JOURNAL OF CHEMICAL INFORMATION AND MODELING

Article

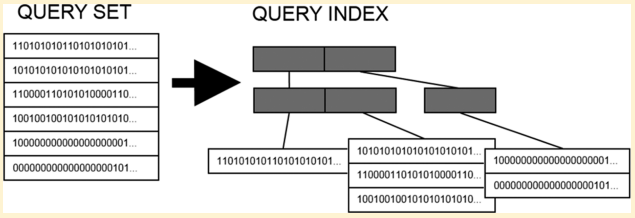
pubs.acs.org/jcim

SymDex: Increasing the Efficiency of Chemical Fingerprint Similarity Searches for Comparing Large Chemical Libraries by Using Query Set Indexing

David Tai and Jianwen Fang*

Applied Bioinformatics Laboratory, University of Kansas, Lawrence, Kansas 66047, United States

ABSTRACT: The large sizes of today's chemical databases require efficient algorithms to perform similarity searches. It can be very time consuming to compare two large chemical databases. This paper seeks to build upon existing research efforts by describing a novel strategy for accelerating existing search algorithms for comparing large chemical collections. The quest for efficiency has focused on developing better indexing algorithms by creating heuristics for searching individual chemical against a chemical library by detecting and eliminating needless similarity calculations. For comparing two chemical collections, these algorithms simply execute searches for each chemical in the query set sequentially. The strategy presented in this paper achieves a speedup upon these algorithms by indexing the set of all query chemicals so redundant calculations that arise in the case of sequential searches are eliminated. We implement this novel algorithm by developing a similarity search program called **Sym**metric in**Dex**ing or SymDex. SymDex shows over a 232% maximum speedup compared to the state-of-the-art single query search algorithm over real data for various fingerprint lengths. Considerable speedup is even seen for batch searches where query set sizes are relatively small compared to typical database sizes. To the best of our knowledge, SymDex is the first search algorithm designed specifically for comparing chemical libraries. It can be adapted to most, if not all, existing indexing algorithms and shows potential for accelerating future similarity search algorithms for comparing chemical databases.



1. INTRODUCTION

The sizes of chemical databases are becoming increasingly large as new chemicals are made available at unprecedented speed. The popular database PubChem in July 2012 when this paper was finalized contained 32.6 million chemical IDs. This represents nearly a 80% growth since December 2007 with growth expected to continue steadily.[1] PubChem only represents one of several online chemical databases and the countless chemical libraries maintained by public and private institutions.

Searching these databases has major applications in many fields such as drug discovery where High Throughput Screening (HTS) is used to discover candidate drugs using large chemical libraries.[2] Searches are used to create custom libraries, generate chemical libraries with dissimilar elements, and find chemicals similar to promising drug candidates.

Chemical fingerprints are often used to conduct these types of searches efficiently. Chemical fingerprinting is a method of simplifying chemical representation by describing the interesting properties of a chemical as a one-dimensional feature string. These are often represented as fixed length bit strings where each bit is set either on or off; each bit's state will be referred to as 1 for "on" and 0 for "off" on the basis of whether or not a feature is present. These generated chemical fingerprints can then be compared against each other in relatively fast searches.

Many methods have been developed to help speed up searches. Within the last four years, there have been numerous developments. For example, Swamidass et al.[3] developed an indexing scheme and heuristic to efficiently *prune*, eliminate comparisons between chemicals that cannot be similar, fingerprint databases by binning fingerprints by popcount (number of 1 bits in a fingerprint) and bounding using the maximal similarity score over bins of fingerprints. They then built onto their initial work, published by Baldi et al.,[4] by developing a XOR hash function to generate finer-grained second level bins for pruning as well a XOR filtering function to short-circuit low similarity comparisons. Smellie[5] created a tree-shaped compression method for traversing a database. Kristensen et al.[6] recently generalized the approaches developed by Smellie and Swamidass by deriving a single-level space partitioning index. They also tested a highly pruning multiple bit-matching tree. Even more recently both Nasr et al.,[7] as a continuation of the work done by Swamidass and Baldi, and Aung[8] independently utilized fingerprint bit group hashing, that is, dividing a string into several parts and hashing based on the popcount of each part, to generate even more fine-grained multi-level indexing systems.

1.1. Rationale.

For a simple single query search, the maximum number of comparisons is the number of chemicals in the database or the database size. The number of comparisons can easily be in the millions. Nevertheless, existing algorithms can often perform a million comparisons in just seconds. In contrast, comparing two large chemical libraries may require comparisons potentially numbering in the billions or even trillions, which can be very time consuming. For example, for an algorithm that can

Received: December 16, 2011
Published: July 31, 2012

dx.doi.org/10.1021/ci200606t | *J. Chem. Inf. Model.* 2012, 52, 1926−1935

compare one chemical to a library of a million chemicals in a second, it would take 11.6 days to compare two one-million chemical libraries (i.e., a trillion comparisons). As data driven techniques become more widespread and inexpensive and consequently more HTS facilities are created, performing large scale comparisons will become more common. For example, a HTS facility needs to compare its current collection to a vendor library before a decision can be made whether to acquire the whole or part of the library. It may also want to perform all-against-all for its current collection in order to investigate its diversity and covered chemical space. Two HTS facilities may want to compare their collections of chemicals before forming collaboration. Moreover, as the screening data accumulated in PubChem increases rapidly, it will become increasingly useful to compare local collections against the PubChem library. As part of the HTS operation at the University of Kansas, we have performed numerous large scale data comparisons, and in fact, the presented algorithm was originally proposed to increase the efficiency of performing these tasks.

Our strategy differs from the previous methods in two ways. First, it is not a new search index but rather an algorithm for comparing two large chemical sets utilizing existing search indices. Thus our main point of comparison is speedup over existing state-of-the-art algorithms. Second, we focus on the case of generally large *query sets* rather than that of sequential single or profile set query submissions.[3] The data structures in both data sets are exploited, and common calculations can be eliminated to gain a nontrivial speedup.

When naively executing a batch search, the number of comparisons is equal to the number of query chemicals (query size) multiplied by either the search target database size or the number of comparisons for single query searches if pruning per query is performed. However, the naïve method of performing batch searches does not take advantage of removing calculations shared between searches. We achieve a performance improvement over the naïve method by structuring and binning fingerprints in the query set to remove redundant comparisons (pruning per bin). The strategy utilized in SymDex is based upon creating a second index or *query index* over the whole query set rather than just a target database index. Pruning is performed using an entire query index bin rather than each query item individually.

1.2. Query Set Indexing. To perform similarity searches, a query fingerprint must first be indexed by hashing. Depending on how the search index prunes, one or more hashes are generated for each query fingerprint and stored as a fingerprint object's member variable or associated some other way with it (see Figure 1 for a simple example). Hashes can also be calculated on the fly as a search is executed, but this incurs penalties for recalculating or checking for recalculation of a fingerprint's hashes when using multi-tiered (an index of indices) indexing.[4,6−8] Performing searches over an example search index from Figure 2 using this type of index is simple and straightforward:

ID	L1 Hash	L2 Hash
FP 1	100	50,50
FP 2	100	50,50
FP 3	100	30,70
FP 4	200	100,100

Figure 1. This is an example query set with associated hashes for the Nasr indexing scheme[7] that uses the Swamidass bounding index[3] to index modulus 2 indices.

