The kernel of graph indices for vector search

Anonymous authors
Paper under double-blind review

Abstract

The most popular graph indices for vector search use principles from computational geometry to build the graph. Hence, their formal graph navigability guarantees are only valid in Euclidean space. In this work, we show that machine learning can be used to build graph indices for vector search in metric and non-metric vector spaces (e.g., for inner product similarity). From this novel perspective, we introduce the Support Vector Graph (SVG), a new type of graph index that leverages kernel methods to establish the graph connectivity and that comes with formal navigability guarantees valid in metric and non-metric vector spaces. In addition, we interpret the most popular graph indices, including HNSW and DiskANN, as particular specializations of SVG and show that new navigable indices can be derived from the principles behind this specialization. Finally, we propose SVG-L0 that incorporates an ℓ_0 sparsity constraint into the SVG kernel method to build graphs with a bounded out-degree. This yields a principled way of implementing this practical requirement, in contrast to the traditional heuristic of simply truncating the out edges of each node. Additionally, we show that SVG-L0 has a self-tuning property that avoids the heuristic of using a set of candidates to find the out-edges of each node and that keeps its computational complexity in check.

1 Introduction

Vector search has become a critical component of AI infrastructure. For example, in retriever-augmented generation (RAG) (Lewis et al., 2020), vector search is used to ground knowledge and prevent hallucinations. The literature on vector search evolves quickly, trying to keep up with ever-increasing requirements: more vectors with larger dimensionality, higher speeds, and lower deployment costs, all while maintaining high accuracy. The most classical methods, trees (Beygelzimer et al., 2006; Navarro, 2002; Krauthgamer & Lee, 2004; Bentley, 1975) and hashing (Wang et al., 2018; Jafari et al., 2021), often struggle to achieve high accuracy at high speeds. Inverted indices (Muja & Lowe, 2014; Johnson et al., 2021) are simple to build, but involve a large number of similarity computations.

In recent years, graph-based indices (e.g., Dearholt et al., 1988; Arya & Mount, 1993; Malkov & Yashunin, 2020; Fu et al., 2019; Subramanya et al., 2019) strike an excellent balance of accuracy and speed and, as a consequence, have been widely deployed in the real world with great success. Here, a directed graph, where each vertex corresponds to a database vector and edges represent neighbor-relationships between vectors, is efficiently traversed to find the (approximate) nearest neighbors of a query vector in sublinear time. The graph edges need to be carefully selected to ensure that this traversal yields correct results (i.e., the graph is navigable). Starting with the seminal works by Dearholt et al. (1988) and Arya & Mount (1993), navigable graphs are built using computational geometry principles to perform edge selection. The Delaunay graph is a fully navigable triangulation (Wang et al., 2021) but it is too dense in higher dimensions. Informally, graph-building algorithms sparsify the graph by examining these triangles and only keeping a small subset. The Delaunay graph and the triangle pruning rules are defined in Euclidean space, which limits the navigability guarantees of the resulting graphs to this specific case. However, these algorithms are commonly used in non-Euclidean spaces, where their underlying principles do not hold, to build graphs that work well in practice but lack formal guarantees.

In this work, we analyze graph indices from a new perspective: we rely on machine learning instead of computational geometry. In particular, we study graph indices formally in light of kernel methods. In this setting, we have a similarity function $\text{sim}(\mathbf{x}, \mathbf{x}') : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ and an associated kernel $K(\mathbf{x}, \mathbf{x}') = h(\text{sim}(\mathbf{x}, \mathbf{x}'))$, where h is a (possibly) nonlinear function. Throughout this work, we assume that the kernel is positive semidefinite (PSD), i.e., the feature expansion $K(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\top \phi(\mathbf{x}')$ is valid for (possibly infinite-dimensional) feature vectors $\phi(\mathbf{x})$. The exponential kernel is an important class of kernels, widely used in experimental and theoretical studies,

$$K_{\text{EXP}}(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=} \exp\left(\text{sim}(\mathbf{x}, \mathbf{x}')/\sigma^2\right), \tag{1}$$

where the hyperparameter $\sigma > 0$ is implicit. Two leading examples are given by the similarity functions

$$\text{sim}_{\text{EUC}}(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=} -\|\mathbf{x} - \mathbf{x}'\|_2^2, \tag{2}$$

$$\text{sim}_{\text{DP}}(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=} \mathbf{x}^\top \mathbf{x}' \tag{3}$$

that define the standard Radial Basis Function (a.k.a. Gaussian) and exponential dot product kernels, respectively. The similarity sim_{DP} corresponds to the commonly used maximum inner product (MIP) retrieval problem. The exponential kernel is a natural choice, as it is used in the training loss (e.g., the entropy loss) of the embedding models that produce the vectors commonly used in practice (Radford et al., 2021; Karpukhin et al., 2020). Note that defining $\text{sim}(\mathbf{x}, \mathbf{x}') \stackrel{\text{def}}{=} -\text{dist}(\mathbf{x}, \mathbf{x}')^2$ for any distance function (e.g., Manhattan, Hamming, etc.) yields a valid exponential kernel.

Using kernels as our vantage point, we present the following contributions (all proofs in the appendix):

- We propose a new graph index, the Support Vector Graph (SVG). This new index arises from a novel perspective on graph indices (Section 2) and an accompanying formulation that models graph construction as a kernelized nonnegative least squares (NNLS) problem. This NNLS is equivalent to a support vector machine (SVM) whose support vectors provide the connectivity of the graph (Section 3).
- We provide formal results that show the navigability of the SVG for general PSD kernels (Section 3.1). To the best of our knowledge, this is the first graph index that is navigable in non-Euclidean vector spaces.
- We derive an interpretation of the most popular graph indices as SVG specializations, where the SVG optimization problem is used within the aforementioned traditional triangle pruning approach (Section 4). In particular, our results cover the popular HNSW (Malkov & Yashunin, 2020) and DiskANN (Subramanya et al., 2019). We also show that new triangle-pruning algorithms, valid in Euclidean and non-Euclidean spaces, can be derived from the principles behind this specialization. We prove that these algorithm variants lead to other new navigable indices.
- Finally, we address the construction of graphs with a bounded out-degree, a common feature in most practical deployments. For this, we propose SVG-L0 that includes a hard sparsity (ℓ_0) constraint in the SVG optimization (Section 5). SVG-L0 yields a principled way of handling the requirement, in contrast with the traditional heuristic, which simply truncates the out edges of each node. Additionally, we show that SVG-L0 has a self-tuning property, which avoids setting a set of candidate edges for each graph node and yet still has a computational complexity sublinear in the number of indexed vectors.

Although this work focuses on a formal analysis of SVG and SVG-L0, we also present some preliminary empirical results to show that the proposed techniques have practical value beyond their theoretical merits (Section 6). For reproducibility, we make our implementation available at `[anonymized_url]`.

Notation. We denote the set of natural numbers from 1 to n by $[1 \dots n]$. We denote vectors/matrices by lowercase/uppercase bold letters, e.g., $\mathbf{v} \in \mathbb{R}^n$ and $\mathbf{A} \in \mathbb{R}^{m \times n}$. Individual entries of a matrix $\mathbf{A}$ (resp. vector $\mathbf{v}$) are denoted by $\mathbf{A}_{[ij]}$ (resp. v_i). The i-th row and column of $\mathbf{A}$ are denoted by $\mathbf{A}_{[i:]}$ and $\mathbf{A}_{[:i]}$, respectively. The matrix containing a subset $\mathcal{I} \subset [1 \dots m]$ (resp. $\mathcal{J} \subset [1 \dots n]$) of the rows (resp. columns) of $\mathbf{A} \in \mathbb{R}^{m \times n}$ is denoted by $\mathbf{A}_{[\mathcal{I}:]}$ (resp. $\mathbf{A}_{[:\mathcal{J}]}$). A directed graph $G = ([1 \dots n], \mathcal{E})$ is composed by the node set $[1 \dots n]$ and the edge/vertex set $\mathcal{E}$, i.e., a set of ordered pairs $\overrightarrow{ij}$ with $i, j \in [1 \dots n]$. We define the neighborhood of node i as $\mathcal{N}_i \stackrel{\text{def}}{=} \left\{ j \mid \overrightarrow{ij} \in \mathcal{E} \right\}$. A path $[v_1, \cdots, v_l]$ in G is a list of nodes such that $(\forall i = 1, \cdots, l-1)$ $\overrightarrow{v_i v_{i+1}} \in \mathcal{E}$.

Algorithm 1: Greedy graph search

Input : Query $\mathbf{q} \in \mathbb{R}^d$, dataset $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$, graph $G = ([1 \dots n], \mathcal{E})$, entry point i_{ep}.
Output: Approximate nearest neighbor i^*.

$i^* \leftarrow i_{\text{ep}}$;
Repeat
  $i \leftarrow \underset{j \in \mathcal{N}_{i^*}}{\operatorname{argmax}} \operatorname{sim}(\mathbf{q}, \mathbf{x}_j)$; `//` $\mathcal{N}_{i^*}$ `is the neighborhood of` i^*
  if $\operatorname{sim}(\mathbf{q}, \mathbf{x}_i) > \operatorname{sim}(\mathbf{q}, \mathbf{x}_{i^*})$ **then** $i^* \leftarrow i$; `// progress, continue`
  else return i^*; `// no progress, exit`

2 Graph indices for vector search in Euclidean Space

Using navigable graphs for vector search has a long history (Dearholt et al., 1988; Arya & Mount, 1993) but only became prominent in the last ten years (e.g., Subramanya et al., 2019; Malkov & Yashunin, 2020) with the increasing scale of unstructured data. Navigability is defined as the ability to reach any node when conducting a greedy graph traversal (Algorithm 1) using that node as the query. The following definitions from the literature formalize this concept for the Euclidean distance. In this case, the similarities in Algorithm 1 are transformed into Euclidean distances by switching the maximization of the similarity with the minimization of the distance.

Definition 1 (Monotonic Path (Fu et al., 2019))**.** *Given a set of n vectors $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$, let $G = ([1 \dots n], \mathcal{E})$ denote a directed graph and $s, t \in [1 \dots n]$ be two nodes of G. A path $[v_1, \cdots, v_l]$ from $s = v_1$ to $t = v_l$ in G is a monotonic path if and only if $(\forall i = 1, \cdots, l-1)$ $\|\mathbf{x}_{v_i} - \mathbf{x}_t\|_2 > \left\|\mathbf{x}_{v_{i+1}} - \mathbf{x}_t\right\|_2$.*

Definition 2 (Monotonic Search Network (Fu et al., 2019))**.** *Given a set of n vectors $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ and the kernel K, a graph $G = ([1 \dots n], \mathcal{E})$ is a generalized monotonic search network if and only if there exists at least one monotonic path from s to t for any two nodes $s, t \in [1 \dots n]$.*

A Monotonic Search Network (MSNet) is a navigable graph as demonstrated by the following lemma.

