УДК 004.22+004.93′11

Д.А. РАЧКОВСКИЙ

ИНДЕКСНЫЕ СТРУКТУРЫ ДЛЯ БЫСТРОГО ПОИСКА ПО СХОДСТВУ ВЕЩЕСТВЕННЫХ ВЕКТОРОВ. II[1]

Аннотация. Дан обзор индексных структур для быстрого поиска по сходству объектов, представленных вещественными векторами. Рассмотрены структуры как для точного, так и для более быстрого, но приближенного поиска. Представлены главным образом индексные структуры на основе разбиения на области (в том числе иерархические) и графов соседства. Обсуждено также ускорение поиска по сходству с использованием преобразования исходных данных. Изложены идеи конкретных алгоритмов, включая недавно предложенные. Проведено сравнение подходов к ускорению поиска по сходству в индексных структурах рассмотренных типов, а также на основе сохраняющего сходство хэширования.

Ключевые слова: поиск по сходству, ближайший сосед, ближний сосед, индексные структуры, метод ветвей и границ, деревья и леса, кластеризация, граф соседства, локально-чувствительное хэширование.

ВВЕДЕНИЕ

Индексные структуры (ИС) для быстрого поиска по сходству вещественных векторов описаны в [1–8]. (Введение в проблематику поиска по сходству также см. в [1–8].) В отличие от [2–4] в настоящей статье представлены недавно появившиеся ИС для вещественных векторов, а также современные усовершенствования известных.

Основными запросами поиска по сходству являются: диапазонный запрос rNN, возвращающий объекты базы, расстояния которых от объекта-запроса (по мере расстояния, заданной в запросе) не превышают радиуса запроса r, а также запрос ближайшего соседа NN, возвращающий объект базы, ближайший к объекту-запросу (этот объект также обозначим NN). Запрос k ближайших соседей обозначим kNN. Выполнение запросов линейным поиском по сходству заключается в вычислении расстояний/сходств объекта-запроса до всех объектов базы и в возвращении объектов, удовлетворяющих условиям запроса. Сложность (время) вычисления расстояния/сходства между двумя векторами размерности D составляет $O(D)$ для многих мер, таких как косинусное сходство [1] или расстояние L_s (например, для манхэттенова расстояния L_1 ($||\mathbf{a}-\mathbf{b}||_1$) или для евклидова расстояния L_2 ($||\mathbf{a}-\mathbf{b}||_2$)). Поэтому время линейного поиска по сходству в базе из N объектов-векторов составляет $O(ND)$, что зачастую недопустимо медленно для больших N (и D). Для ускорения поиска по базе строят ИС и затем используют ее при выполнении запросов. Во многих ИС при выполнении запроса применяют процедуры, которые можно считать вариантами двухэтапной стратегии фильтрации и уточнения F&R. На первом этапе осуществляется быстрый отбор объектов-кандидатов. Результаты первого этапа уточняются на втором этапе (обычно линейным поиском среди объектов-кандидатов по мере расстояния/сходства, заданной в запросе). Если суммарное количество объектов-кандидатов меньше N и их отбор выполняется достаточно быстро, можно получить ускорение выполнения запроса относительно линейного поиска (т.е. сублинейный от N поиск).

Для точного поиска при выполнении запроса кандидаты выбираются так, что они гарантированно содержат ответ на запрос. Многие ИС точного поиска по сходству используют разновидности метода ветвей и границ B&B (см. обзор [7] и разд. 1). При выполнении запроса запоминается текущая верхняя граница рас-

[1] Окончание. Начало см. в № 1, 2018.

стояния от объекта-запроса до объектов базы, которые могут являться ответом на запрос (граница решения). Для запроса rNN это r, а для запроса NN — расстояние до текущего ближайшего соседа. Кроме того, определяются границы расстояний от объекта-запроса до еще необработанных подмножеств объектов ИС. Если нижняя граница для некоторого подмножества больше, чем текущая граница решения, подмножество исключают, иначе переходят к его обработке. Для неисключенных объектов-кандидатов вычисляют расстояние до объекта-запроса. Такую процедуру можно выполнять многократно. Конкретные алгоритмы B&B используют различные способы разбиения базы на подмножества, очередности обхода подмножеств, определения границ.

К сожалению, сублинейное время выполнения запросов точного поиска не гарантируется и не наблюдается на практике как для ИС типа B&B, так и для других ИС. Анализ всех известных алгоритмов точного выполнения запроса NN с сублинейным от N временем показывает, что затраты времени или памяти растут экспоненциально от D [9]. Согласно предположению «проклятия размерности» [9] 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 ускорения поиска по сходству методом B&B процедуру выполнения запроса модифицируют так, что в множество кандидатов включается только часть тех объектов базы, которые являются кандидатами при точном поиске. При этом можно потерять объекты-ответы на запрос, т.е. поиск становится приближенным. Например, ранний останов [10] прекращает поиск до гарантированного достижения точного результата. Критерием останова может быть количество обработанных объектов базы. Для повышения качества результатов приближенного поиска ИС используют алгоритмы обхода подмножеств/областей, которые в первую очередь посещают подмножества, ближайшие к объекту-запросу (например, обход best-first, см. разд. 1).

Многие ИС не имеют количественных гарантий отличия результатов от результатов точного поиска. Для оценки качества результатов и времени поиска используют эксперименты на конкретных базах. Часто применяют меры качества выполнения конкретного запроса, известные (см. обзор [1] и ссылки к нему) как полнота (recall), равная $n1 / n2$, и точность (precision), равная $n1 / n3$, где $n1$ — количество возвращенных объектов, совпавших с релевантными запросу объектами базы, $n2$ — количество релевантных запросу объектов базы, $n3$ — количество возвращенных объектов. Для запроса kNN $n2 = n3 = k$, поэтому точность равна полноте. В качестве релевантных объектов используют объекты, возвращаемые запросами точного поиска либо указанные экспертом. При выполнении множества запросов значения мер качества индивидуальных запросов усредняют. Так, для запроса NN полноту (равную точности) измеряют как процент запросов, для которых был возвращен эталонный NN [11].

Кроме ИС на основе областей и вариантов B&B (разд. 1), в обзоре рассмотрены ИС, использующие преобразования исходных векторов (разд. 2). Некоторые ИС, описанные в разд. 1 и 2, выполняют точный поиск (без гарантий сублинейного времени), другие — приближенный (без гарантий качества результатов). Разд. 3 посвящен ИС для быстрого приближенного поиска на основе графов соседства. В заключении обсуждаются и сравниваются подходы к ускорению поиска по сходству в ИС, представленных в настоящей статье, а также в ИС на основе сохраняющего сходство хэширования, которые имеют количественные гарантии сублинейного времени для некоторых типов запросов и рассматривались в [1] (см. также обзоры [9, 6] и ссылки к ним).

1. ИНДЕКСНЫЕ СТРУКТУРЫ С ЯВНЫМ ИСПОЛЬЗОВАНИЕМ ОБЛАСТЕЙ

Для ускорения построения и использования ИС разбиение базы на множества объектов в ряде случаев проводят иерархически: полученные в результате разбиения множества рекурсивно разбивают на более мелкие подмножества. Часто

иерархические ИС для поиска по сходству имеют древовидную структуру. В дереве узел-родитель соединен ребрами с узлами-сыновьями. Количество сыновей узла называют коэффициентом ветвления (степенью). Корневой (верхний) узел не имеет родителя. У всех остальных узлов только один родитель. Листовые узлы не имеют сыновей. Уровень узла есть количество ребер до корня.

Подмножеству объектов базы (и их области) обычно соответствует узел дерева, который содержит информацию о том, относится ли объект к подмножеству/области узла. Объекты базы или ссылки на них содержатся в корзинах ИС. В деревьях корзинами часто являются листья. Листовой корзине может принадлежать один или более объектов базы. В сбалансированных деревьях все пути от корня к листьям отличаются не более чем на 1. Для бинарного дерева (коэффициент ветвления 2), которое разбивает объекты каждого узла на два сбалансированных по количеству объектов непересекающихся подмножества, высота дерева близка к $\log N$. Затраты памяти на такую ИС составляют $O(N)$.

При выполнении запроса NN на основе метода B&B порядок обхода областей существенно влияет на время поиска. В древовидных ИС вначале выполняют спуск по дереву из корневого узла в листовой (за время $O(\log N)$ для сбалансированных деревьев). Для перехода в каждом узле выбирают узел-сына (область) с минимальным расстоянием от объекта-запроса (обычно область, которой принадлежит объект-запрос). В листовой корзине линейным поиском находят ближайший объект и используют в качестве текущего NN. Обычно найденный объект является хорошим кандидатом, однако точный NN может находиться в других корзинах. Поэтому продолжают обход узлов дерева в порядке, который зависит от алгоритма обхода.

При обходе depth-first [3, 4] далее выполняют обратное прослеживание. Из посещенного листового узла возвращаются в узел-родитель и спускаются в другие его листовые узлы, если их области пересекают текущую область запроса. Текущий радиус модифицируют, когда встречают более близкий к запросу объект базы. По исчерпании листьев текущего узла выполняется возврат к его родительскому узлу уровнем выше и рекурсивно продолжается обход непосещенных узлов-сыновей, области которых пересекают текущую область запроса.

При обходе best-first (или priority search) [3, 4, 12] вычисленные при первоначальном спуске из корня в лист расстояния до узлов (их областей) различных уровней заносят в приоритетную очередь с сортировкой по увеличению. Для дальнейшего обхода из приоритетной очереди извлекают первый узел и обходят его (непосещенные) узлы-сыновья, области которых пересекают текущую область запроса, добавляя в очередь узлы с их расстояниями и модифицируя текущий радиус запроса. Это аналог обратного прослеживания при обходе depth-first. Для точного поиска обход прекращают, когда расстояние до первого узла очереди превысит текущий радиус запроса. При приближенном поиске best-first позволяет быстро находить «хороших» соседей и останавливать поиск раньше, чем при точном.

