DOI 10.37882/2223-2966.2024.05.09

СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ ПОИСКА БЛИЖАЙШИХ СОСЕДЕЙ С ИСПОЛЬЗОВАНИЕМ ГРАФОВ NSG И HNSW

COMPARATIVE ANALYSIS OF GRAPH-BASED NEAREST NEIGHBOR SEARCH ALGORITHMS NSG AND HNSW

B. Goryachkin
A. Pavlovskaya
Yu. Grigoriev

Summary. In this paper, two graph-based nearest neighbor search algorithms are discussed: NSG (Navigating spreading-out graph) and HNSW (Hierarchical navigable small world). The relevance of the K-NNS method in modern technologies is substantiated, and specific examples of the application of neighbor search methods are given. Justification for the effectiveness of using graph structures to search for K nearest neighbors is given. Theoretical calculation of the time and volume complexity of the HNSW and HSG algorithms, as well as the quantitative characteristics of the Python implementation, was performed. The obtained theoretical values were confirmed experimentally. It is concluded that NSG is a more reliable algorithm in the case of large data volumes, but it uses more memory for graph construction per stage than HNSW.

Keywords: K-NSS, K nearest neighbors search, graph-based NNS, NSG, Navigating Spreading-out Graph, HNSW, Hierarchical navigable small world.

Горячкин Борис Сергеевич
кандидат технических наук, доцент, Московский государственный технический университет им. Н.Э. Баумана
bsgor@mail.ru
Павловская Анастасия Андреевна
Московский государственный технический университет им. Н.Э. Баумана
paa4851@gmail.com
Григорьев Юрий Александрович
д.т.н., профессор, Московский государственный технический университет им. Н.Э. Баумана
grigorev@bmstu.ru

Аннотация. В данной работе рассмотрены два алгоритма поиска ближайших соседей с использованием графов: NSG (Navigating Spreading-out Graph) и HNSW (Hierarchical navigable small world). Обоснована актуальность метода поиска ближайших соседей в современных технологиях, приведены конкретные примеры применения алгоритмов поиска соседей. Приводится обоснование эффективности применения графовых структур для поиска K ближайших соседей. Выполнен теоретический расчет временной и емкостной сложности алгоритмов HNSW и HSG, а также количественных характеристик конкретной реализации на Python. Полученные теоретические значения были подтверждены экспериментально. Был сделан вывод о том, что NSG является более предпочтительным вариантом в случае больших объемов данных, но использует больше памяти для стадии построения графа, чем HNSW.

Ключевые слова: K-NSS, поиск K ближайших соседей, основанные на графах методы поиска ближайших соседей, NSG, Navigating Spreading-out Graph, HNSW, Hierarchical navigable small world.

Введение

В настоящее время объем информации, нуждающейся в обработке, постоянно растет. Это закономерно привело к необходимости использования эффективных алгоритмов поиска, снижающих затраты на просмотр всего объема данных.

Одним из известных методов является поиск ближайшего соседа (K-NNS), суть которого заключается в поиске в некотором метрическом множестве K элементов, расположенных ближе всего к заданному. Мера близости является функцией, задаваемой предметной областью: чем меньше значения близости, тем более схожими можно считать сравниваемые элементы и наоборот. Проблема поиска ближайшего соседа встречается в множестве областей, например, в задачах распознавания образов, классификации объектов, рекомендательных системах, задачах сжатия данных, размещении рекламы в сети Интернет, семантического поиска и др.

Однако, сложность классического подхода к алгоритму K-NNS растет линейно в зависимости от количества хранимых элементов и размерности этих элементов, что для крупномасштабных данных зачастую приводит к слишком долгой обработке или большим объемам требуемой памяти, к значительным потерям в точности. Поэтому стала актуальной проблема разработки наиболее эффективных и быстрых алгоритмов поиска.

Методы поиска ближайшего соседа с использованием графов значительно снижают сложность вычислений и ускоряют процесс получения результата. К таким методам относятся, например, GNNS [1], IEH [2]. Efanna [3],

FANNG [4], RNG [5], NSG [6] и HNSW [7]. В данной работе будет проведен анализ и сравнение HNSW и NSG с целью выявления особенностей, преимуществ, недостатков и возможных вариантов применения.

Описание выбранных методов

Navigating Spreading-out Graph

NSG [6] — это основанный на графе алгоритм приближенного поиска ближайших соседей (ANNS). Он основан на аппроксимации графовой структуры Monotonic Relative Neighborhood Graph (MRNG). NSG устанавливает центральное положение в качестве навигационной вершины, а затем использует определенную стратегию выбора краев для управления степенью отклонения каждой точки. Таким образом, это может уменьшить использование памяти и быстро определить местоположение цели поблизости во время поиска векторов.

Хотя MRNG может гарантировать малое время поиска, его высокое время индексирования непрактично для проблем большого масштаба. Поэтому был предложен практический подход построения приближенного MRNG, названный Navigating Spreading-out Graph (NSG). Алгоритм построения NSG состоит из следующих шагов:

1. Построить kNN граф одним из популярных в данное время методов
2. Найти приближенный медоид датасета:

2.1. Рассчитать центроид датасета

2.2. Принять его за запрос, выполнить поиск по kNN-графу и принять возвращенного ближайшего соседа как приближенный медоид. Эта вершина называется навигационной вершиной, потому что все поиски будут начинаться с этой конкретной вершины

3. Для каждой вершины сгенерировать множество кандидатов в соседи и выбрать соседей из множеств кандидатов. Это можно сделать следующими шагами:

3.1. Для каждой вершины p принять ее за запрос и выполнить поиск по графу, начиная с навигационной вершины на заранее построенном графе kNN.

3.2. Во время поиска каждую посещенную вершину q (то есть такую, расстояние от которой до p рассчитывалось) добавить в множество кандидатов (расстояние также сохраняется)

3.3. Выбрать m ближайших соседей для p из множества кандидатов с помощью стратегии выбора ребер MRNG

4. Выполнить поиск в глубину (Depth-First-Search) [8] на графе, полученном на предыдущих шагах, при этом принять навигационную вершину как корень дерева. Когда DFS завершится проверить, есть ли вершины, которые не были затронуты поиском. Если такие вершины есть, соединить их с их приближенными ближайшими соседями, которые были затронуты поиском, и продолжить DFS.

