают первый вектор-кандидат $\mathbf{y}$: $\text{dist}_{\text{Ham}}(\mathbf{y},\mathbf{x})\le cr$. Если ограничить количество кандидатов (стратегия 1), получим ответ на запрос (c,δ)-rNN1 за гарантированное время $DN^{\rho+o(1)}$, однако с возможностью false negative. В противном случае получим поиск без false negative со средним временем запроса (c)-rNN1 менее $DN^{\rho+o(1)}$ (рандомизация Las Vegas). Для определения границ времени с высокой вероятностью рассматривают набор из $O(\log N)$ ИС cLSH. Если поиск в ИС превысил в два раза среднее время, переходят к следующей ИС, а если все ИС пройдены, применяют линейный поиск, что случается с полиномиально малой вероятностью от N. Для запроса rNN из корзин с коллизиями извлекают всех кандидатов (стратегия 2). Так

как g имеют большую размерность D, вариантами универсального хэширования их преобразуют в хэш-значения приемлемой размерности $O(\log N)$.

В быстром covering LSH (fcLSH) [81] сложность получения (итоговых, т.е. после универсального хэширования) значений fcLSH-функций уменьшают с $O(LD)$ до $O(D + L\log L)$ с помощью использования матрицы Адамара и быстрого преобразования Адамара FHT. Для r, равного до 20, наблюдается превосходство по времени над LSH, cLSH, MIH [51].

Недостатком cLSH и fcLSH является экспоненциальная зависимость L от r.

3.6.3. Локально чувствительная фильтрация. Разбиения пространства на области с помощью LSH-функции выполняются без перекрытия. Кроме того, области довольно малые, что ограничивает количество несходных векторов в корзине. В [82] предложена альтернатива LSH — локально чувствительные фильтры (Locality Sensitive Filters, LSF). В отличие от LSH каждому фильтру LSF соответствует одна случайная область пространства, области различных фильтров могут перекрываться и их количество может быть очень большим. Для определения, каким областям принадлежит объект, за время, пропорциональное количеству таких областей, используют специальные структуры данных и не вполне случайные области, которые, однако, обеспечивают нужные характеристики [82–84]. Индексные структуры LSF позволяют гибкий компромисс между затратами памяти и временем поиска (включая режим LSH), а также меньшие значения ρ в режиме LSH для данных небольшой размерности $D = O(\log N)$. Подобно LSH в обычном LSF нулевая вероятность false negatives обеспечивается только при количестве областей, стремящемся к бесконечности.

В [84] предложено обобщение LSH, LSF и ИС в категорию Locality-Sensitive Maps. Рассматривают запросы (c, δ)-rNN1 по $\mathrm{sim}_{\mathrm{BB}}$. Для векторов с одинаковой хэмминговой нормой это позволяет выполнять поиск и по другим мерам расстояния/сходства для бинарных векторов (см. подразд. 1.1). Для таких разреженных векторов получено ρ лучшее, чем для других LSH-функций (MinHash, сэмплирующий LSH, сферический LSH), и даже лучшее, чем для зависимого от данных LSH (подразд. 3.6.4) для значительной области значений сходств.

На основе LSF в [85] предложен теоретический алгоритм поиска (c)-rNN1 по $\mathrm{dist}_{\mathrm{Ham}}$ без false negatives с $\rho \le 1/c + o(1)$, т.е. лучшим, чем $\rho = 1.38/c$ в [30]. При $cr \approx D/2$ получено $\rho = 1/(2c-1)$ (подразд. 3.6.4). Для $\mathrm{sim}_{\mathrm{BB}}$ и $\mathrm{sim}_{\mathrm{Jac}}$ достигают значения ρ из [84], но без false negatives.

3.6.4. Зависимое от данных LSH. Для зависимого от данных LSH (но с гарантиями худшего случая) для $\mathrm{dist}_{\mathrm{Ham}}$ в [86] получено оптимальное значение $\rho = 1/(2c-1) + o(1)$, т.е. лучшее, чем $\rho = 1/c$ для независимого от данных сэмплирующего LSH. Однако зависящая от данных LSH-функция вычислительно слишком сложна для практического применения. В [87] проведен анализ модификации LSH Forest [88] (вместо LSH-таблиц используются префиксные деревья), построенного на случайно переставленных векторах базы. Хотя полученное ρ хуже оптимального $\rho = 1/(2c-1)$, этот алгоритм может применяться на практике и ρ лучше $1/c$.

4. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ ДЕРЕВЬЕВ

Для поиска бинарных векторов по метрическим расстояниям, например $\mathrm{dist}_{\mathrm{Ham}}$, в принципе применимы древовидные ИС с иерархическими разбиениями на области, разработанные для более общих случаев метрических и векторных пространств [21, 22, 23, 1].

Древовидные ИС для точного поиска по $\mathrm{dist}_{\mathrm{Ham}}$ (включая небинарные строки с различными алфавитами для компонентов с различными ID) предложены в [89, 90]. К ИС класса разбиения данных (типа R-tree [22]) относится ND-tree [89], а к ИС класса разбиения пространства (типа KD-tree [22]) относится NSP-tree [90]. Преимуществами деревьев для точного поиска являются: детерминированность алгоритма, возможность выполнения запросов rNN с изменяемым радиусом и запро-

сов kNN. К их недостаткам относятся: экспоненциальная зависимость времени поиска от D (в худшем и даже в среднем случае), отсутствие учета особенностей бинарных векторов, а для ND-tree [89] и NSP-tree [90] — экспериментальное сравнение только с метрическими деревьями [23, 1] и линейным поиском.

Широко распространено ускорение поиска древовидными ИС за счет раннего останова поиска [4], что делает его приближенным, без строгих гарантий качества результатов относительно линейного поиска. В [75] в задаче поиска приближенного NN для бинарных векторов по $\mathrm{dist}_{\mathrm{Ham}}$ экспериментально исследованы ИС для вещественных векторов: лес (множество) KD-trees [39], KM-tree [39] на основе иерархической кластеризации K-means, VP-tree [4] (нелистовые узлы соответствуют центрам кластеров, данные находятся в листовых узлах). Получены плохие результаты. Для леса KD-trees их объясняют [75] чувствительностью к шуму в данных: смена одного бита приводит к переходу на другое поддерево. Наличие большого количества векторов с одинаковым $\mathrm{dist}_{\mathrm{Ham}}$ от центров областей Вороного («толстые границы»), особенно при малых D, приводило к их случайному назначению одной или другой области в использованной реализации KM-tree и VP-tree.