Lemma 1 (Fu et al., 2019)**.** *Let $G = ([1 \dots n], \mathcal{E})$ be a monotonic search network. Let $s, t \in [1 \dots n]$, then Algorithm 1 with $\mathbf{x}_t$ as the query and s as the entry point finds a monotonic path from s to t in G.*

The Delaunay graph (DG) is an MSNet (Kurup, 1992). For a graph with n nodes, the number of edges in the DG rapidly approaches $O(n^2)$ as the dimensionality grows, limiting its usability for large datasets with high-dimensional vectors (the memory and computational complexities approach $O(n^2)$ and $O(n^{\lceil d/2 \rceil})$ (McMullen, 1970), respectively). As a consequence, many graph construction algorithms (e.g., Dearholt et al., 1988; Arya & Mount, 1993; Malkov & Yashunin, 2020; Fu et al., 2019; 2022; Subramanya et al., 2019) have been proposed over the years, operating under the (sometimes implicit) principle of sparsifying the DG. These algorithms work as depicted in Algorithm 2. For each node i, a candidate pool is selected (this is commonly implemented as an approximate nearest neighbor search), and then a pruning algorithm is used to select a maximally diverse set of nodes (i.e., far away from each other) while being close to i, see Figure 1. In essence, these algorithms were carefully designed to analyze the DG triangles (or a superset (Subramanya et al., 2019)) and discard those edges that are redundant for navigability. While these graphs were designed to have formal navigability guarantees when the candidate set $\mathcal{C}_i = [1 \dots n] \setminus \{i\}$ in Algorithm 2, they are lost when $\mathcal{C}_i \subset [1 \dots n] \setminus \{i\}$.

The DG and the main graph indices (e.g., Malkov & Yashunin, 2020; Fu et al., 2019; Subramanya et al., 2019) rely on principles from computational geometry, such as triangular inequalities and (as we show in Section 4) on the law of cosines, which are only valid in Euclidean space. These graph indices have been used in non-Euclidean vector spaces by extending their edge pruning rules to other similarities in an ad hoc fashion, resulting in a lack of understanding of their practical behavior.

Algorithm 2: Graph index construction

Input : Dataset $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$.
Output: Graph $G = ([1 \dots n], \mathcal{E})$.

$\mathcal{E} \leftarrow \emptyset$;
for $i \in [1 \dots n]$ **do**
  Selection: choose a candidate pool $\mathcal{C}_i \subseteq [1 \dots n] \setminus \{i\}$;
  Pruning: create set $\mathcal{N}_i \subseteq \mathcal{C}_i$ containing the out neighbors of node i by applying a pruning algorithm;
  $\mathcal{E} \leftarrow \mathcal{E} \cup \left\{\overrightarrow{ij} \mid j \in \mathcal{N}_i\right\}$;

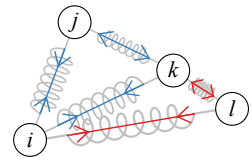


Figure 1: Conceptual depiction of the pruning strategy in Euclidean space to find the out-edges of node i in the graph index $G = ([1 \dots n], \mathcal{E})$. Attractive (inward arrowheads) and repulsive (outward arrowheads) forces promote similarity with i or diversity between candidates, respectively. Blue and red arrows depict favorable and less favorable forces, respectively. Here, $\{\overrightarrow{ij}, \overrightarrow{ik}\} \subset \mathcal{E}$ but $\overrightarrow{il} \notin \mathcal{E}$ as one can move from i to k and then from k to l using the greedy search in Algorithm 1.

2.1 From graph search to multiclass classification

We now adopt an alternative viewpoint that will ultimately lead to new developments. For this, we think of a greedy search in DG, the original monotonic search network, using Algorithm 1 as finding an ascending path in a multiclass classification problem.

The Voronoi diagram is a tessellation of the space, where each node i of the DG corresponds to a distinct convex cell C_i (see Figure 14 in the appendix). Two nodes are connected in the DG if the corresponding Voronoi cells share a facet. We associate with each cell C_i a decision function $f_i : \mathbb{R}^d \to \mathbb{R}$ such that: $\mathbf{x} \in C_i$ if $f_i(\mathbf{x}) \geq 0$, $\mathbf{x} \notin C_i$ if $f_i(\mathbf{x}) < 0$, and $f_i(\mathbf{x})$ decreases as the distance between $\mathbf{x}$ and C_i, $\min_{\mathbf{x}' \in C_i} \|\mathbf{x} - \mathbf{x}'\|_2$, increases. Each f_i is determined by the intersection of the half-spaces of the boundaries of C_i. Finding the nearest neighbor of a query $\mathbf{q}$ is equivalent to finding i such that $f_i(\mathbf{q}) \geq 0$. This corresponds to a multiclass classification problem with $n = |\mathcal{X}|$.[1] Of course, this is not computationally very useful, as we are evaluating n classifiers (i.e., scanning the entire set $\mathcal{X}$). Seeking fewer evaluations, we conceptualize Algorithm 1 as follows: if $f_{i^*}(\mathbf{q}) \geq 0$, the vector $\mathbf{x}_{i^*}$ is the nearest neighbor of $\mathbf{q}$; if $f_{i^*}(\mathbf{q}) < 0$, move to the adjacent cell i so that $f_i(\mathbf{q})$ is maximum. That is, instead of directly solving the multiclass classification problem, we use the decision functions of adjacent cells (i.e., given by the Delaunay edges) to find an ascending path $[v_1, \cdots, v_l]$ such that $f_{v_i} < f_{v_{i+1}}$. Through this ascent algorithm, we only evaluate a small subset of the n classifiers.

This qualitative viewpoint raises several questions. Can we use machine learning (ML) to build graph indices? And in non-Euclidean spaces? Can we leverage ML to build parsimonious graphs? Is there a connection between the ML approach and "traditional" graph indices? In the remainder of this paper, we answer these questions in the affirmative support vector machines, to develop new graph indices in Euclidean and non-Euclidean spaces, with the properties and contributions discussed in the introduction.

3 The Support Vector Graph

We now define a new type of graph inspired by the ideas in Section 2. Instead of relying on principles from computational geometry, we directly leverage the result of an optimization algorithm, the nonnegative least squares problem in kernel space. The positive semidefinite kernel matrix $\mathbf{K}$ with entries $\mathbf{K}_{[ij]} = K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ can be written as $\mathbf{K} = \mathbf{\Phi}^\top \mathbf{\Phi}$ where $\mathbf{\Phi} = [\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_n)]$, with the vectors horizontally stacked. From now on, and unless otherwise specified, the similarity between two vectors $\mathbf{x}_i$ and $\mathbf{x}_j$ will only be determined by the value of $K(\mathbf{x}_i, \mathbf{x}_j)$. With these elements, we present the proposed graph index.

Definition 3. *We define the Support Vector Graph (SVG) as the result of connecting node i to the non-zero elements of the minimizer $\mathbf{s}^{(i)}$ of*

$$\min_{\mathbf{s}} \frac{1}{2} \|\phi(\mathbf{x}_i) - \mathbf{\Phi}\mathbf{s}\|_2^2 \quad s.t. \quad \mathbf{s} \geq \mathbf{0},\ s_i = 0,\ \|\mathbf{s}\|_2^2 \leq n^{-1}, \tag{4}$$

[1] Inverted indices use a similar computational motif derived from Voronoi diagrams (Jégou et al., 2011).

where $\mathbf{\Phi} = [\phi(\mathbf{x}_1), \cdots, \phi(\mathbf{x}_n)]$ *are the stacked feature vectors of a PSD kernel* K*.*

In essence, Problem (4) can be seen as finding the projection of $\phi(\mathbf{x}_i)$ onto a convex set,

$$\underset{\mathbf{y}}{\operatorname{argmin}} \|\phi(\mathbf{x}_i) - \mathbf{y}\|_2^2 \quad \text{s.t.} \quad \mathbf{y} \in \operatorname{sc}(\mathcal{X}, i), \tag{5}$$

where

$$\operatorname{sc}(\mathcal{X}, i) \stackrel{\text{def}}{=} \left\{ \sum_{j=1}^{n} s_j \phi(\mathbf{x}_j) \,|\, s_j \geq 0, s_i = 0, \sum_{j=1}^{n} s_j^2 \leq n^{-1} \right\}. \tag{6}$$

The set $\operatorname{sc}(\mathcal{X}, i)$ is the conical combination of the vectors in $\mathcal{X} \setminus \{\phi(\mathbf{x}_i)\}$ with an additional constraint on the squares of the weights s_j. This set is convex, making Problem (4) convex. Interestingly, the minimizer of Problem (5) is just a scaled version of the projection of $\phi(\mathbf{x}_i)$ onto the conical combination of the vectors in $\mathcal{X} \setminus \{\phi(\mathbf{x}_i)\}$. Thus, it seems that the constraint $\|\mathbf{s}\|_2^2 \leq n^{-1}$ is redundant, as the non-zero elements of both minimizers coincide. However, this constraint is needed to establish the navigability of the graph in the next section. Moreover, we found that having a "budget restriction" of the form $\|\mathbf{s}\|_2^2 \leq n^{-1}$ helps in the numerical solution to allocate the non-zeros more judiciously by enforcing exact sparsity more accurately, with fewer infinitesimal entries.

When $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)' \geq 0$ for any i, j (a common choice), the angle between any pair of vectors is less than $\pi/2$ and we can find a rotation such that all the feature vectors lie in the positive orthant. Then, the columns of $\mathbf{\Phi}$ satisfy the conditions that ensure a unique solution (e.g., Wang & Tang, 2009; Slawski & Hein, 2011; 2014). Moreover, in this setting, Problem (4) is self-regularizing, in the sense that its minimizer is naturally sparse (Slawski & Hein, 2011) without including any explicit constraints promoting sparsity (in contrast to the DG whose sparsity depends on the input dimensionality, i.e., less sparse at higher dimensions).

The use of sparsity-regularized regression problems to build graphs is not new in machine learning. Some notable applications include the estimation of sparse inverse covariance matrices (Meinshausen & Bühlmann, 2006), subspace learning and clustering (Cheng et al., 2010; Hosseini & Hammer, 2018), spectral clustering (Xiao et al., 2012), and nonnegative matrix factorization (for bipartite graphs) (Kumar et al., 2013). In particular, a variant of Problem (4), which relies on the selection of a candidate pool as in Algorithm 2, was used for manifold learning (Shekkizhar & Ortega, 2023). However, our analysis of graphs built with nonnegative sparse regression for vector search is new.

Furthermore, the SVG establishes an interesting link between graph indices and parsimonious vector coding. That is, with a linear kernel, the loss in Problem (4) becomes $\frac{1}{2}\|\mathbf{x}_i - \mathbf{X}\mathbf{s}\|_2^2$, where $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]$. This formulation is commonly used to represent (e.g., Elhamifar et al., 2012) and quantize vectors (in additive quantization (Martinez et al., 2016), for example). There is a conceptual parallelism with inverted indices (Jégou et al., 2011), which are derived from vector quantizers (k-means).

As we show next, the connection between Problem (4) and navigable graphs starts emerging as we dig deeper into the problem's properties. By analyzing the expanded form of Problem (4),

$$\min_{\{s_j\}_{j=1}^n} \frac{1}{2}K(\mathbf{x}_i, \mathbf{x}_i) + \underbrace{\frac{1}{2}\sum_{j,k\neq i} s_j s_k K(\mathbf{x}_j, \mathbf{x}_k)}_{\text{term A}} \underbrace{- \sum_{j\neq i} s_j K(\mathbf{x}_i, \mathbf{x}_j)}_{\text{term B}} \quad \text{s.t.} \quad (\forall j)\, s_j \geq 0,\ s_i = 0, \|\mathbf{s}\|_2^2 \leq n^{-1}, \tag{7}$$

it becomes clear that its solution balances diversity (repulsion) and similarity (attraction) forces using similar principles as those shown in Figure 1 and analyzed in detail in Section 4 for other popular graph indices. The minimization of term A promotes the selection of a diverse set of edges, i.e., indices j, k such that $K(\mathbf{x}_j, \mathbf{x}_k)$ is small. When using the RBF kernel, it favors out-neighbors that are far away from each other. The minimization of term B promotes the selection of edges that are similar to $\mathbf{x}_i$, i.e., indices j such that $K(\mathbf{x}_i, \mathbf{x}_j)$ is large. When using the RBF kernel, it favors out-neighbors that are close to $\mathbf{x}_i$.

