Fast Motif Discovery in Short Sequences

Honglei Liu[1] , Fangqiu Han[1], Hongjun Zhou[2], Xifeng Yan[1], Kenneth S. Kosik[2]

[1]Department of Computer Science, University of California, Santa Barbara, USA
{honglei,fhan,xyan}@cs.ucsb.edu
[2]Neuroscience Research Institute, University of California, Santa Barbara, USA
{robin.zhou,kenneth.kosik}@lifesci.ucsb.edu

***Abstract*—Motif discovery in sequence data is fundamental to many biological problems such as antibody biomarker identification. Recent advances in instrumental techniques make it possible to generate thousands of protein sequences at once, which raises a big data issue for the existing motif finding algorithms: They either work only in a small scale of several hundred sequences or have to trade accuracy for efficiency. In this work, we demonstrate that by intelligently clustering sequences, it is possible to significantly improve the scalability of all the existing motif finding algorithms without losing accuracy at all. An anchor based sequence clustering algorithm (ASC) is thus proposed to divide a sequence dataset into multiple smaller clusters so that sequences sharing the same motif will be located into the same cluster. Then an existing motif finding algorithm can be applied to each individual cluster to generate motifs. In the end, the results from multiple clusters are merged together as final output. Experimental results show that our approach is generic and orders of magnitude faster than traditional motif finding algorithms. It can discover motifs from protein sequences in the scale that no existing algorithm can handle. In particular, ASC reduces the running time of a very popular motif finding algorithm, MEME, *from weeks to a few minutes* with even better accuracy.**

I. Introduction

Motif discovery, finding sequence patterns from a set of protein sequences, is a very basic problem in many critical biological and medical applications. It has been extensively studied for more than a decade. For example, it is widely used to identify transcription factor binding sites (TFBS) [1] and antibody biomarkers [2]. Studying TFBS could help us learn the mechanisms that regulate gene expression, while antibody biomarkers are useful for diagnosis of diseases.

Figure 1 shows an example of protein sequences. In these sequences, there are subsequeces that are almost identical to each other with only a few mismatches. When aligning these subsequences together, we can extract a sequence pattern, which is also known as motif. A motif could be represented as a consensus string or a model describing probabilities of characters appearing at different positions. A data set usually contains more than one motifs which makes it hard to deal with even for a small number of sequences.

Advances in instrumental techniques like Random Peptide Libraries (RPLs) [2] that generate massive sequences with complex alphabets, e.g., protein sequences, post a Big Data challenge for motif finding algorithms. For example, it takes MEME [3], a very popular motif finding algorithm based on Expectation Maximization (EM), almost two weeks to finish running for a data set with 10k sequences. Several algorithms such as DREME [4], MEME-ChIP [5] and STEME

Fig. 1. A Motif in Multiple Protein Sequences

[6] adopted combinatorial approaches to improve the speed of MEME. Unfortunately, they can only work with DNA sequences (4 types of alphabets, A,G,C,T) as they either do exhaustive search or rely on index structure like suffix tree. A recently published algorithm MUSI [7], faster than MEME, can be applied to protein sequences (20 types of amino acids). Unfortunately, its accuracy is far below MEME according to our experiments.

To the best of our knowledge, there is no algorithm that could work with $> 10k$ sequences with a complex alphabet set and achieve comparable accuracy with MEME. Rather than developing another motif finding algorithm to outperform MEME, we propose a new strategy, that is pre-processing sequences with clustering, to divide the data into multiple small clusters and run existing motif finding algorithms on each cluster. This strategy has several advantages. First, it could be used in any existing motif finding algorithm. Second, by limiting the input to only a subset of sequences, those time-consuming algorithms may finish in a reasonable amount of time. Figure 2 shows a sketch of the clustering and motif discovery pipeline. Sequences are first grouped into clusters before a motif finding algorithm is applied. Motifs discovered from each cluster are merged together and delivered to users.

Now the key problem is how to cluster the sequences.

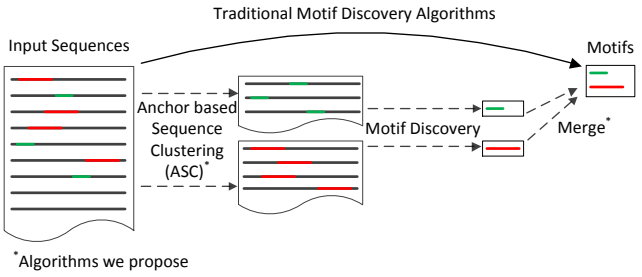


Fig. 2. Clustering Sequences for Motif Discovery

Could the problem be solved if we just arbitrarily divide the sequences into several clusters (Partitioning)? The answer is no since this method will miss most of low frequent motifs. For the same reason, randomly sampling a subset of sequences will not work either. Another straightforward approach is clustering sequences based on their overall similarities, for instance, K-means with edit distance as similarity measure. This will not work for the reason that motifs only appear in subregions of sequences. Table I compares the number of motifs found by aforementioned methods on a real data set. It is clear that direct sampling, partitioning, and K-means do not work well though they significantly reduce the runtime.

TABLE I. COMPARISONS BETWEEN DIFFERENT METHODS ON A REAL DATASET WITH 11,642 PROTEIN SEQUENCES

Methods	# of motifs found	Runtime (Min.)
MEME	20	two weeks
Sampling	11	79
Partitioning	5	9
K-means	14	32
ASC	24	6

Can we compare sequences based on their most similar subsequences, e.g., longest common subsequence (LCS)? This solution is intuitive but not scalable: Calculating LCS of two sequences is nontrivial and the number of comparisons could be quadratic in order to select a cluster center. In this paper, we propose an *anchor based sequence clustering* (ASC) algorithm that could efficiently identify sequences potentially containing the same motif. ASC could bypass the problem of directly calculating LCSs and identify cluster centers with only one scan. In this algorithm, each sequence is decomposed into a set of *anchors* which are similar to gapped q-grams in other literatures, but with variable shapes. Sequences are clustered based on anchors they contain to avoid pair-wise sequence comparisons. In particular, these anchors are not randomly selected. They are from the most significant ones in the dataset and then iteratively refined. Afterwards, a set of sequences are sampled from each cluster and provided to an existing algorithm to find motifs. Table I shows the superior performance of ASC. We are able to reduce the running time of MEME from weeks to less than 10 minutes and discover even more motifs than MEME.

The main contributions of this paper are summarized as follows.

- We introduce a new strategy for speeding up motif discovery by pre-processing sequences with clustering. This strategy is generic for any existing motif finding algorithm and a post-processing pipeline consisting of sampling, filtering and merging is also built to discover significant motifs.
- We propose an anchor based sequence clustering (ASC) algorithm that could efficiently group sequences containing the same motif together. As far as we know, ASC is the first anchor (gapped q-grams with variable shapes) based clustering algorithm.
- We provide theoretical analysis for our anchor-based similarity measure and illustrate that it can help check if two sequences contain the same motif with high accuracy.
- We perform extensive experiments with both synthetic and real data sets. The results show that ASC can discover motifs from protein sequences in the scale that none of the existing algorithms can handle.

II. PRELIMINARIES

Let $S = \{s_1, s_2, ..., s_N\}$ denote a set of N sequences over a fixed alphabet set $\Sigma = \{\beta_1, \beta_2, ..., \beta_{|\Sigma|}\}$ with $|\Sigma|$ symbols, e.g., $|\Sigma| = 20$ for protein sequences. Let $s[i, j]$ denote the subsequence between the i^{th} and j^{th} (both inclusive) character of sequence s. Usually, the input sequences have the same length, so we use l to represent the length of the sequences. Let w be the length of a motif m. We use a vector $\Theta = (\theta_1, \theta_2, ..., \theta_{|\Sigma|})$ to represent the background model which describes probabilities of seeing a character β_i appearing at any position in all the input sequences. θ_i is called the background probability of β_i, which can be calculated as the number of occurrences of β_i divided by the total number of characters $N \times l$. Table II summarizes some common notations that we are going to use in this paper.

TABLE II. NOTATIONS

Notations	Description
S	A set of input sequences
N	Total number of input sequences
Σ	Alphabet set
l	Length of input sequences
m	A motif
w	Width of the motif
θ_i	Background probability of β_i

Intuitively, a motif is a sequence pattern that repeatedly appears in S. The colored subsequences in Figure 1 follow a common sequence pattern as they differ from each other with only a few mismatches. That is, their Hamming distance, which is the number of different characters when aligned together, is small.

