JOURNAL OF CHEMICAL INFORMATION AND MODELING

ARTICLE

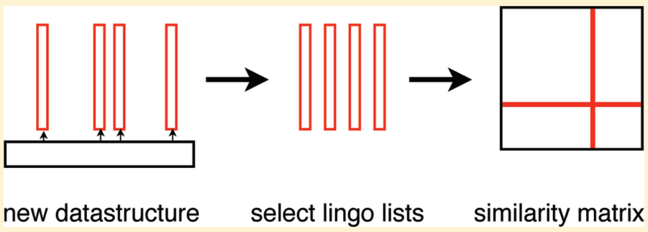
pubs.acs.org/jcim

Using Inverted Indices for Accelerating LINGO Calculations

Thomas G. Kristensen,* Jesper Nielsen,* and Christian N. S. Pedersen*

Bioinformatics Research Centre, Aarhus University, C.F. Møllers Allé 8, 8000 Aarhus C., Denmark

ABSTRACT: The ever growing size of chemical databases calls for the development of novel methods for representing and comparing molecules. One such method called LINGO is based on fragmenting the SMILES string representation of molecules. Comparison of molecules can then be performed by calculating the Tanimoto coefficient, which is called LINGOsim when used on LINGO multisets. This paper introduces a verbose representation for storing LINGO multisets, which makes it possible to transform them into sparse fingerprints such that fingerprint data structures and algorithms can be used to accelerate queries. The previous best method for rapidly calculating the LINGOsim similarity matrix required specialized hardware to yield a significant speedup over existing methods. By representing LINGO multisets in the verbose representation and using inverted indices, it is possible to calculate LINGOsim similarity matrices roughly 2.6 times faster than existing methods without relying on specialized hardware.



INTRODUCTION

A common task in drug discovery is the computational analysis of chemical compounds, which can take the form of, for example, predicting numerical properties such as the log P value or performing screening studies in which new drug candidates are sought from a large database of available molecules. Both problems are often managed by representing molecules as either graphs or 3-D structures. The number of available molecules is increasing rapidly: the ZINC database contains more than 13 million molecules[1] and GDB-13, containing all synthesizable molecules up to size 13, contains 970 million chemical compounds.[2] Therefore, novel vector models such as fingerprints and numerical descriptors have been proposed and tested as predictors and screening tools. Several studies have examined the effectiveness of these methods,[3,4] and other studies have already examined the acceleration of queries into molecular databases represented as vectors.[5−9] If a database is stored as canonical SMILES strings, both fingerprints and numerical descriptors require an explicit model of the chemical structure as a graph or as a 3-D structure to be constructed. The LINGO representation[10] avoids this problem by generating a representation of a molecule directly from canonical SMILES strings[11] of the molecules.

Given a canonical SMILES string for a molecule, the set of LINGOs in this study is generated by replacing all ring closure numbers with zero and all occurrences of Br and Cl with R and L, respectively. The LINGOs are defined as all substrings of size q in the resulting string. Panels (a), (b), and (c) of Figure 1 illustrate the generation of the LINGOs for a small molecule with $q = 4$, as this value of q is suggested as optimal by two previous empirical studies.[10,12] In the article introducing LINGOs, the model was used to predict ADME (absorption, distribution, metabolism, and excretion) properties with an r^2 correlation of 0.93 to experimentally observed values of log P.[10] The study furthermore compared the intermolecular similarity between bioisosteric molecules with that of randomly sampled pairs of molecules using the LINGOsim measure, finding a large discriminatory power.

PREVIOUS WORK

A multiset is a set where elements are allowed to occur multiple times. Given two multisets of LINGOs, the LINGOsim is defined as the Tanimoto coefficient between the sets. If $A = \{$ "cccc", "c0 cm^3", "cccc", "cccc" $\}$ and $B = \{$ "cScc", "cccc", "cccc" $\}$ then

$$\text{LINGOsim}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

$$= \frac{|\{\text{"cccc"}, \text{"cccc"}\}|}{|\{\text{"cccc"}, \text{"c0cc"}, \text{"cccc"}, \text{"cccc"}, \text{"cScc"}\}|} = \frac{2}{5}$$

Grant et al. used the LINGOsim as a ranking tool in a screening study in which it performed comparable to Daylight fingerprints.[12]