Для устранения этих недостатков в [39, 75] применяют лес рандомизированных KM-trees, модифицированных за счет случайного (без повторения) выбора векторов базы в качестве центров кластеров. В FLANN [39] для всех деревьев используют общую приоритетную очередь, куда при спуске из корневого в листовой узел (в листьях хранятся объекты базы), области которого принадлежит вектор-запрос, заносят узлы промежуточных уровней в порядке возрастания расстояний до вектора-запроса. По достижении листа все его объекты сравнивают с вектором-запросом линейным поиском по $\mathrm{dist}_{\mathrm{Ham}}$. После однократного спуска корень–лист по всем деревьям поиск продолжают, извлекая из очереди ближайший узел и осуществляя обход дерева от него. Поиск останавливают по посещении заданного количества векторов базы. В лесу Parc-trees [75] используют однократный спуск корень–лист без дальнейшего обхода, количество посещенных векторов регулируют коэффициентом ветвления K и количеством деревьев. Такие ИС позволяют выполнять запрос приближенных kNN и на практике используются с векторами размерностью до сотен битов ($D = 256$). По сравнению с линейным поиском выполнение запроса ускоряется в 10–100 раз для точности в диапазоне 50–99 %. Это меньше, чем ускорение подобных ИС для вещественных векторов, так как линейный поиск для бинарных векторов очень быстрый. Сравнение с LSH показало примерно одинаковую или лучшую скорость FLANN при в несколько раз меньших затратах памяти (например, в шесть раз для точности большей 90 %). Таким образом, при использовании леса получают неплохие результаты для приближенного поиска по $\mathrm{dist}_{\mathrm{Ham}}$.

В работе [91] центры кластеров выбирают не случайные, а удаленные от уже отобранных более чем на пороговое расстояние (другие подобные эвристики см. в [4]). Рассматриваемые ИС строят по векторам сниженной размерности, которые получают упорядочением компонентов по величине дисперсии и отборам заданного количества. Результаты превышают MIH и бинарный FLANN. Настройка центров кластеров под данные (например, в KMedians компоненту присваивают такое же значение, что и у большинства векторов кластера) рассмотрена в [92–94].

В работе [95] решают задачу поиска по $\mathrm{sim}_{\mathrm{dot}}$ в базе плотных случайно сгенерированных бинарных векторов $\{-1, +1\}^D$ большой размерности D, вектор-запрос является случайно искаженным вектором базы (разд. 6). Последнюю иерархически разбивают на непересекающиеся части, для каждой из которых строят прототип суммированием ее векторов, взвешенных случайными числами из $\{-1, +1\}$. Прототипам соответствуют узлы дерева. При выполнении запроса находят $\mathrm{sim}_{\mathrm{dot}}$ вектора-запроса с векторами узлов и переходят в узел с наибольшим значением. Остальные значения и их узлы запоминают в приоритетной очереди. В листьях за-

поминают значения $\mathrm{sim}_{\mathrm{dot}}$ с соответствующими векторами базы. Поиск останавливают по превышении $\mathrm{sim}_{\mathrm{dot}}$ порогового значения, которое определяют с учетом величины искажения вектора-запроса. Если оно не превышено в листе, из приоритетной очереди извлекают узел с наибольшим значением. Для повышения надежности поиск продолжают и после выполнения критерия останова. При надлежащем выборе порога вероятность получения точного NN близка к 1.

В [96] ИС HA-index для поиска с изменяемым r можно рассматривать как иерархический граф, где промежуточные узлы представляют подмножества компонентов, не пересекающиеся в различных путях от верхнего уровня к нижнему. Перед построением векторы упорядочивают в порядке Грэя. Векторы базы хранятся в листьях и представляют собой конкатенацию подмножеств компонентов в различных путях от узлов верхнего уровня в лист. При поиске выполняют обход методом breadth-first, в очередь заносят непосещенные пути, которые удовлетворяют условиям запроса. Ускорения выполнения запроса достигают за счет устранения избыточных и дублирующихся вычислений расстояний. Метод значительно лучший, чем MH [47] и HEngine [48] для rNN, а также LSH и LSB-tree [97] для приближенного kNN, как по затратам памяти, так и по времени запроса, поэтому актуально его сравнение с другими ИС.

Отметим, что все рассмотренные ИС применялись для плотных бинарных векторов.

5. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ ГРАФОВ СОСЕДСТВА

Для приближенного поиска ближайших соседей в метрических и неметрических пространствах успешно используют графы соседства (объекты базы представлены узлами, каждый объект базы соединен с ближайшими объектами) (см. обзор в [4]). Запрос выполняют стартом с некоторого объекта базы и переходом на непосещенный объект-сосед, более близкий к объекту-запросу. Переходы выполняют, пока существуют такие объекты-соседи.

Поскольку лучший из найденных объектов может не являться ближайшим (или даже «хорошим») соседом, выполняются рестарты поиска с других объектов или переход на не самые близкие к объекту-запросу объекты окрестностей. Точный поиск требует посещения всех объектов базы. Для ограничения времени запроса поиск останавливают по достижении заданного количества посещенных объектов. Качество результатов в значительной степени зависит от того, насколько сходны с ближайшим соседом стартовые объекты.

В работе [98] предложена модификация ИС этого класса для бинарных векторов, которая строит дополненный граф соседства. Он является комбинацией графа соседства векторов базы (каждый вектор базы соединен с ближайшими соседями) и мостового графа (каждый мостовой вектор соединен с ближайшими соседями базы). Мостовые векторы не принадлежат базе, но являются хорошими стартовыми векторами. Их формируют следующим образом. Векторы разбивают на части. Все N векторов подвергают кластеризации на K кластеров по каждой из M частей. Начальными центрами являются случайные векторы базы, затем центры модифицируют назначением компоненту значения, наиболее часто встречающегося у векторов кластера (см. разд. 4). Мостовой вектор (всего их K^M) является комбинацией центральных векторов кластеров. Для сокращения вычислений мостовой граф строят приближенно, выбирая для каждого вектора базы 1000 ближайших мостовых, а затем для каждого мостового 50 ближайших векторов базы. Приближенный граф соседства базы строят приближенным поиском FLANN ([39] и см. разд. 4).

При поиске по дополненному графу соседства вначале помещают в приоритетную очередь мостовой вектор, ближайший к запросу. На каждой итерации, если вектор базы первый в очереди, его непосещенных соседей по графу соседства помещают в очередь. Если мостовой вектор первый в очереди, в нее помещают его соседей по мостовому графу, а также следующий ближайший мостовой вектор. Показано преимущество по скорости поиска над FLANN и MIH при близкой к 1 точности.

Отметим, что разновидности графов соседства, известные как графы тесного мира [99, 4], демонстрируют наилучшие результаты как среди графов соседства, так и среди других ИС.

6. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ НЕЙРОСЕТЕВОЙ АВТОАССОЦИАТИВНОЙ ПАМЯТИ

Разновидности ассоциативной памяти [100] (или памяти, адресуемой по содержимому) можно рассматривать как ИС для некоторых типов поиска по сходству. В автоассоциативной памяти на выходе получают бинарный вектор, наиболее сходный с вектором на входе, т.е. она выполняет запрос NN для бинарных векторов.

В распределенной памяти различные векторы хранят в общих ячейках памяти. Нейробиологически правдоподобные варианты распределенной памяти можно представить в виде искусственных нейронных сетей, в которых запоминание вектора обычно выполняется за одно его предъявление локальным обучающим правилом, модифицирующим веса межнейронных связей, а восстановление вектора памяти в ответ на вектор-запрос осуществляется итеративной процедурой распространения активности между нейронами по связям. Поиск обычно выполняется по $\mathrm{sim}_{\mathrm{dot}}$.