Стратегия выбора ребер в MRNG принимает все остальные вершины в S как кандидатов в ближайшие соседи текущей вершины, что приводит к большой сложности вычислений по времени. Для ускорения этого процесса в NSG генерируется небольшое множество кандидатов для каждой вершины.

Так как процесс построения точного NNG занимает много времени, используется приближенный kNN граф. В нем допустимо, чтобы только несколько вершин не были соединены со своими ближайшими соседями.

Так как поиск в NSG всегда начинается с навигационной вершины n, для данной вершины p нужно рассматривать только те вершины, которые находятся на пути поиска от m к p. Поэтому p принимается как запрос и выполняется поиск на заранее построенном kNN графе. Вершины, затронутые поиском, и ближайшие соседи вершины p сохраняются как кандидаты (рис. 1). Вершины, формирующие монотонный путь от m до p имеют большой шанс быть добавленными в кандидаты. Затем производится стратегия выбора ребер MRNG на этих кандидатах, и есть большая вероятность, что NSG наследует монотонный путь MRNG от m до p.

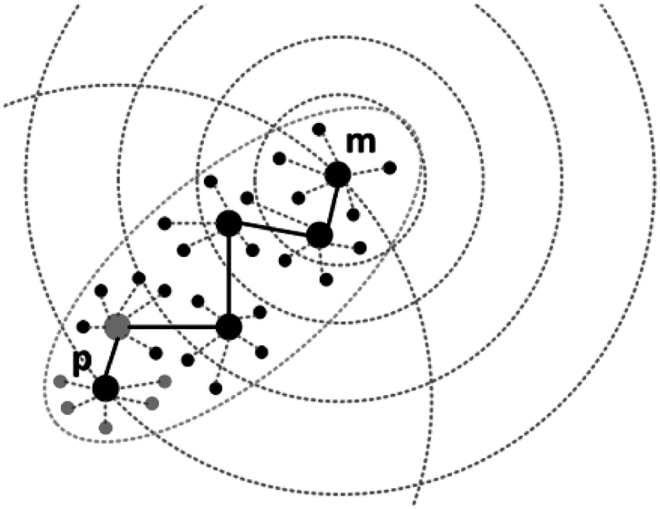

Рис. 1. Кандидаты для выбора ребра в NSG

Для параметров NSG введем следующие обозначения:

N — количество элементов исходного набора данных и количество вершин в графе.

G — объект NSG, хранящий структуру графа.

G имеет следующие входные параметры:

K_conns — количество устанавливаемых соединений на вершину на стадии инициации;

l_constr — размер пула кандидатов для жадного поиска;

m — максимальная полустепень исхода вершины (максимальное количество исходящих соединений). Параметр m вводится для того, чтобы решить проблему с вершинами, имеющими очень много исходящих из них ребер. Количество исходящих ребер для всех вершин ограничено параметром m <<n с помощью удаления длиннейших ребер. Как следствие, после такого вмешательства связность графа больше не гарантируется.

G имеет следующие поля:

G.indexes — список индексов всех вершин;

G.data — список, содержащий векторы, ассоциированные с вершинами;

G.B — список исходящих связей каждой вершины;

G.R — список входящих связей каждой вершины;

G.NNs — общий список соседей вершины;

G.navigating_node — индекс навигационной вершины NSG.

Функция поиска в NSG имеет следующие основные входные параметры:

l — размер пула кандидатов для жадного поиска;

K — количество ближайших соседей для возврата;

В алгоритме NSG указано, что стадию инициализации можно выполнить любым из популярных методов построения аппроксимации kNN графа. Одним из таких методов является HyRec [9][10], и он будет применяться в данной работе в начальной стадии NSG.

HyRec использует подход случайного выбора K вершин как K кандидатов в ближайшие соседи для каждой вершины. Сначала всем вершинам назначаются случайные соседи. Пусть Nu содержит аппроксимацию текущих k ближайших соседей вершины u. Множество кандидатов получается за счет объединения трех множеств:

- Nu — текущая аппроксимация ближайших соседей вершины u
- текущие ближайшие соседи вершин в Nu
- k случайных вершин

Установив конкретный размер множества кандидатов, можно ограничить стоимость расчетов. Использование случайных вершин предотвращает поиск от попадания в локальный минимум. Для NSG можно оставить только одну итерацию для каждой вершины, так как дальнейшее построение по алгоритму NSG исправляет неточности и занимает больше времени.

Hierarchical navigable small world

Hierarchical NSW, HNSW [7] — подход к поиску K ближайших соседей, основанный на графах NSW [11] (Navigable small world) с управляемой иерархией. Данное решение полностью основано на графах, без необходимости каких-либо дополнительных структур для поиска. HNSW инкрементно строит многоуровневую структуру, состоящую из иерархического множества графов близости (уровней, слоев) для подмножеств хранимых элементов.

Идея алгоритма HNSW заключается в том, чтобы разделить связи, основываясь на масштабе их длины, в отдельные уровни, и затем проводить поиск в многоуровневом графе. В данном случае можно оценивать только необходимую часть связей для каждого элемента независимо от размера сети. В такой структуре поиск начинается с верхнего уровня, который имеет только самые длинные связи. Алгоритм жадно следует через элементы верхнего уровня, пока не будет достигнут локальный минимум (рис. 2). После этого поиск перемещается на уровень ниже (который имеет более короткие связи) и начинается заново с того элемента, который был локальным минимумом на прошлом уровне, и процесс повторяется. Максимальное количество связей на элемент на слое может регулироваться константой, что позволяет достичь логарифмической временной сложности поиска в сети NSW.

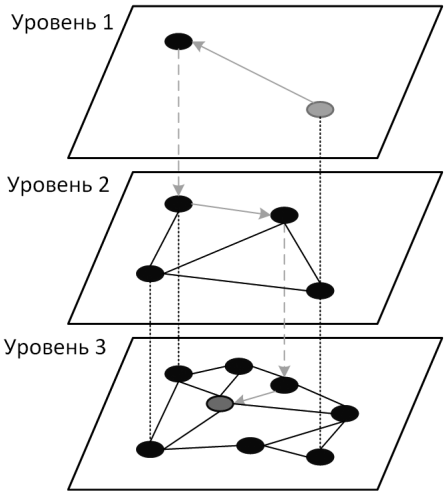

Рис. 2. Поиск в HNSW

Для параметров HNSW введем следующие обозначения:

N — количество элементов исходного набора данных и количество вершин в графе.