Whereas most previous work concerning fingerprints and numerical descriptors has been focused on accelerating queries into large databases,[5−8] most of the work concerning LINGOs has been focused on the calculation of LINGOsim between pairs of molecules. The first fast LINGOsim method was based on constructing a finite state machine (FSM) based on the transformed SMILES string in panel (b) of Figure 1. In the initial study of this FSM, it was described as being constructed as repeatedly inserting LINGOs into a trie,[12] which can be done in linear time.[13]

A less complicated method called SIML is based on encoding the frequency table from panel (c) of Figure 1 as two word-arrays: one containing the codes for the LINGOs and one for

Received: November 4, 2010
Published: February 18, 2011

S = c1ccccc1Cl
(a) Example SMILES

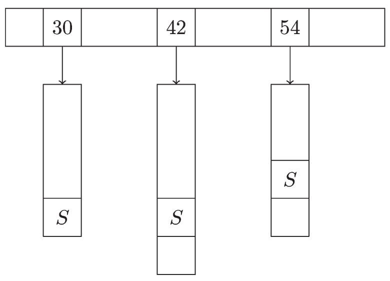
S' = c0ccccc0L
(b) Simplified string

LINGO	freq.
c0cc	1
0ccc	1
cccc	2
ccc0	1
cc0L	1

(c) LINGOs

LINGO	id
c0cc	54
0ccc	22
cccc	30
cccc'	42
ccc0	7
cc0L	101

(d) Verbose rep.



(e) Inverted indices rep.

Figure 1. Example of generating LINGO multisets in the verbose and inverted indices representation. Inverted index lists contain references to the original SMILES string S.

encoding their frequency.[14] One study suggests that a size of four is optimal,[12] and as every ASCII character can be encoded in 8 bit, a LINGO of size four can be encoded in 32 bits, which is precisely a word on common hardware. If the lists are sorted by LINGO code, the LINGOsim can be calculated by a parallel run through two pairs of lists, which is linear in the number of distinct LINGOs in the two strings. The SIML study used SIMD instructions and GPU hardware and compared its performance to that of a commercial implementation by OpenEye with speedups on the order of a factor of 80. It is, however, limited to a q of four or less on 32-bit hardware and requires a very fast GPU to obtain a significant speedup.

VERBOSE REPRESENTATION

A problem with the SIML encoding scheme is that it is bound to both q and the word size of the underlying hardware. Instead of encoding each character of the LINGO, we propose generating ids for each LINGO in the SMILES strings as they are observed when reading the strings. Multiple occurrences of the same LINGO in one SMILES string is given different ids as illustrated in panel (d) of Figure 1. Generating these ids is done by using a trie as in the FSM method, storing ids in the leaves. Inserting LINGOs in the trie is done in linear time, and the entire set of SMILES strings is inserted in linear time. If each id is stored in one word, the verbose representation can store more different LINGOs than the SIML encoding in the same word size. This is because nonoccurring LINGOs are not stored, in contrast to SIML that also encodes impossible LINGOs such as "h43f" or "c(]S". The SIML representation uses two arrays for representing a LINGO multiset, but the multisets can be represented more verbosely with only one array with multiple occurrences of the same LINGO represented as multiple different ids, as illustrated with the two "cccc" LINGOs in panel (d) of Figure 1. This verbose representation has the downside that it will potentially take up more space: the SMILES string "cccccccc" would take up two words in SIML but five in the verbose representation. However, analyzing the test data from Maybridge presented in the Experiments section reveals that the average SMILES length is 35.2 (yielding 64.4 words in SIML), whereas the average

Table 1. Frequencies of Size Four LINGOs in the Maybridge and ZINC Data

times LINGOs repeated	Maybridge	ZINC
	avg. per molecule	
1	26.353	42.793
2	2.434	8.391
3	0.159	3.352
more than 4	0.000	9.001

Table 2. Average Word Consumption of Size Four LINGO Multisets in the SIML and Verbose Representation on the Maybridge and ZINC Data[a]

encoding	Maybridge	ZINC
SIML rep.	64.4	127.1
verbose rep.	31.8	155.4

[a] The ZINC data is generated using CDK that adds explicit hydrogen information, which accounts for them being larger than the Maybridge data.

number of ids in the verbose representation is only 31.8 as demonstrated in Table 2. The ZINC data has slightly longer SMILES strings with more repetitions (Table 1), and the SIML representation is slightly shorter than the verbose representation. If 4 byte integers are used for representing the ids, the verbose representation takes up less than four times the memory of the original SMILES strings.

The presented verbose representation can be interpreted as sparse fingerprints, in which case the LINGOsim is the Tanimoto coefficient between fingerprints. This implies that all data structures developed for effectively performing queries into fingerprint databases[5−9] can be used on LINGOsim queries into databases of LINGO multisets.

