Discovering Consensus Patterns in Biological Databases

Mohamed Y. ElTabakh[1], Walid G. Aref[1],
Mourad Ouzzani[2], and Mohamed H. Ali[1]

[1] Dept. of Computer Science, Purdue University, West Lafayette IN 47906, USA,
[2] Cyber Center, Purdue University, West Lafayette IN 47906, USA,
meltabak,aref,mourad,mhali@cs.purdue.edu

Abstract. Consensus patterns, like motifs and tandem repeats, are highly conserved patterns with very few substitutions where no gaps are allowed. In this paper, we present a progressive hierarchical clustering technique for discovering consensus patterns in biological databases over a certain length range. This technique can discover consensus patterns with various requirements by applying a post-processing phase. The progressive nature of the hierarchical clustering algorithm makes it scalable and efficient. Experiments to discover motifs and tandem repeats on real biological databases show significant performance gain over non-progressive clustering techniques.

1 Introduction

A consensus pattern is a highly conserved pattern with very few substitutions where no gaps are allowed within the pattern. Discovering consensus patterns has several applications especially in biological databases for the case of motifs and tandem repeats. Motifs are highly conserved patterns that appear in the upstream region of genes. Motifs are regulatory elements that regulate the expression of genes and hence the functionality of the cell. Tandem repeats are highly conserved patterns too. They appear several times after each other in a DNA sequence. The regions in which tandem repeats appear are called *repeat regions*. Tandem repeats are considered DNA signatures, and have an important evolutionary role [30]. Discovering motifs and tandem repeats in biological databases is crucial for understanding the genetics of the cell. Discovering consensus patterns raises several challenges that make applying data mining tools such as clustering techniques a nontrivial task. In particular, the patterns that we are looking for are usually unknown, the length of the consensus patterns is unknown.

In this paper, we propose a progressive hierarchical clustering technique, called Bio-CP, for discovering consensus patterns. Bio-CP discovers consensus patterns by clustering similar patterns together over a range of fixed lengths. Each cluster represents a candidate set of consensus patterns. The processing of Bio-CP is divided into phases. In each phase, we discover the candidate consensus patterns for a certain length. Then, we proceed to the next phase in an incremental manner to obtain the candidate patterns with the subsequent length. At

the end of each phase, we perform a post-processing phase to apply any domain specific requirements over the candidate patterns. Bio-CP is applicable to a wide range of applications since it allows any domain specific requirements to be applied independently in a post-processing phase. Furthermore, Bio-CP executes progressively and hence significantly reduces the processing overhead compared to non-progressive clustering techniques. However, in its original form, Bio-CP involves a high overhead in the first phase due to computing and storing a large distance matrix. To address this issue, we propose several scalability techniques to reduce both the storage and CPU overheads.

The rest of the paper is organized as follows. We discuss the related work in Section 2. In Section 3, we present Bio-CP concepts and methods. Scalability issues are discussed in Sections 4. The experimental results are presented in Section 5. We conclude in Section 6.

2 Related Work

Pattern similarity is studied in several application domains. In data mining, frequent pattern mining is the problem of discovering similar patterns that appear a number of times above a certain threshold in the database, e.g., [2, 4]. Frequent pattern mining techniques cannot handle efficiently the problem of discovering consensus patterns for the following reasons: (1) Consensus patterns allow approximate matching, whereas frequent pattern mining techniques (even the techniques that allow gaps) usually search for only exact matches. (2) Frequent pattern mining techniques involve high overhead in the early phases in which too many short frequent patterns are discovered. The length of these short patterns can be out of the interesting range of the consensus patterns. Similarity search techniques aim at searching for a query string in a database of sequences, e.g., [1, 3, 5, 17]. Several data structures are developed for searching string and sequence data, e.g., suffix trees, e.g., [28, 29], and suffix arrays, e.g., [27, 29]. While these techniques and data structures are related to our targeted problem, they cannot be applied directly for discovering consensus patterns since we do not have a query string to search for in the first place.

Clustering techniques rely on grouping similar patterns or objects together [10, 14]. MOPAC [13] is an agglomerative clustering technique to discover motif consensus patterns in biological databases. MOPAC solves the problem for a specific motif length. For a length range, MOPAC needs to be re-executed for each candidate motif length. Usually, using existing clustering techniques to discover consensus patterns is limited to discovering such consensus for a specific length, which is not general enough since the length of such patterns is not known a priori. In contrast, our techniques extend clustering techniques to work over a range of lengths in a scalable way.

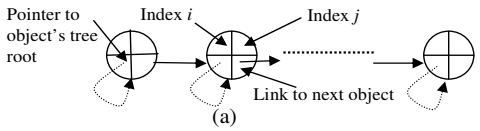
Several statistical and non-statistical approaches have been proposed in biological databases. Most of these approaches target either motifs or tandem repeats but not both. Examples of statistical techniques for discovering motif patterns can be found in [7, 8, 16, 20]. A statistical technique for finding tandem repeats in DNA sequences is proposed in [9]. Other non-statistical techniques in-

cludes PROJECTION [11], COPIA [21], and WINNOWER [24] for discovering motifs and one technique [19, 22] for discovering tandem repeats.