Definition 1: (LOCAL HAMMING SIMILARITY) Given two sequences s_1, s_2 and motif width w, the local hamming similarity between s_1 and s_2 under w is

$$lh(s_1, s_2, w) = \max_{\substack{0 \le i \le l_1 - w \\ 0 \le j \le l_2 - w}} h(s_1[i, i + w - 1], s_2[j, j + w - 1]),$$

where l_1 and l_2 are the length of s_1 and s_2 respectively, and $h(s_1[i, i + w - 1], s_2[j, j + w - 1])$ is the hamming similarity defined as the number of exactly matched characters between $s_1[i, i + w - 1]$ and $s_2[j, j + w - 1]$).

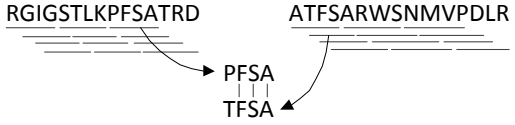


Fig. 3. Local Hamming Similarity

Figure 3 shows the local hamming similarity between two sequences is 3 when $w = 4$.

A motif is a sequence pattern extracted from a set of similar subsequences. One way of representing motifs is using *consensus string*.

Definition 2: (CONSENSUS STRING) Given n similar subsequences $S = \{s_1, s_2, ..., s_n\}$, the consensus string is a sequence m whose i^{th} character $m[i] = \arg\max_{\beta \in \Sigma} f(i, \beta)$, where $f(i, \beta)$ is the number of times that character β appears in position i for all $s \in S$.

For the example shown in Figure 1, the consensus string representation of the motif is *PFSE*. There are other representations such as position weight matrix (PWM) [3] (Section V). Our proposed algorithm can work with all the representations.

An occurrence of a motif is a subsequence that matches the motif (approximately). As shown in Figure 1, all the colored subsequences are occurrences of the motif, *PFSE*. Given a motif, we usually need to check which sequences contain this motif. Assuming the motif m is represented as a consensus string, we define the occurrence of motifs as follows.

Definition 3: (OCCURRENCE OF MOTIFS) Given a motif m of length w, a sequence s of length w is an occurrence of m if their hamming similarity is equal to or greater than t. We refer to the occurrence of a motif in a sequence as motif region.

Similarly, we could say a sequence s contains an occurrence of motif m with similarity t if their local hamming similarity $lh(s, m, w) \geq t$. How to choose the value of t is vague. A statistical approach [8] can bypass this problem by assigning a p-value to each sequence. A sequence is considered significant when it has a very small p-value, e.g., less than 0.05.

Motif Discovery: Given a set of sequences S, the task of motif discovery is to identify significant subsets of S that contain motifs and extract them. In this paper, we focus on the setting where each sequence is short and contains at most one motif. This is a widely used setting for motif discovery from peptide sequences [7]. Even under this simplified setting, none of the existing algorithms works well for a large number of sequences.

III. ANCHOR BASED SIMILARITY

As briefly discussed, clustering sequences based on motifs they contain has several advantages and could lead us to an efficient solution without developing yet another motif finding algorithm. However, the dilemma is given a sequence dataset, we do not know the motifs beforehand. Therefore, we need to develop a measure that is able to cluster the data without extracting motifs first.

Definition 4: (ANCHOR) A q-anchor consists of q characters $(\beta_{i_1}, \beta_{i_2}, ..., \beta_{i_q})$ drawn from Σ with replacement, and β_{i_j} appears before $\beta_{i_{j+1}}$ with gaps between them.

Definition 5: (ANCHOR BASED SIMILARITY) Given two sequences s_1, s_2 and their corresponding anchor sets A_1, A_2, the anchor based similarity between s_1 and s_2 is $|A_1 \cap A_2|$ which is the number of common anchors they share.

For example, the set of 2-anchors for $PFSE$ is {*PF, FS, SE, P_S, F_E, P_ _E*}. The concept of anchor is similar to gapped q-gram [9] except that it has variable shapes. Instead of using local hamming similarity, we propose to represent a sequence as a vector of anchors and use the number of common anchors to cluster sequences. In order to make anchor-based clustering work, we need to build a connection between anchor-based similarity and sequences that contain a motif.

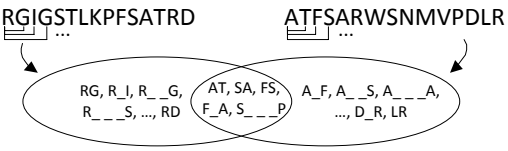


Fig. 4. Common 2-Anchors

Figure 4 shows two sequences and their 2-anchors. Among all the five common 2-anchors, {*FS, SA, F_A*} are from motif regions shown in Figure 3. It is possible for two random sequences to contain some common anchors. However this probability is relatively small compared to the cases where two sequences contain the same motif. We first analyze the probability of two sequences sharing at least d common anchors in these two cases as follows. Then we will show that when q-anchors are selected first, the chance for a random sequence of length l to contain d q-anchors is much smaller.

Here and throughout this section, we assume in a random sequence model, called *the background model*, all characters in the alphabet set Σ appear with equal probability. This assumption simplifies the presentation of theorems. We say a sequence pair contains an anchor if both sequences in the pair contain the anchor.

Theorem 1: Given a pair of sequences of length l drawn from the background model, the probability that this sequence pair contains at least d common q-anchors is at most $\frac{1}{d|\Sigma|^q} \sum_{i=q}^{l} \binom{i-2}{q-2}(l-i+1)^2$. When $q = 2$, this probability is upper bounded by $\frac{l^3}{3d\Sigma^{2l-2}}$.

We prove this theorem by following a counting argument: we first focus on the number of common anchors shared by a sequence pair and compute the sum, denoted as T_{ca}, of this number among all sequence pairs; then the number of sequence pairs that contain at least d common anchors is at most $\frac{T_{ca}}{d}$; thus the probability of one sequence pair containing at least d common anchors will be upper bounded by $\frac{T_{ca}}{d}$ divides by the total number of possible sequence pairs T_s. We compute T_{ca} and T_s in the following two lemmas.

Lemma 1: The total number of possible length l sequence pairs $T_s = |\Sigma|^{2l}$. Those sequence pairs appear with equal probability in the background model.

Proof: Each position in length l has $|\Sigma|$ possible characters. Then the total number of possible length l sequence pairs $T_s = |\Sigma|^{2l}$, as one sequence pair has $2l$ positions. Note that all characters in alphabet set Σ appear with equal probability in the background model. Therefore all sequence pairs appear with equal probability. ■

Lemma 2: Let S be the set of all possible sequences and let $N_q(s_1, s_2)$, s_1, $s_2 \in S$, be the number of common q-anchors shared by s_1 and s_2. Then $T_{ca} = \sum_{s_1, s_2 \in S} N_q(s_1, s_2) = |\Sigma|^q \cdot \sum_{i=q}^{l} \binom{i-2}{q-2}(l-i+1)^2$.

Proof: Firstly, for each length k q-anchor, here we use a length 3 anchor A_B as an example, we count the number of possible positions in one sequence that contains this anchor. If a sequence s_1 contains anchor A_B, this anchor could appear in $l-k+1 = l-2$ positions in this sequence. Then if two sequences in a sequence pair both contain anchor A_B, this anchor could appear in $(l-k+1)^2$ positions pairs in this sequence pair. Secondly, there are $|\Sigma|^q$ possible choices for the characters in a q-anchor. Since the length of sequences is l, the length of a q-anchor including gaps could range from q to l. And for each possible length i, there are $\binom{i-2}{q-2}$ ways to place the $q-2$ characters other than the start and end of the q-anchor. Therefore the total number of anchor matches between all possible sequence pairs will be $|\Sigma|^q \cdot \sum_{i=q}^{l} \binom{i-2}{q-2}(l-i+1)^2$. ■

Now we are ready to show the proof of Theorem 1.

Proof: Clearly, the number of sequences pairs which contain at least d q-anchors is upper bounded by the number of common q-anchors T_{ca} divides by d. Then the probability of a sequence pair containing at least d q-anchor is $T_{ca}/(dT_s) = \frac{1}{d|\Sigma|^{2l-q}} \sum_{i=q}^{l} \binom{i-2}{q-2}(l-i+1)^2$. Specially, when $q=2$, $T_{ca}/(dT_s) = \frac{1}{d|\Sigma|^{2l-q}} \sum_{i=q}^{l} \binom{i-2}{q-2}(l-i+1)^2 \leq \frac{l^3}{3d\Sigma^{2l-2}}$. ■

Corollary 1: Given a sequence s of length l drawn from the background model and d random q-anchors, the probability of s containing all d anchors is $(\frac{\binom{l}{q}}{|\Sigma|^q \sum_{i=q}^{l}\binom{i-2}{q-2}})^d$. When $q=2$, this probability is $(\frac{l}{2|\Sigma|^2})^d$.

