# Parallel Algorithms for Mining Frequent Structural Motifs in Scientific Data

Chao Wang and Srinivasan Parthasarathy*

Department of Computer and Information Science, The Ohio State University
Contact Email: {wachao,srini}@cis.ohio-state.edu

## ABSTRACT

Discovery of important substructures from molecules is an important data mining problem. The basic motivation is that the structure of a molecule has a role to play in its biochemical function. There is interest in finding important, often recurrent, substructures both within a single molecule and across a class of molecules.

Recently, we have developed a general purpose suite of algorithms – the MotifMiner Toolkit – that can mine for structural motifs in a wide area of biomolecular datasets. While the algorithms have proven to be extremely useful in their ability to identify novel substructures, the algorithms themselves are quite time consuming. There are two reasons for this: i) inherently the algorithm suffers from the curse of subgraph isomorphism; and ii) handling noise effects (e.g. protein structure data) results in a significant slowdown.

To address this problem in this paper we propose parallelization strategies in a cluster environment for the above algorithms. We identify key optimizations that handle load imbalance, scheduling, and communication overheads. Results show that the optimizations are quite effective and that we are able to obtain good speedup on moderate sized clusters.

## 1. INTRODUCTION

The importance of data mining and knowledge discovery [15, 14], in scientific arenas, is growing. Fields as diverse as astronomy[22], bioinformatics[4], materials science[16], and medicine[34], are among many facing the situation where the data collected, be it from observations, experiments, simulation studies, or health screening tests, are becoming larger and more intricate. Dataset size impedes easy understanding and thus there has been an increasing clamor for mining of such scientific and biomedical data [18, 12]. The objective is to fundamentally advance the understanding of science via the semi-automated analysis of such datasets.

*This work is supported in part by an NSF CAREER grant (IIS-0347662) and an NSF ACR grant (ACI-0234273)

The challenge is that the data in its raw form is often intricate and unstructured. Mining meaningful patterns from such data is often impossible without accounting for the complex scientific relationships embodied in such data. Modeling such complex relationships (e.g. activity as a function of shape or structure) accurately is thus imperative. However, accurately modeling and then mining such data is both compute and I/O intensive. A further complication is that the algorithms and algorithmic parameters that can achieve desirable results on a given problem and dataset needs to be determined through an interactive and iterative process. In such an environment response time is crucial because lengthy time delay between responses of two consecutive user requests can disturb the flow of human perception and formation of insight.

In this paper we focus on meeting the above challenge in the context of analyzing biomolecular and scientific datasets. Discovering important structures in molecular datasets has been the focus of many recent research efforts in scientific data analysis[2, 9, 19, 4, 11, 36, 30, 23, 13, 8, 37, 5, 28, 29, 16]. To address the needs of such applications we have recently developed a general-purpose prototype toolkit, MotifMiner[27, 32, 6, 7], that detects frequently occurring structural motifs. We have successfully conducted an indepth evaluation of the MotifMiner toolkit from a qualitative perspective on various datasets including pharmaceutical data[6], tRNA, protein data (from the PDB)[32, 6] and data from molecular dynamics simulations[5].

To maintain the accuracy, general-purposeness and flexibility of MotifMiner toolkit, while significantly improving the execution time, in this paper we propose parallel implementations of some of the key algorithms that the toolkit provides. The resulting algorithms demonstrate good speedup on moderate sized clusters. Along the way we have also identified some key optimizations that improve the scaleup properties of both the sequential and parallel algorithms. Specifically our key contributions are:

- We present basic parallelization of the intra-structure and inter-structure MotifMiner Algorithms. This relies on standard privatize-and-reduce parallelization principles to reduce communication costs.
- We propose bitonic scheduling in conjunction with a self-adaptive extension scheme (frontier node expansion described in Section 3) to obtain better workload distribution among processors, thereby improving the overall parallel performance.
- We also propose a backward pruning optimization, along

with some other more basic optimizations that apply to both parallel and sequential algorithms. In particular, the proposed backward pruning optimization draws inspiration from some recent work in maximal itemset mining[3]. This optimization helps improve both the memory and computation performance of both sequential and parallel algorithms quite significantly.

- We demonstrate the performance of the above algorithms, in conjunction with the proposed optimizations and are able to achieve good speedup on queries on real datasets.

The rest of the paper is organized as follows. Work related to the current effort is described in section 2. Section 3 details the problem statement and then describes the two parallel algorithms, i.e., intra-structure p-MotifMiner and inter-structure p-MotifMiner. The optimizations for the algorithms are given in Section 4. The experimental results validating the proposed algorithms and optimizations are described in Section 5. Finally we present our conclusions and outline directions for future work in Section 6.

## 2. RELATED WORK AND BACKGROUND

**Substructure Analysis:** Discovering important structures in molecular datasets has been the focus of many recent research efforts in scientific data analysis[2, 9, 19, 4, 11, 36, 30, 23, 13, 8, 37, 5, 28, 29, 16, 9]. These efforts have targetted substructure analysis in small molecules, material defect analysis in molecular dynamics simulations, and more recently macromolecules such as proteins and nucleic acids[21, 31, 40, 17, 20, 33, 35].

Several methods for secondary level motif finding in proteins have been proposed in the past. An algorithm based on subgraph isomorphism was proposed in [30]; it searches for an exact match of a specific pattern in a database. Search for distantly related proteins using a graph to represent the helices and strands was proposed in [23]. An approach based on maximally common substructures between two proteins was proposed in [13]; it also highlights areas of structural overlap between proteins. SUBDUE [8] is an approach based on Minimum Description Length for finding patterns in proteins. Actually, the only work we are aware of in terms of parallel substructure analysis is the parallel implementation of the Subdue algorithm [10]. The authors propose three approaches for parallelization based on computational distribution, static data distribution and a dynamic data distribution. However, they find that linear speedup could not be achieved. Another graph based method for structure discovery, based on geometric hashing, was presented in [37]. Recent work on graph data mining is also related to this effort [24, 39, 25, 30, 23, 13, 8, 37].

**MotifMiner Toolkit** To address the needs of such applications, we have recently developed a prototype toolkit, MotifMiner[27, 32, 6, 7], that detects frequently occurring structural motifs. We have successfully conducted a fairly in depth evaluation of the MotifMiner toolkit on various datasets ranging from pharmaceutical data[6] to tRNA data, from protein data (from the PDB)[32, 6] to data from molecular dynamics simulations[5, 16].

Given a graphical representation of a lattice or molecule, MotifMiner represents the information between a pair of nodes (atoms), $A_i$ and $A_j$, in a *mining bond*. The mining bond $M(A_iA_j)$ is a 3-tuple of the form:

$$M(A_iA_j) = \{A_i type, A_j type, AttributeSet(A_iA_j)\}$$

A k-atomset is a substructure containing k connected (within range) atoms, and is represented as:

$$X = \{\mathbf{S_X}, A_1, A_2, \ldots, A_k\},$$

where $A_i$ is the $i^{th}$ atom and $\mathbf{S_X}$ is the set of mining bonds describing the atomset. By defining pairs of atoms with mining bonds, the graph is completely represented, such that two atomsets $X$ and $Y$ are considered to be the same substructure if $\mathbf{S_X} = \mathbf{S_Y}$.