3 Bio-CP Concepts and Methods

Bio-CP is a progressive technique for discovering consensus patterns for a given database of sequence, a length range, and a threshold for clustering patterns together at a certain length. More precisely, given (1) a database D of N sequences, i.e., D=$\{S_1, S_2, ..., S_N\}$ where each sequence S_i ($1 \leq i \leq N$) has length L_i, (2) a length range $[min\_len \cdots max\_len]$ over which consensus patterns need to be discovered, and (3) a user-specified threshold ϵ, $0 \leq \epsilon \leq 1$, that specifies the distance threshold beyond which no further patterns can be clustered together (a small value indicates tighter clusters and higher similarity among the discovered consensus patterns), Bio-CP returns a set of clusters; each cluster represents a group of consensus patterns for a given length within the user-specified length range. The distance between a pair of patterns can be measured using the Hamming Distance [15] or any distance metric that does not allow insertions or deletions like the substitution matrix. In this paper, we use the Hamming Distance.

Bio-CP proceeds in phases where each phase discovers the clusters for a certain length within the specified range. In the first phase (P_{min_len}), we discover the clusters for patterns of length *min_len*, and in the subsequent phases (P_{min_len+1}, P_{min_len+2}, $\cdots$, P_{max_len}), we incrementally extend existing patterns to discover the subsequent clusters corresponding to each length. Initially, Bio-CP divides the sequences in the database D into sliding windows of length *min_len*. Each sliding window W_{ij}^k is identified by three indexes i, j, and k, where i specifies the sequence identifier, j specifies the start position within sequence S_i, and k specifies the length of the sliding window. The index k is initially set to *min_len* and k increases by one as we move from one phase to the next one. We build a list Q that contains all possible sliding windows (Figure 1(a)). Each node in the list represents a pattern or sliding window, and consists of four fields: index i, index j, a pointer to the pattern, and a pointer to the next pattern in the list. These nodes will form the leaf level of a clustering hierarchy that will be built on top of them. In the rest of the paper, we use the terms 'window' and 'pattern' interchangeably to refer to the sliding window pattern.



(a)

Pointer to first obj.	Pointer to second obj.	Ref_distance	Distance	Cluster_pointer
O_i	O_j	D_{i-j}	D_{i-j}	NULL
---	---	---	---	---

(b)

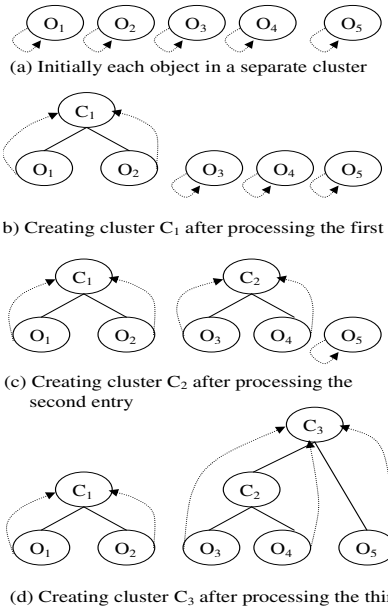
Fig. 1. (a) Patterns list structure (Q). (b) Sorted list structure (S)

In phase P_{min_len}, Bio-CP starts by measuring the distance between each possible pair of patterns on length *min_len*, and then inserting an entry for this pair into a sorted list S. List S is maintained in an ascending order of distance values such that the pairs that are the closest to each other will be at the top of the list and will be the first to be merged together. The structure of list S consists of five columns (see Figure 1(b)): The first column holds a pointer to a pattern in Q, the second column holds also a pointer to a pattern in Q, the third and fourth columns hold distance values, and if that entry will create a new node in the clustering hierarchy, then the fifth column will hold a pointer to this new node, otherwise it is NULL. The values in the *Ref_distance* column serve as reference values and will not change while proceeding from one phase to another. However, the values in the *Distance* column will be updated as we progress in the algorithm.

Our target is to generate clusters over a range of lengths. Thus, it is important to normalize the distance values relatively to the patterns lengths (the distance between any pair is always between 0 and 1.) Furthermore, generating the distance matrix as a sorted list (S) does not involve additional overhead. Indeed, since we have prior knowledge of the possible values that will be inserted into the list, (the Hamming Distance between two patterns of length L is between 0 and L) then each new entry can be directly hashed to its proper sorted location in the list. We maintain a linked list for each distance value. After all the entries are inserted into their corresponding lists, we link these lists together to form list S. Bio-CP is applied over two clustering metrics, single-link [26], presented in Section 3.1 and complete-link [18] presented in Section 3.2. Typically, single-link clustering involves less processing overhead than complete-link clustering. However, in the case of large databases, single-link clustering generates poor quality clusters due to the chaining effect [23].

