An Alignment-free Approach to Cluster Proteins Using Frequency of Conserved k-mers

Armen Abnousi
School of EECS
Washington State University
Pullman, WA 99164-2752
aabnousi@eecs.wsu.edu

Shira L. Broschat
School of EECS
Washington State University
Pullman, WA 99164-2752
shira@eecs.wsu.edu

Ananth Kalyanaraman
School of EECS
Washington State University
Pullman, WA 99164-2752
ananth@eecs.wsu.edu

ABSTRACT

Identifying conserved regions in protein sequences is a fundamental operation that is recurrent in numerous sequence-driven analysis pipelines. It is used as a way to decode domain-rich regions within proteins, compute protein clusters, annotate sequences with function, and compute evolutionary relationships among protein sequences. Current approaches to clustering and annotating protein sequences based on conserved regions depend either on prior knowledge of domains or on computing pairwise sequence similarity, which is not feasible for very large collections of protein sequences. In this paper we present a new clustering method, *afClust*, that uses the abundance of exact matching short subsequences (k-mers) to quickly detect conserved regions. Our method also lends itself to parallelization under the MapReduce paradigm. Our experimental results are promising. For bacterial protein domains from the SMART database, we detected up to 85% of the targeted domain regions. Our parallel implementation processed 700,000 sequences in approximately one minute. We provide scalability experiments for a smaller data set.

Categories and Subject Descriptors

J.3 [**Life and Medical Sciences**]: Biology and genetics; I.5 [**Pattern Recognition**]: Clustering

General Terms

Algorithms; Proteins; Parallel Processing

Keywords

sequence homology; protein clustering; conserved region; domain; MapReduce algorithm; high performance computing

1. INTRODUCTION

Proteins play a vital role in every living organism. Determining the structure and function of proteins is of fundamental importance to the understanding of how cells work. On the other hand, advances in sequencing technology and completion of many genome projects have led to a rapid expansion of the known-proteins universe. For example, the UniProt knowledge base currently[1] contains more than 50 million automatically-annotated and well over 500,000 manually-annotated sequences (`http://www.uniprot.org/`). Furthermore, individual sequencing projects are leading to the generation of millions of newly identified putative protein sequences or Open Reading Frames (ORFs) [1, 2]. For instance, the Sorcerer II Global Ocean Sampling (GOS) project alone generated about $\sim 10 \times 10^6$ sequences [3].

One of the first steps that is carried out in the analysis of these sequences is the detection of homologous groups (i.e., subsets) of sequences. Computing these homologous groups, or "clusters," not only allows data reduction, but it also provides the first preview into the functional space encoded within these sequences. However, the growing volumes of data are making homology detection prohibitively expensive.

Related work: Current methods for protein clustering can be classified into two categories: i) methods that are based on pairwise similarity [3, 4] and ii) methods that compare each query sequence with profiles of sequence families [5, 6, 7]. The first category is based on pairwise sequence comparison using either optimal alignment methods such as the Smith-Waterman algorithm [8] or heuristic alignment techniques such as BLAST and its variants such as PSI-BLAST [9, 10]. The second category of methods uses sequence-profile or profile-profile matching to define groups for sequences [7]. Use of Hidden Markov Models is typical for this category (e.g., [11]) as is the case in databases such as Pfam [5] and SMART [6]. Machine learning techniques have also been used for detecting similar proteins (e.g., [12, 13, 14, 15]).

It is widely accepted that profile-based methods are more sensitive than similarity-based approaches [16]. However, all these current approaches rely on computing pairwise alignments at either the sequence level or the profile level. Computing alignments for modern day data set sizes can be exorbitantly costly. For example, the GOS paper reports spending a million CPU hours to perform an all-against-all BLAST analysis of their metagenomic ORF data sets (totaling 28.6×10^6 sequences) to detect protein families. In another paper, the analysis of a smaller subset containing 2.56×10^6 sequences using a parallel tool that uses a combination of sophisticated string matching filters to filter out the search space and the Smith-Waterman algorithm to com-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
BCB'15, September 09–12, 2015, Atlanta, GA, USA.
Copyright 2015 ACM 978-1-4503-3853-0/15/09 ...$15.00.
http://dx.doi.org/10.1145/2808719.2812223.

[1]as of August 2015

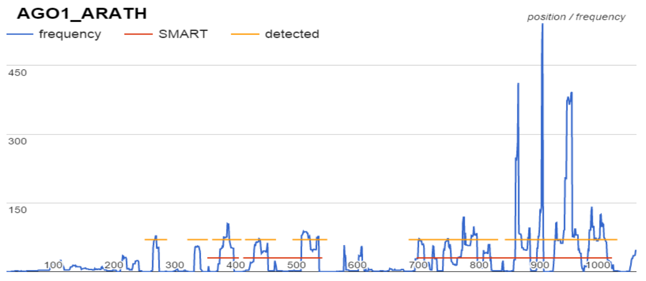

Figure 1: Example of k-mer profiles for detecting conserved regions
Regions containing highly conserved k-mers shown for the protein sequence AGO1_ARATH. The x-axis corresponds to sequence coordinates, the y-axis to the number of other sequences from a set of $700,000$ that also contain the k-mer starting at a given location. The chart also shows the location of three domains as annotated by the SMART database (red/darker line). The orange (brighter) lines represent conserved regions detected by our algorithm.

pute optimal alignments took over 2 hours of computation on 2K processors [4].

In this paper, our approach to the problem of detecting homology among protein sequences differs from the approaches above. Because the input is a set of protein sequences, it can be expected that the nature of alignments that can be shown to exist among sequences is *local*. This is because protein sequences contain *domains* that are regions highly conserved among multiple sequences as well as *motifs* that are shorter regions highly conserved among multiple sequences. Computing local alignments between pairs of sequences, however, is expensive, particularly as the magnitude of sequences collected from modern day sequencing technologies grows.

As a more scalable alternative, we propose an alignment-free approach called *afClust* to compute homology among protein sequences. More specifically, we focus on identifying regions of high sequence conservation within each sequence. These are regions in which a significant fraction of the k-mers are shared with other sequences in the input. Our approach quickly detects these regions first and then uses these regions as the basis for grouping protein sequences into homologous *clusters*.