Our technique for detecting frequent structural motifs relies on 4 key techniques: 1) *Range pruning* to limit the search for viable strongly connected sub-structures, 2) *Candidate pruning*[1], for pruning the search space of possible frequent structures (based on a novel technique that identifies extremal substructures[7][1]), and most importantly 3) *Recursive Fuzzy Hashing* for rapid matching of structures (to determine frequency of occurrence), 4) *Distance binning and resolution* to work in conjunction with Recursive Fuzzy Hashing to deal with noise in the input data. Range pruning and Candidate pruning reduce the candidate search space thereby reducing the memory footprint and improve the scalability of the algorithm significantly. Recursive fuzzy hashing, which is similar in principle to geometric hashing[26, 38], was designed to efficiently handle noise effects in data. It is extremely effective in capturing relevant substructures in protein datasets[32] (noisy). To further handle noisy datasets MotifMiner relies on *resolution* based distance binning. Essentially, the raw Euclidean distance between two atoms is discretized by binning; This task is accomplished by choosing a *resolution* value and dividing the inter-atom distance into equiwidth bins based on this value, represented efficiently as bits in the mining bond. Binning of the data simplifies calculations and helps MotifMiner handle *minor* fluctuations in distance. These steps are discussed in more detail in ensuing sections when describing the parallel algorithms.

## 3. ALGORITHMS

for the rest of the section. In this section we define the two problems to be examined, intra-structure and inter-structure mining. We then present our solution to the intra-structure mining problem, called intra-structure p-MotifMiner. Finally, we give our solution to the inter-structure mining problem in the form of intra-structure p-MotifMiner. This solution uses the solution to the intra-structure mining problem to solve the inter-structure problem.

### 3.1 Problem statement

- **Intra-Structure Frequent Motif Mining Problem:** The goal is to identify all motifs that occur frequently in a single molecule. The intra-structure *support* of a motif is the number of atomsets of the motif within the molecule. A motif is frequent if its

[1]Extremal atoms for a given k-atom structure are the two (or more) atoms that are furthest away from each other within the structure. These atoms are used to identify extremal substructures.

```
Input: $\mathbf{L_k}$, set of frequent $k$-atomsets
Output: $\mathbf{C_{k+1}}$, set of candidate $k+1$-atomsets
1. for all atomsets $i$ in $\mathbf{L_k}$
2.   for all atomsets $j$ in $\mathbf{L_k}$ s.t. $j > i$ and $j$ and $i$ have
     k-1 spatially matched and common atoms,
     and the two different atoms are extremal
3.     Create candidate that combines $i$ and $j$ and add to
       $\mathbf{C_{k+1}}$
```

**Figure 1: Simplified candidate atomset generation algorithm**

intra-structure *support* is greater than or equal to a user-defined support threshold (*minsup*).

- **Inter-Structure Frequent Motif Mining Problem:** The goal is to identify all motifs which are frequent across a set of molecules. The inter-structure *support* of a motif is the number of the molecules containing the motif, and a motif is frequent if its inter-structure *support* is greater than or equal to a user-defined inter-structure support threshold ($s_1$). We can generalize this problem by introducing the parameter of intra-structure support threshold($s_2$), which specifies a minimum intra-structure support constraint for a motif to become an inter-structure frequent motif. Thus, the inter-structure motif mining problem can be restated as: given the inter-structure support threshold ($s_1$) and intra-structure support thre- shold($s_2$), identify all motifs whose inter-structure support and intra-structure support is greater than or equal to $s_1$ and $s_2$ respectively.

## 3.2 Intra-structure p-MotifMiner

Like the well-known Apriori association mining algorithm[1] for market basket analysis, MotifMiner is a level-wise algorithm. During iteration $k$, we generate $C_k$, the set of candidate $k$-atomsets from the frequent atomsets in the previous iteration ($L_{k-1}$) and then prune infrequent atomsets from $C_k$ until we get $L_k$, the set of frequent $k$-atomsets. The key innovation in the candidate generation phase is the use of extremal atomsets, and the key innovation in the pruning phase is the use of a recursive fuzzy hash tree for performance reasons and to handle noise effects in data[6, 32].

For the purpose of basic parallelization we consider the two phases of MotifMiner independently and describe how to parallelize each phase. Then we describe how to integrate these two phases.

### *3.2.1 Parallel candidate generation algorithm*

We first look at a simplified version of the sequential candidate atomset generation algorithm in Figure 1. A critical feature of this algorithm is that we generate and maintain all instances (atomsets) of motifs in the candidate generation phase. This is indispensable to the subsequent pruning phase, since Apriori-style support counting, which evaluates whether a spatial pattern is found in a dataset, will be computationally infeasible (given the well known issues with subgraph isomorphism). The exact algorithm details can be found elsewhere[6, 32].

From the perspective of parallelization, this component easily lends itself to a divide and conquer style approach.

```
Input: $\mathbf{L_k}$, set of frequent $k$-atomsets
Output: $\mathbf{C_{k+1}}$, set of candidate $k+1$-atomsets
/*Initialization phase */
1. Determine the number of compute processors
2. Determine the label of the calling processor

/* Asynchronous phase */
3. Partition the iteration space amongst the processors.
   For each processor
4.   Compute_Local_$C_{k+1}$($\mathbf{L_k}$);

/* Final gather phase */
   For each processor
5.   Gather the results of local candidate $k+1$-atomsets
     from other processors
6.   Construct the global set of candidate $k+1$-atomsets,
     i.e. $\mathbf{C_{k+1}}$
```

**Figure 2: Parallel candidate generation algorithm**

```
Input: $\mathbf{C_k}$, set of $k$-candidate atomsets;
Output: $\mathbf{L_k}$, set of frequent $k$-atomsets;
1. $m = 1$     ; Examine first mining bond
2. $\mathbf{C}$ = candidateAtomsets     ;
/*Start by examining all candidate atomsets*/
3. for all $atomset_i$ in $\mathbf{C}$
4.    hash($M_m, M_m + 1, M_m - 1$ in $atomset_i$)
5. for all $\mathbf{H_i}$ in hash_bins
6.    remove $\mathbf{H_i}$ if $|\ \mathbf{H_i}\ | <$ minSupport
7. $m = m + 1$
8. while $m \leq\ |\ \mathbf{M}\ |$
9.    for all $\mathbf{H_i}$ as $\mathbf{C}$
10.      goto step 3
11. frequentAtomsets = $\mathbf{H_1} \cup \mathbf{H_2} \ldots \mathbf{H_n}$
```

**Figure 3: Sequential recursive fuzzy hash pruning algorithm**

Given a total of $m$ frequent $k$-atomsets, we will have to evaluate exactly $\binom{m}{2}$ possible candidates. Since we use extremal atomsets to generate candidate atomsets[32], it is guaranteed that there will be no repeated candidate atomsets occurring during the generating process. Thus, we can partition the iteration space, which is an upper triangle matrix, into independent subspaces in a straightforward fashion. $Row_i$ of the matrix is assigned to $processor_{i \ mod \ p}$, where p is the number of processors involved. In practice, such a partitioning scheme works quite well. The parallel candidate generation algorithm is shown in Figure 2.

