QOMA2: Optimizing the alignment of many sequences

Xu Zhang Tamer Kahveci
Computer and Information Science and Engineering
University of Florida , Gainesville, FL, 32611
{xuzhang,tamer}@cise.ufl.edu

***Abstract*—We consider the problem of aligning multiple protein sequences with the goal of maximizing the SP (Sum-of-Pairs) score, when the number of sequences is large. The QOMA (Quasi-Optimal Multiple Alignment) algorithm addressed this problem when the number of sequences is small. However, as the number of sequences increases, QOMA becomes impractical. This paper develops a new algorithm, QOMA2, which optimizes the SP score of the alignment of arbitrarily large number of sequences. Given an initial (potentially sub-optimal) alignment , QOMA2 selects short subsequences from this alignment by placing a window on it. It quickly estimates the amount of improvement that can be obtained by optimizing the alignment of the subsequences in short windows on this alignment. This estimate is called the SW (Sum of Weights) score. It employs a dynamic programming algorithm that selects the set of window positions with the largest total expected improvement. It partitions the subsequences within each window into clusters such that the number of subsequences in each cluster is small enough to be optimally aligned within a given time. Also, it aims to select these clusters so that the optimal alignment of the subsequences in these clusters produces the highest expected SP score. The experimental results show that QOMA2 produces high SP scores quickly even for large number of sequences. They also show that the SW score and the resulting SP score are highly correlated. This implies that it is promising to aim for optimizing the SW score since it is much cheaper than aligning multiple sequences optimally. The software and the benchmark data set are available from the authors on request.**

I. Motivation

Multiple sequence alignment of protein sequences is one of the most fundamental problems in computational biology. It is an alignment of three or more protein sequences. Multiple sequence alignment is widely used in many applications such as protein structure prediction [10], phylogenetic analysis [20], identification of conserved motifs and domains [29], and protein classification [6].

One common measure of the quality of a multiple alignment is its SP (Sum-of-Pairs) score [14], [15], [31], [25]. The SP Score of an alignment, A, of N sequences $P_1, P_2, \cdots, P_N$ is computed by adding the alignment scores of all of its induced pairwise alignments. Table I defines the variables frequently used in the rest of paper. It can be expressed as $SP(A) = \sum_{i<j} Score_{\text{induced}}(P_i, P_j)$, where $Score_{\text{induced}}(P_i, P_j)$ is the score of the alignment of P_i and P_j induced by A. Another measure is the BAliBASE score [29]. Given a gold-standard alignment A^*, this measure evaluates how similar the alignments A and A^* are. The BAliBASE score is commonly used in the literature as an alternative to the SP score, however, BAliBASE score can only be computed for sets of sequences for which the gold standard is known. In contrast, the SP score can be computed for any set of sequences. Most of the existing methods aim to maximize a linear variation of the SP score by weighting the sequences (or subsequences) in order to converge to the BAliBASE score for known benchmark [28], [19]. This paper focuses on optimizing the SP score which is computationally an equivalent problem to the weighted versions in the literature. The problem of finding appropriate weights to converge the SP and the BAliBASE score is orthogonal to this paper and should be considered separately.

TABLE I
The list of variables used in the paper

Variable	Meaning
N	Total number of sequences to be aligned.
W	Window size.
K	Maximum number of sequences of length W that can be optimally aligned.
P_i	Sequence or profile.
f_i	Subsequence of P_i that lies in a given window.
v_i	Vertex corresponding to f_i.
$e_{i,j}$	Weight of the edge between v_i and v_j.
L	Length of a sequence or a profile.
M	Number of windows that are optimized.

The alignment of two sequences that has the maximum score can be found in $O(L^2)$ time using dynamic programming [17], where L is the length of the sequences. This algorithm can be extended to align N sequences optimally, but requires $O(L^N 2^N)$ time [15], [22]. Indeed, finding the multiple sequence alignment that maximizes the SP (Sum-of-Pairs) score is an NP-complete problem [32]. A variety of heuristics have been developed to find a suboptimal alignment. Progressive methods form an important class of the heuristic methods. Such methods progressively align pairs of profiles in a certain order and produce a new profile until a single profile is left. A profile is either a sequence or the alignment of a set of sequences. Figure 1(a) illustrates this. Here, sequences a and b are optimally aligned. Then, c and d are optimally aligned. Their resulting alignments are aligned next. Progressive methods, however, have an important shortcoming. The order that the profiles are chosen for alignment affects the quality of the alignment significantly. The optimal alignment may be different than all possible alignments obtained by considering all possible orderings of sequences [33].

Our previous work [34], QOMA, eliminated the drawbacks of the progressive methods. It partitioned an initial alignment into short subsequences by placing a window. It then optimally realigned the subsequences in each window. This is

shown in Figure 1(b). Optimally aligning each window costs $O(W^N 2^N)$, significantly less than $O(L^N 2^N)$ for $W \ll L$. However, when N is large, even $O(W^N 2^N)$ becomes too costly. The value of W needs to be reduced significantly to make QOMA practical. For example, assume that QOMA works for $W = 32$ when $N = 6$. When N becomes 18, W should be reduced to two in order to run at roughly the same time. This, however, reduces the SP score of the alignments found by QOMA since each window contains extremely short subsequences.