In the previous section, we qualitatively connected graph indices with a multi-class classification problem. It turns out that Problem (4) is a classification problem disguised as a regression problem.

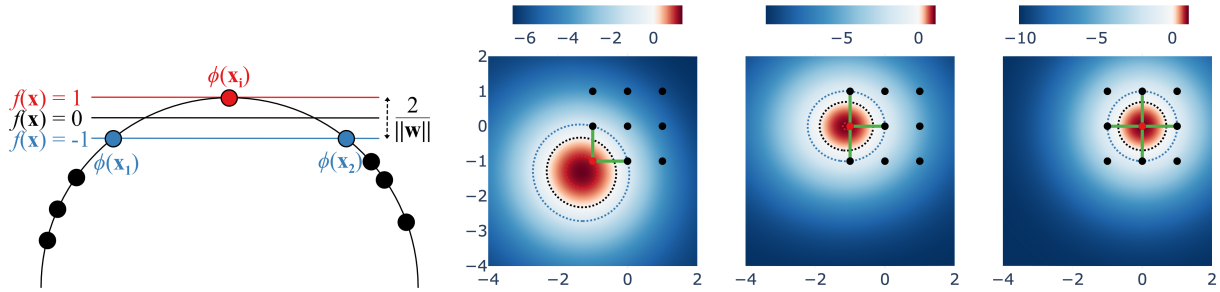


Figure 2: (Left) Conceptual representation of the SVM hyperplane and margins involved in SVG. Here, the vector $\mathbf{x}_i$ is connected to its support vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ for which $f_i(\mathbf{x}_1) = f_i(\mathbf{x}_2) = -1$, see Equation (9). (Right) Example of the SVM decision function values (the level sets $f_i(\mathbf{x}) = 1, 0, -1$ are marked in dotted red, black and blue lines, respectively). We observe that the function f_i adjusts its shape to the topology of its surrounding points (i.e., the area where $f_i(\mathbf{x}) > 0$ adapts to its surroundings).

Theorem 1. *Problem (4) is equivalent to a hard-margin support vector machine classifier using the labels*

$$y_j = \begin{cases} 1 & \text{if } j = i, \\ -1 & \text{otherwise.} \end{cases} \tag{8}$$

The nonzero elements of the minimizer $\mathbf{s}^{(i)}$ *of Problem (4) are the support vectors.*

We refer the reader to Appendix A for a quick primer on SVMs. For the i-th vector, after computing the minimizer $\mathbf{s}^*$ to Problem (4) we obtain the SVM decision function

$$f_i(\mathbf{x}) = \frac{{\mathbf{w}_i}^\top \phi(\mathbf{x}) + b_i}{{\mathbf{w}_i}^\top \phi(\mathbf{x}_i) + b_i} \quad \text{where} \quad \mathbf{w}_i = \phi(\mathbf{x}_i) - \sum_{j \neq i} s_j^* \phi(\mathbf{x}_j), \quad b_i = -\frac{1}{2} {\mathbf{w}_i}^\top \left(\phi(\mathbf{x}_i) + \phi(\mathbf{x}_{j'})\right), \tag{9}$$

for any j' such that $s_{j'}^* > 0$. By construction, $f_i(\mathbf{x}_i) = 1$ and $f_i(\mathbf{x}_j) = -1$ for every $j \in \mathcal{N}_i$, and $f_i(\mathbf{x}_j) < -1$ for every $j \in [1..n] \setminus (\mathcal{N}_i \cup \{i\})$. In Figure 2 (left), we present a conceptual representation of these level sets as hyperplanes in feature space. Figure 2 (right) illustrates that these level sets materialize in the original space as nonlinear boundaries that adapt their shape to the topology of the vectors surrounding $\mathbf{x}_i$.

This alternative formulation of Problem (4) makes it easy to see why $|\mathcal{N}_i| \ll n$: The support vector set $\mathcal{N}_i$ is sparse in separable and non-degenerate settings. Although we leave formal sparsity results for future work, we illustrate this point with an important example. For D-dimensional feature vectors, the number of support vectors in SVMs is at most $D + 1$. In MIP (maximum inner product) retrieval, possibly the most common vector search today, the kernel is linear $K(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$. In this case, each node in the SVG will have $d + 1$ out-edges at most for $\mathbf{x} \in \mathbb{R}^d$. The dimensionalities used today ($d = 1024$ to $d = 4096$) imply a sparse graph for $n \gg d$.

The decision functions f_i induce a tessellation of the space, as observed in Figure 3. We can find this tessellation by considering the function $F(\mathbf{x}) = -\max_i f_i(\mathbf{x})$ as a topographic map and separating adjacent catchment basins (following its gradient) using a watershed algorithm (Couprie & Bertrand, 1997). The link between the SVG and this tessellation is analogous to that of the Delaunay graph and the Voronoi diagram.

SVG also shares a deep connection with the DG. Other graph indices are subgraphs of the DG by applying pruning rules to its edges. As shown next, when using a kernel based on the Euclidean distance (e.g., the RBF kernel), SVG sparsifies the DG by solving optimization problems (see Figures 4 and 5).

Theorem 2. *Let G be the Delaunay graph computed from the original vectors $\{\mathbf{x}_i\}_{i=1}^n$. When using a kernel based on the Euclidean distance (e.g., RBF), the support of the solution to Problem (4) is a subset of the neighbors of node i in G.*

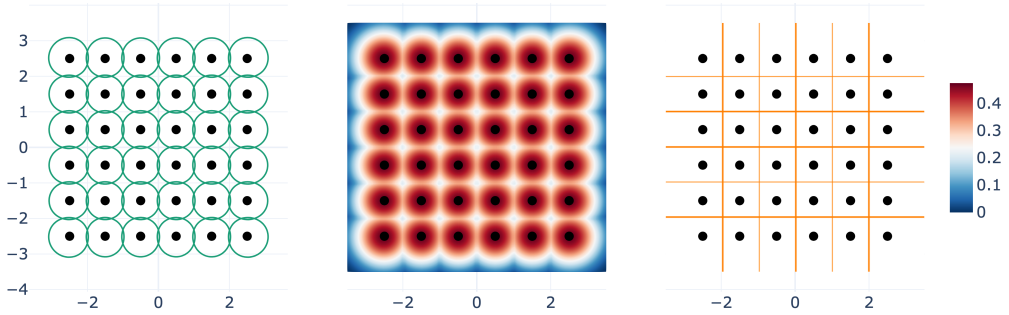


Figure 3: The SVM decision boundaries (left), i.e., $f_i(\mathbf{x}) = 0$, for each point in a regular 2D grid; see Equation (9). The function $f(\mathbf{x}) = \max_i f_i(\mathbf{x})$ (center) induces a tessellation. As expected, the tessellation, found by running a watershed algorithm on $f(\mathbf{x})$, forms a regular grid (right).

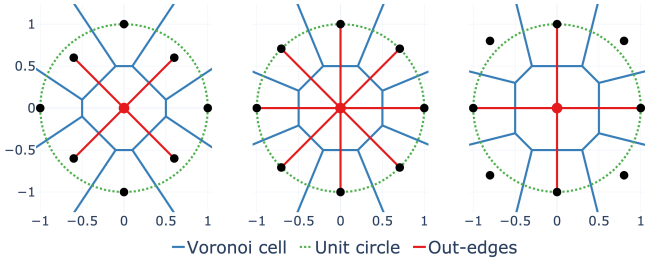


Figure 4: With the RBF kernel, Problem (4) connects the central node i to a subset of its Delaunay neighbors $\mathcal{D}_i$. For most configurations, the SVG neighbors $\mathcal{N}_i \subset \mathcal{D}_i$ (left/right plots). However, $\mathcal{N}_i = \mathcal{D}_i$ in odd configurations, e.g., when the vectors in $\mathcal{D}_i$ are equi-spread and equidistant to i (center).

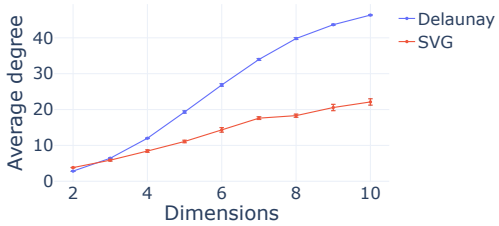


Figure 5: The average cardinality of the SVG neighbors grows slower than that of the Delaunay neighbors for (ten realizations of) 100 randomly distributed vectors as their dimension grows.

To conclude this section, we briefly analyze the setting where we have an indefinite kernel. These kernels do not induce a Reproducing Kernel Hilbert Space (RKHS) and consequently do not have features $\phi(\mathbf{x})$. We could still derive a graph from such kernels by starting from Problem (7). Multiple differences arise in this scenario. First, the interpretation of Problem (7) as a NNLS and/or a SVM classifier are lost. Much of the geometry insights that we gain from such perspectives are foregone too. Second, Problem (7) becomes non-convex and there might be multiple edge configurations that are valid solutions. Lastly, but perhaps more interestingly, the graph remains navigable as the results in the next section are still valid in non-RKHS settings.

3.1 Navigability

So far, we have described how to build an SVG and how it shares some key properties with the DG and other graph indices. We now turn our attention to the analysis of its navigability, showing that SVGs are navigable for general kernels (all proofs are in the appendix).

There is abundant literature (e.g., Dearholt et al., 1988; Arya & Mount, 1993; Malkov & Yashunin, 2020; Fu et al., 2019; 2022; Subramanya et al., 2019) on graph indices that were designed to have formal navigability guarantees only in the Euclidean case, with this guarantee forgone when using other similarities. Before proceeding, we provide new definitions that generalize the notion of graph navigation for arbitrary similarities. With non-metric similarities, the terminal node may be different from the query. That is, whereas we always have $\text{sim}_{\text{EUC}}(\mathbf{x}_j, \mathbf{x}_j) = \max_{i=1,\dots n} \text{sim}_{\text{EUC}}(\mathbf{x}_i, \mathbf{x}_j)$ for $j \in [1 \dots n]$, it is possible that $\text{sim}_{\text{DP}}(\mathbf{x}_j, \mathbf{x}_j) < \max_{i=1,\dots n} \text{sim}_{\text{DP}}(\mathbf{x}_i, \mathbf{x}_j)$ (see Equations (2) and (3)).

Definition 4 (Generalized Monotonic Path)**.** *Given a set of n vectors $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ and the similarity function* sim*, let $G = ([1 \dots n], \mathcal{E})$ denote a directed graph, $s, k \in [1 \dots n]$ be two nodes of G, and $t =$*

$\operatorname*{argmax}_{i=1,\dots n} \operatorname{sim}(\mathbf{x}_i, \mathbf{x}_k)$. *A path* $[v_1, \cdots, v_l]$ *from* $s = v_1$ *to* $t = v_l$ *in* G *is a generalized monotonic path if and only if* $(\forall i = 1, \cdots, l-1)$ $\operatorname{sim}(\mathbf{x}_{v_i}, \mathbf{x}_t) < \operatorname{sim}(\mathbf{x}_{v_{i+1}}, \mathbf{x}_t)$.