In the initialization phase, each processor gets information about its own identity and number of processors involved. This information is used to partition the iteration space. In the asynchronous phase, each processor takes care of its own portion and finds local set of candidate atomsets, $local\_C_{k+1}$. In the final gather phase, each processor exchanges its local set of candidate atomsets with all other processors. After the exchanging process, all processors get a consistent view of the global set of candidate atomsets, i.e., $C_{k+1}$.

### *3.2.2 Parallel Recursive Fuzzy Hashing(RFH) pruning algorithm*

In MotifMiner, to effectively address the issue of coordinate noise when counting frequency of substructures to

```
Input: $\mathbf{C_k}$, set of $k$-candidate atomsets;
Output: $\mathbf{L_k}$, set of frequent $k$-atomsets;
/*Initialization Phase*/
1. determine the number of compute processors
2. Determine the label of the calling processor

/*Pre-filter phase*/
3. Hash atomsets of $\mathbf{C_k}$ into bins based on atom types
   only. Each bin contains the atomsets consisting of
   the same types of atoms
4. Remove atomsets whose corresponding bin count is
   below the support threshold.

/*BFS extension phase */
5. Breadth first expansion of the first few levels of the tree
   to get more fine-grained work units, i.e., H($\mathbf{H_1}$, $\mathbf{H_2}$,...)

/*Asynchronous RFH pruning phase*/
6. Partition the pruning space, each compute processor
   will be assigned a subset H'of H
   For each processor
7.     Compute_Local_$L_k$(H')

/*Final gather phase*/
   For each processor
8.     Gather results of local frequent atomsets from
        other processors
9.     Construct $\mathbf{L_k}$, global set of frequent $k$-atomsets
```

**Figure 4: Parallel RFH pruning algorithm**

enable pruning of candidates, we allow for the specification of a fuzziness parameter. The fuzziness parameter specifies to what extent two different atomsets can be thought as belonging to the same motif. In such a case, the difference between these two atomsets is attributed to noise effects. Fuzzy analysis is significantly more expensive than the standard exact match analysis [6, 32].

The sequential RFH algorithm for candidate pruning is described in Figure 3. RFH analysis produces a top-down hash tree of height k, with the root node representing the entire set of candidate atomsets for k. At each level, starting from the root node, we recursively split the atomsets into child hash bins based on the atomsets' mining bond for that particular tree level. When the fuzziness parameter is set to 1, we hash both to the exact hash location and +/-1 resolution unit. Thus, the larger the allowable fuzziness, the larger the storage costs of the tree. Hash bins whose atomset count is less than the support threshold are pruned from further consideration. At the end of this procedure the atomsets in each bin are evaluated and the motifs at level k are identified.

Effectively, in our pruning phase there is a need to traverse the whole hash tree, pruning the infrequent nodes of the tree. We propose a parallel strategy that requires some redundant work for the sake of identifying large enough independent pieces that balances the computation among processors. Specifically we propose the following algorithm:

- 1. In an initial data partitioning phase, the root node information is broadcasted to all processors, which redundantly expand the first few tree levels (k) in a breadth first fashion. The nodes comprising these first few levels are henceforth referred to as *frontier* nodes. Neither communication nor synchronization is required in this phase. At the end of this phase the frontier nodes are divided among the processors using a predefined hash function (simple or bitonic (described later)).
- 2. In the main asynchronous search phase, each processor $P_i$ starts expanding the sub-trees comprising its frontier nodes $n_i$, $n_{p+i}$, $n_{2p+i}$,... in a depth first fashion, where p is the number of processors involved.

Essentially, once the first phase completes, each processor can traverse its assigned portion of hash tree independently. Since every node in the tree actually is a bin of atomsets, if its support is below the support threshold, this node will be removed from the tree. When we have processed the whole tree, the remaining leaf nodes are the frequent bins, with each corresponding to a motif for the current level. We expect this dynamic partitioning scheme to produce fairly even work distribution.

Figure 4 shows the parallel RFH pruning algorithm. First, each processor obtains the information about its identity and the number of compute processors involved. This information will be used in the hash function for partitioning the work. In the pre-filter phase, we hash atomsets into bins based only on their atom types. Consequently, each bin contains the atomsets that share the same types of atoms. Then we remove the bins whose cardinality is below the support threshold, since they cannot possibly contain any frequent atomsets. Next, we expand the first few levels of the hash tree in a breadth first manner, to get more fine-grained work units(i.e., $\mathbf{H_1}$, $\mathbf{H_2}$,...). Then subtrees, representing each work unit, are assigned amongst processors and each processor independently processes its portion to find local frequent atomsets. Finally, each processor gathers the local frequent atomsets from other processors and constructs the global set of frequent atomsets, $L_k$.

### 3.2.3 *Contracting motifs*

Another notable aspect is that when we partition the search space across multiple processors, each processor calculates the motif set for its own portion. Due to the allowance of fuzziness, it is possible that two processors will return the same motif in terms of allowable fuzziness. Thus, we must remove this kind of redundancy. When each processor collects the local information and constructs the global motif set at each level, it needs to do some extra processing work and contract the motif set. If two motifs are the same due to fuzziness, one of them will be removed. Note that such an operation on the motif set will not affect the set of frequent atomsets(instances of motifs). Experimental data show that the cost of this step is negligible (usually less than one second).

### 3.2.4 *Complete intra-structure p-MotifMiner algorithm*

We combine the candidate generation phase and pruning phase to get the complete algorithm of intra-structure p-MotifMiner in Figure 5. During iteration $k$, we apply the parallel candidate generation algorithm to generate $C_k$, the set of candidate $k$-atomsets and then apply the parallel RFH pruning algorithm to get $L_k$, the set of frequent $k$-atomsets. We also need to contract motifs at the end of each iteration to remove repeated motifs. This process is repeated until there are no new frequent atomsets.

In practice we start parallel processing from $k = 3$, rather than from $k = 2$ because the generation of frequent 2-atomsets is trivial compared to operations once $k \geq 3$.