Proof: It is clear to see that sequence s contains at most $\binom{l}{q}$ different anchors. By Lemma 4 (Section IV-A), the number of possible q-anchors is $|\Sigma|^q \sum_{i=q}^{l}\binom{i-2}{q-2}$. Therefore the probability of s containing one random q-anchor is $\frac{\binom{l}{q}}{|\Sigma|^q \sum_{i=q}^{l}\binom{i-2}{q-2}}$. As these d anchors are independently selected, the probability of s containing all d anchors is $(\frac{\binom{l}{q}}{|\Sigma|^q \sum_{i=q}^{l}\binom{i-2}{q-2}})^d$. ■

Corollary 1 shows that the chance for a random sequence to contain multiple q-anchors from a motif could be very low. If we are able to extract a few q-anchors from potential motifs, we can quickly remove those sequences that do not contain any motif and also group those sequences that contain the same motif together. This property will be used in the design of anchor-based clustering in Section IV. Theorem 1 and Corollary 1 together show that the chance for two sequences accidently sharing d common anchors is much higher than the chance for one sequence containing d pre-selected anchors.

Now we move to estimate the probability of discovering common anchors in motif regions of a pair of sequences. Let s_1 and s_2 be two sequences which contain motif m of width w with similarity t. Common anchors in s_1 and s_2 can be either anchors in motif m or other anchors happen to be contained in both sequences. However, anchors in motif m more likely appear as common anchors than other random anchors. We here use the probability of discovering common anchors only from the motif as a lower bound. Denote the motif region in s_1 and s_2 as $o_1 = o_{11}o_{12}\dots o_{1w}$ and $o_2 = o_{21}o_{22}\dots o_{2w}$, respectively. The following lemma shows that we could compute the number of common anchors in o_1 and o_2 by simply counting the number of identical characters found in the same positions of o_1 and o_2.

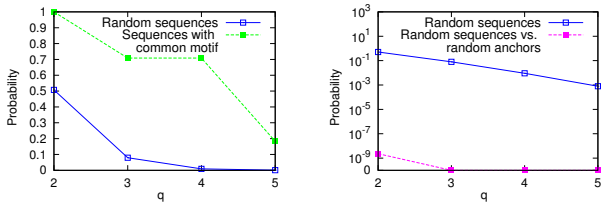
Lemma 3: Given two sequences s_1 and s_2 and their corresponding motif occurrence o_1 and o_2, let $I = \{i|\delta(o_{1i}, o_{1i}) = 1\}$, where $\delta(\cdot,\cdot)$ returns 1 if its two arguments are the same and returns 0 otherwise. Then I is the set of positions at which o_1 and o_2 contain the same character. Let $k = |I|$ is the cardinality of set I, then the number of the common anchors in the motif regions of s_1 and s_2 is at least $\binom{k}{q}$.

Proof: It is easy to see that characters at the ith position of o_1 and o_2 are the same if $i \in I$. Then any q-anchor composed by characters at q positions drawn from set I without replacement is a common anchor of of s_1 and s_2. Therefore the number of the common anchors in the motif regions of s_1 and s_2 is at least the number of draws, which is $\binom{k}{q}$. ■

The above theorem shows that if we want to compute the probability of discovering d common anchors in o_1 and o_2, we could (1) find the smallest integer k which satisfies $\binom{k}{q} \geq d$, and (2) compute the probability of having the same characters in k positions of o_1 and o_2. We apply this idea in the following theorem.

Theorem 2: Let k be the smallest integer which satisfies $\binom{k}{q} \geq d$. Given two sequences s_1 and s_2 which contain motif m of length w with similarity t, the probability that s_1 and s_2 share at least d common q-anchors is at least $\frac{\sum_{i=0}^{k-1}\binom{t}{i}\binom{w-t}{t-i}}{\binom{w}{t}}$.

Proof: We only need to compute the probability of having the same characters in k positions of o_1 and o_2. Motif occurrence o_1 share the same character with motif m at at least t positions. Without loss of generality, we assume that o_1 and m share the same characters at the first t positions. Motif occurrence o_2 also share the same character with motif m at at least t positions. Consider event $A_i = \{$When drawing t positions without replacement from a length w motif, i positions are chosen in the first t positions and $t-i$ positions are chosen from the rest $w-t$ positions$\}$. The probability of o_1 and o_2 share the same characters at no less than k positions is $\sum_{i=k}^{t} \Pr\{A_i\}$, as we assume that o_1 and m share the same characters at the first t positions. Note that the probability of event A_i is $\frac{\binom{t}{i}\binom{w-t}{t-i}}{\binom{w}{t}}$. Therefore, the probability of o_1 and o_2 sharing at least d common q-anchors is lower bounded by $\sum_{i=k}^{t} \Pr\{A_i\} = \frac{\sum_{i=k}^{t}\binom{t}{i}\binom{w-t}{t-i}}{\binom{w}{t}}$. ■



(a) Probabilities from Theorem 1 and Theorem 2 (b) Probabilities from Theorem 1 and Corollary 1

Fig. 5. The probabilities that two random sequences and two sequences containing the same motif share at least d common q-anchors, and the probability that a random sequence contains d random q-anchors when $l=15, w=10, t=7, d=5$.

In order to investigate how different values of q will affect our anchor based similarity measure, we vary q and calculate the probabilities according to Theorem 1, Corollary 1 and Theorem 2. Figure 5 shows the probabilities when we fix $l = 15, w = 10, t = 7, d = 5$ and range q from 2 to 5. As we can see, even though all the probabilities drop when q is increased, for two sequences containing the same motif, their probability of sharing d common q-anchors is always much higher than sequences that are merely generated according to the background model. And given d random q-anchors, the probability that a sequence contains them is extremely low. With this property, we could use the anchor based similarity between two sequences to indicate their probability of containing the same motif and further make clustering decisions.

IV. Anchor based clustering

Recall that there are mainly two challenges for clustering sequences based on motifs they contain: (1)Motifs are unknown beforehand, and (2) Pairwise sequence comparison shall be avoided. By adopting the concept of anchor based similarity measure, both problems can be avoided. In this section, we are going to introduce the design of anchor-based clustering.

According to our theoretical analysis in Section 3, the probability for two random sequences to have common anchors is much lower than sequences containing the same motif. Thus we derive an initial design of our algorithm as follows. A sequence is randomly picked as a cluster center, and then all the other sequences are compared with it using anchor based similarity measure. For any sequence, if its anchor based similarity with the center is equal to or greater than a threshold, it will be captured by this cluster. Repeat this for the rest of sequences until there is no sequence left. This method has two drawbacks. If we continuously choose sequences not containing any motif as cluster centers, the number of comparisons could be as large as $O(N^2)$ which is no better than pairwise comparisons. Furthermore, it might generate many small useless clusters for sequences that do not contain any motif; yet one still needs to run a motif finding algorithm on these clusters.

Running a traditional K-means algorithm on the anchor representation of sequences is also problematic. The number of comparisons could be quadratic with respect to the number of sequences in order to calculate the *mean* or *centroid* of a cluster. Hence we propose an *anchor based sequence clustering* (ASC) algorithm. ASC iteratively clusters sequences and selects a few anchors that are likely from potential motifs as cluster centers. Each sequence is first decomposed to a set of anchors and then clustered based on their corresponding anchors. Instead of using the *mean* or *centroid* of the sequences like traditional K-means, we carefully (re)select a set of anchors to represent the most distinctive features of a motif as the cluster center at each iteration. At first, K centers are initialized. In each iteration, sequences are assigned to their closest cluster and then the center of each cluster is adjusted. Not only is the center adjusted based on the new membership, but the anchor set used to represent the center is also adjusted. This process is repeated until all the clusters are stabilized.

Algorithm 1 outlines the anchor based sequence clustering algorithm. In our algorithm, a cluster center is a set of anchors, not the motif or the consensus string of sequences in the cluster.