G — объект HNSW, хранящий структуру графа.

G имеет следующие входные параметры:

M — количество устанавливаемых соединений на вершину;

$\mathbf{m_L}$ — нормализующий параметр для генерации максимального уровня присутствия элемента;

Mmax — максимальное количество соединений на вершину на уровне (так как при добавлении входящих связей количество соседей может превысить M);

efConstruction — размер динамического списка кандидатов;

G имеет следующие поля:

G.indexes — список индексов всех вершин;

G.data — список, содержащий векторы, ассоциированные с вершинами;

G.NNs — общий список соседей вершины;

G.layers — список, содержащий индексы вершин по уровням HNSW;

G.enter_point — индекс навигационной вершины NSG.

Функция поиска в HNSW имеет следующие основные входные параметры:

K — количество ближайших соседей для возврата;

ef — размер динамического списка кандидатов;

Алгоритм построения графа основан на последовательном добавлении хранимых элементов в граф. Для каждого добавляемого элемента случайно выбирается максимальный уровень (слой) с экспоненциально затухающим распределением вероятности (нормализованным параметром m_L):

$$l \leftarrow \left[-\ln\left(unif\left(0..1\right)\right) \cdot m_L \right]$$

Первая фаза процесса добавления элементов начинается с верхнего уровня, путем жадного поиска по графу, чтобы найти ef ближайших соседей к добавляемому элементу q на слое. После этого, алгоритм продолжает поиск на следующем слое, используя найденных на предыдущем слое ближайших соседей как точки входа, и процесс повторяется.

Ближайшие соседи на каждом слое находятся с помощью варианта алгоритма жадного поиска. Чтобы получить приближенных ef ближайших соседей на уровне l_c, во время поиска сохраняется динамический список W, содержащий ef ближайших найденных элементов (первоначально заполненный точками входа). Список W обновляется на каждом шаге с помощью оценивания соседей ближайшего неоцененного ранее элемента в списке, до тех пор, пока соседи каждого элемента в списке не оказываются оценены. Условие остановки в HNSW позволяет избавляться от кандидатов на оценивание, которые дальше от запроса, чем самый дальний элемент в списке W.

Во время первой фазы поиска параметру ef установлено значение 1 (простой жадный поиск), чтобы избежать добавления лишних параметров.

Когда поиск достигает уровня, который меньше либо равен l, начинается вторая фаза алгоритма построения графа. Вторая фаза отличается от первой следующим: 1) параметр ef увеличивается от 1 до efConstruction, чтобы контролировать метрику recall процедуры жадного поиска; 2) найденные ближайшие соседи на каждом слое также используются как кандидаты ля соединения с добавленным элементом.

Возможно использование двух методов выбора M соседей из кандидатов: простое соединение с ближайшими элементами и эвристика, которая учитывает расстояния между элементами-кандидатами для создания связей в различных направлениях.

Алгоритм поиска приближенных K ближайших соседей, используемый в HNSW, является грубым эквивалентом алгоритма добавления вершин на уровне l = 0. Различие в том, что ближайшие соседи, найденные на нулевом уровне, которые используются как кандидаты для соединений, здесь возвращаются как результат поиска. Качество поиска контролируется параметром ef.

Сравнительный анализ NSG и HNSW

Для сравнения алгоритмов NSG и HNSW были выбраны следующие параметры:

- Source Lines of Code (SLOC) — количество строк кода
- Количество циклов
- Количество условных переходов
- Временная сложность алгоритма в нотации «O» большое Бахмана-Ландау в зависимости от N — количества элементов в исходном наборе данных, для стадии построения и для стадии поиска.

- Емкостная сложность алгоритма в нотации «O» большое Бахмана-Ландау в зависимости N — количества элементов в исходном наборе данных, для стадии построения и для стадии поиска.

Рассчитаем временную и емкостную сложность для алгоритма NSG. Временную сложность в нотации «O» большое от N будем оценивать для стадии построения графа и для стадии поиска.

NSG (G, data, K_conns, l_constr, m) — конструктор объекта NSG, то есть основной алгоритм, связывающий внутренние функции (рис. 3).

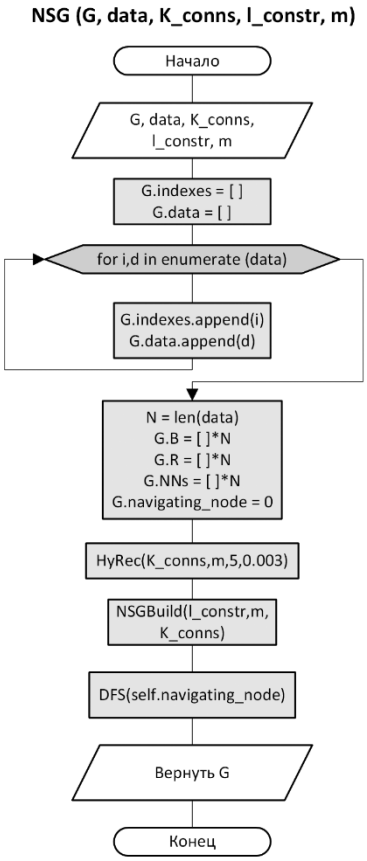

Рис. 3. Общий алгоритм построения NSG

Сначала выполняются две операции присвоения общей сложностью O(1), затем следует цикл по всем элементам набора данных, который повторяется N раз. Внутри цикла каждую итерацию выполняются две операции добавления элемента в конец списка сложностью O(1). Поэтому итоговая сложность цикла составит O(N).

Затем продолжается присвоение значений основным полям объекта, общая сложность этих операций O(N).

Следующий этап — инициализация графа алгоритмом HyRec. Его сложность была вычислена и составляет

$$O(N + d + K\_conns^3 * \log K\_conns)$$

Дальше наступает этап применения стратегии выбора ребер MRNG — алгоритм с определенной сложностью

$$O\left(\begin{array}{c} N * \log N + K\_conns^2 + \\ + l\_constr * \log l\_constr + m^2 + d \end{array}\right).$$

Последний этап — поиск в глубину по графу с целью обеспечения хотя бы одного пути из навигационной вершины. Его сложность оказалась равна

$$O\left(N + +m + d + l\_constr * \log l\_constr\right).$$

Таким образом, **временная сложность построения NSG** от N составляет:

$$NSG_{constr}\left(N\right) = O(N * \log N + K\_conns^3 * \log K\_conns + \\ + d + m^2 + l\_constr * \log l\_constr) = O\left(N * logN\right)$$

Это значит, что функция времени построения от N в NSG будет расти не быстрее, чем $N * \log N * Const$, где Const — некоторая константа.

