Exact and/or Fast Nearest Neighbors

Matthew Francis-Landau, Benjamin Van Durme
Johns Hopkins, Department of Computer Science
{mfl,vandurme}@cs.jhu.edu

Abstract
Prior methods for retrieval of nearest neighbors in high dimensions are fast and approximate—providing probabilistic guarantees of returning the correct answer—or slow and exact performing an exhaustive search. We present Certified Cosine (C2), a novel approach to nearest-neighbors which takes advantage of structure present in the cosine similarity distance metric to offer certificates. When a certificate is constructed, it guarantees that the nearest neighbor set is correct, possibly avoiding an exhaustive search. C2’s certificates work with high dimensional data and outperforms previous exact nearest neighbor methods on these datasets.

1 Introduction
Abstractly, the nearest neighbor problem is defined as given a query \( q \in \mathbb{R}^n \), find the nearest vector \( v_i \in V \) from a discrete set of points according to a distance function \( d(x, y) \) (argmin\(, d(q, v_i) \)). Nearest neighbors occurs frequently as a subproblem in document retrieval (Miller et al. 2016), image search (Johnson, Douze, and Jégou 2017) and language modeling/generation (Bengio et al. 2003; Sugawara, Kobayashi, and Iwasaki 2016). Because \( V \) is often very large, the time spent searching \( V \) dominates the time to evaluate and train a machine learning model. Hence, this has motivated the development of fast nearest neighbor methods.

1.1 Prior Nearest Neighbor Methods
Prior fast nearest neighbor (NN) methods fall into two main categories: exact and approximate. Exact methods, such as KD-trees (Bentley 1975), VP-tree (Yianilos 1993) and cover-trees (Beygelzimer, Kakade, and Langford 2006), only work well in low dimensional settings (such as graphics with 3-dimensions). These methods work by first building an index in the form of a tree data structure which at every level will split the data according to some separating hyperplane. The separating hyperplane may be chosen according to the standard basis such as in KD-trees or a radial basis as in VP-trees. When searching for the nearest neighbor of \( q \), these methods first locate an initial guess \( \hat{v} \) by greedily searching their tree index. Using \( \hat{v} \), these methods will search branches which might contain a better neighbor \( |v_i - q| \leq \|\hat{v} - q\| \) and prune any branch of the tree that probably does not contain anything better than \( \hat{v} \) (Bentley 1975; Beygelzimer, Kakade, and Langford 2006; Yianilos 1993; Ciaccia, Patella, and Zezula 1997; Chen et al. 2018).

The difficulty with these exact methods in high dimensional settings—such as used with machine learning methods—is that clustering of vectors into different branches is incapable of eliminating regions of the search space. Essentially, the distance between the nearest neighbor and the furthest neighbor vanishes making it impossible to prune branches of the search tree\(^1\) (Beyer et al. 1999).

Being unable to prune branches of a search tree has motivated the development of many approximate nearest neighbor (ANN) methods for working with high dimensional data. Instead, these methods search/prune the space probabilistically. Generally, these methods have an \( \epsilon \) parameter to trade-off the recall against the runtime and storage complexity (E.g. \( P(\hat{v} = v^*) \geq 1 - \epsilon \) with a runtime of \( O(\epsilon^{-Q(1)}) \)). The exact details for how search is performed and how the parameter \( \epsilon \) integrates varies significantly between different ANN methods. One common approach has been to use random projections to a lower-dimensional space, such as used by Locality Sensitive Hashing (LSH) and its derivatives (Indyk and Motwani 1998; Charikar 2002; Johnson, Douze, and Jégou 2017; Muja and Lowe 2009; Li and Malik 2017; Kula 2016). With methods like LSH, vectors \( V \) are partitioned into “hashed buckets.” When searching, the probability that a bucket contains the nearest neighbor can be computed by comparing hashing difference between the bucket and the query \( q \). This, in turn, is used to bound the probability that the nearest neighbor is not found. However, if an application requires that the exact nearest neighbors (\( \epsilon = 0 \)), then bounding the probability of not finding \( v^* \) does not work. In the exact case, these methods require searching all buckets and achieve no speedup.

