Proximity Graphs for Approximate Nearest Neighbor Search: from Theory to Practice

Shangqi Lu

Data Science and Analytics Thrust
Hong Kong University of Science and Technology (Guangzhou)

Problem Definition

Consider a **metric space** $(\mathcal{M}, D)$ where

- $\mathcal{M}$ is a set where each element is called a **point**;
- $D : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a **distance function**.

Problem Definition

Consider a **metric space** $(\mathcal{M}, D)$ where

- $\mathcal{M}$ is a set where each element is called a **point**;
- $D : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a **distance function**.

The function D satisfies (i) identity of indiscernibles: $D(p_1, p_2) = 0$ if and only if $p_1 = p_2$, (ii) symmetry: $D(p_1, p_2) = D(p_2, p_1)$, and (iii) triangle inequality: $D(p_1, p_2) \leq D(p_1, p_3) + D(p_2, p_3)$.

Problem Definition

Consider a **metric space** $(\mathcal{M}, D)$ where

- $\mathcal{M}$ is a set where each element is called a **point**;
- $D : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ is a **distance function**.

ANN search. Let P be a set of n **data points** from $\mathcal{M}$.

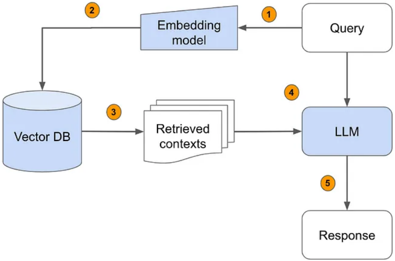
- Given a point $q \in \mathcal{M}$, a point $p^* \in P$ is a **nearest neighbor** (NN) of q if $D(p^*, q) \leq D(p, q)$ holds for all $p \in P$.
- For a constant $\epsilon \in (0, 1]$, a point $p \in P$ is called a **$(1 + \epsilon)$-approximate nearest neighbor (ANN)** of q if $D(p, q) \leq (1 + \epsilon) \cdot D(p^*, q)$.

Goal: Build a data structure on P to answer ANN queries efficiently.

Vector search: Build a data structure on a set D of d-dimensional points in $\mathbb{R}^d$ to support:

- Given a query point $q \in \mathbb{R}^d$ and an integer $k > 0$, return (approximate) k-closest points of D to q under some distance function, e.g., Euclidean distance.

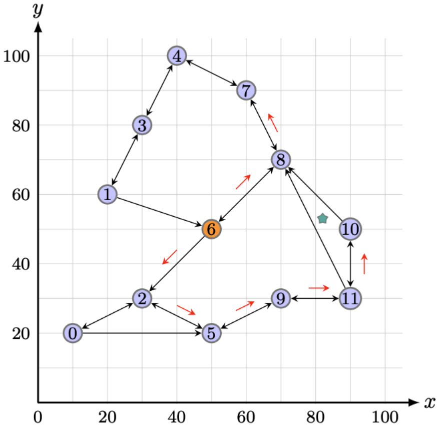
An emerging application is retrieval augmented generation (RAG) for LLMs, where Vector DBs serve as external datastores to provide additional context.



https://medium.com/@bijit211987/rag-vs-vectordb-2c8cb3e0ee52

Proximity Graphs

A **proximity graph** (PG) G of P is a simple directed graph, where each point of P corresponds to a vertex in G, and vice versa.



Proximity Graphs

Given a query point $q \in \mathcal{M}$, we start from an arbitrary point $p_{\text{start}} \in P$, and use a greedy algorithm to do the search:

greedy(p_{start}, q)

1. $p \leftarrow p_{\text{start}}$ /* the first hop */
2. **repeat**
3. $\quad p^+_{\text{out}} \leftarrow$ the out-neighbor of p closest to q
4. $\quad$ **if** $p^+_{\text{out}} =$ nil **or** $D(p, q) \leq D(p^+_{\text{out}}, q)$ **then return** p
5. $\quad p \leftarrow p^+_{\text{out}}$ /* the next hop */