Поиск по графу NSG использует в качестве параметров размер списка кандидатов l и количество ближайших соседей для поиска K. **Временная сложность поиска по NSG** была рассчитана и равна:

$$NSG_{search}\left(N\right) = O\left(\begin{array}{c} \log N + l * \log l + \\ + m + d + \log K \end{array}\right) = O(\log N)$$

Емкостная сложность алгоритма NSG в нотации «O» большое от N можно оценить следующим образом: максимальная по размеру создаваемая алгоритмом построения структура данных, зависящая от N, это список соседей размером $N * m$, что означает, что емкостная сложность построения и поиска по NSG будет составлять O(N). Однако, при построении NSG использует еще 5 структур примерной длиной N, помимо структуры длиной $N * m$.

Рассчитаем временную и емкостную сложность для алгоритма HNSW:

Временную сложность в нотации «O» большое от N будем оценивать для стадии построения графа и для стадии поиска.

HNSW (G, data, M, mL, Mmax, efConstruction, select) — конструктор объекта HNSW, то есть основной алгоритм, связывающий все внутренние этапы (рис. 4).

Во время выполнения функции выполняется несколько операций присваивания значений полям класса сложностью O(1). Помимо этого, выполняются два цикла cycle1 и cycle2.

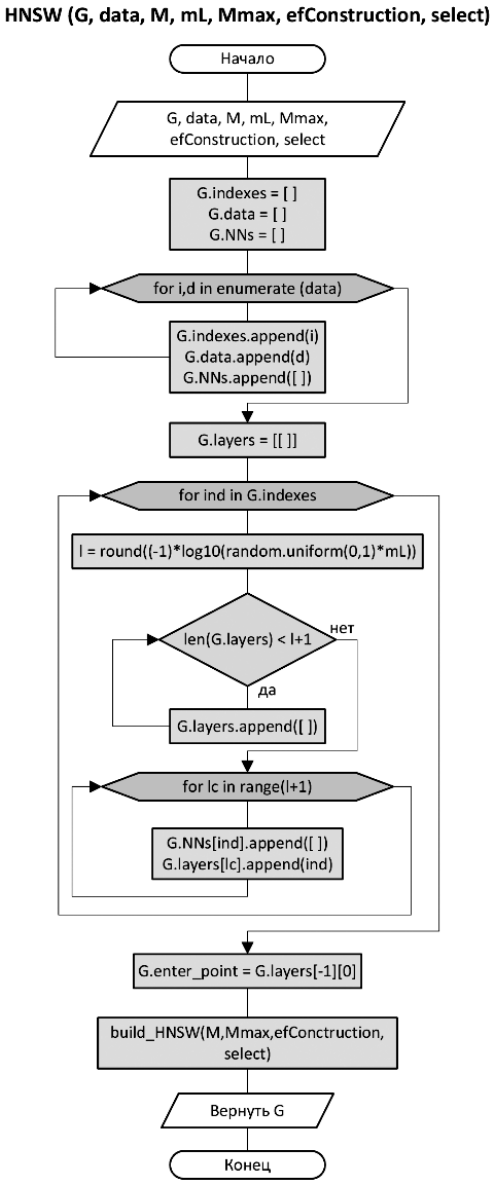

Рис. 4. Общий алгоритм построения HNSW

Цикл cycle1 N раз повторяет операции сложностью O(1), поэтому его сложность составит O(N).

Цикл cycle2 повторяется также N раз. Внутри цикла выполняется одна операция присвоения сложностью O(1) и два вложенных цикла cycle2.1 и cycle2.2. Оба цикла выполняются малое число раз, которое можно принять за константу, которая зависит от mL и много меньше N Const << N, при этом внутри них выполняются операции сложностью O(1). Получаем, что сложность цикла cycle2 составит O(N).

После всех циклов происходит вызов функции построения графа с определенной сложностью $O(N * \log N)$.

Таким образом, **временная сложность алгоритма построения HNSW** равна:

$$HNSW_{constr}(N) = O(2 * N + N * (\log N + d + M + \\ + Mmax * \log Mmax + efConstruction * \log efConstruction)) = \\ = O(2 * N + \log N + d + M + Mmax * \log Mmax + \\ + efConstruction * \log efConstruction) = \\ = O(2 * N + N*\log N) = (N*\log N)$$

Поиск по графу HNSW содержит несколько операций общей сложностью O(1), после которых следует цикл cycle1, повторяющийся число раз, не зависящее от N и много меньшее N, то есть Const << N число раз. На каждой итерации цикл вызывает операцию поиска по слою сложностью $O(\log N)$. Поэтому, временная сложность цикла cycle 1 составит $O(\log N)$.

После цикла следует еще одна операция поиска по слою сложностью $O(\log N)$, операция выбора ближайшего элемента сложностью $O(ef * \log ef + d)$ и выбор соседей из списка кандидатов сложностью $O(d + ef * \log ef + K)$.

Таким образом, **временная сложность алгоритма поиска по HNSW** составляет:

$$HNSW_{search} = O(\log N + ef * \log ef + d + K) = \\ = O(\log N)$$

Емкостная сложность алгоритма HNSW в нотации «O» большое от N можно оценить следующим образом: максимальная по размеру создаваемая алгоритмом построения структура данных, зависящая от N, это список соседей размером $N * num\_layers * Mmax$, что означает, что емкостная сложность построения и поиска по HNSW будет составлять O(N). На стадии построения HNSW использует еще 2 структуры длиной N, помимо структуры длиной $N * num\_layers * Mmax$, что позволяет предположить, что при одинаковой временной сложности HNSW будет использовать меньше памяти на стадии построения, чем NSG (который, как было показано ранее, использует 5 дополнительных структур длиной N).

Таким образом, в ходе выполненных вычислений была получена оценка временных и емкостных сложностей алгоритмов NSG и HNSW (Таблица 1).

Также были оценены количественные характеристики для конкретной реализации на Python 3.8:

- NSG: Source Lines of Code (SLOC) = всего 197, для поиска 50.
- NSG: Количество циклов = 21.
- NSG: Количество условных переходов = 21.
- HNSW: Source Lines of Code (SLOC) = всего 257, для поиска 99.
- HNSW: Количество циклов = 22.
- HNSW: Количество условных переходов = 21.

