JOURNAL OF CHEMICAL INFORMATION AND MODELING

Application Note

pubs.acs.org/jcim

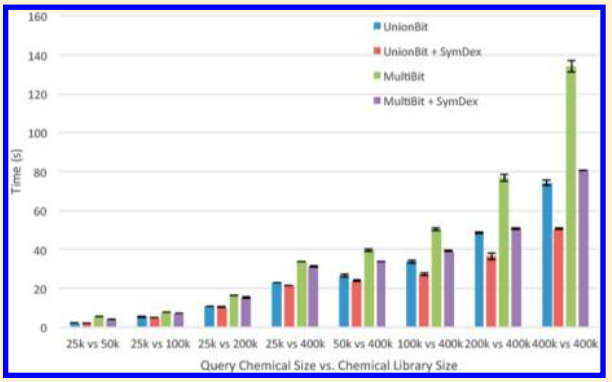
ChemCom: A Software Program for Searching and Comparing Chemical Libraries

Sirus Saeedipour,[‡] David Tai,[‡] and Jianwen Fang*[,†,‡]

[†]Biometric Research Branch, Division of Cancer Treatment and Diagnosis, National Cancer Institute, 9609 Medical Center Dr., Rockville, Maryland 20850, United States

[‡]Applied Bioinformatics Laboratory, The University of Kansas, 2034 Becker Drive, Lawrence, Kansas 66047, United States

ABSTRACT: An efficient chemical comparator, a computer application facilitating searching and comparing chemical libraries, is useful in drug discovery and other relevant areas. The need for an efficient and user-friendly chemical comparator prompted us to develop ChemCom (Chemical Comparator) based on Java Web Start (JavaWS) technology. ChemCom provides a user-friendly graphical interface to a number of fast algorithms including a novel algorithm termed UnionBit Tree Algorithm. It utilizes an intuitive stepwise mechanism for selecting chemical comparison parameters before starting the comparison process. UnionBit has shown approximately an 165% speedup on average compared to its closest competitive algorithm implemented in ChemCom over real data. It is approximately 11 times faster than the Open Babel FastSearch algorithm in our tests. ChemCom can be accessed free-of-charge via a user-friendly website at http://bioinformatics.org/chemcom/.



1. BACKGROUND

A chemical comparator, a computer application facilitating similarity-based searches and comparisons of chemical libraries, has many practical applications. For example, it is necessary to search hit compounds against specialized libraries to determine whether they are patentable. As the screening data deposited in PubChem accumulates rapidly, it has become increasingly useful to compare hit compounds against PubChem libraries to predict whether they are biologically active toward other drug targets. For library comparisons, a high throughput screening (HTS) center may need to compare its current collection to a vendor library before a decision can be made whether to acquire the whole or a part of the library. An all-against-all comparison of a chemical library is necessary to investigate its diversity and covered chemical space. Two HTS facilities may want to compare their collections of chemicals before forming collaboration. As more HTS facilities are established and the number of available chemicals increases rapidly, performing large-scale searches and comparisons are becoming even more common and important.

Numerous methods have been developed to help speed up searches against chemical libraries.[1−8] Chemical fingerprinting, for example, is a common method of simplifying chemical representation by describing the structural properties of a chemical as a one-dimensional feature string. However, searching a large number of query structures or especially when comparing two large chemical libraries may still take considerable investment in time and computing power.

Recently, we developed SymDex,[7] which was built upon existing research efforts by utilizing a novel strategy to achieve a speedup upon existing chemical search algorithms. SymDex indexes the set of all query chemicals so redundant calculations that arise in the case of sequential searches are eliminated. SymDex, however, like most other software programs for the purposes, is a command line application, and therefore, it is not particularly user friendly. This situation prompted the need for producing an application using a graphical, user-friendly interface.

In this report, we present ChemCom (**Chem**ical **Com**parator), a software program for searching and comparing large scale chemical databases. ChemCom has been developed in Java. In addition to three existing algorithms, we have proposed and implemented a novel algorithm, UnionBit, in ChemCom. UnionBit is an algorithm designed to improve upon the speed of multibit tree[2] by creating Tanimoto similarity calculations that achieve better pruning per evaluation run.