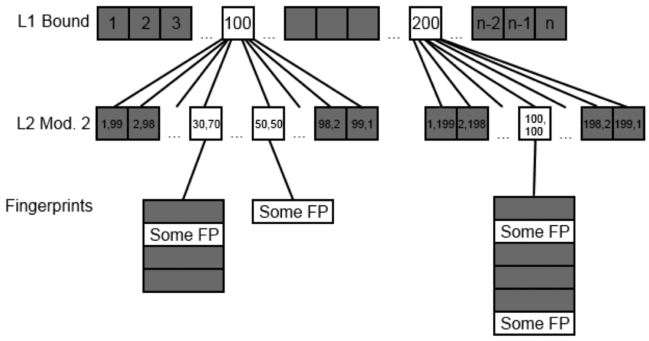


Figure 2. This shows an example search index being pruned for an exact search using the query set presented in Figure 1. Grayed out sections are pruned while white sections are unpruned. A search using single queries would take query set size (4) multiplied by number of hashes each (2) or $4 \times 2 = 8$ pruning steps. A search using the SymDex batch query set index would only take total number of distinct hashes or 5 pruning steps.

- Submit FP1 for the search.
  - Prune level 1 index for hash 100.
    - Prune level 2 index for hash (50,50).
      - Perform similarity calculations and thresholding for unpruned section of database.
- Submit FP2 for the search.
  - Prune level 1 index for hash 100.
    - Prune level 2 index for hash (50,50)
      - Perform similarity calculations and thresholding for unpruned section of database.
- Submit FP3 for the search.
  - Prune level 1 index for hash 100.
    - Prune level 2 index for hash (30,70).
      - Perform similarity calculations and thresholding for unpruned section of database.
- Submit FP4 for the search.
  - Prune level 1 index for hash 200.
    - Prune level 2 index for hash (100,100)
      - Perform similarity calculations and thresholding for unpruned section of database.

This methodology is adequate for searching a single query at a time, but it is inefficient compared to SymDex for executing batch searches. For SymDex, instead of simply associating hash keys by fingerprint, fingerprints are binned by their hash keys as illustrated in Figure 3. By using a structure based on associating fingerprints with hashes instead of based on associating hashes with fingerprints, pruning steps are allowed to be shared between fingerprints in each hash bin to be eliminated. Performing searches over Figure 3 is done by traversing the tree structure in a depth first fashion.

An example search would occur like so:

- Prune for level 1 hash 100.
  - Prune for level 2 hash (50,50)
    - Perform similarity calculations and thresholding using FP1 for unpruned database.

dx.doi.org/10.1021/ci200606t | J. Chem. Inf. Model. 2012, 52, 1926−1935

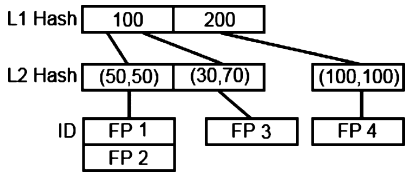


Figure 3. This shows the example query set from Figure 1 arranged in a tree structure used for the database search index instead of simply by member variable or some other type of association.

- Perform similarity calculations and thresholding using FP2 for unpruned database.
  - Prune for level 2 hash (30,70)
    - Perform similarity calculations and thresholding using FP3 for unpruned database.
- Prune for level 1 hash 200
  - Prune for level 2 hash (100,100)
    - Perform similarity calculations and thresholding using FP4 for unpruned database.

1.3. Performance. The total amount of time used by a search can be approximated by

$$S_{\text{Time}} \approx H_{\text{Time}} + P_{\text{Time}} + CT_{\text{Time}}$$

where S_{Time} is the total time and H_{Time}, P_{Time}, and CT_{Time} represent the time taken for the three major parts of the search algorithms: hashing, pruning, and comparison calculation. For typical single query searches

$$H_{\text{Time}} = N_{\text{Query}} \times \sum_{t=1}^{\text{Tiers}} h(t)$$

$$P_{\text{Time}} \approx N_{\text{Query}} \times \sum_{t=1}^{\text{Tiers}} p(t)$$

$$CT_{\text{Time}} \approx N_{\text{Query}} \times N_{\text{DB}} \times ct \times \prod_{\text{Tiers}}^{t=1} p_{\text{unpruned}}(t)$$

where N_{Query} is the size of the query set; Tiers is the number of index levels for the search index; $h(t)$ is the hash time for a particular level of index; $p(t)$ is the prune time for a particular level of index; N_{DB} is the size of the database; ct is the compare and threshold time, and $p_{\text{unpruned}}(t)$ is the percent pruned for each level.

The batch search algorithm proposed in this paper instead provides an upper bound for pruning time, which as a side effect slightly increases the hash time to account for structuring the query set. The batch searching algorithm used in SymDex only prunes per used hash instead of per query fingerprint hash. As the number of all possible hashes generated by current pruning methods is finite and relatively small; this can drastically reduce the number of instances that the pruning routine is executed by limiting it to the number of used hashes. Thus two variables are changed in the time equation and CT_{Time} remains the same

$$\bar{H}_{\text{Time}} \approx N_{\text{Query}} \times [h_{\text{insert}} + \sum_{t=1}^{\text{Tiers}} h(t)]$$

$$\bar{P}_{\text{Time}} \approx \sum_{\text{Tiers}}^{t=1} h_{\text{used}}(t) \times (p_{\text{retreive}}(t) + p(t))$$

where h_{insert} is the time it takes to insert an entry into the query index; $h_{\text{used}}(t)$ is the number of used hashes on a level of index; and $p_{\text{retrieve}}(t)$ is the cost of retrieving all hashes out of a structure at a level of index. Plainly then $\bar{P}_{\text{Time}}$ gives an upper bound for pruning time on the basis of the number of possible unique hashes per level $h_{\max}(t)$

$$\bar{P}_{\text{Time}} \leq \sum_{t=1}^{\text{Tiers}} h_{\max}(t)*(p_{\text{retrieve}}(t) + p(t))$$

Thus, for sufficiently large query sets, pruning time becomes constant and, therefore, very small compared to the overall time costs for sufficiently large searches. The trade off, as mentioned before, is that h_{insert} is introduced as an additional parameter in $\bar{H}_{\text{Time}}$. However, because there are many highly efficient data structures but few low cost pruning functions, h_{insert} generally is much less than $\sum_{t=1}^{\text{Tiers}} p(t)$ which gives an overall speedup for search times.

