Indexes for Efficient Search

Karl Stratos

This note assumes that you understand quantization (Appendix A) and graph-based search (Appendix C).

1 Setting

Let $x \in \mathbb{R}^D$ denote a query vector and $\mathcal{Y} = \{y_1 \dots y_N\} \subset \mathbb{R}^D$ a set of N target vectors. Our goal is to find

$$y^* = \arg\min_{y \in \mathcal{Y}} ||x - y|| \tag{1}$$

An **index** (plural: indexes) is a data structure to store $\mathcal{Y}$, paired with an inference scheme for finding (1). For each index, we are interested in (i) exactness, (ii) memory usage, and (iii) runtime.

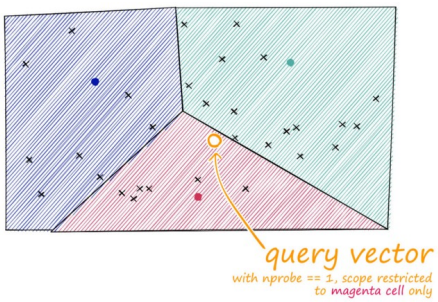
The baseline is the **flat index**, which stores $\mathcal{Y}$ in memory as is and conducts an exhaustive search. It is trivially exact. It uses $4ND$ bytes since each float requires 4 bytes. It has $O(ND)$ runtime since it computes N distances between two D-dimensional vectors.

2 Inverted File (IVF) Index

We can speed up search by constraining the search space.[1] The **IVF index** does this by learning a codebook $C = (\mu_1 \dots \mu_K) \in \mathbb{R}^{K \times D}$ from $\mathcal{Y}$ and maintaining a partition $\mathcal{Y}_k = \{y \in \mathcal{Y} : \textbf{code}(y) = k\}$. Given $P \leq K$, IVF computes

$$y_{\text{IVF}} = \arg\min_{y \in \mathcal{Y}_{k_1} \cup \dots \cup \mathcal{Y}_{k_P}} ||x - y||$$

where $||x - \mu_{k_1}|| < \dots < ||x - \mu_{k_P}|| < \dots < ||x - \mu_{k_K}||$. Clearly IVF is exact with $P = K$. With $P < K$ we may lose exactness. Let $P = 1$ and consider the case x lies close to the boundary (image credit: James Briggs):



In this picture, y^* is outside $\mathcal{Y}_1$ (the magenta cell), thus $y_{\text{IVF}} \neq y^*$. There is no improvement in memory usage since we still need to store all target vectors (plus additional memory to store the codebook and the inverted file structure). However, runtime has improved to $O(N_{\text{IVF}}D)$ where N_{IVF} is the expected size of the constrained search space. For instance, assuming uniform cluster sizes, IVF is $\frac{K}{P}$ times faster.

[1] In general, we achieve this by constructing K smaller subsets $\mathcal{Y}_1 \dots \mathcal{Y}_K \subset \mathbb{R}^D$ of $\mathcal{Y}$, inferring what subsets to search for x, and only going through those subsets. A classical example is the inverted index with a sparse representation (e.g., TFIDF) where each dimension corresponds to a word in some vocabulary $\mathcal{V}$. Since a vector has nonzero weights only for activated words, we can construct subsets $\mathcal{Y}_1 \dots \mathcal{Y}_{|\mathcal{V}|}$ where $\mathcal{Y}_w = \{y \in \mathcal{Y} : y_w > 0\}$ (they may overlap). Given x, we only search $\mathcal{Y}_w$'s such that $x_w > 0$. Note that the search remains exact in this case.

3 Product Quantization (PQ) Index

The **PQ index** performs product quantization on $\mathcal{Y} \subset \mathbb{R}^D$ (Appendix A.2). It expects a number of subspaces M and a number of bits to encode the code assignment b, which imply M sub-codebooks of the form $C_j = (\mu_1^{(j)} \dots \mu_K^{(j)}) \in \mathbb{R}^{K \times \frac{D}{M}}$ where $K = 2^b$. The associated quantizer q can output K^M possible values. PQ exploits the fact that the (squared) distance between the query x and the quantized target $q(y_i)$ can be written as

$$||x - q(y_i)||^2 = \sum_{j=1}^{M} \left|\left| \mathbf{sub}(x, j) - \mu^{(j)}_{\mathbf{code}_j(y_i)} \right|\right|^2$$

where many values can be precomputed. Note that the query vector is not quantized: this design choice is termed **assymetric distance computation (ADC)**.[2] We will assume $b \ll \log N$ and ignore overheads in K. The storage requirement of PQ is

- $C = [C_1 \dots C_M] \in \mathbb{R}^{K \times D}$ which takes $4KD$ bytes (constant in N).
- $O \in \{1 \dots K\}^{N \times M}$ where $O_{i,j} = \mathbf{code}_j(y_i)$. This takes $\frac{1}{8}NMb$ bytes (e.g., NM if $b = 8$).

which uses roughly $\frac{D}{M}$ times less memory. At inference time, given the query x, PQ computes

- $\Delta(x) \in \mathbb{R}^{M \times K}$ where $\Delta_{j,k}(x) = \|\mathbf{sub}(x, j) - \mu_k^{(j)}\|^2$. This takes $O(KD)$ time (constant in N).
- $y_{\text{PQ}} = y_{i^*}$ where

$$i^* = \arg\min_{i=1}^{N} \sum_{j=1}^{M} \Delta_{j,O_{i,j}}(x)$$

  This take $O(NM)$ time.

Thus PQ is roughly $\frac{D}{M}$ times faster. PQ is exact if quantization is lossless (i.e., $q(y_i) = y_i$, for instance with $M = D$, $K = N$, and $\mu_k^{(j)} = [y_k]_j$), but in general $y_{\text{PQ}} \neq y^*$.

4 IVFPQ Index

IVF speeds things up by non-exhaustive search. PQ reduces memory usage by quantization. The **IVFPQ** index pipelines IVF and PQ to achieve both benefits, though it also suffers from error propagation. A naive approach is to build the two indexes independently. A better approach is to use IVF as a coarse quantizer, then performing PQ on the resulting residuals, which can be effective in combating variance issues (Appendix A.3). Specifically, we compute

$$q_c \leftarrow \mathbf{Quantizer}\left(\mathcal{Y}, K, P\right)$$
$$q_p \leftarrow \mathbf{ProductQuantizer}\left(\bar{\mathcal{Y}}, M, b\right) \qquad \bar{\mathcal{Y}} = \left\{y - q_c(y) \in \mathbb{R}^D : \ y \in \mathcal{Y}\right\}$$

The final quantization of y_i is given by $q(y_i) = q_c(y_i) + q_p(y_i - q_c(y_i))$. Let $L = 2^b$ denote the number of centroids in each sub-codebook of q_p (to distinguish it from the number of centroids K in the codebook of q_c). The storage requirement of IVFPQ is

- $(\mu_1 \dots \mu_K) \in \mathbb{R}^{K \times D}$, the codebook of q_c
- $o \in \{1 \dots K\}^N$ where $o_i = \mathbf{code}_c(y_i)$ is the code assigned to the i-th target by q_c
- Inverted file structure $\mathbf{inv}(k) = \{i : \ o_i = k\}$
- $[C_{p,1} \dots C_{p,M}] \in \mathbb{R}^{L \times D}$, the sub-codebooks of q_p where $C_{p,j} = (\nu_1^{(j)} \dots \nu_L^{(j)}) \in \mathbb{R}^{L \times \frac{D}{M}}$
- $O \in \{1 \dots L\}^{N \times M}$ where $O_{i,j} = \mathbf{code}_{p,j}(y_i - q_c(y_i))$ is the j-th code assigned to the i-th residual by q_p

[2] In **symetric distance computation (SDC)**, we consider $||q(x) - q(y_i)||^2$. The upshot is that all possible $O(MK^2)$ differences between centroids can be precomputed. However, this incurs additional loss in accuracy since we quantize the query (worse, the centroids are fit to the target vectors). Thus most works focus on ADC.

