COMPUTATIONAL AND STRUCTURAL BIOTECHNOLOGY JOURNAL

Volume No: 5, Issue: 6, February 2013, e201302009, http://dx.doi.org/10.5936/csbj.201302009

CSBJ

Methods for Similarity-based Virtual Screening

Thomas G. Kristensen [a,#], Jesper Nielsen [a,†], Christian N. S. Pedersen [a,*]

Abstract: Developing new medical drugs is expensive. Among the first steps is a screening process, in which molecules in existing chemical libraries are tested for activity against a given target. This requires a lot of resources and manpower. Therefore it has become common to perform a virtual screening, where computers are used for predicting the activity of very large libraries of molecules, to identify the most promising leads for further laboratory experiments. Since computer simulations generally require fewer resources than physical experimentation this can lower the cost of medical and biological research significantly. In this paper we review practically fast algorithms for screening databases of molecules in order to find molecules that are sufficiently similar to a query molecule.

Mini Review Article

Introduction

Proteins are a class of macromolecules that play some of the most important roles in nature. Proteins have functions as catalysts, in signaling and in structural roles. A protein consists of one chain (or multiple chains) of amino acid residues that fold into a more or less rigid structure that has a biological function. A protein that functions as a catalyst will have a certain place, called the *binding site*, where other molecules will *dock* as the protein performs its function. The binding site is usually an indentation or cave in the structure of the protein. A *ligand* is a molecule that docks with another molecule, such as a protein, to perform some function, see figure 1.

One way to combat a disease is to find a ligand that will dock with a protein important for that disease, and disrupt its normal function. In general one will have a chemical library of molecules that are available for manufacturing. Using computers for predicting the activity of very large libraries of molecules to identify the most promising leads for further laboratory experiments is called *virtual screening*. Simulating the docking between the protein and each ligand on a computer in order search for promising ligands in a library of available molecules requires a lot of computing time and available protein structures.

Instead one may rely on the idea that similar structure leads to similar properties, and predict the properties of a molecule by studying the properties of similar molecules. Hence, if one has identified a ligand that binds to a given target, for example from another medical drug, or observed in nature, one may find other candidate ligands by looking for ligands in a chemical library or database that are similar to the known binder. This similarity- and ligand-based approach to virtual screening works well for the right formalizations of how to represent molecules and quantify their similarity [25]. Due to the size of chemical databases such as PubChem [4] and ChemDB [6], the similarity-baed approach to virtual screening also needs efficient methods for screening a database of molecular representations for molecules that are sufficiently similar to a query molecule. In this paper we review such screening methods for molecules represented as fingerprints or SMILES strings.

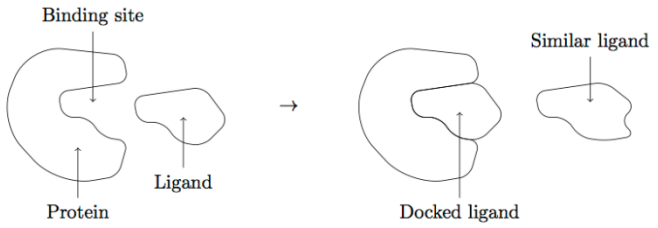

Figure 1. A ligand docking to a protein. Another ligand may dock with the same protein, if it is sufficiently similar.

Representing molecules

It is not immediately obvious how to measure the similarity between two molecules. However, some quite simple measures have proven to be surprisingly good when used for virtual screening [14,22]. For example one might compute a bit-string encoding representative information about the molecules and use the similarity between the bit-strings as a measure of the similarity between the molecules. Such a bit-string for a molecule is denoted a *fingerprint*.

There are many ways to compute the actual fingerprints [5]. One general approach is to select a set of features, each of which a molecule may or may not have. Each feature will then correspond to one bit in the fingerprint, and that bit will be set or not, according to whether the given molecule has the feature [26]. Fingerprints of this form will often be quite long, and with many bits set to zero. To use space more efficiently they can be hashed compressed into shorter fingerprints [1,2,15]. One might also represent a molecule by a *counting vector* of integers, where each integer counts how many

[a]*Bioinformatics Research Center, Aarhus University, C. F. Møllers Allé 8, DK-8000 Aarhus C, Denmark*
#Now employed by Trifork Gmbh
†Now employed by Google Inc

* Corresponding author.
E-mail address: cstorm@birc.au.dk (Christian N. S. Pedersen)

times a certain feature occurs in the molecular. A counting vector allows for a more detailed description of the molecule as a multi-set of features, where a binary fingerprint as introduced above simply describes the molecule as a set of features. However, as counting vectors can easily be converted into binary vectors, for example as illustrated in figure 2, algorithms for handling binary vectors such as the ones reviewed in this paper are also applicable for counting vectors.