With this change, the generalization of a navigable graph follows naturally.

Definition 5 (Generalized Monotonic Search Network)**.** *Given a set of* n *vectors* $\left\{\mathbf{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ *and the similarity function* sim*, a graph* $G = ([1 \dots n], \mathcal{E})$ *is a generalized monotonic search network if and only if there exists at least one generalized monotonic path from* s *to* $t = \operatorname*{argmax}_{i=1,\cdots,n} \operatorname{sim}(\mathbf{x}_i, \mathbf{x}_k)$ *for any two nodes* $s, k \in [1 \dots n]$.

Definition 5 is equivalent to Definition 2 when using a similarity based on the Euclidean distance as $t = k$. Next, we show that a generalized monotonic path can be successfully traversed with Algorithm 1.

Lemma 2. *Let* $G = ([1 \dots n], \mathcal{E})$ *be a generalized monotonic search network. Let* $s, k \in [1 \dots n]$*, then Algorithm 1 with* $\mathbf{x}_k$ *as the query and* s *as the entry point finds a generalized monotonic path from* s *to* $t = \operatorname*{argmax}_{i=1,\cdots,n} \operatorname{sim}(\mathbf{x}_i, \mathbf{x}_k)$ *in* G.

The upcoming results about SVG navigability do not require the use of a PSD kernel, as they rely on KKT conditions of Equation (7). As such, they are valid even when using indefinite kernels that do not induce a RKHS.

As a partial step towards a formal navigability result, using the optimality conditions of Problem (4), we first show in Lemma 3 that (1) there is always an edge in the SVG that improves the current similarity, or (2) we have reached our objective. From this lemma, the following result follows immediately.

Lemma 3. *Let* k *be a query node in an SVG and* $t = \operatorname*{argmax}_{j=1,\cdots,n} K(\mathbf{x}_j, \mathbf{x}_k)$ *be the target node. For* $i \neq t$*,* $\exists j \in \mathcal{N}_i$ *such that* $K(\mathbf{x}_i, \mathbf{x}_t) \leq K(\mathbf{x}_j, \mathbf{x}_t)$.

Analyzing the inequality in Lemma 3 in terms of the feature vectors can help understand its geometric meaning: $K(\mathbf{x}_i, \mathbf{x}_t) \leq K(\mathbf{x}_j, \mathbf{x}_t)$ implies that $\phi(\mathbf{x}_t)^\top (\phi(\mathbf{x}_j) - \phi(\mathbf{x}_i)) > 0$, where the vector $\phi(\mathbf{x}_j) - \phi(\mathbf{x}_i)$ can be interpreted as a "discrete gradient" that needs to be positively correlated with the target vector $\phi(\mathbf{x}_t)$.

Equipped with these new concepts, we provide our main theoretical result, the navigability of the SVG.

Theorem 3. *An SVG is a generalized monotonic search network (Definition 5).*

The SVG, derived using completely different tools (i.e., kernel methods from machine learning) than existing graph indices, is a suitable graph index for vector search. To the best of our knowledge, this is the first graph index guaranteeing navigability in non-Euclidean and even non-metric spaces.

3.2 Computing the SVG

Writing Problem (7) in vectorial form, we obtain an instance of the problem

$$\min_{\mathbf{s}} \frac{1}{2}\mathbf{s}^\top \mathbf{A}\mathbf{s} - \mathbf{b}^\top \mathbf{s} \quad \text{s.t.} \quad \mathbf{s} \geq 0, \|\mathbf{s}\|_2^2 \leq n^{-1}, \tag{10}$$

where the PSD matrix $\mathbf{A}$ is result of removing the i-th row and the i-th column from the matrix $\mathbf{K}$ and $\mathbf{b}$ is result of removing the i-th element from $\mathbf{K}_{[:i]}$, the i-th column of $\mathbf{K}$. Sha et al. (2007) proposed a solver for the following related problem

$$\min_{\mathbf{s}} \frac{1}{2}\mathbf{s}^\top \mathbf{A}\mathbf{s} - \mathbf{b}^\top \mathbf{s} \quad \text{s.t.} \quad \mathbf{s} \geq 0, \tag{11}$$

based on multiplicative updates. These updates take the form

$$\mathbf{s} \leftarrow \mathbf{s} \odot \boldsymbol{\gamma} \quad \text{where} \quad \boldsymbol{\gamma} = \left(\mathbf{b} + \sqrt{\mathbf{b}^2 + 4(\mathbf{A}^+\mathbf{s}) \odot (\mathbf{A}^-\mathbf{s})}\right) \oslash 2(\mathbf{A}^+\mathbf{s})_j \tag{12}$$

where $\odot$ and $\oslash$ denote the Hadamard product and division, respectively, the square and square root operations are applied element-wise, $\mathbf{A}^+ = \max\{\mathbf{A}, \mathbf{0}\}$, and $\mathbf{A}^- = |\min\{\mathbf{A}, \mathbf{0}\}|$. The update in Equation (12) has

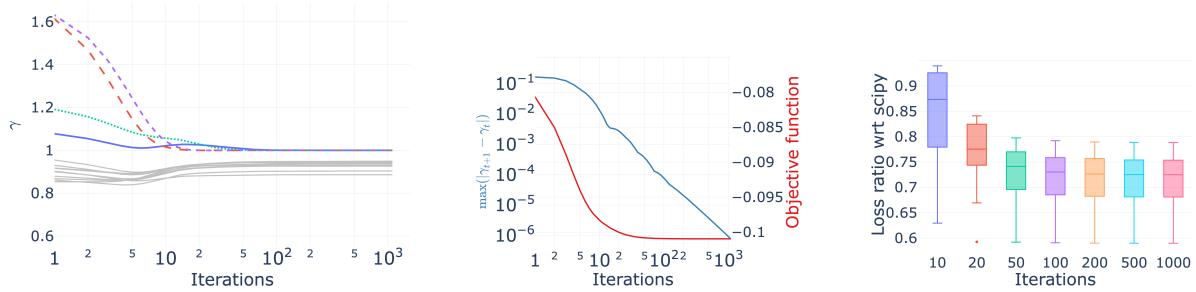


Figure 6: The multiplicative updates in Equation (14) is a competent solver with simple and hardware-friendly implementations. In this example, we use an RBF kernel and the update in Equation (13) to show the behavior of the algorithm as the iterations progress. **Left:** we show the trajectories of several γ_j, the ones that converge to one (corresponding to the nonzero entries of $\mathbf{s}$) in color/dashed lines, the ones that do not in gray lines. There is a clear separation of these two groups. **Center:** we show the evolution of the loss (in red) and the change in the entries γ_j. **Right:** we show the ratio between the loss obtained with the SLSQP solver in scipy: a ratio lower than one indicates a lower loss attained by the multiplicative algorithm.

fixed points wherever the objective function $F(\mathbf{s}) = \frac{1}{2}\mathbf{s}^\top \mathbf{A}\mathbf{s} - \mathbf{b}^\top \mathbf{s}$ attains its minimum value. Let $\mathbf{s}^*$ denote the global minimum of Problem (10). At this minimum, either $s_j^* = 0$ or $s_j^* > 0$ and $(\partial F/\partial s_j)(s_j^*) = 0$. Fixed points of these multiplicative updates occur when either $s_j^* = 0$ or $s_j^* > 0$ and $\gamma_j = 1$. Since $(\partial F/\partial s_j)(s_j^*) = 0$ implies $\gamma_j = 1$, the equivalence holds (Sha et al., 2007).

When working with nonnegative kernels (e.g., any exponential kernel), the matrix $\mathbf{A}$ and the vector $\mathbf{b}$ are nonnegative and the update vector $\boldsymbol{\gamma}$ simplifies to

$$\boldsymbol{\gamma} = \mathbf{b} \oslash \mathbf{A}\mathbf{s}. \tag{13}$$

Because Problem (10) is a convex problem, we can safely use the algorithm above in conjunction with a projection onto $\|\mathbf{s}\|_2^2 \leq n^{-1}$, i.e., the hypersphere of radius $\sqrt{n}$. This yields the update

$$\mathbf{s} \leftarrow (\mathbf{s} \odot \boldsymbol{\gamma}) \oslash \max\left\{\sqrt{n}\,\|\mathbf{s} \odot \boldsymbol{\gamma}\|_2, 1\right\}, \tag{14}$$

where $\boldsymbol{\gamma}$ is defined in Equations (12) and (13), depending on the nonnegativity of the kernel. Each iteration is dominated by the cost of the matrix-vector multiplication at a cost of $O(n^2)$. It is important to note that the problem constraints are satisfied exactly throughout the iterations.

We found that this algorithm is particularly fast compared to standard solvers for convex problems. Additionally, the multiplicative updates rely solely on matrix-vector multiplications and standard vector operations, which can be implemented with great efficiency on modern hardware (on GPUs or x86 CPUs using AVX instructions). Interestingly, we can exploit the equivalence between $s_j^* > 0$ and $\gamma_j = 1$ to track and select the support of $\mathbf{s}^*$ using $\boldsymbol{\gamma}$.

4 Connecting SVG to other graph indices

We now study the link between SVG and other popular graph indices. Algorithm 3 provides a blueprint for most pruning techniques (Malkov & Yashunin, 2020; Subramanya et al., 2019; Fu et al., 2022) used in Algorithm 2. Given a candidate pool $\mathcal{C}_i$, Algorithm 3 considers triplets of nodes i, j, k, as depicted in Figure 7. Throughout this section, we use the standard assumption $\mathcal{C}_i = [1 \ldots n] \setminus \{i\}$ (we discuss this choice in the next section).

Next, we show that Problem (4), when applied to the analysis of these triplets, leads to a traditional graph sparsification algorithm with navigability guarantees for general kernels. Given a PSD kernel K, for each triplet i, j, k encountered in Line 6 of Algorithm 3, we solve Problem (4) using just $\phi(\mathbf{x}_i)$, $\phi(\mathbf{x}_j)$, and $\phi(\mathbf{x}_k)$ and connect i to j and/or k if $s_i > 0$ and/or $s_k > 0$.

Algorithm 3: Pruning meta-algorithm to determine the outgoing edges for node i

Input : Dataset $\mathcal{X} = \left\{\mathbf{x}_j \in \mathbb{R}^d\right\}_{j=1}^n$, node $i \in [1 \dots n]$, candidate pool $\mathcal{C}_i$, maximum out-degree $M \in \mathbb{N}^+$.

Output: Set $\mathcal{N}_i$ of outgoing neighbors for node i

1 $\mathcal{E}_i \leftarrow \emptyset$;
2 **while** $\mathcal{C}_i \neq \emptyset$ **do**
3 $\quad j \leftarrow \underset{j' \in \mathcal{C}}{\operatorname{argmax}}\, K(\mathbf{x}_i, \mathbf{x}_{j'})$;
4 $\quad \mathcal{N}_i \leftarrow \mathcal{E}_i \cup \{j\}$;
5 $\quad \mathcal{C}_i \leftarrow \mathcal{C} \setminus \{j\}$;
6 $\quad \mathcal{C}_i \leftarrow \{k \in \mathcal{C}_i \,|\, \text{connectivity rule between } i,\, j, \text{ and } k \text{ is met}\}$;

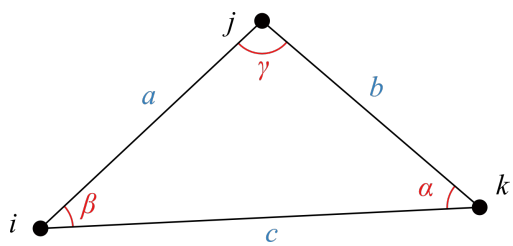


Figure 7: We show that, when using an RBF kernel, the graph pruning rules of most popular graph construction algorithms, including the popular HNSW (Malkov & Yashunin, 2020) and DiskANN (Subramanya et al., 2019), can be written as applications of the law of cosines and the inequality $a^2 + b^2 > c^2$.