Contributions. This paper addresses the problem of aligning multiple protein sequences with the goal of achieving a large SP score when the number of sequences is large. We develop an algorithm, QOMA2, which works well even when the number of sequences is large. Figure 1(c) illustrates the QOMA2 algorithm. It takes N sequences and a initial (potentially suboptimal) alignment of them as input. QOMA2 selects short subsequences from these sequences by placing a window on their initial alignment. Each window position defines N subsequences, and each subsequence has at most W letters. It quickly estimates the amount of improvement that can be obtained by optimizing the alignment of the subsequences in each window. This estimate is called the *SW (Sum of Weights) score*. It uses a dynamic programming algorithm to select the set of window positions with the largest total expected improvement. It then recursively forms clusters of K, $K \ll N$, subsequences and optimally aligns each cluster. The clusters are created by iteratively partitioning the subsequences into clusters and updating the SW score according to these clusters. Thus, different windows can result different partitioning of subsequences to clusters (see Figure 1(c)). This is desirable since the optimal clustering of the subsequences may differ for different window positions. The value of K is determined by the allowed time budget for QOMA2 for the alignment of the subsequences in clusters governs the overall running time. As K increases both the alignment score and the running time increase. The experimental results show that QOMA2 achieves high SP scores quickly even for large number of sequences. They also show that the SW score and the resulting SP score are highly correlated. This implies that it is promising to aim for optimizing the SW score since it is much cheaper than aligning multiple sequences optimally.

Paper organization. The rest of the paper is organized as follows. Section II discusses related work. Section III discusses the algorithm on how to select the positions of the window. Section IV discusses the algorithm on how to align a window. Section V presents experimental results and analysis. Section VI concludes the paper.

II. Background

Multiple sequence alignment tools. Finding the multiple sequence alignment that maximizes the SP (Sum-of-Pairs) score is an NP-complete problem [32]. MSA [7], DCA [25], COSA [21] and QOMA [34] are among the tools that aim toward the same goal. Most of these tools have the shortcoming that they are unable to process large number of sequences. It is appropriate to apply dynamic programming on subdivisions of alignments. "Jumping alignments" [23] applies a similar idea. QOMA also applies this idea. We will elaborate on

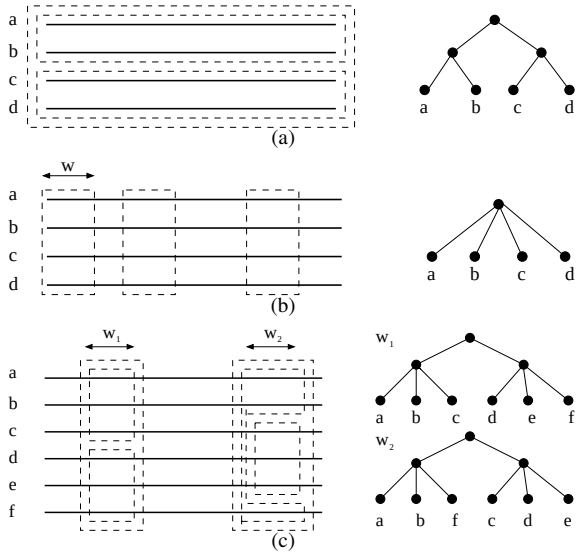

Fig. 1. Alignment strategies at a high level: (a) progressive alignment, (b) the QOMA algorithm (c) the QOMA2 algorithm. The solid lines denote sequences a, b, ..., f. Dashed polygons denote the (sub)sequences whose alignments are optimized. The trees next to alignments show the guide tree used by the underlying algorithm to align the sequences. In (a), a and b are optimally aligned. Then, c and d are optimally aligned. Their resulting alignments are aligned next. In (b), small subsequences of a, b, c, and d in each window is aligned optimally. In (c), the window on the left indicates that the subsequences from a, b and c are optimally aligned, the subsequences from d, e and f are optimally aligned, and then their results are aligned. Similarly, the window on the right indicates that the subsequences from a, b and f are optimally aligned, the subsequences from c, d and e are optimally aligned, and then their results are aligned.

QOMA later in this section. For the current limitation of computer processing capability, heuristic methods are much more popular than optimal methods because of their low time complexity. These heuristic methods can be classified into four clusters: progressive, iterative, anchor-based and probabilistic. Progressive methods [26] include ClustalW [28], [30], T-coffee [19], Treealign [8] and POA [14]. Iterative methods include Muscle [4] and DIALIGN [16]. Anchor-based methods include MAFFT [13], Align-m [31], L-align [9], Mavid, PRRP [5] and DIALIGN [16]. Probabilistic methods include ProbCons [3], Hmmt [24], SAGA [18] and Muscle [4].

There are other type of tools that try to improve on an existing alignment, such as RASCAL [27], REFINER [2] and ReAligner [1]. QOMA2, belongs to this type in general. QOMA2 is different from RASCAL and REFINER because that QOMA2 focuses on optimizing the SP score of alignments and requires only sequence information, while RASCAL is a knowledge-based approach and REFINER targets for optimizing score of core regions. ReAligner uses a round-robin algorithm and improves DNA alignment.

QOMA: A tool for maximizing the SP score. QOMA [34] mainly works in two steps. First, it constructs an initial alignment, by either using an existing tool or by constructing an alignment from pairwise optimal alignments of sequences. Although the time complexity of both strategies are $O(N^2 L^2)$, the latter one is faster. After constructing the initial alignment, the vertices are placed roughly in their correct positions (or in a close by position) in the alignment. Second, the alignment is iteratively improved. At each iteration, a short

window of length W is placed on the existing alignment. The subsequences contained in this window are then replaced by their optimal alignments. Although QOMA works well for small number of sequences ($N \leq 7$), it becomes impractical for large N. This is because optimizing a single window costs $O(W^N 2^N)$, which is still too much even for small W. For example, when $N = 20$, and $W = 4$, a single window requires one tera-operations. This will take over 12 days even with a computer that performs one million of those operations per second. Efficient algorithms that balance the time and score are needed.

Graph Partitioning. METIS [12], [11] is a popular tool for partitioning unstructured graphs, partitioning meshes, and computing fill-reduced ordering of sparse matrices. The algorithms implemented in METIS are based on the multilevel recursive-bisection, multilevel k-way, and multi-constraint partitioning schemes. It can provide high quality partitions fast.

III. Selecting window positions

Let A be an alignment of N sequences $P_1, P_2, \cdots, P_N$. Let $W > 1$ be an integer that denotes the window length. Assume that we are allowed to place a window on A in M different locations and optimize the alignment of the subsequences in these M locations. The first problem that needs to be addressed is the identification of the M locations that maximize the overall improvement. Figures 1(b) and 1(c) show two examples in which three and two positions are selected respectively. It is important to mention that the number of windows, M, is governed by the total time allowed for improving the alignment.

