ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KĨ THUẬT MÁY TÍNH

BÁO CÁO BÀI TẬP LỚN HỌC PHẦN MỞ RỘNG CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT

Tìm kiếm gần đúng bằng đồ thị HNSW
(Hierarchical Navigable Small World)

GVHD: TS.Lê Thành Sách

STT	MSSV	Họ	Tên	Ghi chú
1	2411261	Phạm Quốc	Huy	
2	2411205	Nguyễn Đình	Huy	
3	2411278	Trần Quang	Huy	

Tháng 06/2025

MỤC LỤC

LỜI NÓI ĐẦU

Từ thuở sơ khai của máy tính cá nhân và Internet, các hệ thống tìm kiếm (search engine) luôn là một ứng dụng quan trọng khi người dùng có nhu cầu tìm kiếm thứ gì đó trên Internet, với đại diện tiêu biểu mà phần lớn người dùng trên thế giới đều trải nghiệm qua, đó là Google. Hay Youtube, không phải một cách ngẫu nhiên mà Youtube luôn cho ra những video đề xuất đúng với thứ mà người dùng cần tìm khi nhập vào thanh tìm kiếm. Từ những phép so sánh chuỗi giống nhau để xuất ra kết quả tìm kiếm, người ta bắt đầu sử dụng các kĩ thuật hiện đại hơn để dễ dàng "hiểu ý"người dùng. Từ đó mà cơ sở dữ liệu vector (vector database) ra đời.

Ngày nay, search engines đóng vai trò không nhỏ trong đời sống của mỗi cá nhân. Các hệ thống này càng ngày càng hiểu ý người dùng và đưa cho ta những câu trả lời cho những gì mà ta nhập vào. Các thuật toán tìm kiếm tương tự mới (similarity search algorithms) tương tác với vector database cũng ra đời để làm cho truy vấn của người dùng, trong thời gian ngắn nhất, đến được với những dữ liệu gần với truy vấn nhất. Cùng với sự phát triển của trí tuệ nhân tạo (Artificial Intelligence - AI) mà trên thị trường, doanh nghiệp nào có các thuật toán càng hiện đại, càng nhanh, càng chính xác, thì sẽ càng thu hút người dùng và càng có được nguồn doanh thu càng lớn.

Thấy được tầm quan trọng và ứng dụng vào doanh nghiệp của các thuật toán tìm kiếm tương tự, cụ thể ở đây là thuật toán dựa trên cấu trúc đồ thị HNSW, nhóm sinh viên chọn đề tài này để nghiên cứu và phát triển.

Nhóm sinh viên.

LỜI CẢM ƠN

Nhóm sinh viên gửi lời cảm ơn sâu sắc đến giảng viên hướng dẫn - TS.Lê Thành Sách, vì đã truyền tải những kiến thức quý báu về các vấn đề liên quan và hỗ trợ nhóm thực hiện bài báo cáo cho học phần mở rộng này.

Cảm ơn ban lãnh đạo Trường Đại học Bách khoa - ĐHQG TP.HCM vì đã đưa học phần mở rộng vào chương trình tài năng của các sinh viên, góp phần rất lớn giúp nhóm sinh viên nâng cao các kĩ năng chuyên môn để áp dụng cho nghề nghiệp sau này.

Cảm ơn tập thể các sinh viên chương trình tài năng cùng lớp trong học phần mở rộng vì đã đưa ra các phản biện quý báu, qua đó giúp cho bài báo cáo của nhóm sinh viên hoàn thiện hơn.

Nhóm sinh viên ý thức rằng, với kinh nghiệm còn hạn chế và kiến thức chưa sâu rộng, bài làm của nhóm chắc chắn không tránh khỏi những thiếu sót. Rất mong nhận được những ý kiến đóng góp, nhận xét quý báu từ giảng viên hướng dẫn và bạn đọc để báo cáo của nhóm được hoàn thiện hơn.

Nhóm sinh viên.

CHƯƠNG 1: MỞ ĐẦU

I MỤC TIÊU

Trong bài tập lớn này, nhóm sinh viên nghiên cứu xây dựng hệ thống tìm kiếm vector gần đúng (Approximate Nearest Neighbor – ANN) sử dụng cấu trúc đồ thị HNSW – một trong những thuật toán tiên tiến nhất đang được sử dụng rộng rãi trong các hệ thống vector database hiện đại như FAISS, Milvus, Weaviate. Đề tài giúp sinh viên hiểu rõ nguyên lý tổ chức dữ liệu bằng đồ thị phân tầng, cơ chế tìm kiếm tham lam (greedy), cũng như khả năng mở rộng và ứng dụng thực tiễn của HNSW.

II YÊU CẦU

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

Trong phần này, nhóm sinh viên nghiên cứu các nền tảng liên quan trong lĩnh vực khoa học máy tính để xây dựng thuật toán tìm kiếm bằng đồ thị HNSW.