Время выполнения запроса точного NN для сбалансированных деревьев в среднем (для данных, случайно независимо сгенерированных, например из равномерного распределения) составляет $O(\log N)$ при постоянном D [13]. Время для худшего случая составляет $O(N)$, так как обратное прослеживание может посетить все объекты базы. Даже для среднего случая наблюдается экспоненциальная зависимость времени от D (для $D << \log N$). Однако время выполнения запроса не превышает $O(ND)$ (узел посещают не более одного раза). Для некоторых типов ИС время поиска зависит не от внешней D, а от внутренней размерности (обзор [7] и ссылки к нему) данных.

Эксперименты показывают, что поиск best-first обычно быстро находит точных соседей (или хорошее приближение), а большая часть времени поиска идет на проверку этого обратным прослеживанием [15, 16]. Это позволяет ускорить поиск ограничением (или даже отказом от применения) обратного прослеживания ценой потери гарантий точности поиска.

ИС, использующие разбиение на неиерархические области, рассмотрены в подразд. 1.5. Их можно считать вырожденными деревьями (с одним уровнем).

1.1. Классические древовидные индексные структуры и их модификации. Рассмотрим семейства классических древовидных ИС, изначально предложенных для быстрого точного поиска по сходству для векторов невысокой размерности. К семейству ИС, разбивающих пространство, относится KD-tree: оно исчерпывающе разбивает пространство на непересекающиеся области на каждом уровне иерархии, и любой объект принадлежит только одной области некоторого уровня иерархии. Недостатком является большой объем областей вследствие покрытия пустых участков пространства. К семейству ИС, разбивающих данные, относится R-tree: объекты помещают в области так, чтобы минимизировать их объем, причем области даже одного уровня иерархии могут пересекаться, хотя объект принадлежит только одной области.

1.1.1. KD-деревья. KD-tree [13] иерархически конструирует гиперпрямоугольные области, выбирая одно из измерений пространства (компонент вектора) и задавая в нем пороговое значение. Измерения выбирают по очереди либо используют измерение с максимальной дисперсией значений у разбиваемого подмножества объектов базы. Медианное значение порога обеспечивает равные мощности двух результирующих подмножеств. Разбиение (под)множеств рекурсивно повторяют до тех пор, пока количество объектов в подмножествах не уменьшится до заданного. В результате каждой области (и объектам в ней) соответствует узел бинарного сбалансированного (по количеству уровней и объектов в узле одного уровня) дерева. Объекты хранят только в листьях [13] (в прежних вариантах их хранили и во внутренних узлах). Высота дерева $O(\log N)$, затраты памяти $O(N)$, время построения $O(N \log N)$. Если $N < 2^D$, данных недостаточно для использования при разбиении всех измерений.

Обработка различных типов запросов для расстояний L_s (главным образом L_2) осуществляется одним и тем же KD-tree. Запрос rNN выполняется спуском по дереву с посещением только тех узлов, области которых пересекают область запроса. Для запроса kNN в классическом KD-tree используют обход depth-first с обратным прослеживанием. (Время запроса обсуждалось во вводной части данного раздела.) KD-tree обычно показывает лучшие результаты, чем линейный поиск, при D, не превышающим 10–20 [14]. Для гиперпрямоугольного запроса (типа L_∞) сложность поиска в худшем случае оценивают как $O(DN^{1-1/D})$ [17], что также требует $D = O(1)$ для существенно сублинейного времени.

Запрос приближенного NN, возвращающий объект базы, расстояние которого до объекта-запроса не более чем в $c > 1$ раз превышает расстояние до точного NN, называют c-NN. Модификацией KD-tree для c-NN является BBD-tree [12], где используют обход best-first и агрессивную обрезку (уменьшают текущий радиус запроса для исключения объектов дополнительно к исключаемым точным поиском). Хотя эмпирически получен быстрый поиск на многих базах, время поиска в худшем случае остается экспоненциальным от D: $O(D(1+6/(c-1))^D \log N)$. Другие модификации KD-tree рассматриваются далее.

1.1.2. R-деревья и их разновидности. В R-tree [18] объекты в каждом узле заключают в минимальные ограничивающие гиперпрямоугольники (MBR). Листовому узлу соответствует один объект и его MBR. Для узлов промежуточных уровней строят MBR вокруг MBR нижележащего уровня, вплоть до максимального коэффициента ветвления M. Конструируют R-tree многократной операцией вставки объекта. Вставку выполняют в листовой узел, в который попадают спуском из корня через узлы, MBR которых требует для этого наименьшего увеличения. При переполнении емкости листового узла его MBR расщепляется на два, что может инициировать процесс расщепления MBR узлов-предков и перестрое-

ние всего дерева. Запрос NN (для различных L_s) может выполняться обходом depth-first с обратным прослеживанием. Сложность операций вставки и поиска $O(\log_M N)$ в среднем и $O(N)$ в худшем случае. Экспоненциальная зависимость среднего времени поиска определяется внутренней размерностью данных (анализ для L_∞, L_2 приведен в [19]).

Для оптимизации использования памяти и времени поиска предложены разновидности R-tree, а именно R+-tree, R*-tree, X-tree и др. (см. обзоры [2, 3] и ссылки к ним). Вариантами R*-tree считают деревья с иерархическим разбиением пространства ограничивающими сферами: SS-tree [20] и Ball-tree [21]. Для улучшения исключения в многомерных пространствах SR-tree [22] использует более компактные области в виде пересечения ограничивающей сферы и MBR. Большей оптимизации добиваются массовой загрузкой (bulk loading) всей базы [23].

1.2. Рандомизированные деревья. Некоторые древовидные ИС с рандомизированными разбиениями пространства позволяют [24]: избежать проблемы с «патологическими» конфигурациями данных, которые приводят к линейному поиску в KD-tree; аналитически оценить время поиска для некоторых данных; построить множества ИС (леса деревьев) для быстрого приближенного поиска по сходству (подразд. 1.4).

1.2.1. Рандомизированные KD-деревья. Одно из первых рандомизированных KD-tree (для L_2) предложено в [25]. Объекты базы — векторы, случайно равномерно распределенные в $[-1,1]^D$. При построении ИС векторы базы проецируют на случайные ортонормальные векторы (один и тот же вектор во всех узлах одного уровня), порог разбиения устанавливают вблизи медианного значения проекций множества объектов узла. Затраты памяти $O(N)$. При поиске переход в узле может выполняться на одну или обе ветви в зависимости от результата сравнения значения проекции вектора-запроса с двумя пороговыми значениями. Поэтому спуск может привести в несколько листовых узлов. Обратного прослеживания не применяют. Ближайший к запросу объект посещенных листовых узлов является ответом на запрос. Подобный поиск без обратного прослеживания назван defeatist search [16]. Обрезка является агрессивной, так как допускает потерю точного NN с (регулируемой) вероятностью δ. Расстояние $bD^{1/2}$ от объекта-запроса до NN ограничено средним расстоянием между случайными объектами. Это позволило получить время запроса для среднего случая DN^{α}, $\alpha < 1$ и зависит от b, δ. Также в [16] предложено использовать для повышения вероятности нахождения точного NN перекрывающиеся области разбиения (подразд. 1.2.3) и леса деревьев (подразд. 1.4).

Для создания (в значительной степени) независимых KD-деревьев в [26] рассматривают: случайный выбор очередного измерения для разбиения из измерений с наибольшей дисперсией (RKD-tree), а для поиска по L_2 — построение дерева для случайно повернутых векторов базы (NKD-tree) с соответствующим поворотом вектора объекта-запроса, включая вариант с предобработкой данных базы посредством анализа главных компонент PCA.

1.2.2. Деревья случайных проекций. В random projection tree (RP-tree) [27] для разбиений используют плоскости, перпендикулярные случайным направлениям (в каждом узле свое направление). Вычисление проекций данных на эти направления можно рассматривать как случайное проецирование. (Обзоры общей проблематики случайного проецирования см. в [5, 6], нейробиологически релевантные варианты исследуют в [28–31].) Так, для L_2 применяют гауссовы случайные векторы (или случайные векторы с S^{D-1}). Плоскость случайно смещают относительно медианного значения проекций объектов. В листьях содержится не по одному, а по множеству объектов. Разбиение RP-tree дает более гибкие и компактные разбиения, чем KD-tree. Для RP-tree диаметр областей (наименьший диаметр сфер, включающих области) уменьшается в зависимости от количества

разбиений (уровней дерева) со скоростью, зависящей от внутренней размерности данных, а не от внешней, как для KD-tree [27].

В [24] показано, что для данных с размерностью расширения D_e defeatist search находит NN за время $O(D_e \log D_e)^{D_e} + O(\log N)$ с произвольно малой вероятностью ошибки. Анализ гарантий RP-tree для поиска по расстоянию L_1, где случайное проецирование использует распределения Коши, проведен в [32]. Отметим, что если применять обратное прослеживание, то для RP-tree можно получить точный поиск (но без гарантий времени).

1.2.3. Размытое дерево. Spill-tree [16] — дерево с перекрывающимися областями. При построении spill-tree в узле выполняется разбиение по двум пороговым значениям, сдвинутым относительно медианы проекции объектов узла на его случайное направление. В результате часть объектов может принадлежать обеим областям узла (и нескольким листовым узлам) и затраты памяти растут от N быстрее, чем линейно. В листьях содержится по множеству объектов. Поиск выполняется без обратного прослеживания. Переход осуществляется на один узел-сын по результату сравнения проекции объекта-запроса с медианным пороговым значением. Предлагается также снижать размерность векторов случайным проецированием.