3.1 Progressive Single-link Clustering

In single-link clustering, the distance between two clusters is the distance between the closest pair of patterns in these clusters. The objective is to merge in each step the closest pair of clusters into one cluster. We refer to Bio-CP with single-link clustering by Bio-CP/S. In the first phase P_{min_len}, we scan the list S until we reach the first entry with distance larger than or equal to ϵ. Any entry after that entry has a distance value larger than or equal to ϵ. For each entry, say e, we check if the two patterns in e belong to the same cluster; If this is the case, then e is skipped, otherwise, the corresponding two clusters, say C_i and C_j, are merged into a new cluster C_k. The merge operation involves three steps: (1) Create a new cluster C_k in the clustering hierarchy. Make C_i and C_j children for C_k. (2) Traverse clusters C_i and C_j to reach all their members at the leaf level. Update the pointer associated with each member to point to cluster C_k instead of C_i or C_j. (3) Add a pointer in entry e to point to the newly created cluster C_k. Figure 2 shows an example of clustering five patterns. Note that the fourth entry in list S does not create a new cluster node, therefore its *Cluster_pointer* value remains NULL. Recall that each window is initially in a separate cluster and the pointer associated with it points to itself. Also all



Ptr to first obj.	Ptr to second obj.	Ref_distance	Distance	Cluster_pointer
O_1	O_2	$D_{1\text{-}2}$	$D_{1\text{-}2}$	NULL
O_3	O_4	$D_{3\text{-}4}$	$D_{3\text{-}4}$	NULL
O_3	O_5	$D_{3\text{-}5}$	$D_{3\text{-}5}$	NULL
O_4	O_5	$D_{4\text{-}5}$	$D_{4\text{-}5}$	NULL
---	---	---	---	---

List *S* before the processing

Ptr to first obj.	Ptr to second obj.	Ref_distance	Distance	Cluster_pointer
O_1	O_2	$D_{1\text{-}2}$	$D_{1\text{-}2}$	C1
O_3	O_4	$D_{3\text{-}4}$	$D_{3\text{-}4}$	C2
O_3	O_5	$D_{3\text{-}5}$	$D_{3\text{-}5}$	C3
O_4	O_5	$D_{4\text{-}5}$	$D_{4\text{-}5}$	NULL
---	---	---	---	---

List *S* after the processing

Fig. 2. Updating the clustering hierarchy and S after processing the first four entries

the pointers in the sorted list S are initially NULL. Maintaining the pointers in steps (2) and (3) is crucial for efficient processing in the subsequent phases. For example the pointer associated with each window, and maintained in step (2), allows us to detect if two members belong to the same cluster in a constant time.

Now that we generated the clusters of the consensus patterns of length *min_len*, we can start generating the clusters for the subsequent phases P_{min_len+1}, P_{min_len+2}, $\cdots$, P_{max_len} in an incremental way. Since the windows are sorted in S based on length *min_len*, then all the subsequent phases will reference these windows for further extensions. In addition, these subsequent phases will reference the distance values measured in phase P_{min_len}. For this reason, each entry in S keeps a copy of this reference distance value ($Ref\_Distance$).

Generally speaking, generating the clusters in any phase P_{min_len+t} involves extending the windows in phase P_{min_len} by t letters. Our progressive processing is based on two key observations.

Observation 1 *The change in the distance values among the windows due to extending them by* t *letters is bound. The bound is computed using the Equation* $\Delta_{max\_t} = \frac{t}{min\_len+t}$.

Observation 2 *The value of* Δ_{max_t} *increases monotonically with respect to* t*; the size of C(t) is always increasing: C(t-1)* $\subseteq$ *C(t); where* $2 \leq$ t.

Observation 1 implies that the maximum change in the distance value due to appending t letters is $\pm\Delta_{max\_t}$. Therefore, the only entries in S that may cross

the threshold ϵ in phase P_{min_len+t}, and thus may change the clustering hierarchy are within $\pm\Delta_{max\_t}$ from ϵ. Let *C(t)* be the set of entries that are within $\pm\Delta_{max\_t}$ from ϵ; entries in *C(t)* are the only entries that need to be updated in phase P_{min_len+t}. This would simply consist in comparing the newly appended t letters and modifying the distance value. Finding the entries in S that belong to $C(t)$ is performed by scanning S in both directions starting from the last processed entry in phase P_{min_len} (i.e., the last entry that has $Ref\_Distance < \epsilon$). All entries that have $Ref\_Distance$ value within $\pm\Delta_{max\_t}$ from ϵ will be in $C(t)$.

Observation 2 implies that the entries in *C(t)* that need to be updated in phase P_{min_len+t} can be divided into two types: (1) the entries that were in *C(t-1)* in the previous phase $P_{min\_len+t-1}$ and (2) the entries that are added during the current phase P_{min_len+t}. The entries in the latter type require comparing the newly appended t letters to update their distances. In contrast, the entries in the former type require only one letter comparison (the last appended letter) to update their distances. In summary, Observation 1 determines which entries to update and Observation 2 allows for an efficient update of these entries' distance values. The main framework for progressive processing is given in Figure 3. Note that the equation for Δ_{max_t} is general for the Hamming distance as well as for any substitution matrix having values ranging from 0 to any positive number.

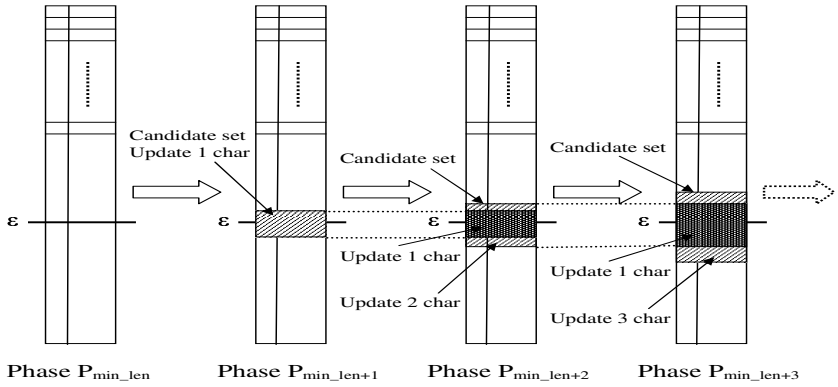


Fig. 3. Progressive processing for the candidate entries

The next step is to identify which of these entries trigger a change over the existing hierarchy; i.e., split or merge operations:

- If the distance before the update is less than ϵ and the distance after the update is still less than ϵ, then this entry will not trigger any change.
- If the distance before the update is larger than or equal to ϵ and the distance after the update is still larger than or equal to ϵ, then this entry will not trigger any change.
- If the distance before the update is less than ϵ and the distance after the update is larger than or equal to ϵ, then this entry will trigger a split to take place only if the entry has a cluster attached to it.