## 3.3 Inter-structure p-MotifMiner

This algorithm is obtained by extending intra-structure p-MotifMiner to deal with multiple proteins. The basic idea is that we run intra-structure p-MotifMiner for each protein in a pipelined fashion. Assuming we have $n$ proteins, $P_1$, $P_2$, ..., $P_n$, during the $k^{th}$ iteration of the algorithm, we apply intra-structure algorithm to generate $L_k$(we use intra-

```
1. Prune infrequent atoms(1-atomsets)
2. Generate $\mathbf{C_2}$, candidate 2-atomsets from frequent atoms
3. Prune infrequent 2-atomsets to get $L_2$
4. $k = 3$
5. while (| frequent (k-1)atomsets |> 0)
6.     Partition the candidate generation amongst the processors
       For all processors
7.         Compute local set of candidate $k$-atomsets, i.e. Local_$C_k$
8.         Gather results of local candidate $k$-atomsets from other processors
9.         Compute global set of candidate $k$-atomsets, i.e. $\mathbf{C_k}$

10.    Breadth first expand the first few levels of RFH tree
11.    Partition the pruning space amongst processors
       For all processors
12.        Compute local set of frequent $k$-atomsets, i.e. Local_$L_k$
13.        Gather results of local frequent $k$-atomsets from other processors
14.        Compute global set of frequent $k$-atomsets, $\mathbf{L_k}$ on each processor
15.        Contract motifs
16.    $k = k + 1$
```

**Figure 5: Complete intra-structure p-MotifMiner algorithm**

```
Input: a sequence of $n$ proteins
inter-structure support threshold $s_1$
intra-structure support threshold $s_2$
Output: F, the set of inter-structure frequent motifs
1. $k = 2$
2. while (there is new frequent motifs emerging)
4.     For each protein $P_i$
5.         Apply intra-structure algorithm to compute $L_k$(i)
           ; we use $s_2$ when pruning

6.     Compute U = $\mathbf{L_k(1)} \cup \mathbf{L_k(2)} \ldots \mathbf{L_k(n)}$
7.     For each motif e in U
8.         Determine inter-structure support for e
9.         if e.support $> s_1$
10.            add e to $F$

11.    For each protein $P_i$
12.        Prune infrequent motifs from $L_k$(i)

13.    $k = k + 1$
```

**Figure 6: Inter-structure p-MotifMiner algorithm**

structure support threshold $s_2$ when pruning) for all proteins in turn, $P_1$ first, then $P_2$, $P_3$,... and so on, until we have processed $P_n$, at which point we have the global data set of $L_k$ for all proteins. We are then able to determine the inter-structure support for each motif in $\bigcup_{i=1}^{n} L_k(i)$ and identify all inter-structure frequent motifs for level $k$(we use inter-structure support threshold $s_1$ when pruning). Then based on this information we prune the infrequent motifs for all proteins in turn again, $P_1$ first, then $P_2$, $P_3$... and so on, until we have processed $P_n$, at which point we move on to the next iteration. This process is repeated until there are no new inter-structure frequent motifs. The algorithm is presented in Figure 6.

Considering that we are always more interested in larger motifs, thus when applying inter-structure p-MotifMiner, we can give some priority to larger motifs by decreasing the intra-structure support threshold $s_2$ as we move on to the next iteration until $s_2$ reaches some lower bound, e.g. 1.

## 4. OPTIMIZATIONS

Since inter-structure p-MotifMiner is derived from the intra-structure algorithm, its performance relies heavily on the latter. In this section we propose several optimizations to improve the performance of intra-structure p-MotifMiner.

### 4.1 More fine-grained work partition

When we raise the fuzziness, normally the pruning phase will dominate the whole analysis process. The key point to improve the performance of pruning is to partition the work as evenly as possible, and this becomes more important considering there is no load balancing at a later stage. We present two approaches to obtain more fine-grained work partitions.

- 1. To achieve a good level of load-balancing, the bins to be analyzed are assigned to the processors using the scheme of bitonic partitioning. The bitonic scheme uses a greedy algorithm. First, we sort all the bins on $w_i$. Here, $w_i$ represents the number of atomsets within $bin_i$, which is a workload indicator. Next, the bin with the maximum $w_i$ is extracted and assigned to $processor_0$. The second largest workload bin is assigned to $processor_1$. If only two processors are being used, $processor_1$ (once again) gets assigned the third largest workload task and so on. Essentially, bitonic partitioning takes a sub-sorting process within each round, such that, if a processor gets a smaller portion in this round, then it will be given a bigger portion in the next round and vice versa, thereby enabling a globally better assignment.
  The experimental data shows that bitonic partitioning is usually helpful, though the performance gain is not significant compared to a simpler partitioning scheme in which we only sort bins without sub-sorting. For all experiments we conducted, except for the experiments used to show the effect of bitonic partitioning, we have used this optimization.

- 2. We can achieve more fine-grained work partitions using more complex techniques, as we expand the first few levels of the RFH tree. In a default *2-level expansion scheme*, when dealing with $C_3$, all processors will expand the first level of the hash tree redundantly. Subsequently, $C_4$ and beyond, all processors will expand the first two levels of the hash tree. We modified the second step as follows: when pruning $C_4$, in which case the depth of the RFH tree is $\binom{4}{2}$=6, all processors expand the hash tree for the first 2 levels. When pruning $C_k(k > 4)$, in which the depth of the hash tree is at least $\binom{5}{2}$=10, all processors expand the hash tree for the first 3 levels redundantly. Beyond this we account for the number of compute processors involved in order to determine if further expansion is warranted. We title this optimization as the *self-adaptive expansion scheme.*

### 4.2 Dynamic duplicate screening (DDS)

Dynamic Duplicate Screening (DDS)[6], is an efficient approach to reduce the search space in RFH tree. Basically, the idea here is to reduce redundant computation by recognizing situations when two tree nodes or bins (typically neighboring) contain exactly the same atomsets. On recognizing this event the algorithm can remove one of the bins to avoid redundant calculations. We have shown that DDS can save significant time and memory[32], at no cost to accuracy.

In the parallel version, we execute the DDS analysis locally on each processor, in an independent manner. When a processor is calculating its local_$L_k$, it takes DDS analysis at each level of the RFH tree, the set of the atomset bins are analyzed, and the duplicates are discarded from further consideration. The tradeoff here is that we lose some pruning ability but we are able to avoid unnecessary synchronization costs. Experimental evaluation indicates that the additional prunability gained is insufficient to warrant the extra synchronization overhead.

## 4.3 Reordering mining bonds and backward pruning

In our representation of an n-atomset, we have to use $\binom{n}{2}$ items to represent the mining bond information. When $n > 10$, $\binom{n}{2}$ will be greater than 45, which means the depth of the corresponding RFH tree will be greater than 45. Also, due to the allowance for fuzziness, the search space will blow up easily. In our original sequential algorithm, when we represent an atomset, we sort its mining bonds in a static fashion – lexicographical increasing order. However, if we dynamically re-order these mining bonds in the order of increasing occurrence within a given structural dataset we observe significant improvements in performance. For example, in proteins, C-C bonds are most frequent and thus will be ordered after C-O bonds. With this re-ordering, the infrequent candidates will be pruned very quickly. The idea here is very similar to dynamic reordering in MAFIA[3]. Experimental results (described next) show that usually we can get a remarkable five-fold improvement in execution time using this strategy.