Таблица 1.

Оценка временных и емкостных сложностей алгоритмов NSG и HNSW

Алгоритм	Временная сложность построения	Временная сложность поиска	Емкостная сложность построения	Емкостная сложность поиска
NSG	$O(N*\log N + d + m^2 + K\_conns^3 * \log K\_conns + l\_constr * \log l\_constr) = O(N*\log N)$	$O(\log N + l*\log l + m + d + \log K) = O(\log N)$	O(N)	O(N)
HNSW	$O(N*\log N + d + M + Mmax * \log Mmax + efConstruction * \log efConstruction) = O(N*\log N)$	$O(\log N + ef*\log ef + d + K) = O(\log N)$	O(N)	O(N)

На основе полученной информации можно сделать предварительный вывод о том, что NSG при примерно той же временной сложности будет выполнять стадии поиска и построения быстрее, так как содержит меньше строк кода и операторов. Стадии поиска у обоих методов должны использовать примерно одинаковое количество памяти, в то время как стадия построения у HNSW будет менее затратной по памяти, так как в алгоритме используется меньше структур данных длиной *N*.

Экспериментальное исследование

Для подтверждения полученных результатов расчета временных и емкостных сложностей алгоритмов NSG и HNSW были проведены тесты конкретных реализаций алгоритмов на языке Python 3.8 на различных объемах входных данных *N*. Данные представляют собой синтетический набор векторов размерностью 25, разделенных на 4 класса.

В ходе тестов замерялись следующие показатели:

$\boldsymbol{t_{constr}}$ — время построения графа, сек;

$\boldsymbol{t_{search}}$ — время поиска по графу, сек;

$\boldsymbol{peak\_memory_{constr}}$ — максимальная использованная память во время построения графа, Мб;

$\boldsymbol{peak\_memory_{search}}$— максимальная использованная память во время поиска по графу, Мб;

Результаты тестов HNSW представлены в таблице 2. Для всех *N* параметры HNSW были следующими: M = 3, Mmax = 6, efConstruction = ef = 7, $\mathbf{m_L}$ = 0,91, K = 5.

На рис. 5 представлена зависимость t_{constr} от *N* для HNSW. Можно видеть, что функция $t_{constr}(N)$ действительно растет не быстрее, чем N * log*N* * *Const*, в данном случае Const = 0,08. Теоретическая временная сложность построения HNSW подтверждена.

На рис. 6 представлена зависимость t_{search} от *N* для HNSW. Можно видеть, что функция $t_{search}(N)$ действительно растет не быстрее, чем log*N**Const, в данном случае Const = 0,0035. Теоретическая временная сложность поиска в HNSW подтверждена.

Таблица 2.

Результаты тестов HNSW

N	t_{constr}, с	t_{search}, с	$peak\_memory_{constr}$ Мб	$peak\_memory_{search}$ Мб
10	0,0274	0,00950	0,02904	0,02904
50	0,7391	0,01719	0,02994	0,02994
100	1,5150	0,02210	0,03113	0,03590
200	3,7662	0,02357	0,03191	0,03191
500	11,8683	0,02747	0,03210	0,03210
800	22,6070	0,03030	0,03210	0,03210
1000	33,0528	0,03101	0,03206	0,03206
1500	60,5570	0,03265	0,03812	0,03812
2000	89,4688	0,03315	0,04182	0,04182
2500	105,6203	0,03471	0,04643	0,04643
3000	133,0710	0,03337	0,04661	0,04661
3500	167,4600	0,03317	0,05419	0,05419
4000	189,2060	0,03450	0,05813	0,05813
4500	223,3377	0,03407	0,06222	0,06222
5000	281,3915	0,03317	0,06252	0,06252
5500	317,5270	0,03503	0,06910	0,06910
6000	358,7515	0,03619	0,06997	0,06997

На рис. 7 представлена зависимость $peak\_memory_{constr}$ от *N* для HNSW. Можно видеть, что функция $peak_memory_{constr}(N)$действительно растет не быстрее, чем *N**Const, в данном случае Const = 0,001. Теоретическая емкостная сложность построения HNSW подтверждена.

На рис. 8 представлена зависимость $peak\_memory_{search}$ от *N* для HNSW. Можно видеть, что функция $peak_memory_{search}(N)$действительно растет не быстрее, чем *N**Const, в данном случае Const = 0,00001. Теоретическая емкостная сложность поиска в HNSW подтверждена.

Результаты тестов NSG представлены в таблице 3. Для всех *N* параметры NSG были следующими: K_conns = 3, m = 6, l_constr = l = 7, K =5.