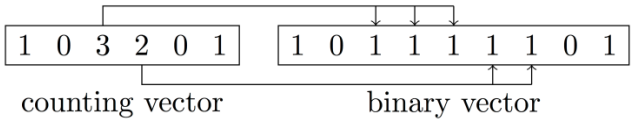

Figure 2. An illustration of converting a counting vector into a binary vector.

The Simplified Molecular Input Line Entry Specification (SMILES) [23] is a standard way to encode the two dimensional structure of a molecule in a one dimensional string that has a canonical form such that every molecule can be represented by a unique SMILES. The SMILES string is generated by writing a sequence of letters, one for each atom type, marking branches with parentheses and rings with numerical indexes. As an example, consider the visualization of 3-cyanoanisole in figure 3, which can be represented by the SMILES string "COc(c1)cccc1C\# N". The main path of the molecule is the string "COccccC\# N", the hash mark symbolizing a triple bond. (c1) marks the branch containing just one carbon atom, and the number "1" here and later in the path defines the bond between the two carbon atoms.

Any string of length $n \geq q$ will have exactly n-q+1 substrings of length q. In [21], a substring, of length q, of a SMILES string is called a LINGO. Thus the SMILES string of a ligand can be viewed as a multi-set of LINGOs, which in [21] is called the LINGO profile of the molecule.

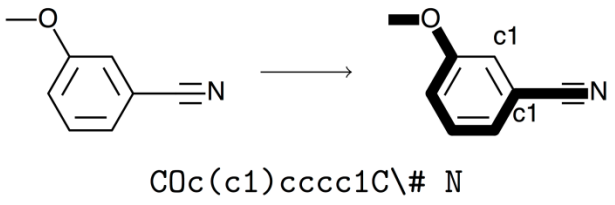

Figure 3. Illustration of a possible SMILES string for 3-cyanoanisole. The primary backbone is highlighted with thick lines. c1 indicates the two points where the ring is merged.

Similarity between molecules

There are of course several ways to quantify the similarity between two sets (or multi-sets) of features, but the *Tanimoto coefficient* has proven very useful [24,26]. If A and B are sets, or multi-sets, of features, then the Tanimoto coefficient, $S_T(A, B)$, is:

$$S_T(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

If A and B are given as two bit-strings, then the Tanimoto coefficient becomes:

$$S_T(A,B) = \frac{|A \wedge B|}{|A \vee B|},$$

where $\wedge$ and $\vee$ are bitwise logical 'and' and logical 'or' respectively, and $|A|$ is the number of bits set to one in the bit-string A. See figure 4 for an example.

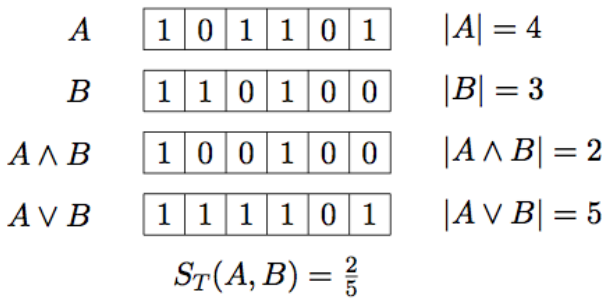

Figure 4. The notation used for bit-strings.

The Tanimoto coefficient as defined above quantifies the similarity between two bit-strings as a number in the interval [0;1], where 0 says that the two bit-strings have no one-bits in common, and 1 says that the two bit-strings are equal. The coefficient is only defined if there is at least one bit set to one in the two bit-strings (i.e. one feature is shared), which is a very reasonable assumption for molecular fingerprints.

Recall, that the LINGO profile of a molecule is the multi-set of LINGOs in its SMILES string. The similarity between two ligands can thus be measured as the Tanimoto coefficient between their LINGO profiles. This measure is called the LINGOsim between the ligands [21].