Our approach has several advantages: i) Computing k-mers and their frequencies within each sequence relative to the entire set of sequences is a faster alternative to computing pairwise alignments, and ii) aggregating regions of k-mer conservation is an operation that is amenable to data parallelism. In fact, we present a parallel algorithm and its implementation using the MapReduce parallel programming paradigm [17].

Our approach can be compared to motif-based approaches (e.g., [18]) because rather than computing pairwise similarities to identify similar regions and then validating them for further support among multiple sequences, our approach directly looks for support among multiple sequences based on k-mer composition, obviating the need for pairwise or multiple alignments. In Figure 1, we provide an example that shows regions highlighted by conserved k-mers do exist and that in fact we can use them to recover domain-rich regions.

The contributions of this paper are as follows:

- A new approach to detect conserved regions within protein sequences based on k-mer frequency;
- A method to generate protein clusters based on these conserved regions;
- A parallel algorithm and implementation based on the MapReduce paradigm; and
- Experimental results for our proposed method showing its effectiveness in computing homologous protein clusters.

The remainder of the paper is organized as follows: In Section 2, we present our method in detail. In Section 3, the parallel implementation of our algorithm is discussed, and in Section 4, we present the results of our experiments on a subset of bacterial protein sequences and evaluate our parallel implementation. Section 5 concludes the paper and briefly describes future directions.

2. GENERATING PROTEIN CLUSTERS

Given a set S of n sequences, the *protein clustering problem* requires finding a collection of subsets of S such that each pair of proteins in each subset has high pairwise similarity. A protein can belong to different protein families (i.e., contain multiple domains) and, as such, overlapping can exist between the subsets defining clusters.

From the problem definition above, a simple approach is to compare each pair of sequences ($\binom{n}{2}$ pairs) and use the results to form protein clusters. To perform alignment, one can use local alignment [8] or related heuristics. However, this simple approach is not scalable due to its $\Omega(n^2)$ runtime complexity.

Our approach takes an alignment-free route—i.e., rather than computing pairwise similarities, we first determine "conserved regions" in sequences based on their k-mers and then use these conserved regions as anchors to determine cluster membership. The main idea behind this approach is as follows. Protein sequences contain domains representing conserved segments, and these domains are typically used as the basis for identifying protein clusters in a number of databases [5, 6]. The average domain length for a protein sequence is estimated to be 100 amino acid residues [19]. If a domain is highly conserved, we expect to see many exact matching subsequences in the domain regions of these sequences. Our algorithm exploits this property of domains to detect conserved regions.

2.1 Detection of Conserved Regions and Partial Grouping

Notation: A k-mer is a string of length k. Given a sequence s, we denote the i^{th} character of s by $s[i]$ and the substring of length l that starts at index i of s by $s(i,l)$. We denote the length of a sequence s by $|s|$. Let $S = \{s_1, s_2, \ldots, s_n\}$ denote an input set of n protein sequences.

Definition 1. *A k-mer is said to be* t-conserved *if it occurs in exactly t of the n sequences in S, where $t \geq 0$. For a t-conserved k-mer, we refer to t as its* frequency.

Definition 2. *A substring $s(i,\omega)$, starting at the i^{th} position of string s, is said to be a* conserved window *of length ω (for some constant $\omega \geq k$) if the average frequency of any k-mer originating in that window is at least τ for some constant $\tau > 1$.*

Note that by the definition above, not all k-mers in a conserved window need to be t-conserved such that $t \geq \tau$. This relaxation is necessary to tolerate sequence variations within conserved regions of protein sequences. We refer to the parameter τ as the *frequency threshold* and the parameter ω as the *window length*.

Definition 3. *Given a sequence $s \in S$, a substring $s(i, \ell)$ ($\ell \geq \omega$) is said to be a* conserved region *if:*

a) *(conservation) every substring of length ω contained within $s(i, \ell)$ represents a conserved window, and*
b) *(maximality) neither the substring $s(i-1, \omega)$ nor the substring $s(i+\ell-\omega+1, \omega)$ (if either exists) is a conserved window.*

The proposed method consists of two major steps. First, it detects *conserved regions* of a sequence and uses them to generate "partial clusters" relative to the sequence by forming a union of any other sequence that contributes to k-mers in any of the conserved regions. In the second step, the partial groups are consolidated through a process of merging into the final set of output clusters. As a result of this process, each output cluster will contain any two protein sequences s_r and s_t if either i) they have a common k-mer in one of their *conserved regions* or ii) s_r has a k-mer in common with the conserved region of s_i, s_t has a k-mer in common with the conserved region of s_j, and there exists a cluster that contains both s_i and s_j as intermediate sequences.

Algorithm 1 provides the routine for identifying conserved regions in the set of input sequences and the procedure for extracting partial clusters. The basic steps are as follows:

1. The k-mers present in the input set of sequences are hashed, and the hash entries are used to group all sequences containing each of the k-mers.
2. Next, a *k-mer profile* is generated for each sequence using the size of individual hashed entries of the hash table generated in the previous step. This profile is a chart that records the frequency of the k-mer starting at each sequence index.
3. Slide a window of length ω over each sequence's k-mer profile and check whether or not it is a conserved window (by Definition 2). If it is and if the previous window was also conserved, then extend the conserved region marked by the previous window to this window (by Definition 3). If, on the other hand, the previous window is not conserved, then we start a new conserved region at this newly detected conserved window. To check whether or not a given window is conserved, we compute the average of the frequencies of the k-mers in the window—note that each of these frequencies is available from the hash table in constant time (i.e., number of distinct sequences mapped to the corresponding hash entry in 1).
4. For each conserved region of a sequence, let $S' \subseteq S$ denote the subset of sequences that have at least one k-mer in common with that region. Subsequently, the partial cluster corresponding to that region is set to S'. The subset S' is constructed using the grouped entries within the hash table corresponding to each of the k-mers within a conserved region.

Complexity: Let $N = \Sigma_{i=1}^{n}|s_i|$. The algorithm makes several passes over the set of input sequences, with each pass costing $\Theta(N)$ time. The only other step in computation is the time taken to merge lists of sequences to construct and later output the final clusters. This time is an input dependent property. At the end of this stage, each input sequence has a list of clusters to which it belongs. Each list also contains an incomplete list of the id's of other members of these clusters together with the position number of the starting point of the conserved region including the k-mer of any sequence present in the cluster.

