Efficient Identification of Tanimoto Nearest Neighbors

David C. Anastasiu* and George Karypis†

*Department of Computer Engineering
San José State University, San José, CA, USA
Email: david.anastasiu@sjsu.edu

†Department of Computer Science and Engineering
University of Minnesota, Twin Cities, MN, USA
Email: karypis@cs.umn.edu

***Abstract*—Tanimoto, or (extended) Jaccard, is an important similarity measure which has seen prominent use in fields such as data mining and chemoinformatics. Many of the existing state-of-the-art methods for market-basket analysis, plagiarism and anomaly detection, compound database search, and ligand-based virtual screening rely heavily on identifying Tanimoto nearest neighbors. Given the rapidly increasing size of data that must be analyzed, new algorithms are needed that can speed up nearest neighbor search, while at the same time providing reliable results. While many search algorithms address the complexity of the task by retrieving only some of the nearest neighbors, we propose a method that finds all of the exact nearest neighbors efficiently by leveraging recent advances in similarity search filtering. We provide tighter filtering bounds for the Tanimoto coefficient and show that our method, TAPNN, greatly outperforms existing baselines across a variety of real-world datasets and similarity thresholds.**

***Index Terms*—Tanimoto, extended Jaccard, similarity search, all-pairs, nearest neighbors, graph construction, similarity graph, NNG.**

I. Introduction

Tanimoto, or (extended) Jaccard, is an important similarity measure which has seen prominent use both in data mining and chemoinformatics. In data mining, for example, it was shown to outperform other similarity functions in text analysis tasks such as clustering [1]–[3], plagiarism detection [4]–[6], and automatic thesaurus extraction [7]. It has also been successfully used to visualize high-dimensional datasets [8], analyze market-basket transactional data [9], recommend items [10], and detect anomalies in spatio-temporal data [11]. In the chemoinformatics domain, data mining and machine learning approaches are increasingly used to boost the effectiveness of the drug-discovery process [12]. Fueled by the generally valid premise that structurally similar molecules exhibit similar binding behavior and have similar properties [13], many chemoinformatics methods use the computation of pairwise similarities as a kernel within their algorithms. Virtual screening (VS), for example, uses similarity search, clustering, classification, and outlier detection to identify structurally diverse compounds that display similar bioactivity, which form the starting point for subsequent chemical screening [14].

In this work, we address the problem of computing pairwise similarities above a threshold ϵ, also known as the *all-pairs similarity search* (APSS) problem, and focus on objects represented numerically as non-negative real-valued vectors. Examples of such objects include text documents [15], user and item profiles in recommender systems [10], market basket data [9], and most existing chemical descriptors. We use the Tanimoto coefficient to measure the similarity of two objects.

Within the chemoinformatics community, a great deal of effort has been spent trying to accelerate pairwise similarity computations using the Tanimoto coefficient. Swamidass and Baldi [16] described a number of *bounds* for fast exact threshold based Tanimoto similarity searches of binary and integer based vector representations of chemical compounds. These bounds allow skipping many object comparisons that will theoretically not be similar enough to be included in the result, a technique ofter referred to as *filtering*, or *pruning*. Other pruning methods relied on hashing techniques [17], [18] or tree-based data structures [19], [20] to accelerate neighbor searches. However, most recent approaches focus on speeding up chemical searches using inverted index data structures borrowed from information retrieval [17], [21], [22].

Data mining methods initially designed to efficiently search databases [23] or the Web [24] were later adapted to solve the APSS problem [25]. Most of the existing work addresses either binary vector object representations [26]–[28] or cosine similarity [29], [30]. However, Bayardo et al. [25] and Lee et al. [31] show how their cosine filtering based APSS methods can be extended to the Tanimoto coefficient for binary and real-valued vectors, respectively. Focusing on real-valued vectors, Kryszkiewicz [32]–[34] proves several theoretic bounds on the Tanimoto similarity and sketches an inverted index based algorithm for efficient similarity search.

We describe a new method for Tanimoto APSS of non-negative real-valued vectors, named `TAPNN`, which solves the problem *exactly*, finding *all* pairs of objects with a Tanimoto similarity of at least some input threshold ϵ. Our method extends the indexing techniques prevalent in the literature with tighter bounds on the similarity of two vectors, which yield dramatic performance improvements. We experimentally evaluated our method against several baselines on chemical datasets derived from the Molecular Libraries Small Molecule Repository (MLSMR) and the SureChEMBL database, and on text collections comprised of newswire stories and USPTO patents. We show that `TAPNN` significantly outperforms baselines for both chemical and text datasets. In particular, it was able to find all near-duplicate pairs among 5M SureChemBL chemical compounds in minutes, using a single CPU core, and is over two orders of magnitude more efficient than linear search in general at $\epsilon = 0.99$.

The remainder of the paper is organized as follows. We give a formal problem statement and describe our notation in Section II. In Section III, we present our algorithm. In Section IV, we describe the datasets, baseline algorithms, and performance measures used in our experiments. We present our experiment results and discuss their implications in Section V, and Section VI concludes the paper.

II. Problem statement

Given a set of objects $D = \{d_1, d_2, \ldots, d_n\}$, such that each object d_i is represented by a (sparse) non-negative vector in an m dimensional feature space, and a minimum threshold ϵ on the similarity of two vectors, we wish to find the set of all pairs (d_i, d_j) satisfying $d_i, d_j \in D$, $d_i \neq d_j$, and $\text{sim}(d_i, d_j) \geq \epsilon$, and compute their similarities. Let $\mathbf{d}_i$ indicate the feature vector associated with the ith object, and $d_{i,j}$ its value (or weight) for the jth feature. We measure vector similarity as the Tanimoto coefficient for real-valued vectors, computed as,

$$\text{T}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle}, \tag{1}$$

where $\langle \mathbf{d}_i, \mathbf{d}_j \rangle = \sum_{l=1}^{m} d_{i,l} \times d_{j,l}$ denotes the vector dot-product, and $\|\mathbf{d}_i\| = \sqrt{\langle \mathbf{d}_i, \mathbf{d}_i \rangle}$ denotes its Euclidean norm, or length. For a given object d_i, we call an object d_j a *neighbor* of d_i if $\text{sim}(d_i, d_j) \geq \epsilon$.

The majority of feature values in sparse vectors are 0. As a result, a vector $\mathbf{d}_i$ is generally represented as the set of all pairs $(j, d_{i,j})$ satisfying $1 \leq j \leq m$ and $d_{i,j} > 0$. For a set of objects represented by sparse vectors, an *inverted index* representation of the set is made up of m lists, $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$, one for each feature. List I_j contains pairs $(d_i, d_{i,j})$, also called postings in the information retrieval literature, where d_i is an indexed object that has a non-zero value for feature j, and $d_{i,j}$ is that value. Postings may store additional statistics related to the feature within the object it is associated with.

Given a vector $\mathbf{d}_i$ and a dimension p, we will denote by $\mathbf{d}_i^{\leq p}$ the vector $(d_{i,1}, \ldots, d_{i,p}, 0, \ldots, 0)$, obtained by keeping the p leading dimensions in $\mathbf{d}_i$, which we call the *prefix* (vector) of $\mathbf{d}_i$. Similarly, we refer to $\mathbf{d}_i^{>p} = (0, \ldots, 0, d_{i,p+1}, \ldots, d_{i,m})$ as the *suffix* of $\mathbf{d}_i$, obtained by setting the first p dimensions of $\mathbf{d}_i$ to 0. Vectors $\mathbf{d}_i^{<p}$ and $\mathbf{d}_i^{\geq p}$ are analogously defined. Table I provides a summary of the notation used in this work.

TABLE I
NOTATION USED THROUGHOUT THE PAPER

	Description
D	set of objects
d_i	the ith object
$\mathbf{d}_i$	vector representing ith object
$d_{i,j}$	value for jth feature in $\mathbf{d}_i$
$\mathbf{d}_i^{\leq p}, \mathbf{d}_i^{>p}$	prefix and suffix of $\mathbf{d}_i$ at dimension p
$\mathbf{d}_i^{\leq}, \mathbf{d}_i^{>}$	un-indexed/indexed portion of $\mathbf{d}_i$
$\hat{\mathbf{d}}_{\mathbf{i}}$	normalized version of $\mathbf{d}_i$
$\mathcal{I}$	inverted index
$\mathbf{f}_j$	vector with jth feature values from all $\hat{\mathbf{d}}_i$
ϵ	minimum desired similarity

III. Methods

We now describe our algorithm that can be used to solve the Tanimoto APSS problem for non-negative real-valued vectors. First, we describe how a bound on the length of indexed vectors can be efficiently integrated into an inverted index APSS approach. We then show how bounds on the cosine similarity of non-negative real-valued vectors can be used to achieve additional pruning. Finally, we introduce new theoretic bounds on the Tanimoto similarity that rely on partially computed dot-products of vectors, and we show how our method can efficiently use these bounds to eliminate object comparisons.