2. IMPLEMENTATION

Other Algorithms. To understand the inner workings of the UnionBit tree algorithm requires knowledge about two previous algorithms: *bit bound*[6] and *multibit tree*.[2] Bit bound indexing bins every bit string in a library into a collection of bins (usually implemented as an array or hashmap) by the popcount or number of 1s (trues) in the bit string. To search a

Received: November 25, 2014
Published: June 12, 2015

DOI: 10.1021/ci500713s
J. Chem. Inf. Model. 2015, 55, 1292−1296

bit bound index for similarity to a candidate chemical, a pruning heuristic derived from the Tanimoto similarity equation (Figure 1) prunes most of the bins in the index. It does this by

$$Tanimoto\ Similarity = \frac{|A \cap B|}{|A \cup B|}$$

Observe that:

$$\min(|A|, |B|) \geq |A \cap B|$$

$$\max(|A|, |B|) \leq |A \cup B|$$

Thus:

$$Tanimoto\ Similarity \leq \frac{\min(|A|, |B|)}{\max(|A|, |B|)}$$

Figure 1. Derivation of Tanimoto similarity heuristic for bit bound algorithm where A and B are 2 bit strings being compared.

analyzing the maximum similarity calculated by the pruning heuristic between the candidate and all bit strings in a bin based on popcount of candidate and the popcount stored on each bin that summarizes the contents of each bin. Afterward, all bit strings in each unpruned bin are compared.

Multibit tree indexing was built upon bit bound by replacing simple bit string lists in the bins of bit bound with pruning trees to further minimize the number of strings compared. A multibit pruning tree is defined by four properties:

(1) Bit strings are stored in the leaves of the tree.

(2) Any subtree in the pruning tree defined by a node in the pruning tree stores in this node the position and value of the set of bits that are the same across all bit strings in the subtree's leaves.

For example, let there be a node N with two leaves L1 and L2 each containing a single bit string [0,1,0,1,0] and [1,1,1,1,0], respectively. N would store the set of ordered pairs of position and value for L1 and L2, which are [(2, 1), (4, 1), (5, 0)].

More efficient implementations would reduce the amount of calculations by storing a bit string for intersection and union operations and a bit mask just for union operations.

(3) Any subtree in the pruning tree defined by a node in the pruning tree also stores in this node the position of the bit used to partition the bit strings in the leaves of the left and right subtrees. Given a partition bit of position 1, the bit strings stored on the leaves of the left subtree would look like [0, 0] and [0, 1] and the bit strings stored on the leaves of the right subtree would look like [1, 0] and [1, 1]. The choice of this splitting bit depends upon implementation.

(4) The depth of a pruning tree cannot exceed some maximum depth. Otherwise, the time gains of pruning more bit strings is lost to spending more time pruning. The best maximum depth for a given bit string is a function of the length and features checked of each bit string generation algorithm.

To search a multibit tree for similarity with a candidate chemical, first, bit bound is used to prune and then the pruning tree stored in each bin is used to prune. The pruning algorithm of multibit pruning trees calculates the exact Tanimoto similarity with the bit information stored at each node and combines it with the same heuristic used by bit bound. Thus, as more bit information is gained deeper in the tree, the best case similarity is reduced and subtrees can be pruned. Afterward, all remaining chemical bit strings stored in unpruned leaves are then compared.

UnionBit Tree Algorithm. UnionBit tree is a novel algorithm designed to improve upon the speed of multibit tree.[2] Speeding up chemical similarity searches relies heavily on heuristics used to reduce unnecessary comparisons based on the knowledge of common characteristics of the candidate chemical and the set of library chemicals—thereby saving time. In the case of multibit tree and UnionBit tree, the heuristics used are more complex because both iteratively add information to the heuristics. UnionBit tree improves over multibit tree because it approaches this knowledge gaining iteration by walking the search tree in a different way. Speed improvements result from deriving a Tanimoto similarity calculation (Figure 2) that achieves better pruning speeds compared to multibit tree at the trade-off of having deeper trees.

$$similarity = \frac{|A \cap B|}{|A \cup B|}$$

$$|A \cup B| = |A| + |B| - |A \cap B|$$

B' is the union of B with n feature sets each with |B| features which we refer to as the union-bits

Therefore:

$$|A \cap B| \leq |A \cap B'|$$

$$similarity = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \leq \frac{|A \cap B'|}{|A| + |B| - |A \cap B'|}$$

Figure 2. Derivation of union-bit Tanimoto similarity heuristic where B′ is the union-bits of library bits of bit count |B|.