Кратко рассмотрим нейросетевые варианты распределенной автоассоциативной памяти (обозначим ее Neural Associative Memory, NAM). Более подробный обзор представлен в [100].

6.1. Обобщенная схема и характеристики NAM. Будем рассматривать, главным образом, распределенные NAM матричного типа, которые являются полносвязными сетями бинарных нейронов. Каждый из D нейронов представляет компонент бинарного вектора $\mathbf{z}$ размерности D, т.е. нейрон может быть в состоянии 0 или 1. Каждая пара нейронов имеет две взаимные связи, элементы матрицы связей $\mathbf{W}(D \times D)$ соответствуют весам связей. В режиме обучения векторы базы $\mathbf{y}$ «запоминают» в матрице $\mathbf{W}$, применяя некоторые «обучающие правила», которые изменяют значения w_{ij} (изначально w_{ij} обычно нулевые).

При выполнении запроса на сеть подают бинарный вектор $\mathbf{x}$ активацией нейронов: $\mathbf{z} = \mathbf{x}$. Вычисляют входную сумму нейрона $\mathbf{x}$: $s_i = \sum_{j=1}^{D} w_{ij} z_j$. Состояние нейрона определяют как $z_i(t+1) = 1$ (активен) при $s_i(t) \geq T_i(t)$ и $z_i(t+1) = 0$ (неактивен) при $s_i(t) < T_i(t)$, T_i — значение порога нейрона. При параллельной (синхронной) динамике сети на каждом шаге t определяют (обновляют) состояния всех D нейронов. Параметры $\mathbf{W}$, T устанавливают так, чтобы после одного или нескольких шагов динамики сеть приходила в стабильное состояние. В этом состоянии вектор $\mathbf{z}$ — выход сети.

Вектор-запрос $\mathbf{x}$ — обычно искаженный вектор базы. Если количество запомненных векторов не слишком велико, выход $\mathbf{z}$ является запомненным вектором базы $\mathbf{y}$, ближайшим к $\mathbf{x}$ по $\mathrm{sim}_{\mathrm{dot}}(\mathbf{x}, \mathbf{y})$, т.е. NAM возвращает точного NN. Сложность (время выполнения) одного шага динамики сети $O(D^2)$. Поэтому, если в NAM удается запомнить $N > D$ векторов с возможностью восстановления из искаженных версий, время выполнения запроса может оказаться меньшим $O(DN)$. Так как затраты памяти этого типа NAM $O(D^2)$, с ростом D возможно увеличение количества N векторов, которое может быть запомнено и восстановлено.

К сожалению, вектор на выходе NAM может не являться ближайшим соседом вектора-запроса, и даже вектором базы. Кроме того, иногда в NAM удается восстановить только $N << D$ векторов, особенно если векторы-запросы сильно искажены.

В качестве векторов базы зачастую рассматривают бинарные векторы, случайно выбранные из некоторого распределения (например, векторы с вероятностью $p = 1/2$ либо $p < 1/2$ компонентов со значениями 1). Векторы-запросы являются искаженными векторами базы. Искажение удалением случайно стирает

часть единичных компонентов вектора базы (остальные компоненты гарантированно совпадают). Более сложное для поиска искажение шумом случайно удаляет и добавляет единичные компоненты с сохранением (точным или приближенным) их общего количества. Отметим, что последнее распределение известно как the light bulb problem [101] и его исследуют в LSH [83, 86].

При запоминании слишком большого количества векторов NAM «перегружается» и перестает выполнять функцию автоассоциативной памяти. Значение N, при котором NAM продолжает восстанавливать вектор, зависит от ее конструкции, распределения векторов базы и запросов, степени искажения векторов-запросов. Часто исследуют информационные характеристики NAM (в терминах информации Шеннона, которая запоминается и извлекается из NAM, и ее максимальных значений), из них можно определить характеристики в терминах N.

6.2. Сети Hopfield. В NAM Hopfield [102] используют «плотные» случайные бинарные векторы (с компонентами из $\{0,1\}$ с вероятностью $p = 1/2$ единичного значения). Процедура обучения формирует (симметричную) **W** поочередным запоминанием векторов базы **y** по правилу Hopfield: $w_{ij} = w_{ij} + (y_i - p)(y_j - p)$, $p = 1/2$, $w_{ii} = 0$. Динамика восстановления в [102] последовательная с порогом $T = 0$ (во многих последующих исследованиях и реализациях динамика параллельная). Показано [102], что такая сеть приходит в устойчивое состояние, причем до $N = 0.14D$ запомненным векторам соответствуют устойчивые состояния, при меньших N возможно восстановление искаженных векторов. Сходные результаты получены как асимптотически (при $D \to \infty$), так и при конечных D.

Для случайных разреженных векторов (при $p < 1/2$) в сети Hopfield достижимы лучшие характеристики, чем при $p = 1/2$ [103]. Например, восстановление может выполняться и при $N >> D$. Значение порога $T(t)$ в этом случае выбирают положительным, например таким, чтобы обеспечить pD активных нейронов при параллельной динамике. Детальные теоретические и экспериментальные исследования информационных характеристик сети Hopfield при различных искажениях входного вектора и D, p, N [104, 105] показали, что использованные методы теоретического анализа плохо предсказывают характеристики конечных сетей для малых pD. Наилучшие характеристики в терминах извлекаемой из сети информации получены при $p = 0.001$–0.01, что соответствует активности нейронов мозга.

В [106–108] оценивают количество векторов памяти, которые являются устойчивыми состояниями, а также могут быть восстановлены из искаженных шумом векторов-запросов как пропорциональное $(D / \ln D)^2$ при $p = \ln D / D$ (асимптотически, т.е. при $D \to \infty$, со стремящейся к 1 вероятностью). Если аппроксимировать количество шагов воспроизведения как $\ln D$, ускорение поиска по сравнению с линейным поиском составляет порядка $D / (\ln D)^3$. Исследование в [109] привело авторов к выводу о возможности ускорения поиска с помощью сети Hopfield по сравнению с другими ИС для малых p и больших D, N.

6.3. Сети Willshaw. В автоассоциативном варианте NAM Willshaw [110–112] связи бинарные из $\{0,1\}$. Правило обучения Willshaw: $w_{ij} = w_{ij} \vee (y_i \wedge y_j)$, где $\vee$ — дизъюнкция, $\wedge$ — конъюнкция. В отличие от сети Hopfield при постоянном p и росте N пропорционально D сеть Willshaw перестает работать, так как все связи становятся единичными. Поэтому используют $p \sim \ln D / D$.

Из аналитического и экспериментального исследований [113, 114] следует, что в отличие от сети Hopfield, в которой информационные характеристики (см. [100]) монотонно растут с D, в сети Willshaw для не слишком больших D их значения больше, чем при $D \to \infty$. Вследствие бинарности связей удельные характеристики (на бит памяти) лучшие, чем у сети Hopfield.