Another class of approximate methods is based on the \( K \)-nearest neighbor graph (KNNG) (Arya and Mount 1993; Os-1Colloquially this problem of vanishing distances between different neighbors is known as the curse of dimensionality.
A KNNG represents all vectors \( v_i \in \mathcal{V} \) as vertices in the graph. Edges of the KNNG correspond with the \( k \)-nearest neighbors for every vector which is computed once during preprocessing. During search, regions that are near to the query \( q \) are prioritized using a queue and searching is cut off heuristically (Boytsov and Naidan 2013; Dong, Moses, and Li 2011; Iwasaki 2016; Iwasaki and Miyazaki 2018; Johnson, Douze, and Jégou 2017; Muja and Lowe 2009; Hajebi et al. 2011; Baranchuk, Babenko, and Malkov 2018; Malkov and Yashunin 2016; Malkov et al. 2014). KNNG based methods tend to perform better than tree-based and bucketing search procedures on dense learned embeddings, as we study in this paper. Unfortunately, KNNG search methods usually do not provide a formal proof for the quality of the return results. These methods still include a tunable parameter \( \epsilon \) to control stopping heuristics which trade-off recall and runtime. KNNG methods, like bucketing methods, are unable to be provably exact without having to search the entire nearest neighbor graph.

1.2 Certified Cosine (\( C_2 \))

In this paper we introduce Certified Cosine (\( C_2 \)), a novel approach for generating certificates for fast nearest neighbor methods. \( C_2 \) builds on prior KNNG based ANN techniques to search for nearest neighbors. However, unlike prior probabilistic and heuristic approaches, \( C_2 \) constructs a certificate which guarantees that the nearest neighbor returned is 100\% correct (\( \epsilon = 0 \)). This allows for \( C_2 \) to be exact and fast when a certificate is successfully constructed. Unfortunately, certificates can not always be efficiently constructed. In this case, depending on the needs of an application, a user of \( C_2 \) can choose to either use the current best guess \( \hat{v} \)— which is akin to current ANN methods—or request that the result is exact and perform a linear scan over all vectors.

In section \( \S2 \), we begin by defining equivalent definitions of what it means to be the nearest neighbor, which can then be used to construct a certificate. In section \( \S3 \), we discuss exactly how we implement our certification strategy such that it is tractable to process while simultaneously performing a fast nearest neighbor search. Finally, in section \( \S5 \), we demonstrate that the additional overhead introduced by our certification processes is manageable and that \( C_2 \) achieves query runtime performance that is comparable to the current state-of-the-art approximate nearest neighbor methods.

2 Definition of 1-Nearest Neighbor

Here we introduce the major definitions that we will use throughout this paper. Given that \( C_2 \) builds on the KNNG, we adopt similar terminology to Sebastian and Kimia (2002) and NGT (Iwasaki 2016; Iwasaki and Miyazaki 2018) as both of these use an exact KNNG in their search procedure as we do here.

We will explain \( C_2 \) as searching for and certifying the 1-nearest neighbor for ease of presentation. However note, \( C_2 \) can be easily generalized to the certify top-\( k \) nearest neighbor set.\(^2\)

We start by defining the query \( q \in \mathbb{R}^n_{\| \cdot \|_1} \) as the target vector and our dataset \( \mathcal{V} \subset \mathbb{R}^n_{\| \cdot \|_1} \) as a set of discrete vectors \( v_i \in \mathcal{V} \) in our vector space. For convenience, we additionally define \( v^* \) as the true 1-nearest neighbor (\( v^* := \text{argmin}_{v_i \in \mathcal{V}} d(v_i, q) \)). For reasons that will become apparent, our method’s distance function is specific to cosine similarity \( d(x, y) = 1 - \cos(x, y) \).\(^3\) Given that our distance metric is cosine similarity, we will assume that all vectors are unit norm, which allows us to write cosine similarity as an inner product between two vectors (\( \cos(x, y) = x^T y \)).

\[
\begin{align*}
v^* &:= \text{argmin}_{v_i \in \mathcal{V}} (1 - \cos(v_i, q)) \\
&= \text{argmax}_{v_i \in \mathcal{V}} v_i^T q \\
\end{align*}
\]