A simple way to select the positions to place the window is to slide a window from the left to the right (or from the right to the left), shifting by some predefined amount Δ at each iteration. Another simple solution is to select the window positions randomly. Clearly, both of these solutions do not distinguish promising window positions from unpromising ones. We suggested a greedy solution in our QOMA paper. This algorithm greedily selects the most promising window position from the unselected positions until M positions are selected. We discuss how we quantify how promising a window is later in this section. This greedy strategy, however, does not guarantee to find the best set of M window positions. Here, we develop a dynamic programming algorithm that guarantees to find the M optimal window positions.

For each window position, we compute an upper bound to the improvement of the SP score that could be achieved by replacing that window with its optimal alignment as follows. Let X_i denote the upper bound to the optimal SP score for the subsequences in the window starting at position i of the alignment. This number can be computed as the sum of the scores of all pairwise optimal alignments of the subsequences in this window. Let Y_i denote the current SP score of that window. The upper bound to the improvement of the SP score is computed as $U_i = X_i - Y_i$. We say that a window position i is promising if U_i is large.

We propose to select the M window positions, $\pi_1, \pi_2, \cdots, \pi_M$ ($\forall i, \pi_i < \pi_{i+1}$) whose sum of upper bounds (i.e., $\sum_i U_{\pi_i}$) is the largest. Note that, if two windows overlap greatly, their combined improvement over the initial alignment can be much less than their individual improvements. This is because they improve almost the same regions, and thus, they are highly dependent. The sum of their upper bounds includes the upper bound for their common region twice. In order to prevent this, we also enforce a minimum distance between the positions of different windows as $\forall i$, $\pi_{i+1} - \pi_i \geq \tau$. Thus, if a window is positioned at π_i, no other window can be placed on a position in the $[\pi_i - \tau, \pi_i + \tau]$ interval.

The value of τ determines how independent the windows are. As τ increases, windows become more independent. For $\tau \geq W$, the windows are completely non-overlapping. On the other hand, large values of τ limit the number of possible window positions. We use $\tau = W/4$ as it provided a good balance in our experiments.

We develop a dynamic programming solution to determine the optimal window positions. Let SU(a, b) denote the largest possible sum of upper bounds of b window positions selected from the first a possible window positions. We would like to determine SU($L - W + 1$, M) to solve our problem, where L is the length of the alignment. Clearly, SU(a, 1) $= \max_{i=1}^{a}\{U_i\}$.This is because if a single window is selected it should be the one with the largest upper bound. For $b > 1$, there are two possibilities: 1) If $a < b\tau$, SU(a, b) $= 0$. This is because, from Dirichlet principle, it is impossible to select b window positions that overlap with less than τ positions in this case. 2) If $a \geq b\tau$, we compute SU(a, b) recursively as

$$SU(a,b) = \max \begin{cases} SU(a-\tau, b-1) + U_a, & \text{if } U_a \text{ is selected} \\ SU(a-1, b), & \text{otherwise} \end{cases}$$

In this computation, the first condition implies that the bth window starts at position a. Thus, the first $b - 1$ windows should be selected in the interval $[1, a - \tau]$ to ensure that they do not overlap with the bth window by more than τ. The second condition implies that the window at position a is not a part of the solution. Therefore, the b window positions should be selected in the interval $[1, a-1]$. The value of SU($L-W+1$, M) is the optimal sum of upper bounds. The window positions that lead to this optimal solution can be found by tracking back the values of SU after the dynamic programming computation completes.

Figure 2 shows the average SP score of the improved alignment for the first eleven window positions when the windows are selected using our dynamic programming method, greedily, and by sliding a window. For the window sliding strategy, we shift the window by $W/2$ at each iteration. The results are obtained by averaging the results of 82 BAliBASE benchmarks. We use $W = 8$ and $N = K = 4$ (i.e., each window of length eight is optimally aligned). The figure shows that the proposed selection strategy improves the SP score much faster than the sliding and the greedy strategies.

IV. Aligning a window

The goal of aligning a window is to maximize the SP score of the subsequences within each window. We propose a divide-and-conquer strategy, which clusters the set of N subsequences into smaller sets of K subsequences so that the subsequences in each subset can be optimally aligned. This method has two major differences from the progressive methods. First, progressive methods align two sequences (or

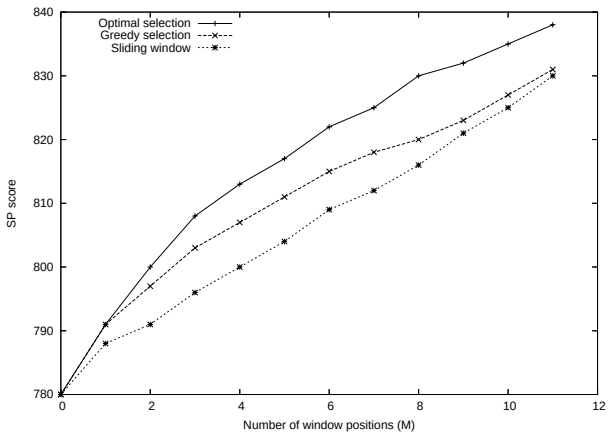

Fig. 2. Comparison of the SP score found by different strategies of selection of window positions: using the proposed optimal selection, the greedy selection and the sliding window.

profiles) at a time. Thus $K = 2$ for the progressive methods, whereas QOMA2 can use larger K values since it focuses on a short window. Second, once the clusters are determined, progressive methods align the entire sequences based on that clustering. However, QOMA2 can find different ways of clustering the data for different window positions (see Figure 1(c) as an example). This is desirable for different regions in sequences may evolve at different conservation rates. For example, regions that serve important functions show much less variation then the remaining regions. Therefore, the best clustering for one region of the sequences may not be good for another region. QOMA2 addresses this by treating each region independently.