Proximity Graphs

Given a query point $q \in \mathcal{M}$, we start from an arbitrary point $p_{\text{start}} \in P$, and use a greedy algorithm to do the search:

greedy(p_{start}, q)
1. $p \leftarrow p_{\text{start}}$ /* the first hop */
2. **repeat**
3. $p^+_{\text{out}} \leftarrow$ the out-neighbor of p closest to q
4. **if** $p^+_{\text{out}} =$ nil **or** $D(p, q) \leq D(p^+_{\text{out}}, q)$ **then return** p
5. $p \leftarrow p^+_{\text{out}}$ /* the next hop */

Definition. We call G a **$(1+\epsilon)$-proximity graph** (PG) if, given any query point $q \in \mathcal{M}$ and any data point $p_{\text{start}} \in P$, **greedy**(p_{start}, q) always returns a $(1+\epsilon)$-ANN of q.

The **complete graph** on P is already a $(1+\epsilon)$-PG for any $\epsilon > 0$. However, this graph has $\Theta(n^2)$ edges and $\Omega(n)$ query time.

The **complete graph** on P is already a $(1+\epsilon)$-PG for any $\epsilon > 0$. However, this graph has $\Theta(n^2)$ edges and $\Omega(n)$ query time.

We ask the following questions:

- **Q1:** How to build a sparse PG with fast query time?
- **Q2:** What are the limitations of PGs?

Ideally, the graph should have a low maximum out-degree and the greedy search can terminate in a small number of steps.

In this talk, we will discuss

1. The theoretical foundations of PG;
    - The best upper bound on space and query time;
    - A lower bound on the number of edges.
2. Some heuristics when implementing a PG;
3. Our recent results.

A Local Property

A proximity graph G is **$(1+\epsilon)$-navigable** if the following condition holds for every data point $p \in P$ and every query point $q \in \mathcal{M}$:

- either p is a $(1+\epsilon)$-ANN of q,
- or p has an out-neighbor p_{out} satisfying $D(p_{\text{out}}, q) < D(p, q)$.

Lemma. G is a $(1+\epsilon)$-PG of P if and only if G is $(1+\epsilon)$-navigable.

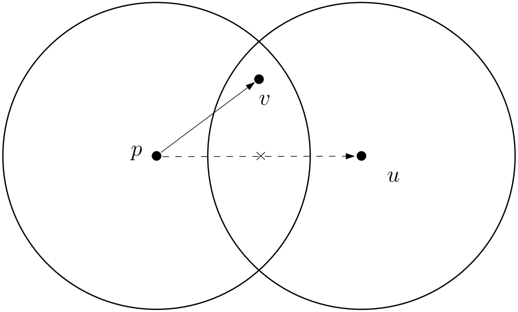
We can thus focus on finding a $(1+\epsilon)$-navigable graph.

A Simple Case: When $q \in P$

Sparse relative neighbor hood graph: For each point $p \in P$, construct the out-neighbors of P as follows [Arya and Mount, SODA'93]:

A pruning procedure:

1. Set $C = P \setminus \{p\}$ and sort C in ascending order of distance to p;
2. Let v be the point in C closest to p;
3. Add a directed edge from p to v;
4. Remove all the points $u \in C$ satisfying $D(u, v) < D(p, u)$
5. Repeat steps 2-4 until C is empty.



Given any $p \in \mathcal{M}$ and $r > 0$, define the **ball** $B(p, r)$ with radius r as the set $\{p' \in \mathcal{M} \mid D(p, p') \leq r\}$.

> u is "pruned" if there exists an out-neighbor v of p falling within the intersection of $B(p, D(p, u))$ and $B(u, D(p, u))$.

A Simple Case: When $q \in P$

By definition, the sparse RNG is $(1+0)$-navigable when $q \in P$: for any $p \neq q$ (p is not the exact NN)

- either p is connected to q;
- or p has an out-neighbor v that is closer to q than p

A greedy search on the sparse RNG can always find q itself.