One of the major motivations for quantifying molecular similarity is to identify molecules for medical drugs. The problem can be formalized as: We are given a database of representations (for example fingerprints or SMILES) of synthesizable molecules, a query molecule A, and a minimal similarity S_{MIN}. The task is then to find all molecules B in the database where $S_T(A, B) \geq S_{MIN}$. This query can of course be performed by a naive screening of the database, where we examine every fingerprint A in the database to compute $S_T(A, B)$. However, due to the typical size of the database, this is not a desirable approach. In the following sections, we review how to perform such queries more efficiently in practice. We first consider the problem for molecules represented as bit-strings (fingerprints), and secondly, for molecules represented as SMILES.

Searching for molecules with similar fingerprints

Given a database of N fingerprints of length n, a query fingerprint A (also of length n), and a minimal similarity S_{MIN}. We want to find all molecules B in the database where $S_T(A, B) \geq S_{MIN}$. In [19] it is observed that since $|A \wedge B| \leq \min(|A|, |B|)$ and $|A \vee B| \geq \max(|A|, |B|)$, then we can upper-bound the similarity between A and B by

$$S_T(A,B) = \frac{|A \wedge B|}{|A \vee B|} \leq \frac{\min(|A|,|B|)}{\max(|A|,|B|)} = \text{COUNT-MAX}\,(A,B).$$

Such an upper-bound can be used to make queries faster than a simple linear search by sorting the fingerprints B in the database into bins depending on their counts of one-bits $|B|$. When a query is performed we can compute which bins has a COUNT-MAX $(A, B) \geq S_{MIN}$ and only examine the fingerprints in those bins, i.e. only compute $S_T(A, B)$ for the fingerprints B in those bins.

In a later paper [3], it is suggested to use a filter based on XOR signatures to improve this pruning even further. The idea is to first split the fingerprints into k equal-sized fragments such that $A = A_1 A_2 \cdots A_k$ and $B = B_1 B_2 \cdots B_k$ and then compute XOR-signatures, a and b of A and B, as $a = A_1 \oplus A_2 \oplus \cdots \oplus A_k$, and $b = B_1 \oplus B_2 \oplus \cdots \oplus B_k$. Since $|A \wedge B| = (|A| + |B| - |A \oplus B|) / 2$ and $|A \vee B| = (|A| + |B| + |A \oplus B|) / 2$ and we can lower-bound the size of the XOR of the fingerprints by the size of the XOR of the signatures, i.e. $|A \oplus B| \geq |a \oplus b|$, then we can bound the similarity as

$$S_T(A,B) = \frac{|A| + |B| - |A \oplus B|}{|A| + |B| + |A \oplus B|} \leq \frac{|A| + |B| - |a \oplus b|}{|A| + |B| + |a \oplus b|}$$

$$= \text{XOR-MAX}\,(A,B)\,.$$

Since the XOR signatures a and b are shorter than the original fingerprints A and B, we can compute $a \oplus b$ faster than $A \oplus B$. This is used as a filter, where the fingerprints B in the database DB are still stored in bins depending on $|B|$, but the signature of each fingerprint is stored with it, and the final $S_T(A, B)$ is only computed for fingerprints where XOR-MAX $(A, B) \geq S_{MIN}$.

In [16] it is suggested to store the database DB as a trie [7]. The key observation is that by walking down the trie one can bound the similarity between the query fingerprint B and any fingerprint B in a leaf below the current node in the trie. Consider a node at a level d in a trie. Let A_{HEAD} be the first d bits of the query fingerprint A, and A_{TAIL} be the remaining $n-d$ bits. Similarly for an arbitrary database fingerprint B below the node. We may now observe that

$$|A \wedge B| \leq |A_{\text{HEAD}} \wedge B_{\text{HEAD}}| + |A_{\text{TAIL}}|\,,$$

$$|A \vee B| \geq |A_{\text{HEAD}} \vee B_{\text{HEAD}}| + |A_{\text{TAIL}}|\,,$$

and hence

$$S_T(A,B) = \frac{|A \wedge B|}{|A \vee B|} \leq \frac{|A_{\text{HEAD}} \wedge B_{\text{HEAD}}| + |A_{\text{TAIL}}|}{|A_{\text{HEAD}} \vee B_{\text{HEAD}}| + |A_{\text{TAIL}}|}$$

$$= \text{TRIE-MAX}\,(A,B)\,.$$

Thus, like above, we need only visit the children of the current node if TRIE-MAX $(A, B) \geq S_{MIN}$. A trie easily takes up a lot of memory, so in [17] the trie is compressed by collapsing long runs of zero-bits into one node. This works well, because molecular fingerprints tend to be sparse.