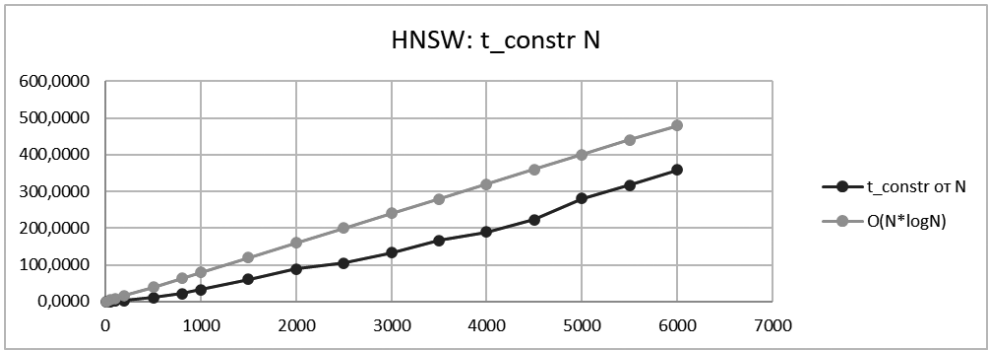

Рис. 5. t_{constr} от N для HNSW

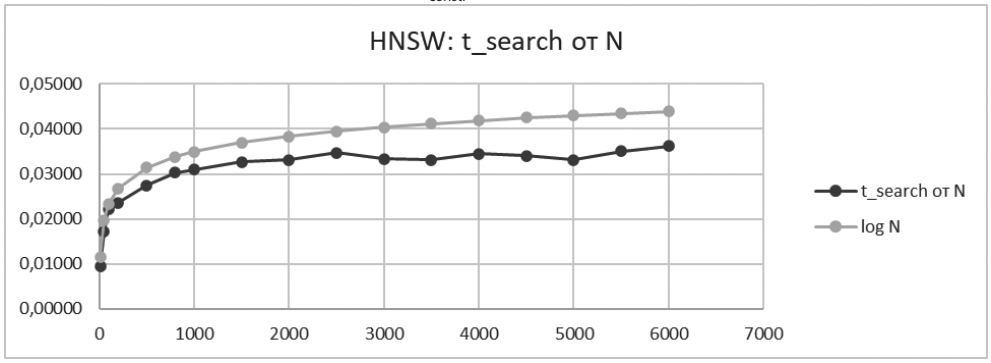

Рис. 6. t_{search} от N для HNSW

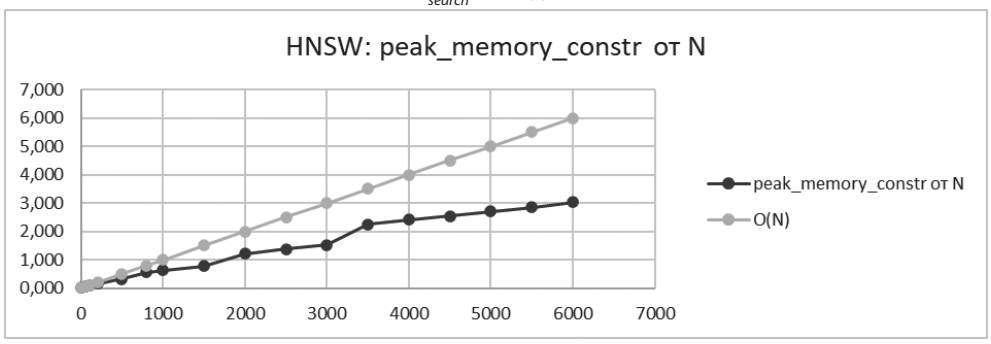

Рис. 7. $peak\_memory_{constr}$ от N для HNSW

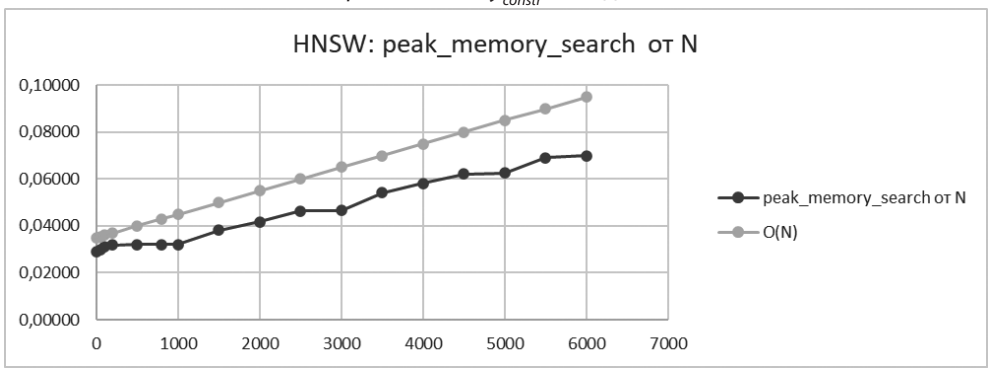

Рис. 8. $peak\_memory_{search}$ от N для HNSW

На рис. 9 представлена зависимость t_{constr} от N для NSG. Можно видеть, что функция $t_{constr}(N)$ действительно растет не быстрее, чем N*log$N$*Const, в данном случае Const = 0,08. Теоретическая временная сложность построения NSG подтверждена.

На рис. 10 представлена зависимость t_{search} от N для NSG. Можно видеть, что функция $t_{search}(N)$ действительно растет не быстрее, чем log$N$*Const, в данном случае Const = 0,0015. Теоретическая временная сложность поиска в NSG подтверждена.

Таблица 3.

Результаты тестов NSG

N	t_{constr}, с	t_{search}, с	$peak\_memory_{constr}$, Мб	$peak\_memory_{search}$, Мб
10	0,0342	0,00100	0,0409	0,02904
50	0,2784	0,00171	0,068656	0,02994
100	0,5658	0,00312	0,141962	0,03113
200	1,2963	0,00340	0,254479	0,03191
500	5,3650	0,00478	0,529235	0,03210
800	6,9280	0,00545	0,951509	0,03210
1000	10,6170	0,00619	1,051904	0,03206
1500	18,0330	0,00696	1,858819	0,03812
2000	30,3730	0,00768	2,076249	0,04182
2500	27,8749	0,00770	3,602522	0,04643
3000	44,4270	0,00872	3,855196	0,04661
3500	52,1270	0,00931	4,0725	0,05419
4000	69,4840	0,01001	4,29952	0,05813
4500	76,4710	0,01050	4,620226	0,06222
5000	85,5640	0,01170	7,340464	0,06252
5500	86,4210	0,01150	7,546114	0,06910
6000	124,9070	0,01090	7,838858	0,06997

На рис. 11 представлена зависимость $peak\_memory_{constr}$ от N для NSG. Можно видеть, что функция $peak\_memory_{constr}(N)$ действительно растет не быстрее, чем N*Const, в данном случае Const = 0,002. Теоретическая емкостная сложность построения NSG подтверждена.

На рис. 12 представлена зависимость $peak\_memory_{search}$ от N для NSG. Можно видеть, что функция $peak\_memory_{search}(N)$ действительно растет не быстрее, чем N*Const, в данном случае Const = 0,00001. Теоретическая емкостная сложность поиска в NSG подтверждена.

Подтвердились теоретические предположения о том, что NSG будет выполнять поиск и построение быстрее, чем HNSW (рис.13 — рис.14).

Память, используемая при поиске, оказалась примерно равной у обоих методов (рис. 15).

Использование памяти во время построения для NSG оказалось выше, чем для HNSW (рис. 16).

Таким образом, можно сделать вывод, что NSG более предпочтителен для работы с большими объемами данных, так как он выполняет стадии построения и поиска намного быстрее, чем HNSW. Однако, стоит учитывать большее использование памяти на этапе построения NSG и удостовериться, что ЭВМ обладает достаточными объемами оперативной памяти.