There is one drawback of this method that should be mentioned. At the point P_{Time} decreases to a constant, the speedup compared to nonquery indexed searches will begin to decrease. This is due to the fact that the search algorithms still have a time complexity of $O(n)$. The SymDex algorithm reduces the leading coefficient from a larger constant. Therefore, for very large query sets, the upper bound of P_{Time} will be so small compared to H_{Time} and C_{Time} as not to have a meaningful contribution. However, as we will demonstrate in following section, SymDex can achieve significant speedup for comparing chemical sets with tens of thousands to a few million chemicals, sizes of most chemical libraries. This potential problem nevertheless can be addressed more concretely. Algorithms that can be modified or reconfigured to prune better or novel algorithms with better pruning can be developed to potentially reduce C_{Time} by reducing $\Pi_{t=1}^{\text{Tiers}} p_{\text{unpruned}}(t)$. As long as the upper bound of P_{Time} for those algorithms is not extraordinarily larger than current algorithms, SymDex will be able to produce good results.

1.4. Implementation. The data structure used for the query index may vary in complexity. A query index can be intuitively and generically abstracted and implemented as a hash table of lists of fingerprints or a hash table of hash tables in the case of multi-tiered indices. A query set may also be indexed as if it were a database. For our implementations, we chose the second option that contributes the "Symmetrical" part of the algorithm name as it appears more time efficient (though perhaps less space efficient depending on implementation) as most of the published indexing schemes use a variation of n-ary trees[4−8] or a simple array in the case of Swamidass.[3]

SymDex also has an advantage in that it allows for the comparison of extremely large databases that have pre-existing indices. Large databases must be indexed regardless of efficient querying, so this scheme can avoid the need to rebuild the query set index. While out-of-memory search implementations may be required to handle such a large operation, it is worth considering when the appropriate cases arise.

2. METHODS

2.1. Databases and Fingerprints. The first 1 million chemicals from PubChem were downloaded for the initial phase of testing presented in Section 3.1. An additional 3 million were downloaded for the second part (Section 3.2) of testing that involved searching two very large databases for a total of 4 million chemicals used across the experiments used in this paper. The database consisted of all chemicals downloaded, and the query

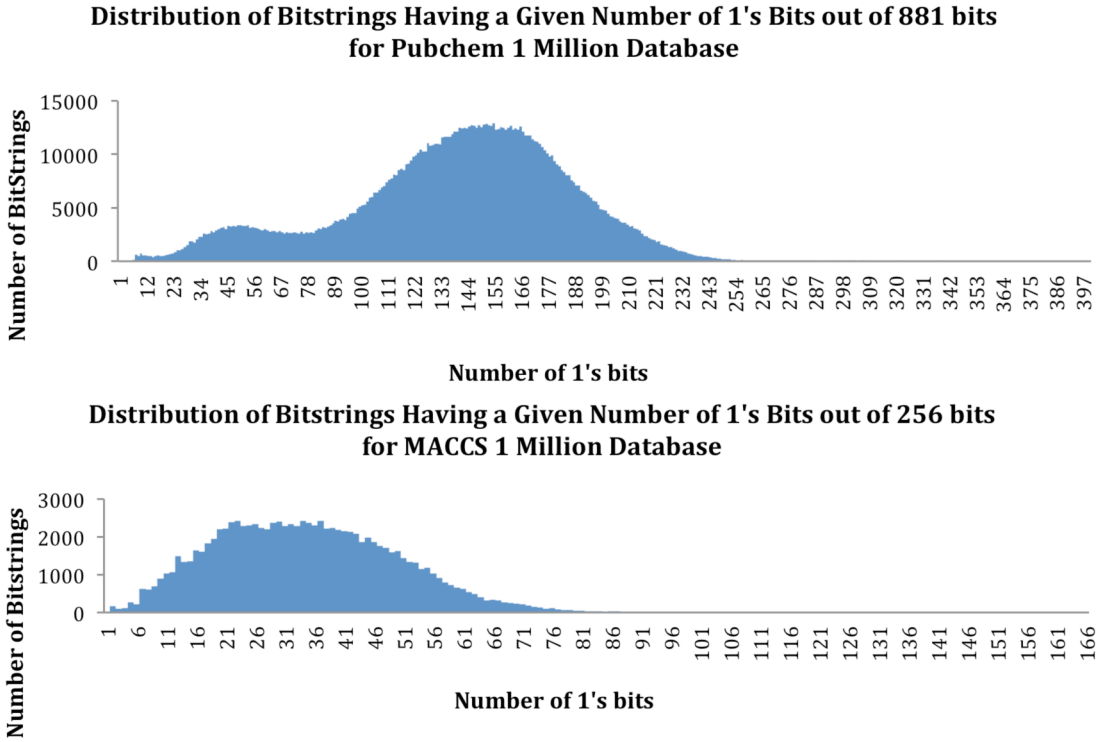


Figure 4. The number of 1s in each fingerprint for both fingerprint databases used for the experiments. The *x* axis for the PubChem fingerprint is truncated.

sets consisted of a random selection sampled from the database. To measure the effects of fingerprint length and show differences in search time for different bit lengths and feature sets, we used two widely used fingerprints: the 881-bit fingerprint generated by Pubchem and the 256-bit MACCS (166-bits padded to 256-bits) fingerprint generated using OpenBabel (http://openbabel.org/). For each fingerprint, the number of 1s per fingerprint database is listed in Figure 4. This is of interest when analyzing the full case because it shows that the large difference in the number of used hashes and the number of total hashes. This means the actual full case is reached when a fraction of bins have been generated. For example, the bit bound algorithm by Swamidass et al.[3] theoretically has a number of hashes equal to the number of bits. However, only a fraction of the hashes is used given that the maximum number of 1s is much less 881 or 256 for each fingerprint and that the average number of 1s is roughly a normal distribution about a fraction of the total number of bits.

2.2. Binary Bit-String Fingerprints. For our research, we focused on simple fingerprints represented as fixed length binary strings as these are the most common. SymDex is not biased toward other fingerprint representations, so the general idea could be extended to integer and other fingerprint types. However, it should be recognized that the speedup improvement factor may vary on the basis of the complexity of comparison function and the size of the fingerprint. Our particular implementation of binary fingerprints was as a binary bit-string over 64-bit integers for which we used word level processor instructions to achieve the maximum comparison speed.