6.4. Сети Potts. В работах [115–117] NAM рассматривается как сеть нейронов, которые разбиты на $\sim \ln D$ не пересекающихся частей («колонок»), в каждой колонке только один активный нейрон. Для обучения используют правило Hopfield, а также Willshaw [118, 119]. Между нейронами в колонке не существу-

ет связей. Динамика сети активирует в каждой колонке один нейрон с максимальной входной суммой. Согласно [118] информационные характеристики подобны сети Willshaw при аналогичной разреженности векторов.

Аналитическое и экспериментальное сравнения сетей Hopfield, Willshaw, Potts с правилом Willshaw для векторов с $p \sim \ln D / D$ и искажениями векторов-запросов удалением проведено в [108]. Теоретически исследуют восстановление за один шаг (асимптотически, т.е. при $D \to \infty$, со стремящейся к 1 вероятностью). Для всех моделей получена нижняя граница N порядка $(D / \ln D)^2$, а для сети Willshaw показано совпадение с верхней границей.

Отметим, что функции NAM выполняются и в неполносвязных сетях как для плотных, так и для разреженных векторов (например, [120, 121]). Это можно использовать для уменьшения затрат памяти NAM с квадратичных до линейных от D.

6.5. Другие направления в NAM. Заслуживают внимания ИС для ускорения поиска по сходству, в которых модули NAM используются на отдельных этапах поиска. В [122] ИС применяют несколько NAM для запоминания частей базы, а сходство результата одношагового воспроизведения NAM с запросом используют для выбора «лучшей» NAM для выполнения точного линейного поиска по ее векторам.

Реальные данные во многих случаях не являются бинарными разреженными векторами большой размерности, с которыми лучше всего работают рассмотренные NAM. Поэтому требуются сохраняющие сходство преобразования представлений различного типа в этот формат (см. обзоры в [16, 17, подразд. 5.1]). Однако полученные векторы (как и исходные реальные данные) не являются случайными и независимыми, поэтому аналитические и экспериментальные результаты, имеющиеся в основном для таких векторов, обычно не могут предсказывать характеристики NAM для реальных данных.

В NAM рассмотренного типа возникает функция, отличная от функции ассоциативной памяти, которая может рассматриваться как функция обобщения [100]. Даже при запоминании случайных векторов в нейросети возникают дополнительные устойчивые состояния. Для коррелированных векторов нейроны, представляющие их общие единичные компоненты, становятся «тесно» связанными, что приводит к возникновению устойчивого состояния сети, соответствующего этим компонентам. Выявление таких состояний можно использовать для анализа данных (например для бинарного факторного анализа [123]) и как модель обобщения информации в мозге, включая появление иерархий категоризации [100].

В NAM со связями высоких порядков $n > 2$ [124–128] связи имеются не между парой ($n = 2$), а между большим количеством нейронов n. Такие NAM позволяют запомнить количество плотных векторов, пропорциональное D^{n-1}, однако это достигается соответствующим увеличением количества связей, а значит, памяти и времени обработки. Обобщение этого типа NAM [126, 127] позволяет провести интересные аналогии с персептронами и ядерными методами в задаче классификации. Однако для поиска NN время запроса линейно от N.

В последнее время появились [129–132] автоассоциативные NAM со структурой двудольного графа, где матрицы связей задают линейные ограничения, налагаемые на (небинарные) векторы. Они позволяют восстанавливать $N \sim \exp(D)$ векторов. Однако это справедливо для данных из моделей, которым реальные данные зачастую не соответствуют.

Отметим, что разновидности сети Hopfield используют для оптимизации (см., например, публикации [133, 134] и ссылки к ним).

ЗАКЛЮЧЕНИЕ

Рассматриваемые в разд. 2–5 ИС при конструировании разбивают множество объектов базы на подмножества, обычно не пересекающиеся. При выполнении

запроса поиска по сходству быстро определяют «перспективные» подмножества, используют их объекты в качестве кандидатов, ответ на запрос получают вычислением расстояний/сходств кандидатов до объекта-запроса и принятием решения о включении нужных кандидатов в множество объектов-ответов. Применение нескольких независимых ИС улучшает результаты поиска. Если общее количество кандидатов меньше количества объектов базы N, возможно ускорение поиска относительно линейного поиска.

В ИС на основе хэш-таблиц (см. разд. 2, 3) значение хэша объекта определяет корзину ИС, которой он принадлежит. В отличие от обычного хэширования, где значения хэшей стремятся сделать различными для различных объектов, используют хэширование, которое сходным объектам генерирует одинаковые хэш-значения. При поиске объекты-кандидаты извлекаются из корзины, соответствующей хэшу объекта-запроса.

Для бинарных векторов в качестве хэша может непосредственно использоваться совокупность значений некоторого подмножества их компонентов. Если подмножество включает все компоненты вектора, в корзине каждого вектора базы содержится один этот вектор. Чтобы найти все векторы базы с $\mathrm{dist}_{\mathrm{Ham}}$ от вектора-запроса, не превышающем заданного радиуса, выполняют множество запросов на совпадение. Вектор-запрос в каждом таком запросе получают из исходного, изменяя значения некоторых его компонентов (число изменяемых компонентов вектора варьируется от нуля до величины радиуса). Количество запросов на совпадение, требующихся для точного выполнения исходного запроса, быстро растет с увеличением размерности векторов и радиуса и может превысить N. Поэтому в ИС, рассмотренных в разд. 2, компоненты векторов разбивают на непересекающиеся части (подмножества) и для каждой части конструируют свою табличную ИС, в которой находятся все N векторов базы. Это уменьшает размерность и позволяет уменьшить радиус модификации каждой части (иногда модификации части не требуется) с сохранением гарантий извлечения из всех частичных ИС кандидатов, содержащих точный ответ на исходный запрос. Такой подход позволяет создавать ИС для точного поиска с гарантиями сублинейного времени для данных среднего случая. Часть таких ИС работает только для фиксированного радиуса запроса, другие позволяют изменять радиус.

В ИС на основе LSH-хэширования (см. разд. 3) LSH-функция назначает одинаковые хэш-значения сходным объектам с большей вероятностью, чем несходным. Поэтому в одной корзине находятся, в основном, сходные объекты. В базовом варианте ИС LSH состоит из набора хэш-таблиц, для каждой используется своя реализация LSH-функции (задающая исчерпывающее разбиение пространства на непересекающиеся области). В каждой таблице находятся ссылки на все объекты базы, объект принадлежит одной корзине. При выполнении запроса извлекают кандидатов из одной корзины (соответствующей хэшу объекта-запроса) каждой таблицы. Использование множества таблиц снижает вероятность потери объекта, который является ответом на запрос.