Algorithm 1 Generate Partial Clusters (S:$\{s_1, s_2, \ldots, s_n\}, \omega, \tau$)

```
Initialize hash table $H$ with $4^k$ empty entries, one for every possible $k$-mer
for each sequence $s \in S$ do
  for each $k$-mer in $s$ do
    Insert $s$ into the hash entry for $k$-mer
    Update size of the hash entry
  end for
end for
for each sequence $s \in S$ do
  Generate the $k$-mer profile for $s$ using the frequency of every $k$-mer in $s$
  for $i = 1$ to $|s| - \omega + 1$ do
    Let window $\leftarrow s(i, \omega)$
    Compute the average frequency of $k$-mers in the window
    if average frequency $>$ threshold $\tau$ then
      Mark the window as conserved
      Let $S_{curr}$ denote the subset of sequences that contain $k$-mers in the current window (obtained from the hash table)
      if Previous window (if it exists) was also marked as conserved then
        Extend previous conserved region to include this new window
        Let $S_{reg}$ denote the subset of sequences that contain $k$-mers in this region
        $S_{reg} \leftarrow S_{reg} \cup S_{curr}$
      else
        Begin a new conserved region to begin at this new window
        $S_{reg} \leftarrow S_{curr}$
      end if
      Let $S'$ denote the set of sequences that contain $k$-mers corresponding to this conserved window $s(i, k)$
    end if
  end for
end for
for each $s \in S$ do
  Output $S_{reg}$ for each conserved region in $s$
end for
```

Reporting Final Clusters

The probability of observing a k-mer present in τ random sequences is exponentially related to τ but is also related to k [20]. From this observation, we assume that the regions of two sequences to which a common k-mer belongs refer to the same protein domain and, thus, clusters representing these regions should be merged. According to [20], further studies to optimize the choice of k and τ are possible.

Each input sequence selects its smallest member for each cluster and labels the cluster with the smallest member's id

Algorithm 2 Report Final Clusters

```
for each partial cluster associated with every sequence do
    Label the cluster with its smallest member
    while cluster label changes do
        The sequence owning every partial cluster "sends" its
        label to all the members of that cluster
        Each cluster label is updated by comparing previous
        label with the received ones
    end while
end for
Send the cluster members list to the cluster label
Cluster label sequence outputs the final members list of
the cluster
```

and start position. After determining the label for each cluster, it informs all member sequences of a particular cluster (addressing the match position in that sequence) about the cluster label with which they are matched. (One can think of the sequence id as an IP address and the conserved region position as a port number. Each cluster label is sent to an IP and port number). On the other hand, upon receipt of such information, the receiving sequence finds the correponding cluster and updates its cluster label by choosing the smaller between its current label and the received label. If the label changes, it informs all member sequences about the change. This operation iterates until no more label changes occur. The routine for this operation is presented in Algorithm 2.

As a result of this operation, if sequence i has two different k-mers in one conserved region, one of them matching with sequence j and the other with sequence k, the intermediate sequence i will result in updating labels so that all three of them will have the same label, even if sequences j and k do not share any k-mers.

After convergence of the algorithm, each sequence will inform its cluster label about the members of that cluster from its perspective. The sequence whose id constitutes a cluster label will receive membership information from all other sequences in the cluster and will form one final cluster with all these sequences in it. These merged clusters are the final output of our algorithm.

3. PARALLELIZATION OF OUR METHOD

We implement our algorithm in the MapReduce framework [17]. A MapReduce program consists of two main functions of *map* and *reduce*. In the map phase a set of mapper tasks generate KeyValues based on the input. These KeyValue objects are hashed based on their key and redistributed based on the result of the hash. This intermediate hash-based grouping phase is called the *shuffle* phase. Each reducer task then processes the set of values that are hashed to the same key. A reducer, in turn, can also generate KeyValue objects. The MapReduce paradigm suits data-parallel applications naturally. For our purpose, we exploit the shuffle phase to implement many of our hash-based functionalities in parallel.

Our MapReduce algorithm works as follows (Figure 2). Initially each mapper reads the input set of sequences. It then generates KeyValue pairs using the k-mers of the sequences as keys and sequence id's and k-mer positions as values. The shuffle step subsequently sorts and redistributes these KeyValues such that each reducer receives one KeyMultiValue object, where all the values corresponding to the same key are sent to a single reducer. Based on the number of different sequence id's present in its MultiValue, each reducer determines the frequency of the k-mer mapped to it and emits a KeyValue, where the key is the sequence id and the value is the frequency of the k-mer and the position of the k-mer in that sequence. After this first complete stage of MapReduce, each sequence can construct its own k-mer profile.

In the second stage, each mapper determines the conserved regions for the sequence it is assigned, using the sliding window procedure discussed in Section 2. For all the detected regions, it repeats the operation of the first round by generating KeyValues, where the k-mer is the key and the value is the sequence id and the start position of the corresponding conserved region. After redistributing these pairs between reducers, each reducer generates a list of all the sequences that contain a corresponding k-mer (key) and the start position of the conserved region containing the k-mer in each of the sequences. Each reducer then emits a set of KeyValue pairs, where the key is one of the sequences in the list and the values are all other sequences in that list and their corresponding positions. Based on the received list of sequences and also the detected conserved regions, each reducer working on a sequence constructs a set of clusters, labeling each cluster with its smallest member (sequence id and conserved region start position). This concludes the partial clustering step. Note in Figure 2 that in many of the operations the output of the reducers is fed directly into shuffle, indicating that mappers do not perform any operations in these steps.

After forming partial clusters, each reducer handling a sequence starts emitting KeyValues to all other reducers that handle sequences present in one of its clusters, informing them that due to a common k-mer, the sequence is now a member of a cluster with the announced cluster label. Each such reducer by receiving this information, locates the corresponding partial cluster in its values, compares the received label with its existing label, and updates it to the smaller one. If an update is required, it repeats the same operation by emitting KeyValues, informing all reducers working on member sequences about the update. This operation continues until no more cluster labels are modified.