## 5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed algorithms and optimizations.

## 5.1 Experimental Setup

The parallel algorithms were implemented using MPI [2]. The cluster on which we conducted most of our experiments has 16 compute nodes, each with one 1GHz PIII processor, 256 KB of secondary cache, 512MB RAM and was running Linux kernel 2.4.18. We evaluate the performance of our algorithms on the following queries:

- **Query 1** is an intra-molecule frequent substructures analysis query on protein lysozyme(PDB ID: 193L).

- **Query 2** is an intra-molecule frequent substructures analysis query on protein hemoglobin(PDB ID: 1BZ0). In terms of performance requirements, we found that this query is remarkably similar to processing a single frame of a 512 × 512 × 512 $Si$ MD simulation dataset[16]. Thus, performance results obtained on Query 2 are representative of performance obtained when processing a single frame of the MD simulation dataset.

- **Query 3** is an intra-molecule frequent substructures analysis query for the phenylalanine tRNA of yeast(PDB ID: 1EVV) that finds a novel substructure[6] in the codon regions.

**Table 1: Iteration space partitions for candidate atomset generation (Query1)**

| Level | Processor 1 | Processor 2 | Processor 3 | Processor 4 |
|---|---|---|---|---|
| 3 | 22223 | 21876 | 21809 | 21745 |
| 4 | 53166 | 53709 | 53433 | 53827 |
| 5 | 6881 | 6681 | 6795 | 6928 |
| 6 | 159 | 173 | 174 | 159 |
| 7 | 18 | 13 | 13 | 16 |

**Table 2: Iteration space partitions for candidate atomset generation (Query2)**

| Level | Processor 1 | Processor 2 | Processor 3 | Processor 4 |
|---|---|---|---|---|
| 3 | 23493 | 23820 | 23673 | 23251 |
| 4 | 58547 | 58709 | 58364 | 58514 |
| 5 | 9092 | 9320 | 9114 | 8977 |
| 6 | 578 | 634 | 598 | 622 |
| 7 | 22 | 31 | 19 | 27 |

- **Query 4** is an inter-molecule frequent substructure analysis query operating on the proteins with PDB IDs: 193L, 1BZ0, 1I9V and 1FIR.

## 5.2 Intra-structure p-MotifMiner

### 5.2.1 *Candidate atomset generation*

In Section 3, we discussed a simple approach to partition the iteration space for candidate atomset generation. Table 1 and Table 2 present the number of candidate atomsets generated at each level by each processor, when using 4 processors, for Query 1 and Query 2 respectively. The parameters for Query 1 and Query 2 are (support=70, fuzziness=2, resolution=0.06) and (support=100, fuzziness=2, resolution=0.06) respectively. The number of candidate atomsets generated serves as an estimate of the workload of a processor. From the tables, we see that the partitioning approach provides good load balancing. The number of candidate atomsets generated by each processor are almost equal.

### 5.2.2 *Basic experimental results*

Figure 7 and Figure 8 give the response time for Query 1

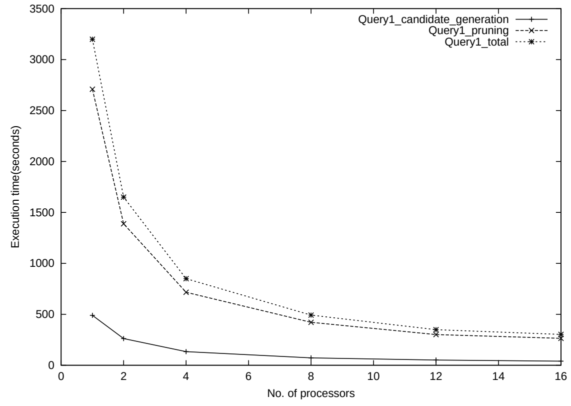

**Figure 7: Response time for query 1, without optimization**

[2]http://www-unix.mcs.anl.gov/mpi

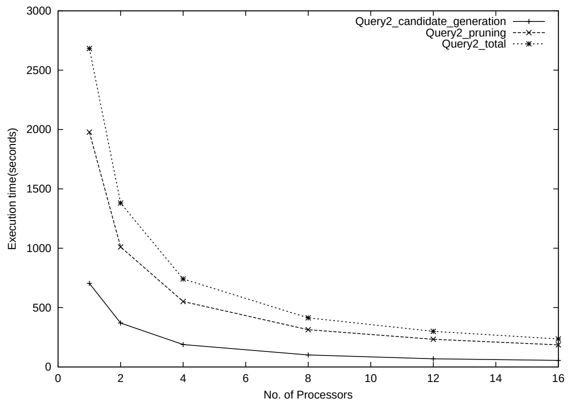

**Figure 8: Response time for query 2 without optimization**

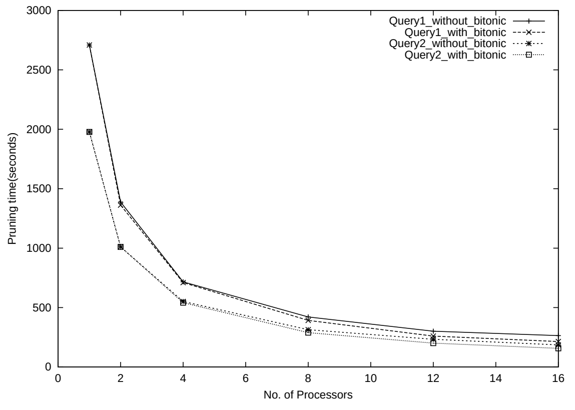

**Figure 9: Pruning time with/without bitonic partitioning**

and Query 2 respectively. The parameters for Query 1 and Query 2 are (support=70, fuzziness=2, resolution=0.06) and (support=100, fuzziness=2, resolution=0.06) respectively. We used our 2-level expansion scheme for the RFH tree. No other optimizations were applied.

Results show that we get reasonably good speedup for both the candidate atomset generation phase and the pruning phase upto 16 processors. Specifically, consider Query 1. On 4 and 8 processors, the speedups are 3.7 and 6.8 for the candidate generation phase, and 3.8 and 6.4 for the pruning phase, respectively. The overall speedups (combining both pruning and candidate generation phases) are 3.8 and 6.5 on 4 and 8 processors. Note, when we increase the number of processors used, the efficiency reduces. For instance, the efficiency on 8 processors is 0.81, but on 16 processors, the efficiency falls to 0.66. We can attribute this to the fact that as more processors are involved, the work load imbalance may increase.