which is roughly the same as in PQ (so $\frac{D}{M}$ times less memory). IVFPQ exploits the fact that the (squared) distance between the query x and $q(y_i)$ can be written as

$$\begin{aligned}||x - q(y_i)||^2 &= ||x - q_c(y_i) - q_p(y_i - q_c(y_i))||^2 \\ &= ||x - q_c(y_i)||^2 + ||q_p(y_i - q_c(y_i))||^2 + 2\langle q_c(y_i), q_p(y_i - q_c(y_i))\rangle - 2\langle x, q_p(y_i - q_c(y_i))\rangle\end{aligned}$$

where we can further expand the latter terms as

$$||q_p(y_i - q_c(y_i))||^2 = \sum_{j=1}^{M} \left|\left|\nu_{O_{i,j}}^{(j)}\right|\right|^2 \tag{2}$$

$$\langle q_c(y_i), q_p(y_i - q_c(y_i))\rangle = \sum_{j=1}^{M} \left\langle \mathbf{sub}(\mu_{o_i}, j), \nu_{O_{i,j}}^{(j)} \right\rangle \tag{3}$$

$$\langle x, q_p(y_i - q_c(y_i))\rangle = \sum_{j=1}^{M} \left\langle \mathbf{sub}(x, j), \nu_{O_{i,j}}^{(j)} \right\rangle$$

In principle the terms required to compute (2) and (3) can be precomputed offline as $\pi(j,l) = \|\nu_l^{(j)}\|^2$ and $\tau(k,j,l) = \langle \mathbf{sub}(\mu_k, j), \nu_l^{(j)}\rangle$, though it uses $O(KML)$ memory. In summary, at inference time, given the query x IVFPQ computes

- $k_1 \dots k_P \in \{1 \dots K\}$ where $||x - \mu_{k_1}|| < \dots < ||x - \mu_{k_P}|| < \dots < ||x - \mu_{k_K}||$
- $t(x) \in \mathbb{R}^K$ where $t_k(x) = ||x - \mu_k||^2$. This takes $O(KD)$ time (constant in N).
- $\Delta(x) \in \mathbb{R}^{M \times L}$ where $\Delta_{j,l}(x) = \left\langle \mathbf{sub}(x, j), \nu_l^{(j)} \right\rangle$. This takes $O(LD)$ time (constant in N).
- $y_{\text{IVFPQ}} = y_{i^*}$ where

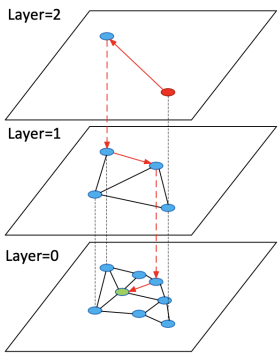
$$i^* = \underset{i \in \cup_{k \in \{k_1 \dots k_P\}} \mathbf{inv}(k)}{\arg\min} \; t_{o_i}(x) + \sum_{j=1}^{M} \pi(j, O_{i,j}) + 2\tau(o_i, j, O_{i,j}) - 2\Delta_{j,O_{i,j}}(x)$$

  This takes $O(N_{\text{IVF}} M)$ time where $N_{\text{IVF}} = \left|\cup_{k \in \{k_1 \dots k_P\}} \mathbf{inv}(k)\right|$.

Thus IVFPQ is roughly $\frac{K}{P} \times \frac{D}{M}$ times faster.

5 Hierarchical Navigable Small World (HNSW) Index

The **HNSW index** (Malkov and Yashunin, 2018) performs graph-based search using a structure that we call HNSW. An HNSW H has $L+1$ layers (or levels) $H.\mathbf{layer}(L) \dots H.\mathbf{layer}(0)$ (top to bottom). The layers nested graphs over $\mathcal{Y}$ that are increasingly more connected. At search time, we start from the top, "step down" layer by layer, and conduct a local search at each layer. The main idea is that the sparsely connected top layers have long-range edges and will allow us to zoom around the entire $\mathcal{Y}$ really fast (corresponding to the top branches of a binary search tree or top levels of a skip list (Pugh, 1990)), and the densely connected lower layers will allow us to do fine-grained search.



We will assume an approximate nearest neighbor search function

$$\mathbf{ANN}(G = (V, E), u, S, \mathbf{dist}, K) \subset V$$

that performs graph-based local search over G from a starting set $S \subset V$ to return K locally optimal vertices $v \in V$ in minimizing $\mathbf{dist}(u, v)$.[3] H is constructed by incrementally adding elements in $\mathcal{Y}$, using the insertion function below.

InsertHNSW
Input: HNSW H with $L+1$ levels, element to insert $u \in \mathbb{R}^D$, metric $\mathbf{dist} : \mathbb{R}^D \times \mathbb{R}^D \to [0, \infty)$, level normalizer m_L, number of nearest neighbors K, oversampling parameter $K_{\text{over}} \geq K$, max node degree at non-bottom layers $R_{\max}$, max node degree at bottom layer $R^0_{\max}$,
Output: updated H, with u inserted and possibly with $> L+1$ levels

1. $S \leftarrow \{H.\mathbf{entry}()\}$
2. $L_u \leftarrow \lfloor z \rfloor$ where $z \sim \text{Exp}(\frac{1}{m_L})$
3. For $l = L \ldots L_u + 1$:
   (a) $S \leftarrow \mathbf{ANN}(H.\mathbf{layer}(l), u, S, \mathbf{dist}, 1)$
4. For $l = \min(L, L_u) \ldots 0$:
   (a) $S_{\text{over}} \leftarrow \mathbf{ANN}(H.\mathbf{layer}(l), u, S, \mathbf{dist}, K_{\text{over}})$
   (b) $S \leftarrow K$ nearest neighbors of u in S_{over}
   (c) $H.\mathbf{vertices}(l) \leftarrow H.\mathbf{vertices}(l) \cup \{u\}$
   (d) $H.\mathbf{edges}(l) \leftarrow H.\mathbf{edges}(l) \cup \{(u, v) : v \in S\}$
   (e) Limit the number of edges at any node to be at most $R_{\max}$ nearest neighbors if $l > 0$, $R^0_{\max}$ if $l = 0$
5. If $L_u > L$, set $H.\mathbf{entry}() \leftarrow u$.

Note that the level L_u is drawn from an exponentially decaying distribution, so the lower layers of H will be exponentially more dense. Once H is constructed, we traverse top to bottom following 1 nearest neighbor until we reach a promising point in the bottom layer, then get K locally optimal neighbors around that point.

SearchHNSW
Input: HNSW H with $L+1$ levels, target $t \in \mathbb{R}^D$, metric $\mathbf{dist} : \mathbb{R}^D \times \mathbb{R}^D \to [0, \infty)$, number of nearest neighbors K, oversampling parameter $K_{\text{over}} \geq K$
Output: K approximate nearest neighbors to t under $\mathbf{dist}$

1. $S \leftarrow \{H.\mathbf{entry}()\}$
2. For $l = L \ldots 1$: $S \leftarrow \mathbf{ANN}(H.\mathbf{layer}(l), t, S, \mathbf{dist}, 1)$
3. Return K nearest neighbors of t in $\mathbf{ANN}(H.\mathbf{layer}(0), t, S, \mathbf{dist}, K_{\text{over}})$.

There are effective heurstics for choosing hyperparameter values. The level normalizer is set to be $m_L = \frac{1}{\log K}$, which implies $\Pr(L_u \geq 1) = \frac{1}{K}$. The max degrees are set to be $R_{\max} = K$ and $R^0_{\max} = 2K$. The main parameter to choose is the number of nearest neighbors K for graph construction: a reasonable choice is $K \in \{16, 32\}$. A reasonable choice of the oversampling parameter is $K_{\text{over}} = 40$ or bigger (e.g., $100, 500$) (smaller for search, e.g., $16, 32, 64, \ldots$).

Storage requirement.

- Original N vectors $\mathcal{Y} \subset \mathbb{R}^D$, so $4DN$ bytes
- About $(m_L R_{\max} + R^0_{\max})N$ edges: this will scale proportional to K with the heuristic above

Since the number of edges grows linearly with N, HNSW is rather memory intensive. It can be combined with PQ in a composite index to avoid storing the original vectors.