2.3. Tanimoto Similarity Metric. For our similarity metric we used the Tanimoto similarity score. Not only is this one of the most popular metrics, it seems to have become by convention the defacto standard for benchmarking chemical similarity search indices.[3−10] A great deal of research has been devoted to figuring out new methods of exploiting this metric,[9,10] and thus, we feel no desire to break with tradition.

$$S_t = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} = \frac{|A \cap B|}{|A \cup B|}$$

where s_t is the desired minimum Tanimoto similarity threshold; A and B are fingerprints; $|x|$ represents cardinality (number of 1s bits); and ∩ and ∪ represent bit-wise intersection and union. The standard formulations of the Tanimoto similarity metric can also be represented in a way that improves performance

$$s_t = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Let $X = |A \cap B|$ and $Y = |A| + |B|$ so that

$$s_t = \frac{X}{Y - X}$$

$$s_t(Y - X) = s_tY - s_tX = X$$

$$s_tY = X + s_tX = X(1 + s_t)$$

$$\frac{s_tY}{1 + S_t} = X$$

$$\frac{s_t}{1 + s_t}(|A| + |B|) = |A \cap B|$$

Though one has to be cautious of mitigating machine precision in practice, the optimized formulation reduces by an extra subtraction that is off-loaded into the left side, which is a constant that only must be calculated once. Additionally |A| and |B| are calculated for certain algorithms[3,6] as part of pruning, while |A U B| is not. Thus the equation's entire left side reduces into a constant, and the cost of running a single similarity query is equal

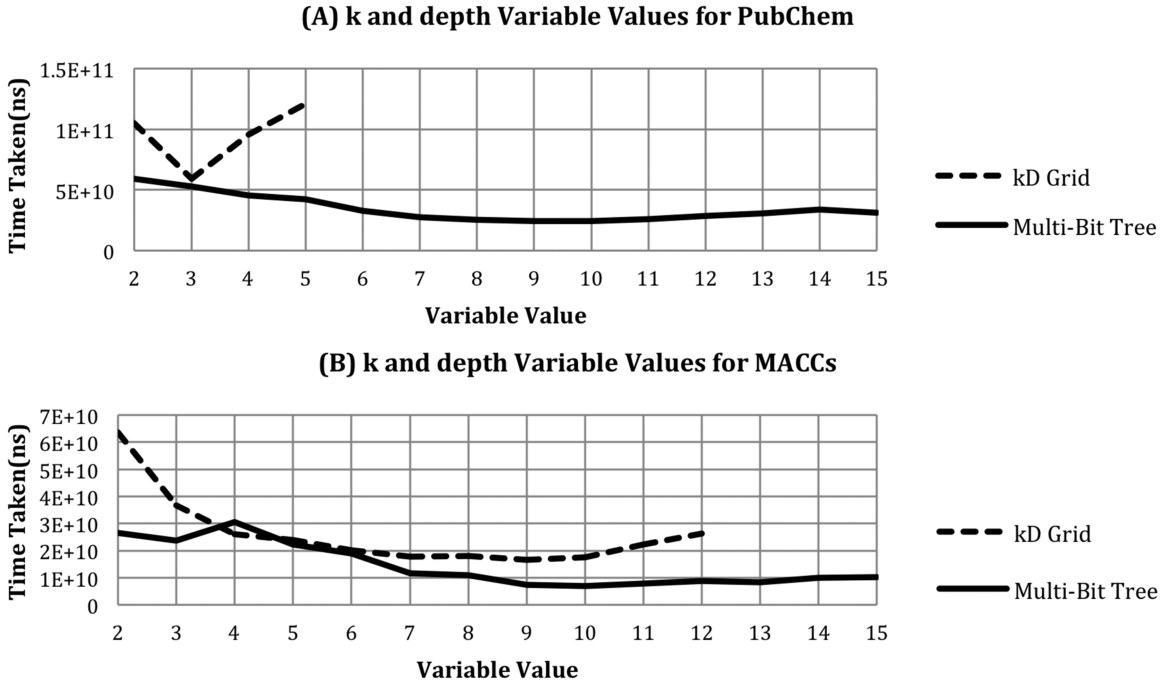


Figure 5. Average time taken for a single query for each similarity search algorithm for query set sizes of 10,000 and a database size 1,000,000. For the PubChem database, the best $k = 3$ and the best depth = 10. For the MACC database, the best $k = 9$ and the best depth = 10.

to the combined cost of a bit-wise intersection, popcount, and comparison.

Both these representations can then be bounded to create pruning heuristics[3] that rely on only logical branches, division, and comparison and is efficient for cases where bit-strings are scattered widely across a candidate space leading to relatively low scores for the majority of comparisons.

Since

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

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

It follows that

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

$$\frac{s_t}{1 + s_t}(|A| + |B|) \leq \min(|A|, |B|)$$

A 90% Tanimoto similarity threshold was used for all the experiments in the study.

2.3. Computational Resources. All experiments were performed on an Intel i5-2500K with 16GB of memory. All algorithms and tests were implemented using Java. Tests were run with a virtual machine heap size starting at 12GB with a maximum amount of 12GB and using the −Xincgc to enable continuous parallel garbage collection on a separate process. This was done to prevent long waits for major garbage collection on the same thread.

2.4. Base Algorithms. We selected three algorithms aside from the naïve brute force pairwise comparison algorithm for our experiments. Algorithms selected were based on speed and any interesting properties. Each algorithm was reimplemented in Java using a consistent object oriented pattern to ensure fairness for comparison purposes. For algorithms that required an additional input value either for string partition (k) or tree depth, 10,000 queries were run against 1 million chemicals from both fingerprint databases at a 90% similarity threshold to determine which settings were best for which algorithms at these sizes. The value of 10,000 chemicals in the query set to search against the database was selected because it was found to be by the authors large enough to not skew results in favor of small query set size but small enough to allow for easy computing. The values of k and tree depth of course change depending on databases and query set sizes as well as fingerprint property. However, altering k and depth on the basis of these variables would introduce a new dependent variable, and thus, it was required to find single values of k and depth that would work well for a variety of query set sizes. The result of these experiments are shown in Figure 5.

While we are aware of the XOR folding check[4,6] and other indexing algorithms that use forms of short circuiting to abort similarity calculations,[6,8] we made a decision not to implement any of these algorithms. This is because short circuiting similarity calculations alter the cost per similarity comparison on the basis of content of database and query set and therefore CT_{Time}. Because we are comparing query set indexing implemented in SymDex with standard sequential search for the same indexing algorithm only, CT_{Time} should be kept as constant as possible. Thu,s we decided not to implement this useful optimization or Aung's algorithm[8] for the same reason.

The first algorithm implemented was the bit count maximum similarity bounding algorithm proposed by Swamidass.[3] This algorithm will be referred to as the *bit bound* algorithm. The cost for pruning in this algorithm is extremely small and thus would show how query set indexing would fare if the pruning cost is miniscule compared to the compare and threshold step CT_{Time}. This contains 881 and 256 possible hash values; one for each possible number of 1s possible in a bit-string.