We say that a kernel is normalized if $(\forall \mathbf{x})\, K(\mathbf{x}, \mathbf{x}) = 1$. Using Proposition 1 in the appendix, we derive the following equivalence for normalized kernels, which leads to a monotonicity certificate.

Lemma 4 (Kernel connectivity rule)**.** *Let K be a normalized PSD kernel. Solving Problem (4) for each triplet i, j, k encountered in Line 6 of Algorithm 3 is equivalent to $k \in \mathcal{N}_i$ if and only if*

$$K(\mathbf{x}_i, \mathbf{x}_j) K(\mathbf{x}_j, \mathbf{x}_k) < K(\mathbf{x}_i, \mathbf{x}_k). \tag{15}$$

Theorem 4. *Let s, k be two distinct nodes of a graph built using Algorithm 3 with $\mathcal{C}_i = [1 \dots n] \setminus \{i\}$ and the kernel connectivity rule in Lemma 4 and let $t = \underset{i \in V}{\operatorname{argmax}}\, K(\mathbf{x}_i, \mathbf{x}_k)$. There is a generalized monotonic path between s and t.*

The following corollary follows immediately from Definition 2 and Theorem 4.

Corollary 1. *A graph built using Algorithm 3 with $(\forall i)\, \mathcal{C}_i = [1 \dots n]$ and the connectivity rule in Lemma 4 is a generalized monotonic search network.*

This is the second graph construction algorithm with full navigability guarantees regardless of the similarity function underlying the kernel (SVG is the first). In this sense, it applies to any metric space and even non-metric vector spaces, e.g., spaces only equipped with an inner product.

For the RBF kernel, the connectivity rule in Lemma 4 amounts to the familiar inequality $a^2 + b^2 > c^2$, as depicted in Figure 7 and shown next.

Corollary 2. *When using the RBF kernel in the same setting as Proposition 1, we have*

$$K(\mathbf{x}_i, \mathbf{x}_j) K(\mathbf{x}_j, \mathbf{x}_k) < K(\mathbf{x}_i, \mathbf{x}_k) \quad \Leftrightarrow \quad \|\mathbf{x}_i - \mathbf{x}_j\|_2^2 + \|\mathbf{x}_j - \mathbf{x}_k\|_2^2 > \|\mathbf{x}_i - \mathbf{x}_k\|_2^2 . \tag{16}$$

From Corollary 2, we can derive many graph construction rules by combining $a^2 + b^2 > c^2$ with different formulas from the law of cosines. Corollaries 3 and 4 provide two specific examples (in the following, we connect them with existing graph indices).

Corollary 3 (Shekkizhar & Ortega, 2023)**.** *When using an RBF kernel in the same setting as Lemma 4, the necessary condition to connect node i with node k is*

$$(\cos\alpha)\,\|\mathbf{x}_i-\mathbf{x}_k\|_2 < \|\mathbf{x}_j-\mathbf{x}_k\|_2\,, \tag{17}$$

where α is the angle between the vectors $\mathbf{x}_i-\mathbf{x}_k$ and $\mathbf{x}_j-\mathbf{x}_k$.

Corollary 4. *When using an RBF kernel in the same setting as Lemma 4, the necessary condition to connect node i with node k is*

$$(\cos\beta)\,\|\mathbf{x}_i-\mathbf{x}_k\|_2 < \|\mathbf{x}_i-\mathbf{x}_j\|_2\,, \tag{18}$$

where β is the angle between the vectors $\mathbf{x}_k-\mathbf{x}_i$ and $\mathbf{x}_j-\mathbf{x}_i$.

4.1 Graph indices under the lens

In the following, we show that the most popular graph indices can be interpreted as thresholded applications of different formulas from the law of cosines combined with $a^2+b^2>c^2$ (see Figure 7). In particular, our results cover the popular HNSW (Malkov & Yashunin, 2020) and DiskANN (Subramanya et al., 2019).

We start with the connectivity rule shared by MRNG (Fu et al., 2019) and HNSW (Malkov & Yashunin, 2020). For $i,j\in[1\dots n]$, let

$$lune_{ij}=\left\{\mathbf{x}\in\mathbb{R}^d \mid \|\mathbf{x}_i-\mathbf{x}\|_2\le\|\mathbf{x}_i-\mathbf{x}_j\|_2 \wedge \|\mathbf{x}_j-\mathbf{x}\|_2\le\|\mathbf{x}_i-\mathbf{x}_j\|_2\right\} \tag{19}$$

An MRNG is a directed graph $G=(\mathcal{V},\mathcal{E})$ with $\mathcal{V}=[1\dots n]$ and the edge set $\mathcal{E}$ satisfying the following property: For any pair $i,k\in[1\dots n]$, $\overrightarrow{ik}\in\mathcal{E}$ if and only if $lune_{ik}\cap S=\emptyset$ or $(\forall r)\,\mathbf{x}_r\in lune_{ik}\cap S\Longrightarrow\overrightarrow{ir}\notin\mathcal{E}$. MRNG and HNSW use the following connectivity rule in Algorithm 3: Keep k in $\mathcal{C}_i$ if

$$\|\mathbf{x}_i-\mathbf{x}_k\|_2\le\|\mathbf{x}_j-\mathbf{x}_k\|_2\,. \tag{20}$$

HNSW applies this rule within a hierarchical structure. A direct application of Corollary 3 yields the following result.

Corollary 5. *Running Algorithm 3 with the RBF kernel and the MRNG connectivity rule in Equation (20) is equivalent to applying the necessary condition in Corollary 3 with the additional simplification that $\cos\alpha=1$.*

Vamana, the algorithm behind DiskANN (Subramanya et al., 2019), is an extension of MRNG (Fu et al., 2019) that seeks to accelerate the graph traversal speed. Recently, this acceleration has been formally proven in the worst case (Indyk & Xu, 2023). Vamana uses the following connectivity rule in Algorithm 3: keep k in $\mathcal{C}_i$ if

$$\|\mathbf{x}_i-\mathbf{x}_k\|_2\le\lambda\,\|\mathbf{x}_j-\mathbf{x}_k\|_2\,. \tag{21}$$

A direct application of Corollary 3 yields the following result.

Corollary 6. *Running Algorithm 3 with the RBF kernel and the Vamana connectivity rule in Equation (21) is equivalent to applying the necessary condition in Corollary 3 with the additional simplification that $\alpha=\arccos\lambda^{-1}$.*

We also extend these results to the recently introduced SSG (Fu et al., 2022) using the following definitions

$$\mathrm{angle}(\mathbf{x}\,,\,\mathbf{y})=\arccos\frac{\langle\mathbf{x}\,,\,\mathbf{y}\rangle}{\|\mathbf{x}\|_2\,\|\mathbf{y}\|_2}, \tag{22}$$

$$\mathrm{ball}(i,\delta)=\left\{\mathbf{x}\in\mathbb{R}^d \,\middle|\, \|\mathbf{x}-\mathbf{x}_i\|_2\le\delta\right\}, \tag{23}$$

$$\mathrm{cone}^{\theta}_{ij}=\left\{\mathbf{x}\in\mathbb{R}^d \mid \mathrm{angle}(\mathbf{x}-\mathbf{x}_i\,,\,\mathbf{x}_j-\mathbf{x}_i)\le\theta\right\}. \tag{24}$$

An SSG is a directed graph $G=(\mathcal{V},\mathcal{E})$ with $\mathcal{V}=[1\dots n]$ and the edge set $\mathcal{E}$ satisfying the following property: For any pair $i,k\in[1\dots n]$, $\overrightarrow{ik}\in\mathcal{E}$ if and only if $\mathrm{cone}^{\theta}_{ik}\cap\mathrm{ball}(i,\|\mathbf{x}_i-\mathbf{x}_k\|_2)\cap S=\emptyset$ or $(\forall r,\mathbf{x}_r\in\mathrm{cone}^{\theta}_{ik}\cap\mathrm{ball}(i,\|\mathbf{x}_i-\mathbf{x}_k\|_2)\cap S)\,\overrightarrow{ir}\notin\mathcal{E}$, where $0\le\theta\le60^\circ$ is a hyperparameter. SSG uses the following connectivity rule in Algorithm 3: for $0\le\theta\le60^\circ$, keep k in $\mathcal{C}_i$ if

$$\mathrm{angle}(\mathbf{x}_j-\mathbf{x}_i\,,\,\mathbf{x}_k-\mathbf{x}_i)\ge\theta\vee\|\mathbf{x}_j-\mathbf{x}_i\|_2\ge\|\mathbf{x}_i-\mathbf{x}_k\|_2\,. \tag{25}$$

A direct application of Corollary 4 yields the following result.

Algorithm 4: Subspace pursuit for SVG construction

Input : Dataset $\mathcal{X} = \left\{\mathbf{x}_j \in \mathbb{R}^d\right\}_{j=1}^n$, element $i \in [1 \dots n]$, maximum out-degree $M \in \mathbb{N}^+$.
Output: Set $\mathcal{N}_i$ of outgoing neighbors for node i.

Select the kernel width σ used for node i;
$\mathcal{N}^{(0)} \leftarrow \emptyset$;
for $t \in [1 \dots T]$ **do**
  Let $\mathcal{C}$ be the set of the M largest entries in $\left\{K(\mathbf{x}_i, \mathbf{x}_k) - \sum_{j \in \mathcal{N}^{(t-1)}} s_j^{(t-1)} K(\mathbf{x}_j, \mathbf{x}_k) \,\middle|\, k \in [1 \dots n]\right\}$;
  $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{N}^{(t-1)}$;
  Find the solution $\mathbf{s}$ to Problem (26), restricted to vectors in $\mathcal{C}$;
  Let $\mathcal{N}^{(t)}$ be the set of indices corresponding to the M largest entries of $\mathbf{s}$;
  $\mathcal{N}^{(t)} \leftarrow \left\{j \mid s_j^{(t)} > 0\right\}$;
  if $\mathcal{N}^{(t)} = \mathcal{N}^{(t-1)}$ **then** break;
$\mathcal{N}_i \leftarrow \mathcal{N}^{(t)}$;

Corollary 7. *Running Algorithm 3 with the RBF kernel and the SSG connectivity rule in Equation (25) rule is equivalent to applying the necessary condition in Corollary 4 with the additional simplification that* $\theta = \arccos\left(\|\mathbf{x}_i - \mathbf{x}_j\|_2 / \|\mathbf{x}_i - \mathbf{x}_k\|_2\right)$.

In summary, the edge pruning rules used in Algorithm 3 by some of the most popular graph indices can be regarded as specializations of the SVG optimization. These specializations, described in Figure 7, emerge from applying the optimization to triplets of points.