В virtual spill-tree [24] области не перекрываются, но при поиске пороги выбирают так же, как при конструировании spill-дерева, т.е. в узле возможен переход в две области. Оба варианта spill-tree возвращают точного NN, но с некоторой вероятностью. В [24] показано, что время поиска NN (по L_2) составляет $O(D_e)^{D_e} + O(\log N)$ с произвольно малой вероятностью ошибки.

Недостатком рандомизированных деревьев по сравнению с KD-tree является необходимость вычисления проекции (плотных векторов) скалярным произведением вместо сравнения значений одного компонента. Для уменьшения сложности проецирования TP-tree [33] использует векторы с малым числом ненулевых компонентов из $\{-1, 0, +1\}$. В [34] показано, что так же быстро, как в RP-tree, диаметр областей уменьшается для дерева со случайным ортогональным вращением данных NKD-tree [26]. При этом для NKD-tree можно применять простой алгоритм обхода с обратным прослеживанием обычного KD-tree. Однако взаимосвязь скорости уменьшения диаметра областей со временем поиска не известна.

1.3. Древовидные индексные структуры с адаптацией к данным. В рассмотренных деревьях направления разбиений выбирались независимо от данных. Адаптация к данным сводилась к выбору порога разбиения. Бóльшая степень адаптации к особенностям распределения объектов в базе позволяет получить более гибкое и компактное разбиение, что улучшает исключение и ускоряет поиск, особенно для данных с низкой внутренней размерностью.

1.3.1. Деревья K-средних. Области Вороного вокруг центров кластеров являются компактными областями с кусочно-линейными границами. Часто используют кластеризацию K-means (например, [35–39]). Результат практического алгоритма K-means зависит от выбора начальных условий (центров), поэтому такая кластеризация является рандомизированной.

Уменьшить время разбиения базы на кластеры и время поиска ближайшего кластера позволяет иерархическая кластеризация (неиерархичекие ИС см. в подразд. 1.5). В [40] предложена ИС, в которой множество объектов рекурсивно разбивается на небольшое количество подмножеств кластеризацией K-means. Однако для упрощения исключения областей объекты кластеров помещали в сферические области, что нивелировало преимущества K-means.

Исследование непосредственного использования иерархического дерева K-means (KM-tree), но для приближенного поиска возобновлено в работе [41] и продолжено в [11], где применялся обход best-first и ранний останов. В экспериментах [11] KM-tree превосходит лес RKD-деревьев (подразд. 1.4.1) для данных не слишком малой внутренней размерности.

В POC-tree [42] границы каждой пары кластеров на всех уровнях дерева максимально отодвигают от разделяющей их центры гиперплоскости. Это обеспечивает бóльшую компактность областей и более плотные нижние границы расстояний областей от объекта-запроса для исключения при точном поиске (см. также подразд. 1.5). Для приближенного поиска ограничивают количество посещений как узлов-кластеров на каждом уровне дерева, так и объектов (ранний останов). В [43] бинарные векторы, сохраняющие сходство представленных ими объектов, используют как для быстрого нахождения ближайших центров KM-tree при его обходе, так и для предварительного ранжирования объектов-кандидатов.

1.3.2. Адаптация к данным деревьев с исчерпывающим разбиением пространства. В PCA-tree [44, 45] направление разбиения является главным собственным вектором ковариационной матрицы подмножества векторов. В TP-tree [33] выбирают направления с максимальной дисперсией для случайных разреженных тернарных векторов. Для разбиения 2M-tree [46, 47] использует направление, задаваемое центрами, полученными K-means с $K = 2$. Кластеризация на два кластера с разделительной полосой (separation margin) применяется в MM-tree [48, 47].

Для вариантов бинарных деревьев с исчерпывающим разбиением пространства в [46] экспериментально обнаружили зависимость качества поиска NN без обратного прослеживания от скорости уменьшения диаметра областей данных на уровнях дерева. В [47] показали прямую зависимость качества поиска NN в худшем случае (в терминах расстояния до истинного NN и ранга) от ошибки векторного квантования и величины разделительной полосы. Наилучшие результаты показывают деревья с адаптацией (2M-tree, PCA-tree и особенно MM-tree в [47]), затем следует RP-tree, а наихудшие результаты у вариантов KD-tree. В [49] исследованы spill-tree с такими же разбиениями, наилучшие результаты у PCA spill-tree.

1.4. Леса рандомизированных деревьев. Вероятность δ ошибки поиска точного NN можно уменьшить до δ^L применением множества L независимых структур [25, 50]. Кроме того, эксперименты с KD-tree [26] показывают, что вероятность отыскания точного NN при обходе best-first растет с увеличением числа посещенных листовых узлов быстрее, чем при обходе depth-first с обратным прослеживанием, однако при увеличении количества просмотренных листьев эта вероятность растет все медленнее. Это обусловило идею использования множества (леса) независимых рандомизированных деревьев как для повышения вероятности нахождения kNN (при том же общем количестве просматриваемых узлов), так и для ускорения поиска при том же качестве результатов. Кроме работы [51], теоретический анализ не проводился.

1.4.1. Лес KD-деревьев. При поиске в лесу деревьев [26] в качестве аналога обратного прослеживания используют единую для всех деревьев приоритетную очередь. Эксперименты с лесами RKD-, NKD-, PKD-trees показали улучшение результатов по сравнению с одним деревом при том же количестве обработанных объектов. Эксперименты с лесом RKD-trees в пакете FLANN с автоматической конфигурацией параметров [11] дали ускорение поиска (при той же вероятности нахождения NN по L_2) по сравнению с KM-tree на данных с малой внутренней размерностью (также показано улучшение по сравнению со spill-tree [16] и LSH [50, 1]).

В [51] для алгоритма поиска L_2-kNN лесом NKD-tree с быстрым случайным вращением данных в каждом дереве дан теоретический анализ вероятности нахождения точных NN и среднего расстояния L_2 от них для возвращенных объектов. Для отбора k кандидатов в каждом дереве используют линейный поиск в листовой корзине, которой принадлежит объект-запрос (содержит приблизительно k объектов), а также в листовых корзинах с кодами, отличающимися на 1 по $\mathrm{dist}_{\mathrm{Ham}}$. Код корзины получают конкатенацией битов выбора области на каждом уровне дерева. Для каждого из k кандидатов извлекают по k объектов из списка приближенных NN, сформированного на этапе построения ИС (граф соседства, см. разд. 3). Таким обра-

зом, для каждого дерева получают приблизительно k^2 кандидатов, в качестве ответа на запрос из их общего списка линейным поиском находят k лучших. Эта ИС очень хорошо работает для $D = 20$, неплохо — для $D = 60$, заметно хуже — для $D = 100$. Возможно, результаты изменило бы использование различных алгоритмов выбора координат для разбиения подмножеств объектов.

Эксперименты с лесами рандомизированных KD-деревьев различных типов с общей приоритетной очередью в пакете GeRaF [52] показали, что вращение и отражение данных, а также случайное возмущение разбиения по медиане не приводят к улучшению результатов относительно леса RKD-trees. Автоматическая конфигурация осуществлялась аппроксимацией вручную настроенных параметров на ряде баз. При $D = 10^3 - 10^4$ и $N \approx 10^5 - 10^6$ GeRaF показал преимущество над FLANN, BBD-tree, LSH и PQ (подразд. 2.1).

1.4.2. Лес RP-деревьев. Лес TP-trees [33] с общей приоритетной очередью значительно улучшает точность поиска по сравнению с TP-tree при малом времени запроса.

В лесах RP-trees MRPT [53] применяют проецирование разреженными векторами, где ненулевые компоненты — гауссовы случайные; для сбалансированности разбиение проводят по медиане. Для снижения затрат памяти на одном уровне RP-tree используют один случайный вектор, как и в [25]. Кандидатами, полученными с помощью defeatist search, являются объекты, которые наиболее часто встречались в листьях (порог «частоты» — настраиваемый параметр), что позволяет уменьшить их количество при высоком «качестве» (см. также в обзоре [1] LSH с такой схемой). Эксперименты на базах с D, равным от 128 до 9216 (в основном $D = 1000$ и 4000), показали, что MRPT быстрее сравниваемых алгоритмов при большой точности поиска (более 95%).

1.4.3. Лес деревьев K-средних. В экспериментах [11] лес KM-trees (с рандомизацией различными начальными центрами кластеризации) не улучшил результатов по сравнению с одиночным KM-tree. В лесах POC-trees [42] для бóльшего разнообразия используют KM-trees с различными (небольшими) K и параметрами, оптимизированными для каждого дерева.

1.5. Индексные структуры на основе неиерархической кластеризации. В ИС с неиерархическим разбиением объектов и пространства для точного поиска (по расстояниям L_2 и Махаланобиса) методом B&B разбиение часто проводят методом K-means (см. подразд. 1.3.1). Эффективность исключения кластеров повышает применение предложенных в [54] нижних границ расстояний от объекта-запроса до объектов кластеров с использованием расстояний запроса и объектов кластеров до (кусочно-линейных) границ областей Вороного.

В [55] сохраняют (упорядоченные) расстояния между объектами кластера и его центром. При поиске рассматривают кластеры по возрастанию расстояний центров от объекта-запроса и используют неравенство треугольника для исключения объектов (или целых кластеров). Ускорение [54] за счет введения условий исключения объектов в кластерах также предложено в [56–58]. Кроме того, в [57] случайным проецированием ускоряют вычисление нижних границ расстояний до кластеров (с эмпирическими гарантиями). В [58] для исключения используют иерархию кластеров, полученную разбиением «густозаселенных» кластеров.