The second fingerprint algorithm is the *multi-bit tree* algorithm proposed in the Kristensen paper.[6] This algorithm is a multi-tiered algorithm similar to a radix tree that uses the bound algorithm as the first level index with each bin containing an

dx.doi.org/10.1021/ci200606t | J. Chem. Inf. Model. 2012, 52, 1926−1935

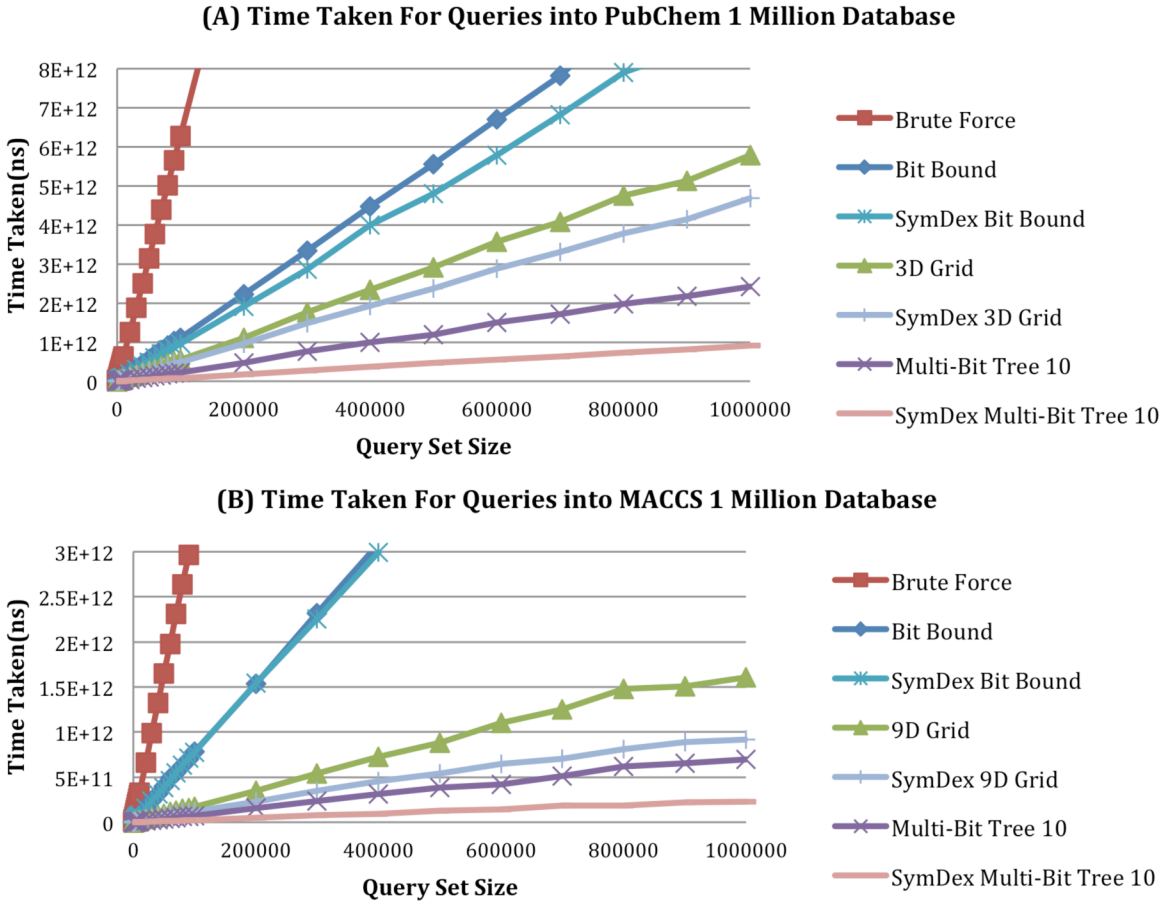


Figure 6. (A) Time taken for searches using query sets sizes 100 to 1000 incrementing by 100, 2000 to 10000 incrementing by 1000, and 20,000 to 100,000 incrementing by 10,000 for 881-bit PubChem Fingerprint. (B) Experiment was repeated for MACCS. Time measured was the amount of time spent hashing or indexing the query set and executing the search. Values for brute force are estimated based on trend line due to extremely large time requirements required to perform experiments.

index that uses a binary tree to divide the database recursively, recording all common bits at each tree node. We selected this algorithm because its approach is extremely quick. Another reason that this algorithm is of interest is that the trees inside each bin do not provide a straightforward way of generating query indexes. Thus, only the first tier is optimized by SymDex. It was found that a depth of 9 and 10 were ideal for the 881-bit and 256-bit algorithms, respectively. This data is shown in Figure 5.

The third fingerprint algorithm we implemented is the single tier *k*D Grid search also proposed in the Kristensen paper.[6] This algorithm is similar to the Nasr modulus[7] and Aung[8] algorithms in that it divides bit strings into k number of equal-sized bit groupings much like a general purpose space partitioning tree. The interesting difference between this and other bit grouping algorithms is that this uses an extended version of the bound algorithm to prune bit strings with a lower-cost similarity threshold instead of doing pairwise comparisons between the query string and every unpruned fingerprint bin. This approach is also a generalization of the compressed bit tree by Smellie. This algorithm generates $((\text{Fingerprint Size})/k)^k$ hashes or in the case of 881-bit and 256-bit fingerprints, 2.5 billion and 4 million using the best values of $k = 3$ and $k = 9$, respectively, as shown in Figure 5.

3. RESULTS AND DISCUSSION

3.1. Queries into Large Databases. Figure 6 shows search time for the SymDex query set indexing approach scales with query size. As predicted, searches with indexed query sets run in less time for large query set sizes when compared with each algorithm's sequential querying counterparts. This supports the idea that removing the redundant pruning by indexing the query set can achieve a significant speedup. This is explicitly seen in Figure 7 where the time difference spent on pruning and hash between the sequential search and the SymDex version of each algorithm can be seen. Observe that the increase cost of hashing is greatly offset by the eliminated cost of pruning. In fact, the cost for hashing was (roughly) estimated to require only a few seconds for even complex in-memory trees, and thus, only algorithms with extremely complex indices would see a significant hashing cost.

The quantitative results can be seen in Figure 8 where the speedup compared to brute force is shown. This data mirrors the data collected by Kristensen et al[6] (compare Figure 6A to Figure 7 from the cited paper) for the speed rankings of the algorithms where multi-bit tree was faster than *k*D Grid, and *k*D Grid was faster than bit bound. These results also show that the SymDex implementation of multi-bit tree achieved the greatest speedup (multi-bit tree sees large improvements between 126% and 304%) in Figure 8. This is interesting because there is not a strong correlation between algorithm speed and speedup shown in Figure 8 because the speedup for *k*D Grid and bit bound were similar (Figure 9). However, there are a variety of factors that help explain why multi-bit tree achieves such a large speedup, the most important of which is the proportion of hashing and pruning compared to thresholding all possible candidate chemicals from the database. Recall the equation