### 5.2.3 *Bitonic partitioning*

Here, we specifically focus on pruning time as it is the only phase affected by bitonic partitioning. Figure 9 presents the pruning time for Query 1 and Query 2 with and without the bitonic partitioning optimization. For Query 1, speedup with the optimization is 6.9 on 8 processors, while

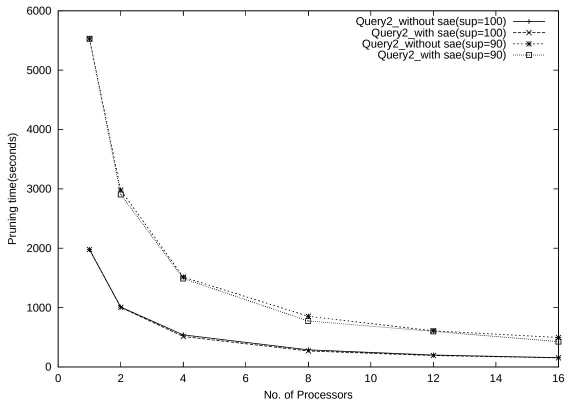

**Figure 10: Pruning time with/without the self-adaptive expansion scheme**

the speedup without this optimization is 6.4. Moreover, the benefit from bitonic partitioning becomes more prominent as we increase the number of processors. For example, on 16 processors, the speedup with bitonic partitioning is 12.7. In contrast, the speedup without the optimization is only 10.3. Note that all subsequent experiments will use this optimization.

### 5.2.4 *Self-adaptive expansion scheme*

Like bitonic scheduling, the self-adaptive expansion scheme only affects pruning time. Figure 10 presents the pruning time for Query 2 with the 2-level expansion and the self-adaptive expansion scheme described in Section 4. For Query 2, with support=100, on 8 processors, the speedup is 7.4 (1978s to 269s), which is better than that without the optimization, which is 6.8 (1978s to 289s). Specifically, the improvement becomes more prominent when pruning is expensive. For example, when we lower the support threshold for Query 2 from 100 to 90, the speedup with the optimization on 8 processors is 7.2. In contrast, the speedup without the optimization is only 6.5. This is attributed to the fact that when the RFH tree is deep, the cost of redundantly expanding the first few levels of the RFH tree is outweighed by the benefits of fine-grained partitioning.

### 5.2.5 *Fuzziness*

As pointed out by Parthasarathy and Coatney [32], the fuzziness value can increase the computational complexity significantly. Given different fuzziness values, MotifMiner behaves differently. When fuzziness is 0, in which case we only allow exact matches between atomsets, candidate generation will dominate the computation time. In this case, improved execution time for intra-structure p-MotifMiner is mainly due to the exploitation of parallelism in candidate generation. Similar results hold when fuzziness equals 1. But when fuzziness is greater than or equal to 2, often the pruning phase will dominate the computation time. In such a situation, the performance gain for p-MotifMiner mainly comes from the exploitation of parallelism in pruning. Figure 11 presents the response time for Query 1 with fuzziness = 1. Here, the performance gain mainly comes the candidate generation phase. In comparison, Figure 7 shows that when fuzziness is 2, the performance gain mainly comes from the pruning phase.

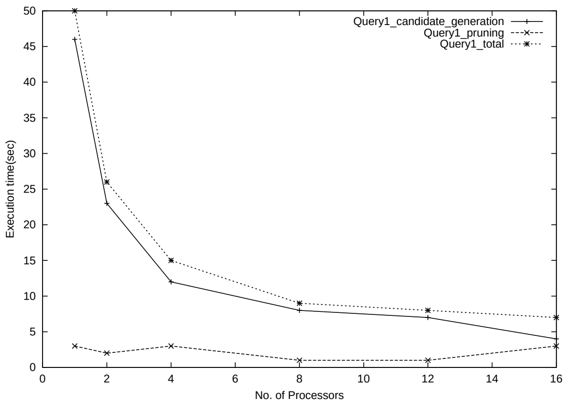

**Figure 11: Response time for Query 1(fuzziness=1,support=70,resolution=0.06)**

### 5.2.6 *Resolution*

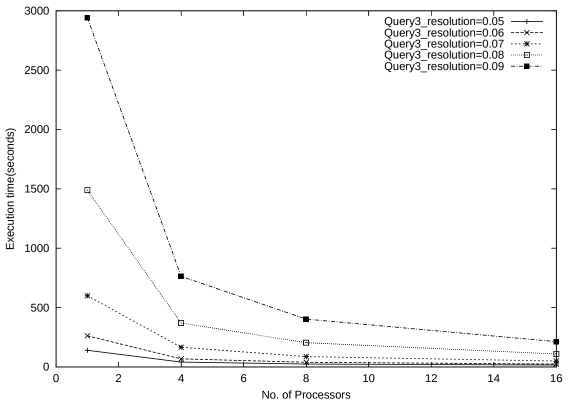

**Figure 12: Varying the resolution for Query 3(support=70, fuzziness=1)**

Resolution is the width of the bins when we discretize the distance between the atoms. It directly affects the size of the search space for both candidate generation and pruning. As we increase the resolution, the computational complexity increases significantly. Figure 12 presents the response time for Query 3 with different resolutions. We see that intra-structure p-MotifMiner demonstrates good speedup as we increase the resolution. For example, when the resolution is set to 0.09, the speedup on 8 processors and 16 processors is 7.3 and 13.8 respectively.

### 5.2.7 *Support*

As we lower the support threshold, the number of structural motifs that need to be processed increase. Intra-structure p-MotifMiner demonstrates good speedup in this case as well. This is evident in Figure 13. For instance, when support threshold is 90, the speedup on 8 and 16 processors is 7.2 and 12.9 respectively. Moreover, when support threshold is 80, sequential analysis does not finish even after 12,000 seconds, at which time memory is exhausted. In contrast, intra-structure p-MotifMiner finished the analysis in 1975 seconds when 16 processors were used. This validates parallel MotifMiner's ability to handle more challenging queries when compared to serial MotifMiner.

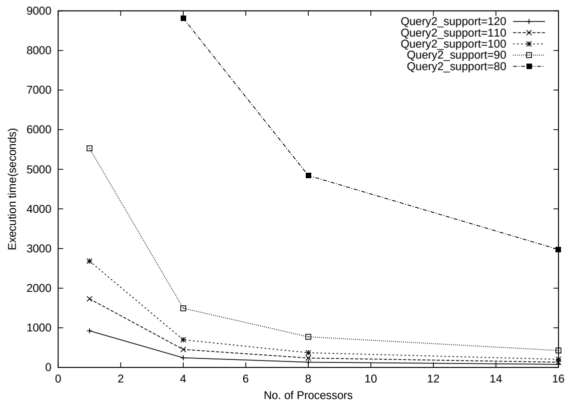

**Figure 13: Varying the support threshold for Query 2(resolution=0.06, fuzziness=2)**

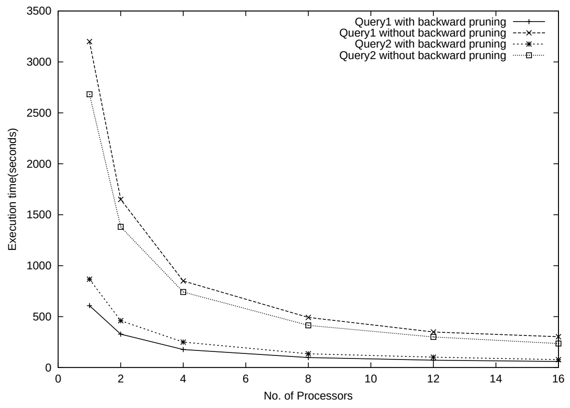

**Figure 14: Response time with/without backward pruning for Query 1(support=70, fuzziness=2, resolution=0.06) and Query 2(support=100, fuzziness=2, resolution=0.06)**