Our search index is based on an exact K nearest neighbor graph (KNNG) (Arya and Mount 1993; Sebastian and Kimia 2002) \( G = (\mathcal{V}, \Gamma) \). KNNG is a directed graph with vertices being the vectors from the original dataset \( \mathcal{V} \) and edges \( \Gamma \) being the top-\( k \) nearest neighbor to a vertex according to our distance metric, cosine similarity. The KNNG is constructed once during a preprocessing phase and then reused for every query requiring \( O(nk) \) to store, as is typical with ANN methods. We denote the edge set of each vertex as \( \Gamma_i \) which contains the \( k \) nearest neighbors. The choice of \( k \) impacts the success rate of constructing certificates vs. storage and search efficiency.

For certification, we additionally define the neighborhood around a point and its associated size \( b_i \). A neighborhood is constructed such that we know for certain that all vertices within a distance \( b_i \) are contained in the neighborhood:

\[
\forall v_j \in \mathcal{V}, \quad v_i^T v_j \geq b_i \implies v_j \in \Gamma_i. \quad (2)
\]

This, in turn, lets us define a ball of size \( b_i \) centered at \( v_i \) that is entirely contained inside of the neighborhood and thus can serve as a compact summary of \( v_i \)’s edge set: \( \mathcal{B}_{b_i}(v_i) \cap \mathcal{V} \subseteq \Gamma_i \).\(^4\) Observe, given that \( \Gamma_i \) is constructed via an exact KNNG, \( b_i \) simply becomes the distance of the \( k \)th nearest neighbor as show in figure 2.

We further define \( \hat{v} \) as our current best guess and \( \mathcal{N}_{v_i} \) as the query’s neighborhood parameterized by some \( v_i \) as \( \mathcal{N}_{v_i} := \mathcal{B}_{b_i}(v_i) \) (Figure 1). Observe, when \( \mathcal{N}_{v_i} \) is parameterized by the current best guess, it represents the space that might contain better \( v_i \). Only in the case that \( \mathcal{N}_{v_i} \) is parameterized by the true 1-nearest neighbor, will its intersection

\(^2\)To prove the top-\( k \) nearest neighbors note, our definition (to follow) of \( \mathcal{N}_{v_i} \) only requires that we know the distance between our current best guess \( \hat{v} \) and the query \( q \). As such, to prove the top-\( k \) nearest neighbors, we instead check the entire region of \( \mathcal{N}_{v_i} \) contains exactly \( k - 1 \) vectors rather than being an empty set, \( |\mathcal{N}_{v_i} \cap \mathcal{V}| = k - 1 \). This, in turn, means that there can not be a better vector located within this region that we have observed (and included in the top-\( k \).

\(^3\)It is possible to preprocess the data such that Euclidean (\( \text{argmin} \|v_i - q\| \)) or maximal inner product (\( \text{argmax} v_i^T q \)) can be converted to cosine and used as the distance metric as shown in Bachrach et al. (2014).

\(^4\)The ball \( \mathcal{B} \) here is defined as usual, but we use the cosine distance: \( \mathcal{B}_C(v) := \{x \in \mathbb{R}^n_{\| \cdot \|_1} : v^T x \geq r \} \) and \( \mathcal{B}_C(v) := \{x \in \mathbb{R}^n_{\| \cdot \|_1} : v^T x > r \} \).
that be the true 1-nearest neighbor.

tarily define how we shrink 

unchecked regions shrinking,

over all of the data is exactly what we are trying to avoid as 

would be to perform a linear scan over all of 

ing a certificate for 

that we need some way to check this statement or an equiv-

ation. A certificate is then the case where we have proven that

counterexample. As such, once we prove that 

rather, we are going to start with the 

assumption 

that 

must be the true 1-nearest neighbor.

with the vertices be empty:

\[ N_v \cap \mathcal{V} = \emptyset \iff \hat{v} = v^* \]  

(3)

2.1 What are Certificates?

A certificate needs to guarantee that \( \hat{v} = v^* \), which means that we need some way to check this statement or an equivalent statement. Equation (3) introduced an equivalence between checking for an empty set intersection and constructing a certificate for \( \hat{v} \). The question remains how to efficiently check this intersection is empty. A simple strategy would be to perform a linear scan over all of \( \mathcal{V} \). Given that \( \mathcal{V} \) is a discrete set, this is tractable. However, a linear scan over all of the data is exactly what we are trying to avoid as a fast NN method.