I CÁC VẤN ĐỀ NỀN TẢNG

Khi những hệ thống tìm kiếm đầu tiên ra đời, ý tưởng của các nhà phát triển là làm một phép so sánh. Các doanh nghiệp sẽ có một cơ sở dữ liệu về các website, là tập hợp của các từ, câu, đoạn văn, người dùng nhập các từ cần tìm vào, đầu vào này sẽ được đem so sánh với các dữ liệu có sẵn trong cơ sở dữ liệu, và các kết quả trùng khớp sẽ được hiển thị. Hiển nhiên các doanh nghiệp cũng có một tập các từ đồng nghĩa (ví dụ như "xe hơi"và "xe 4 bánh"đều cho ra cùng một kết quả khi tìm kiếm). Mô hình như thế được gọi là lexical similarity search (tạm dịch là tương đồng về từ ngữ). Tuy nhiên, sẽ ra sao nếu doanh nghiệp không cập nhật tập các từ đồng nghĩa? Hơn nữa, ta không chỉ tìm kiếm từ ngữ. Đôi lúc ta cũng cần tìm kiếm hình ảnh (phổ biến là Google Lens). Do đó, ta cần xây dựng một hệ thống tìm kiếm mới hiện đại hơn, thông minh hơn, hiểu ý người dùng hơn. Từ đó mà semantic similarity search (tìm kiếm ngữ nghĩa) ra đời.

Nền tảng của hệ thống này là ta sẽ mã hóa tất cả các dữ liệu mà người dùng nhập vào, cũng như các dữ liệu có sẵn, thành dãy các con số dưới dạng một vector nhiều chiều. Vector embedding là một vector khi ta sử dụng kĩ thuật embedding để đưa vector ban đầu, thưa, về một vector có số chiều bé hơn, dày hơn [1]. Ngày nay, từng kiểu dữ liệu (như ảnh, văn bản, audio) thường có thể được đưa về các vector embedding. Vector database là một cơ sở dữ liệu mà ta lưu các vector embeddings đó [2]. Thông thường, 512 là số chiều của các vector mà nhà phát triển sử dụng.

Quá trình xác định giá trị tương ứng với mỗi chiều của một vector được gọi là trích xuất đặc trưng (feature extraction hay feature engineering) [3]. Trong quá trình làm giảm số chiều, ta giữ lại các đặc trưng mà có tính quyết định lớn đến sự phân biệt giữa các điểm dữ liệu. Quá trình này được gọi là chọn lọc đặc trưng (feature selection) [4].

Bài toán được đặt ra rằng đâu là thuật toán tối ưu nhất để trả ra các vector có độ tương thích cao nhất với vector đầu vào. Ban đầu các nhà phát triển nghĩ rằng ta đi so sánh đầu vào p với các vector $q \in \mathbb{R}^n$ trong cơ sở dữ liệu, kết quả trả ra khi $p = q$. Tuy nhiên, điều này thường không khả thi vì hiếm khi các trường dữ liệu của hai vector được so sánh hoàn toàn bằng nhau, nhất là khi n càng lớn. Do đó, ta chỉ có thể tìm ra k vector có mức độ tương tự cao nhất khi so sánh với đầu vào. Bài toán k-NN (k láng giềng gần nhất) có thể được phát biểu: Cho N vector nhiều chiều $D = u_1, u_2, ..., u_N$ và một vector truy vấn q cùng số chiều, bài toán k-NN truy vấn

một tập $kNN(D, q, k)$ gồm k vector sao cho $\forall u \in kNN(D, q, k)$ và $v \in D \setminus kNN(D, q, k)$, $dis(q, u) \leq dis(q, v)$ trong đó $dis(\cdot, \cdot)$ là toán tử khoảng cách giữa hai vector toán hạng [5].

Về vấn đề khoảng cách giữa hai vector, có nhiều hàm tính khoảng cách phù hợp trong từng ngữ cảnh nhất định. Các biểu thức 2.1, 2.2, 2.3 lần lượt biểu diễn khoảng cách Euclide, Manhattan, và cosine giữa hai vector. Trong ngữ cảnh các bài toán liên quan đến vector embeddings, chiều dài của các vector không quá chênh lệch, do đó ta quan tâm đến sự tương đồng giữa hai vector thông qua góc của chúng. Do đó, từ phần này về sau, khoảng cách giữa hai vector luôn được ngầm hiểu là khoảng cách cosine. Giá trị này càng bé thì hai vector đầu vào càng tương đồng nhau và ngược lại.