Для поиска приближенных NN (без анализа гарантий) во многих работах (например, [59–61]) в качестве кандидатов используют объекты из некоторого количества кластеров с ближайшими к объекту-запросу центрами (без определения пересечения области запроса с кластерами). Этот подход еще называют обратным файлом [59] или обратным индексом [61] (не следует путать с обратным индексом для скалярного произведения, см. обзор [8]).

Несбалансированность кластеров по количеству объектов увеличивает время поиска, так как малозаселенные кластеры выбираются редко. В [60] предложена процедура балансировки уменьшением более «заселенных» кластеров.

Улучшение качества кандидатов требует большого количества кластеров, что увеличивает время кластеризации и время нахождения ближайших кластеров при выполнении запроса. В обратном мультииндексе (IMI) [61] для снижения этих затрат векторы разбивают на малое количество частей (обычно на две половинной размерности) и выполняют кластеризацию базы для каждой части. Конкатенация всех пар центров частей дает K^2 векторов центров полной размерности, каждый из которых определяет ячейку. Вектор базы относят к той ячейке, которая соответствует ближайшему центру для каждой части вектора. Это дает намного более мелкозернистое разбиение пространства, чем обычная кластеризация с $2K$ центрами, однако количество векторов в ячейках может существенно отличаться. При поступлении вектора-запроса вычисляют расстояния L_2 его частей до всех K центров каждой части. Затем multi-sequence algorithm (MSA) быстро определяет последовательность пар центров (т.е. ячейки) в порядке возрастания суммарного расстояния от вектора запроса. Объекты ближайших ячеек являются кандидатами.

В алгоритме BDH [62] для ускорения выбора ячеек с кандидатами вместо MSA используют B&B с верхними и нижними границами расстояний до ячеек. Кроме того, в частях, которые дают наибольшую ошибку квантования, увеличивают количество центров. Более точный выбор заданного количества лучших кандидатов (потенциально из всех кластеров полноразмерных векторов) за счет запоминания количества объектов в заданных диапазонах расстояний до центров их кластеров предложен в [63]; рассматривается также вариант с IMI.

В [64] используют набор L квантователей K-means полноразмерных векторов, полученных за счет различных начальных центров алгоритма K-means. В качестве кандидатов возвращают векторы из кластера с ближайшим к вектору-запросу центром для каждого квантователя. Этот метод, названный K-means LSH по аналогии с методами [50, 1], дает хорошие результаты поиска, хотя и увеличивает затраты памяти. Для преодоления недостатков, связанных с возможной близостью центров различных квантователей, приводящей к дублированию векторов их кластеров, в [65] используют алгоритм K-means с LK кластерами, делят полученные центры на K групп по L центров с помощью RP-tree и в каждый из L квантователей отбирают случайно (без замещения) один центр из каждой из K групп.

Для быстрого приближенного поиска ближайших центров в подходе AKM [66] используют лес рандомизированных KD-trees [26], построенных по центрам кластеров.

Отметим перспективность применения в ИС других методов кластеризации [67–71].

2. ИС ДЛЯ ПОИСКА ПО ПРЕОБРАЗОВАННЫМ ПРЕДСТАВЛЕНИЯМ

Преобразования исходных представлений объектов в некоторых случаях позволяют ускорить поиск по сходству (приближенный, иногда точный) как за счет ускорения оценки сходства, так и за счет использования специализированных ИС, хотя и не решают проклятия размерности. В работах [5, 6, 72] приведены обзоры методов формирования векторных представлений без использования обучения, позволяющих быстро оценивать расстояния/сходства исходных объектов, а в [73–75] — методов с обучением. Выделим следующие преобразования и ИС.

2.1. Квантование [36]: преобразование в компактный целочисленный код для аппроксимации исходных представлений. Неиерархическое исчерпывающее разбиение пространства гиперпрямоугольниками путем независимого разбиения каждого измерения на несколько частей (скалярное квантование) используется в VA-File [14]. Компактные коды границ областей запоминают в быстрой памяти. Скорость точного линейного поиска обеспечивается за счет быстрого последовательного доступа к кодам границ и повторного использования расстояний между отдельными компонентами вектора-запроса и векторов границ областей. Существуют многочисленные ускорения и приближенные версии [3, 4].

В векторном квантовании каждый вектор базы относят к ближайшему вектору-центру. Для получения центров применяют кластеризацию (например, K-means

[35–37] и см. подразд. 1.5). Для оценки расстояния от вектора-запроса до векторов кластера используют его расстояние до центра кластера. Для увеличения количества центров с экономным использованием памяти и эффективным вычислением расстояния до них предложен подход «квантование произведения» PQ [59]. Компоненты вектора делят на M непересекающихся частей. Для подвекторов (размерности D / M) каждой части K-means находит небольшое число кластеров (например, $K = 256$) и их центры. Затем вектор (базы или запроса) аппроксимируют конкатенацией ближайших центров в каждой части, которые кодируют конкатенацией их номеров. Методы улучшения точности PQ см. в обзорах [72, 74] и ссылках к ним.

2.2. Индексные структуры на основе B+дерева и одномерного индексирования. Для одномерных данных существует быстрый поиск дихотомией [1] со временем $O(\log N)$ в худшем случае и идеологически близкая к нему ИС B+tree [76]. Для преобразования векторов в одномерные индексные значения, отражающие сходство, используют расстояния до опорных объектов (см. обзор [7]). Выбор опорных объектов выполняют как без учета распределения данных (например, Pyramid [77] и iDistance [78]), так и методами кластеризации (адаптивный iDistance [78] и его многочисленные модификации). Одномерное индексирование на основе заполняющих пространство кривых [79] используется совместно с B+tree как для точного поиска [79, 3] (громоздкими процедурами), так и для простых процедур приближенного поиска в множестве ИС [80] или в одной ИС [81], где запоминают искаженные объекты базы.

2.3. Индексирование представлений сниженной размерности. ИС для быстрого поиска по сходству (точного и приближенного) хорошо работают для векторов малой и умеренной размерности. Поэтому их можно использовать для поиска по сходству векторов, полученных снижением размерности исходных. Методы снижения размерности без обучения (не зависящие от данных) со специфицированным малым искажением расстояний/сходств рассмотрены в [5] и включают методы случайного проецирования (и леммы JL для L_2). В [50, 9] обсуждаются теоретические (ввиду больших затрат памяти) алгоритмы быстрого приближенного поиска со снижением размерности и гарантиями худшего случая.

В практической ИС SRS-tree [82] исходные векторы преобразуют в векторы малой размерности от 6 до 10 с помощью гауссовых случайных проекций (см. обзор [5]). К полученным векторам применяют ИС для инкрементного поиска kNN (используют R-tree). В [83] также снижают размерность до значений меньших, чем по лемме JL, и используют с ними BBD-tree [12].

2.4. Использование бинарных векторных представлений исходных векторов. Формирование векторов с бинарными (или целочисленными) компонентами, которые отражают сходство исходных объектов, позволяют ускорить поиск по сходству за счет более эффективной обработки бинарных векторов, а также наличия эффективных ИС для них (см. обзор [8]). Обзоры методов формирования таких бинарных представлений без обучения см. в [6, 72], а с обучением — в [73, 74]. Так, хэширование LSH [1, 6] преобразует исходные векторы в бинарные/целочисленные, которые позволяют как непосредственно оценивать исходное сходство, так и использовать табличные ИС для быстрого приближенного поиска.

В [85] исходные векторы преобразуют в бинарные разреженные и применяют ИС на основе обратного индексирования [8] для быстрого поиска по косинусу угла исходных векторов. В [86, 29] используют аналогичное преобразование в бинарные разреженные векторы, а в [87, 88] — в тернарные разреженные векторы. Ссылки на другие преобразования в бинарные разреженные векторы см. в обзорах [6, 89].

Для быстрого поиска по сходству невекторных данных используют их сохраняющие сходство преобразования в векторные (с L_s, в основном с L_1, см. обзоры [5, 90]).

3. ИНДЕКСНЫЕ СТРУКТУРЫ НА ОСНОВЕ ГРАФОВ СОСЕДСТВА

В [7] приведен обзор ИС на основе графа соседства для приближенного поиска по сходству по мерам расстояния (метрическим и неметрическим) и сходства для данных с произвольным представлением. Для векторных представлений предложены модификации этого подхода.

Граф соседства [15] — ориентированный граф, где узлы соответствуют объектам базы, а ориентированные ребра соединяют объект с «соседями». Алгоритм жадного поиска стартует со случайного узла и ранжирует его соседей относительно объекта-запроса x. Если текущий узел ближе к x, чем все соседи узла («локальный минимум»), то он и является ответом на запрос. В противном случае переходят к соседу, наиболее близкому к x. Для векторного пространства граф с минимальным числом ребер, позволяющий жадно находить NN, — это граф Делоне, соединяющий центры граничащих областей Вороного. Время его построения в худшем случае $O(N^{D/2})$ [91], причем с ростом D граф становится почти полносвязным. Однако для поиска приближенного NN не требуется точного графа Делоне, поэтому на практике используют графы, подграфы которых являются его аппроксимацией. В [92] предложено использовать граф k_0NN (граф, в котором каждый узел имеет ориентированные связи с k_0 ближайшими соседями).