Algorithm 1: Anchor based Sequence Clustering

```
input : A set of sequences S, the number of clusters
        K, the number of anchors d in a cluster center
output: K clusters of sequences
for s_i ∈ S do
    Decompose s_i into a set of anchors
end
Calculate the odd score (Section IV-A) of all the
anchors
Select top d × K anchors based on their odd scores and
randomly divide them into K anchor sets; each has d
anchors
repeat
    Assign each sequence to a cluster
    Adjust the center of each cluster
until termination condition (Section IV-D)
return sequences in each cluster
```

A. Choose Initial Anchors

Lemma 4: Given a set of sequences with length l, the maximum number of q-anchors ($q \geq 2$) that could possibly appear in it is $|\Sigma|^q \cdot \sum_{i=q}^{l} \binom{i-2}{q-2}$. It is $|\Sigma|^2 \cdot (l-1)$ when $q = 2$.

Proof: For each possible length i, there are $\binom{i-2}{q-2}$ ways to place the $q-2$ characters other than the start and end of the q-anchor. And there are $|\Sigma|^q$ possible choices for the characters for each of q-anchor placement. Therefore the maximum number of possible q-anchors in length l sequence is $|\Sigma|^q \cdot \sum_{i=q}^{l} \binom{i-2}{q-2}$. ■

For example, if $|\Sigma| = 20$ and $l = 15$, the number of possible 2-anchors is 5,600. The number of possible q-anchors increases exponentially with q as shown in Figure 6. Rather than using all q-size anchors, we propose an anchor filtering method by comparing the background probability and the observed probability of an anchor.

Definition 6: (Anchor's Background Probability) Given a q-anchor a, let t be its length including gaps. Its background probability

$$P_{background}(a) = 1 - (1 - \prod_{\beta_i \in a} \theta_i)^{l-t+1},$$

where l is the length of sequences.

Definition 7: (Anchor's Observed Probability) Given a q-anchor a, its observed probability

$$P_{observed}(a) = \frac{f(a)}{N},$$

where $f(a)$ is the number of sequences that contain the anchor a and N is the total number of sequences.

Those anchors that are over-represented are more useful for clustering sequences. So all the anchors that have $P_{observed} \lesssim P_{background}$ will be discarded.

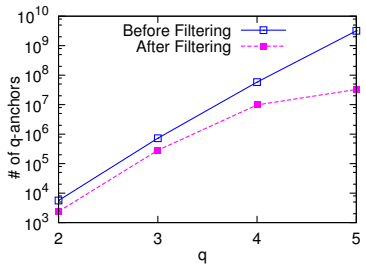


Fig. 6. The numbers of q-anchors before and after filtering for a data set of 11,642 protein sequences

Figure 6 illustrates the effectiveness of our filtering method on a data set of 11,642 protein sequences. More than half of anchors are filtered out by this method. It is getting more effective with the increase of q.

Intuitively, we want to use anchors that are uniquely derived from motif regions as the **cluster center**, such that the number of anchors a sequence shares with the center could indicate how likely this sequence contains the motif. That being said, we use d anchors as the center of a cluster. These d anchors represent the most distinguishable features of a motif. Thus the clustering algorithm will cluster the sequences based on which features they contain the most.

The first step of the clustering algorithm would be initializing centers for clusters. One way of doing this is to randomly assign some anchors as the center of a cluster. But this could make our algorithm take more iterations to converge or easily get stuck in local optimal. Here, we propose an initialization method based on the *odd score* of anchors.

Recall that $P_{background}$ and $P_{observed}$ are the background probability and the observed probability of seeing a sequence containing an anchor respectively. We use the odd score to indicate how much an anchor is different from the background.

Definition 8: (ODD SCORE) The odd score of anchor a is defined as the log ratio between the observed and background probabilities of a,

$$S(a) = \log P_{observed}(a) - \log P_{background}(a).$$

For an anchor, a higher odd score means it is more distinguishable from the background, thus having a higher probability of belonging to a motif. Therefore, before initializing centers, we first calculate the odd score for anchors and then rank them accordingly. Assume we have K cluster centers (later we will remove this requirement) and each cluster center has d anchors, we select the top $K \times d$ anchors and randomly draw d anchors without replacement as each cluster's center.

B. Adjusting Anchors

Sequence Assignment. Let $\mathcal{C} = (C_1, C_2, ..., C_K)$ refer to the K centers of clusters. In each iteration. the i^{th} sequence s_i will be assigned to its closest center $C \in \mathcal{C}$. Since each sequence is represented as a set of anchors, we use A_i to denote the set of anchors of s_i. The distance between s_i and a cluster center C_k is calculated as

$$dist(s_i, C_k) = |C_k| - |A_i \cap C_k|,$$

where $|C_k|$ (i.e, d) is the number of anchors in the k^{th} cluster's center and $|A_i \cap C_k|$ is the number of common anchors between the sequence and the center. Then for s_i, the closest center C can be found by

$$C = \arg\min_{C \in \mathcal{C}} dist(s_i, C).$$

We restrict that one sequence can only belong to one cluster as in our setting one sequence contains at most one motif.

Center Adjustment. In each iteration, centers of clusters are adjusted according to the sequences assigned to each cluster. In order to select the anchors that not only belong to a motif, but also represent the motif's most distinguishable features, we propose a ranking function based on the abundance score proposed as follows.

Definition 9: (ABUNDANCE SCORE) For the k^{th} cluster and an anchor a, let $f_k(a)$ be the number of sequences in this cluster containing the anchor and N_k be the total number of sequences in this cluster. The abundance score is defined as

$$S_k(a) = \log \frac{f_k(a)}{N_k} - \log \frac{f(a)}{N}.$$

For each cluster, we select d anchors with the highest abundance scores because they are more abundant in the cluster than in the whole set of sequences. The anchors of a motif are likely the most abundant ones when most of the sequences in the cluster contain the motif.

C. Extra cluster

In addition to the K clusters, we set up an extra cluster to collect sequences that do not share any anchor with the existing cluster centers. This is essential due to two cases: (1) Not all the sequences in the data set contain motifs, and (2) If the number of sequences that contain a motif is small, the anchors of that motif might not be selected. In this case, the corresponding sequences will not belong to any cluster. If we allow the sequences in these cases to be placed into other clusters, they could potentially affect the center adjustment process.

D. Termination Condition

The goal of our clustering algorithm is to minimize the distance between sequences and their cluster center so that sequences in a cluster will have a higher probability of containing the same motif. This objective can be captured by calculating the entropy of anchors in each cluster. The entropy of the k^{th} cluster is

$$H(C_k) = -\frac{1}{|C_k|} \sum_{a \in C_k} \frac{f_k(a)}{N_k} \log \frac{f_k(a)}{N_k}.$$

We want to minimize this entropy to ensure that most of sequences in the cluster contain anchors in the center. Therefore, the entropy can be used to measure the quality of the cluster. For each cluster, we re-calculate $H(C_k)$ using this objective function after the assignment of sequences. Assume $H^i(C_k)$ and $H^{i+1}(C_k)$ are the entropy scores after the i^{th} and $(i+1)^{th}$ iteration for the k^{th} cluster. Define

$$\delta_k = H^{i+1}(C_k) - H^i(C_k).$$

We stop updating the k^{th} cluster if δ_k is smaller than a pre-defined threshold.

For each iteration, the computational complexity comes from two parts: comparing each sequence with centers and selecting top anchors according to their abundance scores. Since we have N sequences, K clusters, and d anchors per cluster, the number of sequence-anchor comparisons would be $O(d \cdot K \cdot N)$. And the time complexity of checking whether an anchor is contained by a sequence is constant since we have built inverted index with bit vectors of sequence ids for anchors. The time complexity of enumerating of all q-anchors in sequences of length l is $O(l^q \cdot N)$. Assume the total number of anchors is $|A|$, the complexity of selecting top d anchors for K clusters is $O(K \cdot |A|)$ in the worst case. The complexity of each iteration is $O(d \cdot K \cdot N + l^q \cdot N + K \cdot |A|)$ which is linear with respect to the number of sequences N, the number of clusters K, and the number of anchors d. Note that this complexity is only valid for one iteration. ASC takes multiple iterations to finish the clustering task. Though the number usually is small, it might change with respect to d.

One issue we shall pay attention to is that $|A| = |\Sigma|^q \cdot \sum_{i=q}^{l} \binom{i-2}{q-2}$ according to Lemma 4. This complexity is exponential in terms of q. Fortunately, the results are already pretty good when we set $q = 2$ according to our experiments. If not specifically mentioned, we will be using 2-anchors in our implementation.