A Simple Case: When $q \in P$

By definition, the sparse RNG is $(1+0)$-navigable when $q \in P$: for any $p \neq q$ (p is not the exact NN)

- either p is connected to q;
- or p has an out-neighbor v that is closer to q than p

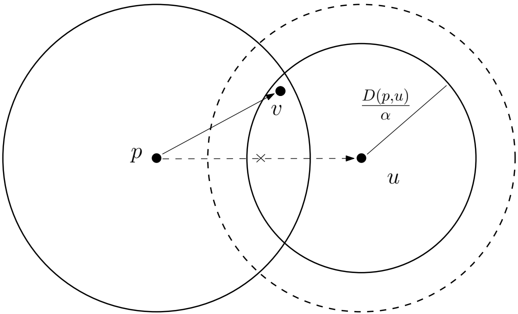
A greedy search on the sparse RNG can always find q itself.

Lemma. [Arya and Mount, SODA'93]

- In the Euclidean space, the sparse RNG has a maximum out-degree $O(1.32^{d-1})$.

However, the number of search steps is unbounded.

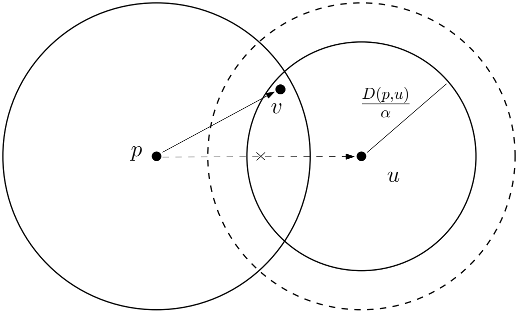
Another Pruning Rule



Let $\alpha > 1$ be a parameter.
The **Vamana** graph in the DiskANN method slightly modifies the pruning rule [Subramanya et al., NIPS'19]:

> u is "pruned" if there exists an out-neighbor v of p falling within the intersection of $B(p, D(p, u))$ and $B(u, \frac{D(p,u)}{\alpha})$.

The General Case for Arbitrary q



In the Vamana graph:

The α-shortcut reachable property:
Consider any points $p, u \in P$. If u is not an out-neighbor of p, p must have an out-neighbor v such that $D(v, u) \leq D(p, u)/\alpha$.

The General Case for Arbitrary q

Δ: the aspect ratio of P, i.e., the ratio between the maximum and minimum pairwise distances of P.

d: the **doubling dimension** of the metric space $(\mathcal{M}, D)$.

The General Case for Arbitrary q

Δ: the aspect ratio of P, i.e., the ratio between the maximum and minimum pairwise distances of P.
d: the **doubling dimension** of the metric space $(\mathcal{M}, D)$.

Theorem. [Indyk and Xu, NIPS'23] Let G be the Vamana graph on P with $\alpha = 1 + \frac{4}{\epsilon}$. The following are true:

- G is $(1+\epsilon)$-navigable;
- Each node has a out-degree $O(\log \Delta)$;
- A greedy search finds an $(1+\epsilon)$-ANN of q in $O(\log \Delta)$ steps.

The space of G is $O(n \cdot \log \Delta)$.
The query time is $O(\log^2 \Delta)$.

A Lower Bound

Question: can we do better in the space and query time?

A Lower Bound

Question: can we do better in the space and query time?

> **Theorem.** [Lu and Tao, PODS'26] For any constant $\epsilon > 0$, there is a set P whose doubling dimension 1 such that any $(1+\epsilon)$-PG for P must have $\Omega(n \log \Delta)$ edges, regardless of the query time allowed.

The space of the Vamana graph is optimal for constant ϵ.

Implementation in Systems

The **beam search** algorithm is used to find k ANNs of q.

Algorithm 1 beam-search(G, q, s, L, k)

Input: graph G, query point q, entry point s, queue size L
Output: k ANN of q

```
candidate queue Q ← {s}
explored set ℰ = ∅
while Q \ ℰ ≠ ∅ do
    u* ← arg min{δ(x, q) | x ∈ Q \ ℰ}
    for each out-neighbor v of u* do
        Q ← Q ∪ {v}
    end for
    keep the L entries in Q that are closest to q
    ℰ ← ℰ ∪ {u*}
end while
return k points in Q closest to q
```

Implementation in Systems

The construction time of the sparse RNG and Vamana are both $\Omega(n^2)$.