Гарантии сублинейного времени для некоторых типов запросов приближенного поиска для данных худшего случая дают ИС LSH. К преимуществам ИС LSH также относится возможность создания ИС для различных мер расстояния/сходства (для которых разработаны LSH-функции). Так, для $\mathrm{dist}_{\mathrm{Ham}}$ используют подмножество компонентов векторов, однако в отличие от ИС, рассмотренных в разд. 2, они выбираются случайным сэмплированием с замещением, поэтому могут содержать одинаковые компоненты. К LSH для бинарных разреженных векторов относится LSH для $\mathrm{sim}_{\mathrm{Jac}}$. Недостатком ИС LSH является необходимость расчета и создания отдельной многотабличной ИС для каждой комбинации радиуса запроса и степени аппроксимации (чтобы выполнялись гарантии, эмпирически хорошие результаты могут давать ИС LSH без такого расчета). Кроме того, поиск допускает вероятность потери объектов, которые являются ответами на запрос (false negatives).

В результате усовершенствования LSH возможен поиск без потерь, но с гарантиями только для среднего случая. Развития LSH на основе LSF используют нетабличные структуры, позволяют добиться гибких компромиссов затрат времени и памяти и улучшения характеристик LSH для некоторых параметров. Варианты LSH, зависимые от данных, направлены на ускорения поиска для данных худшего случая.

В древовидных ИС (см. разд. 4) подмножествам объектов базы соответствуют иерархически организованные области, обычно не пересекающиеся на одном уровне иерархии. Последняя позволяет быстро строить большое количество малых областей ИС и определять принадлежность к ним объектов. При поиске ближайшего соседа спуском из корня в лист находят листовую область, которой принадлежит вектор-запрос. Объекты базы этой области обычно являются хорошими кандидатами на ближайших соседей (или непосредственно содержат точных соседей). Однако для гарантии точного поиска исследуют все другие листовые области, которые могут содержать точный ответ. Таких областей может быть много, особенно в многомерных пространствах, что замедляет поиск. Однако объекты в них обычно лишь незначительно улучшают (или вообще не изменяют) качество ответов. Поэтому для быстрого приближенного поиска ближайших соседей ограничивают количество просматриваемых областей (вплоть до одной, которой принадлежит запрос), а качество результатов улучшают, применяя множества ИС с различными разбиениями. При этом не имеется строгих гарантий качества результатов, но эмпирические результаты хорошие.

Общим недостатком ИС, рассмотренных в разд. 2–4, является необходимость использования нескольких ИС с соответствующими затратами памяти.

В ИС на основе графов соседства (см. разд. 5) для каждого объекта базы сохраняют список объектов его окрестности. Множества объектов окрестностей соседних объектов пересекаются. При выполнении запроса приближенного поиска ближайших соседей стартуют с некоторого объекта и переходят на еще непосещенный объект из его окрестности, наиболее близкий к объекту-запросу, пока такие объекты имеются. Рестарт поиска с других объектов является аналогом использования нескольких ИС. К преимуществам таких ИС относится высокая скорость и качество приближенного поиска, к недостаткам — сложность построения и отсутствие гарантий качества поиска.

Размерности векторов, с которыми на практике работают ИС, невелики. Так, для точного поиска плотных (бинарных) векторов они не превышают десятков или сотен, для приближенного поиска — тысячи, при радиусе запроса до четверти размерности, а обычно гораздо меньше. Поиск для разреженных (бинарных) векторов большой размерности выполняется, в основном, с помощью обратного индекса, LSH для $\mathrm{sim}_{\mathrm{Jac}}$ и $\mathrm{sim}_{\mathrm{BB}}$, а также ИС, рассмотренных в разд. 6.

Являясь моделью биологической памяти, нейросетевая распределенная автоассоциативная память (NAM, см. разд. 6) может рассматриваться и как ИС для поиска по сходству, значительно отличающаяся от традиционных ИС других типов. В данном случае не используются хранение отдельных векторов базы, разбиение базы на подмножества, двухэтапная стратегия фильтрации и уточнения при выполнении запроса. Поиск возвращает только один вектор, который восстанавливается на выходе ИС итеративно, но может не являться вектором базы.

В распределенных NAM N векторов базы распределенно запоминают в D^2 элементах памяти. Время выполнения запроса также пропорционально D^2. Поэтому, если запомнить в NAM $N >> D$ векторов с возможностью восстановления, появляются предпосылки сублинейного поиска. Такое запоминание достижимо для случайных разреженных бинарных векторов большой размерности. Необходимы дальнейшие исследования возможности ускорения поиска для некоторых NAM по сравнению с другими ИС для разреженных бинарных векторов. Перспективным направлением является использование неполносвязных сетей, что снижает затраты памяти и время поиска до линейного от размерности.

Для ряда NAM лучшие характеристики достигаются при $D \to \infty$, что отлично от поведения ИС, описанных в разд. 2–5. Вследствие этих различий перспективно исследование совместного использования NAM с другими ИС.

СПИСОК ЛИТЕРАТУРЫ

1. Manning C., Raghavan P., Schütze H. Introduction to information retrieval. New York: Cambridge University Press, 2008. 506 p.
2. 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.
3. Fouad M.M. Content-based search for image retrieval. *I.J. Image, Graphics and Signal Processing*. 2013. Vol. 5, N 11. P. 46–52.
4. Rachkovkij D.A. Distance-based index structures for fast similarity search. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 4. P. 636–658.
5. Heinly J., Dunn E., Frahm J.-M. Comparative evaluation of binary features. Proc. *ECCV'12*. 2012. P. 759–773.
6. Khalifa F.A., Semary N.A., El-Sayed H.M., Hadhoud M.M. Local detectors and descriptors for object class recognition. *International Journal of Intelligent Systems and Applications*. 2015. Vol. 7, N 10. P. 12–18.
7. Uchida Y. Local feature detectors, descriptors, and image representations: A survey. arXiv:1607.08368. 28 Jul 2016.
8. Rastegari M., Ordonez V., Redmon J., Farhadi A. Xnor-net: Imagenet classification using binary convolutional neural networks. *Proc. ECCV'16.* 2016. P. 525–542.
9. Hubara I., Courbariaux M., Soudry D., El-Yaniv R., Bengio Y. Binarized neural networks. *Proc. NIPS'16*. 2016. P. 4107–4115.
10. Tang W., Hua G., Wang L. How to train a compact binary neural network with high accuracy? *Proc. AAAI'17*. 2017. P. 2625–2631.
11. Kumar S., Desai J.V., Mukherjee S. Copy move forgery detection in contrast variant environment using binary DCT vectors. *I.J. Image, Graphics and Signal Processing*. 2015. Vol. 7, N 6. P. 38–44.
12. Faruqui M., Dyer C. Non-distributional word vector representations. *Proc. ACL-IJCNLP'15*. 2015. Vol. 2. P. 464–469.
13. Ren S., Cao X., Wei Y., Sun J. Face alignment at 3000 fps via regressing local binary features. *Proc. CVPR'14*. 2014. P. 1685–1692.
14. Pavlov D.N., Mannila H., Smyth P. Beyond independence: probabilistic models for query approximation on binary transaction data. *IEEE TKDE*. 2003. Vol. 15, N 6. P. 1409–1421.
15. Wang J., Shen H.T., Song J., Ji J. Hashing for similarity search: A survey. arXiv:1408.2927. 13 Aug 2014.
16. Rachkovskij D.A., Kussul E.M., Baidyk T.N. Building a world model with structure-sensitive sparse binary distributed representations. *BICA*. 2013. Vol. 3. P. 64–86.
17. Rachkovskij D.A. Binary vectors for fast distance and similarity estimation. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 1. P. 138–156.
18. Wang J., Liu W., Kumar S., Chang S.-F. Learning to hash for indexing big data: A survey. Proceedings of the IEEE. 2016. Vol. 104, N 1. P. 34–57.
19. 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
20. Rachkovskij D.A. Real-valued vectors for fast distance and similarity estimation. *Cybernetics and Systems Analysis*. 2016. Vol. 52, N 6. P. 967–988.
21. Gaede V., Gunther O. Multidimensional access methods. *ACM Comput. Surv*. 1998. Vol. 30, N 2. P. 170–231.
22. Bohm C., Berchtold S., Keim D.A. Searching in high-dimensional spaces: Index structures for improving the performance of multimedia databas. *ACM Comp. Surv*. 2001. Vol. 33, N 3. P. 322–373.
23. Samet H. Foundations of multidimensional and metric data structures. San Francisco: Morgan Kaufmann, 2006. 1024 p.
24. Haque I.S., Pande V.S., Walters W.P. Anatomy of high-performance 2d similarity calculations. *Journal of Chemical Information and Modeling*. 2011. Vol. 51, N 9. P. 2345–2351.
25. Donaldson R., Gupta A, Plan Y., Reimer T. Random mappings designed for commercial search engines. arXiv:1507.05929. 21 Jul 2015.
26. Brodal G., Gasieniec L. Approximate dictionary queries. *Proc. CPM'96.* 1996. P. 65–74.
27. Carter. L., Wegman M.N. Universal classes of hash functions. *Journal of Computer and System Sciences*. 1979. Vol. 18, N 2. P. 143–154.
28. Fredman M.L., Komlos J., Szemeredi E. Storing a sparse table with $O(1)$ worst case access time. *Journal of the ACM*. 1984. Vol. 31, N 3. P. 538–544.