In [11], we present the kD-grid, which is a data structure for supporting fast queries in practice building upon the ideas outlined above in the sense that it corresponds to the approaches in [17] and [19] for certain choices of the parameter k.

In the kD-grid, we split all fingerprints into k equal-sized fragments such that $A = A_1 A_2 \cdots A_k$ and $B = B_1 B_2 \cdots B_k$, and all database fingerprints B in the database DB are placed into bins in a k dimensional grid, based on the bit counts of the fragments $|B_i|$. Figure 5 illustrates how a fingerprint is stored in a 3D-grid. Like above, we can compute bounds on the similarity between a query fingerprint A and the database fingerprints in any of these bins

$$S_T(A,B) = \frac{|A \wedge B|}{|A \vee B|} \leq \frac{\sum_{i=1}^{k} \min\{|A_i|,|B_i|\}}{\sum_{i=1}^{k} \max\{|A_i|,|B_i|\}}$$

$$= \text{GRID-MAX}\,(A,B)\,.$$

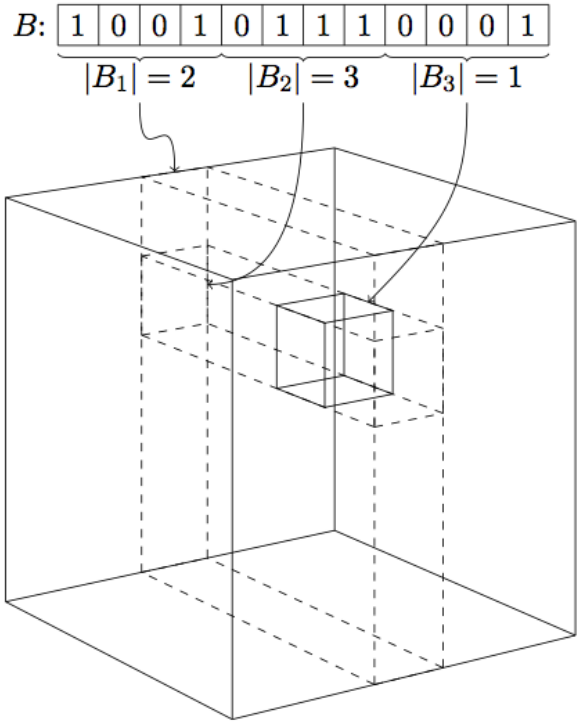

Figure 5. An example of a fingerprint being stored in a 3D-grid.

In practice we implement the grid as a tree with k levels and leaves of degree n/k, but with branches without leaf fingerprints pruned, see figure 6. When looking up a query, we walk down the tree and can compute bounds on all sub-branches. Assume we are visiting a node at level l in the tree. The bound is then

$$S_T(A,B) \leq \frac{\sum_{i=1}^{k} \min\{|A_i|,|B_i|\} + \sum_{i=l+1}^{k} |A_i|}{\sum_{i=1}^{k} \max\{|A_i|,|B_i|\} + \sum_{i=l+1}^{k} |A_i|}\,.$$

Note that if we set $k = 1$ this corresponds to the approach of [19] and if we set $k = n$ this becomes the trie of [17].

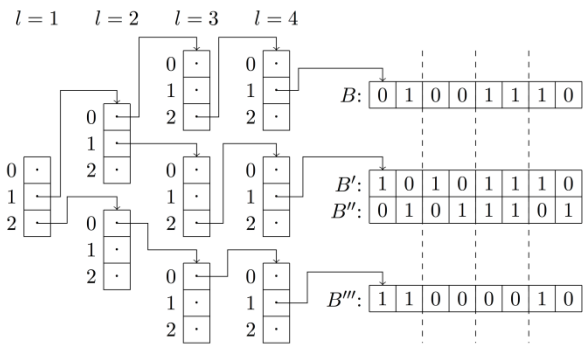

Figure 6. An example of four fingerprints being stored in a tree representing a 4D-grid.