Rather, we are going to start with the assumption that \( \hat{v} = v^* \) and will search for counterexamples to this assumption. A certificate is then the case where we have proven that a counterexample can not exist. To identify where a counterexamples \( v' \in \mathcal{V} \) might exist, we define the unchecked region (Figure 3) starting with \( S_0 := N_v \) with successive unchecked regions shrinking, \( S_t \subseteq S_{t-1} \). We will momentarily define how we shrink \( S_t \). Now, if \( \hat{v} \) is not the true 1-neighbor, then there must exist a \( v' \in S_t \cap \mathcal{V} \) as a counterexample. As such, once we prove that \( S_t = \emptyset \), then it is impossible for there to exist \( v' \in S_t \) in which case \( \hat{v} \) must 

This follows from a short proof on the sequence \( S_t \):

\[ \begin{align*}
S_t = 0 \implies & S_t \cap \mathcal{V} = S_{t-1} \cap \mathcal{V} = \ldots = S_0 \cap \mathcal{V} = 0 \\
\iff & N_v \cap \mathcal{V} = \emptyset \iff \hat{v} = v^* 
\end{align*} \]  

(4)

Where this chain of equalities follows from equation (5) where we restrict the region that we are checking to \( \mathcal{V} \) and thus only have discrete points that need to be checked.

\[ \begin{align*}
\mathcal{V} \cap (S_{t-1} - S_t) = 0 \iff & \exists v' \in \mathcal{V} \text{ s.t. } v' \in S_{t-1} - S_t. \\
\iff & N_v \cap \mathcal{V} = \emptyset \iff \hat{v} = v^*. 
\end{align*} \]  

(5)

When checking \( \mathcal{V} \cap (S_{t-1} - S_t) \), if we find \( v' \) then that implies a contradiction with the original assumption \( \hat{v} = v^* \).

\[ \exists v' \in \mathcal{V} \text{ s.t. } v' \in S_{t-1} - S_t \implies v' \in S_0 \cap \mathcal{V} \]

\[ \iff \exists v' \in N_v \cap \mathcal{V} \iff \hat{v} \neq v^* \]  

(6)

Procedurally, when finding \( v' \in \mathcal{V} \cap S_t \), \( C_2 \) restarts the certification process using \( v' \) as the new guess for the 1-nearest neighbor (\( \hat{v} \leftarrow v' \)).

To check only this subset \( \mathcal{V} \cap (S_{t-1} - S_t) \) without having to scan all of \( \mathcal{V} \), we make use of the KNNG and the fact that we have preprocessed the neighborhood around every vertex in the graph. Essentially, for some \( v_j \) that is selected during \( C_2 \)'s search procedure, we will have \( S_{t-1} - S_t \subseteq b_j(v_j) \). Thus, we can check the neighborhood of \( v_j \) to mark this area as checked \( (S_{t-1} - S_t) \cap \mathcal{V} \subseteq \Gamma_j \). Checking \( \Gamma_j \) is easy since it is a small set of size \( k \) for which we can check all vectors referenced.

All that we need now to complete \( C_2 \)'s certification process is an efficient way to track \( S_t \) and identify when this set is empty.
3 Tracking the Unchecked Region $S_t$

Our eventual goal is to prove $S_t = \emptyset$ as that indicates that a certificate has been successfully constructed. To make this tractable, we essentially want a compact summary of where we have searched. Now, when searching for the nearest neighbor, once we have searched all of the vectors referenced in a neighbor set $\Gamma_i$, then we know that we have fully searched the neighborhood around $v_i$ (equation (2)).

This in turn means that we can summarize the searched area as: $\{x : x^T v_i \geq b_i\}$. This follows from the fact that we are using cosine similarity as our distance metric which allows us to represent the distance using an inner product (equation (1)), and that we know that all vectors within distance $b_i$ of $v_i$ must be contained within $\Gamma_i$.