To reduce the construction time, the industry often adopts a heuristic framework. For each data point $p \in P$,

1. find a small candidate set $\mathcal{V}$ with size at most C (e.g., $C = 500$), rather than set $\mathcal{V} = P \setminus \{p\}$;
2. run the pruning procedure on $\mathcal{V}$; let S be the set of points returned;
3. pick the closest M points in S as the out-neighbors of p;
4. (more details)
   - add reverse edges: for each added out-neighbor u of p, add an edge from u to p;
   - prune each node's out-neighbor set if its size is larger than M.

Implementation in Systems

Question: How to find a small candidate set?

Incremental strategy (e.g., DiskANN and HNSW):

- insert each data point into the graph individually;
- for each new point, generate a candidate set by performing beam search on **the partially constructed graph.**

Implementation in Systems

Question: How to find a small candidate set?

Incremental strategy (e.g., DiskANN and HNSW):

- insert each data point into the graph individually;
- for each new point, generate a candidate set by performing beam search on **the partially constructed graph.**

Base graph strategy (e.g., NSG):

- construct an approximate K-NN graph first (each node connects to its K closest ANNs);
- for each point, generate a candidate set by performing beam search on the **approximate k-NN graph**.

However, with these heuristics, the theoretical guarantees on query performance no longer hold.

Our Recent Progress

Fast construction:

Theorem. [Lu and Tao, PODS'26] For $d = O(1)$, we can construct a $(1+\epsilon)$-PG in $O(n \operatorname{polylog}(n\Delta))$ time that has

- $O(n \log \Delta)$ edges (optimal) and
- $O(\log^2 \Delta)$ query time.

Our Recent Progress

Fast construction:

Theorem. [Lu and Tao, PODS'26] For $d = O(1)$, we can construct a $(1+\epsilon)$-PG in $O(n \operatorname{polylog}(n\Delta))$ time that has

- $O(n \log \Delta)$ edges (optimal) and
- $O(\log^2 \Delta)$ query time.

We can have a better space in **Euclidean space**:

Theorem. [Lu and Tao, PODS'26] For Euclidean space and $d = O(1)$, there is a $(1+\epsilon)$-PG that has

- $O(n)$ edges and
- $O(\log n \cdot \log^2 \Delta)$ query time.

We can construct such a graph in $O(n \operatorname{polylog}(n\Delta))$ time.

A Log-drop Property of Our PG

q: an arbitrary query in $\mathcal{M}$
p: an arbitrary point in P
p^*: an exact NN of q.
p_{out}^{+} = the out-neighbor of p closest to q.

A Log-drop Property of Our PG

q: an arbitrary query in $\mathcal{M}$
p: an arbitrary point in P
p^*: an exact NN of q.
p^+_{out} = the out-neighbor of p closest to q.

If p is not a $(1+\epsilon)$-ANN of q, we have

1. $D(p^+_{\text{out}}, q) < D(p^\circ, q)$;
2. let u be any point in P satisfying $D(u, q) \leq D(p^+_{\text{out}}, q)$. If ρ is not a $(1+\epsilon)$-ANN of q, then

$$\lceil \log D(u, p^*) \rceil < \lceil \log D(p, p^*) \rceil .$$

Discussion

There are actually non-PG-based structures with better theoretical guarantees [Har-Peled et al., SIAM J'06, Cole et al., STOC'06]:

Theorem. When $d = O(1)$, there is a structure of $O(n)$ space that answers a $(1+\epsilon)$-ANN query in $O(\log n) + (1/\epsilon)^{O(d)}$ time. The structure can be constructed in $O(n \log n)$ time.

Discussion

There are actually non-PG-based structures with better theoretical guarantees [Har-Peled et al., SIAM J'06, Cole et al., STOC'06]:

Theorem. When $d = O(1)$, there is a structure of $O(n)$ space that answers a $(1+\epsilon)$-ANN query in $O(\log n) + (1/\epsilon)^{O(d)}$ time. The structure can be constructed in $O(n \log n)$ time.

Question: why do we still use PG?

Possible Research Directions

- Better query time/lower bounds on query time;
- I/O-efficient solutions;
- Supporting updates;
- Similarity search with attribute filtering;
- Practical solutions that can be construct fast and has theoretical guarantees.

Thank You!