29. Chegrane I., Belazzougui D. Simple, compact and robust approximate string dictionary. *J. Discrete Algorithms*. 2014. Vol. 28. P. 49–60.
30. Pagh R. Locality-sensitive hashing without false negatives. *Proc. SODA'16*. 2016. P. 1–9.
31. Andoni A., Indyk P. Nearest neighbors in high-dimensional spaces. In: Handbook of Discrete and Computational Geometry, 3rd edition. 2017. Chap. 43. P. 1133–1153.
32. Zobel J., Moffat A. Inverted files for text search engines. *ACM Comput. Surv*. 2006. Vol. 38, N 2. P. 6:1–6:56.
33. Rachkovskij D.A., Slipchenko S.V. Similarity-based retrieval with structure-sensitive sparse binary distributed representations. *Computational Intelligence*. 2012. Vol. 28, N 1. P. 106–129.
34. Ferdowsi S., Voloshynovskiy S., Kostadinov D., Holotyak T. Fast content identification in high-dimensional feature spaces using sparse ternary codes. *Proc. WIFS'16*. 2016. P. 1–6.
35. 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.
36. Tatti N., Mielikainen T., Gionis A., Mannila H. What is the dimension of your binary data? *Proc. ICDM'06*. 2006. P. 603–612.
37. Alman J., Williams R. Probabilistic polynomials and hamming nearest neighbors. *Proc. FOCS'15*. 2015. P.136–150.
38. Powers D.M.W. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Tech*. 2011. Vol. 2, N 1. P. 37–63.
39. Muja M., Lowe D.G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI*. 2014. Vol. 36, N 11. P. 2227–2240.
40. Yao A.C., Yao F.F. Dictionary look-up with one error. *Journal of Algorithms*. 1997. Vol. 25, N 1. P. 194–202.
41. Brodal G. S., Srinivasan V. Improved bounds for dictionary look-up with one error. *Information Processing Letters*. 2000. Vol. 75, N 1–2. P. 57–59.
42. Cole R., Gottlieb L.-A., Lewenstein M. Dictionary matching and indexing with errors and don't cares. *Proc. STOC'04*. 2004. P. 91–100.
43. Chan H.-L., Lam T.-W., Sung W.-K., Tam S.-L., Wong S.-S. A linear size index for approximate pattern matching. *Journal of Discrete Algorithms*. 2011. Vol. 9, N 4. P. 358–364.
44. Chan H., Lam T. W., Sung W., Tam S., Wong S. Compressed indexes for approximate string matching. *Algorithmica*. 2010. Vol. 58, N 2. P. 263–281.
45. Greene D., Parnas M., Yao F. Multi-index hashing for information retrieval. *Proc. FOCS'94*. 1994. P. 722–731.
46. Wu S., Manber U. Fast text searching allowing errors. *Communications of the ACM*. Vol. 35, N 10. 1992. P. 83–91.
47. Manku G.S., Jain A., Sarma A.D. Detecting near-duplicates for web crawling. *Proc. WWW'07*. 2007. P. 141–150.
48. Liu A.X., Ke S., Torng E. Large scale hamming distance query processing. *Proc. ICDE'11*. 2011. P. 553–564.
49. Gog S., Venturini R. Fast and compact Hamming distance index. *Proc. SIGIR'16*. 2016. P. 285–294.
50. Zhang X., Qin J., Wang W., Sun Y., Lu J. Hmsearch: An efficient hamming distance query processing algorithm. *Proc. SSDBM'13*. 2013. P. 19:1–19:12.
51. 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.
52. Wan J., Tang S., Zhang Y., Huang L., Li J. Data driven multi-index hashing. *Proc. ICIP'13*. 2013. P. 2670–2673.
53. Ma Y., Zou H., Xie H., Su Q. Fast search with data-oriented multi-index hashing for multimedia data. *KSII TIIS*. 2015. Vol. 9, N 7. P. 2599–2613.
54. Wang M., Feng X., Cui J. Multi-index hashing with repeat-bits in hamming space. *Proc. FSKD'15*. 2015. P. 1307–1313.
55. Song J., Shen H.T., Wang J., Huang Z., Sebe N., Wang J. A distance-computation-free search scheme for binary code databases. *IEEE Trans. Multimedia*. 2016. Vol. 18, N 3. P. 484–495.
56. Ong E.-J., Bober M. Improved Hamming distance search using variable length hashing. *Proc. CVPR'16*. 2016. P. 2000–2008.
57. Eghbali S., Tahvildari L. Cosine similarity search with multi-index hashing. arXiv:1610.00574. 14 Sep 2016.
58. 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.
59. Har-Peled S., Indyk P., Motwani R. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput*. 2012. Vol. 8. P. 321–350.
60. Shrivastava A., Li P. Asymmetric LSH (ALSH) for sublinear time maximum inner product search (MIPS). *Proc. NIPS'14*. 2014. P. 2321–2329.