To make this more concrete, we define a constraint store $C$, which tracks the regions that is still unchecked. Any time that we have completed processing the neighborhood $\Gamma_j$, we add the constraint $C_t \leftarrow C_{t-1} \cup \{x : x^T v_i \leq b_i\}$, which represents the area that we have not checked. We can now track $S_t$ as the intersection of $C$ and the subspace $\mathbb{R}^n_{\|x\|=1}$ as follows:

$$S_t = \{x : \|x\| = 1\} \cap \{x : x^T q \geq q^T \hat{v}\} \cap \left(\bigcap_{\{x:x^T v_i \leq b_i\} \in C_t} \{x : x^T v_i \leq b_i\}\right)$$  \hspace{1cm} (7)

We can easily handle most of these constraints, as $q$ and $v_i$ are constants, which makes these linear constraints. However, the surface of the sphere, $\|x\| = 1$, is non-convex and thus requires special handling. To actually implement checking if $S_t$ is empty, we employ a number of different strategies to check and relaxations of the above non-convex relaxations that we will cover in the next sections.

3.1 Single Point Certificate

First, the easiest case is where we can prove that $S_1 = \emptyset$ with a single neighbor. This occurs when the distance between the query $q$ and $v_i$ is sufficiently close. Then, it is possible that the query neighborhood will be completely contained inside of neighborhood of $v_i$ as shown in figure 4 ($N_{\hat{v}} \subseteq \mathbb{B}_{b_i(v_i)}$). The check for this case is simply $\cos^{-1}(q^T \hat{v}) + \cos^{-1}(q^T v_i) < \cos^{-1}(b_i)$ where we check if the angle between $v_i$ and $q$ plus the angle defining the query neighborhood for certification ($\cos^{-1}(q^T \hat{v})$) fits inside of the angle that corresponds with $v_i$’s neighborhood size ($\cos^{-1}(b_i)$).

![Figure 4: Single point certificate, $N_{\hat{v}}$ is entirely contained inside $\mathbb{B}_{b_i(v_i)}$.](image)

3.2 Convex Relaxation

Our main goal is to determine if $S_t$ is empty. If we are unable to do that with the single point certificate, then we have to use multiple points to determine if $S_t$ is empty. Our general approach is to identify a counterexample to the claim that $S_t = \emptyset$ by finding a $x \in S_t$. If we have a complete method that only fails to find $x \in S_t$ if and only if $S_t = \emptyset$ then we can use failure to locate $x$ as the certificate.

Unfortunately, it is in general difficult to directly locate $x \in S_t$ as it is non-convex and potentially not even connected due to constraint $\|x\| = 1$. Instead, we use a convex relaxation of $S_t$ by including the inside of the unit ball $\|x\| \leq 1$. With all convex constraints, this problem is known as the convex feasibility problem where we are trying to determine if a set defined as the intersection of multiple convex sets is empty. This problem can be reduced to locating a point inside of the intersection of all of the sets (Bauschke and Borwein 1996). To solve this, we use alternating projection (Listing 1) as it has low overhead, is easy to implement, and provides guarantees of finding some point inside of the intersection if it is non-empty (Bauschke and Borwein 1993). We can additionally alter the order and frequency we check constraints for better efficiency. In the case that the intersection of constraints is empty (also implying $S_t = \emptyset$), then the sequence generated by alternating projection does not converge but rather oscillates between different points in the sets that we are intersecting. When we detect this case, we report that the intersection is empty and the certificate is complete (Line 9 in listing 1).

$$\text{Proj}_c(x) = x - v_i \cdot \max(0, v_i^T x - b_i)$$  \hspace{1cm} (8)

$$\text{Proj}_\hat{v}(x) = x + q \cdot \max(0, q^T \hat{v} - q^T x)$$  \hspace{1cm} (9)

$$\text{Proj}_{\|x\| \leq 1}(x) = \frac{x}{\max(\|x\|, 1)}$$  \hspace{1cm} (10)

Listing 1

Alternating projection loops over all constraints until we identify a point in the intersection.