At this stage all sequences have been identified with their respective final cluster labels. Each reducer that handles a sequence generates a set of KeyValues where the key is the label (sequence id, conserved region start position), and the values are the id's of the member sequences. These KeyValues are then redistributed to different reducers, resulting in the formation of final clusters based on the smallest sequence id and the corresponding conserved region's start position in that sequence. These clusters are the final output.

4. RESULTS

4.1 Experimental Setup

First we conduct a parametric study to evaluate the effect of different parameter values. Then we evaluate the effectiveness of our method, and finally we investigate the scalability of our parallel implementation. For the first two experiments we use the SMART database to generate a data set of proteins and their corresponding domains. Domains in SMART are detected based on profile Hidden Markov

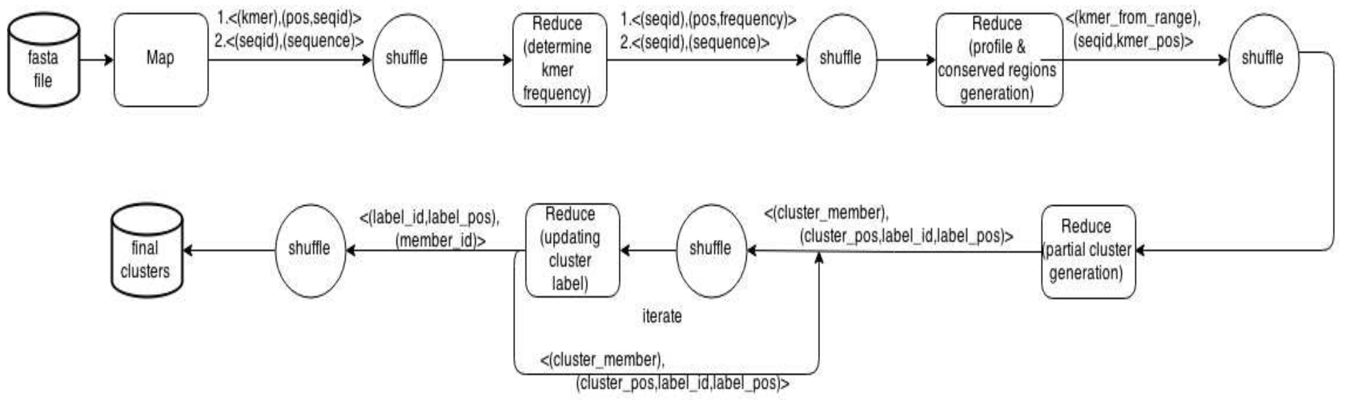
fasta file
Map
1.<(kmer),(pos,seqid)>
2.<(seqid),(sequence)>
shuffle
Reduce (determine kmer frequency)
1.<(seqid),(pos,frequency)>
2.<(seqid),(sequence)>
shuffle
Reduce (profile & conserved regions generation)
<(kmer_from_range), (seqid,kmer_pos)>
shuffle
Reduce (partial cluster generation)
<(cluster_member), (cluster_pos,label_id,label_pos)>
shuffle
Reduce (updating cluster label)
iterate
<(cluster_member), (cluster_pos,label_id,label_pos)>
<(label_id,label_pos), (member_id)>
shuffle
final clusters

Figure 2: An illustration of our MapReduce algorithm to generate protein clusters based on k-mer profiling. Two major steps are represented in this pipeline, the first is generating partial clusters and the second is using the partial clusters to generate the set of final clusters. The latter step involves an iteration, the convergence of which occurs when there is no longer an update in the sequence to cluster assignments.

Table 1: Common Bacterial Protein Domains in Our Data Set

Protein Domain Name	Number of Sequences
TOP1Bc	11545
CBM_2	726
ZnMc	2967
ZipA_C	1508
HLH	16
NADH-G_4Fe-4S_3	5476
POLAc	7110
PP2Ac	907
Resolvase	16007
S_TKc	1519
Endonuclease_NS	2435

Models constructed from multiple sequence alignments [6]. The data set consists of 50,214 bacterial protein sequences that contain at least one of the eleven common bacterial protein domains listed in Table 1. This data set is generated by taking the union of all proteins containing any of the eleven specified domains. However, each sequence can also contain other domains not listed in Table 1. Our second data set, used for our scalability evaluation, contains 700,000 bacterial protein sequences derived from the RefSeq protein database [21]. Scalability experiments are run on a subset of 100,000 sequences from this data set. For the scalability experiments we used the MapReduce implementation of our proposed algorithm. We called the MapReduce-MPI library [22][2] from our C++ code. The library provides the ability to set parameters so that all computation is performed in memory without any intermediate disk I/O. All experiments were performed on a Linux cluster with 10 compute nodes of 8 Intel processors (2.33GHz), 9 nodes of 64 AMD processors, and 128GB memory. Intel processors were used.

Setup for Comparative Evaluation: We perform a comparative analysis between the conserved regions identified by our method, *afClust*, and the domains characterized by the SMART database.

To quantify our results, we use the measures defined by:

$$OverlapQuality = \frac{TP}{TP + FP + FN}$$

$$Specificity = \frac{TP}{TP + FP}$$

$$Selectivity = \frac{TP}{TP + FN}$$

where parameters TP, FP, FN, and TN are defined as follows: We consider two different evaluation criteria, a region-wise evaluation and a position-wise evaluation.

For region-wise evaluation: A SMART domain region is labeled as a True Positive (TP) if it overlaps with one or more conserved regions detected by afClust. Note that regardless of the number of conserved regions that overlap with a SMART domain, we count each such domain region as a single TP. Similarly, a SMART non-domain region is labeled as a False Positive (FP) if there exist one (or more) conserved region detected by afClust that are completely contained in the SMART non-domain region. Again, regardless of the number of afClust conserved regions that overlap with a SMART non-domain, we count each such non-domain region as a single FP. A False Negative (FN) is a SMART domain region that has no overlap with any of the conserved regions output by afClust. All SMART non-domain regions that do not contain any of the afClust conserved regions are True Negatives.