Like multibit tree, UnionBit relies on bit bound as a first step and then creates a tree for every bin of bit bound. It also shares three out of four properties with multibit trees only differing in the second rule:

- Any subtree in the pruning tree defined by a node in the pruning tree stores in this node the union of all bit strings, or *union-bits*, stored in the leaves of this subtree.

For example, let there be a node N with two leaves L1 and L2 each containing a single bit string [0,1,0,1,0] and [1,1,1,1,0], respectively. N would store the union-bits [1,1,1,1,0].

Instead of storing only common bits of a subtree, UnionBit Tree unions all bit strings in the subtree together to create a set of *union-bits*. Because union is a purely additive function, the result of the intersection of the bit string A with the union of the library B′ must be greater than or equal to that of A with any single member of the library. At each iteration, the library of chemicals is subdivided by selecting a bit position and creating 2 bins, one with all strings where the bit is 0 and one with all strings where the bit is 1. Subdividing the library reduces the cardinality of the set of union-bits in each sublibrary. Compared to multibit tree, the UnionBit Tree uses both less CPU time and memory:

Calculation Complexity. UnionBit Tree performs a single calculation $|A \cap B|$ at each node when pruning for a bit string:

(1) 1 logical AND per 64 bits to calculate the intersection of bit string with the union-bits

(2) 1 popcount per 64 bits to count the results of 1

DOI: 10.1021/ci500713s
J. Chem. Inf. Model. 2015, 55, 1292−1296

(3) 1 addition per 64 bits minus 1 to sum the popcounts from 2
(4) Some constant number of arithmetic operations to calculate the similarity metric

Compared to multibit tree of the same depth in the most efficient implementation requires at each node when pruning for a bit string:

(1) 1 logical AND per 64 bits to calculate the intersection with the common bits of the subtree
(2) 1 popcount per 64 bits to count the results of 1
(3) 1 addition per 64 bits minus 1 to sum the popcounts from 2
(4) 1 logical AND per 64 bits to calculate the intersection with the masking bits
(5) 1 logical OR per 64 bits to calculate the union using the result step 2 and the common bits
(6) 1 popcount per 64 bits to count the results of 5
(7) 1 addition per 64 bits minus 1 to sum the popcounts from 6
(8) Some constant number of arithmetic operations to calculate the similarity metric

Because fingerprints can be very large, the constant cost of calculating the similarity metric can be ignored in most realistic cases so UnionBit Tree costs can be expressed as $c_u(x) = 3x - 1$, and multibit tree can be expressed as $c_m(x) = 7x - 2$. If the constants are ignored for the same reason as the other constant costs, the ratio $c_u(x)/c_m(x)$ becomes $3x/7x$, which simplifies into 3/7. Thus, in the case of very large fingerprints, UnionBit tree does only about 43% of the calculations as a multibit tree.

Memory Space. UnionBit tree stores a single string of union-bits. Multibit tree requires storing both a mask bit string and the common bit string. UnionBit tree therefore uses half the memory space of multibit tree if one were to use a large array to represent these binary trees.

ChemCom. ChemCom is implemented as a Java application and provided through the JavaWS (https://en.wikipedia.org/wiki/Java_Web_Start) framework. The Java programming language provides many benefits including cross-platform compatibility as well as robust memory management. Moreover, the JavaWS framework has several additional advantages such as seamless version updating over conventional methods.