1: function SOLVEPROJECT($x$, $C$)
2: \>$\triangleright$ Returns $(S_t$ is empty, $x \in \text{conv}(S_t))$
3: for $t \in 1, 2, 3, \ldots$
4: \> $x_o \leftarrow x$
5: for $c \in C : x \leftarrow \text{Proj}_c(x)$ \>$\triangleright$ Eqn. (8)
6: \> $x \leftarrow \text{Proj}_\hat{v}(x)$ \>$\triangleright$ Eqn. (9)
7: \> $x \leftarrow \text{Proj}_{\|x\| \leq 1}(x)$ \>$\triangleright$ Eqn. (10)
8: if $x = x_o$ : return (false, $x$)
9: if $\|x - x_o\| > \alpha^t$ : return (true, $x$)

3.3 Linear Programming Relaxation

We can choose to relax $S_t$ by removing the norm constraint, leaving only linear constraints. By choosing to use the original query $q$ and maximize the objective $q^T x$ with the constraint $q^T x \leq 1$, we can be certain that we will find $x^T q \geq v^T q$ if it exists. As such, if $x^T q < \hat{v}^T q$ then $S_t = \emptyset$ as $\hat{v}^T q$ and by solving the linear programming relaxation, we can find a solution such that we are confident there is nothing better. Off the shelf simplex (B. Dantzig, Orden, and Wolfe 1955) solvers
are optimized for large problems with many sparse constraints, whereas here we have a small number of dense constraints. In support of our experiments, we implement a custom solver optimized for this condition. By not enforcing \( \|x\| \leq 1 \) and using simplex, the solver usually locates a sparse solution that is far outside of the unit ball, figure 5b.

\[
\begin{align*}
\max_{x \in \mathbb{R}^n} & \quad q^T x \\
\text{subject to} & \quad q^T x \leq 1 \\
& \quad \forall \{x : v_i^T x \leq b_i\} \in C, \quad v_i^T x \leq b_i
\end{align*}
\] (11)

\[\begin{align*}
x \in S_t = x_{\text{proj}} + \frac{x_{\text{lp}} - x_{\text{proj}}}{\|x_{\text{lp}} - x_{\text{proj}}\|} & \times \\
& \sqrt{(1 - \|x_{\text{proj}}\|^2)(1 - \cos(x_{\text{proj}}, x_{\text{lp}})^2)} \quad (12)
\end{align*}\]

3.4 Finding a Counterexample, \( x \in S_t \)

If we are merely interested in checking if \( S_t \) is empty, then using either the convex or linear programming relaxations is sufficient. However, when tracking if \( S_t \) is empty, we are also finding counterexamples in the convex relaxations. We can use these points (both outside and inside of the unit ball) to locate a point on the surface of the unit ball inside of the original non-convex \( S_t \). We can further use these new points to target our search towards areas which we have not explored to avoid getting ourselves stuck in a dense, well-connected cluster (Section § 4). To do this, we can draw a line between the result from the projection method and the linear program to find the point along the line that has unit norm (Figure 5c and equation (12)).

4 Finding Good Guesses \( \hat{v} \)

\( C_2 \) certification procedure requires that we first find a good guess \( \hat{v} \) before we can attempt certification. We employ techniques similar to other ANN KNNG based searched methods (Iwasaki and Miyazaki 2018; Iwasaki 2016; Dong, Moses, and Li 2011). In practice, \( C_2 \)’s ANN search procedure could be based on any ANN method, though using a KNNG for search allows us to reuse operations between searching and certification. \( C_2 \)’s search procedure, listing 2, uses a priority queue to track the current nearest unexplored neighbors. We initialize the priority queue with a single seed vector that is selected using LSH using the first \( m \) sign bits (Indyk and Motwani 1998; Charikar 2002). \( C_2 \) eagerly jumps to the current best vector as it is located (listing 2 line 7). When a vertex’s neighbor set \( \Gamma_i \) is fully explored, \( C_2 \) add \( v_i \)'s constraint to \( C \), in turn shrinking \( S_t \). Tracking \( S_t \) also help the search procedure better target its efforts. By locating a \( x \in S_t \), (e.g. \( \text{argmax}_{v \in \mathbb{R}^n} x^T v \)), \( C_2 \) re-prioritizes its search towards areas that are currently unexplored. This helps quickly find vertices which can help with constructing a certificate (listing 2 line 13). \( C_2 \) only terminates its search when a certificate is successfully constructed or when a pre-specified budget has been exceeded.