$$d(p, q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{2.1}$$

$$d(p, q) = \sum_{i=1}^{n} |p_i - q_i| \tag{2.2}$$

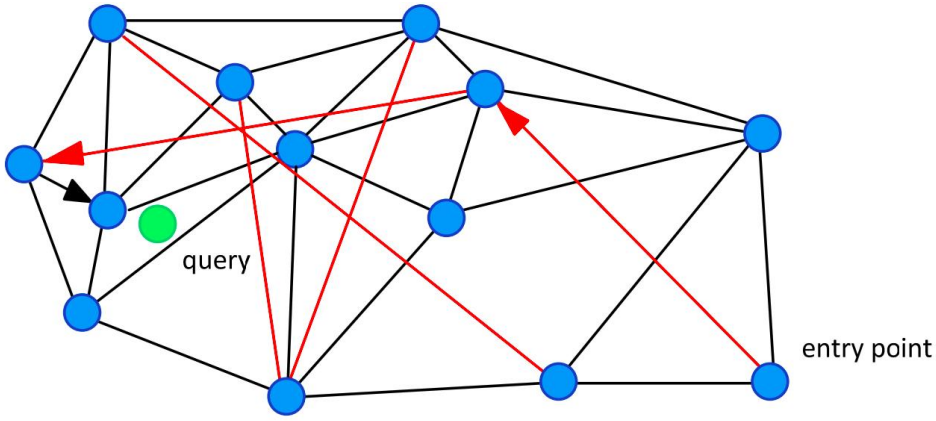
$$d(p, q) = 1 - \frac{\sum_{i=1}^{n} p_i \times q_i}{\sqrt{\sum_{i=1}^{n} p_i^2} + \sqrt{\sum_{i=1}^{n} q_i^2}} \tag{2.3}$$

II THUẬT TOÁN TÌM KIẾM DỰA TRÊN ĐỒ THỊ HNSW

1 NSW (Navigable Small World) và giải thuật tham lam (greedy algorithm)

Một ý tưởng được đặt ra là ta mô hình hóa các vector trong vector database dưới dạng một đồ thị $G(V, E)$ mà ở đó mỗi vector $v \in V$ đại diện cho một đỉnh, mỗi cạnh $e \in E$ của đồ thị thể hiện khoảng cách giữa hai vector ở hai đỉnh, và mỗi vector chỉ kết nối với một số lượng nhất định các vector có khoảng cách gần nhất [6]. Ở hình 2.1, các vòng tròn xanh (các đỉnh) là các vector trong không gian, các cạnh đen là các kết nối giữa hai đỉnh có khoảng cách gần nhau, các đường màu đỏ là các kết nối giữa các điểm dữ liệu xa nhau, đảm bảo độ tăng trưởng logarit (rất chậm) khi dữ liệu tăng đáng kể, các mũi tên cho thấy đường đi theo giải thuật tham lam từ điểm ban đầu đến điểm gần với truy vấn. Thuật toán 1 biểu diễn mã giả của giải thuật tham lam được sử dụng [6]. Giải thuật nhận vào hai tham số: truy vấn q và điểm bắt đầu $V_{entry\_point} \in V$. Bắt đầu từ điểm bắt đầu, giải thuật tính toán khoảng cách từ q đến các lân cận của đỉnh hiện tại, sau đó chọn đỉnh có khoảng cách ngắn nhất. Nếu khoảng cách bé nhất từ q đến các lân cận này bé hơn khoảng cách từ q đến điểm hiện tại, ta di chuyển đến lân cận này. Chương trình dừng khi không tìm được lân cận nào có khoảng cách đến q gần hơn điểm hiện tại, và điểm hiện tại chính là kết quả cần tìm. Thuật toán 2 biểu diễn quá trình tìm ra

top-k vector gần với truy vấn q nhất [6]. Quá trình tìm kiếm tham lam trong đồ thị NSW được gọi là zoom-out[7].



Hình 2.1: Biểu diễn đồ thị của cấu trúc NSW [6]

Algorithm 1 Greedy Search Algorithm

function GREEDY_SEARCH(q, $V_{\text{entry\_point}}$)
    $V_{\text{curr}} \leftarrow V_{\text{entry\_point}}$
    $\delta_{\text{min}} \leftarrow \delta(q, V_{\text{curr}})$
    $V_{\text{next}} \leftarrow$ NIL
    for all $V_{\text{friend}} \in V_{\text{curr}}$.getFriends() **do**
        $\delta_{\text{fr}} \leftarrow \delta(q, V_{\text{friend}})$
        if $\delta_{\text{fr}} < \delta_{\text{min}}$ **then**
            $\delta_{\text{min}} \leftarrow \delta_{\text{fr}}$
            $V_{\text{next}} \leftarrow V_{\text{friend}}$
        end if
    end for
    if $V_{\text{next}} =$ NIL **then**
        return V_{curr}
    else
        return GREEDY_SEARCH(q, V_{next})
    end if
end function

Độ phức tạp về mặt thời gian của các phép toán dựa trên đồ thị NSW tăng trưởng theo lũy thừa của logarit. Cụ thể, phép tìm kiếm và chèn cho thấy độ phức tạp là $O(\log^2 N)$, và phép toán khởi tạo có độ phúc tạp là $O(N \log^2 N)$ [6].

Tuy nhiên, giải thuật tham lam dựa trên đồ thị NSW mắc một nhược điểm nghiêm trọng: nó dễ bị mắc kẹt trong các cực tiểu địa phương mà chưa kịp đi đến cực tiểu toàn cục. Vì vậy mà tác giả của [6] đã phát triển nên hierarchical NSW (HNSW) để khắc phục nhược điểm này.