– If the distance before the update is larger than or equal to ϵ and the distance after the update is less than ϵ, then this entry will trigger a merge to take place only if the windows in this entry belong to different clusters.

If none of the entries in $C(t)$ triggers splits or merges, then the clusters in the current phase P_{min_len+t} will be the same as the clusters generated in the previous phase $P_{min\_len+t-1}$. If there are entries in $C(t)$ that trigger splits or merges, then the entries in $C(t)$ need to be scanned to perform the required changes.

Performing a split operation, triggered by an entry e in list S, on a cluster C involves three steps: (1) Traverse the left child cluster of C, i.e., C_1, to reach all its members in the leaf level. Update the pointer associated with each member to point to cluster C_1 instead of C. (2) Traverse the right child cluster of C, i.e. C_2, to reach all its members in the leaf level. Update the pointer associated with each member to point to cluster C_2 instead of C. (3) Invalidate the pointer attached to entry e by setting it to NULL.

Recall that the pointers maintained at the leaf members of the hierarchy and the entries in the sorted list allow performing the identification process efficiently. Detecting whether an entry in the sorted list has a cluster attached to it is performed in constant time by checking the pointer associated with this entry. Also, detecting whether two windows belong to different clusters is performed in constant time by checking the pointers associated with these windows. Finally, performing a merge or split operation is performed, as explained previously, in the order of the cluster size. The issue of maintaining the hierarchy pointers is thightly related to the union-find problem [25]. For example, another way of maintaining the pointers is to make each node points to its direct parent only. In that case, detecting whether two windows belong to different clusters or not is performed in order of the cluster size (in the worst case), however a merge or split operation is performed in a constant time.

It is important to keep the clustering hierarchy consistent while performing the splitting and merging operations. To achieve this consistency, all the split operations are performed before any merge operations in the backward direction (i.e., we process the entries of $C(t)$ in a bottom-up fashion), and then all the merge operations are performed in the forward direction (i.e., we process the entries of $C(t)$ in a top-down fashion). Algorithm 1 describes the incremental processing for discovering the clusters at phase P_{min_len+t}. The following lemma states that Bio-CP/S produces exactly the same clusters as a non-progressive single link technique. We omit the proof due to the lack of space.

Lemma 1. *Bio-CP/S produces exact results in all phases compared to the results generated by the non-progressive technique.*

3.2 Progressive Complete-link Clustering

Bio-CP/S has two advantages: (i) it produces exact results and (ii) the single-link clustering it employs has less processing overhead than that of complete-link

Algorithm 1 Progressive clustering at phase P_{min_len+t}

Inputs From the Previous Phase:
-Sorted list S
- The Clustering hierarchy from the previous phase $P_{min\_len+t-1}$
-Δ_{max_t-1} from the previous phase $P_{min\_len+t-1}$
Computations at the Current Phase:
-Compute Δ_{max_t}
-Entries within $\pm\Delta_{max\_t-1}$ from ϵ are updated
by comparing the last letter appended to the patterns
-Entries within $\pm\Delta_{max\_t}$ but outside $\pm\Delta_{max\_t-1}$ from ϵ
are updated by comparing the last t letters appended to the patterns
-Identify the entries that cross ϵ and trigger
split or merge operations
-Perform the required split operations in the backward
direction (process the entries bottom-up)
-Perform the required merge operations in the forward
direction (process the entries top-down)

clustering. However, for large databases, Bio-CP/S produces clusters with poor quality due to the chaining effect. In this section, we propose Bio-CP/C where we apply Bio-CP using the complete-link metric for clustering. In complete-link clustering, the distance between two clusters is measured as the distance between the farthest pair in the two clusters. Therefore, the diameter of any resulting cluster is always less than the user specified threshold ϵ.

Building the clustering hierarchy in the first phase, P_{min_len}, is similar to building the hierarchy using the single-link metric, except in the merging condition. After constructing the sorted list S, S is scanned until we reach the first entry with a distance larger than or equal to ϵ. For each entry e in S, if the two patterns in that entry, i.e., W_1 and W_2, belong to the same cluster then e is skipped, otherwise if W_1 and W_2 belong to different clusters, i.e., C_1 and C_2, then Bio-CP/C checks whether W_1 and W_2 are the farthest pair in C_1 and C_2. If this is the case, then C_1 and C_2 are merged into a new cluster C, otherwise, entry e is skipped. The merge step in Bio-CP/C is similar to the merge step in Bio-CP/S except that we do not need to maintain the pointers attached to the entries in list S (Step 3 in the merge procedure). These pointers are used in Bio-CP/S to identify efficiently which entries will trigger splits in the subsequent phases. However, in Bio-CP/C, the splitting condition is different and does not depend on these pointers.