[3] For instance, this can be obtained by modifying **KNNGraphSearch** in Appendix C.2.

Runtime. The expected max level of an HNSW over N vectors in $\mathcal{Y}$ is (Appendix E)

$$\mathbf{E}\left[\max_{y\in\mathcal{Y}} L_y\right] = \Theta(\log N)$$

The node degree in each layer is constant in N. The number of steps in **ANN** can also be argued as constant in an idealized scenario.[4] In this case, the search time is $O(\log N)$ and the graph construction time is $O(N \log N)$.

6 Practical Issues

Maximum inner product (MIP). Instead of the nearest neighbor in Euclidean space (NNE) y^* , we often seek

$$y^*_{\text{IP}} = \arg\max_{y\in\mathcal{Y}} \langle x, y\rangle \tag{4}$$

In general, $y^*_{\text{IP}} \neq y^*$ unless every $y \in \mathcal{Y}$ has the same length. But it is possible to transform vectors so that NNE is MIP. Specifically, let $\nu^2_{\max} = \max_{j=1}^N ||y_j||^2$ and redefine query and target vectors to be

$$\tilde{x} = (x, 0) \in \mathbb{R}^{D+1} \qquad \tilde{y}_i = \left(y_i, \underbrace{\sqrt{\nu^2_{\max} - ||y_i||^2}}_{\geq 0}\right) \in \mathbb{R}^{D+1} \qquad \forall i \in \{1 \dots N\}$$

Then $||\tilde{y}_i||^2 = \nu^2_{\max}$ for all i, so that

$$\arg\min_{i=1}^N ||\tilde{x} - \tilde{y}_i||^2 = \arg\min_{i=1}^N \left(||\tilde{x}||^2 + ||\tilde{y}_i||^2 - 2\langle \tilde{x}, \tilde{y}_i\rangle\right) = \arg\min_{i=1}^N \left(||x||^2 + \nu^2_{\max} - 2\langle x, y_i\rangle\right) = \arg\max_{i=1}^N \langle x, y_i\rangle$$

In practice, we can directly compute MIP, so we rarely need to transform MIP to NNE unless we have an index that only supports NNE.

6.1 Experiments

To gain an empirical understanding of the indexes, we conduct experiments on the open-domain version of Natural Questions (NQ) dataset (Karpukhin *et al.*, 2020). We have $N = 21015324$ target embeddings with dimension $D = 768$. For each of 3600 query embeddings, we retrieve top-100 MIP candidates (4) using various indexes implemented in Faiss (Johnson *et al.*, 2019), and calculate Recall@k for $k \in \{1, 5, 20, 100\}$ where a candidate is considered correct iff it contains the answer string for the query. For PQ, we also explore learning a rotation matrix ("OPQ", Appendix A.2.1); but note that Faiss treats OPQ as a preprocessing step, rather than performing full alternating optimization.

index	load/train	search memory	ms/query	R@1	R@5	R@20	R@100
Flat	4m	80.5G	46.5	46.3	68.4	79.5	86.2
IVF(K=1000, P=20)	4m	89.1G	10.5	44.6	65.7	76.2	83.4
IVF(K=1000, P=10)			5.1	42.9	63.7	73.9	81.2
IVF(K=1000, P=100)			49.5	46.3	68.2	79.3	85.8
PQ(M=32, b=8)	6m	21.0G	15.1	20.6	40.2	56.6	72.6
PQ(M=32, b=8) w/ OPQ	9m			36.0	61.0	74.9	84.2
PQ(M=16, b=8)			8.2	7.4	19.9	34.1	52.4
PQ(M=16, b=8) w/ OPQ				25.5	50.2	66.5	80.0
IVFPQ(K=1000, P=100, M=32, b=8)	6m	21.1G	1.8	22.0	42.9	60.1	74.1
IVFPQ(K=1000, P=20, M=32, b=8)			0.4	21.6	42.0	58.6	72.6
IVFPQ(K=1000, P=100, M=32, b=8) w/ OPQ	10m			37.6	61.1	75.2	84.3
IVFPQ(K=45842, P=5000, M=32, b=8)	2h19m		2.5	23.8	45.1	62.4	76.5
HNSW(K=32, K_{over}=500|256)	2h28m	122.0G	0.4	46.1	68.1	79.2	85.7
HNSW(K=32, K_{over}=100|256)	31m			45.5	67.3	78.1	84.5
HNSW(K=32, K_{over}=40|256)	22m			15.7	36.9	55.4	70.2

We see that IVF offers no memory reduction but can improve speed at a modest cost in recall. PQ vastly reduces memory and also improves speed, but recall suffers heavily. With the rotation preprocessing, however, PQ becomes

[4] Assume each layer is an exact Delaunay triangulation. Then **ANN** terminates before it reaches a node that belongs to a higher layer. The level is geometrically distributed independently of N, so each node has some probability p of belonging to the layer above. The expected number of steps in **ANN** is then independent of N.

much more competitive. IVFPQ vastly reduces memory and vastly improves speed, but recall is poor; note that the performance of IVFPQ is fundamentally limited by that of PQ, though residual fitting may give a slight edge. IVFOPQ has a significantly better recall (again, however, it is no better than just OPQ). HNSW has a heavy memory overhead, but is extremely fast and a decisive win in recall, nearing the performance of exhaustive search. HNSW is rather sensitive to K_{over}, the oversampling parameter in graph construction. Training time is negligible except for IVF with large K and HNSW with large K_{over}.

References

Beaumont, O., Kermarrec, A.-M., Marchal, L., and Rivière, É. (2007). Voronet: A scalable object network based on voronoi tessellations. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE.

Eisenberg, B. (2008). On the expectation of the maximum of iid geometric random variables. *Statistics & Probability Letters*, **78**(2), 135–143.

Gärtner, B. and Hoffmann, M. (2013). Computational geometry lecture notes hs 2013. *Dept. of Computer Science, ETH, Zürich, Switzerland*.

Ge, T., He, K., Ke, Q., and Sun, J. (2013). Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2946–2953.

Jegou, H., Douze, M., and Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, **33**(1), 117–128.

Johnson, J., Douze, M., and Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, **7**(3), 535–547.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Kleinberg, J. (2000). The small-world phenomenon: An algorithmic perspective. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 163–170.

Malkov, Y., Ponomarenko, A., Logvinov, A., and Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, **45**, 61–68.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, **42**(4), 824–836.

Norouzi, M. and Fleet, D. J. (2013). Cartesian k-means. In *Proceedings of the IEEE Conference on computer Vision and Pattern Recognition*, pages 3017–3024.

Pugh, W. (1990). Skip lists: a probabilistic alternative to balanced trees. *Communications of the ACM*, **33**(6), 668–676.

A Quantization

Let $C = (\mu_1 \dots \mu_K) \in \mathbb{R}^{K \times D}$ where C is called a **codebook** and $\mu_k \in \mathbb{R}^D$ the k-th **code embedding** or **centroid**. We assume a lower bounded "distortion" function $\delta : \mathbb{R}^D \times \mathbb{R}^D \to [\delta_{\min}, \infty)$. We define

$$\begin{aligned} \mathbf{code} &: \mathbb{R}^D \to \{1 \dots K\} & \mathbf{code}(u) &= \operatorname*{arg\,min}_{k=1}^{K} \delta(u, \mu_k) \\ q &: \mathbb{R}^D \to \mathbb{R}^D & q(u) &= \mu_{\mathbf{code}(u)} \end{aligned} \tag{5}$$

We call q a **quantizer**. We can train q on a set of N points $u_1 \dots u_N \subset \mathbb{R}^D$ to minimize the total distortion:

$$\sum_{i=1}^{N} \delta(u_i, q(u_i)) = \sum_{i=1}^{N} \delta(u_i, \mu_{\mathbf{code}(u_i)}) = \sum_{i=1}^{N} \min_{k=1}^{K} \delta(u_i, \mu_k) = \min_{a_1 \dots a_N \in \{1 \dots K\}} \sum_{i=1}^{N} \delta(u_i, \mu_{a_i}) \tag{6}$$