For position-wise evaluation: Let n denote the total number of positions in all protein sequences. A position is considered a TP if it it is part of both a SMART domain and an afClust conserved region. A position is considered a FP if it is only part of an afClust conserved region, and a FN if it is only part of a SMART domain. All other positions are labeled TN.

4.2 Parametric Study

We consider the effect of three parameters k (k-mer size), τ (frequency threshold), and ω (window size). For our parametric study, we use position-wise evaluation. Intuitively,

[2]http://mapreduce.sandia.gov

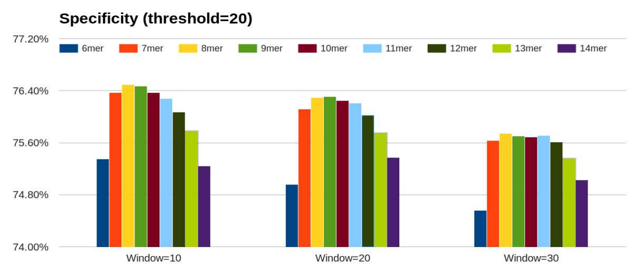

Figure 3: Effect of k-mer size on the specificity of detected regions

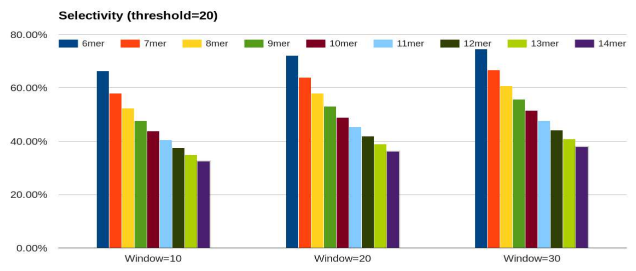

Figure 4: Effect of k-mer size on the selectivity of detected regions

reducing k implies reducing the length of the motifs we are trying to find. A smaller k increases the chance of finding exact matches, adding to their abundance. Choosing a reasonable value for k is crucial because a small value of k, (e.g., 1 or 2), will produce many random short exact matches of no significance. On the other hand, as we increase k, we expect to see greater conservation between matched sequences. As a result, if in one domain the conservation is too low, the algorithm will not be able to detect it. This effect can be observed in Figure 3 where the specificity is the ratio of the number of positions in conserved regions correctly detected and the total number of positions detected as conserved. As can be seen, using smaller values of k result in more incorrect positions detected as conserved and smaller specificity. At the same time, larger values of k result in fewer numbers of total regions being detected and smaller specificity. Increasing k results in a decrease both in the number of true positive (TP) position instances and false positive (FP) instances. Similarly, increasing k also results in a decrease in selectivity (Figure 4).

In Figures 3 and 4 we have used three window sizes ω, and we can see a trend of decreasing specificity and increasing selectivity for larger windows. Increasing ω results in requiring more k-mers with high frequency to exist in a region in order to identify it as conserved. As a result with fixed k and τ, increasing ω implies greater selectivity in identifying conserved regions (fewer FNs). On the other hand, if two detected conserved regions exist that lie close to each other, a larger ω will result in overlapping windows between them and therefore will generate one extended conserved region. If a domain really exists in the region with many conserved segments close to each other, a larger ω can be useful in joining them, but if it exceeds the optimal value, it will result in non-domain positions between two different domains that will be included in a conserved region, increasing FP and therefore slightly decreasing specificity.

The effect of the frequency threshold is opposite to the effect of the window size. Increasing the frequency threshold τ implies greater selectivity in identifying a region as conserved. This is because we require a higher average frequency of k-mers in that region. As a result, an increased threshold reduces FP, increasing the specificity, but it also reduces TP as well as FN because it requires a stricter condition than previously. This results in a slightly reduced selectivity. Recall that we allowed sequences in our data set to contain other domains in addition to our selected ones. As a result we do not expect our algorithm to find all existing domains but only the ones that are conclusively present in the data set. The data for the two plots above are collected based on all annotated domains for our data set. However, the behavior of the algorithm when we consider only the selected eleven domains is similar. On the other hand, other measures demonstrate a different behavior for these two cases. This difference is not unexpected as we discuss below.

Table 2: Detection of SMART domains by Conserved Regions based on overlapping regions

Measure	11 Domains	All Domains
SMART domains with no overlap	15%	28%
afClust conserved regions with no overlap	56%	7%
Overlap Quality	58%	68%
Specificity	64%	94%
Selectivity	85%	71%

Region-wise evaluation of conserved regions vs SMART domains. Note that using all domains rather than the selected 11 domains increases the performance; this implies the presence of other domains in our data set.

4.3 Evaluation based on k-mer frequency

In this section, unless otherwise stated we consider fixed values of $k = 7$, $\tau = 20$, and $\omega = 20$ as our method's parameters. We compare the conserved regions detected by our algorithm afClust against the SMART domains and then present a case study for some arbitrarily selected sequences.

4.3.1 *Comparison of afClust conserved regions and SMART domains*

First we evaluate the ability of our method to detect domain regions. For this purpose we perform region-wise and then position-wise evaluations. Table 2 summarizes the results for region-wise evaluation. Our method is able to detect 85% of the domains annotated in SMART. Accordingly, there is a comparably smaller set of domains in SMART (15%) that our algorithm was unable to detect. We hypothesize that one reason for this is the scarcity of sequences containing these domains which makes it difficult to find their corresponding regions with the threshold used. For example, it may be more difficult to detect a region corresponding to the HLH domain because it is only present in 16 sequences of our data set. In order to be able to detect the conserved regions in these few sequences, we would have to decrease our frequency threshold. This decrease in threshold will result in detection of such regions, but it will also yield to much less specificity in region selection as we saw earlier.

Table 3: Detection of SMART domains by Conserved Regions based on number of residues in each region

Measure	11 Domains	All Domains
Exclusively SMART domain positions	34%	36%
Exclusively afClust Conserved Regions	73%	23%
Overlap Quality	22%	53%
Specificity	26%	76%
Selectivity	65%	63%

Position-wise evaluation of detected conserved regions vs SMART domains. For each residue, a comparison is performed based on whether or not it belongs to a SMART domain and whether or not it is part of a detected conserved region.