E. Eliminating K

The cluster number K corresponds to the number of motifs within the dataset, which usually is not known beforehand. In practice, our algorithm still works when K is not equal to the true number of motifs.

1) K is smaller than the true number of motifs. In this case, one cluster may contain two or more motifs. The existing motif discovery algorithms such as MEME [3] can handle sequence datasets with multiple motifs inside.
2) K is larger than the true number of motifs. In this case, two clusters may contain the same motif. It is fine to have duplicate motifs. Section V will discuss merging motifs.

Since our goal is to find motifs in sequences, not to achieve the best clustering result, we do not have a high requirement on the quality of clusters. The above two cases show that the setting of K is not critical in our method.

To further reduce the number of parameters, we propose a recursive clustering framework to eliminate parameter K. At first the whole pool of sequences are divided into two groups using our anchor based sequence clustering algorithm. Then each group is recursively clustered until all of its children can not be further divided into more clusters. Finally, we combine all the extra clusters into one set of sequences and rerun this whole process again to generate more clusters. This recursive process runs till the size of the combined extra clusters is small enough for the traditional motif finding algorithms to handle.

Figure 7 gives an example where the sequences are divided into 11 clusters. As the sequences are continuously being divided into two groups, some are marked as final clusters as the size of their children is smaller than a threshold τ, which is the maximum number of sequences that existing motif finding algorithms such as MEME can handle (we set $\tau = 600$ by default). Then after 8 clusters are generated, all the extra clusters are combined together and go through the process recursively to generate 3 more clusters.

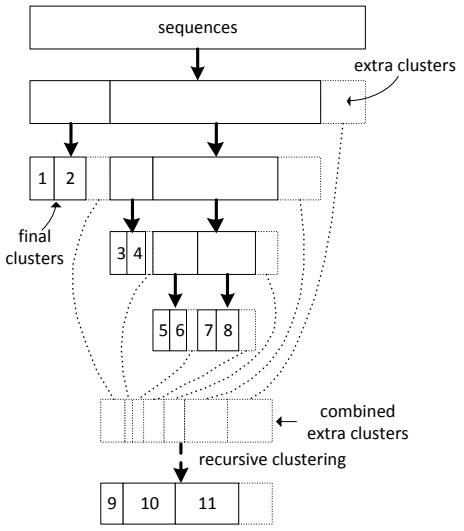


Fig. 7. An example of the recursive clustering framework in which a set of sequences is divided into 11 groups

This framework is different from traditional hierarchical clustering as we actually have two layers: inner loop and outer loop. Inner loop carries out recursive clustering while the outer loop collects the sequences that are considered as "random" in each iteration to ensure the capture of low frequent motifs.

V. Motif Merging and Filtering

Similar motifs might be discovered in different clusters. Hence, we need to merge them if they are close to each other. If motifs are represented as consensus string, we could use string distance measure such as hamming distance to cluster them. If motifs are represented as position weight matrix (PWM), we could use a similarity measure based on Kullback-Leibler (KL) divergence.

Definition 10: (Position weight matrix (PWM)) For a motif with width w, the PWM is a matrix M with size of $w \times |\Sigma|$. $M_{ij} = \frac{f(i,\beta_j)}{n}$ which is the probability of seeing the character β_j in position i of the motif.

Since discovered motifs might have different lengths, we need to consider all the possible ways of aligning two motifs. We first segment motifs with a sliding window and compare the motif segments using the slide window. The length of the sliding window is chosen as the minimum length of motifs w_{min}.

The distance of two PWMs is defined as the minimum distance between their segments of length w_{min} and the

distance of segments is defined as the average distance of their rows. Let M and M' denote the PWMs of two motifs, the distance between their i^{th} and j^{th} rows is defined as

$$D_{KL}(M_{i*}||M'_{j*}) = \sum_{t=1}^{|\Sigma|} M_{it} \log \frac{M_{it}}{M_{jt}},$$

where M_{i*} and M_{j*} are the i^{th} and j^{th} rows in the corresponding PWMs. The distance between M and M' is

$$D_{KL}(M||M') = \min_{i,j} \frac{1}{w_{min}} \sum_{t=0}^{w_{min}-1} D_{KL}(M_{(i+t)*}||M'_{(j+t)*}),$$

where $1 \leq i \leq w - w_{min} + 1$ and $1 \leq j \leq w' - w_{min} + 1$.

If $D_{KL}(M||M') \leq \theta$ (we set $\theta = 1.5$), the two motifs are considered to be similar. The sequences in their corresponding clusters will be merged together and we go through the motif discovery process again to get a unified motif.

VI. Experiments

In this section, we perform experiments to evaluate the anchor based sequence clustering (ASC) algorithm on both synthetic and real data sets. We have built the whole pipeline of motif discovery upon MEME [10], MUSI [7], GibbsCluster [11] and ACME [12]. With ASC being applied, we refer to them as ASC+MEME, ASC+MUSI, ASC+GibbsCluster and ASC+ACME. It is worth mentioning that ASC can be adopted by any existing motif finding algorithm. (1) We first show how much ASC can actually improve existing motif finding algorithms in terms of runtime without a quality trade-off. (2) We then examine the characteristics of ASC and verify the design.

MEME is the most popular motif discovery algorithm with high discovery rate. In comparison with MEME, MUSI and GibbsCluster are the two state-of-the-art probabilistic methods that were developed to improve the runtime with an accuracy trade-off. ACME is a recently proposed parallel motif extracting algorithm based on suffix tree. It's designed to extract motifs from an extremely long sequence and reported scaling to large alphabet set. When ASC is applied to these algorithms as a pre-processing step, it can be five orders of magnitude faster than MEME and 50+ times faster than MUSI/GibbsCluster/ACME, without losing any accuracy. All the experiments are conducted on a server with 2.67GHz Intel Xeon CPU (32 cores) and 1TB RAM. The real datasets and the source code are available at http://www.cs.ucsb.edu/~honglei/abp/download.htm.

A. Data Sets

We compare ASC with other algorithms using both real and synthetic data sets. Only protein sequences are used in our experiments as there already exist several methods [4], [5], [6] that work very well for large scale DNA sequences. In the real dataset, we directly compare the quality of discovered motifs, while in the synthetic data, we embed some motifs and treat them as ground truth. Most of our data sets contain fairly short sequences, which is a common setting in deep sequencing phage-selected peptide datasets [13], [14] for which none of the existing tools scales well.

Real data. For the real data, we use 5 datasets of protein sequences as shown in Table III. *Celiac* is from [2], consisting of 11,642 peptide sequences recognized by serum antibodies from patients with Celiac Disease. Each sequence in this data set has a length of 15. The other 4 datasets, *FXIIa*, *uPA*, *SrtA* and *PK* are from [13], containing shorter sequences ranging from 8 to 10.

TABLE III. Real datasets

Name	# of sequences	Length of sequences
Celiac	11,642	15
FXIIa	13,945	10
uPA	5,525	9
SrtA	4,993	8
PK	2,149	8

Synthetic data. We generate synthetic data sets using the same procedure as shown in [15]. A set of N sequences with length l are generated by randomly choosing characters from the protein alphabet set Σ (total 20 amino acids). Then K sequences with length w are generated as parent motifs in the same manner. Finally, each parent motif is planted to p percent of all the sequences. And e characters of the parent motif are randomly mutated to other characters in Σ each time before being planted into a sequence. The values of w, p and e are integers randomly drawn from $[6, 12]$, $[1, 20]$ and $[0, \lceil 0.3w \rceil]$ respectively to mimic the real data set, where $\lceil 0.3w \rceil$ represents the smallest integer not less than $0.3w$. We generate different data sets by varying l, K and N. In order to test how the noise in the data set would affect our results, we also generate data sets by gradually increasing e from $\lceil 0.3w \rceil$ to $\lceil 0.9w \rceil$.

B. Motif Discovery

Real Data. Since MEME usually finds more and better motifs than MUSI and GibbsCluster, both of which trade accuracy for efficiency. We first compare the quality of motifs discovered by ASC+MEME with MEME.