This problem is generally NP-hard. The K-means clustering algorithm finds an approximate solution by alternating optimization: initialize $\mu_1 \dots \mu_K \in \mathbb{R}^D$ and iterate

$$a_i = \arg\min_{k=1}^{K} \delta(u_i, \mu_k) \qquad \forall i \in \{1 \dots N\} \tag{7}$$

$$\mu_k = \arg\min_{\mu \in \mathbb{R}^D} \sum_{i=1:\ a_i=k}^{N} \delta(u_i, \mu) \qquad \forall k \in \{1 \dots K\} \tag{8}$$

Since each step can never increase the loss, the algorithm will converge to a local optimum (whose quality depends on the initialization scheme). The assignment problem (7) is easy assuming δ is efficiently computable. The centroid optimization (8) is also easy for a large class of distortion functions: in fact, if δ is a Bregman divergence (including the squared Euclidean distance $\delta(u, v) = ||u - v||^2$), (8) is given in closed form by the mean of the points assigned to the k-th code:

$$\mu_k = \frac{1}{|\{i :\ a_i = k\}|} \sum_{i=1:\ a_i=k}^{N} u_i \tag{9}$$

The algorithm is also related to the expectation maximization (EM) algorithm. When δ is the squared Euclidean distance, it coincides with hard EM for a K-mixture of Gaussians where (7) is the E-step and (8) is the M-step.

A.1 Inner Product Variant

If we set $\delta(u, v) = -\langle u, v \rangle$ in (6), the distortion function is unbounded so there is no solution. But we can consider maximizing cosine similarity, which yields the objective

$$\max_{\substack{\mu_1 \dots \mu_K \in \mathbb{R}^D \\ a_1 \dots a_N \in \{1 \dots K\}}} \sum_{i=1}^{N} \frac{\langle u_i, \mu_{a_i} \rangle}{||u_i||\,||\mu_{a_i}||}$$

WLOG assume $||u_i|| = 1$ for all i and constrain the centroids to have unit norm. The objective becomes

$$\max_{\substack{\mu_1 \dots \mu_K \in \mathbb{R}^D:\ ||\mu_k||=1\ \forall k \\ a_1 \dots a_N \in \{1 \dots K\}}} \sum_{i=1}^{N} \langle u_i, \mu_{a_i} \rangle$$

We can again perform alternating optimization: initialize $\mu_1 \dots \mu_K \in \mathbb{R}^D$ with normalization so that $||\mu_k|| = 1$ and iterate

$$a_i = \arg\max_{k=1}^{K} \langle u_i, \mu_{a_i} \rangle \qquad \forall i \in \{1 \dots N\}$$

$$\mu_k = \arg\max_{\mu \in \mathbb{R}^D:\ ||\mu||=1} \sum_{i=1:\ a_i=k}^{N} \langle u_i, \mu \rangle \qquad \forall k \in \{1 \dots K\}$$

It is easy to verify (e.g., by Lagrangian relaxation) that μ_k is given by (9) plus renormalization. This variant is called **spherical K-means**. The only difference from the usual K-means is that we preprocess the inputs to have unit norm and renormalize the centroids in the M-step. It coincides with hard EM for a K-mixture of von Mises-Fisher distributions over a unit sphere.

A.2 Product Quantization

A limitation of basic quantization (10) is that it is difficult to use a large number of clusters since the size of the codebook $C \in \mathbb{R}^{K \times D}$ grows linearly in K. To address this limitation, we can consider decomposing $\mathbb{R}^D$ into M orthogonal subspaces each with dimension $d := D/M$, and quantizing in the subspaces independently. Let $\mathbf{sub}(u, j) \in \mathbb{R}^d$ denote the j-th sub-vector of $u \in \mathbb{R}^D$ (i.e., projection onto the d-dimensional subspace spanned by the orthonormal basis $e_j \dots e_{j+d-1} \in \mathbb{R}^D$). For $j = 1 \dots M$, we compute a "sub-codebook" in the j-th subspace:

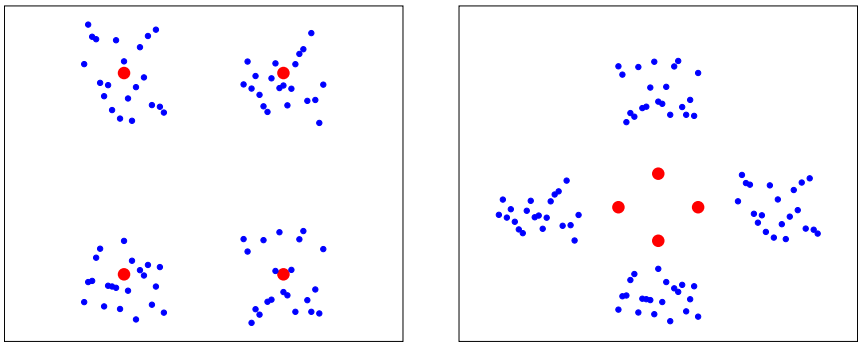
$$C_j = \left(\mu_1^{(j)} \dots \mu_K^{(j)}\right) \in \mathbb{R}^{K \times d} \leftarrow \mathbf{Quantize}\left(\{\mathbf{sub}(u_i, j)\}_{i=1}^{N}, K\right)$$

This is **product quantization (PQ)** (Jegou *et al.*, 2010), where we define

$$\mathbf{code}_j(u) = \arg\min_{k=1}^{K} \delta(\mathbf{sub}(u,j), \mu_k^{(j)}) \qquad q(u) = \begin{bmatrix} \mu^{(1)}_{\mathbf{code}_1(u)} \\ \vdots \\ \mu^{(M)}_{\mathbf{code}_M(u)} \end{bmatrix} \tag{10}$$

Importantly, $q(u) \in \mathbb{R}^D$ can output any of K^M centroid combinations. Since we quantize each subspace independently, the runtime and memory grows linearly in M, but the size of the output space grows exponentially in M.

A limitation of PQ is that it is hardcoded to the standard basis. Consider the examples with $D = 2$ and $M = 2$:



PQ is successful in the first example, but not so in the second.

A.2.1 PQ with rotation.

We can alleviate the limitation of PQ by simultaneously optimizing the basis of $\mathbb{R}^D$. That is, instead of insisting on using the standard basis $I_{D\times D}$, we allow for any orthonormal basis that may yield a smaller quantization error. We can formalize this compactly in matrix form. Let $X = (u_1 \dots u_N) \in \mathbb{R}^{N\times D}$ and $\mathcal{A} = \{(a_1 \dots a_N) \in \{0,1\}^{N\times K} : ||a_i||_1 = 1 \; \forall i\}$. With the square Euclidean distance as distortion, we optimize

$$\min_{\substack{C_1 \dots C_M \in \mathbb{R}^{K\times d} \\ A_1 \dots A_M \in \mathcal{A}^{N\times K} \\ R \in \mathbb{R}^{D\times D}:\, R^\top R = I_{D\times D}}} ||X - [A_1 C_1 \dots A_M C_M] R||_F^2 \tag{11}$$

We can again consider alternating optimization with some initial value of $C_1 \dots C_M$ and orthogonal R (e.g., $R = I_{D\times D}$). Holding R fixed, we optimize $\left|\left|X - [A_j C_j]_{j=1}^M R\right|\right|_F^2 = \left|\left|XR^\top - [A_j C_j]_{j=1}^M\right|\right|_F^2$ over the sub-codebooks C_j and assignments A_j. This can be done by the usual PQ on $XR^\top \in \mathbb{R}^{N\times D}$. Holding $B = [A_1 C_1 \dots A_M C_M] \in \mathbb{R}^{N\times D}$ fixed, we compute the optimal rotation matrix in closed form:

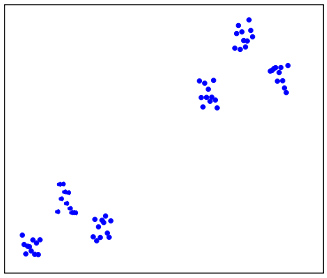
$$R^* = \arg\min_{R\in\mathbb{R}^{D\times D}:\, R^\top R = I_{D\times D}} ||X - BR||_F^2 = \arg\max_{R\in\mathbb{R}^{D\times D}:\, R^\top R = I_{D\times D}} \operatorname{tr}\left(X^\top B R\right) = VU^\top$$

where $U\Sigma V^\top \in \mathbb{R}^{D\times D}$ denotes the SVD of $X^\top B$. The final equality can be quickly verified by von Neumann's trace inequality. Using the fact that the singular values of a rotation matrix are ones, the objective is upper bounded as $\operatorname{tr}\left(X^\top B R\right) \le \operatorname{tr}(\Sigma)$. Setting $R = VU^\top$ achieves this upper bound.