A. A basic indexing approach

One approach to find neighbors for a given query object that has been reported to work well in the similarity search literature [17], [21], [22], [25], [29]–[31] has been to use an inverted index, which makes it possible to avoid computing similarities between the query and objects that do not have any non-zero features in common with it. A map-based data structure, called an *accumulator*, can be used to compute the dot-product of the query with all objects encountered while iterating through the inverted lists for non-zero features in the query. We call an object that has a non-zero accumulated dot-product a *candidate*. Using precomputed lengths for the object vectors, the dot-products of all candidates can be transformed into Tanimoto coefficients according to Equation 1 and those coefficients at or above ϵ can be stored in the output.

One inefficiency with this approach is that it does not take advantage of the commutativity property of the Tanimoto coefficient, computing $\text{sim}(d_i, d_j)$ both when accumulating similarities for d_i and for d_j. To address this issue, authors in [25] and [30] have suggested building the index dynamically, adding the query vector to the index only after finding its neighbors. This ensures that the query is only compared against previously processed objects in a given processing order. We suggest a different approach that is equally efficient given modern compute architectures. Given an object processing order, we first re-label each document to match the processing order, then build the inverted index fully, adding objects to the index in the given processing order. The result will be inverted lists sorted in non-decreasing order of document labels. Then, when iterating through each inverted list, we can stop as soon as the encountered document label is greater or equal to that of the query. Since the document label will have already been read from memory to perform the accumulation operation and will be resident in the processor cache, the additional check against the value of the query label will be very fast, and will be hidden by the latency associated with loading the next cache line from memory.

Kryszkiewicz [32] has shown that some of the objects whose vector lengths are either too small or too large compared to that of the query object cannot be its neighbors and can thus be ignored. An object d_j cannot be a neighbor of a query object d_i if its length $\|\mathbf{d}_j\|$ falls outside the range $[(1/\alpha)\|\mathbf{d}_i\|,\ \alpha\|\mathbf{d}_i\|]$, where $\|\mathbf{d}_i\|$ is the length of the query vector and

$$\alpha = \frac{1}{2}\left(\left(1 + \frac{1}{\epsilon}\right) + \sqrt{\left(1 + \frac{1}{\epsilon}\right)^2 - 4}\right). \quad (2)$$

In Section III-C, we show this bound is actually the limit of a new class of Tanimoto similarity bounds we introduce in this paper. Here, we will show how candidate length pruning can be efficiently integrated into our indexing approach.

A given object will be encountered as many times in the index as it has non-zero features in common with the query. To avoid checking its length against that of the query each time, we could use a data structure, such as a map or bit vector, to mark when a candidate has been checked. While checking this data structure may be less demanding than a multiplication and a comparison, it can actually be slower if the number of candidates is high and the data structure does not fit in the processor cache. Instead, we propose to process objects in non-decreasing vector length order. By re-labeling objects as discussed earlier, objects whose lengths are too short will be potentially found at the beginning of the inverted lists, while objects whose lengths are too long can be automatically ignored, as they will come after the query object in the processing order. Note also that, for an object d_j following d_i in the processing order,

$$\frac{1}{\alpha}\|\mathbf{d}_j\| \geq \frac{1}{\alpha}\|\mathbf{d}_i\|,$$

since $\|\mathbf{d}_j\| \geq \|\mathbf{d}_i\|$ and both vector lengths and α are non-negative real value. As such, the label of the maximum candidate that can be ignored will be non-decreasing. Our approach thus uses a list of starting points, one for each inverted list, and updates the starting point of a list each time a new candidate whose length is too small is found in it.

Algorithm 1 provides a pseudo-code sketch for our basic inverted index based approach. The method first permutes objects in non-decreasing vector length order and indexes them. Then, for each query object d_q, in the processing order, the maximum object d_{max} satisfying $(1/\alpha)\|\mathbf{d}_{max}\| < \|\mathbf{d}_q\|$ is identified. When iterating through the jth inverted list, TAPNN avoids objects in the list whose lengths have already been determined too small by starting the iteration at index $S[j]$, which is incremented as more objects are found with small lengths. At the end of the accumulation stage, the accumulator contains full dot-products between the query and all objects that could be its neighbors. For each such object, the algorithm computes the Tanimoto coefficient using the dot-product stored in the accumulator and adds the object to the result set if its similarity meets the threshold.

Algorithm 1 TAPNN inverted index approach

```
function TAPNN-1($D$, $\epsilon$)
    $A \leftarrow \varnothing$                                    ▷ accumulator
    $S \leftarrow \varnothing$                                    ▷ list starts
    $N \leftarrow \varnothing$                                    ▷ set of neighbors
    Compute and store vector lengths for all objects
    Permute objects in non-decreasing vector length order
    for each $q = 1,\ldots,|D|$ s.t. $\|\mathbf{d}_c\| \leq \|\mathbf{d}_q\| \; \forall c \leq q$ do
        for each $j = 1,\ldots,m$ s.t. $d_{q,j} > 0$ do              ▷ Indexing
            $I_j \leftarrow I_j \cup \{(d_q, d_{q,j})\}$
    for each $q = 1,\ldots,|D|$ s.t. $\|\mathbf{d}_c\| \leq \|\mathbf{d}_q\| \; \forall c \leq q$ do
        Find label $d_{max}$ of last object that can be ignored
        for each $j = 1,\ldots,m$ s.t. $d_{q,j} > 0$ do
            for each $k = S[j],\ldots,|I_j|$ do
                $(d_c, d_{c,j}) \leftarrow I_j[k]$
                if $d_c \leq d_{max}$ then
                    $S[j] \leftarrow S[j] + 1$
                else if $d_c \geq d_q$ then
                    break
                else                                          ▷ Accumulation
                    $A[d_c] \leftarrow A[d_c] + d_{q,j} \times d_{c,j}$
        for each $d_c$ s.t. $A[d_c] > 0$ do                     ▷ Verification
            Scale dot-product in $A[d_c]$ according to Equation 1
            if $A[d_c] \geq \epsilon$ then
                $N \leftarrow N \cup (d_q, d_c, A[d_c])$
return $N$
```

B. Incorporating Cosine Similarity Bounds

A number of recent methods have been devised that use similarity bounds to efficiently solve the cosine similarity APSS problem. Moreover, Lee et al. [31] have shown that, for non-negative vectors and the same threshold ϵ, the set of Tanimoto neighbors of an object is actually a subset of its set of cosine neighbors. This can be seen from the formulas of the two similarity functions.

$$\mathrm{T}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle}$$
$$\mathrm{C}(d_i, d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|}$$

Given a common numerator, it remains to find a relationship between the denominators in the two functions. Since, for any real valued vector lengths, $(\|\mathbf{d}_i\| - \|\mathbf{d}_j\|)^2 \geq 0$, it follows that,

$$\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2 - 2\|\mathbf{d}_i\|\|\mathbf{d}_j\| \geq 0,$$
$$\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle \geq \|\mathbf{d}_i\|\|\mathbf{d}_j\|,$$

where the last equation follows from the Cauchy-Schwarz inequality, which states that $\langle \mathbf{d}_i, \mathbf{d}_j \rangle \leq \|\mathbf{d}_i\|\|\mathbf{d}_j\|$. As a result, the following relationships can be observed between the cosine and Tanimoto similarities of two vectors,

$$\mathrm{T}(d_i, d_j) \leq \mathrm{C}(d_i, d_j),$$
$$\mathrm{T}(d_i, d_j) \geq \epsilon \Rightarrow \mathrm{C}(d_i, d_j) \geq \epsilon,$$
$$\mathrm{C}(d_i, d_j) < \epsilon \Rightarrow \mathrm{T}(d_i, d_j) < \epsilon.$$