5 Experimental Results

To compare \( C_2 \), we use ANN-benchmark (ANNB) (Aumüller, Bernhardt, and Faithfull 2018) as a standard testing framework. ANNB includes wrappers for existing state-of-the-art methods as well as standard test sets and hyperparameter configurations for running experiments. Except for KD-trees (Bentley 1975), ball trees (Yianilos 1993; Boytsov and Naidan 2013) and brute force linear scan, the prior work that we compare against only provides probabilistic guarantees of returning the correct answer. Figure 6 plots the runtime (queries per second) vs the recall of the top-10 nearest neighbors. To limit the maximum runtime, \( C_2 \) has a tunable budget parameter (listing 2 line 6) which allows us to limit how many vertices \( C_2 \) expands before returning the current best guess.

In figure 6a, \( C_2 \) dominates the other exact methods. \( C_2 \) run 3 to 30 times faster than Ball-trees (Yianilos 1993; Boytsov and Naidan 2013) for a similar recall. The linear scan using BLAS achieves 10.7 queries per second and KD-trees\(^5\) (Bentley 1975) achieves 1.5 queries per second both with 100% recall. \( C_2 \) similarly achieved 26.9 queries per second at 99.2%. For these experiments, 99.2% was the maximum recall that \( C_2 \) achieved. We did not require that \( C_2 \) construct a certificate for all returned results rather using the best guess in the case that the budget was exceeded.

\(^5\)Note: KD-trees runtime performance on this dataset isn’t uncharacteristic given this is a high dimensional and dense embeddings (Beyer et al. 1999). We primarily include KD-trees as it is the canonical baseline for exact NN methods.
Recall Queries per second (1/s)

(a) GloVe (Pennington, Socher, and Manning 2014) 200 dim 1,183,514 entries

(b) NYT (Dua and Graff 2017) 256 dim 290,000 entries

Figure 6: Recall-Queries per second (1/s) tradeoff of top 10 - up and to the right is better. To sweep out $C_2$’s time vs. recall we adjust the budget of how many vertices we can expand ranging from 1000 to 50000 and controlling $K$, the number of neighbors in the KNNG graph. Plots are generated by ANN-Benchmark (Aumüller, Bernhardsson, and Faithfull 2018). Experiments were run on an Intel E5-2667 v3.

Certified %
Top 1 Recall %
Top 10 Recall %
Query Rate

Figure 7: Here we experiment with queries of different distances from their nearest neighbor. The query distribution has a large impact on performance. The area of $N_{v^*}$ grows very quickly at a rate of $O(d(v^*, q)^{n + 2})$ where $n$ the dimension is generally between 100 and 1000 with this plot at $n = 200$. 
For the ANN methods that we compare against, NGT’s (Iwasaki and Miyazaki 2018) search procedure is the closest to \( C_2 \) s as it also uses an exact KNN to as its primary search data structure. NGT, however, uses a stopping heuristic which allows it to stop searching much earlier than \( C_2 \) and thus greatly benefits it when a lower recall is acceptable. With high recall (> 99%) \( C_2 \) is 14% slower than NGT, likely due to NGT heuristic allowing it to stop earlier. For lower recall, < 95%, other ANN methods are even times faster than \( C_2 \).

These results with \( C_2 \) represent a significant improvement over previously available options for applications which require the exact NN set. The factor of 3 difference from SotA approximate methods also indicates that the ability to heuristically stop earlier is helpful for performance, though this does run counter to \( C_2 \) ability to generate certificates.