Algorithm 2 K-NN Search

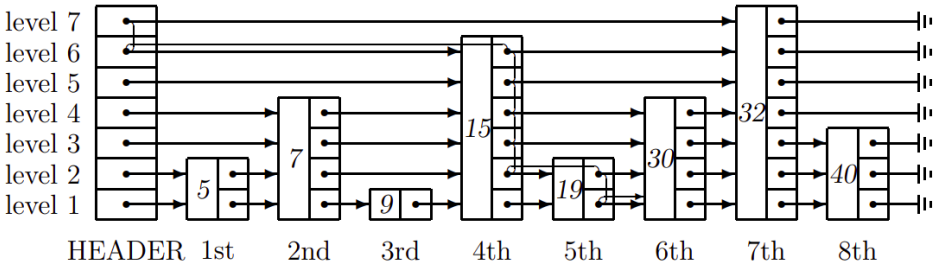
```
1:  procedure K-NNSEARCH(q, m, k)
2:      TreeSet tempRes, candidates, visitedSet, result
3:      for i ← 1 to m do
4:          Put a random entry point into candidates
5:          tempRes ← ∅
6:          loop
7:              c ← get element from candidates closest to q
8:              Remove c from candidates                    ▷ Check stop condition
9:              if c is further from q than the k-th element in result then
10:                 break
11:             end if                                      ▷ Update list of candidates
12:             for all element e in friends of c do
13:                 if e ∉ visitedSet then
14:                     Add e to visitedSet, candidates, and tempRes
15:                 end if
16:             end for
17:         end loop                                        ▷ Aggregate the results
18:         Add objects from tempRes to result
19:     end for
20:     return best k elements from result
21: end procedure
```

2 HNSW (Hierarchical Navigable Small World) và đồ thị phân tầng

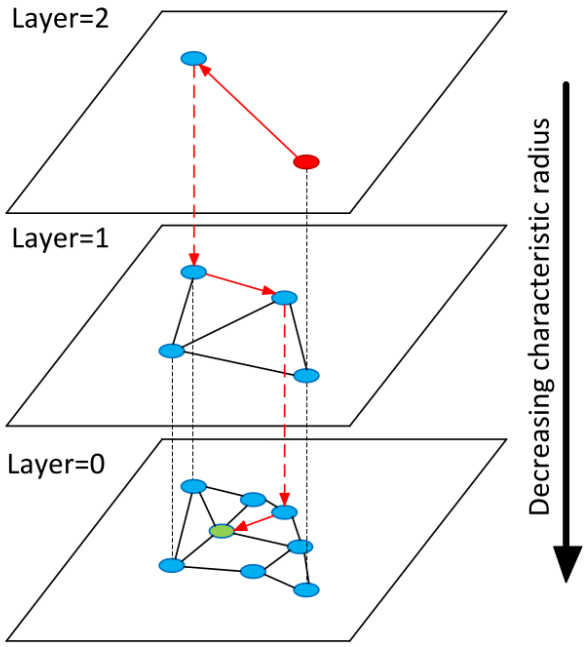
a Đồ thị phân tầng - nền tảng của HNSW

Trước khi đến với các khái niệm về đồ thị phân tầng, ta tìm hiểu một cấu trúc dữ liệu tương tự - probabilistic skip list (PSL). PSL ra đời với mục đích làm giảm độ phức tạp thời gian với những danh sách đã được sắp xếp sẵn. Ý tưởng của PSL là trong danh sách đã sắp xếp này, ta chia nó làm nhiều khu vực. Ta sẽ xác định phần tử cần tìm nằm trong khu vực nào, và ta chỉ việc tìm bên trong khu vực đó mà không cần quan tâm các khu vực đã bị loại trừ. Đó là lý do mà PSL được chia làm nhiều tầng (layer) từ cao xuống thấp với tầng càng cao thì danh sách càng thưa (biểu thị cho việc mỗi phần tử trong tầng này đại diện cho các khu vực tương ứng). Độ phức tạp thời gian cho việc tìm kiếm và thêm vào PSL là $O(\log n)$ [8].

Áp dụng ý tưởng đó, đồ thị HNSW được xây dựng dựa trên đồ thị NSW nhưng được chia ra làm nhiều tầng, với tầng 0 là tầng dày nhất và thưa dần lên trên. Ý tưởng là ta chia đồ thị ban đầu thành nhiều khu vực, ta xác định truy vấn cần tìm có khuynh hướng nằm trong khu vực nào, và ta chỉ tìm trong khu vực đó, loại bỏ các khu vực còn lại. Trong hình 2.3, việc tìm kiếm bắt đầu từ một đỉnh ở tầng trên cùng (được thể hiện màu đỏ), các mũi tên đỏ biểu thị quá trình tìm kiếm tham lam đến truy vấn cần tìm (đỉnh màu xanh lá). Khác với quá trình zoom-out ở



Hình 2.2: Một PSL với 8 phần tử, mục tiêu cần tìm là phần tử thứ 6 [9]



Hình 2.3: Minh họa cho ý tưởng HNSW [7]

đồ thị NSW, đồ thị HNSW thay thế bằng quá trình zoom-in, bao gồm việc tìm kiếm tham lam ở các tầng và việc chuyển từ tầng cao đến tầng thấp [7].