Some history: PQ with rotation was first formalized by Norouzi and Fleet (2013) as **Cartesian K-means**. It was also investigated concurrently by Ge *et al.* (2013), who additionally presented a parametric model and popularized the term **OPQ** (optimized PQ) for this method.

A.3 Hierarchical Variant

If the data is generated hierarchically (e.g., from a hierarchical mixture distribution), we can adapt the quantization scheme to fit the data accordingly. Consider the example:



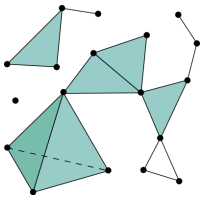
which might have been generated by a mixture of 2 GMMs that are shifted versions of one another. Naively we can use a quantizer with $K = 6$ centroids. We can alternatively consider a hierarchical scheme where we use a "coarse" quantizer $q_c : \mathbb{R}^D \to \{\mu_1, \mu_2\} \subset \mathbb{R}^D$ and a refinement quantizer $q_r : \mathbb{R}^D \to \{\nu_1, \nu_2, \nu_3\} \subset \mathbb{R}^D$. The final quantizer is

$$q(u) = q_c(u) + q_r(u - q_c(u))$$

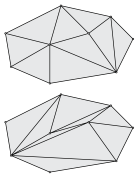
The coarse and refinmenet quantizers can be trained in a pipeline where q_c is trained to minimize the distortion on the original N inputs $u_1 \dots u_N$, then q_r is trained to minimize the distortion on the N residuals $u_1 - q_c(u_1) \dots u_N - q_c(u_N)$ (which have potentially much lower variance).

B Triangulation

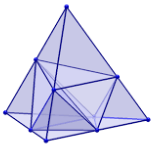
A d-**simplex** $\boldsymbol{\Delta}^d$ is the simplest possible polytope (i.e., geometric object with flat sides) with d dimensions. For instance, $\boldsymbol{\Delta}^0$ is a point, $\boldsymbol{\Delta}^1$ is a line, $\boldsymbol{\Delta}^2$ is a triangle, and $\boldsymbol{\Delta}^3$ is a tetrahedron. We may define it as $\boldsymbol{\Delta}^d = \text{conv}(\text{def}(\boldsymbol{\Delta}^d))$ where $\text{def}(\boldsymbol{\Delta}^d) \subset \mathbb{R}^d$ is a set of $d+1$ affinely independent "defining" points of $\boldsymbol{\Delta}^d$ and conv is the convex hull. Any $S \subset \text{def}(\boldsymbol{\Delta}^d)$ of size $m+1$ defines an associated m-**face** of $\boldsymbol{\Delta}^d$ by $f = \text{conv}(S)$ (thus f is an m-simplex itself). We call 0-faces **vertices**, 1-faces **edges**, $(d-1)$-faces **faucets** of $\boldsymbol{\Delta}^d$ (the d-face is $\boldsymbol{\Delta}^d$). A **simplicial complex** is a set of (possibly various dimensional) simplices $\mathcal{K}$ such that (1) if f is a face of some $\boldsymbol{\Delta} \in \mathcal{K}$, then $f \in \mathcal{K}$, and (2) if $\boldsymbol{\Delta}, \boldsymbol{\Delta}' \in \mathcal{K}$ and $\sigma = \boldsymbol{\Delta} \cap \boldsymbol{\Delta}' \neq \varnothing$, then σ is a face in each of $\boldsymbol{\Delta}$ and $\boldsymbol{\Delta}'$.



simplicial complex



two traingulations of $\mathcal{X} \subset \mathbb{R}^2$



a triangulation of $\mathcal{X} \subset \mathbb{R}^3$

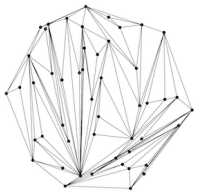
Let $\mathcal{X} \subset \mathbb{R}^d$ be a set of N points. A **(point-set) triangulation** of $\mathcal{X}$ is a (not necessarily unique) simplicial complex that covers $\text{conv}(\mathcal{X})$, with vertices in $\mathcal{X}$. Under a mild non-collinearity assumption, we can always find a triangulation by starting from a singleton and adding each point, connecting it to all vertices of the convex hull of the previously added points. With presorting, it can be done in $O(N \log N)$ time. A triangulation constructed this way is called a **scan triangulation**.

On the plane. In $\mathbb{R}^2$, we can give a simpler definition as a maximal planar (i.e., no edges crossing each other) subdivision whose vertices are $\mathcal{X}$. The number of triangles and edges in any triangulation is counted as $T = 2V - V' - 1$ and $E = 3V - V' - 3$, where $V = N$ and $V' \leq V$ is the number of outermost vertices.[5]

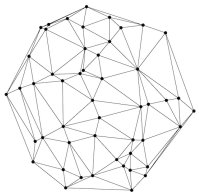
B.1 Delaunay Triangulation

A **Delaunay triangulation** of $\mathcal{X}$, denoted $\mathbf{DT}(\mathcal{X})$, is a triangulation of $\mathcal{X}$ that maximizes the smallest angle between any two edges. In this section, we show examples from Gärtner and Hoffmann (2013).

[5]By Euler's formula for planar graphs, $V - E + F = 2$ where $F = T + 1$ is the number of faces (in this context, triangles plus the unbounded outer region). The number of edge-face pairs can be counted as either $2E$ (each edge borders two faces) or $3(F-1) + V'$ (each triangle borders three edges, and the outer region borders V' outermost edges). The claim can be verified by solving the equations.

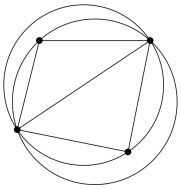


non-Delaunay (scan)

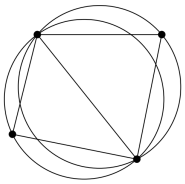


Delaunay

An equivalent characterization of a Delaunay triangulation is as follows: it is a triangulation of $\mathcal{X}$ such that the circumsphere of any $\boldsymbol{\Delta}^d$ (i.e., hypersphere that touches the vertices of $\boldsymbol{\Delta}^d$) in $\mathbf{DT}(\mathcal{X})$ is empty. On the plane, the condition is that the circumcircle of any triangle is empty. Not every triangulation satisfies this condition, and one that does has a larger smallest angle, as shown below.



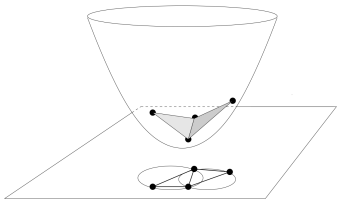
non-Delaunay



Delaunay

Existence and uniqueness of a Delaunay triangulation is guaranteed under a mild "non-cocircularity" condition on $\mathcal{X}$.[6]

Computation. It turns out finding $\mathbf{DT}(\mathcal{X})$ can be reduced to finding the convex hull of the $(d+1)$-dimensional projection of $\mathcal{X}$ onto a hyperboloid $\mathcal{X}_{\text{lifted}} = \{(x, ||x||^2) \in \mathbb{R}^{d+1} : \ x \in \mathcal{X}\}$, then projecting it back to the first d dimensions (i.e., discard the last dimension). This leads to an incremental algorithm called the **Lawson flip algorithm** that starts with some triangulation and modifying a subtriangulation of $d+2$ points that violates the condition, which corresponds to "segmenting" the bottom part of the projection.