3.1. Конструирование приближенных графов соседства. Квадратичная относительно N сложность построения точного k_0NN-графа неприемлема на практике. Сложность зачастую уменьшают построением приближенных k_0NN-графов. Так как для конструирования графа не требуется работать с объектами не из базы, использование ИС для поиска приближенных k_0 соседей избыточно. Многие методы построения применяют стратегию «разделяй и властвуй». Объекты базы некоторым способом рекурсивно разделяют на подмножества. В подмножествах линейным поиском строят точные графы соседства. Их объединяют и уточняют соседей. Для векторных данных графы соседства обычно строят по расстоянию L_2 и при построении используют структуру векторных представлений (в основном на этапе разделения).

В [93] рекурсивно разбивают множество объектов базы на два или три пересекающихся подмножества. Для разбиения подмножества используют его главное направление, полученное малозатратной бисекцией Ланцоша. При слиянии подграфов для объектов из пересечения подмножеств выбирают из $2k_0$ их соседей k_0 ближайших. При уточнении соседей для каждого объекта в качестве кандидатов рассматривают текущих соседей, а также и соседей этих соседей.

В [94] применяют иерархические случайные непересекающиеся разбиения по главным направлениям случайных подмножеств разбиваемых объектов. Разбиение повторяют несколько раз и объединяют графы, полученные для подмножеств. Для уточнения графа для каждого объекта в приоритетную очередь заносят приближенных соседей, затем извлекают из очереди текущий ближайший объект и помещают в очередь его непосещенных соседей до тех пор, пока очередь не опустошится или пока не будет достигнуто максимального количества посещенных объектов. Последние являются кандидатами для уточнения приближенных k_0NN.

В [95] векторы, полученные LSH-хэшированием, проецируют на случайное направление, упорядочивают проекции по величине и разбивают упорядоченные объекты на непересекающиеся подмножества одинаковой мощности. Процедуру повторяют несколько раз и объединяют полученных кандидатов. Уточнение проводят аналогично [93]. В работе [96] применяют для разбиения области леса RP-trees, для уточнения — несколько шагов перехода к соседям соседей. EFANNA [97] в качестве кандидатов использует объекты, полученные обратным прослеживанием с ограниченной глубиной в лесу рандомизированных KD-trees, а уточнение проводят методом для невекторных графов соседства [98]. В работе [99] проводят разбиение лесом модифицированных 2M-trees. В DPG [100] ди-

версифицируют граф k_0NN путем выбора объектов-соседей целевого объекта так, чтобы максимизировать средний угол между соседями, а для преодоления проблемы потери связности графа используют неориентированные ребра.

FAISS [101] применяет PQ для конструирования приближенного графа k_0NN (по L_2).

Построение точного и приближенного графа соседства по косинусу угла для разреженных неотрицательных векторов рассмотрено в [102]. Начинают с построения приближенного графа с применением обратного индекса [8] и подходов из [98]. Затем каждый объект базы используют в качестве запроса к приближенному графу. При поиске для исключения кандидатов применяют частичный обратный индекс и границы значений расстояний. KIFF [103] предназначен для конструирования приближенного графа k_0NN для различных мер сходства разреженных векторов, использует двудольный граф и хорошо параллелизуется. FLASH [104] также хорошо параллелизуется, но применяет модификацию ИС LSH для получения соседей по сходству Жаккара бинарных разреженных многомерных векторов.

3.2. Поиск. При выполнении запроса для повышения качества соседей объекта-запроса, найденных алгоритмом жадного поиска, используют рестарты с различных случайно выбранных узлов, а также осуществляют переход не на один, а на некоторое количество непосещенных узлов, ближайших к объекту-запросу. Также применяют приоритетную очередь, упорядоченную по расстоянию объекта-узла до объекта-запроса, куда помещают один или несколько стартовых узлов. Из очереди извлекают первый (т.е. ближайший к объекту-запросу) объект и его еще непосещенных соседей по графу заносят в очередь. Такое «расширение окрестностей» проводят до заданного количества посещенных объектов (ранний останов).

Для поиска хороших стартовых объектов применяют дополнительные ИС. В [105, 97] используют приближенный поиск в KD-trees. В [106] находят стартовые объекты, близкие к объекту-запросу по сохраняющим сходство хэшам. В [99] используют объекты, соответствующие ближайшим к запросу векторам-центрам, полученным алгоритмом residual vector quantization из векторов базы при построении ИС.

В [107] применяют комбинацию графа соседства и мостового графа, соединяющего мостовые векторы с их приближенными ближайшими соседями базы. Для получения большого количество мостовых векторов, аппроксимирующих векторы базы, используют PQ ([59] и подразд. 2.1). При выполнении запроса быстро проводят упорядочение мостовых векторов по отношению к вектору-запросу с помощью MSA [61] и помещают в приоритетную очередь мостовой вектор, ближайший к запросу. На каждой итерации поиска, если первым в очереди является мостовой вектор, в очередь помещают соседние с ним объекты базы, а также следующий ближайший к запросу мостовой вектор. Если первым в очереди является объект базы, поиск продолжают, как для обычного графа соседства.

Отметим, что для поиска по максимуму скалярного произведения можно использовать расширенное представление исходных векторов и поиск по $\mathrm{sim}_{\cos}$ [108].

Для преодоления недостатка графа k_0NN, связанного с отсутствием гарантий быстрого нахождения NN процедурами жадного поиска, используют графы тесного мира со свойством навигации ([109–111] и обзор [7]). В них имеются не только связи объекта с ближайшими, но и с дальними объектами. ИС на основе таких графов не используют векторности представлений, могут работать с неметрическими расстояниями/сходствами, показывают одни из лучших результатов в задачах поиска по сходству [109–112]. Подбор параметров позволяет на ряде баз получать точные результаты поиска (за счет увеличения времени выполнения запроса по сравнению с приближенным поиском).

ОБСУЖДЕНИЕ И ЗАКЛЮЧЕНИЕ

Рассмотренные в обзоре ИС для поиска по сходству вещественных векторов разделены на три основных типа: на основе сохраняющего сходство хэширования [1], областей и границ (см. разд. 1), графов соседства (см. разд. 3). Различные преобразования исходных векторов (см. разд. 2) используются с различными типами ИС, но в данном изложении более близки к ИС из разд. 1.

ИС первых двух типов во многом схожи, третий тип отличается от них. При конструировании ИС упомянутых типов множество объектов базы разбивают на подмножества (пересекающиеся или непересекающиеся). Разбиения в ИС на основе областей и графов соседства в той или иной степени зависят от данных и их можно рассматривать как методы с адаптацией (обучение без учителя). Разбиения в ИС на основе хэширования в своем базовом варианте не зависят от данных. При выполнении запроса вычисляют расстояния объекта-запроса до объектов только «перспективных» подмножеств, что в ряде случаев позволяет ускорить линейный поиск и получить гарантии ускорения для данных худшего случая для некоторых типов запросов приближенного поиска в ИС на основе хэширования.

ИС на основе областей. В таких ИС подмножествам объектов базы соответствуют области ИС, обычно не пересекающиеся. При разбиении в той или иной степени учитываются данные базы. Возможность определить нижнюю границу расстояний от объекта-запроса до объектов подмножества (или верхнюю границу для сходств) позволяет исключать из рассмотрения подмножества объектов, которые не могут содержать ответ на запрос. В этом заключается суть метода ветвей и границ B&B в поиске по сходству с гарантией точного поиска.

Большое количество областей ИС, плотно ограничивающих малые подмножества своих объектов, дает возможность уменьшить избыточность подмножеств объектов, отбираемых для дальнейшей обработки. Ускорить построение большого количества областей ИС и определение принадлежности к ним объектов (базы и запросов) позволяют иерархические (древовидные) ИС. Область ИС, которой принадлежит объект-запрос, обычно содержит сходные с ним объекты базы, однако ближайший сосед может находиться и в соседних областях, пересекающих текущую область запроса. Для исследования этих областей используют «обратное прослеживание», что замедляет поиск, а в многомерных пространствах приводит к исследованию большей части областей ИС. Поиск при этом вырождается в линейный.

Модификации ИС на основе областей. Поиск методом B&B ускоряют за счет перехода к приближенному поиску, который допускает возвращение неточных результатов (объектов, не являющихся точными ближайшими соседями, хотя обычно расположенных не намного дальше от объекта-запроса). В первую очередь посещают ближайшие к запросу области ИС и останавливают поиск до завершения исследования всех тех областей, которые могут содержать объекты, являющиеся точным ответом на запрос. Предельным случаем такого подхода есть полный отказ от обратного прослеживания и останов поиска после обработки объектов только тех подмножеств/областей ИС, которым принадлежит объект-запрос.

Извлечение объектов-кандидатов также и из областей, соседних с областью запроса, увеличивает шансы нахождения точного (или хорошего) ближайшего соседа. Соседние области деревьев иногда определяют по сходству путей в них из корня, используют также перекрывающиеся области (при конструировании или при поиске). Точность поиска повышает извлечение кандидатов из нескольких ИС, построенных с различными разбиениями на подмножества/области (например, лес деревьев). Если ИС таковы, что объект-кандидат встречается несколько раз, ускорения приближенного поиска можно достичь ограничением списка кандидатов за счет выбора наиболее часто встречающихся в нем объектов.

Для некоторых ИС на основе областей существует анализ времени поиска для данных худшего случая или данных с определенными ограничениями (на

внутреннюю размерность и др.). Однако зависимость времени запроса от размерности данных остается экспоненциальной.