One can then solve the Tanimoto APSS problem by first solving the cosine APSS problem and then filtering out those cosine neighbors that are not also Tanimoto neighbors. Given the computed cosine similarity of two vectors and stored vector lengths, the Tanimoto similarity can be derived as follows.

$$\mathrm{T}(d_i, d_j) = \frac{\frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|}}{\frac{\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2 - \langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|}} = \frac{\frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|}}{\frac{\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|} - \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|}}$$

Applying the definition of cosine similarity, we have,

$$\mathrm{T}(d_i,d_j) = \frac{\mathrm{C}(d_i,d_j)}{\frac{\|\mathbf{d}_i\|^2+\|\mathbf{d}_j\|^2}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|} - \mathrm{C}(d_i,d_j)}. \tag{3}$$

Note that,

$$(\|\mathbf{d}_i\| - \|\mathbf{d}_j\|)^2 \geq 0 \Rightarrow \frac{\|\mathbf{d}_i\|^2 + \|\mathbf{d}_j\|^2}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|} \geq 2,$$

which provides a higher pruning threshold when searching for cosine neighbors given a Tanimoto similarity threshold ϵ,

$$\mathrm{T}(d_i,d_j) \geq \epsilon \Rightarrow \frac{\mathrm{C}(d_i,d_j)}{2 - \mathrm{C}(d_i,d_j)} \geq \epsilon \Rightarrow \mathrm{C}(d_i,d_j) \geq \frac{2\epsilon}{1+\epsilon} = t \tag{4}$$

Unlike the Tanimoto coefficient, cosine similarity is length invariant. Vectors can thus be normalized as a pre-processing step, which reduces cosine similarity to the dot-product of the normalized vectors. Denoting by $\hat{\mathbf{d}}_i = \mathbf{d}_i/\|\mathbf{d}_i\|$, the normalized version of the ith object vector,

$$\mathrm{C}(d_i,d_j) = \frac{\langle \mathbf{d}_i, \mathbf{d}_j \rangle}{\|\mathbf{d}_i\|\|\mathbf{d}_j\|} = \langle \hat{\mathbf{d}}_i, \hat{\mathbf{d}}_j \rangle.$$

This step, in fact, reduces the number of floating point operations needed to solve the problem, and is standard in cosine APSS methods. Note that the method outlined in Algorithm 1 can also be applied to normalized vectors, adding only a normalization step before indexing and replacing the scaling factor in line 22, using Equation 3 instead of Equation 1.

In a recent work [30], we described a number of cosine similarity bounds based on the ℓ^2-norm of prefix or suffix vectors that have been found to be more effective than previous known bounds for solving the cosine APSS problem. It may be beneficial to incorporate this type of filtering in our method. However, some of the bounds we described in that work rely on a different object processing order. Our method, therefore, uses similar ℓ^2-norm based bounds that are processing order independent. This allows our method to still take advantage of the vector length based filtering described in Section III-A. In the remainder of this section, we will describe the ℓ^2-norm based filtering in our method.

Normalized vector prefix ℓ^2-norm based filtering

Given a fixed feature processing order and the prefix and suffix of a query object at feature p, it is easy to see that,

$$\langle \hat{\mathbf{d}}_q, \hat{\mathbf{d}}_c \rangle = \langle \hat{\mathbf{d}}_q^{\leq p}, \hat{\mathbf{d}}_c \rangle + \langle \hat{\mathbf{d}}_q^{>p}, \hat{\mathbf{d}}_c \rangle \leq \|\hat{\mathbf{d}}_q^{\leq p}\|\|\hat{\mathbf{d}}_c\| + \langle \hat{\mathbf{d}}_q^{>p}, \hat{\mathbf{d}}_c \rangle,$$

where the inequality follows from applying the Cauchy-Schwarz inequality to the prefix dot-product. Since the maximum value of $\|\hat{\mathbf{d}}_c\|$ is 1, the prefix dot-product can further be upper-bounded by the length of the prefix vector,

$$\langle \hat{\mathbf{d}}_q^{\leq p}, \hat{\mathbf{d}}_c \rangle \leq \|\hat{\mathbf{d}}_q^{\leq p}\|. \tag{5}$$

Another bound on the prefix dot-product can be obtained by considering the maximum values for each feature among all normalized object vectors. Let $\mathbf{f}_j$ denote the vector of all feature values for the jth feature within the normalized vectors, and $\mathbf{mx}$ the vector of maximum such feature values for each dimension, defined as,

$$\mathbf{f}_j = (\hat{d}_{1,j}, \hat{d}_{2,j}, \ldots, \hat{d}_{n,j}),$$
$$\mathbf{mx} = (\|\mathbf{f}_1\|_\infty, \|\mathbf{f}_2\|_\infty, \ldots, \|\mathbf{f}_m\|_\infty).$$

Then,

$$\langle \hat{\mathbf{d}}_q^{\leq p}, \hat{\mathbf{d}}_c \rangle = \sum_{l=1}^{m} d_{q,l} \times d_{c,l} \leq \sum_{l=1}^{m} d_{q,l} \times mx_l = \langle \hat{\mathbf{d}}_q^{\leq p}, \mathbf{mx} \rangle. \tag{6}$$

Combining the bounds in Equation 5 and Equation 6, we obtain a bound on the prefix similarity of a vector with any other object in D, which we denote by $ps_q^{\leq p}$,

$$\langle \hat{\mathbf{d}}_q^{\leq p}, \hat{\mathbf{d}}_c \rangle \leq ps^{\leq p} = \min(\|\hat{\mathbf{d}}_q^{\leq p}\|, \langle \hat{\mathbf{d}}_q^{\leq p}, \mathbf{mx} \rangle). \tag{7}$$

We define $ps_q^{<p}$ analogously.

Algorithm 2 describes how we incorporate cosine similarity bounds within our method. Following examples in [25] and [30], we use the ps bound to index only a few of the non-zeros in each object. Note that, if $ps_q^{<p} < t$, with t defined as in Equation 4, and an object d_c has no features in common with the query in lists I_j, $p \leq j \leq m$, then its cosine similarity to the query will be below t, and its Tanimoto similarity will then be below ϵ. Conversely, if $\langle \hat{\mathbf{d}}_q^{>p}, \hat{\mathbf{d}}_c \rangle > 0$, the object may potentially be a neighbor. By indexing values in each query vector starting at the index p satisfying $ps_q^{\leq p} \geq t$, and then iterating through the index and accumulating, the non-zero values in the accumulator will contain only the suffix dot-products, $\langle \hat{\mathbf{d}}_q, \hat{\mathbf{d}}_c^{>} \rangle$, where $d_c^{>}$ represents the indexed suffix for some object d_c found in the index. Once some value has been accumulated for an object, we refer to it as a *candidate*. This portion of the method can be thought of as *candidate generation* (CG), and is similar in scope to the screening phase of many compound search methods in the chemoinformatics literature. Our method uses the un-indexed portion of the candidate, $d_c^{\leq}$, to complete the dot-product computation during the verification stage, before the scaling and threshold checking steps. We call this portion of the method, which is akin to the verification stage in other chemoinformatics methods, *candidate verification* (CV).

Our method adopts a non-increasing inverted list size order for indexing features, which heuristically leads to shorter lists in the inverted index. The partial indexing strategy presented in the previous paragraph improves the efficiency of our method in two ways. First, objects that have non-zero values in common with the query only in the un-indexed set of query features will be automatically ignored. Our method will not encounter such an object in the index when generating candidates for the query and will thus not accumulate a dot-product for it. Second, the verification stage will require reading from memory only those sparse vectors for un-pruned candidates, iterating through fewer non-zeros in general than exist in the un-indexed portion of all objects.

We use the ps bound in two additional ways to improve the pruning effectiveness of our method. First, when encountering a new potential object in the index during the CG stage

($A[d_c] = 0$), we only accept it as a candidate if $ps_q^{\le j} \ge t$. Note that we process index lists in reverse feature processing order in the CG and CV stages, and $A[d_c]$ contains the exact dot-product $\langle \hat{\mathbf{d}}_q, \hat{\mathbf{d}}_c^{>j} \rangle$. Therefore, if $A[d_c] = 0$ and $ps_q^{\le j} < t$, the candidate cannot be a neighbor of the query object. Second, as a first step in verifying each candidate, we check whether $ps_c^{<}$, added to the accumulated suffix dot-product, meets the threshold t. The value $ps_c^{<}$ is an upper bound of the dot-product of the un-indexed prefix of the candidate vector with any other vector in the dataset. Thus, the candidate can be safely pruned if the check fails.