For the 11,642 protein sequences in Celiac, MEME is applied to find 20 motifs from it. Then, we redo the process of motif discovery again with ASC+MEME (MEME is set to find 20 motifs in each cluster) and compare their results. We say that a motif x discovered by MEME is successfully recalled if there exists a motif y in ASC+MEME's results and $D_{KL}(M^x||M^y) \leq 1.5$ [16]. Table IV shows the results of ASC+MEME with respect to different numbers of clusters. It is possible for the number of discovered motifs to be smaller than the number of recalled motifs since MEME may return duplicate motifs while we have merged those motifs. With the increasing cluster number, ASC+MEME not only recalls all the 20 motifs discovered by MEME, but also discovers new motifs. When we remove the k constraint and use the framework proposed in Section IV-E to automatically generate clusters, 20 motifs can be fully recovered.

TABLE IV. Motifs returned by ASC+MEME for *Celiac*

# of clusters	# of motifs recalled	# of motifs found
10	17	16
20	18	19
40	20	22
60	20	24
w/o k	20	24

For the other 4 real datasets, we use the motifs reported in [2] as ground truth. For MEME, we set it to find 10 motifs which is already larger than the number of reported motifs. For ASC+MEME, we try $k = 10$ and the case with k removed (MEME is still set to find 10 motifs). Both MEME and ASC+MEME can recall all the motifs that are reported by [2]. Table V shows both of them run better and ASC+MEME finds more.

TABLE V. COMPARISONS WITH THE REPORTED MOTIFS

	FXIIa	uPA	SrtA	PK
# of reported motifs	2	2	1	1
MEME	2	4	1	2
ASC+MEME ($k = 10$)	7	4	2	2
ASC+MEME (w/o k)	7	4	2	2

Though MEME and ASC+MEME both can recall all the originally reported motifs , they differ a lot in efficiency. The only two datasets MEME can finish running within 24 hours are *SrtA* and *PK* while ASC+MEME only takes minutes to run for each of the 4 datasets as shown in Figure 8.

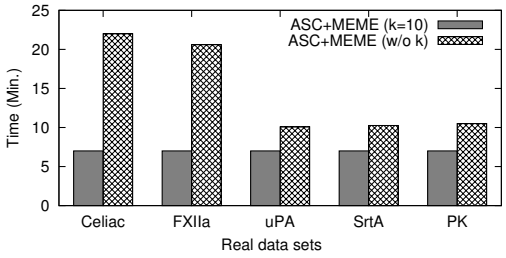


Fig. 8. Runtime of ASC+MEME for the real data sets

Synthetic Data. In the synthetic data experiment, we count the number of planted motifs discovered by each method. For MEME and GibbsCluster, we set the number of clusters identical to the number of planted motifs. For MUSI, since we always get a segment fault whenever we set the number of motifs larger than 10, we set it to 10 through the experiments. For ACME, because it requires several additional parameters (minimum number of occurrences, maximum number of allowed mismatches and number of threads) than those probabilistic methods, we report its results separately at the end of this section.

Since the planted motifs are a kind of consensus strings, we extract the consensus strings from each probabilistic method by using the most frequent characters at each position of a motif. We say that a planted consensus string x is recalled if there exists a consensus string y in the result such that $lh(x, y, min(|x|, |y|)) \leq 2$, where $min(|x|, |y|)$ is the smaller length of x and y. We conduct experiments by fixing $d = 5$ and varying l and K, where d is the number of anchors in each cluster center, l is the length of input sequences and K is the number of planted motifs. Figure 9 shows the number of recalled motifs for these methods when we fix $l = 15$. We also randomly sample 10% of all the sequences in the data set and run MEME, which is referred as Sampling+MEME. Moreover, we refer to the naive method which randomly divides sequences into several clusters as random sequence clustering (RSC). RSC+MEME fails to discover most of the planted motifs. In the contrast, ASC+MEME (ASC+MUSI and ASC+GibbsCluster) outperforms the original algorithm MEME (MUSI and GibbsCluster). We then conduct experiments to test our method's robustness with respect to the sequence length. Figure 10 shows the number of recalled motifs when we fix $K = 20$ and vary l. Due to space constraint, we omitted the results of RSC+MEME since they follow a similar trend. The improvements of ASC over existing motif finding algorithms are consistent for different l.

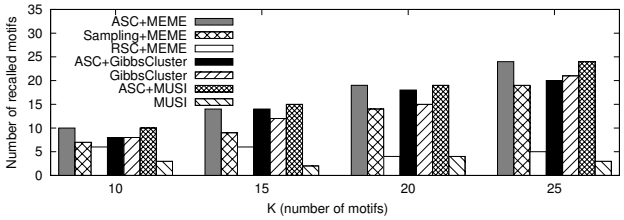


Fig. 9. Recalled Motifs: $d = 5, N = 10k, l = 15$, varying K

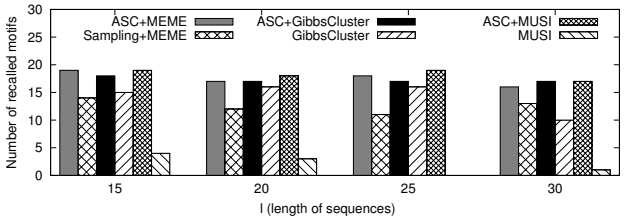


Fig. 10. Recalled Motifs: $d = 5, N = 10k, K = 20$, varying l

Next, we check the robustness of these algorithms by gradually increasing the noise (the number of mutated characters e) when a motif is planted to sequences. We fix $l = 15$, $d = 5$, $K = 10$, $N = 1k$ and compare our results with MEME and GibbsCluster. We do not include MUSI here because it fails to run with unknown error. The reason we choose such a small data set is because it takes too much time for MEME to run on a larger data set. As shown in Figure 11, ASC+MEME's accuracy is comparable to MEME even when there are a large amount of noises. ASC+MEME even recalled more motifs than MEME when $e = 0.6w$ and $0.8w$ where w is the width of the planted motif.

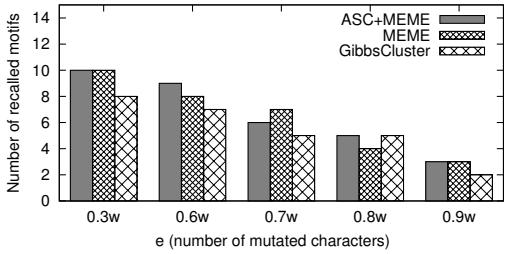
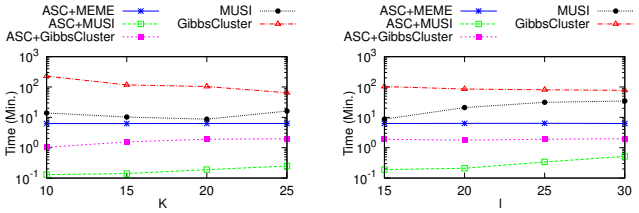


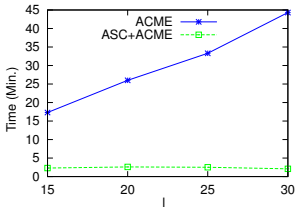
Fig. 11. Number of recalled motifs when $l = 15, d = 5, K = 10, N = 1k$ with different e

We then examine the runtime of these methods by fixing $d = 5$, $N = 10k$, and varying l and K. In Figure 12, we omitted the results of MEME because it takes more than two weeks to run for one data point. Figure 12(a) shows the overall runtime when we fix $l = 15$ and vary K. Note that the runtime of ASC+MEME (ASC+MUSI and ASC+GibbsCluster) is the cumulative running time of ASC and MEME (MUSI

and GibbsCluster). ASC+MEME is much faster than MEME, MUSI and GibbsCluster, and its runtime does not change much over K. ASC+MUSI is still two orders of magnitude faster than MUSI and GibbsCluster. Figure 12(b) shows the runtime when we fix $K = 20$ and vary l. The increase of l does not affect ASC's consistent improvement over runtime.



(a) When $l = 15$ (b) When $K = 20$



(c) When $K = 10$

Fig. 12. Runtime: $d = 5$,$N = 10k$

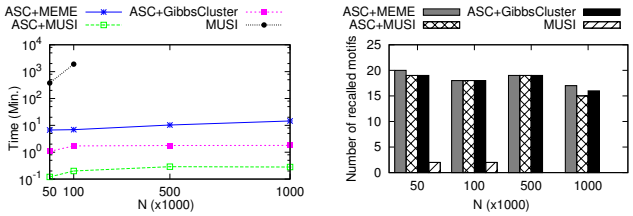
To compare ASC+ACME with ACME, we concatenate short sequences as a long sequence. We also use 3 and 100 for the maximum number of allowed mismatches and the minimum number of occurrences when $N = 10k$. Note that these parameters are set according to how we generated the synthetic data and are usually not accessible to users when working with real data. In the following experiments, we use 30 threads for ACME. When we set $d = 5$,$K = 10$ and vary l, both ASC+ACME and ACME can recall all the planted motifs. Moreover, as shown in Figure 12(c), ASC+ACME has consistent improvements over the runtime. Results with different parameters are omitted as they follow a similar trend.