We first construct an initial weighted complete graph by considering each subsequence in the window as a vertex. We then align the subsequences using two nested loops. The details of the two steps are discussed next.

A. Constructing initial graph

Given a window on the alignment, we first construct a weighted, undirected, complete graph $G = (V, E)$. This graph models how much the SP score can be improved by realigning the subsequences in this window carefully. Let f_i denote the subsequence of the sequence P_i that remains in the window, $\forall i,\ 1 \leq i \leq N$.

Each f_i maps to a vertex $v_i \in V$ in this graph. We compute the weight of the edge $e_{i,j} \in E$ between vertices v_i and v_j as

$$e_{i,j} = Score_{\text{optimal}}(f_i, f_j) - Score_{\text{induced}}(f_i, f_j) \qquad (1)$$

$Score_{\text{optimal}}(f_i, f_j)$ computes the score of the optimal alignment of f_i and f_j. $Score_{\text{induced}}(f_i, f_j)$ denotes the score of the alignment of f_i and f_j induced from the current alignment. In other words, $e_{i,j}$ is an upper bound to the improvement of the SP score due to f_i and f_j after realigning the window.

Definition 1: Let $G = (V, E)$ be the graph constructed for a set of subsequences in a window. We define the sum of the weights of all the edges in E as the *SW (Sum of Weights) score* of G. ■

The SW score is an upper bound to how much the SP score of the subsequences in the underlying window can improve by aligning those subsequences optimally when the edge weights are computed as given in equation (1).

The vertex induced subgraph of any subset $V' \subset V$ defines a complete subgraph $G' = (V', E')$. The SW score of G' is an upper bound to the amount of improvement that can be obtained by optimally aligning only the subsequences that map to the vertices in V'. In the following sections we will exploit the SW score to find a good clustering of the subsequences in a given window.

B. Clustering

The clustering algorithm partitions the set subsequences $\{f_1, f_2, \cdots, f_N\}$ into non-overlapping subsets of size at most K. The eventual goal is that optimally aligning each subset followed by aligning the results of these alignments improves the SP score as much as possible. Recall that each subset can not have more than K subsequences since we can not optimally align more than K subsequences within the allowed time.

We first need to understand how many clusters need to be created. The number of subsequences in each partition should be as large as possible. This is because more subsequences are optimally aligned with each other when the clusters are large. This indicates that there must be $\lceil \frac{N}{K} \rceil$ clusters.

Next, we need to understand the right criteria to partition the set of subsequences. A number of strategies can be developed to address this question. We discuss two solutions with the help of the complete weighted graph G constructed for the subsequences. Notice that partitioning the set of subsequences into clusters of subsequences is equivalent to partitioning the graph G into vertex induced subgraphs of the vertices corresponding to the subsequences in each cluster.

Min-cut clustering. The first strategy aims to optimize the intra-cluster SP score. That is, it maximizes the improvement in the SP score by optimally aligning the subsequences within each cluster. At a high level, this is done by partitioning G into $\lceil \frac{N}{K} \rceil$ subgraphs such that the sum of the SW scores of these subgraphs is as large as possible. This is equivalent to the *Min* $\lceil \frac{N}{K} \rceil$*-Cut* problem with the additional restriction that each subgraph has at most K vertices. In other words, it translates into the problem of finding the set of edges in G such that

- their removal partitions G into $\lceil \frac{N}{K} \rceil$ complete subgraphs of size at most K, and
- the sum of their weights are as small as possible.

Finding the Min $\lceil \frac{N}{K} \rceil$-Cut of a graph is an NP-complete problem. A number of heuristic algorithms have been developed to address this issue. One of the most commonly used tools for partitioning graphs is METIS [12], [11]. METIS partitions an input graph to a given number of subgraphs with the aim of minimizing or maximizing the total weight of the edges between different subgraphs. We use METIS to partition G into $\lceil \frac{N}{K} \rceil$ subgraphs with minimal $\lceil \frac{N}{K} \rceil$-cut.

Although, METIS tries to partition the graph into roughly the same sized subgraphs, it does not guarantee that they will be perfectly balanced in size. As a result, some of the clusters determined by METIS can have more than K vertices.

This is undesirable since the subsequences in each cluster are optimally aligned in the following step. Recall that the cost of optimally aligning a cluster is exponential in the size of that cluster. The maximum size of a cluster, K, is determined by the total amount of time allowed to spend to optimize the alignment. Thus, METIS clusters need to be post-processed to guarantee that the sizes of the clusters do not exceed K.

Next, we describe how we propose to adjust the size of the METIS clusters for the first strategy (i.e., optimizing the intra-cluster SP score) first. It is trivial to adapt this algorithm to the second strategy.

Given a set of subgraphs (i.e., clusters) identified by METIS, we create three sets. The first one is the set of subgraphs with K vertices, named EK (Equal to K). The second one is the set of subgraphs with more than K vertices, named GK (Greater than K). The last one is the set of clusters with less than K vertices, named LK (Less than K). We adjust the size of the clusters by moving vertices from clusters in GK to clusters in LK. Out of all such moves, it greedily picks the one which causes the smallest cut since the goal is to minimize the total weight of the inter-cluster edges. After each move, the number of vertices in one of the clusters in GK decreases by one. Similarly, the number of vertices in one of the clusters in LK increases by one. Thus, the clusters in GK and LK move to EK. The iterations stop when GK is empty. This algorithm is guaranteed to converge to a solution in $\sum_{G' \in GK}(|G'|-K)$ iterations of the while loop, where $|G'|$ denotes the number of vertices in G'. This is because, the number of vertices in a $G' \in GK$ reduces by one at each iteration.

Max-Cut clustering. The second strategy aims to optimize the inter-cluster SP. It achieves this by maximizing the total weight of the edges in the $\lceil \frac{N}{K} \rceil$-cut of G. Similar to the first strategy, we use METIS to identify such a cut.