dx.doi.org/10.1021/ci200606t | J. Chem. Inf. Model. 2012, 52, 1926−1935

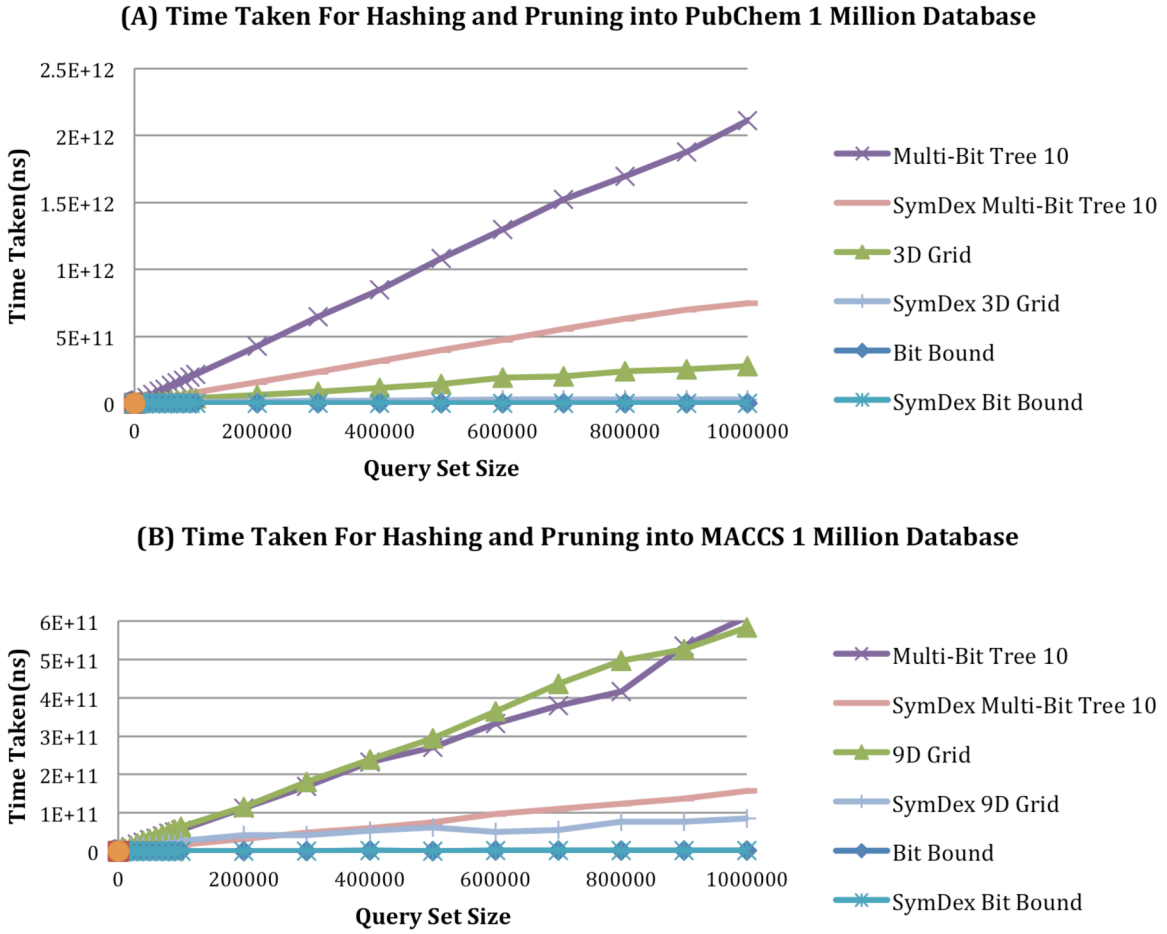


Figure 7. Hashing time cannot be reliably timed separately due to timer resolution problems. While most Java implementations incorporate a nanosecond timer, the reliability of these is suspect for relatively small values. Because hashing is such a small component of the total time, it suffers from such inaccuracies. However, it is not needed to know the exact interactions of hash time costs versus prune time costs, only how SymDex changes the sum of the two values ($H_{\text{Time}} + P_{\text{Time}}$) to show the effects of SymDex applied to some algorithm.

Database	Query Size	Bit Bound			*k*D Grid			Multi-bit Tree		
		original	SymDex	**ratio**	original	SymDex	**ratio**	original	SymDex	**ratio**
PubChem	1000	4.64	5.44	**1.20**	6.79	16.04	**1.00**	14.75	18.64	**1.25**
	10000	5.45	6.24	**1.15**	10.18	17.43	**1.00**	21.18	40.78	**1.89**
	100000	5.48	6.34	**1.16**	11.15	17.93	**1.07**	26.30	67.09	**2.54**
	1000000	5.50	6.36	**1.16**	11.23	18.64	**1.21**	26.75	69.52	**2.70**
MACCS	1000	3.95	4.28	**1.14**	16.04	16.10	**1.00**	41.47	72.99	**1.76**
	10000	4.20	4.18	**0.99**	17.43	18.62	**1.07**	43.19	109.35	**2.53**
	100000	4.19	4.22	**1.01**	17.93	22.59	**1.26**	44.51	128.28	**2.88**
	1000000	4.26	4.33	**1.02**	18.64	30.31	**1.63**	44.18	134.45	**3.05**

Figure 8. Average speedup compared to brute force search trend line from experimental results. Averages are computed over multiple intervals to show how speedup changes with query set size. The database size is 1 million chemicals.

$$S_{\text{Time}} \approx H_{\text{Time}} + P_{\text{Time}} + CT_{\text{Time}}$$

for which if CT_{Time} is very large, then the contribution of $H_{\text{Time}} + P_{\text{Time}}$ to S_{Time} becomes negligible. The best example of this phenomenon is the MACCS fingerprint experiment. Figure 7B shows how the hashing and pruning speedup are very similar for *k*D Grid and mult-ibit tree, but Figure 6B shows that multi-bit tree achieves much better performance and nearly twice the speedup. Looking at Figure 10B, it can be seen that the total number of candidate chemicals thresholded for multi-bit tree are