61. Charikar M. Similarity estimation techniques from rounding algorithms. *Proc. STOC'02*. 2002. P. 380–388.
62. Shrivastava A., Li P. Asymmetric minwise hashing for indexing binary inner products and set containment. *Proc. WWW'15*. 2015. P. 981–991.
63. Andoni A., Datar M., Immorlica N., Indyk P., Mirrokni V. S. Locality-sensitive hashing using stable distributions. P. 61–72. In: Nearest Neighbor Methods for Learning and Vision: Theory and Practice. MIT Press, Cambridge. 2006. 280 p.
64. 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.
65. Broder A.Z. On the resemblance and containment of documents. *Proc. SEQUENCES'97*. 1997. P. 21–29.
66. Broder A.Z., Glassman S.C., Manasse M. S., Zweig G. Syntactic clustering of the web. *Computer Networks and ISDN Systems*. 1997. Vol. 29, N 8–13. P. 1157–1166.
67. Broder A.Z., Charikar M., Frieze A.M., Mitzenmacher M. Min-wise independent permutations. *J. Comput. System Sci*. 1998. Vol. 60. P. 327–336.
68. Tang J., Tian Y. A systematic review on minwise hashing algorithms. *Annals of Data Science*. 2016. Vol. 3, N 4. P. 445–468.
69. Dahlgaard S., Knudsen M.B.T, Thorup M. Fast similarity sketching. arXiv:1704.04370. 14 Apr 2017.
70. Li P., König A.C. Theory and applications of b-bit minwise hashing. *Communications of the ACM*. 2011. Vol. 54, N 8. P. 101–109.
71. Shrivastava A. Optimal densification for fast and accurate minwise hashing. arXiv:1703.04664. 14 Mar 2017.
72. Shrivastava A., Li P. In defense of minhash over simhash. *Proc. AISTATS'14*. 2014. P. 886–894.
73. Ahle T.D., Pagh R., Razenshteyn I., Silvestri F. On the complexity of inner product similarity join. *Proc. PODS'16*. 2016. P. 151–164.
74. Bera D., Pratap R.. Frequent-itemset mining using locality-sensitive hashing. *Proc. COCOON'16*. 2016. P. 143–155.
75. Trzcinski T., Lepetit V., Fua P. Thick boundaries in binary space and their influence on nearest-neighbor search. *Pattern Recognition Letters*. 2012. Vol. 33, N 16. P. 2173–2180.
76. Esmaeili M.M., Ward R.K., Fatourechi M. A fast approximate nearest neighbor search algorithm in the Hamming space. *IEEE Trans. PAMI*. 2012. Vol. 34, N 12. P. 2481–2488.
77. Har-Peled S., Mahabadi S. Proximity in the age of distraction: Robust approximate nearest neighbor search. *Proc. SODA'17*. 2017. P. 1–15.
78. Ahle T.D., Aumuller M., Pagh R. Parameter-free locality sensitive hashing for spherical range reporting. *Proc. SODA'17*. 2017. P. 239–256.
79. Pham N. Hybrid LSH: Faster near neighbors reporting in high-dimensional space. *Proc. EDBT'17*. 2017. P. 454–457.
80. 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.
81. Pham N., Pagh R. Scalability and total recall with fast CoveringLSH. *Proc. CIKM'16*. 2016. P. 1109–1118.
82. 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.
83. 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.
84. Christiani T., Pagh R. Set similarity search beyond MinHash. *Proc. STOC'17*. 2017. P. 1094–1107.
85. Ahle T.D. Optimal Las Vegas locality sensitive data structures. arXiv:1704.02054. April 6 2017.
86. Andoni A., Razenshteyn I. Optimal data-dependent hashing for approximate near neighbors. *Proc. STOC'15*. 2015. P. 793–801.
87. Andoni A., Razenshteyn I., Shekel Nosatzki N. Lsh forest: Practical algorithms made theoretical. *Proc. SODA'17*. 2017. P. 67–78.
88. Bawa M., Condie T., Ganesan P. Lsh forest: self-tuning indexes for similarity search. *Proc. WWW'05*. 2005. P. 651–660.
89. Qian G., Zhu Q., Xue Q., Pramanik S. Dynamic indexing for multidimensional non-ordered discrete data spaces using a data-partitioning approach. *ACM TODS*. 2006. Vol. 31, N 2. P. 439–484.
90. Qian G., Zhu Q., Xue Q., Pramanik S. A space-partitioning-based indexing method for multidimensional non-ordered discrete data spaces. *ACM TOIS*. 2006. Vol. 23. P. 79–110.
91. Yan C.C., Xie H., Zhang B., Ma Y., Dai Q., Liu Y. Fast approximate matching of binary codes with distinctive bits. *Front. Comput. Sci*. 2015. Vol. 9, N 5. P. 741–750.
92. Galvez-Lopez D., Tardos J.D. Bags of binary words for fast place recognition in image sequences. *IEEE Trans. Robotics*. 2012. Vol. 28, N 5. P. 1188–1197.
93. Luo Q., Zhang S., Huang T., Gao W., Tian Q. Scalable mobile search with binary phrase. *Proc. ICIMCS'13*. 2013. P. 66–70.
94. Niedermayer J., Kroger P. Retrieval of binary features in image databases: A study. *Proc. SISAP'14*. 2014. P. 151–163.