Заключение

Методы поиска ближайшего соседа, основанные на графах, имеют большое преимущество в сравнении с другими методами KNNS в отношении времени поиска ближайших соседей по графу. При правильно построен-

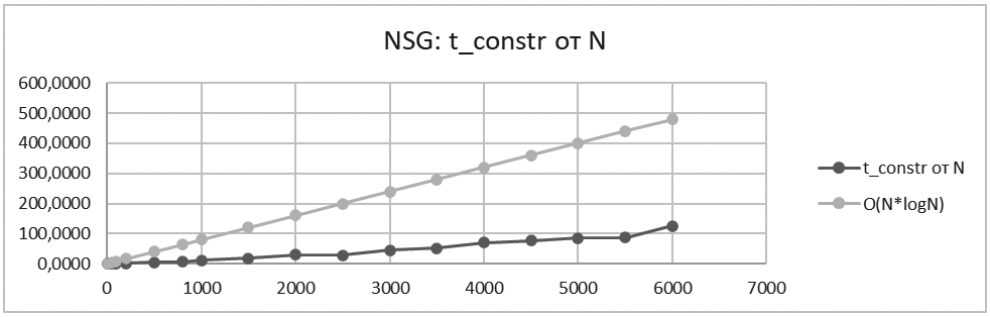

Рис. 9. t_{constr} от N для NSG

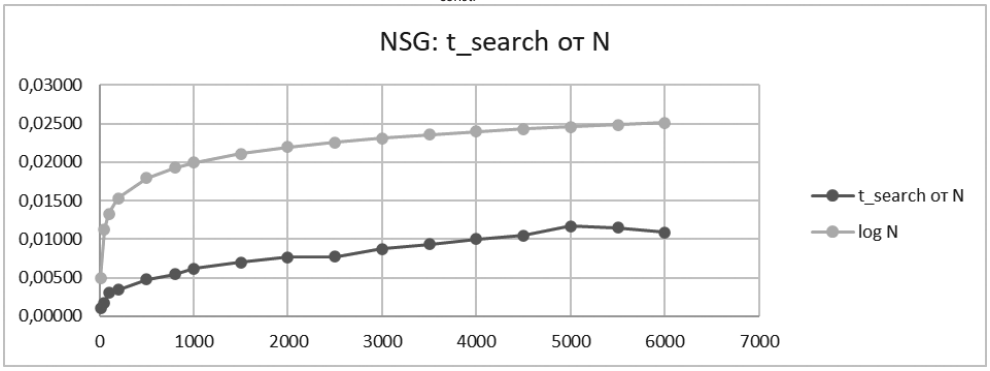

Рис. 10. t_{search} от N для NSG

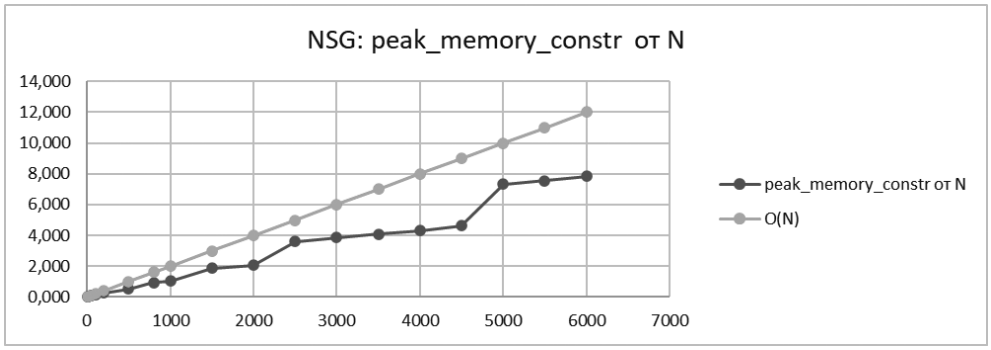

Рис. 11. $peak\_memory_{constr}$ от N для NSG

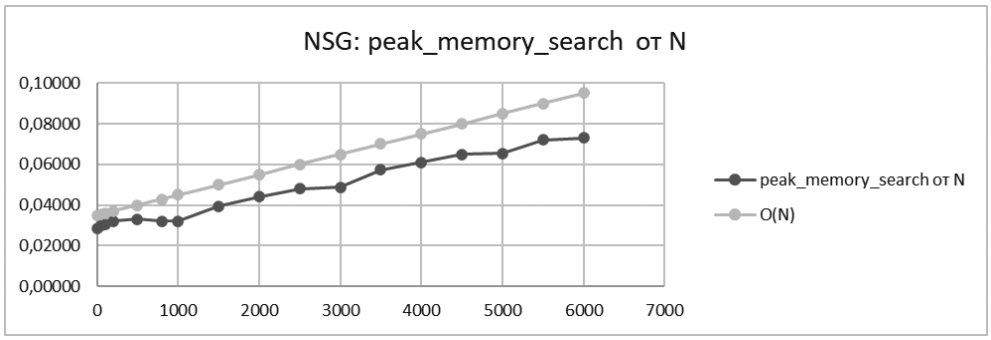

Рис. 12. $peak\_memory_{search}$ от N для NSG

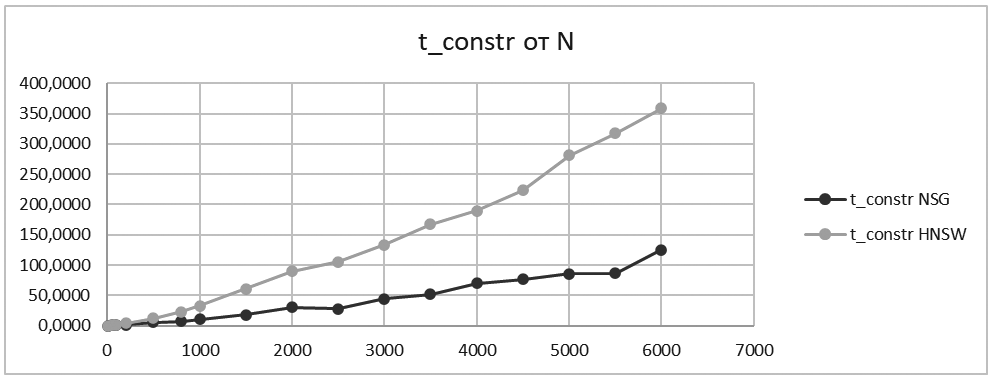

Рис. 13. t_{constr} от N для NSG и HNSW

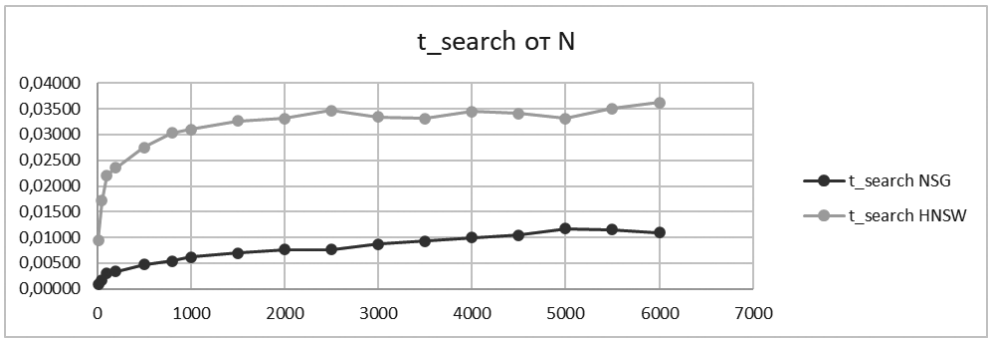

Рис. 14. t_{search} от N для NSG и HNSW

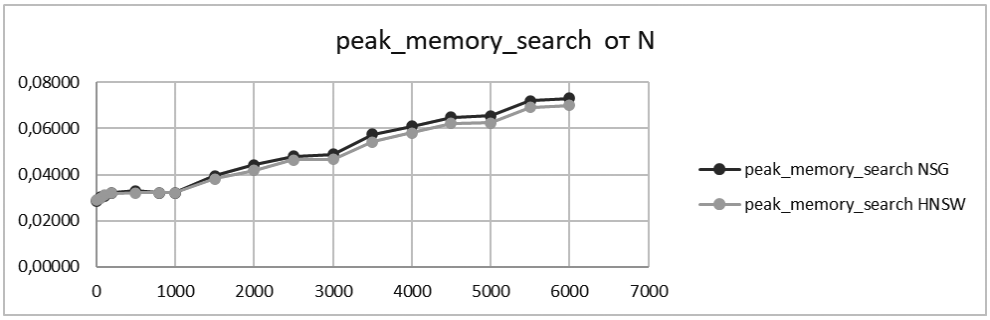

Рис. 15. *peak_memory*$_{search}$ от *N* для NSG и HNSW

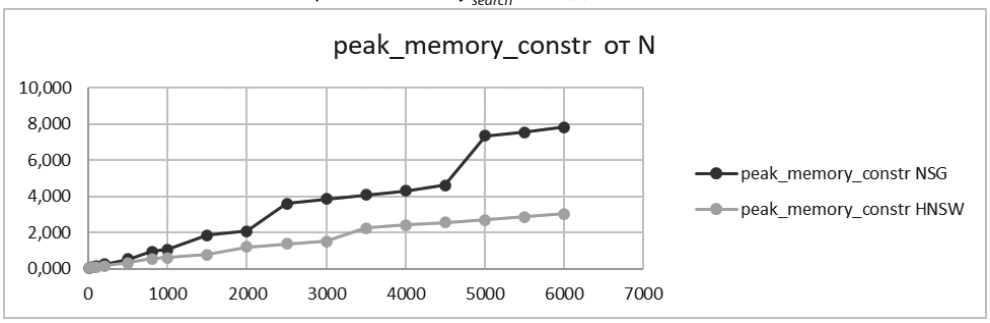

Рис. 16. *peak_memory*$_{constr}$ от *N* для NSG и HNSW

ной структуре гарантируется высокая точность такого поиска, хоть и во многих подобных методах стадия построения графа является очень затратной по времени.

В ходе выполнения работы были реализованы два метода ANNS, основанных на графах — HNSW и NSG. HNSW использует иерархическую структуру, состоящую из уровней (слоев), которая обеспечивает более эффективный поиск благодаря разделению связей по масштабу. NSG основан на аппроксимации графа MRNG, использующей поиск из навигационной вершины для коррекции всех связей в графе с помощью стратегии выбора ребер MRNG.

Было проведено теоретическое сравнение данных методов, которое показало, что оба алгоритма имеют временную сложность построения графа O(*N**log*N*), временную сложность поиска в графе O(log*N*) и емкостные сложности построения и поиска O(*N*). Был сделан предварительный вывод о том, что несмотря на одинаковую скорость роста времени выполнения от количества входных данных, NSG окажется быстрее HNSW, так как использует меньшее количество операторов. В то же время было отмечено, что NSG на стадии построения использует больше структур данных размером *N*, чем HNSW, что позволило предположить, что NSG будет использовать больше памяти при построении графа. Теоретические выводы были подтверждены тестами реализованных методов на синтетических данных.