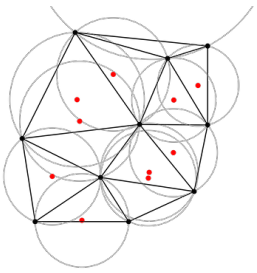


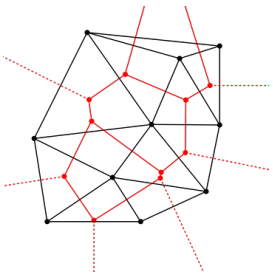
The algorithm takes $O(N^2)$ time. There are other incremental algorithms that take $O(N \log N)$. Unfortunately, the number of $\boldsymbol{\Delta}^d$ in $\mathbf{DT}(\mathcal{X})$ is $O(N^d)$, so the runtime of any algorithm is exponential in dimension (since the number of neighbors to maintain grows exponentially in d).

Duality. Under mild conditions, there is a one-to-one correspondence between $\mathbf{DT}(\mathcal{X})$ and the **Voronoi diagram** of $\mathcal{X} = \{x_1 \dots x_N\}$, denoted by $\mathbf{VD}(\mathcal{X})$, which induces a partition of $\mathbb{R}^d$ into N cells where the i-th cell is $\mathbf{VD}_i(\mathcal{X}) = \{u \in \mathbb{R}^d : \ ||u - x_i|| \leq ||u - x_j|| \ \forall j \neq i\}$. Specifically,

- Each *circumcenter* of $\boldsymbol{\Delta}^d \in \mathbf{DT}(\mathcal{X})$ corresponds to a vertex in $\mathbf{VD}(\mathcal{X})$.
- Two vertices in $\mathbf{VD}(\mathcal{X})$ are connected iff the corresponding simplices in $\mathbf{DT}(\mathcal{X})$ are adjacent.

A Delaunay triangulation with circumcenters and the corresponding Voronoi diagram are shown below (figures from Wikipedia):





[6] The affine hull is d-dimensional and no set of $d+2$ points lie on the boundary of a ball whose interior does not intersect $\mathcal{X}$.

It follows that $\mathbf{DT}(\mathcal{X})$ is the minimal connected planer graph over $\mathcal{X}$ such that when we move from one vertex to another, we move to the corresponding neighboring Voronoi cell. Note that it is different from the nearest neighbor graph (NNG) which is not guaranteed to be connected, or the Euclidean minimum spanning tree (EMST). But NNG and EMST are subgraphs of $\mathbf{DT}(\mathcal{X})$.

C Graph-Based Search

Suppose we have N target vectors and are interested in finding the target closest to a given query under some metric. One way to avoid exhaustive search is to assume a graph structure over the targets and, starting from some initial vertex, greedily optimize the metric by moving to a neighboring target.

GreedyGraphSearch
Input: graph $G = (V, E)$ where $V \subset \mathbb{R}^d$ and $N = |V|$, metric $\mathbf{dist} : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$, target vertex $t \in \mathbb{R}^d$, initial vertex $s \in V$
Output: estimation of $u^* = \arg\min_{u \in V} \mathbf{dist}(u, t)$

1. Initialize $u^{(0)} \leftarrow s$ and $n \leftarrow 0$.
2. While there exists $v \in V$ such that $(u, v) \in E$ and $\mathbf{dist}(v, t) < \mathbf{dist}(u, t)$: set $n \leftarrow n + 1$ and
$$u^{(n)} \leftarrow \underset{v \in V:\ (u^{(n-1)}, v) \in E}{\arg\min} \mathbf{dist}(v, t)$$
3. Return the locally optimal $u^{(n)} \approx u^*$ and the number of steps $n \leq N$

Generally there is no guarantee that $u^{(n)}$ is globally optimal. The graph might not even be complete so that there is no path between s and t. Even if the graph is complete, the algorithm may converge to a local but not global optimum depending on the edge structure.

C.1 Navigable Small World Graphs

Kleinberg considers a *random* graph over N points on the plane that contains the N by N grid as a subgraph plus some stochastic "long-range" edges and asks if the graph has the following properties:

1. (Small) There exists a "short" path between any two vertices with length $O(\log N)$.
2. (Navigable) **GreedyGraphSearch** follows a short path.

Note that the greedy search will eventually return the target vertex thanks to the grid subgraph. A graph that satisfies both properties is called a **navigable small world**. It is possible that a graph is small but not navigable. For instance, if the long-range edges are drawn uniformly at random, random graph theory gives us that there exists a short path between any two vertices, but the greedy algorithm will not find it (special case of the following theorem with $r = 0$).

Theorem C.1 (Kleinberg (2000))**.** Let $G = (V, E)$ be a graph where $V = \{1 \dots N\}^2$, the distance between two vertices is measured by the number of hops $\mathbf{dist}(u, v) = ||u - v||_1$, and E is constructed stochastically as follows: for each $u \in V$

- Add (u, v) for every $v \in V$ within p hops: $\mathbf{dist}(u, v) \leq p$.
- For q times, add (u, v) where $v \in V$ is sampled with probability $\frac{\mathbf{dist}(u,v)^{-r}}{\sum_{v' \in V} \mathbf{dist}(u,v')^{-r}}$.

Draw $s, t \in V$ uniformly at random and run $u^{(n)} = t \leftarrow \mathbf{GreedyGraphSearch}(G, \mathbf{dist}, t, s)$. Then

- If $r \neq 2$, then $\mathbf{E}[n] = \Omega(N^c)$ for some c that depends on r.
- If $p = q = 1$ and $r = 2$, then $\mathbf{E}[n] = O(\log N)$.

Note that $r = 2$ is the only value that yields navigability. The result has been generalized to d-dimensional grids where we need $r = d$ (i.e., a long-range edge (u, v) is sampled with probability $\propto \mathbf{dist}(u, v)^{-d}$). The result has also been generalized to non-grid topologies. For instance, a Delaunay triangulation can be made a navigable small world by inserting appropriate long-range random edges (Beaumont *et al.*, 2007).

C.2 Approximate Navigable Small Delaunay Triangulation

The conceptually ideal graph for greedy nearest neighbor search is a navigable small Delaunay triangulation, since

1. The Delaunay triangulation is the minimal graph (i.e., has fewest edges) that allows for exact greedy search. Due to its duality to the Voronoi diagram, every step of **GreedyGraphSearch** over the graph will strictly decrease the distance between the query and the global optimum until convergence.

2. If the graph is navigable small, **GreedyGraphSearch** will find it in $O(\log N)$ time. Thus we achieve exact search in time logarithmic in N.

However, in practice we do not need this precise ideality. Even if the graph is faulty so that local search tends to get stuck in a local optimum, we can combat it by (1) trying many times from random initial points, and (2) keeping K hypotheses instead of one (which is probably what we want even if the search is exact since the topology itself might be faulty). To this end, Malkov *et al.* (2014) propose a K-nearest-neighbor graph search heurstic, detailed below.

KNNGraphSearch
Input: graph $G = (V, E)$ where $V \subset \mathbb{R}^d$ and $N = |V|$, metric $\mathbf{dist} : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$, target vertex $t \in \mathbb{R}^d$, number of random searches m, number of nearest neighbors K
Output: K approximate nearest neighbors to t under **dist**
Data structure: TreeSet for storing a set K vectors $u \in \mathbb{R}^d$ in increasing $\mathbf{dist}(t, u)$ (it uses a self-balancing binary search tree (Red-Black) to enable $O(\log N)$ time add/search operations)

1. Initialize $S \leftarrow \text{TreeSet}(\varnothing)$.
2. For m times:
   (a) Initialize $S_{\text{trial}} \leftarrow \text{TreeSet}(\{s\})$ where $s \sim \text{Unif}(V)$.
   (b) Loop:
      i. $u \leftarrow S_{\text{trial}}.\mathbf{pop}()$
      ii. If $\mathbf{dist}(u, t) > S.\mathbf{last}()$: **break**
      iii. Add every unvisited neighbor of u to S_{trial}.
   (c) $S.\mathbf{add}(S_{\text{trial}}.\mathbf{dump}())$
3. Return $S.\mathbf{dump}() \subset V$.