As in our cosine APSS method [30], after each accumulation operation, in both the CG and CV stages of the algorithm, we check an additional bound, based on the Cauchy-Schwarz inequality. The objects cannot be neighbors if the accumulated suffix dot-product, added to the upper bound $\|\hat{\mathbf{d}}_q^{<j}\| \|\hat{\mathbf{d}}_c^{<j}\|$ of their prefix dot-product, cannot meet the threshold t. We have tested a number of additional candidate verification bounds described in the literature based on vector number of non-zeros, prefix lengths, or prefix sums of the vector feature values, but have found them to be less efficient to compute and in general less effective than our described cosine pruning in a variety of datasets. The interested reader may consult [25], [29]–[31] for details on additional verification bounds for cosine similarity.

Algorithm 2 TAPNN with cosine bounds

```
1: function TAPNN-2($D$, $\epsilon$)
2:     $A \leftarrow \varnothing$, $S \leftarrow \varnothing$, $N \leftarrow \varnothing$
3:     $t \leftarrow 2\epsilon/(1+\epsilon)$
4:     Compute and store vector lengths for all objects
5:     Permute objects in non-decreasing vector length order
6:     for each $q = 1, \dots, |D|$ s.t. $\|\mathbf{d}_c\| \le \|\mathbf{d}_q\| \; \forall c \le q$ do
7:         Normalize $d_q$
8:         for each $j = 1, \dots, m$ s.t. $\hat{d}_{q,j} > 0$ and $ps_q^{\le p} \ge t$ do
9:             $I_j \leftarrow I_j \cup \{(d_q, \hat{d}_{q,j}, \|\hat{\mathbf{d}}_q^{<j}\|)\}$        ▹ Indexing
10:        Store $ps_q^{<}$
11:    for each $q = 1, \dots, |D|$ s.t. $\|\mathbf{d}_c\| \le \|\mathbf{d}_q\| \; \forall c \le q$ do
12:        Find label $d_{max}$ of last object that can be ignored
13:        for each $j = m, \dots, 1$ s.t. $\hat{d}_{q,j} > 0$ do        ▹ CG
14:            for each $k = S[j], \dots, |I_j|$ do
15:                $(d_c, d_{c,j}) \leftarrow I_j[k]$
16:                if $d_c \le d_{max}$ then
17:                    $S[j] \leftarrow S[j] + 1$
18:                else if $d_c \ge d_q$ then
19:                    break
20:                else if $A[d_c] > 0$ or $ps_q^{\le j} \ge t$ then
21:                    $A[d_c] \leftarrow A[d_c] + \hat{d}_{q,j} \times \hat{d}_{c,j}$
22:                    Prune if $A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\| \|\hat{\mathbf{d}}_c^{<j}\| < t$
23:        for each $d_c$ s.t. $A[d_c] > 0$ do        ▹ CV
24:            Prune if $A[d_c] + ps_c^{<} < t$
25:            for each $j = m, \dots, 1$ s.t. $\hat{d}_{c,j}^{\le} > 0$ and $d_{q,j} > 0$ do
26:                $A[d_c] \leftarrow A[d_c] + \hat{d}_{q,j} \times \hat{d}_{c,j}$
27:                Prune if $A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\| \|\hat{\mathbf{d}}_c^{<j}\| < t$
28:            Scale dot-product in $A[d_c]$ according to Equation 3
29:            if $A[d_c] \ge \epsilon$ then
30:                $N \leftarrow N \cup (d_q, d_c, A[d_c])$
31: return $N$
```

C. New Tanimoto similarity bounds

Up to this point, we have used pruning bounds based on the lengths of the un-normalized vectors and prefix ℓ^2-norms of the normalized vectors to either ignore outright or stop considering (prune) those objects that cannot be neighbors for a given query. We will now present new Tanimoto-specific bounds which combine the two concepts to effect additional pruning. First, we will describe a bound on the prefix length of an un-normalized candidate vector, which we use during candidate generation. Then, we will introduce a bound for the length of the un-normalized candidate vector that relies on cosine similarity estimates we compute in our method.

A bound on the prefix length of an un-normalized candidate vector

Recall that the dot-product of a query with a candidate vector can be de-composed as the sum of its prefix and suffix dot-products, which can be written as a function of the respective normalized vector dot-products as,

$$\begin{aligned}\langle \mathbf{d}_q, \mathbf{d}_c \rangle &= \langle \mathbf{d}_q^{\le p}, \mathbf{d}_c \rangle + \langle \mathbf{d}_q^{>p}, \mathbf{d}_c \rangle \\ &= \langle \hat{\mathbf{d}}_q^{\le p}, \hat{\mathbf{d}}_c \rangle \|\mathbf{d}_q^{\le p}\| \|\mathbf{d}_c\| + \langle \hat{\mathbf{d}}_q^{>p}, \hat{\mathbf{d}}_c \rangle \|\mathbf{d}_q^{>p}\| \|\mathbf{d}_c\|.\end{aligned}$$

For an object that has not yet become a candidate ($A[d_c] = 0$), $\langle \hat{\mathbf{d}}_q^{>p}, \hat{\mathbf{d}}_c \rangle = 0$, simplifying the expression to,

$$\langle \mathbf{d}_q, \mathbf{d}_c \rangle = \langle \hat{\mathbf{d}}_q^{\le p}, \hat{\mathbf{d}}_c \rangle \|\mathbf{d}_q^{\le p}\| \|\mathbf{d}_c\|.$$

From the expression $\mathrm{T}(d_c, d_q) \ge \epsilon$, substituting the Tanimoto formula in Equation 1, we can derive,

$$\begin{aligned}\langle \mathbf{d}_q, \mathbf{d}_c \rangle &\ge \frac{\epsilon}{1+\epsilon} \left(\|\mathbf{d}_q\|^2 + \|\mathbf{d}_c\|^2 \right) \\ \|\mathbf{d}_q^{\le p}\| &\ge \frac{\epsilon}{1+\epsilon} \; \frac{\|\mathbf{d}_q\|^2 + \|\mathbf{d}_c\|^2}{\|\mathbf{d}_c\| \; \langle \hat{\mathbf{d}}_q^{\le p}, \hat{\mathbf{d}}_c \rangle} \\ \|\mathbf{d}_q^{\le p}\| &\ge \frac{\epsilon}{1+\epsilon} \; \frac{\|\mathbf{d}_q\|^2 + \|\mathbf{d}_1\|^2}{\|\mathbf{d}_{q-1}\| \; ps_q^{\le j}} \qquad (8)\end{aligned}$$

Equation 8 replaces the prefix dot-product $\langle \hat{\mathbf{d}}_q^{\le p}, \hat{\mathbf{d}}_c \rangle$ with the *ps* upper bound, which represents the dot-product of the query with any potential candidate. Furthermore, taking advantage of the pre-defined object processing order in our method, we replace the numerator candidate length by that of the object with minimum length (the first processed object, d_1) and the denominator candidate length with that of the object with maximum length (the last processed object, d_{q-1}). Since $\|\mathbf{d}_1\|^2 \le \|\mathbf{d}_c\|^2$, $\|\mathbf{d}_{q-1}\| \ge \|\mathbf{d}_c\|$, and $ps_q^{\le j} \ge \langle \hat{\mathbf{d}}_q^{\le p}, \hat{\mathbf{d}}_c \rangle$, the inequality holds.

We use the bound in Equation 8 during the candidate generation stage of our method as a potentially more restrictive condition for accepting new candidates. It complements the *ps* bound in line 20 of Algorithm 2, which checks whether new candidates can still be neighbors based only on the prefix of the normalized query vector. Once the prefix length of the query un-normalized vector falls below the bound in Equation 8, objects that have not already been encountered in the index can no longer be similar enough to the query.