INVERTED INDICES METHOD

This paper proposes calculating the LINGOsim between a target LINGO multiset and a database of LINGO multisets by storing the database as *inverted indices*. Inverted indices are also used to handle the T-occurrence problem from string algorithms: given a query string and a database of strings, identify all database strings that share more than T substrings of size q with the query.[15,16] The idea is to store the LINGO multisets as a vector, where each cell represents one of the LINGO ids from the verbose representation. A database of n multisets $x_1,...,x_n$ can be stored as a vector I, where each cell I_k stores a list of all the multisets containing k, as illustrated in panel (e) of Figure 1. The LINGOsim between a multiset x_i and every other multiset x_j in the database can be calculated by first calculating the intersection sizes between x_i and every x_j. To do this, let C be a counting vector of length n initialized with all zeros and let x_i contain the m ids $x_{i,1},...,x_{i,m}$. Next, traverse the inverted indices lists $I_{x_{i,1}},...,I_{x_{i,m}}$ and increment the counter C_{x_j} every time x_j is observed. An example of this is shown in Figure 2. Afterward C_{x_j} will contain $|x_i \cap x_j|$, from which the LINGOsim can be calculated by using $|x_i \cup x_j| = |x_i| + |x_j| - |x_i \cap x_j|$. Note that this strategy only works if the verbose representation contains unique ids for multiple occurrences of LINGOs within the same multisets as they would otherwise be counted multiple times when the inverted indices lists are traversed. The inverted indices data structure is not

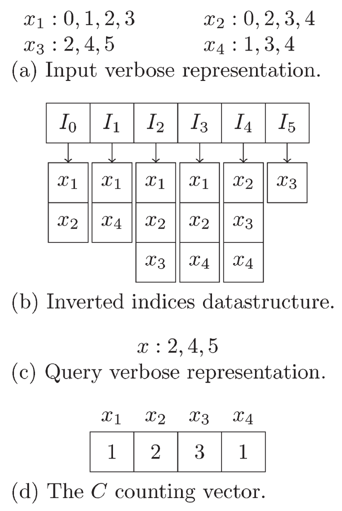

$x_1: 0,1,2,3$ $x_2: 0,2,3,4$
$x_3: 2,4,5$ $x_4: 1,3,4$
(a) Input verbose representation.
I_0 I_1 I_2 I_3 I_4 I_5
(b) Inverted indices datastructure.
$x: 2,4,5$
(c) Query verbose representation.
x_1 x_2 x_3 x_4
1 2 3 1
(d) The C counting vector.

Figure 2. Example of the data structure and algorithm. Input (a) is used to build an inverted indices table (b), which lists all molecules associated with a given id. To perform a query with a molecule x (c), the counting vector C is created by running through list I_2, I_4, and I_5 in the inverted indices table and increasing the entries corresponding to the encountered molecules (d). From this vector the similarities can be computed.

limited to LINGO multisets but can also be used to calculate the Tanimoto coefficients for general fingerprints.

The inverted indices are constructed by first identifying the largest id in the data set. This id is used to find the size of I so that this can be allocated. Next, run through all the LINGO multisets x_i and insert them into the lists $I_{x_{i,1}},...,I_{x_{i,m}}$. All this takes linear time and takes up memory linear in the size of the verbose representation.

EXPERIMENTS

Experiments were performed on data from Maybridge, taken from the SIML study and on data generated using the Chemistry Development Kit[17] canonical SMILES generator on molecules from the ZINC database version 8, subset 10.[1] The SIML data contains 4096 SMILES strings with an average length of 35.2, and the ZINC data contains the first 65536 SMILES strings from ZINC with an average length of 158.4. LINGOs were generated for $q = 4$ as in previous studies using the method from the original paper.[10] Three methods were evaluated: the LINGOsim calculator from OpenEye[18] (OE), the SIML implementation without GPU support, and a C implementation of our inverted indices method (IIM). The OpenEye LINGOsim calculator was chosen because it is used elsewhere in the literature.[10] It is not FSM based. Each method was measured as in the SIML study, that is, on the time it takes to fill out a similarity matrix for a set of LINGO multisets, not including the time to load the LINGOs into memory or outputting the similarity matrix to disk or screen. As the matrix is symmetric, it is only necessary to compute half the entries. However, the matrix in our experiments is too big to fit in memory, and it is therefore streamed, which means that the old values cannot be read, and every entry has to be computed explicitly by all three methods. The resulting matrix can be used for performing statistical calculations, clustering, or similar data analysis.