5 Fast SVG construction with bounded out-degree

Graph construction algorithms based on Algorithm 3, such as those described in Section 4, have two main practical issues that require careful tuning.

First, Algorithm 3 requires a candidate pool $\mathcal{C}_i$. Setting $\mathcal{C}_i = [1 \dots n] \setminus \{i\}$ is usually used to obtain theoretical guarantees, as described in Section 4, but becomes practically untenable as n grows. In practice, $\mathcal{C}_i$ is heuristically determined by finding the (approximate) nearest neighbors of $\mathbf{x}_i$. However, this may be problematic if, for example, $\mathbf{x}_i$ lies on the outskirts of a tight cluster (Indyk & Xu, 2023) as the graph may become disconnected. Take the example of the red point in Figure 9. We would need to create a candidate pool larger than the number of points in the left cluster for Algorithm 3 to ensure navigability between both clusters.[2] Finding a prudent size for $\mathcal{C}_i$ becomes a dataset-specific tuning problem.

Second, although Algorithm 3 produces sparse graphs when paired with a suitable edge selection rule (such as those in Section 4), the graphs are generally not sparse enough. The sparsity of these graphs is critical as it directly determines its footprint and search runtime. Let M be the maximum out-degree we want in a graph. Because Algorithm 3 neither produces a total order nor handles the cardinality constraint intrinsically, the list of neighbors is truncated in an ad-hoc fashion by stopping Algorithm 3 once $|\mathcal{N}_i| = M$. This heuristic may cause navigability problems, as described in Figure 8. In essence, the diversity of the selected edges becomes suboptimal and may prevent navigation if the process is terminated early.

In this section, we show that the SVG framework overcomes these difficulties. Although the solution to Problem (4) is naturally sparse, we would like to impose a more stringent and specific level of sparsity to bound the out-degree of the graph. Moreover, we show that once this restriction is added, precomputing a candidate pool becomes unnecessary.

[2]The graph construction by Shekkizhar & Ortega (2023) that uses Problem (4) for manifold learning shares these issues.

We address both problems simultaneously by altering the SVG optimization. We bound the sparsity level by solving the related problem

$$\min_{\mathbf{s}} \frac{1}{2} \|\phi(\mathbf{x}_i) - \mathbf{\Phi}\mathbf{s}\|_2^2 \quad \text{s.t.} \quad \mathbf{s} \geq \mathbf{0},\ s_i = 0, \|\mathbf{s}\|_2^2 \leq n^{-1}, \|\mathbf{s}\|_0 \leq M, \tag{26}$$

where the so-called ℓ_0 norm measures the number of non-zero entries of a vector. We use Problem (26) to build a graph with a maximum out-degree, since the minimizer $\mathbf{s}_i^*$ will have at most M nonzero entries. In contrast with the typical truncation heuristic, Problem (26) selects the subset of M elements that provides the best tradeoff between diversity and similarity (see the analysis of Problem (7)). This approach has connections with sparse SVMs (Smola et al., 1999), which use parsimony-inducing ℓ_1 (e.g., Bi et al., 2003) or ℓ_0 (e.g., Zhang et al., 2023) constraints to sparsify the support vector set.

The astute reader will notice that solving Problems (4) and (26) quickly becomes impractical as n grows: The size of the kernel matrix $\mathbf{K}$ is quadratic in n and we solve an optimization with n variables. It seems like our computational requirements are still very high. Additionally, problems involving ℓ_0 constraints have always been considered challenging because of their non-convexity and NP-hardness. The dominant paradigm replaces these constraints by convex ℓ_1 constraints (e.g., Candès & Plan, 2009).[3] However, Problem (26) belongs to a particular family of problems, known as subspace pursuit, for which there are very efficient algorithms (Dai & Milenkovic, 2009; Needell & Tropp, 2010). Algorithm 4 presents an algorithm that solves this problem. Additional details on subspace pursuit can be found in Appendix B.

Algorithm 3 does not depend on precomputing an appropriate candidate pool, in contrast to Algorithm 2. This superpower comes from Line 4, which performs a neighbor search with a modified similarity. By finding the vectors that maximize

$$K(\mathbf{x}_i, \mathbf{x}_k) - \sum_{j \in \mathcal{N}^{(t-1)}} s_j^{(t-1)} K(\mathbf{x}_j, \mathbf{x}_k), \tag{27}$$

it becomes clear that the similarity in this neighbor search selects vectors that are close to $\mathbf{x}_i$ and far away from the vectors in $\mathcal{N}^{(t-1)}$. This search focuses its attention on portions of the space not considered in previous iterations (see Figure 9).

As a side note, by writing Equation (27) as

$$\mathbf{v}^\top \phi(\mathbf{x}_k), \quad \text{where} \quad \mathbf{v} = \phi(\mathbf{x}_i) - \sum_{j \in \mathcal{N}^{(t-1)}} s_j^{(t-1)} \phi(\mathbf{x}_j), \tag{28}$$

the computation can be carried out using random features (RF) (Rahimi & Recht, 2007; Reid et al., 2023; Liu et al., 2022; Sernau et al., 2024). However, the literature has understandably concentrated on approximating the central portion of kernels instead of their tails (e.g., for exponential kernels, where $K_{\text{EXP}}(\mathbf{x}, \mathbf{x}') \approx 0_+$). Since Equation (28) is concerned with the tails (notice the small values in the attention area in Figure 9), new RF techniques would be needed. This is an interesting future line of work.

Computational complexity. Algorithm 4 involves solving a least squares problems on the simplex in Line 6 with $2M$ variables. Thus, the complexity of the sub-problem is drastically reduced from $O(n^2)$ to $O(M^2)$ when using the multiplicative algorithm from Section 3.2. Implementing Line 4 in Algorithm 4 using a vector search index that typically offer $O(\log n)$ complexity, the complexity of Algorithm 4 becomes sublinear in the number of indexed vectors. When running Algorithm 4 for each node in the graph, the total SVG-L0 construction complexity is $O(n(M^2 + \log n))$. This complexity, as function of n, is similar to other existing graph indices (Malkov & Yashunin, 2020; Fu et al., 2019; Subramanya et al., 2019).

6 Experimental results

We present a few experimental results to highlight the practical value of SVG and SVG-L0. We use the standard recall@1 measure of search accuracy, which counts how many times we find the ground truth NN

[3]Direct ℓ_0 solvers have gained significant attention in the last few years (Bertsimas et al., 2016; Hastie et al., 2020) due to improvements in mixed integer optimization.

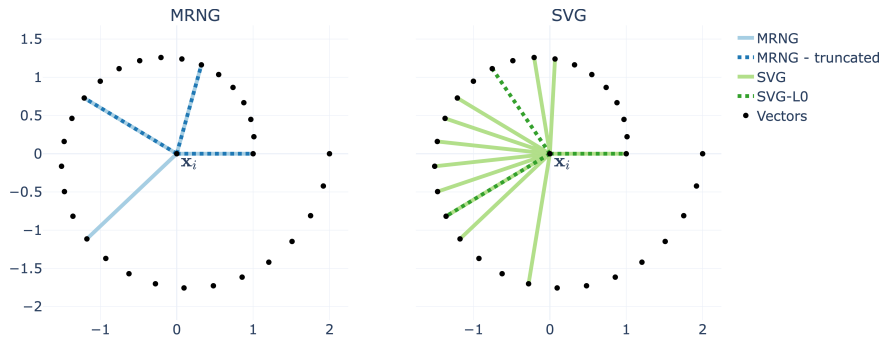


Figure 8: Although the MRNG is sparse, it is not sparse enough in practical situations and its list of neighbors is truncated by stopping Algorithm 3 early after $\mathcal{N}_i$ attains a prescribed size. With the resulting edge set (represented by dotted blue lines on the left plot), navigating downwards from $\mathbf{x}_i$ is not possible. In contrast, by using Problem (26) to build the degree-constrained SVG, we obtain an edge set (represented by dotted green lines on the right plot) with improved diversity and thus navigability. Note that the SVG-L0 edges are not necessarily a subset of the SVG edges.

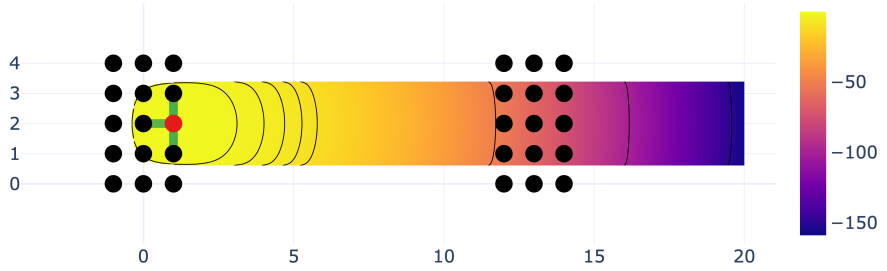


Figure 9: The attention of the search driven by Equation (27) after connecting the red point to its three neighbors (green edges). We show the values of Equation (27) in a logarithmic colorscale in the area where it is positive. The attention will focus on finding the next M points to the right of the red point, seeking to find another support vector for the SVM classifier.

of every vector $\mathbf{x}_i \in \mathcal{X}$ when using it as the query. In addition to purely greedy graph search algorithms, we also experimented with backtracking since it is widely used in practice (see Algorithm 6 in the appendix). We use a small backtracking queues of length 2, 5, and 10.

Our results in Section 3.1 predict the navigability of the SVG. This is empirically verified by the experiment in Figure 10, where the empirical navigability (recall@1) is constant and near perfect (in some cases, it is only 0.9999 due to numerical effects).

We also compared SVG-L0 (Section 5) with the degree-constrained MRNG, the truncated MRNG, and the truncated Vamana. As discussed in Section 4, the MRNG/Vamana are a fully navigable network with no degree constraints. This feature comes at a steep price: the complexity of building with MRNG/Vamana is $O(n^3)$. Here, we use a (still computationally costly) variant that uses a maximum out-degree M and an unlimited candidate pool size, resulting in a complexity of $O(n^2)$. The truncated MRNG/Vamana have the additional constraint of working with a fixed candidate pool size (see Algorithm 2). Note that the truncated MRNG/Vamana are the algorithms used in practice today to build graph indices. We set the size of the candidate pool as a multiplicative factor of M, that is, $|\mathcal{C}| = rM$ for $r > 1$. SVG-L0 uses a maximum out-degree but does not need a candidate pool.

As shown in Figure 11, SVG-L0 is competitive with the degree-constrained MRNG when working with randomly distributed vectors. Its accuracy is slightly lower in the greedy setting, but slightly higher in the backtracking setting. Both indices are clearly superior to the truncated MRNG and the truncated Vamana.

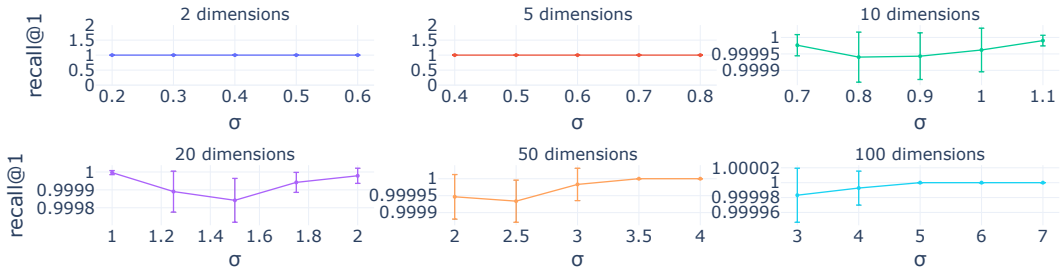


Figure 10: As Theorem 3 predicts, SVGs are navigable graphs in different dimensions and independently of the value of σ. We compute the reachability of Algorithm 1 for every pair of source and target vectors using recall@1 over ten realizations of 500 random vectors with different numbers of dimensions. Small numerical errors in the optimization account for a 0.9999 recall@1.