A tighter bound for the un-normalized candidate vector length

Let $\beta = \|\mathbf{d}_c\|/\|\mathbf{d}_q\|$, and, for notation simplicity, $s = \langle\hat{\mathbf{d}}_q, \hat{\mathbf{d}}_c\rangle = \mathrm{C}(d_i, d_j)$. Given $\mathrm{T}(d_q, d_c) \geq \epsilon$, and the pre-imposed object processing order (i.e. $\|\mathbf{d}_q\| \geq \|\mathbf{d}_c\|$), we derive β as a function of the cosine similarity of the objects,

$$\begin{aligned}
&\mathrm{T}(d_q, d_c) = \frac{s\|\mathbf{d}_q\|\|\mathbf{d}_c\|}{\|\mathbf{d}_q\|^2 + \|\mathbf{d}_c\|^2 - s\|\mathbf{d}_q\|\|\mathbf{d}_c\|} \geq \epsilon \\
&\epsilon\|\mathbf{d}_c\|^2 - s(1+\epsilon)\|\mathbf{d}_c\|\|\mathbf{d}_q\| - \epsilon\|\mathbf{d}_q\|^2 \leq 0 \\
&\epsilon\beta^2 - (1+\epsilon)s\beta - \epsilon \leq 0 \\
&\beta = \frac{s(1+\epsilon)}{2\epsilon} + \sqrt{\left(\frac{s(1+\epsilon)}{2\epsilon}\right)^2 - 1} = \frac{s}{t} + \sqrt{\left(\frac{s}{t}\right)^2 - 1} \qquad (9)
\end{aligned}$$

Replacing s with any of the upper bounds on the cosine similarity we described in Section III-B, the bound in Equation 9 allows us to prune any candidate whose length is less than $\|\mathbf{d}_q\|/\beta$. Note that, for $s = 1$, which is the upper limit of the cosine similarity of non-negative real-valued vectors, $\beta = \alpha$, which is the bound introduced by Kryszkiewicz [32] for length-based pruning of candidate vectors. In the presence of an upper bound estimate of the cosine similarity for two vectors, our bound provides a more accurate estimate of the minimum length a candidate vector must have to potentially be a neighbor for the query.

Algorithm 3 The TAPNN algorithm

```
1:  function TAPNN($D$, $\epsilon$)
2:      Lines 2 – 10 in Algorithm 2
3:      for each $q = 1, \dots, |D|$ s.t. $\|\mathbf{d}_c\| \leq \|\mathbf{d}_q\|\ \forall c \leq q$ do
4:          Find label $d_{max}$ of last object that can be ignored
5:          for each $j = m, \dots, 1$ s.t. $\hat{d}_{q,j} > 0$ do                ▹ CG
6:              for each $k = S[j], \dots, |I_j|$ do
7:                  $(d_c, d_{c,j}) \leftarrow I_j[k]$
8:                  if $d_c \leq d_{max}$ then
9:                      $S[j] \leftarrow S[j] + 1$
10:                 else if $d_c \geq d_q$ then
11:                     break
12:                 else if $A[d_c] > 0$ or $[ps_q^{\leq j} \geq t$ and $EQ8]$ then
13:                     $A[d_c] \leftarrow A[d_c] + \hat{d}_{q,j} \times \hat{d}_{c,j}$
14:                     Prune if $A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\|\|\hat{\mathbf{d}}_c^{<j}\| < t$
15:         for each $d_c$ s.t. $A[d_c] > 0$ do                ▹ CV
16:             Prune if $A[d_c] + ps_c^{<} < t$
17:             Compute $\beta$ given $s = A[d_c] + ps_c^{<}$
18:             Prune if $\|\mathbf{d}_c\| \times \beta < \|\mathbf{d}_q\|$
19:             Find first $j$ s.t. $\hat{d}_{c,j}^{\leq} > 0$ and $d_{q,j} > 0$
20:             $A[d_c] \leftarrow A[d_c] + \hat{d}_{q,j} \times \hat{d}_{c,j}$
21:             Prune if $A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\|\|\hat{\mathbf{d}}_c^{<j}\| < t$
22:             Compute $\beta$ given $s = A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\|\|\hat{\mathbf{d}}_c^{<j}\|$
23:             Prune if $\|\mathbf{d}_c\| \times \beta < \|\mathbf{d}_q\|$
24:             for each $j = \dots, 1$ s.t. $\hat{d}_{c,j}^{\leq} > 0$ and $d_{q,j} > 0$ do
25:                 $A[d_c] \leftarrow A[d_c] + \hat{d}_{q,j} \times \hat{d}_{c,j}$
26:                 Prune if $A[d_c] + \|\hat{\mathbf{d}}_q^{<j}\|\|\hat{\mathbf{d}}_c^{<j}\| < t$
27:             Scale dot-product in $A[d_c]$ according to Equation 3
28:             if $A[d_c] \geq \epsilon$ then
29:                 $N \leftarrow N \cup (d_q, d_c, A[d_c])$
30: return $N$
```

In Algorithm 3, we present pseudo-code for the `TAPNN` method, which includes all the pruning strategies we described in Section III. The symbol $EQ8$ in line 12 refers to checking the query prefix vector length, according to Equation 8.

While our bound β for the un-normalized candidate vector length could be checked each time we have a better estimate of the cosine similarity of two vectors, after each accumulation operation, it is more expensive to compute than the simpler prefix ℓ^2-norm cosine bound. We thus check it only twice for each candidate object, first after computing the cosine estimate based on the candidate *ps* bound (line 17), and again after accumulating the first un-indexed feature in the candidate (line 22). We have found this strategy works well in practice.

IV. Materials

In this section, we describe the datasets, baseline algorithms, and performance measures used in our experiments.

A. Datasets

We evaluate each method using several real-world and benchmark text and chemical compound corpora. Their characteristics, including number of rows (n), columns (m), non-zeros (nnz), and mean row/column length (μ_r/μ_c), are detailed in Table II.

1) **Patents** is a random subset of 100,000 patent documents from all US utility patents[1]. Each document contains the patent title, abstract, and body.
2) **RCV1** is a standard text processing benchmark corpus containing over 800,000 newswire stories from Reuters, Ltd [35].
3) **MLSMR** [36] (Molecular Libraries Small Molecule Repository) is a collection of structures of compounds accepted into the repository of PubChem, NCBI's database of small organic molecules and their biological activity. We used the December 2008 version of the SDF database[2].
4) **SC** contains chemical compounds from the SureChEMBL [37] database, which includes a large set of compounds automatically extracted from text, images and attachments of patent documents. **SC-5M**, **SC-1M**, **SC-500k** and **SC-100k** are random subsets of 5,000,000, 1,000,000, 500,000 and 100,000 compounds, respectively, from the **SC** dataset.

1) Text data processing: We used standard text processing methods to encode documents as sparse vectors. Each document was first tokenized, removing punctuation, making text lower-cased, and splitting the document into a set of words. Each word was then stemmed using the Porter stemmer [38], reducing different versions of the same word to a common token. Within the space of all tokens, a document is then represented by the sparse vector containing the frequency of each token present in the document.

[1] http://www.uspto.gov/

[2] https://mlsmr.evotec.com/MLSMR_HomePage/pdf/MLSMR_Collection_20081201.zip

TABLE II
Dataset Statistics

dataset	n	m	nnz	μ_r	μ_c
Patents	100,000	759,044	46.3M	464	61
RCV1	804,414	45,669	61.5M	77	1,348
MLSMR	325,164	20,021	56.1M	173	2,803
SC	11,519,370	7,415	1,784.5M	155	262,669
SC-5M	5,000,000	7,415	699.9M	155	103,063
SC-1M	1,000,000	6,752	154.9M	155	22,949
SC-500k	500,000	6,717	77.5M	155	11,533
SC-100k	100,000	6,623	15.5M	155	2,336

In the table, n represents the number of objects (rows), m is the number of features in the vector representation of the objects (columns), nnz is the number of non-zero values, and μ_r and μ_c are the mean number of non-zeros in each row and column, respectively.

2) Chemical compound processing: We encode each chemical compound as a sparse frequency vector of the molecular fragments it contains, represented by GF [39] descriptors extracted using the AFGen v. 2.0 [40] program[3]. AFGen represents molecules as graphs, with vertices corresponding to atoms and edges to bonds in the molecule. GF descriptors are the complete set of unique size-bounded subgraphs present in each compound. Within the space of all GF descriptors for a compound dataset, a compound is then represented by the sparse vector containing the frequency of each GF descriptor present in the compound. We used a minimum length of 3 and a maximum length of 5 and ignored Hydrogen atoms when generating GF descriptors (AFGen settings *fragtype=GF, lmin=3, lmax=5, fmin=1, noh: yes*). Before running AFGen on each chemical dataset, we used the Open Babel toolbox [41] to remove compounds with incomplete descriptions.

B. Baseline approaches

We compare our methods against the following baselines.