After generating the clusters of the consensus patterns of length min_len in phase P_{min_len}, Bio-CP/C generates the clusters for the subsequent phases P_{min_len+1}, P_{min_len+2}, $\cdots$, P_{max_len}. In each phase, i.e., P_{min_len+t}, Bio-CP/C computes $C(t)$ (the set of entries to be updated) and then updates their distance values in the same way as that of single-link clustering. Identifying which entries will trigger splits or merges is performed as follows:

- If the distance before the update is less than ϵ and the distance after the update is still less than ϵ, then this entry will not trigger any change.
- If the distance before the update is larger than or equal to ϵ and the distance after the update is still larger than or equal to ϵ, then this entry will not trigger any change.
- If the distance before the update is less than ϵ and the distance after the update is larger than or equal to ϵ, then this entry will trigger a split to take place whenever the windows in this entry belong to the same cluster.
- If the distance before the update is larger than or equal to ϵ and the distance after the update is less than ϵ, then this entry will trigger a merge to take place only if the windows in this entry belong to different clusters and are the farthest in their clusters.

The splitting condition in Bio-CP/C is less strict than that in Bio-CP/S. In Bio-CP/S, the only entry that can split an existing cluster is the one that created the cluster. However, in Bio-CP/C, any pair of windows that belong to the same cluster will split the cluster if their distance becomes larger than or equal to ϵ. This is why Bio-CP/C does not maintain pointers with the entries in the sorted list S. In addition, the merging condition in Bio-CP/C is more strict than in Bio-CP/S; a pair of windows will merge two clusters only if this pair is the farthest pair in these clusters.

Bio-CP/C and Bio-CP/S use the progressive technique in almost the same way. However, the results from Bio-CP/C are approximate in comparison with the non-progressive technique as stated in the following lemma. We omit the proof due to the lack of space.

Lemma 2. *Bio-CP/C produces approximate results in comparison to the results produced by the non-progressive technique. However, the generated clusters still satisfy the condition that the diameter of any generated cluster is less than ϵ.*

3.3 Post-Processing Phase

The post-processing phase allows to apply any available domain-specific requirements to refine the discovered consensus patterns. Here are some examples of biological requirements that can be applied for discovering motifs and tandem repeat. (a) The user may specify a minimum size for the desired clusters, such that any cluster that contains fewer patterns than the specified minimum size will be ignored. (b) The user may specify any position requirements for the discovered patterns. For example, whether the desired tandem repeats should appear immediately after each other or a gap is allowed between the repeats. (c) A DNA palindrome [8] is a sequence whose inverse complement is the same as the original sequence. The user may specify that the desired motif patterns should have at least a certain palindrome degree. In this case, we check each pattern against the specified palindrome threshold and we qualify only the patterns with higher palindrome degree. (d) The user may specify whether the desired

consensus patterns can overlap or not. For example, the occurrences of real motifs usually do not overlap each other. In biological databases, although motifs and tandem repeats have different post-processing requirements, Bio-CP allows both types of patterns to be discovered in a single run.

4 Scalability Issues in Bio-CP

While Bio-CP shows significant improvement compared to non-progressive clustering techniques, it still involves a high overhead in the first phase due to computing and storing the distance matrix. In this section, we propose a top-k nearest-neighbor method to reduce the storage overhead and a heuristic to significantly reduce the number of comparisons needed to get the top-k nearest-neighbors for each pattern. We call the new algorithm Bio-CP/K. Another issue with Bio-CP is that at some point the overhead to process and maintain the entries in $C(t)$ may become comparable to the overhead of resorting the entries in list S. Bio-CP may need to reset the computations for list S. Due to space limitations, we do not discuss the possible solutions for this issue.

4.1 Storage Reduction using Top-k Nearest-neighbor

The top-k nearest-neighbor method is well known to reduce the size of the distance matrix [12]. In this method, only the top-k nearest-neighbors for each pattern are stored and clustering techniques are applied over these stored patterns only. While suing the top-k nearest-neighbor method with hierarchical clustering techniques is straightforward, using it with Bio-CP is nontrivial. Due to progressive processing, the lengths of the patterns increase from one phase to the next. Hence, the top-k nearest-neighbors for each pattern may change across phases. Bio-CP/K ensures that, in each phase, every pattern will have its top-k nearest-neighbor patterns among the patterns being processed.

Since patterns expand across the phases, Bio-CP/K stores for each pattern the union of the pattern's top-k nearest-neighbors over the different phases. For example, if the set of top-k nearest-neighbors for pattern x in phase P_{min_len+t} is K_t, then Bio-CP/K stores for x the union set $\cup_x = \cup(K_t)$; where $0 \leq t \leq max\_len - min\_len$. In this case, while x expands over the different phases, Bio-CP/K ensures that x's top-k nearest-neighbors are among the patterns being processed. Bio-CP/K computes the nearest-neighbors for each pattern while constructing the sorted list S and before any of the processing phases. Constructing the union of the top-k nearest-neighbors for pattern x is performed incrementally as follows:

1. Initially $\cup_x$ is empty
2. Compare pattern x with the remaining patterns based on length *min_len*. The result from the comparison is a list L sorted based on the distance values.
3. Add the first k patterns in L to $\cup_x$. Let the distance value of the last pattern of these k patterns be K_Dist.