Việc phân tầng này đóng vai trò quan trọng: nó chia một đồ thị thành nhiều cụm nhỏ dựa trên các đặc trưng về khoảng cách giữa các đỉnh. Ở các tầng thấp, các đỉnh trong đồ thị có các kết nối với nhau (như đã nhắc đến ở section 1). Ở các tầng càng cao, đồ thị càng thưa, và các đỉnh sẽ có các kết nối với các đỉnh khác ở xa hơn, hàm ý rằng ở mỗi khu vực được chia ra luôn có các kết nối của một đỉnh đến một đỉnh ở khu vực khác, làm giảm đáng kể thời gian di chuyển giữa hai khu vực. Giống như trong thực tế, giả sử một người đang ở Hà Nội, người này biết rằng mình cần thiết đi đến Đà Nẵng. Vậy thì thay vì đi bằng ô tô (đi qua các điểm lân cận người này), một cách hiệu quả hơn là đi bằng máy bay (đi thẳng đến một đỉnh trong khu vực cần đến), từ đó làm giảm thời gian di chuyển.

b Các thuật toán trên đồ thị HNSW

Giống như mọi cấu trúc dữ liệu khác (tiêu biểu là PSL), đồ thị HNSW có các hàm khởi tạo, tìm kiếm, chèn, xóa.

Thuật toán chèn được thể hiện trong thuật toán 3. Thuật toán nhận các tham số đầu vào là cấu trúc đồ thị $hnsw$ hiện có, một truy vấn q, M là số kết nối cho một đỉnh ở mỗi tầng, trong khi M_{max} là giá trị tối đa của M. $efConstruction$ là số lân cận tại tầng 0 mà ta cần xét, nghĩa là ta sẽ chọn M đỉnh lân cận tốt nhất trong $efConstruction$ đỉnh và tạo kết nối từ q đến số đỉnh lân cận tốt nhất này. m_L là một tham số chuẩn hóa (normalization factor), dùng để điều chỉnh độ dốc của phân phối xác suất, và theo tác giả của [7], ta chọn $m_L = \frac{1}{\ln M}$. Với điểm được chèn vào, ta xác định tầng cao nhất l có thể có của đỉnh này, được tính bằng giá trị $l = \lfloor -\ln \text{random}(0,1) \times m_L \rfloor$ với random$(\cdot,\cdot)$ là hàm lấy giá trị ngẫu nhiên nằm giữa hai tham số đầu vào. Quá trình chèn được chia làm hai giai đoạn: từ tầng L là tầng cao nhất của đồ thị đến tầng $l+1$, và từ tầng l về tầng 0. Đầu tiên ta đi từ tầng L đến tầng $l+1$ và tìm các láng giềng gần với q, ở giai đoạn sau, ta kết nối q với các láng giềng đã tìm, đồng thời tìm thêm các láng giềng mới để kết nối, hiển nhiên số các kết nối của q ở mỗi tầng phải không lớn hơn M_{max} ở tầng tương ứng. Qua quá trình nghiên cứu, tác giả của [7] nhận xét ta chọn $M_{max} = 2M$ ở tầng 0 và $M_{max} = M$ ở các tầng còn lại. Độ phức tạp của thuật toán chèn là $O(\log N)$ với N là số vector có trong cơ sở dữ liệu [7]. Xét về vấn đề khởi tạo, giả sử ta muốn khởi tạo một đồ thị HNSW có N phần tử, ta cần chèn từng phần tử vào đồ thị. Độ phức tạp của một thao tác chèn cho một điểm dữ liệu là $O(\log N)$, do đó mà khi khởi tạo một đồ thị, ta cần chèn N điểm vào đồ thị, dẫn đến độ phức tạp cho việc khởi tạo là $O(N \log N)$ [7]. Mặt khác, việc lưu trữ một đồ thị có N đỉnh mà tại mỗi đỉnh có M kết nối làm cho độ phức tạp bộ nhớ là $O(N \times M)$.

Tại mỗi tầng, việc tìm kiếm ef láng giềng gần nhất để cân nhắc kết nối đến được thể hiện ở thuật toán 4, và trong ef đỉnh láng giềng này, thuật toán 5 giúp ta chọn ra M đỉnh để kết nối đến. Trong đó, nhóm tác giả của [7] cũng phát triển một thuật toán tìm kiếm láng giềng mạnh mẽ hơn là thuật toán 6. Thuật toán này không chỉ đơn giản là tìm M láng giềng gần nhất trong ef láng giềng, nó đảm bảo rằng các láng giềng được chọn nằm ở nhiều hướng khác nhau, làm cho đồ thị tăng phần đa dạng. Thực nghiệm cho thấy SELECT-NEIGHBORS-HEURISTIC luôn cho kết quả tốt hơn hoặc tương đương với SELECT-NEIGHBORS-SIMPLE [7].

Thuật toán tìm k láng giềng gần nhất với truy vấn q được thể hiện ở thuật toán 7. Khác với thuật toán chèn, K-NN-SEARCH tập trung vào việc tìm các láng giềng gần nhất ở các tầng mà không tạo thêm kết nối mới.