We evaluated this hypothesis by reducing the threshold and as a result we improved our detection to 92% of SMART domains using $k = 7$, $\tau = 10$, $\omega = 20$, thereby confirming our hypothesis. Our inability to detect such domains also implies lower conservation where not enough k-mers are preserved and therefore the frequency of exact matching k-mers is too low to be detected using our default parameters. To evaluate this hypothesis we reduced our k value from 7 to 6, allowing our algorithm to capture shorter regions of exact matches and thereby lower conservation. The results satisfied our expectation, detecting 93% of the SMART domains ($k = 6$, $\tau = 20$, $\omega = 20$).

Some of the conserved regions identified by our method do not overlap with the selected eleven domains in SMART. It is possible that a group of these regions correspond to other annotated domains in SMART. This was checked by comparing the detected regions with all the domains annotated by SMART in our data set, and the comparison resulted in 93% coverage of our detected regions rather than the previous 44%. Note that this comparison cannot be precise because for such regions we are not using a conclusive set of representative sequences. There might be a domain that spans a region of only a few of the sequences in our data set, and therefore our algorithm is unable to detect it. However, if we increase the number of sequences representing that domain, our algorithm will successfully locate it. This observation also implies the ability of our algorithm to find domain regions that have not yet been annotated. In our experiments we have observed sequences that have regions with extremely high average frequency of matching k-mers. It is unreasonable to consider it a mere coincidence when, for example, a subsequence of length 12 is repeated in as many as 400 sequences. In such situations, the regions detected by our algorithm and not annotated in the database require reconsideration lest they are domains that the database generation method has missed.

In some cases our detected region exceeds the boundaries of the SMART region. In other cases the reverse is true; SMART considers longer regions as a domain than our inferred conserved region. Table 3 summarizes the results for position-wise evaluation.

4.3.2 Case Studies

To further investigate our approach, we compared our detected regions with the domains annotated by SMART for selected sequences from our data set of 50,214 sequences. Figure 5 shows the k-mer profiles for these sequences. The detected conserved and SMART domain regions are represented in these plots by horizontal lines.

In Figure 5(a) our algorithm successfully detects a large segment of the domain annotated by SMART (the red, or darker, line). Moreover it detects a separate conserved region in close proximity to the SMART domain. Given the abundance of exact matches preceding this SMART domain, one can hypothesize that either there exists a different domain in this area that SMART was unable to detect, or the Endonuclease_NS domain should be extended to include this area.

In Figure 5(b) again our algorithm detects a large part of the SMART domain S_TKc. However, our algorithm could not identify the regions corresponding to two other domain annotations for the Transmembrane region and PASTA domain. As explained previously, in general we do not expect our algorithm to detect these low-frequency regions, although it is possible as we will see later. For the detected conserved region, rather than one continuous region we have identified three shorter segments. The use of a larger window size would extend the segments resulting in an overlap between detected segments and conclusively yielding a long conserved region that would fit well with the SMART annotation.

While we can change the parameters to better match with SMART, considering three different conserved regions does not hinder the overall performance of our method. If we choose to use our original parameter settings, in the worst case the three separate segments will result in the generation of three different clusters with slightly different sequences in them. This clustering is still valid, representing three different clusters based on different conserved regions of one domain.

Figure 5(c) demonstrates an example when the frequency threshold τ is too small. The conserved region identified by our algorithm covers almost the entire sequence, combining the Transmembrane region with the Endonuclease_NS domain. This can be remedied by increasing the threshold from the value of 20 that was used. However, the high conservation shown in the k-mer profile leading to the SMART annotation for Endonuclease_NS may possibly imply that SMART has inaccurately identified a domain boundary.

In Figure 5(d) the k-mer profile for the protein A8PNE3 is pictured for three different values of k (k-mer sizes). The plot drawn using 10-mers is basically a horizontal line along the x-axis because there is no significant change in the frequency. In fact, the maximum frequency is two, i.e., the sequence shares one subsequence of length 10 with only one other sequence. Obviously a small number of shared subsequences can be considered random, and our algorithm refrains from selecting it as a domain. Thus, our algorithm was unable to locate any conserved regions in this sequence for the given setting. However, if this sequence contains a domain, we expect to see some conservation in its sequence. This leads us to the conclusion that for this domain the conservation is not high enough to be detected by 10-mers.

Subsequently we decrease k to seven, considering an exact matching subsequence of length seven. For 7-mers the plot shows a maximum peak (66 matches) at position 141. However, our threshold for average frequency in a window is too high to capture this peak as a conserved region. Next we consider using 5-mers. For this setting, our algorithm de-

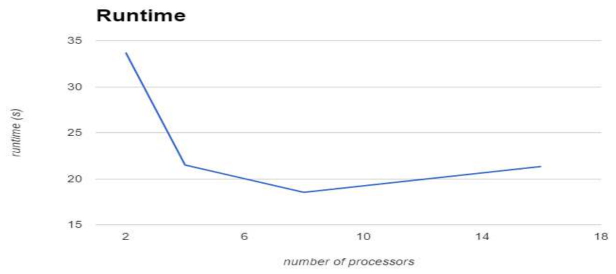

Figure 6: Speed-up chart for MapReduce implementation

tects two conserved regions that overlap with two domains annotated by SMART. This demonstrates the ability of our method to generate phylogeny of protein domains based on how conserved they are. However, $k = 5$ is small enough to generate a highly variable plot due to random exact matches. To prevent detection of unwanted regions (that do not represent a domain) we should increase the threshold for the average frequency or perhaps try a value of 6 for k.

In Figure 5(e) the k-mer profile for protein S6AI00 using 7-mers is pictured. Our algorithm has successfully detected a large fraction of the TOP1Bc domain as annotated by SMART. At the same time, it has identified regions corresponding to the TOP1Ac and TOPRIM domains that were not entirely present in our data set. This demonstrates the ability of our method to detect unannotated domains. Based on this plot, one can conclude that TOP1Bc, TOP1Ac, and TOPRIM are domain regions that appear together in many sequences.