Подытоживая результаты работы, можно сделать вывод, что NSG является более предпочтительным вариантом в случае больших объемов данных, но использует больше памяти для стадии построения графа.

ЛИТЕРАТУРА

1. K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. Proceedings of the International Joint Conference on Artificial Intelligence, 22:1312–1317, 2011
2. Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He. Fast and accurate hashing via iterative nearest neighbors expansion. IEEE transactions on cybernetics, 44(11):2167–2177, 2014.
3. C. Fu and D. Cai. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. arXiv:1609.07228, 2016.
4. H. Ben and D. Tom. FANNG: Fast approximate nearest neighbour graphs. In Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition, pages 5713–5722, 2016.
5. S. Arya and D.M. Mount. Approximate nearest neighbor queries in fixed dimensions. In Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms, pages 271–280, 1993.
6. Fu C. et al. Fast approximate nearest neighbor search with the navigating spreading-out graph //arXiv preprint arXiv:1707.00143. — 2017.
7. Malkov Y.A., Yashunin D.A. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs //IEEE transactions on pattern analysis and machine intelligence. — 2018. — Т. 42. — №. 4. — С. 824–836.
8. Even, Shimon (2011), Graph Algorithms (2nd ed.), Cambridge University Press, pp. 46–48.
9. Antoine Boutet, Davide Frey, Rachid Guerraoui, Anne-Marie Kermarrec, and Rhicheek Patra. Hyrec: leveraging browsers for scalable recommenders. In Middleware, 2014.
10. Olivier Ruas. The many faces of approximation in KNN graph computation. Machine Learning [cs.LG]. Université de rennes 1, 2018. English.
11. Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, «Approximate nearest neighbor algorithm based on navigable small world graphs,» Information Systems, vol. 45, pp. 61–68, 2014.