Algorithm 3 INSERT($hnsw, q, M, M_{max}, efConstruction, m_L$)

procedure INSERT($hnsw, q, M, M_{max}, efConstruction, m_L$)
  $W \leftarrow \emptyset$ ▷ List for the currently found nearest elements
  $ep \leftarrow$ get enter-point for $hnsw$
  $L \leftarrow$ level of ep ▷ Top layer for $hnsw$
  $l \leftarrow \lfloor -\ln(\text{unif}(0..1)) \cdot m_L \rfloor$ ▷ New element's level
  for $l_c \leftarrow L$ **down to** $l + 1$ **do**
    $W \leftarrow$ SEARCH-LAYER$(q, ep, ef = 1, l_c)$
    $ep \leftarrow$ get the nearest element from W to q
  end for
  for $l_c \leftarrow \min(L, l)$ **down to** 0 **do**
    $W \leftarrow$ SEARCH-LAYER$(q, ep, efConstruction, l_c)$
    $neighbors \leftarrow$ SELECT-NEIGHBORS(q, W, M, l_c) ▷ alg. 3 or alg. 4
    add bidirectional connections from $neighbors$ to q at layer l_c
    for all $e \in neighbors$ **do** ▷ Shrink connections if needed
      $eConn \leftarrow$ neighbourhood(e) at layer l_c
      if $|eConn| > M_{max}$ **then** ▷ if $l_c = 0$ then $M_{max} = M_{max0}$
        $eNewConn \leftarrow$ SELECT-NEIGHBORS$(e, eConn, M_{max}, l_c)$
        set neighbourhood(e) at layer l_c to $eNewConn$
      end if
    end for
    $ep \leftarrow W$
  end for
  if $l > L$ **then**
    set enter-point for $hnsw$ to q
  end if
end procedure

Algorithm 4 SEARCH-LAYER(q, ep, ef, l_c)

procedure SEARCH-LAYER(q, ep, ef, l_c)
  $v \leftarrow ep$ ▷ Set of visited elements
  $C \leftarrow ep$ ▷ Set of candidates
  $W \leftarrow ep$ ▷ Dynamic list of found nearest neighbors
  while $|C| > 0$ **do**
    $c \leftarrow$ extract nearest element from C to q
    $f \leftarrow$ get furthest element from W to q
    if distance$(c, q) >$ distance(f, q) **then**
      break ▷ All elements in W are evaluated
    end if
    for all $e \in$ neighbourhood(c) at layer l_c **do** ▷ Update C and W
      if $e \notin v$ **then**
        $v \leftarrow v \cup \{e\}$
        $f \leftarrow$ get furthest element from W to q
        if distance$(e, q) <$ distance(f, q) or $|W| < ef$ **then**
          $C \leftarrow C \cup \{e\}$
          $W \leftarrow W \cup \{e\}$
          if $|W| > ef$ **then**
            remove furthest element from W to q
          end if
        end if
      end if
    end for
  end while
  return W
end procedure

Algorithm 5 SELECT-NEIGHBORS-SIMPLE(q, C, M)

procedure SELECT-NEIGHBORS-SIMPLE(q, C, M)
  return M nearest elements from C to q
end procedure

Algorithm 6 SELECT-NEIGHBORS-HEURISTIC($q, C, M, l_c, ...$)

procedure SELECT-NEIGHBORS-HEURISTIC($q, C, M, l_c, extendCandidates, keepPrunedConnec$
  $R \leftarrow \emptyset$
  $W \leftarrow C$ ▷ Working queue for the candidates
  if $extendCandidates$ **then** ▷ Extend candidates by their neighbors
    for all $e \in C$ **do**
      for all $e_{adj} \in$ neighbourhood(e) at layer l_c **do**
        if $e_{adj} \notin W$ **then**
          $W \leftarrow W \cup \{e_{adj}\}$
        end if
      end for
    end for
  $W_d \leftarrow \emptyset$ ▷ Queue for the discarded candidates
  while $|W| > 0$ and $|R| < M$ **do**
    $e \leftarrow$ extract nearest element from W to q
    if e is closer to q than any element in R **then**
      $R \leftarrow R \cup \{e\}$
    else
      $W_d \leftarrow W_d \cup \{e\}$
    end if
  end while
  if $keepPrunedConnections$ **then**
    while $|W_d| > 0$ and $|R| < M$ **do**
      $R \leftarrow R \cup \{$extract nearest element from W_d to $q\}$
    end while
  end if
  return R
end procedure

Algorithm 7 K-NN-SEARCH($hnsw, q, K, ef$)

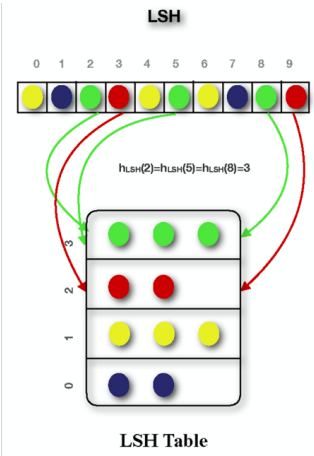
procedure K-NN-SEARCH($hnsw, q, K, ef$)
  $W \leftarrow \emptyset$ ▷ Set for the current nearest elements
  $ep \leftarrow$ get enter-point for $hnsw$
  $L \leftarrow$ level of ep ▷ Top layer for $hnsw$
  for $l_c \leftarrow L$ **down to** 1 **do**
    $W \leftarrow$ SEARCH-LAYER$(q, ep, ef = 1, l_c)$
    $ep \leftarrow$ get nearest element from W to q
  end for
  $W \leftarrow$ SEARCH-LAYER$(q, ep, ef, l_c = 0)$
  return K nearest elements from W to q
end procedure