- `IdxJoin` [30] is a straight-forward baseline that does not use any pruning when computing similarities. `IdxJoin` uses an accumulator data structure to simultaneously compute the dot-products of a query object with all other objects, iterating through the inverted lists corresponding to features in the query. While in [30] the method was used to compute dot-products of normalized vectors, we apply the method on the un-normalized vectors. Resulting Tanimoto similarities are computed according to Equation 1 using previously stored vector norms. Then, those similarities below ϵ are removed.
- `L2AP` [30] solves the all-pairs problem for the cosine similarity, rather than the Tanimoto coefficient. As shown in Section III-B, the Tanimoto all-pairs result is a subset of the cosine all-pairs result. After executing the `L2AP` algorithm, we use Equation 3 and previously stored vector norms to compute the Tanimoto coefficient of all resulting object pairs and filter out those below ϵ.
- `MMJoin` [31] is a filtering based approach to solving the all-pairs problem for the Tanimoto coefficient. It relies on efficiently solving the cosine similarity all-pairs problem using pruning bounds based on vector lengths and the number of non-zero features in each vector.
- `MK-Join` is a method we designed using the Tanimoto similarity pruning bounds described by Kryszkiewicz in [32] and [32]. `MK-Join` uses an accumulator to compute similarities of each query against all candidates found in the inverted lists associated with features present in the query. However, `MK-Join` processes inverted lists in a different order, in non-increasing order of the query feature values. By following this order, Kryszkiewicz has shown that the method can *safely* stop accepting new candidates once the squared norm of the partially processed query vector (i.e. setting values of unprocessed features to 0) falls below $t = 1 - (\frac{2\epsilon}{1+\epsilon})^2$. A candidate is also ignored if its length $\|\mathbf{d}_c\|$ falls outside the range $[(1/\alpha)\|\mathbf{d}_q\|, \alpha\|\mathbf{d}_q\|]$, where α is defined as in Equation 2. We also implemented `MK-Join2`, which further incorporates a tighter bound on candidate lengths described by Kryszkiewicz in Theorem 5 of [34]. The bound is equivalent to our Equation 8 with $s = \sqrt{1 - \sum_{i \in L} \hat{d}_{q,i}}$, given the set L of query features that are not also candidate features. Finding this set requires traversing both the query and candidate sparse vectors, which significantly slows down `MK-Join2` in comparison to `MK-Join`. `MK-Join` was superior in all our experiments and we thus only include its results in Section V.

C. Performance measures

When measure search performance of different methods in terms of CPU runtime, which is measured in seconds. I/O time needed to load the dataset into memory or write output to the file system should be the same for all methods and is ignored. Between a method A and a baseline B, we report speedup as the ratio of B's execution time and that of A's.

D. Execution environment

Our method[4] and all baselines are single-threaded, serial programs, implemented in C and compiled using gcc 5.1.0 with the -O3 optimization setting enabled. Each method was executed on its own node in a cluster of HP Linux servers. Each server is a dual-socket machine, equipped with 24 Gb RAM and two four-core 2.6 GHz Intel Xeon 5560 (Nehalem EP) processors with 8 Mb Cache. We executed each method a minimum of four times for $\epsilon \in \{0.6, 0.7, 0.8, 0.9, 0.99\}$ and report the best execution time in each case. Due to its size (14 Gb), we executed data scaling experiments involving the full SC dataset on a different server, equipped with 64 Gb RAM and two 12-core 2.5 Ghz Intel Xeon (Haswell E5-2680v3) processors with 30 Mb Cache. As all tested methods are serial, only one core was used on each server during the execution.

V. Results & Discussion

Our experiment results are organized along several directions. First, we compare the efficiency of our method against

[3] http://glaros.dtc.umn.edu/gkhome/afgen/download

[4] Source code available at http://davidanastasiu.net/software/tapnn/

existing baselines, demonstrating up to an order of magnitude improvement. Then, we analyze the effectiveness of the new Tanimoto pruning bounds in TAPNN, showing that they provide a significant performance benefit. Finally, we analyze the scaling characteristics of our method when dealing with increasing amounts of data.

A. Execution efficiency

The main goal of our method is to efficiently solve the Tanimoto APSS problem. We compared TAPNN against the baselines described in Section IV-B, for ϵ ranging between 0.6 and 0.99, which are thresholds most often used in nearest neighbor based analysis. Figure 1 displays our timing results for each method on four datasets. In each quadrant, smaller times indicate better performance. Note that the y-scale has been log-scaled.

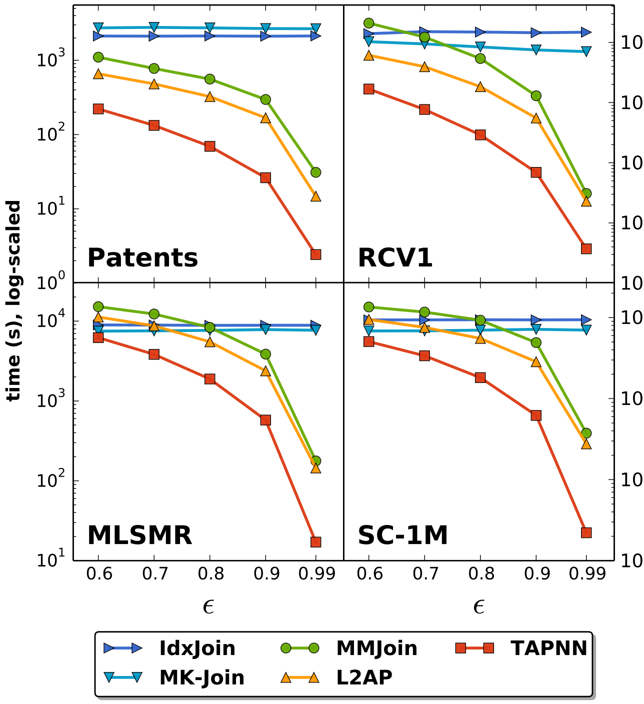

Fig. 1. Efficiency comparison of TAPNN vs. baselines.

The results show that TAPNN significantly outperformed all baselines, by up to an order of magnitude. Speedup of TAPNN versus the next best method ranged between 3.0x–8.0x for text datasets, and 1.2x–12.5x for chemical datasets. Speedup against IdxJoin, which is similar to a linear search and does not employ any pruning ranged between 8.3x–3981.4x for text data and 1.5x–519x for chemical data, highlighting the pruning performance of our method, especially for high values of ϵ.

The best performing baseline in general was our previous cosine APSS method, L2AP, which employs similar cosine based pruning but does not take advantage of un-normalized vector lengths in its filtering. L2AP was shown in [30] to outperform MMJoin for the cosine APSS task. Our results show that it also outperformed MMJoin for Tanimoto APSS, in all experiments. MK-Join was not competitive against L2AP and MMJoin for $\epsilon \geq 0.8$ for chemical datasets and in general for text datasets. In fact, it performed worse than IdxJoin for the Patents dataset, and only slightly better in general. The Patents dataset has a high average vector size (number of non-zeros) and low average index list size, which may have contributed to the poor performance of MK-Join. The results show that the strategy of cosine filtering applied to the Tanimoto APSS problem, which is employed in different ways by TAPNN, L2AP, and MMJoin, works quite well for both text and chemical datasets.

B. Pruning effectiveness

As a way to test the pruning effectiveness of the new Tanimoto length bounds introduced in Section III-C, we compared execution times of TAPNN with two versions of the program which did not take advantage of these bounds. While both programs implement the length based pruning described in Section III-A, TAPNN-c filters cosine neighbors using the threshold ϵ, while TAPNN-t employs the tighter cosine filtering bound from Equation 4. Figure 2 shows the log-scaled execution times for the three methods, for each of the five tested ϵ values.