All experiments were performed on an Intel Core 2 Quad Q9450 2.66 GHz machine with 4 GB of RAM running Ubuntu version 9.04 with GCC 4.3.3 using the −O3 optimization flag. Presented results are from a single experiments.

Table 3. Results from Experiments Replicated from Haque et al.[14] on the Maybridge Data Containing 4096 SMILES Strings, along with the Results of Using the IIM Method[a]

	Haque et al.[14]	replicated
algorithm	time (ms)	time (ms)
OE	15060	14860
SIML	5460	5421
OE 4 cores	3880	3860
SIML 4 cores	1420	1435
SIML GPU (GeForce GTS 250)	270	–
SIML GPU (Tesla T10)	215	–
IIM	–	407
IIM 4 cores	–	125

[a] The GPU running times were not replicated as we did not have access to the same hardware.

Table 4. Timing Results in Seconds from Running the Implementations on the CDK Generated SMILES Strings on the ZINC Database

	OE	OE	SIML	SIML	IIM	IIM
SMILES	1 core	4 cores	1 core	4 cores	1 core	4 cores
16384	1493	378	176	44	67	17
32768	5928	1502	703	178	270	72
65536	23791	6022	2812	718	1093	294

RESULTS

As presented in Table 3, rerunning the OE and SIML implementations on Maybridge benchmark data yields results very similar to those in the original study. The data in the first column is taken from Table 2 and Table 3 in the SIML article[14]; the GPU measurements are not from the same machine as the first four rows. As shown from Table 3, the IIM method is faster than all the tested methods, even those using multiple cores and GPUs. For one core, the speedup compared to the OpenEye implementation is of a factor of 37 and drops to 31 when the implementations use all four cores. Compared to the SIML implementation, the speedup is of a factor of 13 when both implementations are run on one core and 11 when the implementations are allowed to use all four cores. In all cases, the IIM method outperforms the previous methods. Surprisingly, the one core version is faster than all but the GPU implementations.

Running the implementations on the ZINC data set yields the running times presented in Table 4. There is a large difference in the length of the SMILES strings between the two data sets, and the observations from the ZINC set is therefore a bit slower than that of the Maybridge set and gives rise to different observations in speedup. For one core, the speedup between OE and the IIM method drops to a steady factor of 22 across the tested database sizes, and the speedup between the SIML implementation and the IIM method drops to a factor of 2.6. The speedups remain the same when using all four cores.

Experiments not presented here were performed to examine if the data size or the data structure is responsible for the decline in speedup. Experimental data was generated by transforming the Maybridge data, extending the lines using concatenation to match the length of the lines in the ZINC data. The ZINC lines were shortened to match those of the Maybridge data. The experiments revealed that both factors contributed equally to the deterioration of the speedup.

There is also a slight difference in the achieved speedup on four cores when going from the Maybridge to the ZINC data. On the Maybridge data, the IIM achieves 81% of linear speedup, while 93% of the achievable speedup is gained on the ZINC data (using all 65.536 SMILES strings). The previously mentioned experiments revealed that exactly half of the improvement was a result of the ZINC data containing longer SMILES strings. The other half was a result of the structure of the ZINC data, which contains explicit hydrogen, yielding an increased number of shared LINGOs.

The presented tables do not include time to parse the SMILES files. Initial studies revealed that parsing files containing SMILES strings and converting them to the verbose representation takes less than 20% of the total running time on the Maybridge data and less than 1% on the ZINC data. Converting this data to the inverted indices representation is included in the presented execution times but accounts for less than 1% of the time spent in the algorithm.

CONCLUSION

The number of available molecules is ever increasing, and new methods are needed to handle the large chemical databases. This paper presents a reduction from LINGO multisets to sparse fingerprints, making it possible to implement methods for performing rapid queries in molecular databases with the LINGO-sim similarity measure by using the Tanimoto coefficient in fingerprint databases.

This paper also presents the inverted indices method for storing LINGO multisets along with a method for rapidly calculating the similarity matrix for such a collection. The presented algorithm has been implemented and tested on standard hardware and was observed to be more efficient than other current methods, outperforming them in all tests. The SIML method tested against in this study was designed for a *q* of four, whereas the verbose representation is independent of *q*. The presented method makes it possible to analyze very large data sets without the need for GPUs or other types of specialized hardware. The tested implementation along with the test data is available at http://birc.au.dk/~tgk/ii.