The proposed algorithm for post-processing the clusters found by METIS can be adapted to the second strategy as follows. At each iteration of the while loop, the vertex move that maximizes the cut is chosen instead of the one that minimizes. This can be done by modifying Steps 1 and 2.c of the algorithm.

It is worth mentioning that the METIS algorithm for clustering the sequences is a module in QOMA2. It can be replaced by any clustering algorithm that finds better Min $\lceil \frac{N}{K} \rceil$-Cut or Max $\lceil \frac{N}{K} \rceil$-Cut in the future.

C. Refining clusters iteratively

The Min-Cut and the Max-Cut clustering strategies aim to minimize or maximize the cut (see Section IV-B). One drawback of these strategies is that each edge weight is computed by only considering the two subsequences corresponding to the two ends of that edge (see Section IV-A). This is problematic, because the amount of improvement in the SP score by optimally aligning a cluster of subsequences depends on all the subsequences in that cluster. Considering two subsequences at a time greatly overestimates the improvement. We propose to improve the clusters iteratively. Each iteration updates the edge weights by considering all the subsequences in each cluster. We discuss how the edge weights are updated later in this section. Once the edge weights are updated, it reclusters the subsequences using the new weights. The iterations stop when the SW score of the graph G does not increase between two consecutive iterations or a certain number of iterations have been performed.

We would like to estimate how much the two subsequences, f_i and f_j, contribute to the SP score under the restriction that each cluster is optimally aligned. The obvious solution is to optimally align each cluster and measure the new alignment score. This, however, is not practical for two reasons. First, optimally aligning a cluster of K subsequences is a costly operation. Performing this operation will make each iteration of the cluster refinement as costly as QOMA2. Furthermore, this will only update the weight of the edges whose two ends belong to the same subgraph (i.e., intra-cluster edges). The weight of the edges between different subgraphs (i.e., inter-cluster edges) still need to be computed. Thus, a good estimator should be efficient and work for both inter- and intra-cluster edges.

We propose to estimate the edge weights by focusing on the gaps. At a high level, we assume the best scenario (i.e., smallest possible number of gaps) for intra-cluster edges. This is because of the restriction that the subsequences in each cluster are optimally aligned. We then estimate the improvement in the SP score between every pair of subsequences by considering these gaps. We describe our estimator in detail next.

Let L_i be the length of subsequence f_i. After the complete weighted graph G is partitioned into $\lceil \frac{N}{K} \rceil$ complete subgraphs, assume that v_i belongs to the subgraph G'. Recall that v_i is the vertex that denotes f_i. The optimal alignment of all the subsequences in the same cluster as f_i requires insertion of at least

$$g_i = \max_{v_j \in G'} \{L_j\} - L_i$$

letters into f_i. This is because the alignment of all the subsequences in a cluster can not be shorter than the longest subsequence in that cluster. Each such insertion corresponds to a gap in the alignment. Thus, g_i denotes the minimum number of gaps imposed on f_i due to clustering of the subsequences.

Next, we compute the expected number of indels (insertions or deletions) in the alignment of subsequences f_i and f_j. An indel is an alignment of a letter with a gap. The alignment of two letters or two gaps are not considered as indels. Considering all possible arrangement of the letters and gaps in f_i and f_j, the expected ratio of letter-letter alignments between f_i and f_j in their alignments is

$$\frac{L_i L_j}{(L_i + g_i)(L_j + g_j)} \tag{2}$$

Similarly, the expected ratio of gap-gap alignments is

$$\frac{g_i g_j}{(L_i + g_i)(L_j + g_j)} \tag{3}$$

Thus, the expected ratio of indels can be computed by subtracting equations (2) and (3) from one. The total length of the induced alignment of f_i and f_j is at most $\max\{L_i+g_i, L_j+g_j\}$. Therefore, the expected number of indels in the induced alignment of f_i and f_j, denoted by $Gap_{\text{expected}}(f_i, f_j)$ is at most

$$\left(1 - \frac{L_i L_j + g_i g_j}{(L_i + g_i)(L_j + g_j)}\right) \max\{L_i + g_i, L_j + g_j\} \tag{4}$$

Let $Gap_{\text{induced}}(f_i, f_j)$ denote the number of indels in the induced alignment of f_i and f_j. Let $gapcost$ denote the cost of a single indel. We compute the new weight of the edge between vertices v_i and v_j as

$$e_{i,j} = Score_{\text{optimal}}(f_i, f_j) - Score_{\text{induced}}(f_i, f_j) - gapcost \times (Gap_{\text{expected}}(f_i, f_j) - Gap_{\text{induced}}(f_i, f_j)).$$

This computation differs from the one in Section IV-A since it considers the change in the gap cost as imposed by the clusters that f_i and f_j belong to.

Once the weights of the edges are updated, the current partitioning may not be a good one anymore. Therefore, we iteratively run the clustering algorithm again and update the edge weights similarly until the SW score of the complete graph built for the current window does not increase any further or a given maximum number for iterations are reached.

D. Aligning the subsequences in clusters

The clustering algorithm guarantees that each cluster has at most K subsequences. However, the total number of clusters may be greater than K. This happens when $N > K^2$. In that case, finding the optimal alignment of the profiles of clusters becomes infeasible. Although this brings us back to the same problem we are tackling in this paper, it is easier since we have $\lceil \frac{N}{K} \rceil$ profiles which is significantly less than N. We recursively apply the QOMA2 algorithm (Sections IV-A to IV-C) to these profiles until all the subsequences are aligned.

E. Complexity of QOMA2

The time complexity of QOMA2 is $O(M \log_K N(\frac{NW^K 2^K}{K} + cN^2))$, where c is the upper bound for the number of inner loop iterations. n practice $c \leq 10$.

We deduct the time complexity as follows: For each window, we need to apply the clustering algorithm and align the clusters using two nested loops. The outer loop iterates $\lceil log_K N \rceil$ times.