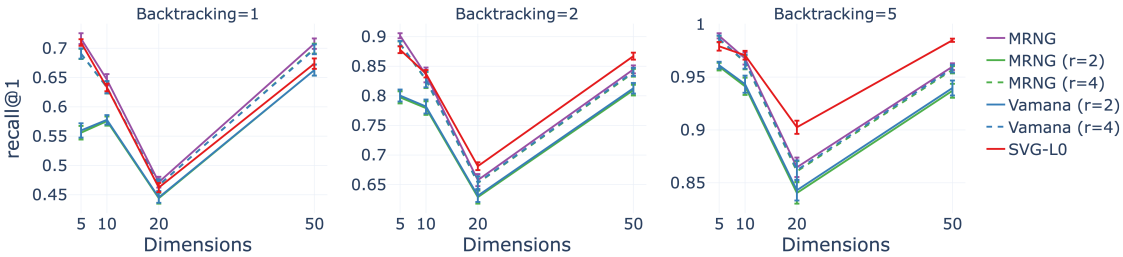


Figure 11: The low-complexity SVG-L0, defined in Problem (26), offers better empirical navigability than that of the truncated MRNG (with similar complexity) and competitive with the degree-constrained MRNG (with a quadratic complexity in the number of vectors). We compute the recall@1 over ten realizations of 10^3 random vectors with different numbers of dimensions. As typical in practical deployments, we include the results of using a backtracking queue (of lengths 2 and 5). For each dimension, we selected the maximum out-degree M that yields reasonable performance (around 85%) for the MRNG. For the truncated MRNG, we define the truncation ratio $r = |\mathcal{C}|/M$, where $\mathcal{C}$ is the candidate pool.

Lastly, we experimented with a few small real-world datasets using a Python implementations of SVG and SVG-L0 that were not optimized to scale. We took 10^4 vectors from the datasets in Table 1 of the appendix with $d = 128, 1024, 1536, 3072$ dimensions. We built indices with SVG-L0, with the truncated MRNG described previously, and with the truncated Vamana. We observe in Figure 12 that SVG-L0 outperforms the truncated MRNG and the truncated Vamana. The effect is more pronounced in higher dimensions. In all dimensions, the value of σ in SVG-L0 has little effect on its accuracy. We include a detailed analysis of the least favorable and a more favorable examples to SVG in Figure 13.

7 Conclusions and future work

We introduced a new type of graph index, the Support Vector Graph (SVG). We derive SVG from a novel perspective that uses machine learning instead of computational geometry to build the index. Concretely, we have formulated the graph construction as a kernelized nonnegative least squares problem. This problem is in turn equivalent to a support vector machine, whose support vectors yield the connectivity of the graph.

We extended the notion of graph navigability to non-Euclidean settings. In this setting, we provide formal navigability results for SVG. To the best of our knowledge, these are the first navigability results that are valid in any metric and non-metric vector spaces (e.g., for inner product similarity).

We formally interpreted the most popular graph indices, including HNSW (Malkov & Yashunin, 2020) and DiskANN (Subramanya et al., 2019), as SVG specializations. We also showed that new traditional (i.e., triangle-pruning) algorithms can be derived from the principles behind this specialization.

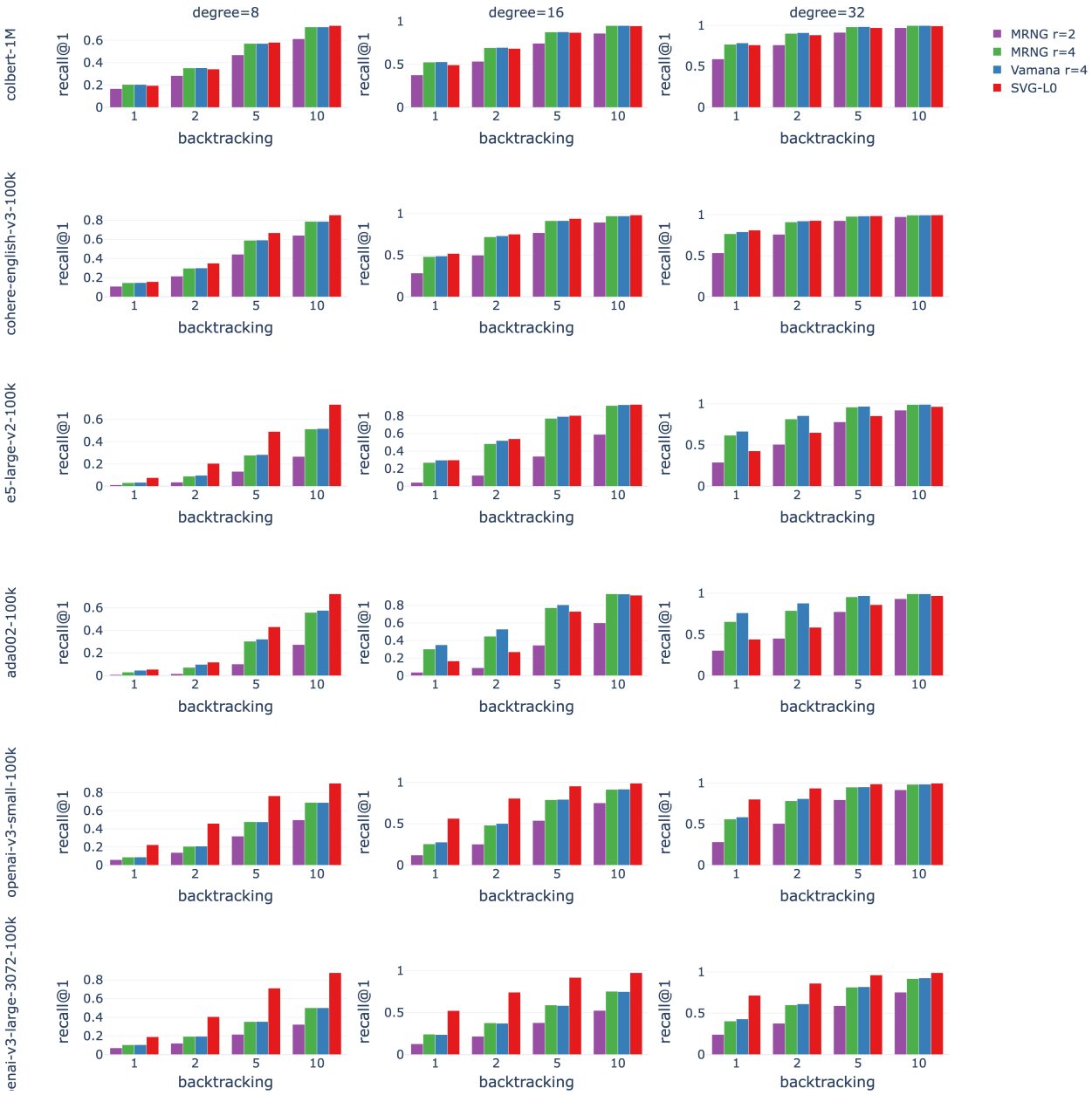
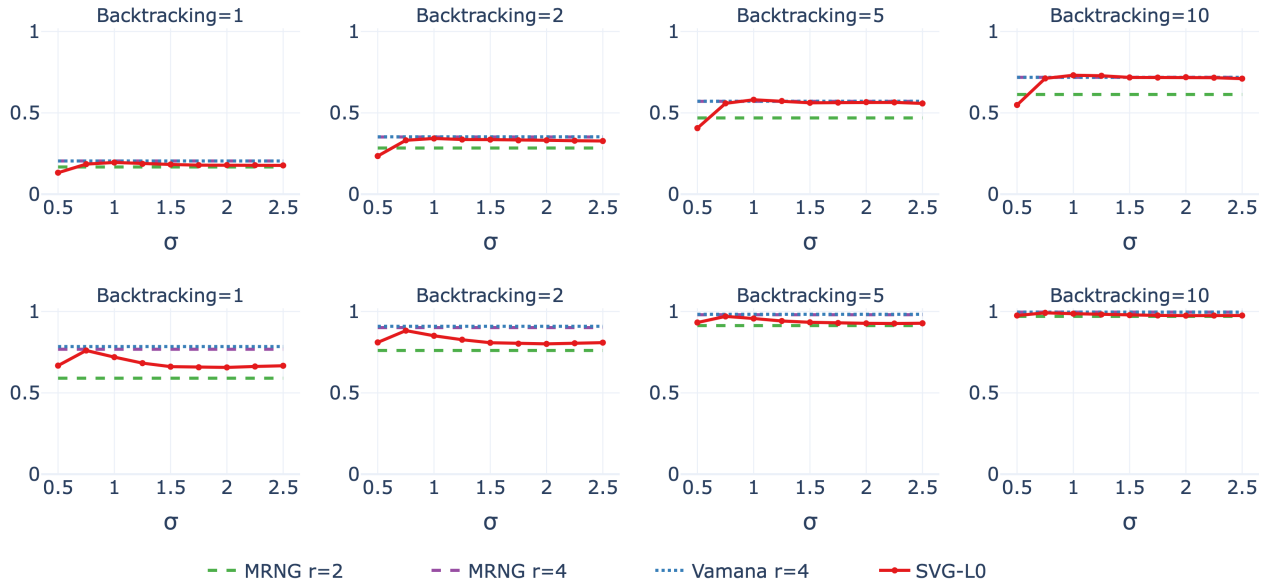
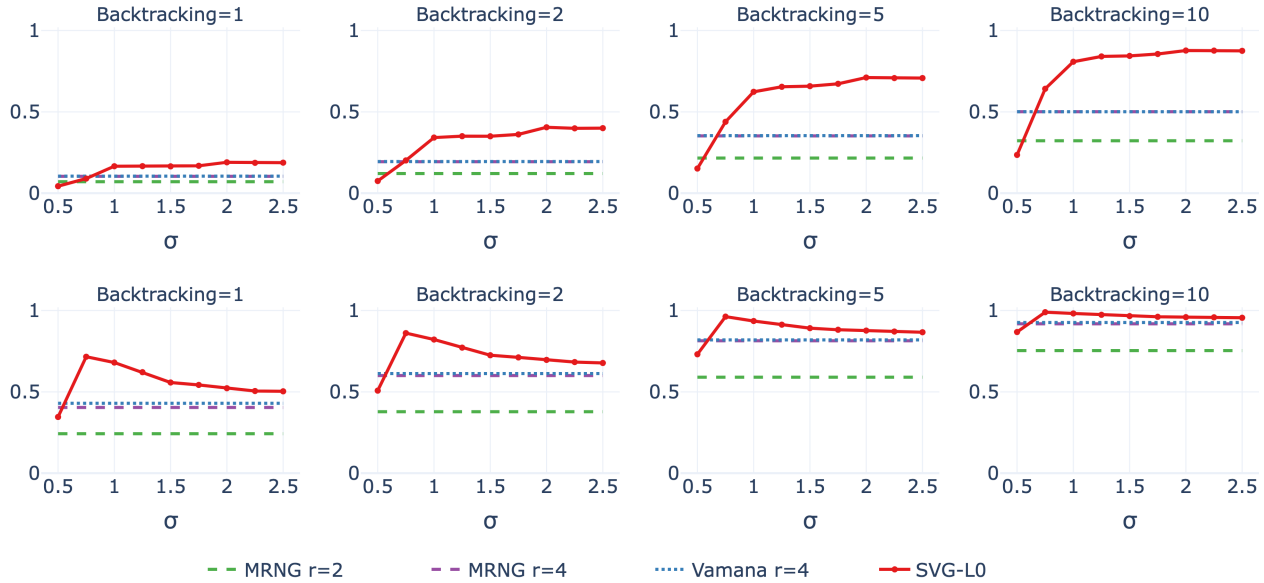


Figure 12: SVG-L0, defined in Problem (26), offers better empirical navigability than that of the truncated MRNG. We compute the recall@1 for different datasets (columns) and maximum out-degrees $M = 8, 16, 32$ (top, center, and bottom rows, respectively). For the truncated MRNG, we define the truncation ratio $r = |\mathcal{C}|/M$, where $\mathcal{C}$ is the candidate pool. For SVG, we set the number of iterations of Algorithm 4 to $T = 4$ so that it uses the same amount of retrieval as $r = 4$. In practice, finding a suitable σ for SVG-L0 is not difficult (automating the selection is left for future work). Full results available in Figure 13 and Appendix C.