Scalability. The scalability of these methods is tested by varying the number of sequences. Figure 13 shows the runtime and the number of recalled motifs when we vary N from 50 thousand to 1 million. Some results are omitted because we couldn't get them within a reasonable amount of time for the corresponding methods. In particular, MEME didn't finish in one month. As we can see, ASC+MEME (ASC+MUSI and ASC+GibbsCluster) takes much less time than MUSI and GibbsCluster. Moreover, ASC+MEME (ASC+MUSI and ASC+GibbsCluster) can work in some cases that MUSI and GibbsCluster can't.

C. Properties of ASC

In this section, we analyze the design of ASC and check its performance using synthetic data.

Effectiveness. The goal of ASC is to group the sequences that contain the same motif into one cluster. We call a pair of sequences a misplaced pair if the two sequences contain the same motif but are placed into different clusters by ASC. We



(a) Runtime (b) Recall

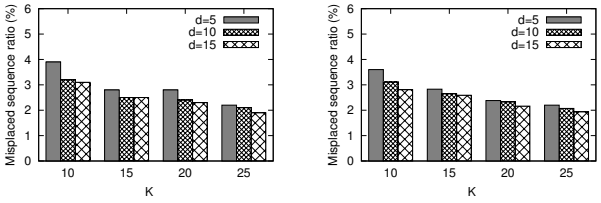
Fig. 13. Scalability w.r.t. Number of Sequences

use misplaced sequence ratio (MSR) to measure the ratio of misplaced pairs. MSR is defined as

$$MSR = \frac{\#\ of\ misplaced\ pairs}{(\#\ of sequences\ containing\ motifs)^2}.$$

If a sequence contains multiple motifs, we will count it multiple times. A lower MSR indicates better clustering accuracy.

We first conduct experiments on two data sets of 1 million sequences with the input sequence length $l = 15$ and $l = 20$ respectively. The number of embedded motifs K and the number of anchors d in each cluster center are varied. Figure 14 demonstrates that the MSRs of ASC follow similar trends for different l. The misplaced sequence ratio is quite small.



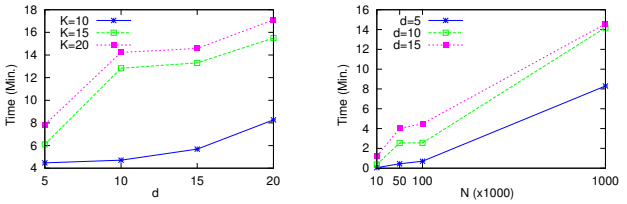
(a) When l=15 (b) When l=20

Fig. 14. Misplaced Sequence Ratio of ASC

Efficiency. We then conduct experiments to test the clustering efficiency of ASC. Figure 15 shows the running time of ASC when l is fixed to 15. Due to space constraint, we omit the results for $l = 20$ since they follow a similar trend. We first fix N to 1 million and vary the values of d and K to see how ASC behaves, where d is the number of anchors in each cluster center and K is the number of planted motifs. Figure 15(a) shows that the runtime increases when either d or K increases. This is because larger K or d would make the clustering process take more time to terminate. This is not a problem since a small d can already give us a very good MSR and the running time of ASC is negligible compared to the time taken by the following motif discovery process. We further conduct experiments to test how ASC responds with an increasing number of sequences by fixing $K = 20$ and vary d and N. Figure 15(b) shows that with different d, the runtime of ASC increases almost linearly with respect to N which is consistent with our time complexity analysis in Section IV-D.

VII. Related Work

Motif finding is a classic problem and has been extensively studied for more than a decade. Federico *et al.* gives a very



(a) When $N = 1M$ (b) When $K = 20$

Fig. 15. Running Time of ASC when $l = 15$

good survey of motif finding algorithms before and after next-generation sequencing era [1]. But still, with new challenges posted by the increasing size and complexity of big data, it is not completely solved. Most of the existing algorithms can be divided into two categories, combinatorial and probabilistic.

Combinatorial methods. Usually these methods use a combinatorial definition of motif, and treat the motif finding problem as an approximate pattern matching problem. For instance, in order to find motifs with length W, all possible 4^W sequences are enumerated (for DNA sequence with $\{A,G,C,T\}$ as the alphabet set). These sequences are modified with ambiguity codes, e.g., setting *K=A or C*, to allow mismatches. An exhaustive matching with the input sequences is conducted to count the frequency of sequences. The most frequent sequences will be outputted as motifs. Methods like SMILE [17] and Weeder [18] adopted this idea and also created index structure to speed up the matching process. However, since they need to do exhaustive searching, they are still very slow and can not scale w.r.t a larger alphabet set. Recently, a method named ACME [12] is reported to be able to scale to large alphabet set by utilizing parallel computing. We compared our algorithm with ACME in the experiments section.

The combinatorial methods usually can get optimal results. However, they need large search space and often require a few parameters like the length of motifs and the number of allowed mismatches, which users may not have knowledge of.

Probabilistic methods. These methods treat the data as composed of two components, the motif model and background model. They can infer the parameters of motif and background models that fit the data, thus classifying the subsequences either to motif or background. Currently, there are two approaches used to perform inference: Expectation Maximization (EM) and Gibbs Sampling.

EM. Multiple EM for Motif Elicitation (MEME) [3], [19], [10] is currently the most popular motif finding software. It was first proposed in 1994 by T. Bailey *etc*. The idea is to first break each input sequence into overlapping k-mers. Then use EM to estimate the model that fits the data best. Even though they kept improving the algorithm, there are still some problems with this method. It can not handle large data sets, e.g., larger than 600 protein sequences of length 15 in our real dataset.

STEME [6] can speed up the EM process by using suffix tree index for the sequences. But due to the complexity of suffix tree, this method can only support DNA sequences.

Recently, T. Kim *et al* proposed MUSI [7] which is fast and can work with protein sequences. MUSI first uses MAFFT [20] to do multiple sequences alignment and then use EM to infer PWMs from the alignment results. We compared our algorithm with MUSI in the experiment section.

Gibbs sampling. Gibbs Sampling has similar mechanism with EM, but adopts a stochastic way to modify the current solution. Algorithms such as motif sampler [21] and AlignACE [22] belong to this category. The advantage of these methods is that they have a better chance to escape from local optima.

GibbsCluster [11] adopts the idea of Gibbs sampling to do multiple sequences alignment and clustering. At first, sequences are randomly divided into several clusters and aligned together. Then, in each iteration, a sequence is selected to do "shifting" and moved to another cluster with some probability. This process is repeated until the alignment score reaches a local optima. Motifs are extracted from aligned sequences in each cluster.

Gapped q-gram. Another related topic is gapped q-gram. A gapped q-gram is a subset of q characters of a fixed non-contiguous shape. For example, the 3-grams of shape *##_#* in the string *ACAGCT* are *AC_G*, *CA_C* and *AG_T*. This concept is similar to the *anchors* we have used in our work, but they have different focuses.

Gapped q-gram is mostly used in string matching problems to filter candidates that could potentially match a target sequence. Most of these studies are focused on how to find some optimal "shapes" that could maximize the filtering effects. S. Burkhardt *et al* [23], [9] proposed to use gapped q-grams in a string matching problem and they also showed how to use experiments to choose the shape of gapped q-grams. M. Fontaine [24] introduced a method of selecting shapes of gapped q-grams for approximate string matching. A very popular local alignment tool named PatternHunter [25], [26] is also based on gapped q-gram. They used gapped q-grams as seeds to find possible aligned regions. It was shown that the problem of finding even one optimal gapped q-gram seed is NP-hard.

There is another work [27] that also represented a motif as a set of anchors like we did. It is used to search for occurrences of a motif in a set of sequences, which is different from our problem setting as we do not know the motif beforehand.

Record matching and deduplication. This problem is trying to identify records in a database that refer to the same entity. A commonly used technique in this domain is called "blocking" which divides the database into blocks and compare only the records that fall into the same block [28], [29]. This idea is similar to our pre-processing strategy, but it cannot be directly applied to motif discovery. Our design of anchor representation, initial anchor selection and iterative anchor set adjustment is essential for successfully identifying motifs.