### 5.2.8 *Backward pruning*

Figure 14 presents the response time when backward pruning is used. We see an impressive improvement in execution time as a result of backward pruning. Take Query 1 as an example. It took 492 seconds on 8 processors without backward pruning. In contrast, it only took 98 seconds with the optimization, in which case we achieve a speedup of 5. Furthermore, backward pruning also improves sequential algorithm performance. It took 3200 seconds to answer Query 1 using original un-optimized sequential algorithm. In contrast, it only took 608 seconds with the optimization. In addition, since backward pruning reduces the search space , the memory usage also decreases, thus allowing us to support more challenging queries

### 5.2.9 *Scalability*

To demonstrate the scalability of intra-structure p-MotifMiner on larger clusters, we executed Query 1 and Query 2 on IA32 cluster at OSC(Ohio Supercomputing Center). The maximum number of compute nodes available were 112. Each compute node has dual 2.4GHz Intel P4 Xeon processors (we

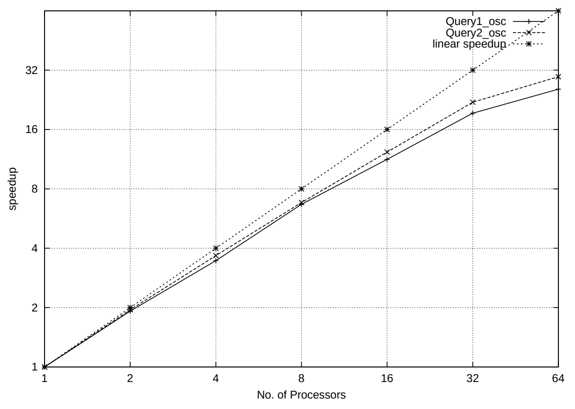

**Figure 15: Speedup for Query 1 and Query2 on IA32 cluster**

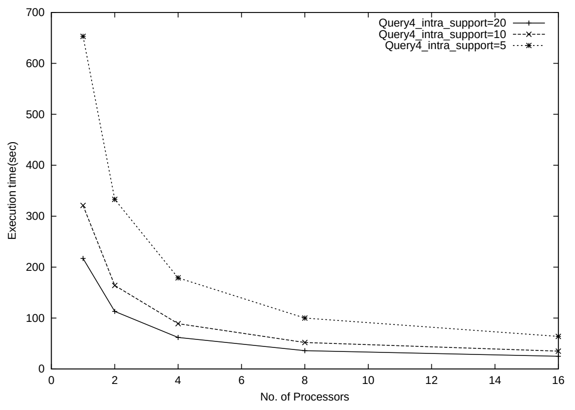

**Figure 16: Response time for Query 4 when varying intra-structure support threshold, inter-structure support threshold $s_1$= 2**

only use one of them), each with 512KB of secondary cache, 4GB RAM, running Linux. Figure 15 presents the speedup curves for these two queries. The ideal linear speedup curve, which is a straight line, was drawn to facilitate comparison. From the experimental results, we see that intra-structure p-MotifMiner scales moderately well on larger clusters. Not surprisingly, the efficiency decreases as more processors are added. As can be seen in the figure, the speedup curves tend to deviate from the linear speedup line as we use more processors. Take Query 2 for example. The efficiencies on 8, 16 and 32 processors are 0.85, 0.77 and 0.69 respectively. The reason for this decrease in efficiency is that there is insufficient computation to warrant such a large cluster for such queries. However, we suspect that for more advanced queries, such clusters would indeed be useful.

## 5.3 Inter-structure p-MotifMiner

The results obtained for inter-structure p-MotifMiner are extremely similar to those for intra-structure p-MotifMiner. Hence, we just present one experiment using Query 4 on the local cluster.

Figure 16 presents the response time when varying intra-structure support $s_2$. As can be seen in the figure, we achieve good speedup using this algorithm to solve the inter-structure motif mining problem. For example, to answer Query 4 with intra-structure support threshold = 5, the sequential analysis took 653 seconds. In contrast, the parallel analysis on 16 processors only took 64 seconds, a speedup of 10.2. Table 3 lists the number of frequent motifs discovered when varying intra-structure support threshold during the analysis. In the experiment, when we lower the intra-structure support, we find larger frequent motifs. For example, when $s_2$ is 20, we find maximum 3 frequent 5-motifs, when $s_2$ is 10, we find maximum 1 frequent 6-motif, and when $s_2$ is 5, we find maximum 6 frequent 7-motifs.

**Table 3: No. of frequent motifs discovered, $s_1$=2**

| Motif Size | $s_2$=20 | $s_2$=10 | $s_2$=5 |
|---|---|---|---|
| 2 | 204 | 244 | 268 |
| 3 | 171 | 657 | 2243 |
| 4 | 42 | 248 | 1180 |
| 5 | 3 | 26 | 457 |
| 6 | 0 | 1 | 104 |
| 7 | 0 | 0 | 6 |

## 6. CONCLUSIONS AND FUTURE WORK

In this article, we present parallel algorithms for mining structural motifs in scientific data. While such algorithms have proven to be extremely useful in their ability to identify novel substructures, the algorithms themselves are quite time consuming. There are two reasons for this: i) inherently, the algorithm suffers from the curse of subgraph isomorphism; and ii) the algorithm needs to effectively handle noise effects (in protein structure data).

To address these problems, in this paper we propose parallelization strategies in a cluster environment for said algorithms. We identify key optimizations such as bitonic scheduling, self-adaptive frontier node expansion and backward pruning that handle problems associated with load imbalance, scheduling and communication overheads respectively. Our results show that the optimizations are quite effective and we are able to obtain good speedup for moderately sized cluster environments.

In terms of future work, we are evaluating the use of a graph partitioning schemes to improve the performance of the algorithms even further[10]. We are also interested in adapting our algorithm to process dynamic structural datasets such as those obtained from an MD simulation[16].

**Acknowledgments:** We thank the anonymous reviewers for their comments and suggestions. We would also like to thank H.Yang, A.Ghoting and K.Marsolo for their comments on initial drafts of this paper.

## 7. REFERENCES

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th VLDB Conf.*, September 1994.

[2] C. Borgelt and M. Berthold. Mining molecular fragments: Finding relevant substructures of molecules. In *IEEE International Conference on Data Mining*, Dec 2002.

[3] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. In *IEEE International Conference on Data Engineering*, 2001.

[4] L. P. Chew, D. Huttenlocher, K. Kedem, and J. Kleinberg. Fast detection of common geometric substructures in proteins. *RECOMB*, 1999.