Open Babel,[9] an open source chemical toolbox, is used to calculate the fingerprints of compared chemical libraries. Open Babel supports Windows, LINUX, and Macintosh. An official version can be downloaded from Open Babel's website (http://openbabel.org/wiki/Get_Open_Babel). ChemCom uses the Tanimoto similarity score, one of the most widely used metrics for measuring chemical similarity.

To perform similarity searches, a query fingerprint must first be indexed by hashing. Once indexed, the fingerprints are binned by their hash keys. Performing searches over the tree structure used for the database search index is done by traversing the tree structure in a depth first fashion.

We implement four algorithms and their SymDex enhanced versions, three of which were originally implemented in SymDex—*bit bound*, *multibit tree*, and *kD Grid*. With the development of ChemCom, a fourth and innovative algorithm *UnionBit*, as described in the previous section, is added.

For all implemented algorithms, an indexed database can be saved as a binary file (termed chemcom file). All defined search options and parameters are kept in the file. This file can be reused for future searches and also shared with others who want to search the same database. It should be pointed out that parameters can no longer be adjusted if a saved indexed database is used.

In ChemCom, searching algorithms are separated from the running environment and each algorithm is implemented as a separated Java class with a shared interface. New algorithms can be implemented as a new class by using the existing algorithms as templates.

3. RESULTS AND DISCUSSION

Benchmarking. Chemicals for benchmarking the performance of the algorithms in ChemCom were randomly selected from the first one million chemicals in the PubChem database.[10] This procedure was repeated five times for each query-database size combination. All experiments were performed on an Intel i7-2600K (quad cores with eight threads) with 16 GB of memory. All algorithms in ChemCom were implemented using Java using a consistent object oriented pattern. Tests were run with a virtual heap size starting at 12 GB with a maximum amount of 12 GB and using the −Xincgc to enable continuous parallel garbage collection on a separate process.

Figure 3 shows the times for the algorithms implemented in ChemCom to compare one 25,000 chemical library against a

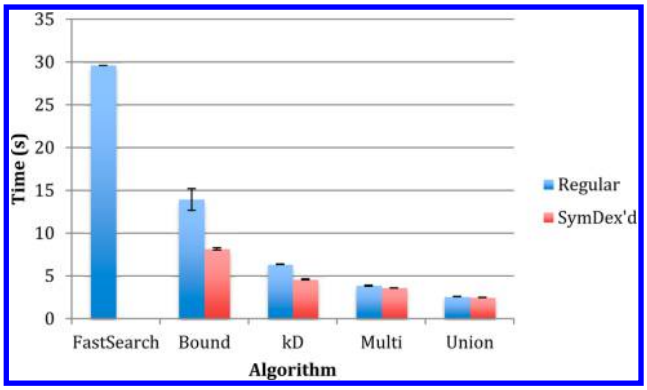


Figure 3. Means and standard deviations (error bars) of the time in seconds for searching 25,000 query chemicals against a 50,000 chemical library. The Open Babel FastSearch was written in C++, while ChemCom algorithms were in Java.

50,000 chemical library. As expected, "SymDex'd" searches performed comparisons in less time when compared with each respective algorithm's sequential querying counterparts. SymDex only achieves negligible improvement for UnionBit (2.58 vs 2.48 s for regular and SymDex'd versions, respectively. As a reference, we perform the same search using the Open Babel FastSearch algorithm. The regular UnionBit is approximately 11 times faster than Open Babel FastSearch algorithm (2.58 vs 29.6 s, respectively) in this test. It is noteworthy that Open Babel is written in C++, while ChemCom is developed in Java. Thus, the real improvement is likely more pronounced because C++ is generally faster than Java.