ИС на основе хэширования [1]. В таких ИС значение хэша объекта определяет корзину ИС, которой он принадлежит. При поиске извлекают объекты-кандидаты из корзины-области, соответствующей хэшу объекта-запроса. В отличие от обычного хэширования, которое применяют для поиска в базе объекта, совпадающего с запросом, при поиске по сходству используют чувствительное к сходству хэширование LSH. Одинаковые хэш-значения LSH-функция с большей вероятностью назначает сходным объектам, чем несходным. Поэтому в одной корзине (подмножестве объектов) находятся, в основном, сходные объекты. При выполнении запроса LSH обеспечивает извлечение объектов-кандидатов, сходных с объектом-запросом.

Преимущество ИС LSH по сравнению со всеми другими рассматриваемыми типами ИС заключается в теоретически обоснованном выборе параметров ИС, позволяющим гарантировать сублинейное время выполнения некоторых типов запросов вероятностного приближенного поиска для худшего случая (а также величину отличия от результатов точного поиска и вероятность невозвращения объектов, которые являются ответом на запрос).

Обычно LSH-функцию конструируют так, что она продуцирует хэш-значения, соответствующие большому количеству корзин. Для одной LSH-функции конкретный объект принадлежит одной корзине. Таким образом, LSH-функция реализует исчерпывающее разбиение пространства на непересекающиеся области. Однако в отличие от ИС на основе областей в базовом варианте ИС LSH границы областей не используются.

LSH-хэши часто конструируют конкатенацией значений нескольких LSH-функций. Это аналогично последовательности индексных значений узлов, выбираемых на каждом уровне дерева специального типа, в котором разбиения в узлах выполняются исчерпывающе без пересечения и в каждом узле уровня они одинаковые.

Как и в ИС на основе областей, сходные объекты могут не оказаться в корзине объекта-запроса. Для снижения вероятности некорректного результата поиска конструируют несколько реализаций ИС с различными LSH-функциями (т.е. разными разбиениями пространства) и извлекают кандидатов из одной корзины (соответствующей хэшу объекта-запроса) каждой ИС.

Модификации ИС на основе хэширования. Для снижения требующегося количества реализаций ИС в мультипробном LSH кандидатов извлекают из корзин-областей, близких к объекту-запросу (это аналог обратного прослеживания в ИС на основе областей).

Для обеспечения теоретических гарантий времени и качества выполнения запроса требуется построение ИС LSH для каждого радиуса запроса и степени приближенности поиска (в дополнение к необходимости существования LSH-функций для используемой в поиске функции расстояния/сходства). Применительно к поиску ближайшего соседа проблему множественности ИС решают эмуляцией запросов с возрастающим радиусом в одной ИС LSH: объекты-кандидаты извлекают из увеличивающегося количества соседних с объектом-запросом корзин аналогично мультипробному LSH.

Для сокращения количества и повышения качества объектов-кандидатов из их исходного списка выбирают наиболее часто встретившихся (с наибольшим количеством совпавших LSH-хэшей с объектом-запросом). Это можно использовать и для раннего останова поиска.

Таким образом, видно, что варианты базовых схем ИС на основе областей и на основе хэширования используют сходные решения для ускорения приближенного поиска: извлечение кандидатов из корзин-областей, которым принадлежит объект-запрос, и близких к ним; применение нескольких ИС с различными разбиениями объектов на подмножества; отбор наиболее перспективных канди-

датов; ранний останов поиска и др. Отметим, что для ускорения разновидностей ИС на основе хэширования могут применяться деревья, например, в локально-чувствительной фильтрации [1].

ИС на основе графов соседства. В таких ИС для каждого объекта базы сохраняют список объектов его окрестности, например, ближайших соседей объекта. Списки объектов окрестностей соседних объектов пересекаются. Выполняется, в основном, быстрый приближенный поиск ближайших соседей.

При выполнении запроса стартуют с некоторого объекта и переходят на еще непосещенный объект из его окрестности, наиболее близкий к объекту-запросу. Пока существуют такие объекты (т.е. до попадания в «локальный минимум»), шаги повторяют. Таким образом, на каждом шаге (до достижения локального минимума) среди объектов окрестностей находят объекты-кандидаты, все более близкие к объекту-запросу. Это можно рассматривать как аналог перехода в корзины-области ИС первых двух типов и нахождения в них ближайших к запросу объектов. Так как найденный объект может не являться ближайшим соседом, выполняется переход на другие близкие объекты окрестностей (аналог обратного прослеживания или мультипробного запроса).

Объекты, посещенные за небольшое количество таких шагов, являются очень хорошими кандидатами на ответ запроса ближайшего соседа. Рестарт поиска с других объектов есть аналог применения нескольких ИС. В качестве стартовых объектов можно использовать результаты быстрого поиска с помощью других ИС. Для ограничения времени поиска практикуется ранний останов — ограничение количества посещаемых объектов и/или рестартов.

К преимуществам графов соседства относятся высокая скорость и качество (приближенного) поиска, а также то, что во многих их типах не используется векторная структура объектов, часто не требуется даже метричности расстояний. К недостаткам относятся сложность построения (например, квадратичная от количества объектов базы для графа ближайших соседей, хотя предложены упрощения и инкрементные процедуры); отсутствие аналитических гарантий времени и качества поиска; применение, главным образом, для поиска приближенных ближайших соседей.

Поиск по сходству в ИС на основе графов соседства показывает конкурентоспособные результаты [109–112] по сравнению со всеми другими ИС. Иерархическая ИС HNSW [111] на основе графов тесного мира со свойством навигации, которая не использует векторной структуры данных, позволяет получить результаты поиска в базах векторов выше, чем лучшие ИС, специализированные для векторов. Отметим, что специализированный для сходства Жаккара между бинарными разреженными векторами очень большой размерности хорошо параллелизуемый алгоритм FLASH [104] (модифицированный вариант ИС LSH на основе MinHash) продемонстрировал преимущество над другими ИС, включая HNSW.

В заключение упомянем ИС для поиска приближенного ближайшего соседа, сильно отличающиеся от рассмотренных. Это ИС на основе нейросетевой распределенной автоассоциативной памяти (NAM) [89, 8, 113, 114]. Векторы базы не хранят в ИС в явном виде изолировано один от другого, а формируют из них «матрицу связей». На вход подают зашумленный вектор базы. Восстановление ближайшего вектора базы на выходе NAM выполняется итеративной динамикой нейросети. В отличие от ИС, представленных в настоящем обзоре, для ряда NAM лучшие характеристики достигаются при больших размерностях векторов. Поэтому перспективно исследование совместного использования NAM с другими ИС. Для вещественных векторов некоторые ИС этого типа [115–117] рассмотрены в [89]. Распознавание наличия искаженного вектора в сумме векторов для ускорения поиска по сходству см. в [118].

СПИСОК ЛИТЕРАТУРЫ

1. Rachkovskij D.A. Index structures for fast similarity search of real-valued vectors. I. C*ybernetics and Systems Analysis.* 2018. Vol. 54, N. 1. P. 152–164.
2. Gaede V., Gunther O. Multidimensional access methods. *ACM Comput. Surv.* 1998. Vol. 30, N 2. P. 170–231.
3. Bohm C., Berchtold S., Keim D.A. Searching in high-dimensional spaces: Index structures for improving performance of multimedia databases. *ACM Comput. Surv*. 2001. Vol. 33, N 3. P. 322–373.
4. Samet H. Foundations of multidimensional and metric data structures. San Francisco: Morgan Kaufmann, 2006. 1024 p.
5. Rachkovskij D.A. Real-valued embeddings and sketches for fast distance and similarity estimation. *Cybernetics and Systems Analysis.* 2016. Vol. 52, N. 6. P. 967–988.
6. Rachkovskij D.A. Binary vectors for fast distance and similarity estimation. *Cybernetics and Systems Analysis.* 2017. Vol. 53, N 1. P. 138–156.
7. Rachkovskij D.A. Distance-based index structures for fast similarity search. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 4. P. 636–658.
8. Rachkovskij D.A. Index structures for fast similarity search of binary vectors. *Cybernetics and Systems Analysis*. 2017. Vol. 53, N 5. P. 799–820.
9. Andoni A., Indyk P. Nearest neighbors in high-dimensional spaces. *Handbook of Discrete and Computational Geometry.* 3rd ed. Boca Raton, USA: CRC Press, 2017. Chap. 43. P. 1135–1155.
10. Patella M., Ciaccia P. Approximate similarity search: a multi-faceted problem. *J. Discrete Algorithms*. 2009. Vol. 7, N 1. P. 36–48.
11. Muja M., Lowe D.G. Scalable nearest neighbor algorithms for high dimensional data. *IEEE TPAMI*. 2014. Vol. 36, N 11. P. 2227–2240.
12. Arya S., Mount D., Netanyahu N., Silverman R., Wu A. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*. 1998. Vol. 45, N 6. P. 891–923.
13. Friedman J.K., Bentley J.L., Finkel R.A. An algorithm for finding best matches in logarithmic expected time. *ACM Tran. on Mathematical Software*. 1977. Vol. 3, N 3. P. 209–226.
14. 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.
15. Arya S., Mount D.M. Approximate nearest neighbor queries in fixed dimensions. *Proc. SODA'93*. 1993. P. 271–280.
16. Liu T., Moore A.W., Gray A., Yang K. An investigation of practical approximate nearest neighbor algorithms. *Proc. NIPS'04.* 2004. P. 825–832.
17. Lee D.T., Wong C.K. Worst-case analysis for region and partial region searches in multidimensional binary trees and balanced quad trees. *Acta Informatica*. 1977. Vol. 9, N 1. P. 23–29.
18. Guttman A. R-trees: A dynamic index structure for spatial searching. *Proc. ACM SIGMOD ICMD'84*. 1984. P. 47–57.
19. Pagel B.U., Korn F., Faloutsos C. Deflating the dimensionality curse using multiple fractal dimensions. *Proc. ICDE'00.* 2000. P. 589–598.
20. White D.A., Jain R. Similarity indexing with the SS-tree. *Proc. ICDE'96*. 1996. P. 516–523.
21. Omohundro S.M. Five balltree construction algorithms. ICSI TR-89-063. 1989. 22 p.
22. Katayama N., Satoh S. The SR-tree: An index structure for high-dimensional nearest neighbor queries. *Proc. ACM SIGMOD ICMD'97*. 1997. P. 369–380.
23. Arge L., de Berg M., Haverkort H.J., Yi K. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. on Algorithms.* 2008. Vol. 4, N 1. P. 9:1–9:30.
24. Dasgupta S., Sinha K. Randomized partition trees for nearest neighbor search. *Algorithmica*. 2015. Vol. 72, N 1. P. 237–263.
25. Yianilos P. Locally lifting the curse of dimensionality for nearest neighbor search. *Proc. SODA'00*. 2000. P. 361–370.
26. Silpa-Anan C., Hartley R. Optimised kd-trees for fast image descriptor matching. *Proc. CVPR'08*. 2008. P. 1–8.
27. Dasgupta S., Freund Y. Random projection trees and low dimensional manifolds. *Proc. STOC'08*. 2008. P. 537–546.