To tolerant data heterogeneity, some methods [30], [31] use q-gram to do matching and blocking. However, none of the methods have adopted gapped q-gram in any way. So, our anchor (gapped q-grams with variable shapes) based clustering algorithm may be also interesting to this domain.

Time series motifs. A time series is a sequence of numerical and continuous data points. Time series motifs are frequent repeated patterns in time series data. Several recent papers

[32], [33] targeted on this problem, but their problem settings are totally different from ours.

VIII. Conclusion

In this work, we examined the motif discovery problem in the context of big data, where massive short sequences are generated by the newest sequencing techniques. Existing motif finding algorithms usually work only in a small scale or have to trade accuracy for efficiency. Our strategy is not to develop another motif finding algorithm and make it scalable. Instead, we resort to a different methodology which clusters a sequence dataset into multiple small subsets and then reuses existing motif finding algorithms. This strategy is generic. Our experimental results are extremely appealing. The anchor-based clustering approach can reduce the runtime in more than two orders of magnitude, without losing any accuracy. Sometimes it even discovers more motifs.

Acknowledgment

This work was partially supported by the National Science Foundation grants IIS-0954125.

References

[1] F. Zambelli, G. Pesole, and G. Pavesi, "Motif discovery and transcription factor binding sites before and after the next-generation sequencing era," *Briefings in bioinformatics*, vol. 14, no. 2, pp. 225–237, 2013.

[2] J. T. Ballew, J. A. Murray, P. Collin, M. Mäki, M. F. Kagnoff, K. Kaukinen, and P. S. Daugherty, "Antibody biomarker discovery through in vitro directed evolution of consensus recognition epitopes," *Proceedings of the National Academy of Sciences*, vol. 110, no. 48, pp. 19 330–19 335, 2013.

[3] T. L. Bailey and C. Elkan, "Fitting a mixture model by expectation maximization to discover motifs in bipolymers," *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, pp. 28–36, 1994.

[4] T. L. Bailey, "Dreme: motif discovery in transcription factor chip-seq data," *Bioinformatics*, vol. 27, no. 12, pp. 1653–1659, 2011.

[5] P. Machanick and T. L. Bailey, "Meme-chip: motif analysis of large dna datasets," *Bioinformatics*, vol. 27, no. 12, pp. 1696–1697, 2011.

[6] J. E. Reid and L. Wernisch, "Steme: efficient em to find motifs in large data sets," *Nucleic acids research*, vol. 39, no. 18, p. e126, 2011.

[7] T. Kim, M. S. Tyndel, H. Huang, S. S. Sidhu, G. D. Bader, D. Gfeller, and P. M. Kim, "Musi: an integrated system for identifying multiple specificity from very large peptide or nucleic acid data sets," *Nucleic acids research*, 2011.

[8] C. E. Grant, T. L. Bailey, and W. S. Noble, "Fimo: Scanning for occurrences of a given motif," *Bioinformatics*, vol. 27, no. 7, pp. 1017–1018, 2011.

[9] S. Burkhardt and J. Kärkkäinen, "One-gapped q-gram filters for levenshtein distance," in *Combinatorial pattern matching*. Springer, 2002, pp. 225–234.

[10] T. L. Bailey, M. Boden, F. A. Buske, M. Frith, C. E. Grant, L. Clementi, J. Ren, W. W. Li, and W. S. Noble., "Meme suite: tools for motif discovery and searching," *Nucleic acids research*, vol. 37, no. 2, pp. 202–208, 2009.

[11] M. Andreatta, O. Lund, and M. Nielsen, "Simultaneous alignment and clustering of peptide data using a gibbs sampling approach," *Bioinformatics*, vol. 29, no. 1, pp. 8–14, 2013.

[12] M. Sahli, E. Mansour, and P. Kalnis, "Acme: A scalable parallel system for extracting frequent patterns from a very long sequence," *The VLDB Journal*, vol. 23, no. 6, pp. 871–893, 2014.

[13] I. R. Rebollo, M. Sabisz, V. Baeriswyl, and C. Heinis, "Identification of target-binding peptide motifs by high-throughput sequencing of phage-selected peptides," *Nucleic acids research*, vol. 42, no. 22, pp. e169–e169, 2014.

[14] W. L. Matochko, K. Chu, B. Jin, S. W. Lee, G. M. Whitesides, and R. Derda, "Deep sequencing analysis of phage libraries using illumina platform," *Methods*, vol. 58, no. 1, pp. 47–55, 2012.

[15] C. Jia, M. B. Carson, and J. Yu, "A fast weak motif-finding algorithm based on community detection in graphs," *BMC bioinformatics*, vol. 14, no. 1, pp. 1471–2105, 2013.

[16] D. E. Schones, P. Sumazin, and M. Q. Zhang, "Similarity of position frequency matrices for transcription factor binding sites," *Bioinformatics*, vol. 21, no. 3, pp. 307–313, 2005.

[17] L. Marsan and M. Sagot, "Algorithms for extracting structured motifs using a suffix tree with an application to promoter and regulatory site consensus identification," *Journal of Computational Biology*, vol. 7, no. 3-4, pp. 345–362, 2000.

[18] G. Pavesi, G. Mauri, and G. Pesole, "An algorithm for finding signals of unknown length in dna sequences," *Bioinformatics*, vol. 17, no. 1, pp. S207–S214, 2001.

[19] T. L. Bailey, N. Williams, C. Misleh, and W. W. Li., "Meme: discovering and analyzing dna and protein sequence motifs," *Nucleic acids research*, vol. 34, no. 2, pp. 369–373, 2006.

[20] K. Katoh, K. Misawa, K. Kuma, and T. Miyata, "Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic acids research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[21] A. F. Neuwald, J. S. Liu, and C. E. Lawrence, "Gibbs motif sampling: detection of bacterial outer membrane protein repeats," *Protein science*, vol. 4, no. 8, pp. 1618–1632, 1995.

[22] J. D. Hughes, P. W. Estep, S. Tavazoie, and G. M. Church, "Computational identification of cis-regulatory elements associated with groups of functionally related genes in saccharomyces cerevisiae," *Journal of molecular biology*, vol. 296, no. 5, pp. 1205–1214, 2000.

[23] S. Burkhardt and J. Kärkkäinen, "Better filtering with gapped q-grams," *Fundamenta informaticae*, vol. 56, no. 1, pp. 51–70, 2003.

[24] M. Fontaine, S. Burkhardt, and J. Kärkkäinen, "Bdd-based analysis of gapped q-gram filters," *International Journal of Foundations of Computer Science*, vol. 16, no. 06, pp. 1121–1134, 2005.

[25] B. Ma, J. Tromp, and M. Li, "Patternhunter: faster and more sensitive homology search," *Bioinformatics*, vol. 18, no. 3, pp. 440–445, 2002.

[26] M. Li, B. Ma, D. Kisman, and J. Tromp, "Patternhunter ii: Highly sensitive and fast homology search," *Journal of Bioinformatics and Computational Biology*, vol. 2, no. 03, pp. 417–439, 2004.

[27] E. Giaquinta, S. Grabowski, and E. Ukkonen, "Fast matching of transcription factor motifs using generalized position weight matrix models," *Journal of Computational Biology*, vol. 20, no. 9, pp. 621–630, 2013.

[28] R. Baxter, P. Christen, and T. Churches, "A comparison of fast blocking methods for record linkage," in *ACM SIGKDD*, vol. 3. Citeseer, 2003, pp. 25–27.

[29] U. Draisbach and F. Naumann, "A generalization of blocking and windowing algorithms for duplicate detection," in *Data and Knowledge Engineering (ICDKE), 2011 International Conference on*. IEEE, 2011, pp. 18–24.

[30] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and efficient fuzzy match for online data cleaning," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. ACM, 2003, pp. 313–324.

[31] C. Xiao, W. Wang, and X. Lin, "Ed-join: an efficient algorithm for similarity joins with edit distance constraints," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 933–944, 2008.

[32] N. Begum and E. Keogh, "Rare time series motif discovery from unbounded streams," *Proceedings of the VLDB Endowment*, vol. 8, no. 2, pp. 149–160, 2014.

[33] Y. Li, U. Leong Hou, M. L. Yiu, and Z. Gong, "Quick-motif: An efficient and scalable framework for exact motif discovery." ICDE, 2015.