There are two interesting directions for future research, namely, statistical analysis of very large data sets using the inverted indices method and acceleration of queries into large databases by using the reduction to sparse fingerprint presented in this paper.

AUTHOR INFORMATION

Corresponding Author

*E-mails: tgk@birc.au.dk (T.G.K.); jn@birc.au.dk (J.N.); cstorm@birc.au.dk (C.N.S.P.).

ACKNOWLEDGMENT

The authors thank Imran Haque for making the SIML implementation along with the SIML test data available and Open Eye for an academic license to the OEChem TK.

REFERENCES

(1) Irwin, J. J.; Shoichet, B. K. ZINC: A free database of commercially available compounds for virtual screening. *J. Chem. Inf. Model.* **2005**, *45*, 177–182.

(2) Blum, L. C.; Reymond, J.-L. 970 million druglike small molecules for virtual screening in the chemical universe database GDB-13. *J. Am. Chem. Soc.* **2009**, *131*, 8732–8733.

(3) von Korff, M.; Freyss, J.; Sander, T. Comparison of ligand- and structure-based virtual screening on the DUD data set. *J. Chem. Inf. Model.* **2009**, *49*, 209–231.

(4) McGaughey, G. B.; Sheridan, R. P.; Bayly, C. I.; Culberson, J. C.; Kreatsoulas, C.; Lindsley, S.; Maiorov, V.; Truchon, J.-F.; Cornell, W. D. Comparison of topological, shape, and docking methods in virtual screening. *J. Chem. Inf. Model.* **2007**, *47*, 1504–1519.

(5) Swamidass, S. J.; Baldi, P. Bounds and algorithms for fast exact searches of chemical fingerprints in linear and sublinear time. *J. Chem. Inf. Model.* **2007**, *47*, 302–317.

(6) Baldi, P.; Hirschberg, D. S.; Nasr, R. J. Speeding up chemical database searches using a proximity filter based on the logical exclusive OR. *J. Chem. Inf. Model.* **2010**, *48*, 1367–1378.

(7) Smellie, A. Compressed binary bit trees: A New data structure for accelerating database searching. *J. Chem. Inf. Model.* **2009**, *49*, 257–262.

(8) Kristensen, T. G.; Nielsen, J.; Pedersen, C. N. S. A tree-based method for the rapid screening of chemical fingerprints. *Algorithms Mol. Biol.* **2010**, *5*, 9:1–10.

(9) Kristensen, T. G. Transforming Tanimoto queries on real valued vectors to range queries in Euclidian space. *J. Math. Chem.* **2010**, *48*, 287–289.

(10) Vidal, D.; Thormann, M.; Pons, M. LINGO: An efficient holographic text-based method to calculate biophysical properties and intermolecular similarities. *J. Chem. Inf. Model.* **2005**, *45*, 386–393.

(11) Weininger, D. SMILES: A chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Comput. Sci.* **1988**, *28*, 31–36.

(12) Grant, J. A.; Haigh, J. A.; Pickup, B. T.; Nicholls, A.; Sayle, R. A. Lingos, finite state machines, and fast similarity searching. *J. Chem. Inf. Model.* **2006**, *46*, 1912–1918.

(13) Aho, A. V.; Corasick, M. J. Efficient string matching: An aid to bibliographic search. *Commun. ACM* **1975**, *18*, 333–340.

(14) Haque, I. S.; Pande, V. S.; Walters, W. P. SIML: A fast SIMD algorithm for calculating LINGO chemical similarities on GPUs and CPUs. *J. Chem. Inf. Model.* **2010**, *50*, 560–564.

(15) Sarawagi, S.; Kirpal, A. Efficient Set Joins on Similarity Predicates, ACM SIGMOD Conference, Paris, June 13−18, 2004.

(16) Li, C.; Lu, J.; Lu, Y. Efficient Merging and Filtering Algorithms for Approximate String Searches, Proceedings of the 2008 IEEE 24th International Conference on Data Engineering, Cancún, Mexico, April 7−12, 2008.

(17) Steinbeck, C.; Han, Y.; Kuhn, S.; Horlacher, O.; Luttmann, E.; Willighagen, E. The Chemistry Development Kit (CDK): An open-source java library for chemo- and bioinformatics. *J. Chem. Inf. Comput. Sci.* **2003**, *43*, 493–500.

(18) *OEChemTK − C++*, version 1.7.4; OpenEye Scientific Software: Santa Fe, NM, 2010.