28. Allen-Zhu Z., Gelashvili R., Micali S., Shavit N. Sparse sign-consistent Johnson–Lindenstrauss matrices: Compression with neuroscience-based constraints. *PNAS*. 2014. Vol. 111. P. 16872–16876.
29. Rachkovskij D.A. Formation of similarity-reflecting binary vectors with random binary projections. *Cybernetics and Systems Analysis*. 2015. Vol. 51, N 2. P. 313–323.
30. Jagadeesan M. Simple analysis of sparse, sign-consistent JL. arXiv:1708.02966. 9 Aug 2017.
31. Dasgupta S., Stevens C.F., Navlakha S. A neural algorithm for a fundamental computing problem. *Science*. 2017. Vol. 358, N 6364. P. 793–796.
32. Sinha K. Fast L1-norm nearest neighbor search using a simple variant of randomized partition tree. *Procedia Computer Science*. 2015. Vol. 53. P. 64–73.
33. Wang J., Wang N., Jia Y., Li J., Zeng G., Zha H., Hua X.-S. Trinary-projection trees for approximate nearest neighbor search. *IEEE Trans. PAMI.* 2014. Vol. 36, N 2. P. 388–403.
34. Vempala S. Randomly-oriented k-d trees adapt to intrinsic dimension. *Proc. FSTTCS'12*. 2012. P. 48–57.
35. MacQueen J.B. Some methods for classification and analysis of multivariate observations. *Proc. MSP'67*. 1967. P. 281–297.
36. Gray R.M., Neuhoff D.L. Quantization. *IEEE Trans. IT.* 1998. Vol. 44. P. 2325–2384.
37. Xu R., Wunsch D. Survey of clustering algorithms. *IEEE TNN*. 2005. Vol. 16. P. 645–678.
38. Fabregas A.C., Gerardo B.D., Tanguilig III B.T. Enhanced initial centroids for kmeans algorithm. *Int. J. of Information Technology and Computer Science.* 2017. Vol. 9, N 1. P. 26–33.
39. Kaur H., Verma P. Comparative Weka analysis of clustering algorithm's. *International Journal of Information Technology and Computer Science*. 2017. Vol. 9, N 8. P. 56–67.
40. Fukunaga K., Narendra P.M. A branch and bound algorithm for computing k-nearest neighbors. *IEEE Trans. Comput.* 1975. Vol. C-24, N 7. P. 750–753.
41. Nister D., Stewenius H. Scalable recognition with a vocabulary tree. *Proc. CVPR'06*. 2006. P. 2161–2168.
42. Pham T.-A. Pair-wisely optimized clustering tree for feature indexing. *Computer Vision and Image Understanding*. 2017. Vol. 154. P. 35–47.
43. Zhang D., Yang G., Hu Y., Jin Z., Cai D., He X. A unified approximate nearest neighbor search scheme by combining data structure and hashing. *Proc. IJCAI'13*. 2013. P. 681–687.
44. Sproull R.F. Refinements to nearest-neighbor searching in k-dimensional trees. *Algorithmica*. 1991. Vol. 6, N 1. P. 579–589.
45. McNames J. A fast nearest-neighbor algorithm based on a principal axis search tree. *IEEE Trans. PAMI.* 2001. Vol. 23, N 9. P. 964–976.
46. Verma N., Kpotufe S., Dasgupta S. Which spatial partition trees are adaptive to intrinsic dimension? *Proc. UAI'09.* 2009. P. 565–574.
47. Ram P., Gray A.G. Which space partitioning tree to use for search? *Proc. NIPS'13*. 2013. P. 656–654.
48. Ram P., Lee D., Gray A.G. Nearest-neighbor search on a time budget via max-margin trees. *Proc. ICDM'12.* 2012. P. 1011–1022.
49. McFee B., Lanckriet G. Large-scale music similarity search with spatial trees. *Proc. ISMIR'11*. 2011. P. 55–60.
50. Har-Peled S., Indyk P., Motwani R. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory Comput*. 2012. Vol. 8. P. 321–350.
51. Jones P.W., Osipov A., Rokhlin V. A randomized approximate nearest neighbors algorithm. *Applied and Computational Harmonic Analysis*. 2013. Vol. 34, N 3. P. 415–444.
52. Avrithis Y., Emiris I.Z., Samaras G. High-dimensional visual similarity search: k-d generalized randomized forests. *Proc. CGI'16.* 2016. P. 25–28.
53. Hyvönen V., Pitkänen T., Tasoulis S., Jääsaari E., Tuomainen R., Wang L., Corander J., Roos T. Fast nearest neighbor search through sparse random projections and voting. *Proc. BigData'16*. 2016. P. 881–888.
54. Ramaswamy S., Rose K. Adaptive cluster distance bounding for high-dimensional indexing. *IEEE Trans. on KDE.* 2011. Vol. 23, N 6. P. 815–830.
55. Wang X. A fast exact *k*-nearest neighbors algorithm for high dimensional search using *k*-means clustering and triangle inequality. *Proc. ICNN'11*. 2011. P. 1293–1299.
56. Hong H., Juan G., Ben W. An improved KNN algorithm based on adaptive cluster distance bounding for high dimensional indexing. *Proc. GCIS'12*. 2012. P. 213–217.
57. Feng X., Cui J., Liu Y., Li H. Effective optimizations of clusterbased nearest neighbor search in highdimensional space. *Multimedia Systems.* 2017. Vol. 23, N 1. P. 139–153.

58. Liu L., Fenghong Xiang F., Mao J., Zhang M. High-dimensional indexing algorithm based on the hyperplane tree-structure. *Proc. IEEE ICIA'15.* 2015. P. 2730–2733.

59. Jegou H., Douze M., Schmid C. Product quantization for nearest neighbor search. *IEEE Trans. PAMI*. 2011. Vol. 33, N 1. P. 117–128.

60. Tavenard R., Jegou H., Amsaleg L. Balancing clusters to reduce response time variability in large scale image search. *Proc. CBMI'11.* 2011. P. 19–24.

61. Babenko A., Lempitsky V. The inverted multi-index. *IEEE Trans. PAMI.* 2015. Vol. 37, N 6. P. 1247–1260.

62. Iwamura M., Sato T., Kise K. What is the most efficient way to select nearest neighbor candidates for fast approximate nearest neighbor search? *Proc. ICCV'13*. 2013. P. 3535–3542.

63. Heo J.P., Lin Z., Shen X., Brandt J., Yoon S.E. Shortlist selection with residual-aware distance estimator for k-nearest neighbor search. *Proc. CVPR'16.* 2016. P. 2009–2017.

64. Pauleve L., Jegou H., Amsaleg L. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters.* 2010. Vol. 31, N 11. P. 1348–1358.

65. Xia Y., He K., Wen F., Sun J. Joint inverted indexing. *Proc. ICCV'13.* 2013. P. 3416–3423.

66. Philbin J., Chum O., Isard M., Sivic J., Zisserman A. Object retrieval with large vocabularies and fast spatial matching. *Proc. CVPR'07*. 2007. P. 1–8.

67. Hu Z., Bodyanskiy Y.V., Tyshchenko O.K., Samitova V.O. Fuzzy clustering data given on the ordinal scale based on membership and likelihood functions sharing. *International Journal of Intelligent Systems and Applications*. 2017. Vol. 9, N 2. P. 1–9.

68. Hu Z., Bodyanskiy Y.V., Tyshchenko O.K., Samitova V.O. Possibilistic fuzzy clustering for categorical data arrays based on frequency prototypes and dissimilarity measures. *International Journal of Intelligent Systems and Applications* 2017. Vol. 9, N 5. P. 55–61.

69. Hu Z., Bodyanskiy Y.V., Tyshchenko O.K, Tkachov V.M. Fuzzy clustering data arrays with omitted observations. *Int. J. Intelligent Systems and Applications.* 2017. Vol. 9, N 6. P. 24–32.

70. Jain A., Mehar P., Buksh B. Advancement in clustering with the concept of correlation clustering — a survey. *Int. J. Engineering Development and Research.* 2016. Vol. 4, N 2. P. 1002–1005.

71. Jain A., Tyagi S. Priority based new approach for correlation clustering. *International Journal of Information Technology and Computer Science*. 2017. Vol. 9, N 3. P. 71–79.

72. Wang J., Shen H.T., Song J., Ji J. Hashing for similarity search: A survey. arXiv:1408.2927. 13 Aug 2014.

73. Wang J., Liu W., Kumar S., Chang S.-F. Learning to hash for indexing big data: A survey. *Proc. of the IEEE.* 2016. Vol. 104, N 1. P. 34–57.