The K-NN search can be used to *build* a connected graph:

BuildKNNGraph
Input: $\mathcal{X} \subset \mathbb{R}^d$, metric $\mathbf{dist} : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$, number of random searches m, number of nearest neighbors K
Output: graph $G = (V, E)$, intended as an approximation of a navigable small Delaunay triangulation $\mathbf{DT}(V)$

1. Initialize $V \leftarrow \varnothing$ and $E \leftarrow \varnothing$.
2. For $x \in \mathcal{X}$ in some order:
   (a) $S \leftarrow \mathbf{KNNGraphSearch}(G = (V, E), \mathbf{dist}, x, m, K)$
   (b) Set $V \leftarrow V \cup \{x\}$ and $E \leftarrow E \cup \{(u, x) : u \in S\}$.
3. Return $G = (V, E)$.

It is empirically shown that $G \leftarrow \mathbf{BuildKNNGraph}(\mathcal{X}, ||u - v||, 20, 10)$ (i.e., 20 random restarts, 10 nearest neighbors in the Euclidean space) needs on average $O(\log N)$ hops between two nodes for various values of d. Furthermore, both the search and graph construction can be parallelized. The authors report 0.999 recall while visiting only 0.031% of the vertices, with a suitable choice of hyperparameters.

D Geometric Distribution

We say $X \in \{0, 1, 2, \ldots\}$ is **geometrically distributed** with parameter $0 < p \leq 1$ and write $X \sim \text{Geo}(p)$ if

$$\Pr(X = k) = (1 - p)^k p$$

(i.e., X is the number of failures before we get a success where the success probability is p). The mean and variance is $\mathbf{E}[X] = \frac{1-p}{p}$ and $\text{Var}(X) = \frac{1-p}{p^2}$. We say $Y \in [0, \infty)$ is **exponentially distributed** with parameter $\lambda > 0$ and write $Y \sim \text{Exp}(\lambda)$ if

$$\Pr(Y = y) = \lambda e^{-\lambda y}$$

The mean and variance is $\mathbf{E}[Y] = \frac{1}{\lambda}$ and $\text{Var}(X) = \frac{1}{\lambda^2}$. The floor of Y is geometrically distributed:[7]

$$\lfloor Y \rfloor \sim \text{Geo}\left(1 - e^{-\lambda}\right) \tag{12}$$

If $U \in [0, 1]$ is uniformly distributed, the following also holds:

$$\left\lfloor \frac{\log(U)}{\log(1-p)} \right\rfloor \sim \text{Geo}(p)$$

In particular, setting $p = 1 - e^{-\frac{1}{m}}$ where $m > 0$ yields

$$T = \lfloor -\log(U)m \rfloor \sim \text{Geo}\left(1 - e^{-\frac{1}{m}}\right)$$

Then by the relationship (12) we also have

$$T = \lfloor Z \rfloor \qquad\qquad Z \sim \text{Exp}\left(\frac{1}{m}\right)$$

where $\mathbf{E}[Z] = m$ and $\text{Var}(Z) = m^2$.

E Expected Maximum

E.1 Exact Solution

Lemma E.1. Let $X_1 \dots X_N \sim \text{Exp}(\lambda)$ be iid where $\lambda > 0$. Then

$$\mathbf{E}\left[\max_{i=1}^{N} X_i\right] = \frac{1}{\lambda} \sum_{k=1}^{N} \frac{1}{k} \tag{13}$$

Proof I. For any $x \geq 0$, we have the CDF $\Pr(X_i \leq x) = 1 - e^{-\lambda x}$. Letting $Y := \max_{i=1}^{N} X_i$ and using the independence, we have $\Pr(Y \leq x) = (1 - e^{-\lambda x})^N$. Then

$$\begin{aligned}
\mathbf{E}[Y] &= \int_0^\infty \Pr(Y > x) dx && \text{(expectation in CDF)} \\
&= \int_0^\infty 1 - (1 - e^{-\lambda x})^N dx \\
&= \frac{1}{\lambda} \int_0^1 \frac{1 - u^N}{1 - u} du && (u\text{-substitution: } u = 1 - e^{\lambda x}) \\
&= \frac{1}{\lambda} \int_0^1 \sum_{k=0}^{N-1} u^k du && \text{(geometric sum)} \\
&= \frac{1}{\lambda} \sum_{k=0}^{N-1} \left(\frac{1}{k+1} u^{k+1} \right) \Big|_0^1 = \frac{1}{\lambda} \sum_{k=1}^{N} \frac{1}{k}
\end{aligned} \tag{14}$$

Details of the u-substitution are as follows: $\int_0^\infty f(g(x))dx = \int_{g(0)}^{g(\infty)} \frac{f(u)}{g'(g^{-1}(u))} du$ where $f(u) = 1 - u^N$ and $g(x) = 1 - e^{-\lambda x}$, so that $g^{-1}(u) = -(1/\lambda)\log(1-u)$ and $g'(x) = \lambda e^{-\lambda x}$ yield $g'(g^{-1}(u))^{-1} = \lambda(1-u)$. □

[7] Aside: with $\lambda = 1$, we have the probability $1 - e^{-1}$ of selecting a particular element in a set of N unique items in N random draws as $N \to \infty$, since $\lim_{N\to\infty}(1 - \frac{1}{N})^N = \frac{1}{e}$.

Proof II. Let $X_{(i)}$ denote the i-th order statistic (i.e., i-th smallest value) of $\{X_1 \dots X_N\}$. Since $X_{(N)} = \max_{i=1}^N X_i$, we can write

$$\begin{aligned}\max_{i=1}^N X_i &= X_{(1)} + (X_{(2)} - X_{(1)}) + \cdots + (X_{(N)} - X_{(N-1)}) \\ &= X_{(1)} + S_1 + \cdots + S_{N-1}\end{aligned}$$

where $S_i := X_{(i+1)} - X_{(i)}$. For $i \in \{1 \dots N-1\}$, given $t \geq 0$ we have

$$\begin{aligned}\Pr(S_i > t) &= \Pr(X_{(i+1)} - X_{(i)} > t) \\ &= \Pr\left(\min\{X_1 \dots X_{N-i}\} - X_{(i)} > t \,\middle|\, \min\{X_1 \dots X_{N-i}\} \geq X_{(i)}\right) \\ &= \Pr\left(Z_i > X_{(i)} + t \,\middle|\, Z_i \geq X_{(i)}\right) \\ &= \Pr(Z_i > t) \qquad (15)\end{aligned}$$

where (15) follows from the fact that $Z_i := \min\{X_1 \dots X_{N-i}\}$ is distributed as $\text{Exp}((N-i)\lambda)$ and, consequently, is memoryless. Thus

$$\begin{aligned}\mathbf{E}\left[\max_{i=1}^N X_i\right] &= \mathbf{E}[X_{(1)}] + \mathbf{E}[S_1] + \cdots + \mathbf{E}[S_{N-1}] \\ &= \mathbf{E}[Z_0] + \mathbf{E}[Z_1] + \cdots + \mathbf{E}[Z_{N-1}] \\ &= \frac{1}{N\lambda} + \frac{1}{(N-1)\lambda} + \cdots + \frac{1}{\lambda} = \frac{1}{\lambda}\sum_{k=1}^N \frac{1}{k}\end{aligned}$$

Since this also shows that $Z_0 \dots Z_{N-1}$ are independent, we can also give the variance formula

$$\text{Var}\left(\max_{i=1}^N X_i\right) = \text{Var}(Z_0 + Z_1 + \cdots + Z_{N-1}) = \sum_{j=0}^{N-1} \text{Var}(Z_i) = \sum_{j=0}^{N-1} \frac{1}{(N-j)^2\lambda^2} = \frac{1}{\lambda^2}\sum_{k=1}^N \frac{1}{k^2}$$

□

Even though the geometric distribution can be obtained by discretizing the exponential distribution, it has no simple closed-form like (13). See Eisenberg (2008) for an exact solution. But it is possible to give a simple bound.