dx.doi.org/10.1021/ci200606t | J. Chem. Inf. Model. 2012, 52, 1926−1935

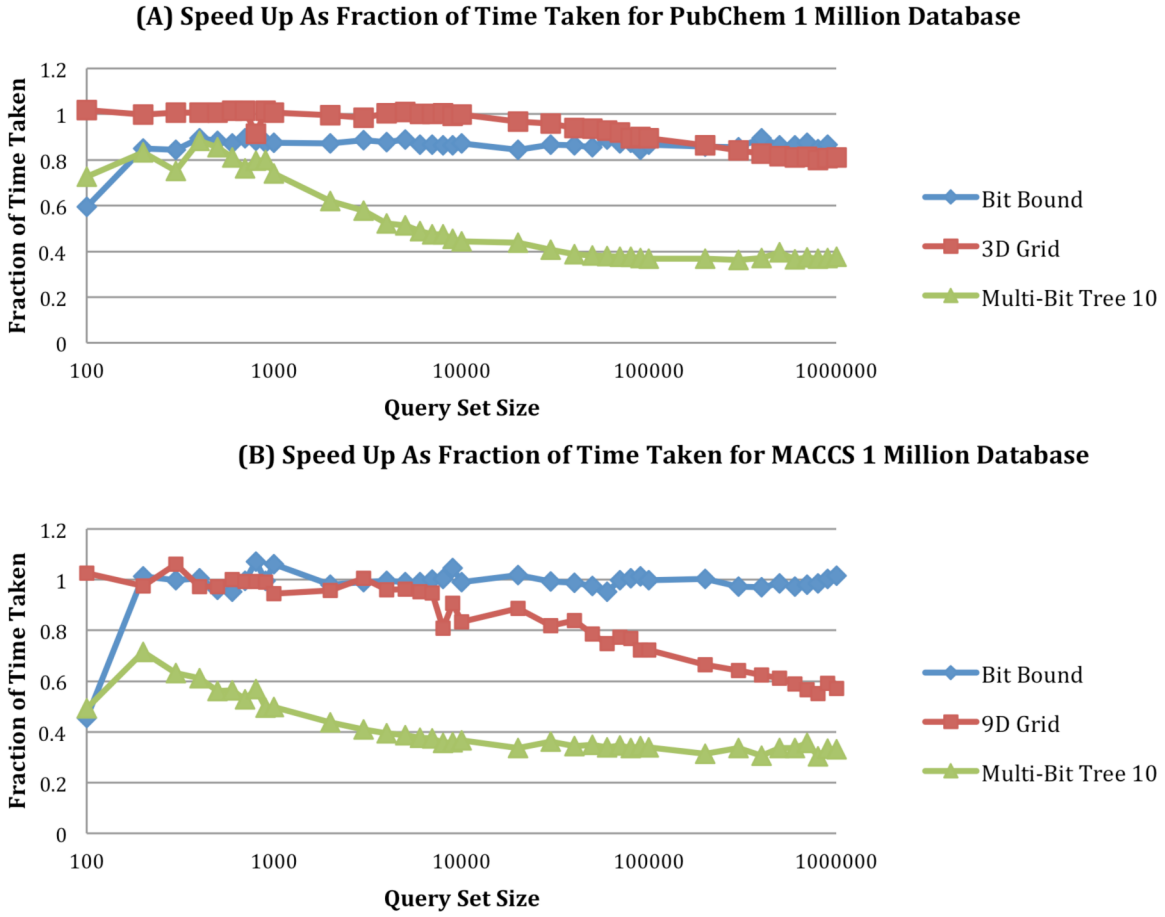


Figure 9. Speedup as fraction of time on a logarithmic scale.

a fraction of the ones thresholded for *k*D Grid. For a more apple-to-apple comparison, examine the *k*D Grid pruning performance from Figure 10 between MACCS and PubChem. PubChem fingerprints did not prune as sharply as MACCS ones leading to less of a speedup. From these experimental results, it can be reasoned that SymDex tends to provide greater speedup for algorithms utilizing expensive but accurate pruning algorithms and fingerprinting algorithms that tend to generate dissimilar fingerprints. If this holds true, then as future indexing and pruning algorithms become more accurate and complex, the application of symmetric indexing will produce even faster searches.

The opposite pattern is shown by bit bound in Figures 8 and 9. Here, the full case is reached almost immediately because the number of unique hashes is extremely low. Logic similar to that used in the previous paragraph for explaining *k*D can be used. In the case of bit bound, the value CT_{Time} becomes so large that the contribution of $H_{\text{Time}} + P_{\text{Time}}$ becomes very small. This is also compounded by the fact that P_{Time} for bit bound is also very small and the fact that H_{Time} is also extremely tiny. Thus, CT_{Time} dominates the calculation time as supported by Figure 7 in which bit bound consistently has very short hashing and pruning times. Thus algorithms utilizing simplistic pruning methods like bit bound make relatively poor candidates for SymDex except for very small query sets that occur before the full case is reached where speedup is still seen. Algorithms exhibiting characteristics of bit bound would benefit most from short circuiting algorithms mentioned in Section 2.4, which reduce CT_{Time} after which they may benefit more from SymDex.

There are several other important points of discussion. First each SymDex algorithm showed some speedup at even relatively small query set sizes of less than 1000 as seen from Figures 8 and 9. However, it is not readily obvious from the data gathered in these experiments how well SymDex performs on small query set sizes as those results vary wildly with how similarly the query strings are.

Second, for the 881-bit PubChem fingerprint data, it took 40 min using the multi-bit tree algorithm without SymDex to perform a similarity comparison at 90% similarity from Figure 6. The SymDex version of the algorithm took only 15 min (i.e., 266% speedup). This translates into a 25 min savings of time, which is a nontrivial amount of time and tied up computational resources. This is representative of searching between two relatively small databases.

Another important observation seen in Figure 9 is that the query set size needed to reach the full case is reached, and speedup levels out is very large for the algorithms tested excepting bit bound. The full case is only reached for multi-bit tree after the query set reaches 10,000, and *k*D Grid does not apparently reach the full case until after the query set becomes larger than the database. For *k*D Grid, reaching the full case is only conditionally useful as the database is generally larger than the query set. In the next section, we will look at very large database-to-database searches and see whether or not *k*D Grid reaches the full case as database sizes increase.

3.2. Comparisons of Large Databases. For this section, we downloaded the first 4 million chemicals from PubChem so we can alter the size of the database. We studied the two faster algorithms, multi-bit tree and *k*D Grid, but skipped bit bound

dx.doi.org/10.1021/ci200606t | J. Chem. Inf. Model. 2012, 52, 1926−1935

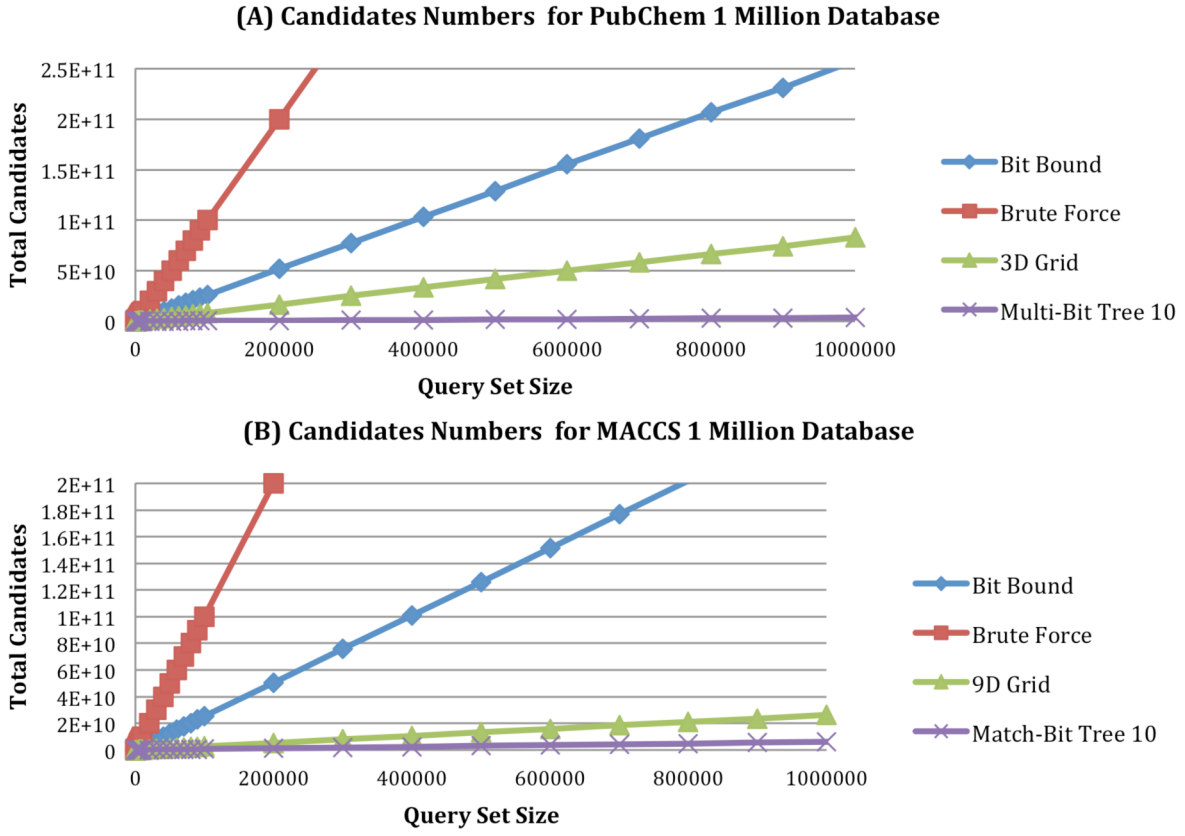


Figure 10. Number of candidates for each algorithm at the query set sizes used by the first experiment.