5.1 The Impact of \( d(v^*, q) \) on Performance

We observe that the distance between the query \( q \) and the 1-nearest neighbor \( v^* \) has a significant impact on certification success and runtime performance, as shown in figure 7. The impact of \( d(v^*, q) \) is an interesting metric to study as it correlates with the entropy of a query. A low entropy query would have a small distance from its nearest neighbor (hence a distribution over \( V \) is peaked at its 1-nearest neighbor). In figure 7b, we kept the same underlying data as figure 6a but change the distribution of queries. Now, with more lower entropy queries—as we might expect from a well trained model—\( C_2 \) outperforms all other ANN methods for high recall. \( C_2 \) certification ability allows it to stop earlier than the heuristics in these cases as it knows for certain that it has located the correct nearest neighbors.

6 Conclusion

We introduced Certified Cosine (\( C_2 \)), a novel approach for certifying the correctness of the nearest neighbor set. To our knowledge, this is the first time that a (sometimes) exact method has been demonstrated to work well on high dimensional dense learned embeddings. While constructing a certificate is not always feasible, we believe that this approach can help in situations which require correctness and are currently utilizing linear scans. Additionally, we have demonstrated that it is possible to use powerful constraint solvers (sections 3.2 and 3.3) inside of a nearest neighbor lookup while still being competitive. Future work may consider adapting \( C_2 \)’s certificate construction process in designing better heuristics for ANN methods.

References

Arya, S., and Mount, D. M. 1993. Approximate nearest neighbor queries in fixed dimensions. In Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Al-

Listing 2 Outline of our lookup procedure intermixed with constructing certificates. The point \( x \in S_t \) is continuously adjusted to towards under-explored regions to better target search (Section § 4).

```
1: function LOOKUP(q, budget)
2:     count ← 0; certified ← false; x ← q
3:     E ← \{\}
4:     \( Q \leftarrow \{\text{SEED}(x)\} \)
5:     C ← \{\}
6:     while \(|Q| > 0 \) and count++ < budget and not certified :
7:         \( v_j \leftarrow \arg\max_{v' \in Q} x^Tv' \)
8:         \( v_n \leftarrow \arg\max_{v' \in \Gamma_j - E} v_j^Tv' \)
9:     E ← E ∪ \( \{v_n\} \); \( Q \leftarrow Q \setminus \{v_n\} \)
10:    if \( \hat{v}^Tv_q < v_n^Tv_q \): \( \hat{v} \leftarrow v_n \)
11:    if \( \Gamma_j \subset E : \)
12:        \( Q \leftarrow Q - \{v_j\}; C \leftarrow C \cup \{\{x : x^Tv_j \leq b_j\}\} \)
13:    \( \langle x, \text{certified} \rangle \leftarrow \text{CONSTRUCTCERTIFICATE}(q, \hat{v}, C, b_j) \)
14:    return \( \langle \hat{v}, \text{certified} \rangle \)
15: function CONSTRUCTCERTIFICATE(q, C, v_j, b_j)
16:    if \( \cos^{-1}(v_j^Tq) + \cos^{-1}((\hat{v})^Tv_j) < \cos^{-1}(b_j) : \) return \( \langle \_; \text{true} \rangle \)
17:    \( \langle x_{\text{proj}}, \text{emptyIntersection} \rangle \leftarrow \text{SOLVEPROJECT}(q, C) \)
18:    if emptyIntersection : return \( \langle \_; \text{true} \rangle \)
19:    if \( x_{\text{proj}}/\|x_{\text{proj}}\| \in S_t : \) return \( \langle x_{\text{proj}}/\|x_{\text{proj}}\|, \text{false} \rangle \)
20:    \( x_{ip} \leftarrow \text{SOLVESIMPLEX}(q, C) \)
21:    if \( x_{ip}^Tv_q < \hat{v}^Tv_q : \) return \( \langle \_; \text{false} \rangle \)
22:    if \( \|x_{ip}\| < 1 : \) return \( \langle \_; \text{false} \rangle \)
23:    \( x = x_{\text{proj}} + \frac{x_{ip} - x_{\text{proj}}}{\|x_{ip} - x_{\text{proj}}\| \sqrt{(1 - \|x_{\text{proj}}\|)(1 - \cos(x_{\text{proj}}, x_{ip})^2)}} \)
24:    return \( \langle x, \text{false} \rangle \)
```