III CÁC CHIẾN LƯỢC ANN KHÁC

Một thuật toán đơn giản nhất là brute-force khi ta đi tính $dis(q, u_i) \forall i \in [1, n]$ và chọn ra k vector có khoảng cách bé nhất. Mặc dù phương pháp này cho kết quả hoàn toàn chính xác, nó có độ phức tạp là $O(N \times d)$ với d là số chiều của các vector, điều này tốn tài nguyên và thời gian, và không phù hợp cho các hệ thống lớn gồm hàng triệu vector [10]. Do đó mà người ta mới bắt đầu mô hình hóa các điểm dữ liệu sử dụng các cấu trúc dữ liệu khác như cây, đồ thị, đồ thị phân tầng (được nói đến ở phần II), làm quá trình truy vấn diễn ra nhanh hơn mặc dù cần đánh đổi thêm tài nguyên lưu trữ [11].

Locality sensitive hashing (LSH) là một chiến lược ANN dựa trên cấu trúc bảng băm (hash table). Ý tưởng của phương pháp này là ta chia tập dữ liệu thành nhiều phần (bucket) mà ở mỗi phần, các điểm dữ liệu bên trong là tương tự nhau. Nó cũng giống như việc ta phân chia sách trong thư viện. Ta chỉ cần đi đến khu vực sách có khả năng chứa cuốn sách ta cần tìm, và tìm trong khu vực đó, thay vì phải đi đối chiếu với toàn bộ sách trong thư viện. Tương tự, khi chương trình nhận một truy vấn, chương trình sẽ xác định bucket mà chứa các điểm dữ liệu có khả năng đáp ứng truy vấn. Sâu sắc hơn, từng cặp dữ liệu sẽ có xác suất được hash vào một bucket nêu khoảng cách giữa chúng không lớn hơn một giá trị cho trước và ngược lại [12]. Yếu tố "xác suất"ở đây đảm bảo tính gần đúng của chiến lược ANN. Nó cho thấy rằng thuật toán đánh đổi một phần độ chính xác nhưng tối ưu hơn trong thời gian tìm kiếm mà vẫn cho ra kết quả thỏa mãn với yêu cầu [12]. Hình 2.4 minh họa cho nguyên lý hoạt động của các buckets. Các điểm dữ liệu gần giống nhau (cùng màu) sẽ được gom vào một bucket.

Về mặt lý thuyết, LSH tối ưu hơn tìm kiếm bằng đồ thị HNSW, đặc biệt ở các tập dữ liệu thưa và nhiều chiều. Tuy nhiên, thực nghiệm cho thấy tìm kiếm bằng đồ thị HNSW cho ra hiệu suất tìm kiếm cao hơn [11]. Một phiên bảng nâng cấp của LSH là Falconn đã cho thấy tính hiệu quả tương đương khi so sánh với tìm kiếm bằng đồ thị HNSW [13].

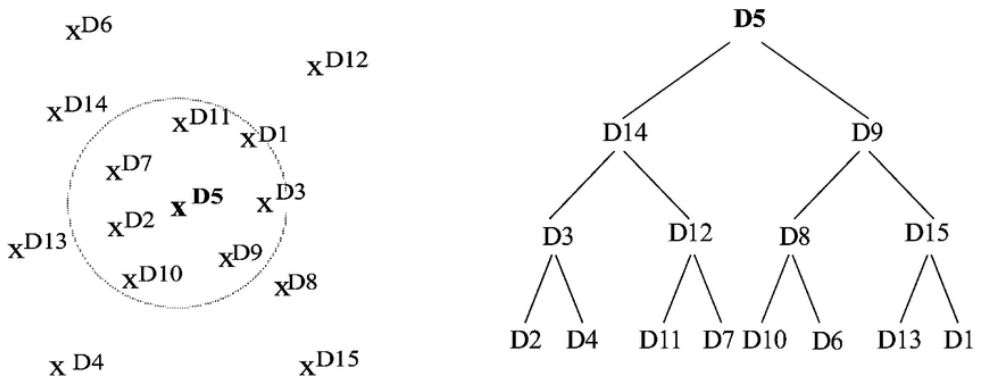


Hình 2.4: Minh họa cho các buckets trong thuật toán LSH [14]

Một chiến lược ANN khác dựa trên cấu trúc dữ liệu cây là vantage point tree (vp-tree).

Trong cấu trúc vp-tree, mỗi đỉnh của đồ thị là một điểm dữ liệu nhiều chiều. Để khởi tạo, bắt đầu từ một tập điểm, ta chọn ngẫu nhiên một điểm gốc, sau đó tính khoảng cách đến các đỉnh còn lại và phân các điểm còn lại thành hai nhóm: bé hơn trung vị các khoảng cách và ngược lại. Giá trị trung vị này còn được gọi là bán kính của vantage point. Sau đó, ta áp dụng quy trình trên một cách đệ quy cho hai tập con vừa phân chia, và một cấu trúc vp-tree sẽ được tạo thành. Việc chèn một điểm mới sẽ tương tự như việc chèn một điểm vào một cây nhị phân với độ phức tạp là $O(\log N)$, và việc khởi tạo một cây vp-tree có độ phức tạp là $O(N \log N)$ với N là số điểm dữ liệu hiện có. Nhược điểm của cấu trúc này là nó sẽ dễ bị suy biến thành một danh sách liên kết, hoặc một cây nhị phân không cân bằng (dẫn đến từ việc chèn điểm).