4. To expand pattern x and re-compute x's top-k nearest-neighbors, we repeat the following steps for t from 1 to $(max\_len - min\_len)$:
   - The maximum change in the distance value due to appending t letters is $\Delta_{max\_t} = \frac{t}{min\_len+t}$
   - Check the patterns in L that are within $\pm 2\Delta_{max\_t}$ from K_Dist by appending and comparing t letters to the patterns with pattern x. These patterns are the only ones that may substitute each other as the top-k nearest-neighbors.
   - If any pattern of the new top-k nearest-neighbors is not in $\cup_x$, we add it to $\cup_x$.

The reason for considering $\pm 2\Delta_{max\_t}$ instead of $\pm\Delta_{max\_t}$ as in Bio-CP is that we do not care about the absolute value of K_Dist. Instead, we care about the relative order among the patterns around the K_Dist point.

After computing x's nearest-neighbors $\cup_x$, Bio-CP/K inserts pairs (x, y) $\forall y \in \cup_x$ in the sorted list S. Then the progressive processing is applied over S as discussed in Section 3. Comparing Bio-CP/K to the non-incremental technique, we observe the following. First, Bio-CP/K involves some overhead in constructing $\cup_x$. However, the non-incremental technique has to perform steps (2) and (3) independently for each pattern in each length, which clearly involves a much higher overhead. Second, since each pattern will have at least its top-k nearest-neighbors in S in each phase, then the results from Bio-CP/K are at least as good as the non-incremental technique.

4.2 Processing Time Reduction

In this section, we propose a heuristic to reduce the average number of comparisons needed to find the nearest neighbors for each pattern in Bio-CP/K. The main idea is that from comparing a given pattern x with the other patterns in the database, we can obtain the top-k nearest-neighbors for several patterns other than x in an efficient and less expensive way.

When pattern x is compared with other patterns in the database and list L is constructed (see Section 4.1), the list is logically partitioned into groups G_0, G_1, ..., G_{min_len}; where group G_i contains the patterns that have i mismatches with pattern x. It is clear that we can directly get the nearest neighbors for all the patterns that belong to group G_0. They will have the same nearest neighbors as pattern x. Similarly, for any pattern y in group G_1, y's nearest neighbors can be obtained efficiently because we know to a large extent in which groups these nearest neighbors will be. For example, the number of mismatches between pattern y and any pattern in group G_0 is one, the number of mismatches between pattern y and any pattern in group G_1 is either zero, one or two, and the number of mismatches between pattern y and any pattern in group G_2 is either one, two or three, and so on. Therefore, the comparisons can be performed incrementally based on the need for more patterns to be added to the nearest neighbor list. In that case we avoid many unnecessary comparisons. We should note that the

patterns that match exactly with pattern y exist only in group G_1 and these patterns will have the same nearest neighbor list as y. The same idea applies for the other groups. However, the power of the heuristic relies on processing only the first few groups since the uncertainty in the number of mismatches is very small for these groups. Most of the comparisons will lead to patterns in the nearest neighbor list. Using the proposed heuristic allows one scan to the database to generate the required nearest neighbors for several patterns. As a result, the overall number of comparisons is reduced significantly.

5 Experimental Results

We evaluate the performance of Bio-CP for discovering motifs and tandem repeats using real datasets from the E.coli genome sequence. We consider two measures of performance; the processing time and the cluster validity. For the latter, we use *Jaccard* and *Rand coefficients* [14] to measure the similarity among the clusters generated from Bio-CP and the non-progressive clustering techniques. All measures for the non-progressive techniques are cumulative values to generate the desired clusters over multiple lengths. The non-progressive single-link

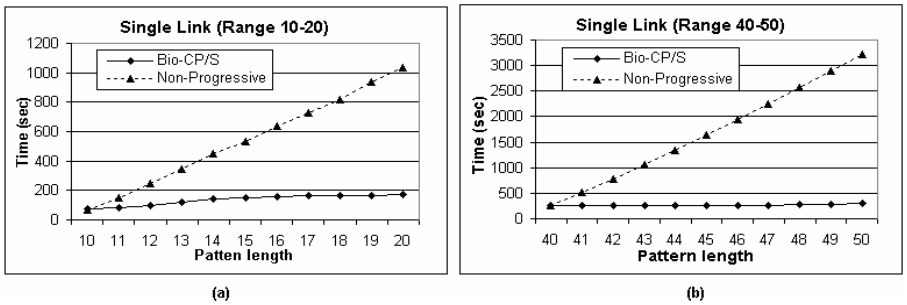
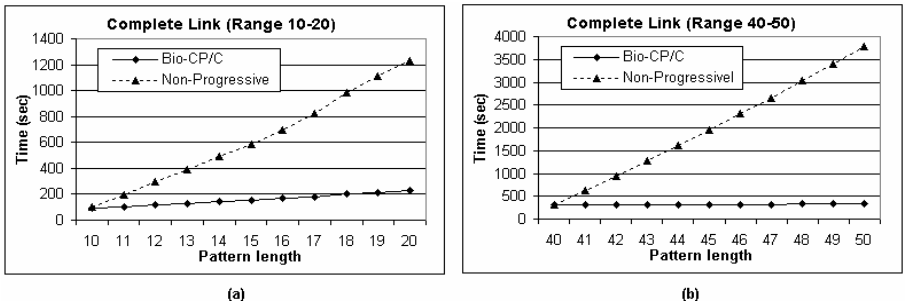