Naively one would store the fingerprints in each bin in a simple list, but one can do better. In [12] we present two alternative data structures for representing the bins. The first is the *singlebit tree*. The fingerprints for a given bin will be stored in the leaves of a tree, while the internal nodes each store the index of a bit. Fingerprints with the indexed bit clear will be stored in the left sub-tree of the given node, and fingerprints with the indexed bit set will be stored in the right sub-tree, see figure 7. Thus it is similar to a trie, except the bits in the bitstring can be examined in any arbitrary order, instead of left-to-right, as they are in a trie. Also, since we know what bucket the singlebit tree is sitting in we have information about the number of set bits for all fingerprints in the entire tree, which allows us to derive tighter bounds than those of [17]. Let M_{ij} be the count of positions where A has an i-bit and B has a j-bit. For example M_{10} is the number of positions where A has a one and B has a zero. Walking down a singlebit tree we will obtain partial knowledge of M_{ij}, as we compare the bits in the nodes of the tree with those in A. Let the number of positions we have knowledge about, and where A has an i-bit and B has a j-bit, be m_{ij} and the unknown difference m_{ij} and M_{ij} be u_{ij}, i.e. $M_{ij} = m_{ij} + u_{ij}$. Now we can bound the Tanimoto coefficient by

$$S_T(A,B) = \frac{M_{11}}{M_{01} + M_{10} + M_{11}} \leq \frac{\min\{|A| - m_{10}, |B| - m_{01}\}}{\max\{|A| + m_{01}, |B| + m_{10}\}},$$

and only visit subtrees where the leaves may be sufficiently similar to the query fingerprint.

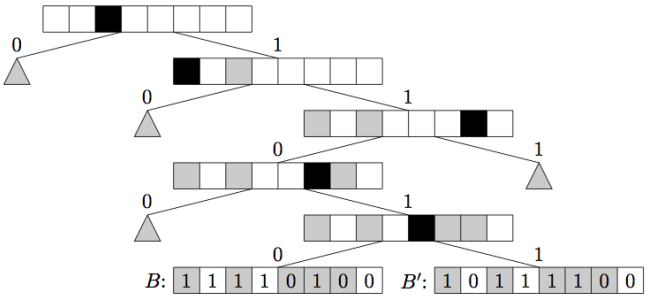

Figure 7. Example of a singlebit tree. The black squares denote the bits on which the data is split; while the gray squares denote bits we have information about from further up the tree.

Improving upon this we also suggested the *multibit tree*. The multibit tree is similar to the singlebit tree, but stores several bits in each internal node instead of only one. This means that we can no longer split the children of a node based on whether they have a one or a zero, thus the semantics needs to change somewhat. For each node in the tree store a list of bit positions, along with a Boolean value. These bits we call the match bits. The match bits of a node are exactly those bits for which all the children of the node have the same value and that is not a match bit further up the tree see figure 8. Walking down a multibit tree we again gain partial knowledge about the leaves of the tree and exactly the same bound as that of the singlebit tree may be used.

How best to build the singlebit and multibit trees are not obvious. The algorithm we used in our implementation is to split the dataset recursively into smaller and smaller subsets. For each set of fingerprints we choose the bit that splits the tree into two subsets that are maximally close to having the same size. The set is then split into two subsets based on whether that bit is set or not for each fingerprint. The reasoning is to attempt to obtain a tree that is as well balanced as possible. Theoretically it is not clear that this is the right way to build the trees, but in practice the methods perform well, as illustrated by the experimental results reported in figure 9. The SymDex method [20] is a recent method that is reported to perform even better.

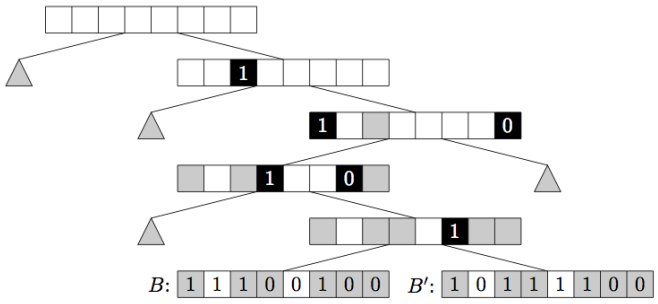

Figure 8. Example of a multibit tree. The black squares denote the match bits, while the gray squares denote bits that are match bits further up the tree.