At each iteration the set of subsequences inside the window is partitioned into clusters and the edge weights are updated. Thus, each iteration of the inner loop costs $O(|G|)$ time. Since G contains N vertices $O(|G|) = O(N^2)$. At the end of each iteration of the inner loop all the clusters are optimally aligned. Optimally Aligning K subsequences costs $O(W^K 2^K)$ time. At the ith iteration of the outer loop, $O(\frac{N}{K^i})$ such optimal alignments are done. Adding these steps, we find that the total cost of the ith iteration of the outer loop is

$$O(\frac{N}{K^i} W^K 2^K + cN^2).$$

The number of outer loop iterations is $\log_K N$. Thus, the total cost of aligning a window is

$$\begin{aligned} & \textstyle\sum_{i=1}^{\lceil \log_K N \rceil} O(\frac{N}{K^i} W^K 2^K + cN^2) \\ = \; & O((\log_K N)(NW^K 2^K (\textstyle\sum_{i=1}^{\lceil \log_K N \rceil} \frac{1}{K^i}) + cN^2)) \\ = \; & O((\log_K N)(NW^K 2^K (\tfrac{N(K-1)}{(N-1)K^2}) + cN^2)) \\ = \; & O((\log_K N)(\tfrac{NW^K 2^K}{K} + cN^2)) \end{aligned}$$ ■

Since we totally have M positions for window to align, the total cost of QOMA2 is $O(M \log_K N(\frac{NW^K 2^K}{K} + cN^2))$.

V. Experimental evaluation

Experimental setup: We used BAliBASE benchmarks [29] reference 1 from version 1 (www-igbmc.u-strasbg.fr/BioInfo/BAliBASE/) and references 1, 2, $\cdots$, 8 from version 3 (www-bio3d-igbmc.u-strasbg.fr/BAliBASE/) for evaluation of our method. We call this dataset *D3* since it contains benchmarks with three or more sequences. We call the subset of D3 that contains all the benchmarks with at least 10 sequences as *D10*. Similarly, we call the subset of D3 that contains all the benchmarks with at least 20 sequences as *D20*. D3, D10, and D20 contain 440, 209, and 84 benchmarks respectively.

We implemented the QOMA2 algorithm using standard C. We downloaded ProbCons [3], T-Coffee [19], Muscle [4], and ClustalW [28], [30] for comparison. We also downloaded DCA [25] since it aims to maximize the SP score as well. However, DCA did not run for the benchmarks in our datasets D10 and D20 since it can not align large number of sequences. We used BLOSUM62 as a measure of similarity between amino acids, since BLOSUM62 is commonly used. Using other popular score matrices, such as BLOSUM90 or PAM250 will produce similar results. We used gap cost = -4 to penalize each indel. In order to be fair, we used the same parameters (i.e., BLOSUM62 and gap cost) for QOMA2, T-Coffee, Muscle, and ClustalW. We used the default parameters for ProbCons for it was impossible to change those parameters for ProbCons.

Among the competing tools, used in our experiments, Muscle aims to maximize the SP score, ClustalW and T-Coffee aims to maximize a weighted version of the SP score. Therefore, one can argue that it is not fair to include ClustalW, T-Coffee and ProbCons in our experiments. We, however, include them since most of the existing tools that aim to maximize the SP score, such as DCA or MSA, do not work for large number of sequences. We improve the fairness of our experiments by using the same parameters for all the tools.

First, we compared different clustering algorithms and showed the relationship between the SP and the SW scores on each window. We then evaluated the impact of the window and the cluster size on the SP score of the QOMA2 alignment and the running time of QOMA2. We also compared the SP scores of QOMA2 with four competing multiple sequence alignment tools. We ran our experiments on a system with dual 2.59 GHz AMD Opteron Processors, 8 gigabytes of RAM, and a Linux operating system.

Correlation between the SP and the SW scores: The main hypothesis that QOMA2 depends on is that optimizing the SW score optimizes the SP score. Thus it aims to optimize the SW score by finding an appropriate clustering of the sequences. For a given window, the SW score is computed in $O(N^2)$ time as it requires estimating the gap cost for each pair of subsequences. The SP score, on the other hand, requires aligning the subsequences. Therefore, it costs $O(\log_K N(\frac{NW^K 2^K}{K} + cN^2))$ time. This makes QOMA2 desirable since the SW score can be measured efficiently without actually finding the alignment of multiple sequences.

TABLE II

THE AVERAGE SW AND SP SCORES OF INDIVIDUAL WINDOWS AFTER APPLYING DIFFERENT CLUSTERING ALGORITHMS FOR DIFFERENT VALUES OF K, WITH W = 16. THE AVERAGE SP SCORES OF INITIAL ALIGNMENT IN THE WINDOW IS 351. THE AVERAGE UPPER BOUND TO THE SP SCORE FOR THE SUBSEQUENCES IN THE WINDOWS IS 1113. BENCHMARKS ARE SELECTED FROM THE D20 DATASET.

K	Min-Cut		Min-Cut Iterative		Max-Cut		Max-Cut Iterative	
	SP	SW	SP	SW	SP	SW	SP	SW
2	-19	1285	157	1315	284	1482	481	1544
3	133	974	197	1031	490	1207	494	1268
4	200	823	266	908	485	1005	499	1104

In this experiment, we evaluate the relationship between the SW and the SP scores. We also measure how each of the proposed clustering strategies performs. We place a window (W = 16) on all possible locations of an initial alignment. We find the clusters using the Min-Cut and the Max-Cut clustering algorithms (see Section IV-B). We also find clusters using the iterative refinement (see Section IV-C) on the results of Min-Cut and Max-Cut. We measure the average SP and SW scores obtained by these algorithms for $K = 2$, 3, and 4. We use D20 dataset in this experiment.