74. 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.

75. Gao L., Song J., Liu X., Shao J., Liu J., Shao J. Learning in high-dimensional multimedia data: The state of the art. *Multimedia Systems*. 2015. P. 1–11.

76. Comer D. The ubiquitous B-tree. *ACM Comput. Surv*. 1979. Vol. 11. P. 121–138.

77. Berchtold S., Bohm C., Kriegel H.-P. The pyramid technique: Towards breaking the curse of dimensionality. *Proc. SIGMOD'98.* 1998. P. 142–153.

78. Jagadish H.V., Ooi B.C., Tan K.L., Yu C., Zhang R. iDistance: An adaptive B+-tree based indexing method for nearest neighbor search. *ACM TODS*. 2005. Vol. 30, N 2. P. 364–397.

79. Lawder J. K., King P.J.H. Querying multi-dimensional data indexed using the hilbert space filling curve. *ACM SIGMOD Record*. 2001. Vol. 30, N 1. P. 19–24.

80. Liao S., Lopez M., Leutenegger S. High dimensional similarity search with space filling curves. *Proc. ICDE'01*. 2001. P. 615–622.

81. Mainar-Ruiz G., Perez-Cortes J. Approximate nearest neighbor search using a single space-filling curve and multiple representations of the data points. *Proc. ICPR'06*. 2006. Vol. 2. P. 502–505.

82. Sun Y., Wang W., Qin J., Zhang Y., Lin X. Srs: Solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. *Proc. VLDB Endowment*. 2014. Vol. 8, N 1. P. 1–12.

83. Anagnostopoulos E., Emiris I.Z., Psarros I. Randomized embeddings with slack, and high-dimensional approximate nearest neighbor. arXiv:1412.1683. 3 Dec 2016.

84. Avarikioti G., Emiris I.Z., Psarros I., Samaras G. Practical linear-space approximate near neighbors in high dimension. arXiv:1612.07405. 22 Dec 2016.

85. Donaldson R., Gupta A., Plan Y., Reimer T. Random mappings designed for commercial search engines. arXiv:1507.05929. 21 Jul 2015.

86. Rachkovskij D.A., Misuno I.S., Slipchenko S.V. Randomized projective methods for construction of binary sparse vector representations. *Cybernetics and Systems Analysis*. 2012. Vol. 48, N 1. P. 146–156.

87. Ferdowsi S., Voloshynovskiy S., Kostadinov D., Holotyak T. Fast content identification in highdimensional feature spaces using sparse ternary codes. *Proc. WIFS'16*. 2016. P. 1–6.

88. Misuno I.S., Rachkovskij D.A., Slipchenko S.V., Sokolov A.M. Searching for text information with the help of vector representations. *Problems of Programming.* 2005. N 4. P. 50–59.

89. Gritsenko V.I., Rachkovskij D.A., Frolov A.A., Gayler R., Kleyko D., Osipov E. Neural distributed autoassociative memories: A survey. C*ybernetics and Computer Engineering*. 2017. N 2 (188). P. 5–35.

90. Indyk P., Matousek J., Sidiropoulos A. Low-distortion embeddings of finite metric spaces. *Handbook of Discrete and Computational Geometry*. 3rd ed. Boca Raton, USA: CRC Press, 2017. Chap. 8. P. 211– 231.

91. Fortune S. Voronoi diagrams and Delaunay triangulations. *Handbook of Discrete and Computational Geometry.* 3rd ed. Boca Raton, USA: CRC Press, 2017. Chap. 27. P. 705–721.

92. Sebastian T., Kimia B. Metric-based shape retrieval in large databases. *Proc. ICPR'02*. 2002. Vol. 3. P. 291–296.

93. Chen J., Fang H., Saad Y. Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection. *Journal MLR.* 2009. Vol. 10. P. 1989–2012.

94. Wang J., Wang J., Zeng G., Tu Z., Gan R., Li S. Scalable k-NN graph construction for visual descriptors. *Proc. CVPR'12*. 2012. P. 1106–1113.

95. Zhang Y.-M., Huang K., Geng G., Liu C.-L. Fast knn graph construction with locality sensitive hashing. *Proc. ECMLPKDD'13*. 2013. P. 660–674.

96. Tang J., Liu J., Zhang M., Mei Q. Visualizing large-scale and high-dimensional data. *Proc. WWW'16*. 2016. P. 287–297.

97. Fu C., Cai D. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. arXiv:1609.07228. 3 Dec 2016.

98. Dong W., Charikar M., Li K. Efficient K-nearest neighbor graph construction for generic similarity measures. *Proc. WWW'11.* 2011. P. 577–586.

99. Zhao W.-L., Yang J., Deng C.-H. Scalable nearest neighbor search based on knn graph. arXiv:1701.08475. 3 Feb 2017.

100. Li W., Zhang Y., Sun Y., Wang W., Zhang W., Lin X. Approximate nearest neighbor search on high dimensional data — experiments, analyses, and improvement. arXiv:1610.02455. 8 Oct 2016.

101. Johnson J., Douze M., Jegou H. Billion-scale similarity search with gpus. arXiv:1702.08734. 28 Feb 2017.

102. Anastasiu D.C., Karypis G. L2knng: Fast exact k-nearest neighbor graph construction with l2-norm pruning. *Proc. CIKM'15*. 2015. P. 791–800.

103. Boutet A., Kermarrec A.M., Mittal N., Taiani F. Being prepared in a sparse world: The case of knn graph construction. *Proc. ICDE'16*. 2016. P. 241–252.

104. Wang Y., Shrivastava A., Ryu J. FLASH: randomized algorithms accelerated over cpu-gpu for ultra-high dimensional similarity search. arXiv:1709.01190. 4 Sep 2017.

105. Wang J., Li S. Query-driven iterated neighborhood graph search for large scale indexing. *Proc. MM'12.* 2012. P.179–188.

106. Jin Z., Zhang D., Hu Y., Lin S., Cai D., He X. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE Trans. on Cybernetics*. 2014. Vol. 44, N 11. P. 2167–2177.

107. Wang J., Wang J., Zeng G., Gan R., Li S., Guo B. Fast neighborhood graph search using cartesian concatenation. *Multimedia Data Mining and Analytics.* Cham: Springer. 2015. P. 397–417.

108. Neyshabur B., Srebro N. On symmetric and asymmetric lshs for inner product search. *Proc. ICML'15*. 2015. P. 1926–1934.

109. Ponomarenko A., Avrelin N., Naidan B., Boytsov L. Comparative analysis of data structures for approximate nearest neighbor search. *Proc. Data Analytics'14*. 2014. P. 125–130.

110. Naidan B., Boytsov L., Nyberg E. Permutation search methods are efficient, yet faster search is possible. *Proc. VLDB Endowment.* 2015. Vol. 8, N. 12. P. 1618–1629.

111. 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.

112. Aumuller M., Bernhardsson E., Faithfull A. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Proc. SISAP'17.* 2017. P. 34–49.

113. 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.

114. 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.

115. Salavati A.H., Kumar K.R., Shokrollahi A. Nonbinary associative memory with exponential pattern retrieval capacity and iterative learning. *IEEE TNNLS.* 2014. Vol. 25, N 3. P. 557–570.

116. Mazumdar A., Rawat A.S. Associative memory using dictionary learning and expander decoding. *Proc. AAAI'17.* 2017. P. 267–273.

117. Ferro D., Gripon V., Jiang X. Nearest neighbour search using binary neural networks. *Proc. IJCNN'16*. 2016. P. 5106–5112.

118. Iscen A., Furon T., Gripon V., Rabbat M., Jegou H. Memory vectors for similarity search in high-dimensional spaces. *IEEE Trans. on Big Data*. 2017. DOI: 10.1109/TBDATA.2017.2677964.

Надійшла до редакції 28.09.2017

Д.А. Рачковський

ІНДЕКСНІ СТРУКТУРИ ДЛЯ ШВИДКОГО ПОШУКУ ЗА СХОЖІСТЮ ДІЙСНИХ ВЕКТОРІВ. II

Анотація. Наведено огляд індексних структур для швидкого пошуку за схожістю об'єктів, що представлені дійсними векторами. Розглянуто структури як для точного, так і для наближеного пошуку. Проаналізовано головним чином індексні структури на основі розбиття на області (у тому числі ієрархічні) та графів сусідства. Обговорено також прискорення пошуку за схожістю з використанням перетворення вхідних даних. Викладено ідеї конкретних алгоритмів (відомих та нещодавно запропонованих). Наведено порівняння підходів до прискорення пошуку за схожістю в індексних структурах розглянутих типів, а також на основі хешування, що зберігає схожість.

Ключові слова: пошук за схожістю, найближчий сусід, ближній сусід, індексні структури, метод гілок і меж, дерева і ліси, кластеризація, граф сусідства, локально-чутливе хешування.

Rachkovskij D.A.

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

Abstract. In this survey paper, we consider index structures for fast similarity search of objects represented by real-valued vectors. Structures for both exact and faster, but approximate, similarity search are considered. We present index structures mainly on the basis of partitioning into regions (including hierarchical ones) and neighborhood graphs. The acceleration of the similarity search using the transformation of the original data is also discussed. The ideas of specific algorithms, including the recently proposed ones, are outlined. The approaches to the speed-up of similarity search in the index structures of the considered types and those based on similarity-preserving hashing are discussed and compared.

Keywords: similarity search, nearest neighbor, near neighbor, index structures, branch and bound, trees and forests, clustering, proximity graph, locality-sensitive hashing.

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.