Searching for molecules using LINGOsim

The LINGOsim [21] similarity measure is attractive because it only relies on the SMILES description of the molecule, and it has proven to be competitive with more computationally expensive methods for predicting ligand properties, despite its simplicity [8]. An efficient method for computing the LINGOsim, using a finete state machine, is presented in [8]. Given a query SMILES string A and a database, they suggest building a finite state machine from A to be able to quickly compare it against any other SMILES string.

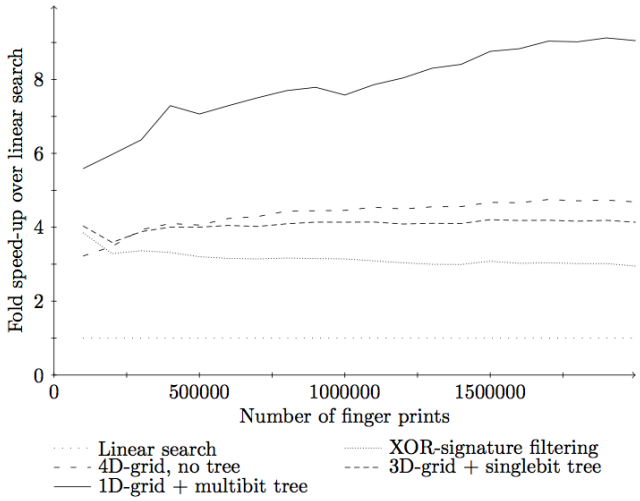

Figure 9. The result of an experiment that compares the running time versus database size for our algorithms, the previous best algorithm (XOR-signature filtering [3]), and a naive linear search. Fingerprints were generated using the CDK fingerprint generator [18] with a standard fingerprint size of 1024. We have performed tests on data set, containing from 100,000 to 2,000,000 fingerprints in 100,000 increments. For each data set size, the entire data structure was created. Next, the first 100 fingerprints in the database are used for queries. Each experiment is performed 100 times, and the average query time is presented as the speed-up compared to the naive linear search. All experiments are performed with a S_{MIN} of 0.9. For each kD-grid, the k (1, 2, 3, or 4) that gave the best results was chosen.

Recall, that the LINGO profile of a molecule is the multi-set of LINGOs in its SMILES string, and that the LINGOsim similarity between two ligands is the Tanimoto coefficient between their LINGO profiles. The algorithm described in [8] starts by building a trie from all length q substrings of A. This trie is then converted into a finite state machine, where states correspond to length q strings, and substrings in A are accepting states. By running this state machine on another SMILES string B, the size of the intersection of A and B can found efficiently, and since $|A| + |B| = |A \cap B| + |A \cup B|$, we can also find the size of the union of A and B efficiently. With the sizes of the intersection and union in hand, the LINGOsim is straightforward to obtain.

A parallel algorithm is suggested in [9]. Their first observation is that since a character in a computer normally uses eight bits, and it is shown in both [21] and [8] that the optimal length of LINGOs is $q = 4$, then a LINGO can be stored in a 32-bit computer word. For each molecule they explicitly store a sorted list of all LINGOs in the SMILES string of the molecule, along with a count of how many times each LINGO occur in the molecule. This allows the intersection size of LINGOs between two molecules to be computed by iterating over the two LINGO lists simultaneously, similar to the merge of a merge-sort. As above the intersection size is enough to compute the LINGOsim. Parallelization is achieved by processing several molecules at the same time and the paper presents implementations for both CPUs and GPUs. They name their algorithm SIML for Single-Instruction Multiple-LINGO.