Lemma E.2. Let $X_1 \dots X_N \sim \text{Geo}(p)$ be iid where $0 < p < 1$. Then for $\lambda = \log(\frac{1}{1-p}) > 0$,

$$\mathbf{E}\left[\max_{i=1}^N X_i\right] \in \left(\frac{1}{\lambda}\sum_{k=1}^N \frac{1}{k},\ 1 + \frac{1}{\lambda}\sum_{k=1}^N \frac{1}{k}\right) \qquad (16)$$

Proof. For any $x \geq 0$, we have the CDF $\Pr(X_i \leq x) = 1 - (1-p)^{x+1}$. Letting $Y := \max_{i=1}^N X_i$ and using the independence, we have $\Pr(Y \leq x) = (1 - (1-p)^{x+1})^N$. Then

$$\begin{aligned}\mathbf{E}[Y] &= \sum_{x=0}^\infty \Pr(Y > x) && \text{(expectation in CDF)} \\ &= \sum_{x=0}^\infty 1 - (1 - (1-p)^{x+1})^N \\ &= \sum_{x=0}^\infty 1 - (1 - e^{-(x+1)\lambda})^N && (p = 1 - e^{-\lambda}) \\ &\in \left(\int_0^\infty 1 - (1 - e^{-(x+1)\lambda})^N dx,\ 1 - (1 - e^{-\lambda})^N + \int_0^\infty 1 - (1 - e^{-(x+1)\lambda})^N dx\right) && \text{(integral test)} \\ &\in \left(\int_0^\infty 1 - (1 - e^{-(x+1)\lambda})^N dx,\ 1 + \int_0^\infty 1 - (1 - e^{-(x+1)\lambda})^N dx\right) \\ &= \left(\frac{1}{\lambda}\sum_{k=1}^N \frac{1}{k},\ 1 + \frac{1}{\lambda}\sum_{k=1}^N \frac{1}{k}\right) && (14)\end{aligned}$$

□

We use Euler's constant $\gamma \approx 0.577$ which satisfies

$$\gamma = \lim_{N\to\infty} \left(\sum_{k=1}^{N} \frac{1}{k}\right) - \log N$$

to give the following corollaries.

Corollary E.3. Let $\lambda > 0$. For a large enough N,

$$\mathop{\mathbf{E}}_{x_1 \dots x_N \sim \text{Exp}(\lambda)} \left[\max_{i=1}^{N} x_i\right] \approx \frac{1}{\lambda}(\log N + 0.577)$$

Corollary E.4. Let $0 < p < 1$ and $\lambda = \log(\frac{1}{1-p}) > 0$. For a large enough N,

$$\frac{1}{\lambda}(\log N + 0.577) \lessapprox \mathop{\mathbf{E}}_{x_1 \dots x_N \sim \text{Geo}(p)} \left[\max_{i=1}^{N} x_i\right] \lessapprox 1 + \frac{1}{\lambda}(\log N + 0.577)$$

E.2 Upper Bounds

Fact E.5 (AM-GM inequality)**.** For any $x_1 \dots x_N \geq 0$

$$\frac{1}{N}\sum_{i=1}^{N} x_i \geq \left(\prod_{i=1}^{N} x_i\right)^{1/N}$$

with equality iff $x_1 = \cdots = x_N$.

Lemma E.6. Let $N \geq 2$ and $X_1 \dots X_N$ be random variables, each with an MGF $m_i(t) := \mathbf{E}[\exp(tX_i)]$ for $t \in \mathbf{Dom}(m_i) \subset (-\infty, \infty)$. Let m be an upper bound $m(t) \geq m_i(t)$ for all $t \in \mathbf{Dom}(m) := \cap_{i=1}^{N} \mathbf{Dom}(m_i)$. Let $\mathbf{Dom}_{>0}(m) := \mathbf{Dom}(m) \cap (0, \infty)$. For any $t^* \in \mathbf{Dom}_{>0}(m)$ such that $m(t^*) = N$:

$$\mathbf{E}\left[\max_{i=1}^{N} X_i\right] \leq \frac{2\log N}{t^*}$$

Proof. Let $Y := \max_{i=1}^{N} X_i$. For any $t \in \mathbf{Dom}_{>0}(m)$

$$\exp\left(t\mathbf{E}\left[Y\right]\right) \leq \mathbf{E}\left[\exp\left(tY\right)\right] = \mathbf{E}\left[\max_{i=1}^{N} \exp\left(tX_i\right)\right] \leq \sum_{i=1}^{N} \mathbf{E}\left[\exp\left(tX_i\right)\right] \leq Nm(t)$$

where the first inequality is Jensen's (using the fact $t > 0$), the equality follows from the monotonicity of exp, the second inequality uses the positivity of exp and also the fact $t \in \mathbf{Dom}(m_i)$ for all $i = 1 \dots N$, and the final inequality is the premise. Taking the log on both sides yields

$$\mathbf{E}\left[Y\right] \leq \frac{\log N + \log m(t)}{t} \qquad \forall t \in \mathbf{Dom}_{>0}(m) \tag{17}$$

For t such that $m(t) > 1$, we can use the AM-GM inequality to lower bound the right hand side:

$$\frac{\log N + \log m(t)}{t} = \frac{\frac{2\log N}{t} + \frac{2\log m(t)}{t}}{2} \geq \frac{2\sqrt{\log N \times \log m(t)}}{t} \qquad \forall t \in \mathbf{Dom}_{>0}(m) :\ m(t) > 1$$

The bound is tight for $t \in \mathbf{Dom}_{>0}(m)$ with $m(t) = N$. Plugging it in (17), we have $\mathbf{E}\left[Y\right] \leq \frac{2\log N}{t}$ for all such t. □

Examples. We can solve for $t^* > 0$ when $X_1 \dots X_N$ are identical samples of a known distribution.

- $X_i \sim \mathcal{N}(0, \sigma^2)$ where $\sigma^2 > 0$: The MGF is $m(t) = \exp(\frac{\sigma^2 t^2}{2})$ with domain $(-\infty, \infty)$. When we solve $m(t) = N$ we get $t^* = \frac{\sqrt{2\log N}}{\sigma} \in (0, \infty)$, hence

$$\mathop{\mathbf{E}}_{x_1 \dots x_N \sim \mathcal{N}(0,\sigma^2)} \left[\max_{i=1}^{N} x_i\right] \leq \sigma\sqrt{2\log N}$$

- $X_i \sim \text{Gamma}(k, \theta)$ where $k, \theta > 0$: The MGF is $m(t) = (1 - \theta t)^{-k}$ with domain $(-\infty, \frac{1}{\theta})$. When we solve $m(t) = N$ we get $t^* = \frac{1 - N^{-1/k}}{\theta} \in (0, \frac{1}{\theta})$, hence

$$\mathop{\mathbf{E}}_{x_1 \ldots x_N \sim \text{Gamma}(k,\theta)} \left[\max_{i=1}^{N} x_i \right] \leq \frac{2\theta \log N}{1 - N^{-1/k}}$$

  Since $\text{Exp}(\lambda) = \text{Gamma}(1, \frac{1}{\lambda})$, we also have

$$\mathop{\mathbf{E}}_{x_1 \ldots x_N \sim \text{Exp}(\lambda)} \left[\max_{i=1}^{N} x_i \right] \leq \frac{2 \log N}{\lambda(1 - \frac{1}{N})}$$

- $X_i \sim \text{Geo}(p)$ where $0 < p \leq 1$: The MGF is $m(t) = \frac{p}{1 - (1-p)e^t}$ with domain $(-\infty, -\log(1-p))$. When we solve $m(t) = N$ we get $t^* = \log(1 - \frac{p}{N}) - \log(1-p) \in (0, -\log(1-p))$, hence

$$\mathop{\mathbf{E}}_{x_1 \ldots x_N \sim \text{Geo}(p)} \left[\max_{i=1}^{N} x_i \right] \leq \frac{2 \log N}{\log(1 - \frac{p}{N}) - \log(1-p)}$$