Fig. 4. Processing time of Bio-CP/S and the non-progressive technique

technique simulates the MOPAC algorithm [13]. In the first experiment, we measure the performance of Bio-CP/S. A file of size 25,000 bases is used, ϵ is assigned a value of 0.3, and two length ranges are evaluated. Figure 4 illustrates that Bio-CP/S and the non-progressive technique take almost the same time in the first phase in which the hierarchy is generated. However, in the subsequent phases, Bio-CP/S takes much less time than the non-progressive technique that rebuilds the hierarchy from scratch. Figure 4 shows that the processing time for building the hierarchy in the first phase in the case of the range [40...50] is higher than in the case of the range [10...20]. However, the time needed to progress from one phase to the next one in the case of the range [40...50] is less than in the case of the range [10...20]. The reason being that the effect of extending longer patterns is less than the effect of applying the same extension over shorter patterns. The clusters generated from both techniques in this experiment are exactly the same since we are using the single-link metric.

In Figure 5, we present the results of applying Bio-CP/C over the same file of size 25,000. The ϵ threshold is assigned a value of 0.5. The behavior of Bio-CP/C and non-progressive techniques is very similar to that in the case of



(a) (b)

Fig. 5. Processing time of Bio-CP/C and the non-progressive technique

Length	Jac. coeff	Rand coeff
10	1	1
11	0.884464	0.999987
12	0.841966	0.999955
13	0.808337	0.999941
14	0.833241	0.999951
15	0.824595	0.999948
16	0.793678	0.999923
17	0.78032	0.999923
18	0.802123	0.999932
19	0.781047	0.999923
20	0.75642	0.999902

(a)

Length	Jac. coeff	Rand coeff
40	1	1
41	0.917552	0.999991
42	0.832075	0.999985
43	0.835635	0.999985
44	0.819972	0.999974
45	0.830797	0.999982
46	0.802416	0.999972
47	0.807746	0.999972
48	0.824261	0.999978
49	0.785684	0.999969
50	0.788491	0.99997

(b)

Length	Jac. coeff	Rand coeff
10	0.9364	0.999989
11	0.873205	0.999985
12	0.848723	0.999983
13	0.841132	0.999983
14	0.807214	0.999977
15	0.815188	0.999982
16	0.780025	0.999968
17	0.81346	0.999982
18	0.768912	0.999959
19	0.74306	0.99994
20	0.719853	0.999926

(c)

Fig. 6. Clustering validation degree

Bio-CP/S. However, the generated clusters from Bio-CP/C in this experiment are approximate clusters. Figures 6(a) and 6(b) give the validity degree of the clusters generated in the case of the ranges [10...20] and [40...50], respectively. The figure illustrates that the coefficient values are very close to 1, which means that the clusters generated from incremental processing are very similar to the ones generated from the non-progressive technique. In addition, the figure illustrates that the *Rand* coefficient detects higher similarity degree among the clusters than the *Jaccard* coefficient. The reason for this difference is that the *Jaccard* coefficient does not take into account one similarity factor, namely the number of pairs, say (w_1, w_2), for which both clustering techniques assign w_1 and w_2 patterns to different clusters. The *Rand* coefficient takes this factor into account.

In Figure 7, we present the results of applying Bio-CP/KC, (Bio-CP using complete-link metric and the top-k nearest-neighbors method). We use a file of size 150,000 bases, with ϵ assigned to 0.5. In this experiment, we store the top 0.1% nearest-neighbors for each pattern. Figure 7(a) gives the processing overhead of the various techniques. The figure illustrates that the non-progressive technique involves infeasible processing overhead in the case of relatively large files. In Figure 7(b), we present the effect of applying Bio-CP/KC along with the proposed heuristic in computing the nearest-neighbors when compared to the naive way. The figure illustrates significant reduction in the processing time due to reducing the number of comparisons. The validity measure for the clusters generated in this experiment is given in Figure 6(c). Note that the coefficient

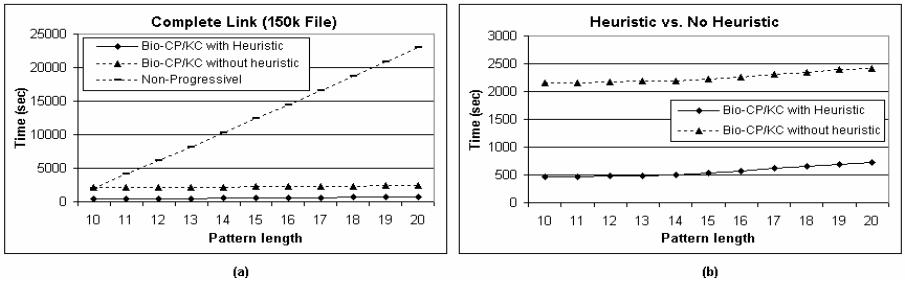


Fig. 7. (a) Processing time of Bio-CP/KC and the non-progressive technique. (b) Heuristic versus naive method

values for the patterns of length 10 are no longer equal to 1. This slight dissimilarity in the first phase is due to the difference between the nearest-neighbor sets maintained by Bio-CP/KC and the non-progressive techniques.