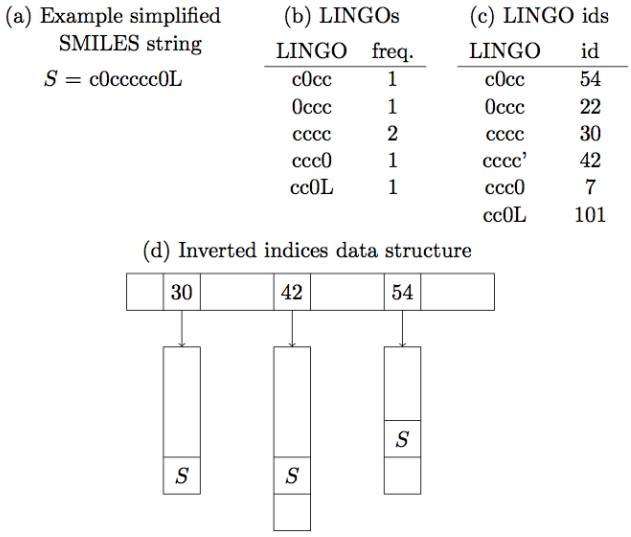

Figure 10. From SMILES strings to inverted index. (a) SMILES string simplified for LINGOsim. (b) LINGOs of example SMILES string. (c) The LINGOs are given ids, with multiple occurrences given unique ids. (d) A reference to the SMILES string S is stored for all the ids of the LINGOs in S.

In [13], we suggest using an inverted index to compute the LINGO intersection size between a query SMILES string A and the entire database quickly. First a preprocessing step is necessary, where each LINGO in the database is given an integer number, such that the first occurrence of a given LINGO in a SMILES string is given a unique id, the second occurrence another one, and so on. This step reduces the problem from multi-sets of LINGOs to ordinary sets of ids. Next we store the database in an inverted index [10], that is, instead of storing a list of LINGOs or ids for each database SMILES string, we store a list of SMILES strings for each LINGO id. See figure 10. Inverted index algorithms also have been applied to speed up fingerprint similarity searches [16].

Now we can compute the LINGO intersection size for the entire database the following way: Create a counter for each database SMILES string. For each LINGO in the query A increase the counter of all database SMILES strings found on the list of that LINGO. When all LINGOs have been processed the counters contain the LINGO intersection sizes, from which the LINGOsims can be computed. This is fast because we only need to visit relevant LINGOs in the database, as opposed to the above methods that always query the entire database. Like above we can parallelize this by processing several molecules at the same time.

For benchmarking we use a method similar to that of [9], computing the LINGOsim similarity of all pairs of fingerprints in the database. We compare our implementation against SIML [9] and the commercial OpenEye implementation. The details of the experiment are described in [13] and the results are summarized in figure 11.

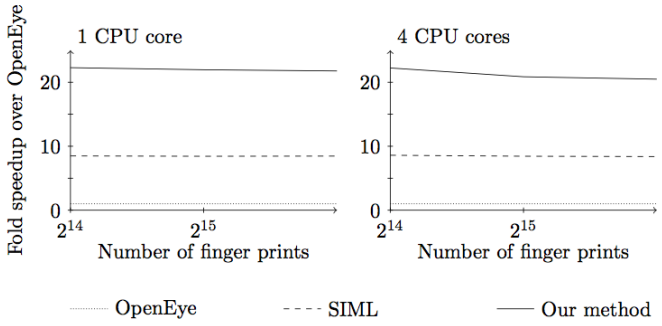

Figure 11. Comparison of our implementation against that of OpenEye and SIML [9], for one and four CPU cores.

Conclusions

In this paper, we have reviewed computationally efficient methods for solving the problem of identifying all molecules stored in database that have a certain similarity to a query molecule. We have considered to problem when molecules were represented by bit-strings, and when molecules were represented by SMILES string. In both cases, the similarity measure used has been the Tanimoto coefficient. The growing size of chemical databases implies a growing need for solutions to this problem that are efficient in practice.

An area for improvement that we have not considered in details is memory usage. Our data structures consume a lot of memory. To store very large molecule databases it might be relevant to create an I/O efficient implementation that stores the data structures on disk in way that can be processed efficiently without reading the entire structure into memory.

Citation
Kristensen TG, Nielsen J, Pedersen CNS (2013) Methods for Similarity-based Virtual Screening. Computational and Structural Biotechnology Journal. 5 (6): e201302009. doi: http://dx.doi.org/10.5936/csbj.201302009