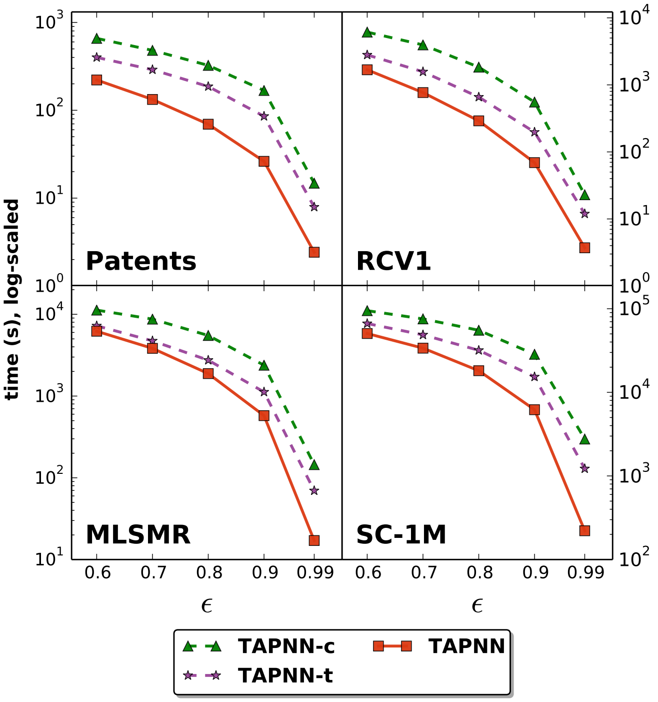

Fig. 2. Effect of Tanimoto bounds on search efficiency.

The results of our experiments indicate that the newly introduced bounds are effective at improving search performance, achieving up to 5.8x speedup against TAPNN-t and 13.3x speedup against TAPNN-c. Chemical datasets exhibit higher performance improvement at high thresholds, but much lower as $\epsilon \rightarrow 0.6$.

C. Scaling

As a way to understand the scalability of our method and baselines, we executed each method on three random subsets from the SC dataset, containing 100K, 500K, and 1M compounds, respectively, and measuring execution time for ϵ ranging between 0.6 and 0.99. Figure 3 shows the results of this experiment. As the problem size was increased, TAPNN maintained a similar advantage over the next best alternative, L2AP. On the other hand, the performance gap between L2AP and MMJoin, as well as between MK-Join and IdxJoin, increased as the problem size was increased.

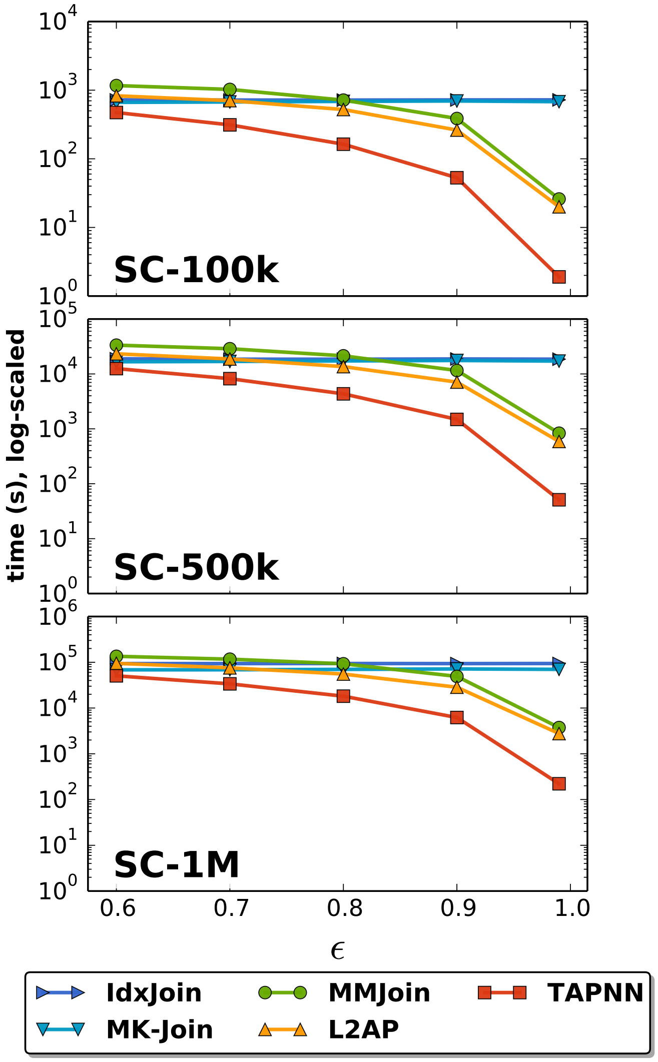

Fig. 3. Execution time scaling given increasing problem size.

We also tested TAPNN in a near-duplicate detection scenario on SC subsets ranging from 500K to 11.5M compounds. Table III presents these results, for $\epsilon \in \{0.95, 0.975, 0.99, 0.999\}$. The columns Δn, Δz, and Δt show relative increases in number of objects, number of non-zeros, and search time, respectively, for corresponding ϵ values, versus the next smaller dataset. For example, $\Delta n = 5.00$, $\Delta z = 4.52$, and $\Delta t = 26.05$ for the SC-5M dataset and $\epsilon = 0.999$, meaning that SC-5M has 5x more compounds, 4.52x more non-zeros, and executed 26.05x slower than SC-1M at $\epsilon = 0.999$. The results show a strong correlation between the increase in the problem size and the search performance in TAPNN. Moreover, the relative performance gap was very similar for the different ϵ values, not showing any significant degradation with decreasing ϵ values. As a near-duplicate detection tool, given $\epsilon = 0.999$, TAPNN was able to search the entire 11.5M compound SC dataset in a little over an hour, and a 5M subset of the compounds in less than 13 minutes, highlighting its effective pruning and efficient search capabilities.

TABLE III
EXECUTION TIME SCALING GIVEN INCREASING PROBLEM SIZE

dataset	ϵ	time (s)	Δn	Δz	Δt
SC	0.999	4,188.06	2.30	2.55	6.51
SC	0.99	41,099.14	2.30	2.55	7.11
SC	0.975	139,887.44	2.30	2.55	7.32
SC	0.95	(371,520.00)	2.30	2.55	(7.23)
SC-5M	0.999	642.93	5.00	4.52	26.05
SC-5M	0.99	5,778.79	5.00	4.52	27.41
SC-5M	0.975	19,122.77	5.00	4.52	24.96
SC-5M	0.95	51,396.57	5.00	4.52	24.12
SC-1M	0.999	24.68	2.00	2.00	3.78
SC-1M	0.99	210.83	2.00	2.00	4.51
SC-1M	0.975	766.03	2.00	2.00	4.34
SC-1M	0.95	2,130.45	2.00	2.00	4.47
SC-500k	0.999	6.53			
SC-500k	0.99	46.78			
SC-500k	0.975	176.55			
SC-500k	0.95	476.55			

Each section of the table shows ϵ-NNG construction times for a different size subset of the SC dataset, given 4 values of ϵ. The columns Δn, Δz, and Δt show relative increases in number of objects, number of non-zeros, and search time, respectively, for corresponding ϵ values, versus the next smaller dataset in the following section in the table. Note that the time for the SC experiment at $\epsilon = 0.95$ is estimated. The experiment was 95% complete when it was terminated at the end of 4 days (96 hours).

VI. Conclusion

We presented TAPNN, a new serial algorithm for solving the Tanimoto all-pairs similarity search problem for objects represented as non-negative real-valued vectors. Unlike many alternatives, our method solves the problem *exactly*, finding *all* pairs of objects with a Tanimoto similarity of at least some input threshold ϵ. Our method incorporates several filtering strategies based on object vector lengths and the dot-product of their normalized vectors. We have shown how these strategies can be effectively used to reduce the number of object pairs that have to be fully compared, and have introduced additional filtering techniques that combine normalized dot-product estimates with un-normalized vector lengths. We experimentally evaluated our method against several baselines on both chemical and text datasets, and found TAPNN significantly outperformed them, especially for high thresholds. In particular, TAPNN was able to find all near-duplicate pairs among 5M SureChemBL chemical compounds in minutes, using a single CPU core, was up to 12.5x more efficient than the most efficient baseline, and outperformed a linear search baseline by two orders of magnitude in general at $\epsilon = 0.99$.

References

[1] A. S. Joydeep, E. Strehl, J. Ghosh, and R. Mooney, "Impact of similarity measures on web-page clustering," in *In Workshop on Artificial Intelligence for Web Search (AAAI 2000*. Citeseer, 2000.

[2] A. Banerjee and J. Ghosh, "Scalable clustering algorithms with balancing constraints," *Data Min. Knowl. Discov.*, vol. 13, no. 3, pp. 365–395, Nov. 2006.

[3] A. Huang, "Similarity measures for text document clustering," in *Proceedings of the sixth new zealand computer science research student conference*, ser. NZCSRSC2008, Christchurch, New Zealand, 2008, pp. 49–56.