Table II presents the results. Results show that there is a strong correlation between the SP and the SW scores. For each value of K, the SP score gets larger when the SW score gets larger. This implies that optimizing the SW score can potentially optimize the SP score. This is an important observation since the cost of computing the SW score is negligible as compared to that of the SP score. Note that the SW scores obtained with different number of clusters are not comparable to each other since they compute the gap cost under different cluster size assumptions. The results also demonstrate that the iterative refinement helps in improving the SW and the SP score of both of the Max-Cut and the Min-Cut algorithms. The Max-Cut algorithm with iterative refinement always has the best SP and SW scores. This implies that if the induced alignment of two subsequences has a high score as compared to that of their optimal alignment, it is advantageous to keep them in the same cluster (i.e., force them to be almost optimally aligned).

The SP score of all the methods increase as the value of K increases. This is intuitive since more subsequences are optimally aligned at once for large values of K.

Another important observation that follows from these results is that optimally aligning clusters does not always improve the SP score of a window. It can actually reduce it. This happens especially for the Min-Cut clustering (with or without iterative refinement) for all values of K as well as the Max-Cut clustering for $K = 2$. This is because when the clusters of subsequences are aligned, they impose a certain alignment for the subsequences in each cluster. These restrictions limit the number of possibilities in which a set of clusters can be aligned together. This indicates that the clusters should be selected carefully.

In the rest of the experiments, we select the Max-Cut clustering algorithm with iterative refinement as the default clustering algorithm of QOMA2.

Impact of W and K on the SP score. The QOMA2 algorithm hypothesizes that the SP score can be optimized by increasing the value of W and K. In this experiment, we evaluate the impact of these parameters on the SP score of QOMA2.

TABLE III

THE AVERAGE SP SCORES OF QOMA2 FOR INDIVIDUAL WINDOWS. "SP BEFORE" AND "UPPER BOUND" DENOTE THE AVERAGE INITIAL SP SCORES AND THE AVERAGE UPPER BOUNDS TO THE SP SCORES FOR INDIVIDUAL WINDOWS RESPECTIVELY. BENCHMARKS ARE SELECTED FROM THE D10 DATASET.

W	SP before	Upper bound	K = 2	K = 3	K = 4	K = 5
4	-186	-67	-171	-158	-152	-147
8	-212	100	-175	-140	-124	-111
12	-264	247	-203	-147	-120	-100
16	-342	358	-257	-183	-148	-117

TABLE IV

THE AVERAGE SP SCORES OF THE ALIGNMENTS OF THE ENTIRE BENCHMARKS IN D10 USING QOMA2. THE AVERAGE SP SCORES OF INITIAL ALIGNMENTS IS -12295. THE AVERAGE OF THE UPPER BOUND TO THE SP SCORES OF THE BENCHMARKS IS 17648. THE AVERAGE RUNNING TIMES ARE ALSO SHOWN IN THE PARENTHESES BY SECONDS.

W	K = 2	K = 3	K = 4	K = 5
4	-7119(1.173)	-6770(0.653)	-6676(0.403)	-6498(0.465)
8	-6197(1.213)	-5348(0.673)	-4762(1.053)	-4236(5.050)
12	-5914(1.116)	-4659(0.808)	-3966(3.619)	-3464(13.485)
16	-5690(1.097)	-4327(1.102)	-3555(8.856)	-2811(40.132)

Table III shows the SP score of individual windows aligned by QOMA2 for different values of W and K. The results show that the SP scores increase when K increases for all values of W.

Table IV shows the SP scores of alignments of the entire benchmarks in D10 using QOMA2 for varying values of W and K. As W and K increase, QOMA2 produces higher scores. The two extreme parameter choices of using very large value for one of these parameters and very small value for the other, i.e., $W = 16$, $K = 2$ or $W = 4$, $K = 5$ do not produce lower SP scores as compared to the intermediate solutions such as $W = 12$, $K = 3$. This is an important observation since it validates that QOMA2 is superior to the two existing extreme solutions (see Figure 1).

Impact of W and K on the running time Table IV shows the average running time of QOMA2 for optimizing a single window for varying values of W and K. The experimental results show that QOMA2 runs very efficiently even for large number of sequences. As we have mentioned in Section IV-E, the time complexity of QOMA2 is $O((\log_K N)(\frac{NW^K 2^K}{K} + cN^2))$ for a single window. The experimental results suggest when W is large, the factor $O(\log_K N \frac{NW^K 2^K}{K})$ quickly dominates the running time.

From Tables III and IV, we conclude a good point for balancing time and quality is at (W = 12, K= 4).

Comparison to existing tools. Table V presents the SP scores of the alignments of the benchmarks in D10 using four existing tools and QOMA2. Note that the compared tools do not aim to maximize the SP score. ClustalW, Muscle, and T-coffee optimize a variation of the SP score by computing weights for sequences or subsequences. We still included this experiment because the existing tools that optimize the SP

TABLE V
THE AVERAGE SP SCORES OF QOMA2 ($W = 12$ AND $K = 4$) AND FOUR OTHER TOOLS ON THE D10 DATASET. THE COMPETING TOOLS (EXCEPT PROBCONS) ARE RUN WITH THE SAME PARAMETERS AS QOMA2. THE BEST RESULT FOR EACH DATASET AND PARAMETER SETTING IS SHOWN IN BOLD.

N	ProbCons	T-coffee	Muscle	ClustalW	QOMA2
10-14	-16921	-16713	-24492	-12586	**-12318**
15-19	-14454	-29751	-31851	-9426	**-9088**
20-24	-5958	-12006	-28866	-778	**-710**
25-29	-24033	-29305	-50576	-9628	**-8989**

score, such as DCA [25], MSA [7] and COSA [21] do not work for large number of sequences. For small number of sequences, QOMA performs significantly better than DCA (see [34]). We divided the queries into four subsets according to the number of sequences they contain. The table shows that QOMA2 has higher SP score than all the tools compared. ClustalW is always the second best. The remaining tools are not competitive in terms of the SP score.