Trong hình 2.5, ứng với mỗi một vantage point cho trước, ta phân không gian làm hai vùng: tập các điểm xa vantage point một khoảng r và ngược lại. Khi đó, một vp-tree sẽ được khởi tạo với nút cha là vantage point đã chọn, hai tập con là hai tập ứng với hai vùng đã chia dựa trên đường tròn bán kính r.



Hình 2.5: Minh họa khởi tạo một vp-tree [15]

So sánh với HNSW, lối lưu trữ bằng vp-tree cho thấy nhu cầu bộ nhớ cần cấp phát thêm do đặc trưng của kiểu dữ liệu (memory overhead) ít hơn (do đồ thị HNSW cần lưu trữ nhiều dữ liệu về kết nối hơn vp-tree) [11]. Do có đánh đổi đó nên đồ thị HNSW cho thấy hiệu năng tìm kiếm cao hơn (độ phức tạp thời gian logarit so với hàm lũy thừa của vp-tree) [11].

IV ỨNG DỤNG CỦA ĐỒ THỊ HNSW TRONG CÁC HỆ THỐNG TRUY VẤN VECTOR

Faiss (Facebook AI Similarity Search) là một thư viện mã nguồn mở giải quyết các bài toán ANNs, được phát triển bởi đội ngũ Meta AI [16]. Faiss mạnh mẽ trong các ứng dụng về tìm kiếm tương tự [16]. Faiss sử dụng cấu trúc đồ thị HNSW để xây dựng vector database cho các dữ liệu có số chiều cao, ví dụ như text (768 chiều), và hình ảnh [16]. Hơn hết, việc sử dụng cấu trúc đồ thị HNSW trong Faiss là nền tảng để cộng đồng phát triển các ứng dụng liên quan khi sử dụng Faiss, ví dụ như Milvus, có thể kể đến các ứng dụng về xử lí ảnh (image processing), thị giác máy tính (computer vision), xử lí ngôn ngữ tự nhiên (natural language processing - NLP), nhận diện giọng nói, hệ thống gợi ý (reccommender systems) và nhiều ứng dụng khác

[2].

TÀI LIỆU THAM KHẢO

[1] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin, "A neural probabilistic language model," *Journal of machine learning research*, vol. 3, no. Feb, pp. 1137–1155, 2003.

[2] J. Wang, X. Yi, R. Guo, H. Jin, P. Xu, S. Li, X. Wang, X. Guo, C. Li, X. Xu *et al.*, "Milvus: A purpose-built vector data management system," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 2614–2627.

[3] S. Sharma, R. Nayak, and A. Bhaskar, "Multi-view feature engineering for day-to-day joint clustering of multiple traffic datasets," *Transportation Research Part C: Emerging Technologies*, vol. 162, p. 104607, 2024.

[4] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM computing surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.

[5] Y. Wang, Z. He, Y. Tong, Z. Zhou, and Y. Zhong, "Timestamp approximate nearest neighbor search over high-dimensional vector data," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2025, pp. 3043–3055.

[6] 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.

[7] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 4, pp. 824–836, 2018.

[8] W. Pugh, "Skip lists: a probabilistic alternative to balanced trees," *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[9] T. Papadakis, *Skip lists and probabilistic analysis of algorithms*. University of Waterloo Ph. D. Dissertation, 1993.

[10] A. J. Gallego, J. Calvo-Zaragoza, and J. R. Rico-Juan, "Insights into efficient k-nearest neighbor classification with convolutional neural codes," *IEEE Access*, vol. PP, pp. 1–1, 05 2020.

[11] M. Wang, X. Xu, Q. Yue, and Y. Wang, "A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search," *arXiv preprint arXiv:2101.12631*, 2021.

[12] A. Gionis, P. Indyk, R. Motwani *et al.*, "Similarity search in high dimensions via hashing," in *Vldb*, vol. 99, no. 6, 1999, pp. 518–529.

[13] N. Pham and T. Liu, "Falconn++: A locality-sensitive filtering approach for approximate nearest neighbor search," *Advances in Neural Information Processing Systems*, vol. 35, pp. 31 186–31 198, 2022.

[14] A. Chakraborty and S. Bandyopadhyay, "conlsh: context based locality sensitive hashing for mapping of noisy smrt reads," *Computational Biology and Chemistry*, vol. 85, p. 107206, 2020.

[15] C. Böhm, S. Berchtold, and D. Keim, "Searching in high-dimensional spaces : Index structures for improving the performance of multimedia databases," *First publ. in: ACM computing surveys 33 (2001), 3, pp. 322-373*, vol. 33, 09 2001.

[16] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," *arXiv preprint arXiv:2401.08281*, 2024.