[4] C. Lyon, J. Malcolm, and B. Dickerson, "Detecting short passages of similar text in large document collections," in *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing*, 2001, pp. 118–125.

[5] J.-P. Bao and J. Malcolm, "Text similarity in academic conference papers," in *Procs 2nd International Plagiarism Conference*, 2006.

[6] S. M. Alzahrani, N. Salim, and A. Abraham, "Understanding plagiarism linguistic patterns, textual features, and detection methods," *Trans. Sys. Man Cyber Part C*, vol. 42, no. 2, pp. 133–149, Mar. 2012.

[7] J. R. Curran and M. Moens, "Improvements in automatic thesaurus extraction," in *Proceedings of the ACL-02 workshop on Unsupervised lexical acquisition-Volume 9*. Association for Computational Linguistics, 2002, pp. 59–66.

[8] A. Strehl and J. Ghosh, "Relationship-based clustering and visualization for high-dimensional data mining," *INFORMS J. on Computing*, vol. 15, no. 2, pp. 208–230, Apr. 2003.

[9] ——, *A Scalable Approach to Balanced, High-Dimensional Clustering of Market-Baskets*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 525–536.

[10] G. Karypis, "Evaluation of item-based top-n recommendation algorithms," in *Proceedings of the Tenth International Conference on Information and Knowledge Management*, ser. CIKM '01. New York, NY, USA: ACM, 2001, pp. 247–254.

[11] N. R. Adam, V. P. Janeja, and V. Atluri, "Neighborhood based detection of anomalies in high dimensional spatio-temporal sensor datasets," in *Proceedings of the 2004 ACM Symposium on Applied Computing*, ser. SAC '04. New York, NY, USA: ACM, 2004, pp. 576–583.

[12] H. Geppert, M. Vogt, and J. Bajorath, "Current trends in ligand-based virtual screening: Molecular representations, data mining methods, new application areas, and performance evaluation," *Journal of Chemical Information and Modeling*, vol. 50, no. 2, pp. 205–216, 2010.

[13] M. J. Keiser, B. L. Roth, B. N. Armbruster, P. Ernsberger, and B. K. Irwin, John J Shoichet, "Relating protein pharmacology by ligand chemistry," *Nat Biotech*, vol. 25, no. 2, pp. 197–206, 2007.

[14] F. L. Stahura and J. Bajorath, "Virtual screening methods that complement hts," *Comb Chem High Throughput Screen*, vol. 7, no. 4, pp. 259–269, 2004.

[15] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. New York, NY, USA: Cambridge University Press, 2008.

[16] S. J. Swamidass, , and P. Baldi, "Bounds and algorithms for fast exact searches of chemical fingerprints in linear and sublinear time," *Journal of Chemical Information and Modeling*, vol. 47, no. 2, pp. 302–317, 2007.

[17] R. Nasr, D. S. Hirschberg, and P. Baldi, "Hashing algorithms and data structures for rapid searches of fingerprint vectors," *Journal of Chemical Information and Modeling*, vol. 50, no. 8, pp. 1358–1368, 2010.

[18] Y. Tabei and K. Tsuda, "Sketchsort: Fast all pairs similarity search for large databases of molecular fingerprints," *Molecular Informatics*, vol. 30, no. 9, pp. 801–807, 2011. [Online]. Available: http://dx.doi.org/10.1002/minf.201100050

[19] T. G. Kristensen, J. Nielsen, and C. N. S. Pedersen, *Algorithms in Bioinformatics: 9th International Workshop, WABI 2009, Philadelphia, PA, USA, September 12-13, 2009. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ch. A Tree Based Method for the Rapid Screening of Chemical Fingerprints, pp. 194–205.

[20] A. Smellie, "Compressed binary bit trees: A new data structure for accelerating database searching," *Journal of Chemical Information and Modeling*, vol. 49, no. 2, pp. 257–262, 2009.

[21] T. G. Kristensen, J. Nielsen, and C. N. S. Pedersen, "Using inverted indices for accelerating lingo calculations," *Journal of Chemical Information and Modeling*, vol. 51, no. 3, pp. 597–600, 2011.

[22] P. Thiel, L. Sach-Peltason, C. Ottmann, and O. Kohlbacher, "Blocked inverted indices for exact clustering of large chemical spaces," *Journal of Chemical Information and Modeling*, vol. 54, no. 9, pp. 2395–2401, 2014.

[23] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *Proceedings of the 22nd International Conference on Data Engineering*, ser. ICDE '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 5–.

[24] A. Moffat, R. Sacks-davis, R. Wilkinson, and J. Zobel, "Retrieval of partial documents," in *Information Processing and Management*, 1994, pp. 181–190.

[25] R. J. Bayardo, Y. Ma, and R. Srikant, "Scaling up all pairs similarity search," in *Proceedings of the 16th International Conference on World Wide Web*, ser. WWW '07. New York, NY, USA: ACM, 2007, pp. 131–140.

[26] C. Xiao, W. Wang, X. Lin, and J. X. Yu, "Efficient similarity joins for near duplicate detection," in *Proceedings of the 17th International Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 131–140.

[27] C. Xiao, W. Wang, X. Lin, and H. Shang, "Top-k set similarity joins," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*, ser. ICDE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 916–927.

[28] L. A. Ribeiro and T. Härder, "Generalizing prefix filtering to improve set similarity joins," *Inf. Syst.*, vol. 36, no. 1, pp. 62–78, Mar. 2011.

[29] A. Awekar and N. F. Samatova, "Fast matching for all pairs similarity search," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, ser. WI-IAT '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 295–300.

[30] D. C. Anastasiu and G. Karypis, "L2ap: Fast cosine similarity search with prefix l-2 norm bounds," in *30th IEEE International Conference on Data Engineering*, ser. ICDE '14, 2014.

[31] D. Lee, J. Park, J. Shim, and S.-g. Lee, "An efficient similarity join algorithm with cosine similarity predicate," in *Proceedings of the 21st International Conference on Database and Expert Systems Applications: Part II*, ser. DEXA'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 422–436.

[32] M. Kryszkiewicz, "Bounds on lengths of real valued vectors similar with regard to the tanimoto similarity," in *Intelligent Information and Database Systems*, ser. Lecture Notes in Computer Science, A. Selamat, N. Nguyen, and H. Haron, Eds. Springer Berlin Heidelberg, 2013, vol. 7802, pp. 445–454.

[33] ——, "Using non-zero dimensions for the cosine and tanimoto similarity search among real valued vectors," *Fundamenta Informaticae*, vol. 127, no. 1-4, pp. 307–323, 2013.

[34] ——, "Using non-zero dimensions and lengths of vectors for the tanimoto similarity search among real valued vectors," in *Intelligent Information and Database Systems*. Springer International Publishing, 2014, pp. 173–182.

[35] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li, "Rcv1: A new benchmark collection for text categorization research," *J. Mach. Learn. Res.*, vol. 5, pp. 361–397, Dec. 2004.

[36] N. Singh, R. Guha, M. A. Giulianotti, C. Pinilla, R. A. Houghten, and J. L. Medina-Franco, "Chemoinformatic analysis of combinatorial libraries, drugs, natural products, and molecular libraries small molecule repository," *Journal of Chemical Information and Modeling*, vol. 49, no. 4, pp. 1010–1024, 2009.

[37] G. Papadatos, M. Davies, N. Dedman, J. Chambers, A. Gaulton, J. Siddle, R. Koks, S. A. Irvine, J. Pettersson, N. Goncharoff, A. Hersey, and J. P. Overington, "Surechembl: a large-scale, chemically annotated patent document database," *Nucleic Acids Research*, vol. 44, pp. D1220–D1228, 2016.

[38] M. F. Porter, "An algorithm for suffix stripping." *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[39] N. Wale, I. A. Watson, and G. Karypis, "Indirect similarity based methods for effective scaffold-hopping in chemical compounds," *J. Chem. Info. Model*, vol. 48, pp. 730–741, 2008.

[40] N. Wale and G. Karypis, "Acyclic subgraph based descriptor spaces for chemical compound retrieval and classification," in *Proceedings of the Sixth International Conference on Data Mining*, ser. ICDM '06, 2006.

[41] N. M. O'Boyle, M. Banck, C. A. James, C. Morley, T. Vandermeersch, and G. R. Hutchison, "Open babel: An open chemical toolbox," *Journal of Cheminformatics*, vol. 3, no. 1, pp. 1–14, 2011.