VI. CONCLUSION

We considered the problem of multiple alignment for a large number of protein sequences, with the goal of achieving a large SP (Sum-of-Pairs) score. We introduced the QOMA2 algorithm, which is practical for aligning a large number of protein sequences. QOMA2 selects short subsequences from the sequences to be aligned by placing a window on their (potentially sub-optimal) alignment. The window position is determined as the subsequences that have the highest improvement potential. It partitions the subsequences within each window into clusters such that the number of subsequences in each cluster is small enough to be optimally aligned within a given time. The experimental results on BAliBASE benchmarks show that QOMA2 produces alignments with high SP scores quickly.

REFERENCES

[1] E. L. Anson and E. W. Myers. ReAligner: A program for refining DNA sequence multi-alignments. In *Proceedings of the1st Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 9–16, Santa Fe, NM, 1997. ACM Press.

[2] S. Chakrabarti, C.J. Lanczycki, A.R. Panchenko, T.M. Przytycka, P.A. Thiessen, and S.H. Bryant. Refining multiple sequence alignments with conserved core regions. *Nucleic Acids Res*, 34(9):2598–606, 2006.

[3] C. Do, M. Brudno, and S. Batzoglou. PROBCONS: Probabilistic Consistency-based Multiple Alignment of Amino Acid Sequences . In *Intelligent Systems for Molecular Biology (ISMB)*, 2004.

[4] R.C. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

[5] O. Gotoh. Significant Improvement in Accuracy of Multiple Protein Sequence Alignments by Iterative Refinement as Assessed by Reference to Structural Alignments. *Journal of Molecular Biology*, 264(4):823–838, 1996.

[6] William Noble Grundy. Family-based Homology Detection via Pairwise Sequence Comparison. In *Annual Conference on Research in Computational Molecular Biology (RECOMB '98)*, pages 94–100, 1997.

[7] S. Gupta, J. Kececioglu, and A. Schaffer. Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment, 1996.

[8] J Hein. A new method that simultaneously aligns and reconstructs ancestral sequences forany number of homologous sequences, when the phylogeny is given. *Molecular Biology and Evolution*, 6(6):649–668, 1989.

[9] X. Huang and W. Miller. A time-efficient, linear-space local similarity algorithm. *Advances in Applied Mathematics*, 12:337–357, 1991.

[10] D. T. Jones. Protein Secondary Structure Prediction based on Position-Specific Scoring Matrices. *Journal of Molecular Biology*, 292(2):195–202, 1999.

[11] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report TR 95-035, 1995.

[12] George Karypis and Vipin Kumar. *MeTis: Unstrctured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0*, 1995.

[13] 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*, 30(14):3059–3066, 2002.

[14] C. Lee, C. Grasso, and M.F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.

[15] D.J. Lipman, S.F. Altschul, and J.D. Kececioglu. A Tool for Multiple Sequence Alignment. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 86(12):4412–4415, 1989.

[16] B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: Finding Local Similarities by Multiple Sequence Alignment. *Bioinformatics*, 14(3):290–294, 1998.

[17] S. B. Needleman and C. D. Wunsch. A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins. *Journal of Molecular Biology*, 48:443–53, 1970.

[18] C Notredame and DG Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24(8):1515–1524, 1996.

[19] C. Notredame, D.G. Higgins, and J. Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217, 2000.

[20] A. Phillips, D. Janies, and W. Wheeler. Multiple Sequence Alignment in Phylogenetic Analysis. *Molecular Phylogenetics and Evolution*, 16(3):317–330, 2000.

[21] K. Reinert, H.-P. Lenhof, P. Mutzel, K. Mehlhorn, and J. D. Kececioglu. A branch-and-cut algorithm for multiple sequence alignment. In *Proceedings of the1st Annual International Conference on Computational Molecular Biology (RECOMB)*, pages 241–250, Santa Fe, NM, 1997. ACM Press.

[22] Gupta SK, Kececioglu JD, and Schaffer AA. Improving the Practical Space and Time Efficiency of the Shortest-paths Approach to Sum-of-pairs Multiple Sequence Alignment. *Journal of Computational Biology*, 2(3):459, 1995.

[23] Rainer Spang, Marc Rehmsmeier, and Jens Stoye. Sequence database search using jumping alignments. pages 367–375.

[24] Eddy SR. Multiple Alignment Using Hidden Markov Models. In *Intelligent Systems for Molecular Biology (ISMB)*, volume 3, pages 114–120, 1995.

[25] Jens Stoye, Vincent Moulton, and Andreas W. M. Dress. Dca: an efficient implementation of the divide-and-conquer approach to simultaneous multiple sequence alignment. *Computer Applications in the Biosciences*, 13(6):625–626, 1997.

[26] S.-H. Sze, Y. Lu, and Q. Yang. A polynomial time solvable formulation of multiple sequence alignment. In *International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 204–216, 2005.

[27] J. D. Thompson, J. C. Thierry, and O. Poch. RASCAL: rapid scanning and correction of multiple sequence alignments. *Bioinformatics*, 19(9):1155–1161, 2003.

[28] J.D. Thompson, D.G. Higgins, and T.J. Gibson. CLUSTAL W: Improving the Sensitivity of Progressive Multiple Sequence Alignment through Sequence Weighting, Position-specific Gap Penalties and Weight Matrix Choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[29] J.D. Thompson, H. Plewniak, and O. Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[30] Rene Thomsen, Gary B. Fogel, and Thiemo Krink. Improvement of Clustal-Derived Sequence Alignments with Evolutionary Algorithms. In *Congress on Evolutionary Computation*, volume 1, pages 312–319, 2003.

[31] I.V. Walle, I. Lasters, and L. Wyns. Align-m–a new algorithm for multiple alignment of highly divergent sequences. *Bioinformatics*, 20(9):1428–1435, 2004.

[32] L Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.

[33] Xu Zhang and Tamer Kahveci. A New Approach for Alignment of Multiple Proteins. In *Pacific Symposium on Biocomputing (PSB)*, pages 339–350, 2006.

[34] Xu Zhang and Tamer Kahveci. QOMA: quasi-optimal multiple alignment of protein sequences. *Bioinformatics*, 23(2):162–168, 2007.