95. Kryzhanovsky V., Malsagov M., Tomas J.A.C., Zhelavskaya I. On error probability of search in high-dimensional binary space with scalar neural network tree. *Proc. NCTA'14*. 2014.
96. Tang M., Yu Y., Aref W.G., Malluhi Q.M., Ouzzani M. Efficient processing of hamming-distance-based similarity-search queries over mapreduce. *Proc. EDBT'15*. 2015. P. 361–372.
97. Tao Y., Yi K., Sheng C., Kalnis P. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. *ACM Trans. Database Syst*. 2010. Vol. 35, N 3. P. 20:1–20:46.
98. Jiang Z., Xie L., Deng X., Xu W., Wang J. Fast nearest neighbor search in the hamming space. *Proc. MMM'16*. 2016. P. 325–336.
99. Malkov Yu. A., Yashunin D. A. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. arXiv:1603.09320. 21 May 2016.
100. 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.
101. Valiant L.G. Functionality in neural nets. Proc. AAAI'88. 1988. Vol. 2. P. 629–634.
102. Hopfield J.J. Neural networks and physical systems with emergent collective computational abilities. *Proc. of the Nat. Acad. Sci. USA*. 1982. Vol. 79, N 8. P. 2554–2558.
103. Tsodyks M., Feigelman M. The enhanced storage capacity in neural networks with low activity level. *Europhysics Letters.* 1988. Vol. 6, N 2. P. 101–105.
104. Frolov A.A., Husek D., Muraviev I.P. Information capacity and recall quality in sparsely encoded Hopfield-like neural network: Analytical approaches and computer simulation. *Neural Networks*. 1997. Vol. 10, N 5. P. 845–855.
105. Frolov A.A., Husek D., Muraviev I.P. Informational efficiency of sparsely encoded Hopfield-like associative memory. *Optical Memory & Neural Networks*. 2003. Vol. 12, N 3. P. 177–197.
106. Amari S. Characteristics of sparsely encoded associative memory. *Neural Networks*. 1989. Vol. 2, N 6. P. 451–457.
107. Heusel J., Lowe M., Vermet F. On the capacity of an associative memory model based on neural cliques. *Statist. Probab. Lett*. 2015. Vol. 106. P. 256–261.
108. Gripon V., Heusel J., Lowe M., Vermet F. A comparative study of sparse associative memories. *Journal of Statistical Physics*. 2016. Vol. 164. P. 105–129.
109. Frolov A.A., Husek D., Rachkovskij D.A. Time of searching for similar binary vectors in associative memory. *Cybernetics and Systems Analysis*. 2006. Vol. 42, N 5. P. 615–623.
110. Palm G. On associative memory. *Biological Cybernetics.* 1980. Vol. 36. P. 19–31.
111. Tsodyks M.V. Associative memory in neural networks with binary synapses. *Mod. Phys. Lett*. 1990. Vol. B4. P. 713–716.
112. Frolov A., Kartashov A., Goltsev A., Folk R. Quality and efficiency of retrieval for Willshaw-like autoassociative networks. Correction. *Network*. 1995. Vol. 6. P. 513–534.
113. Schwenker F., Sommer F.T., Palm G. Iterative retrieval of sparsely coded associative memory patterns. *Neural Networks*. 1996. Vol. 9. P. 445-455.
114. Frolov A.A., Rachkovskij D.A., Husek D. On information characteristics of Willshaw-like auto-associative memory. *Neural Network World*. 2002. Vol. 12, N 2. P. 141–157.
115. Kanter I. Potts-glass models of neural networks. *Physical Rev. A*. 1988. V. 37, N 7. P. 2739–2742.
116. Lowe M., Vermet F. The capacity of q-state Potts neural networks with parallel retrieval dynamics. *Statistics and Probability Letters*. 2007. Vol. 77, N 4. P. 1505–1514.
117. Onizawa N., Jarollahi H., Hanyu T., Gross W.J. Hardware implementation of associative memories based on multiple-valued sparse clustered networks. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*. 2016. Vol. 6, N 1. P. 13–24.
118. Kartashov A., Frolov A., Goltsev A., Folk R. Quality and efficiency of retrieval for Willshaw-like autoassociative networks. Willshaw–Potts model. *Network*. 1997. Vol. 8, N 1. P. 71–86.
119. Gripon V., Berrou C. Sparse neural networks with large learning diversity. *IEEE Trans. on Neural Networks.* 2011. Vol. 22, N 7. P. 1087–1096.
120. Tsodyks M. Associative memory in asymmetric diluted network with low level of activity. *Europhysics Letters.* 1988. Vol. 7, N 3. 203–208.
121. Buckingham J., Willshaw D. On setting unit thresholds in an incompletely connected associative net. *Network*. 1993. Vol. 4. P. 441–459.
122. Yu C., Gripon V., Jiang X., Jegou H. Neural associative memories as accelerators for binary vector search. *Proc. COGNITIVE'15.* 2015. P. 85–89.
123. Frolov A.A., Husek D., Muraviev I.P., Polyakov P. Boolean factor analysis by attractor neural network. *IEEE Trans. Neural Networks*. 2007. Vol. 18, N 3. P. 698–707.
124. Peretto P., Niez J.J. Long term memory storage capacity of multiconnected neural networks. *Biol. Cybern*. 1986. Vol. 54, N 1. P. 53–63.
125. Baldi P., Venkatesh S.S. Number of stable points for spin-glasses and neural networks of higher orders. *Physical Review Letters*. 1987. Vol. 58, N 9. P. 913–916.
126. Krotov D., Hopfield J.J. Dense associative memory for pattern recognition. *Proc. NIPS'16*. 2016. P. 1172–1180.

127. Krotov D., Hopfield J. Dense associative memory is robust to adversarial inputs. arXiv:1701.00939. 4 Jan 2017.
128. Demircigil M., Heusel J., Lowe M., Upgang S., Vermet F. On a model of associative memory with huge storage capacity. *Journal Stat. Phys*. 2017. Vol. 168, N 2. P. 288–299.
129. Karbasi A., Salavati A. H., Shokrollahi A. Iterative learning and denoising in convolutional neural associative memories. *Proc. ICML'13*. 2013. P. 445–453.
130. Salavati A.H., Kumar K.R., Shokrollahi A. Nonbinary associative memory with exponential pattern retrieval capacity and iterative learning. *IEEE Trans. Neural Networks and Learning Systems.* 2014. Vol. 25, N 3. P. 557–570.
131. Mazumdar A., Rawat A.S. Associative memory via a sparse recovery model. *Proc. NIPS'15.* 2015. P. 2683–2691.
132. Mazumdar A., Rawat A.S. Associative memory using dictionary learning and expander decoding. *Proc. AAAI'17*. 2017. P. 267–273.
133. Mansor M.A., Kasihmuddin M.S.M., Sathasivam S. VLSI circuit configuration using satisfiability logic in Hopfield network. *International Journal of Intelligent Systems and Applications (IJISA)*. 2016. Vol. 8, N 9. P. 22–29.
134. Mansor M.A., Kasihmuddin M.S.M., Sathasivam S. Enhanced hopfield network for pattern satisfiability optimization. *International Journal of Intelligent Systems and Applications (IJISA)*. 2016. Vol. 8, N 11. P. 27–33.

Надійшла до редакції 07.02.2017

Д.А. Рачковський
ІНДЕКСНІ СТРУКТУРИ ДЛЯ ШВИДКОГО ПОШУКУ ЗА СХОЖІСТЮ БІНАРНИХ ВЕКТОРІВ

Анотація. Наведено огляд індексних структур для швидкого пошуку за схожістю об'єктів, що представлені бінарними векторами (із компонентами 0 або 1). Розглянуто структури як для точного, так і для наближеного пошуку за відстанню Хеммінга та іншими мірами схожості. Описано, головним чином, індексні структури на основі хеш-таблиць, хешування, що зберігає схожість, а також деревовидних структур, графів сусідства та нейромережевої розподіленої автоасоціативної пам'яті. Викладено ідеї конкретних алгоритмів (відомих та нещодавно запропонованих).

Ключові слова: пошук за схожістю, відстань Хеммінга, найближчий сусід, ближній сусід, індексні структури, мультиіндексне хешування, локально-чутливе хешування, деревовидні структури, граф сусідства, нейромережева автоасоціативна пам'ять.

D.A. Rachkovskij
INDEX STRUCTURES FOR FAST SIMILARITY SEARCH OF BINARY VECTORS

Abstract. We survey index structures for fast similarity search of objects represented by binary vectors (with components 0 or 1). Structures for both exact and approximate search by Hamming distance and other similarity measures are considered. Mainly, we present index structures based on hash tables, similarity-preserving hashing, as well as tree structures, neighborhood graphs, and neural distributed autoassociative memory. The ideas of specific algorithms, including the recently proposed ones, are outlined.

Keywords: similarity search, Hamming distance, nearest neighbor, near neighbor, index structures, multi-index hashing, locally-sensitive hashing, treelike structures, neighborhood graph, neural autoassociative memory.

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.