4.4 Scalability evaluation

To evaluate the scalability of the MapReduce implementation of our algorithm, we ran the program on a subset of 100,000 sequences from our larger data set. We successfully ran the program on the complete data set of 700,000 sequences using nodes with more memory. However, because we needed to run the program using different numbers of processors, we were limited to nodes with 8GB of memory. Runtimes for various numbers of processors are presented in Figure 6 using the parameters $k = 7$, $\tau = 20$, and $\omega = 20$. As can be seen, for smaller numbers of processors there is a sublinear improvement in the runtime. Further investigation shows that the sublinearity is due to communication time. Map and Reduce operations execute in half the time using double the number of processors, but although the shuffle time decreases because of less communication per processor, this decrease is not linear because there are more processors. Increasing the number of processors beyond 16 results in a slight increase in runtime due to lack of memory. With 32 processors, the MRMPI library stops in the first shuffle operation due to insufficient memory (total memory being divided between more processors). The timing details are presented in Table 4. These observations lead us to the following conclusion: increasing the size of the data set will allow the computation time to dominate the communication time, hence providing the algorithm with scalability. However, running the program with larger data sets will require access to larger memory, especially for the initial operation of generating k-mer profiles.

Table 4: Runtimes for each Map and Reduce operation (seconds) using different numbers of processors

Operation	2 proc	4 proc	8 proc	16 proc
Map	10.42	5.15	2.73	2.61
Shuffle	16.32	12.69	13.53	16.46
Sort	2.19	1.13	0.67	0.70
Reduce	4.77	2.51	1.58	1.55
Total	33.71	21.49	18.52	21.33

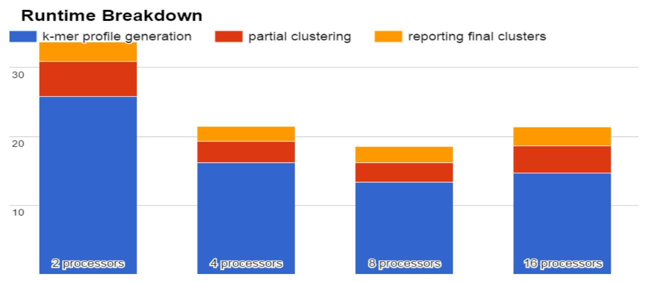

Figure 7: Runtime breakdowns based on the steps of the algorithm

The breakdown plots for the runtimes based on the semantic steps of the algorithm are presented in Figure 7. As can be seen, the main bottleneck of the algorithm is the k-mer profile generation step. However, the main contributor to the large runtime in this step is the communication, which can be ignored if a large enough data set is being used. As discussed before a large data set requires access to large memory.

5. CONCLUSIONS AND REMARKS

We presented an alignment-free approach called *afClust* for identifying conserved regions within protein sequences. We observed that a significant correlation exists between the frequency of occurrence of a k-mer in a data set and the region containing a k-mer being part of a protein domain. This property can be used as a method for detecting potential domain regions. We evaluated both the correctness and efficiency of our proposed method using two different data sets of annotated and unannotated protein sequences. Our method was able to detect a large percentage of SMART-annotated domains depending on the degree of conservation chosen using a parameter setting.

We also presented a MapReduce algorithm for the proposed method. Experimental results showed that the algorithm is able to work with a large number of sequences; in particular, we used the algorithm with 700,000 sequences. However, the larger the data sets that are used, the more memory will be required for each processor.

Among various tools to perform the task of identifying conserved regions we tested the sliding window method which is dependent on three parameters. We also presented a basic approach for finalizing output clusters from partial clusters. The correctness of this approach is dependent on the frequency threshold value (τ) and the length of the substring (k) that are used. An analytical study to identify the best values for these parameters is a future direction for this method.