We run several experiments to compare Bio-CP/C with the MEME algorithm [6]. The closeness of the discovered motifs is highly affected by the ϵ threshold. With ϵ assigned to 0.3 or 0.4, Bio-CP/C usually splits the motifs discovered by MEME into multiple motifs. However, with ϵ assigned to 0.6, the motifs discovered by MEME are a subset of the motifs discovered by Bio-CP/C. This indicates that 0.6 is reasonable value for ϵ. Bio-CP/C always produces more candidate motifs than MEME. However, MEME annotates each motif with more information such as the E-value and background probabilities due its statistical nature.

6 Conclusions

In this paper, we proposed Bio-CP, a progressive hierarchical clustering technique for discovering consensus patterns, namely motifs and tandem repeats, in biological databases over a range of possible lengths. The progressive nature of the hierarchical clustering algorithm makes it scalable and efficient. Bio-CP is also applicable to a wide range of applications since any domain-specific requirements are applied in a post-processing phase. We also proposed several scalability techniques to enhance the performance of Bio-CP in terms of processing time and storage. Our experiments illustrated that Bio-CP scales very well with respect to the processing time, and the clustering validation degrees. In particular, Bio-CP has more than 500% processing time improvement.

References

1. C. C. Aggarwal. On effective classification of strings with wavelets. In *Proceedings of the 8th ACM SIGKDD*, 163–172, 2002.
2. B. Goethals. Survey on frequent pattern mining. *Manuscript*, 2003.
3. R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 69–84, 1993.
4. R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *VLDB*, 487–499, 1994.

5. W. G. Aref and D. Barbara. Supporting electronic ink databases. In *Information Systems: An International Journal 24(4)*, 303–326, 1999.
6. T. Bailey, C. Elkan, and B. Grundy. The meme system: Multiple EM for motif elicitation, http://bioweb.pasteur.fr/seqanal/motif/meme/.
7. T. L. Bailey and C. Elkan. Fitting a mixture model by expectation maximization to discover motifs in biopolymers. In *ISMB*, 28–36, 1994.
8. T. L. Bailey and C. Elkan. The value of prior knowledge in discovering motifs with meme. In *ISMB*, 21–29, 1995.
9. G. Benson. Tandem repeats finder: a program to analyze dna sequences. In *Nucleic Acids Research*, volume 27, 573–580, 1999.
10. P. Berkhin. Survey of clustering data mining techniques. San Jose, CA, 2002.
11. J. Buhler and M. Tompa. Finding motifs using random projections. In *RECOMB*, 69–76, 2001.
12. E. Fix and J. L. Hodges. Discriminatory analysis, nonparametric discrimination: Consistency properties. In *USAF School of Aviation Medicine, Project 21-49004,Report 4*, 1951.
13. R. Ganesh, T.R. Ioerger, and D.A. Siegele. Mopac: Motif finding by preprocessing and agglomerative clustering from microarrays. In *PSB*, 41–52, 2003.
14. M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering algorithms and validity measures. In *Tutorial paper, SSDBM*, 2001.
15. R. W. Hamming. Coding and information theory. In *Prentice-Hall*, 1980.
16. G.Z. Hertz and G.D. Stormo. Identifying dna and protein patterns with statistically significant alignments of multiple sequences. In *Bioinformatics*, 15:563–577, 1999.
17. H. V. Jagadish, N. Koudas, and D. Srivastava. On effective multi-dimensional indexing for strings. In *SIGMOD*, 403–414, 2000.
18. B. King. Step-wise clustering procedures. In *J. Am. Stat. Assoc. 69,*, 1967.
19. G. M. Landau and J. P. Schmidt. An algorithm for approximate tandem repeats. In *CPM*, 120–133, 1993.
20. C.E. Lawrence, S.F. Altschul, M.S. Boguski, J.S. Liu, A.F. Neuwald, and J.C. Wootton. Detecting subtle sequence signals a gibb's sampling strategy for multiple alignment. In *Science*, 262:208–214, 1993.
21. C. Liang. Copia: A new software for finding consensus patterns in unaligned protein sequences. In *Master thesis, University of Waterloo*, 2001.
22. G. Myers and M. Sagot. Identifying satellites and periodic repetitions in biological sequences. In *Journal of Computational Biology*, volume 10, 10–20, 1998.
23. G. Nagy. State of the art in pattern recognition. In *Proc. IEEE 56*, 1968.
24. P.A. Pevzner and S. Sze. Combinatorial approaches to finding subtle signals in dna sequences. In *ISMB*, 269–278, 2000.
25. J. A. La Poutr;. New techniques for the union-find problem. In *SIAM*, 54–63, 1990.
26. P. H. Sneath and R. R. Sokal. Numerical taxonomy. *Freeman,London,UK.*,1973.
27. W. B. Frakes and R. B. Yates, editors. *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.
28. E. M. McCreight. A space-economical suffix tree construction algorithm. *Journal of ACM*, 23(2):262–272, 1976.
29. D. Gusfield. *Algorithms on strings, trees, and sequences: computer science and computational biology.* Cambridge University Press, New York, NY, USA, 1997.
30. H. Hamada, M. Seidman, B. Howard, and C. Gorman. *Enhanced gene expression by the poly(dT-dG) poly(dC-dA) sequence* Molecular and Cellular Biology, 1984.