Database	Query Size	Database Size	*k*D Grid			Multi-bit Tree		
			Original (min)	SymDex (min)	ratio	Original (min)	SymDex (min)	ratio
PubChem	1,000,000	2,000,000	181.16	152.70	**1.19**	63.07	25.34	**2.49**
	1,000,000	3,000,000	206.73	178.64	**1.16**	70.32	28.72	**2.45**
	1,000,000	4,000,000	286.93	249.16	**1.15**	87.94	37.49	**2.35**
	2,000,000	2,000,000	286.91	240.15	**1.19**	125.74	52.35	**2.40**
	3,000,000	3,000,000	653.98	543.64	**1.20**	205.69	83.24	**2.47**
	4,000,000	4,000,000	1180.77	1038.57	**1.14**	361.83	156.83	**2.31**
MACCs	1,000,000	2,000,000	54.03	25.37	**2.13**	17.77	6.78	**2.62**
	1,000,000	3,000,000	55.85	32.61	**1.71**	21.07	8.05	**2.62**
	1,000,000	4,000,000	72.71	45.13	**1.61**	26.26	12.25	**2.14**
	2,000,000	2,000,000	106.70	48.11	**2.21**	36.74	14.64	**2.51**
	3,000,000	3,000,000	174.35	86.39	**2.02**	64.71	26.36	**2.46**
	4,000,000	4,000,000	286.77	164.14	**1.75**	105.35	48.11	**2.19**

Figure 11. This figure shows the number of minutes taken for searches against larger database sizes 2,000,000, 3,000,000, and 4,000,000 with various query sizes along with the speedup seen between SymDex and standard implementations.

because of the time required. We recalculated algorithm variables *k* and depth to get the best performance for each algorithm. Two separate data sets were generated for this section. The first increased database size from 2 to 4 million chemicals with the query set size pegged at 1 million to simulate a smaller database searching against a larger database. The second experiment searched the database against itself to simulate two large databases being searched against each other. The results of both these experiments are summarized in Figure 11.

Performance for larger databases follows the same trends observed in Section 3.1 as expected. It also shows that after database sizes get to a certain point, the percentage of speedup

dx.doi.org/10.1021/ci200606t | *J. Chem. Inf. Model.* 2012, 52, 1926−1935

depends mostly on the properties of the fingerprint and composition of the fingerprints from the database. Figure 11 shows the speedup for each algorithm between standard and SymDex versions. Evidence seems to indicate that speedup is related to composition of the database because the speedup between the experiments performed in Figure 11 does not vary extraordinarily for the same database sizes despite the difference in query set size. However, more research is needed to obtain a definite answer.

4. CONCLUSIONS AND FUTURE PLAN

We have developed a method for accelerating large queries into chemical databases and implemented it as SymDex, a Java application for performing chemical fingerprint searches. SymDex is available, free of charge for nonprofit organizations, upon request.

We have shown that this method produces a noticeable speedup for even small (less than 1000) query sets. For larger query sets and database to database comparisons, we achieve a speedup of an average of 232% for the multi-bit tree indexing algorithm. This is extremely useful as multi-bit tree is the fastest current algorithm. Other algorithms also received a benefit though to a lesser extent. However, even small speedups can represent significant time savings for sufficiently large databases and queries. The experiments also show that SymDex achieves a higher speedup after the full case is reached for methods that have higher pruning ability as seen from Figures 8 and 9. They also support the theory that convergence occurs more quickly for smaller query sizes when the pruning index has a low number of hashes.

There are many further applications for the algorithm that are not explored in this paper. Extending query set indexing to other types of fingerprinting is one major application, specifically those relying on fixed length strings or finite alphabets. Parallelization is also a useful application, and measuring how well SymDex scales over multiple processes is not addressed in this paper because the experiments in the present study were designed to compare SymDex directly to its base algorithms in order to demonstrate the achieved speedup. Nevertheless, conversion to a parallel enabled version would be rather simple to implement. Just break up the task so that each processor assigned is assigned all the calculations for a unique hash in the query set. The nature of the indices being read only requires no locking, so no additional logic is needed except a job dispatching thread. Additionally, because the algorithm implemented in SymDex is independent of indexing algorithms, it may be implemented for new indexing algorithms that are not addressed in this paper.

AUTHOR INFORMATION

Corresponding Author

*Phone: (785) 864-3349. Fax: (785) 864-8141. E-mail: jwfang@ku.edu.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

We are grateful to the three anonymous reviewers and the associate editor for their constructive comments and suggestions. We thank T. G. Kristensen, J. Nielsen, and C. N. Pedersen for releasing the source code for their KD-Grid and Multi Bit Tree implementation. We also want to thank Dr. Rathnam Charguturu and all other members from the High Throughput Screening Laboratory at the University of Kansas for their useful suggestions.

REFERENCES

(1) Baykoucheva, S. A new era in chemical information: Pubchem, discoverygate, and chemistry central. *Online* **2007**, *31* (5), 16−20.

(2) Mayr, L. M.; Fuerst, P. The future of high-throughput screening. *J. Biomol. Screening* **2008**, *13* (6), 443−448.

(3) 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* (2), 302−317.

(4) 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* (7), 1367−1378.

(5) Smellie, A. Compressed binary bit trees: A new data structure for accelerating database searching. *J. Chem. Inf. Model.* **2009**, *49* (2), 257−262.

(6) 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.

(7) Nasr, R.; Hirschberg, D. S.; Baldi, P. Hashing algorithms and data structures for rapid searches of fingerprint vectors. *J. Chem. Inf. Model.* **2010**, *50* (8), 1358−1368.

(8) 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; pp 288−305.

(9) Baldi, P.; Hirschberg, D. S. An intersection inequality sharper than the Tanimoto triangle inequality for efficiently searching large databases. *J. Chem. Inf. Model.* **2009**, *49* (8), 1866−1870.

(10) Baldi, P.; Nasr, R. When is chemical similarity significant? The statistical distribution of chemical similarity scores and its extreme values. *J. Chem. Inf. Model.* **2010**, *50* (7), 1205−1222.