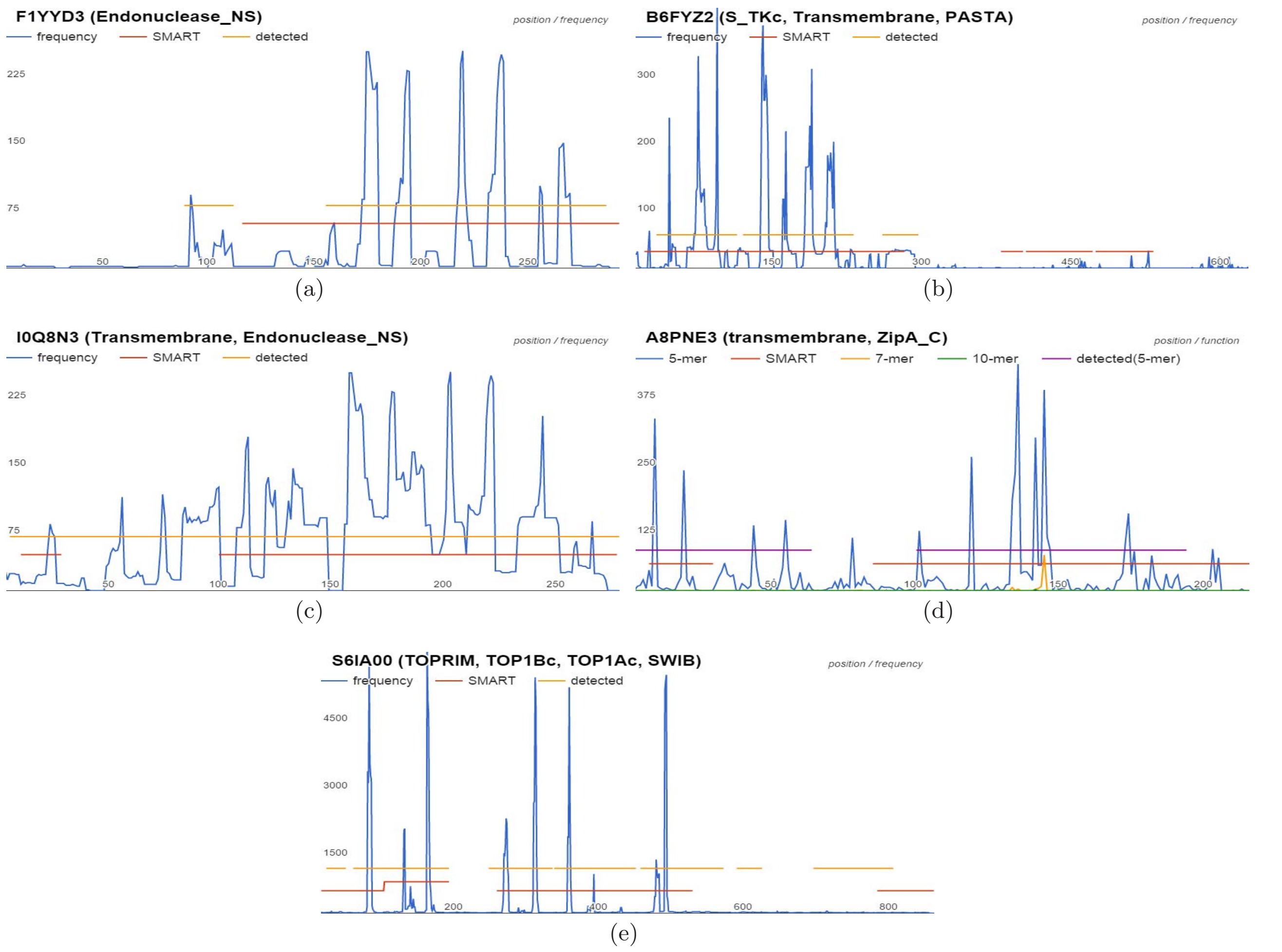

Figure 5: Frequency distribution plots for matching k-mers and corresponding domain regions for selected sequences. The label "detected" refers to those conserved regions detected by our method afClust.

6. ACKNOWLEDGMENTS

This work was funded by support from the U.S. National Science Foundation under the Advances in Biological Informatics program, Award DBI 1262664.

7. REFERENCES

[1] V. M. Markowitz, I.-M. A. Chen, K. Chu, E. Szeto, K. Palaniappan, M. Pillay, A. Ratner, J. Huang, I. Pagani, S. Tringe, *et al.*, "IMG/M 4 version of the integrated metagenome comparative analysis system," *Nucleic Acids Research*, vol. 42, no. D1, pp. D568–D573, 2014.

[2] T. Reddy, A. D. Thomas, D. Stamatis, J. Bertsch, M. Isbandi, J. Jansson, J. Mallajosyula, I. Pagani, E. A. Lobos, and N. C. Kyrpides, "The genomes online database (gold) v. 5: a metadata management system based on a four level (meta) genome project classification," *Nucleic Acids Research*, p. gku950, 2014.

[3] S. Yooseph, G. Sutton, D. B. Rusch, A. L. Halpern, S. J. Williamson, K. Remington, J. A. Eisen, K. B. Heidelberg, G. Manning, W. Li, *et al.*, "The Sorcerer II Global Ocean Sampling expedition: expanding the universe of protein families," *PLoS biology*, vol. 5, no. 3, p. e16, 2007.

[4] C. Wu, A. Kalyanaraman, and W. R. Cannon, "pGraph: Efficient parallel construction of large-scale protein sequence homology graphs," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 23, no. 10, pp. 1923–1933, 2012.

[5] E. L. Sonnhammer, S. R. Eddy, R. Durbin, *et al.*, "Pfam: a comprehensive database of protein domain families based on seed alignments," *Proteins-Structure Function and Genetics*, vol. 28, no. 3, pp. 405–420, 1997.

[6] J. Schultz, F. Milpetz, P. Bork, and C. P. Ponting, "SMART, a simple modular architecture research tool: identification of signaling domains," *Proceedings of the National Academy of Sciences*, vol. 95, no. 11, pp. 5857–5864, 1998.

[7] S. R. Eddy, "HMMER: Profile hidden Markov models for biological sequence analysis," 2001.

[8] T. F. Smith and M. S. Waterman, “Identification of common molecular subsequences,” *Journal of molecular biology*, vol. 147, no. 1, pp. 195–197, 1981.

[9] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, “Basic local alignment search tool,” *Journal of molecular biology*, vol. 215, no. 3, pp. 403–410, 1990.

[10] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” *Nucleic Acids Research*, vol. 25, no. 17, pp. 3389–3402, 1997.

[11] S. R. Eddy, “Profile hidden markov models.,” *Bioinformatics*, vol. 14, no. 9, pp. 755–763, 1998.

[12] T. Jaakkola, M. Diekhans, and D. Haussler, “A discriminative framework for detecting remote protein homologies,” *Journal of Computational Biology*, vol. 7, no. 1-2, pp. 95–114, 2000.

[13] L. Liao and W. S. Noble, “Combining pairwise sequence similarity and support vector machines for detecting remote protein evolutionary and structural relationships,” *Journal of Computational Biology*, vol. 10, no. 6, pp. 857–868, 2003.

[14] H. Saigo, J.-P. Vert, N. Ueda, and T. Akutsu, “Protein homology detection using string alignment kernels,” *Bioinformatics*, vol. 20, no. 11, pp. 1682–1689, 2004.

[15] S. Hochreiter, M. Heusel, and K. Obermayer, “Fast model-based protein homology detection without alignment,” *Bioinformatics*, vol. 23, no. 14, pp. 1728–1736, 2007.

[16] J. Söding, “Protein homology detection by hmm–hmm comparison,” *Bioinformatics*, vol. 21, no. 7, pp. 951–960, 2005.

[17] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[18] A. Ben-Hur and D. Brutlag, “Remote homology detection: a motif based approach,” *Bioinformatics*, vol. 19, no. suppl 1, pp. i26–i33, 2003.

[19] A. Heger and L. Holm, “Exhaustive enumeration of protein domain families,” *Journal of molecular biology*, vol. 328, no. 3, pp. 749–767, 2003.

[20] E. Blais and M. Blanchette, “Common substrings in random strings,” in *Combinatorial Pattern Matching*, pp. 129–140, Springer, 2006.

[21] T. Tatusova, S. Ciufo, B. Fedorov, K. O’Neill, and I. Tolstoy, “RefSeq microbial genomes database: new representation and annotation strategy,” *Nucleic Acids Research*, p. gkv278, 2015.

[22] S. J. Plimpton and K. D. Devine, “MapReduce in MPI for large-scale graph algorithms,” *Parallel Computing*, vol. 37, no. 9, pp. 610–632, 2011.