[5] M. Coatney, S. Mehta, A. Choy, S. Barr, S. Parthasarathy, R. Machiraju, and J. Wilkins. Defect detection in silicon and alloys. In *IEEE Workshop on Visualization in Bioinformatics and Cheminformatics*, 2002.

[6] M. Coatney and S. Parthasarathy. Motifminer: A general toolkit for efficiently identifying common substructures in molecules. In *IEEE International Conference on Bioinformatics and Bioengineering (to appear)*, 2003.

[7] M. Coatney and S. Parthasarathy. Motifminer: Efficient discovery of common substructures in biochemical molecules. In *Knowledge and Information Systems, to appear*, 2003.

[8] D.J. Cook, L.B. Holder, S. Su, R. Maglothin, and I. Jonyer. Structural mining of molecular biology data. *IEEE Engineering in Medicine and Biology*, 20(4):67–74, 2001.

[9] L. Dehaspe, H. Toivonen, and R. King. Finding frequent substructures in chemical compounds. In *The Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1998.

[10] D.J.Cook and et al. L.B.Holder. Approaches to parallel and distributed computing. *Journal of Parallel and Distributed Computing*, 2001.

[11] S. Djoko, D. Cook, and L. Holder. Analyzing the benefits of domain knowledge in substructure discovery. In *The First ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1995.

[12] U. Fayyad et al. Mining Scientific Data. In *Communications of the ACM*, pages 51–57, 1996.

[13] H.M. Grindley, P.J. Artymiuk, D.W. Rice, and P. Willett. Identification of tertiary resemblence in proteins using a maximal common subgraph isomorphism algorithm. *J. of Mol. Biol.*, 229(3):707–721, 1993.

[14] E. Han, G. Karypis, and V. Kumar. Data mining for turbulent flows. In *Data mining for scientific and engineering applications*, pages 239–256, 2001.

[15] D. Hand, H. Mannila, and P. Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, 2001.

[16] M. Jiang, T.-S. Choy, S. Mehta, M. Coatney, S. Barr, K. Hazzard, D. Richie, S. Parthasarathy, R. Machiraju, D. S. Thompson, J. Wilkins, and B. Gatlin. Feature Mining Algorithms for Scientific Data. In *Proceedings of SIAM Data Mining Conference*, May 2003.

[17] I. Jonassen, I. Eidhammer, D. Conklin, and W. Taylor. Structure motif discovery and mining the pdb. In *German Conference on Bioinformatics*, 2000.

[18] C. Kamath. On Mining Scientific Datasets. In et al R. L. Grossman, editor, *Data Mining for Scientific and Engineering Applications*, pages 1–21. Kluwer Academic Publishers, 2001.

[19] G. Karypis, M. Deshpande, and M Kuramochi. Automated approaches for classifying structures. In *BIOKDD02: Workshop on Data Mining in Bioinfomatics (with SIGKDD02 Conf.)*, July 2002.

[20] J. Kim, E. Moriyama, C. Warr, P. Clyne, and J. Carlson. Identification of novel multi-transmembrane proteins from genomic databases using quasi-periodic structural properties. *Bioinformatics*, 2002.

[21] R. King, A. Karwath, A. Clare, and L. Dehaspe. Genome scale prediction of protein functional class from sequence using data mining. In *The Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

[22] S. Kirshner, I.V. Cadez, P. Smyth, and C. Kamath. Learning to classify galaxy shapes using the em algorithm. In *ICPR*, 2002.

[23] I. Koch, T. Lengauer, and E. Wanke. An algorithm for finding maximal common subtopologies in a set of protein structures. *J. of Comp. Biol.*, 3(2):289–306, 1996.

[24] M Kuramochi and G. Karypis. Frequent subgraph discovery. In *IEEE International Conference on Data Mining*, Nov 2001.

[25] M Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. In *IEEE International Conference on Data Mining*, Dec 2002.

[26] Y. Lamdan and H. Wolfson. Geometric hashing: a general and efficient model-based recognition scheme. In *ICCV*, 1988.

[27] H. Li and S. Parthasarathy. Automatically deriving multi-level protein structures through data mining. In *HiPC Conference Workshop on Bioinformatics and Computational Biology*, Hyderabad, India, 2001.

[28] R. Machiraju, S. Parthasarathy, D. S. Thompson, J. Wikins, B. Gatlin, T. S. Choy, D. Richie, M. Jiang, S. Mehta, M. Coatney, S. Barr, and K. Hazzard. Mining Complex Evolutionary Phenomena. In *NSF NGDM Workshop*, 2002.

[29] R. Machiraju, S. Parthasarathy, D. S. Thompson, J. Wikins, B. Gatlin, T. S. Choy, D. Richie, M. Jiang, S. Mehta, M. Coatney, S. Barr, and K. Hazzard. Mining Complex Evolutionary Phenomena. In H. Kargupta *et al.*, editor, *Data Mining for Scientific and Engineering Applications*. MIT Press, 2003.

[30] E.M. Mitchell, P.J. Artymiuk, D.W. Rice, and P. Willett. Use of techniques derived from graph theory to compare secondary structure motifs in proteins. *J. Mol. Biol.*, 212:151–166, 1990.

[31] W. Pan, J. Lin, and C. Le. Model-based cluster analysis of microarray gene-expression data. *Genome Biology*, 2002.

[32] S. Parthasarathy and M. Coatney. Efficient discovery of common substructures in macromolecules. In *IEEE International Conference on Data Mining*, 2002.

[33] L. De Raedt and S. Kramer. The level-wise version space algorithm and its application to molecular fragment finding. In *Seventeenth International Joint Conference on Artificial Intelligence*, 2001.

[34] M. Twa, S. Parthasarathy, T. Rosche, and M. Bullmer. Decision tree classification of spatial data patterns from videokeratography using zernike polynomials. *SIAM International Conference on Data Mining*, 2003.

[35] J. T. L. Wang, B. A. Shapiro, D. Shasha, K. Zhang, and C.-Y. Chang. Automated discovery of active motifs in multiple rna secondary structures. In *International Conference on Knowledge Discovery and Data Mining*, 1996.

[36] X. Wang, J. Wang, D. Shasha, B. Shapiro, S. Dikshitulu, I. Rigoutsos, and K. Zhang. Automated discovery of active motifs in three dimensional molecules. In *The Third ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1997.

[37] X. Wang, J.T.L. Wang, D. Shasha, B.A. Shapiro, I. Rigoutsos, and K. Zhang. Finding patterns in three-dimensional graphs: Algorithms and applications to scientific data mining. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):731–749, jul/aug 2002.

[38] H. Wolfson and I. Rigoutsos. Geometric hashing: An overview. In *In IEEE Computational Science and Engineering*, Oct 1997.

[39] X. Yan and J. Han. gspan: Graph based substructure pattern mining. In *IEEE International Conference on Data Mining*, Dec 2002.

[40] X. Zheng and T. Chan. Chemical genomics: A systematic approach in biological research and drug discovery. *Current Issues in Molecular Biology*, 2002.