(a) Results on colbert-1M with maximum out-degree $M = 8, 32$ (top and bottom rows, respectively).



(b) Results on openai-v3-large-3072-100k with maximum out-degree $M = 8, 32$ (top and bottom rows, respectively).

Figure 13: SVG-L0, defined in Problem (26), offers better empirical navigability (measured by recall@1) than that of the truncated MRNG. For the truncated MRNG, we define the truncation ratio $r = |\mathcal{C}|/M$, where $\mathcal{C}$ is the candidate pool. For SVG, we set the number of iterations of Algorithm 4 to $T = 4$ so that it uses the same amount of retrieval as $r = 4$. In practice, finding a suitable σ for SVG-L0 is not difficult (automating the selection is left for future work).

Finally, we showed that we can build graphs with a bounded out-degree by adding a sparsity (ℓ_0) constraint to the SVG optimization, a combination that we name SVG-L0. SVG-L0 yields a principled way of handling the bound, in contrast to the traditional heuristic of simply truncating the out edges of each node. Additionally, SVG-L0 has a self-tuning property, which avoids selecting a candidate set of edges for each graph node and makes its computational complexity sublinear in the number of indexed vectors.

In future work, we plan to address the following issues to further improve SVG and SVG-L0. First, the kernel width does not matter for SVG but it does matter for SVG-L0 empirically. Tuning the width, although not hard and standard in SVMs (e.g., Chapelle et al., 2002), can be challenging in large-scale scenarios. We will address this problem, seeking an automated selection that does not require cross-validation. Second, determining the maximum out-degree for any graph index remains a challenging problem. Interesting insights may be derived from the tools presented in this work. Lastly, it remains to be seen whether other machine learning techniques, beyond kernel methods, can be utilized to build graph indices.

References

Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*, pp. 271–280, USA, January 1993.

Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18 (9):509–517, September 1975.

Dimitris Bertsimas, Angela King, and Rahul Mazumder. Best Subset Selection Via a Modern Optimization Lens. *The Annals of Statistics*, 44(2):813–852, 2016.

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, ICML '06, pp. 97–104, New York, NY, USA, June 2006.

Jinbo Bi, Kristin Bennett, Mark Embrechts, Curt Breneman, and Minghu Song. Dimensionality reduction via sparse support vector machines. *Journal of Machine Learning Research*, 3:1229–1243, March 2003.

Emmanuel J. Candès and Yaniv Plan. Near-ideal model selection by $\ell 1$ minimization. *The Annals of Statistics*, 37(5A):2145–2177, October 2009.

Olivier Chapelle, Vladimir Vapnik, Olivier Bousquet, and Sayan Mukherjee. Choosing Multiple Parameters for Support Vector Machines. *Machine Learning*, 46(1):131–159, January 2002.

Bin Cheng, Jianchao Yang, Shuicheng Yan, Yun Fu, and Thomas S. Huang. Learning with ℓ^1-graph for image analysis. *IEEE Transactions on Image Processing*, 19(4):858–866, April 2010.

Michel Couprie and Gilles Bertrand. Topological gray-scale watershed transformation. In *Vision Geometry VI*, volume 3168, pp. 136–146. October 1997.

Wei Dai and Olgica Milenkovic. Subspace pursuit for compressive sensing signal reconstruction. *IEEE Transactions on Information Theory*, 55(5):2230–2249, May 2009.

Geoffrey Davis, Stéphane Mallat, and Zhifeng Zhang. Adaptive Time-Frequency Approximations with Matching Pursuits. In *Wavelet Analysis and Its Applications*, volume 5 of *Wavelets: Theory, Algorithms, and Applications*, pp. 271–293. Academic Press, January 1994.

Donald W. Dearholt, N. Gonzales, and Govinda D. Kurup. Monotonic Search Networks For Computer Vision Databases. In *Asilomar Conference on Signals, Systems and Computers*, volume 2, pp. 548–553, October 1988.

Ehsan Elhamifar, Guillermo Sapiro, and René Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1600–1607, June 2012.

Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5):461–474, January 2019.

Cong Fu, Changxu Wang, and Deng Cai. High dimensional similarity search with satellite system graph: efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(8):4139–4150, August 2022.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, New York, NY, 2009.

Trevor Hastie, Robert Tibshirani, and Ryan Tibshirani. Best Subset, Forward Stepwise or Lasso? Analysis and Recommendations Based on Extensive Comparisons. *Statistical Science*, 35(4):579–592, 2020.

Babak Hosseini and Barbara Hammer. Non-negative Local Sparse Coding for Subspace Clustering. In Wouter Duivesteijn, Arno Siebes, and Antti Ukkonen (eds.), *Advances in Intelligent Data Analysis XVII*, pp. 137–150. 2018.

Piotr Indyk and Haike Xu. Worst-case Performance of Popular Approximate Nearest Neighbor Search Implementations: Guarantees and Limitations. In *Advances in Neural Information Processing Systems*, volume 36, pp. 66239 – 66256, 2023.

Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. A Survey on Locality Sensitive Hashing Algorithms and their Applications, February 2021. arXiv:2102.08942 [cs].

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, July 2021.

Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in one billion vectors: Rerank with source coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 861–864, May 2011.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense Passage Retrieval for Open-Domain Question Answering. In *Conference on Empirical Methods in Natural Language Processing*, pp. 6769–6781. November 2020.

Robert Krauthgamer and James R. Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '04, pp. 798–807, USA, January 2004.

Abhishek Kumar, Vikas Sindhwani, and Prabhanjan Kambadur. Fast conical hull algorithms for near-separable non-negative matrix factorization. In *International Conference on Machine Learning*, pp. I–231–I–239, Atlanta, GA, USA, June 2013.

Govinda D. Kurup. *Database organized on the basis of similarities with applications in computer vision*. PhD thesis, New Mexico State University, USA, 1992. UMI Order No. GAX92-01912.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, pp. 9459–9474. December 2020.

Fanghui Liu, Xiaolin Huang, Yudong Chen, and Johan A. K. Suykens. Random features for kernel approximation: a survey on algorithms, theory, and beyond. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):7128–7148, October 2022.

Yu 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*, 42(4):824–836, April 2020.

Julieta Martinez, Joris Clement, Holger H. Hoos, and James J. Little. Revisiting Additive Quantization. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (eds.), *European Conference on Computer Vision*, pp. 137–153. 2016.

P. McMullen. The maximum numbers of faces of a convex polytope. *Mathematika*, 17(2):179–184, 1970. _eprint: https://londmathsoc.onlinelibrary.wiley.com/doi/pdf/10.1112/S0025579300002850.

Nicolai Meinshausen and Peter Bühlmann. High-dimensional graphs and variable selection with the Lasso. *The Annals of Statistics*, 34(3):1436–1462, June 2006.

Marius Muja and David G. Lowe. Scalable Nearest Neighbor Algorithms for High Dimensional Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, November 2014.

Gonzalo Navarro. Searching in metric spaces by spatial approximation. *The VLDB Journal*, 11(1):28–46, August 2002.

Deanna Needell and Joel A. Tropp. CoSaMP: iterative signal recovery from incomplete and inaccurate samples. *Communications of the ACM*, 53(12):93–100, December 2010.

Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: recursive function approximation with applications to wavelet decomposition. In *Asilomar Conference on Signals, Systems and Computers*, pp. 40–44 vol.1, November 1993.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning Transferable Visual Models From Natural Language Supervision. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8748–8763. July 2021.

Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances on Neural Information Processing Systems*, pp. 1177–1184, Red Hook, NY, USA, December 2007.

Isaac Reid, Krzysztof Choromanski, Valerii Likhosherstov, and Adrian Weller. Simplex random features. In *International Conference on Machine Learning*, volume 202, pp. 28864–28888, Honolulu, Hawaii, USA, July 2023.

Luke Sernau, Silvano Bonacina, and Rif A. Saurous. All random features representations are equivalent, June 2024. arXiv: 2406.18802.

Fei Sha, Yuanqing Lin, Lawrence K. Saul, and Daniel D. Lee. Multiplicative Updates for Nonnegative Quadratic Programming. *Neural Comput.*, 19(8):2004–2031, August 2007.

Sarath Shekkizhar and Antonio Ortega. Neighborhood and Graph Constructions using Non-Negative Kernel Regression, April 2023. arXiv:1910.09383.

Martin Slawski and Matthias Hein. Sparse recovery by thresholded non-negative least squares. In *Advances on Neural Information Processing Systems*, pp. 1926–1934, Red Hook, NY, USA, December 2011.

Martin Slawski and Matthias Hein. Non-negative least squares for high-dimensional linear models: consistency and sparse recovery without regularization, February 2014. arXiv:1205.0953.

Alex Smola, Bernhard Scholkopf, and Gunnar Ratsch. Linear programs for automatic accuracy control in regression. In *International Conference on Artificial Neural Networks*, volume 2, pp. 575–580 vol.2, September 1999.

Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. DiskANN: fast accurate billion-point nearest neighbor search on a single node. In *Advances on Neural Information Processing Systems*, pp. 13766–13776, Red Hook, NY, USA, December 2019.

Jingdong Wang, Ting Zhang, jingkuan song, Nicu Sebe, and Heng Tao Shen. A Survey on Learning to Hash. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(4):769–790, April 2018.

Meng Wang and Ao Tang. Conditions for a unique non-negative solution to an underdetermined system. In *Allerton Conference on Communication, Control, and Computing*, pp. 301–307, September 2009.

Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 14(11):1964–1978, July 2021.

Liang Xiao, Bin Dai, Yuqiang Fang, and Tao Wu. Kernel l1 graph for image analysis. In Cheng-Lin Liu, Changshui Zhang, and Liang Wang (eds.), *Pattern Recognition*, Communications in Computer and Information Science, pp. 447–454, Berlin, Heidelberg, 2012.

Penghe Zhang, Naihua Xiu, and Hou-Duo Qi. Sparse SVM with Hard-Margin Loss: a Newton-Augmented Lagrangian Method in Reduced Dimensions, July 2023. arXiv:2307.16281.