We further compared the performance of UnionBit and its closest competitive algorithm (MultiBit) using various sizes of query and databases (Figure 4). On average, UnionBit is 65% and 52% faster than MultiBit for the regular and SymDex'd versions, respectively. The SymDex speedup for UnionBit is more pronounced when the size of query data set is scaled up. For example, the times need for comparing 400,000 chemical library against another in the same size are 74.2 vs 50.7 s for

DOI: 10.1021/ci500713s
J. Chem. Inf. Model. 2015, 55, 1292−1296

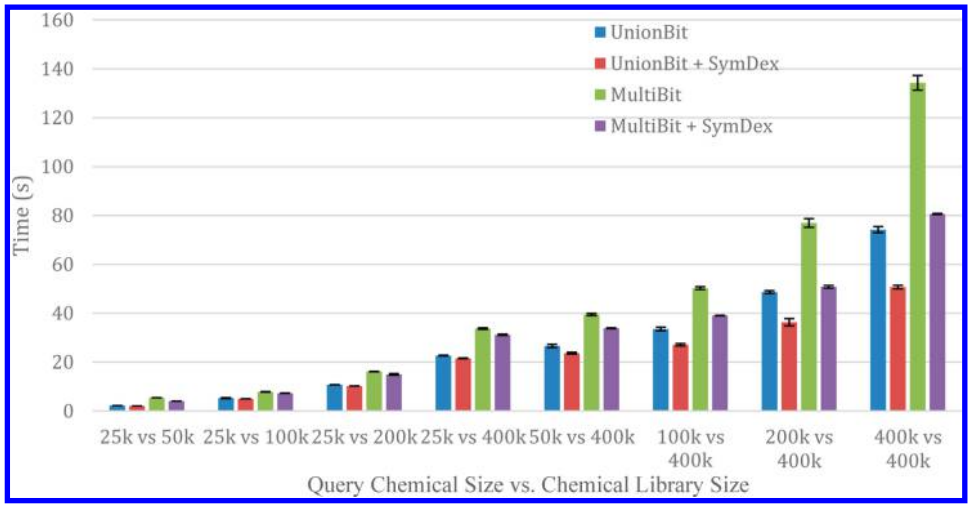


Figure 4. Means and standard deviations (error bars) of the time in seconds for searching 25,000 query chemicals against increasing sizes (50,000, 100,000, 200,000, and 400,000) of chemical libraries.

regular and SymDex'd versions, respectively, a 146% speedup (Figure 4).

We also compared UnionBit and MultiBit using various Tanimoto thresholds (Figure 5). In all cases, UnionBit was faster than MultiBit and SymDex'd version achieved speedup at various degrees.

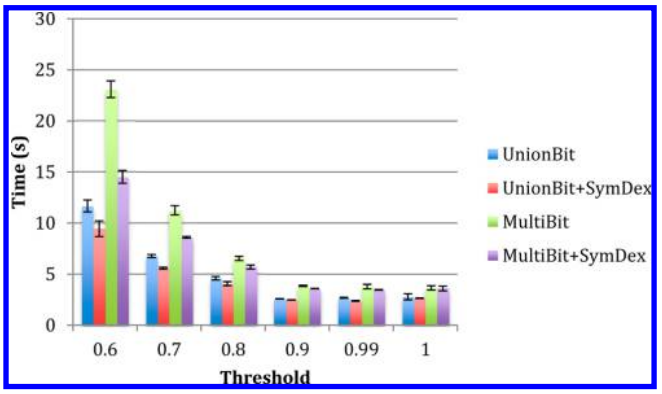


Figure 5. Means and standard deviations (error bars) of the time in seconds for searching 25,000 query chemicals against a 50,000 chemical library at different Tanimoto threshold.

Sample Run. To execute searches and comparisons in ChemCom is rather straightforward, and detailed instructions are shown in ChemCom for every step. The process is broken into three steps:

(1) **Loading Input Files.** On the initial screen, the user is prompted to load a *query file* and a *database file*. The query file needs to be in conventional Structural Data (SDF) format. For the database, the user has the option of loading a SDF File or a saved ChemCom file. Although loading ChemCom files in place of SDFs can save a significant amount of time during comparisons, it is no longer possible to adjust configuration settings used to create the loaded ChemCom file.

(2) **Selecting Algorithm.** On the second screen, the user is able to select one of four algorithm modes, fingerprinting algorithms, minimum Tanimoto threshold, and any other parameters an algorithm mode may have associated with it. On this page, the user also has the option to choose the "preprocessing" option which enables ChemCom to utilize SymDex's version of the chosen algorithm. By checking "keep intermediate files" option available on this page, ChemCom will store the intermediate files used to perform the comparison for the selected SDF files, allowing ChemCom to reuse these files for future comparison, significantly reducing comparisons times.