References

1. Aung Z, Ng S-K (2010) An indexing Scheme for Fast and Accurate Chemical Fingerprint Database Searching. In: Proc SSDBM Lect Notes Comput Sc 6187:288-305.
2. Baldi P, Benz RW, Hirschberg DS, Swamidass SJ (2007) Lossless compression of chemical fingerprints using integer entropy codes improves storage and retrieval. J Chem Inf Model 47(6):2098–2109.
3. Baldi P, Hirschberg DS, Nasr R (2008) Speeding up chemical database searches using a proximity filter based on the logical exclusive OR. J Chem Inf Model 48(7):1367–1378.
4. Bolton E, Wang Y, Thiessen PA, Bryant SH (2008) PubChem: Integrated Platform of Small Molecules and Biological Activities. In: Annu Rep Comp Chem, Amer Chem Soc 4:12.
5. Chen X, Reynolds CH (2002) Performance of similarity measures in 2D fragment-based similarity searching: comparison of structural descriptors and similarity coefficients. J Chem Inf Comp Sci 42(6):1407–14.
6. Chen J, Swamidass SJ, Dou Y, Bruand J, Baldi P (2005) ChemDB: a Public Database of Small Molecules and Related Chemoinformatics Resources. Bioinformatics 21, 4133–4139.
7. Fredkin E (1960) Trie Memory. Commun ACM 3(9): 490–499.
8. Grant JA, Haigh JA, Pickup BT, Nicholls A, Sayle RA (2006) Lingos, finite state machines, and fast similarity searching. J Chem Inf Model 46(5):1912–8.
9. Haque IS, Pande VS, Walters WP (2010) SIML: a fast SIMD algorithm for calculating LINGO chemical similarities on GPUs and CPUs. J Chem Inf Model 50(4):560–4.
10. Knuth DE (1998) Sorting and Searching. In: The Art of Computer Programming, vol 3, pp. 560-563.
11. Kristensen TG, Nielsen J, Pedersen CNS (2009). A Tree Based Method for the Rapid Screening of Chemical Fingerprints. In: Proc WABI Lect Notes Comput Sc 5724:194-205.
12. Kristensen TG, Nielsen J, Pedersen CNS (2010). A tree-based method for the rapid screening of chemical fingerprints. Algorithms Mol Biol 5(9).
13. Kristensen TG, Nielsen J, Pedersen CNS (2011). Using Inverted Indices for Accelerating LINGO Calculations. J Chem Inf Model 51(3), 597-600.
14. McGaughey GB et al. (2007) Comparison of topological, shape, and docking methods in virtual screening. J Chem Inf Model 47(4):1504–19.
15. Nasr R, Hirschberg DS, Baldi P (2010) Hashing algorithms and data structures for rapid searches of fingerprint vectors. J Chem Inf Model 50(8), 1358–1368.
16. Nasr R, Vernica R, Li C, Baldi P (2012) Speeding Up Chemical Searches Using the Inverted Index: The Convergence of Chemoinformatics and Text Search Methods. J Chem Inf Model 52:891-900.
17. Smellie A (2009) Compressed binary bit trees: a new data structure for accelerating database searching. J Chem Inf Model 49(2):257–62.
18. Steinbeck C, Han Y, Kuhn S, Horlacher O, Luttmann E, Willighagen E (2003). The chemistry development kit (cdk): An open-source java library for chemo- and bioinformatics. J Chem Inf Comp Sci 43(2):493–500.
19. Swamidass SJ, Baldi P (2007) Bounds and algorithms for fast exact searches of chemical fingerprints in linear and sublinear time. J Chem Inf Modeling 47(2):302–17.
20. Tai D, Fang J (2012). SymDex: Increasing the Efficiency of Chemical Fingerprint Similarity Searches for Comparing Large Chemical Libraries by Using Query Set Indexing. J Chem Inf Comp Sci 52(8):1926-1935.
21. Vidal D, Thormann M, Pons M (2005). LINGO, an efficient holographic text based method to calculate biophysical properties and intermolecular similarities. J Chem Inf Model 45(2):386–93.
22. von Korff, M, Freyss J, Sander T (2009). Comparison of ligand- and structure-based virtual screening on the DUD data set. J Chem Inf Model 49(2):209–31.
23. Weininger D (1988). SMILES, a chemical language and information system. J Chem Inf Comp Sci 28(1):31–36.
24. Willett P (2003). Similarity-based approaches to virtual screening. Biochem Soc T 31:603-606.
25. Willett P (2006). Similarity-based virtual screening using 2D fingerprints. Drug Discov Today, 11(23-24):1046–53.
26. Willett P, Barnard J, Downs G (1998) Chemical Similarity Searching. J Chem Inf Model 38(6):983–996.

Keywords:
Algorithms, Virtual Screening, Fingerprints, SMILES, LINGOsim

Competing Interests:
The authors have declared that no competing interests exist.