(3) **Selecting Output Options.** On the third screen, if the user chooses to do so, options exist to output the results of the comparison as a CSV file, SDF files, and the indexed database as a ChemCom file. Upon finalizing the details of the comparison, the user will be prompted to confirm the chosen settings before the comparison commences. Real-time progress of the comparison is shown to the user on the screen that follows.

After running the comparison, the similarity information and duration details are presented in a user-friendly format. Detailed examples to run searches and comparisons can be found in a tutorial available at the accompanied website http://www.bioinformatics.org/chemcom.

■ CONCLUSIONS

We have described the development and features of ChemCom, a publicly available and free chemical comparator written in the Java programming language and implemented using the JavaWS framework. With its user-friendly interface and implemented ultrafast algorithms, we believe it will be found useful in drug discovery and other fields for performing similarity searches and comparisons of chemical databases. Moreover, it can be served as a common platform for developing and benchmarking novel algorithms for such purposes.

■ AUTHOR INFORMATION

Corresponding Author

*E-mail: jianwen.fang@nih.gov.

Present Address

S. Saeedipour: Kansas City University of Medicine and Biosciences, 1750 E. Independence Ave., Kansas City, Missouri 64106, United States.

DOI: 10.1021/ci500713s
J. Chem. Inf. Model. 2015, 55, 1292−1296

Author Contributions

J.F. conceived the project. S.S. and D.T. developed the software and performed comparison study with input from J.F. S.S., D.T., and J.F. prepared the manuscript. All authors have given approval to the final version of the manuscript. S.S. and D.T. contributed equally.

Notes

The authors declare no competing financial interest.

■ ACKNOWLEDGMENTS

We wish to thank the three anonymous reviewers and the editor for their constructive comments and suggestions. We are grateful to T. G. Kristensen, J. Nielsen, and C. N. Pedersen for releasing the source code of their KD-Grid and Multi Bit Tree implementation.

■ REFERENCES

(1) Nasr, R.; Hirschberg, D. S.; Baldi, P. Hashing algorithms and data structures for rapid searches of fingerprint vectors. *J. Chem. Inf. Model.* **2010**, *50*, 1358−1368.

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

(3) Aung, Z.; Ng, S.-K. An Indexing Scheme for Fast and Accurate Chemical Fingerprint Database Searching. In *Proceedings of the 22nd International Conference on Scientific and Statistical Database Management*; Springer-Verlag: Heidelberg, Germany, 2010; 288−305.

(4) Smellie, A. Compressed binary bit trees: A new data structure for accelerating database searching. *J. Chem. Inf. Model.* **2009**, *49*, 257−62.

(5) 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.* **2008**, *48*, 1367−1378.

(6) 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.

(7) Tai, D.; Fang, J. SymDex: Increasing the efficiency of chemical fingerprint similarity searches for comparing large chemical libraries by using query set indexing. *J. Chem. Inf. Model.* **2012**, *52*, 1926−35.

(8) Nasr, R.; Vernica, R.; Li, C.; Baldi, P. Speeding up chemical searches using the inverted index: The convergence of chemoinformatics and text search methods. *J. Chem. Inf. Model.* **2012**, *52*, 891−900.

(9) O'Boyle, N. M.; Banck, M.; James, C. A.; Morley, C.; Vandermeersch, T.; Hutchison, G. R. Open Babel: An open chemical toolbox. *J. Cheminf.* **2011**, *3*, 33.

(10) Wang, Y. L.; Xiao, J. W.; Suzek, T. O.; Zhang, J.; Wang, J. Y.; Bryant, S. H. PubChem: A public information system for analyzing bioactivities of small molecules. *Nucleic Acids Res.* **2009**, *37*, W623−W633.

DOI: 10.1021/ci500713s
J. Chem. Inf. Model. 2015, 55, 1292−1296