Faster Algorithms for Optimal Multiple Sequence Alignment Based on Pairwise Comparisons *

Pankaj K. Agarwal[1], Yonatan Bilu[2], and Rachel Kolodny[3]

[1] Department of Computer Science, Box 90129,
Duke University, Durham NC 27708-0129, USA
pankaj@cs.duke.edu
[2] Department of Molecular Genetics,
Weizmann Institute, 76100 Rehovot,Israel
johnblue@cs.huji.ac.il
[3] Department of Biochemistry and Molecular Biophysics, Columbia University
rachel.kolodny@columbia.edu

Abstract. Multiple Sequence Alignment (MSA) is one of the most fundamental problems in computational molecular biology. The running time of the best known scheme for finding an optimal alignment, based on dynamic programming, increases exponentially with the number of input sequences. Hence, many heuristics were suggested for the problem. We consider the following version of the MSA problem: In a preprocessing stage pairwise alignments are found for every pair of sequences. The goal is to find an optimal alignment in which matches are restricted to positions that were matched at the preprocessing stage. We present several techniques for making the dynamic programming algorithm more efficient, while still finding an *optimal* solution under these restrictions. Namely, in our formulation the MSA must conform with pairwise (local) alignments, and in return can be solved more efficiently. We prove that it suffices to find an optimal alignment of sequence segments, rather than single letters, thereby reducing the input size and thus improving the running time.

1 Introduction

Multiple Sequence Alignment (MSA) is one of the central problems in computational molecular biology — it identifies and quantifies similarities among several protein or DNA sequences. Typically, MSA helps in detecting highly conserved motifs and remote homologues. Among its many uses, MSA offers evolutionary insight, allows transfer of annotations, and assists in representing protein families [20].

Dynamic programming (DP) algorithms compute an optimal Multiple Sequence Alignment for a wide range of scoring functions. In 1970, Needleman and Wunsch [19] proposed a DP algorithm for pairwise alignment, which was later improved by Masek and Paterson [13]. Murata *et al.* [17] extended this algorithm to aligning k sequences (each of length n). Their solution constructs a k-dimensional grid of size $O(n^k)$, with

* Part of this work was done while the second author was at the School of Engineering and Computer Science, The Hebrew University of Jerusalem; the third author was at the Department of Computer Science, Stanford University and visiting Duke University.

R. Casadio and G. Myers (Eds.): WABI 2005, LNBI 3692, pp. 315–327, 2005.

each of the sequences enumerating one of its dimensions. The optimal MSA is an optimal path from the furthermost corner (the end of all sequences) to the origin (their beginning). Unfortunately, the $O(n^k)$ running time of this approach makes it prohibitive even for modest values of n and k. There is little hope for improving the worst-case efficiency of algorithms that solve this problem, since the MSA problem is known to be NP-Hard for certain natural scoring functions [10,3]. This is shown by reduction from Max-Cut and Vertex Cover [8], that is, instances of these problems are encoded as a set of sequences. However, the encoding sequences are not representative of protein and DNA sequences abundant in nature, and the alignments are not reminiscent of ones studied in practice. This is the main motivation for our work.

Since MSA is NP-hard, heuristics were devised, including MACAW [22], DIALIGN [14], ClustalW [25], T-Coffee [21], and POA [12]. Many of these methods share the observation that aligned segments of the pairwise alignments are the basis for the multiple alignment process. Lee *et al.* [12] argued that the *only* information in MSA is the aligned sub-sequences and their relative positions. Indeed, many methods (e.g., [14,22,21]) align all pairs of sequences as a preprocessing step and reason about the similar parts; the additional computational cost of $O(n^2k^2)$ is not considered a problem. In progressive methods, this observation percolates to the order of adding the sequences to the alignment [21,25,5]. Other methods assemble an alignment by combining segments in an order dictated by their similarity [22,14]. The Carrillo-Lipman method restricts the full DP according to the pairwise similarities [4]. Unfortunately, none of these methods guarantee an optimal alignment. Another expression of this observation is scoring, and then matching, of full segments rather than single residues [16,14,21,27]. See [20] for recent results on MSA.

Alternatively, researchers designed optimal algorithms for (mostly pairwise) sequence alignment that are faster than building the full DP table. The algorithms of Eppstein *et al.* [6,7] modify the objective function for speedup. Wilbur and Lipman [26,27] designed a pairwise alignment algorithm that offers a tradeoff between accuracy and running time, by considering only matches between identical "fragments". Myer and Miller [18] and Morgenstern [15] designed efficient solutions for special cases of the segment matching problem. In particular, the case considered by Myer and Miller can be solved in polynomial time [18], while the general problem is NP-hard.

In this study, we identify combinatorial properties that are amenable to faster DP algorithms for MSA, and are biologically reasonable. We measure the efficiency of a DP solution by the number of table updates; this number is correlated with both the time and memory required by the algorithm. We suggest a way to exploit the fact that the input sequences are not general, but rather naturally occurring — some of their segments are evolutionary related, while others are not.

We define and study the *Multiple Sequence Alignment from Segments* (MSAS) problem, a generalization of MSA. Intuitively, MSAS accounts for assumptions regarding the pairwise characteristics of the optimal MSA. In MSAS, the input also includes a segmentation of the sequences, and a set of matching segment pairs. As in the original problem, we seek an MSA that optimizes the objective score. However, only corresponding positions in matching segments may be aligned. Trivially, one can segment the sequences into individual letters and specify all possible segment (letter) pairs, each

with their substitution matrix score, getting back the original MSA problem. However, for biological sequences we can often postulate that only solutions that conform to some pairwise alignments are valid, e.g. when segments of different sequences clearly match, or clearly do not match. Using these assumptions, we develop a more efficient DP algorithm.

We then prove that the MSAS problem is essentially equivalent to the *segment matching problem*. This equivalence implies that it is enough to match segments, rather than individual positions. In particular, the complexity of DP algorithm for MSA, and, indeed, *any* algorithm for MSA, depends on the number of *segments* in each sequence, rather than the number of letters. We show that in practice this reduces the number of table updates by several orders of magnitude. For example, aligning five human proteins (denoted by their Swiss-Protidentifiers) GBAS, GBI1, GBT1, GB11, and GB12 requires 4.3×10^8 rather than 6.6×10^{12} table updates. Nonetheless, we prove that in general it is NP-hard.

We can make the algorithm even faster, while still guaranteeing an optimal solution, by further decoupling the sub-problems computation. Essentially, this improved DP algorithm avoids some of the nodes in the k-dimensional grid when calculating the optimal path. Indeed, in practice it outperforms naive DP, and the MSA of the example mentioned above requires only 1.5×10^5 table updates.

Lastly, we further study the combinatorial structure of the problem by considering two additional assumptions, and the performance improvement they imply. The following assumptions may hold in some cases of aligning DNA sequences, where a match indicates (near) identity. Here, we assume that the segment matches have a transitive structure, i.e., if segment A matches segment B, and B matches C, then A necessarily matches C. Also, an optimal alignment is one of minimal width, rather than optimal under an arbitrary scoring function. We prove that under these assumptions, an optimal alignment has a specific structure, which leads to a faster algorithm.

The paper is organized as follows: In Section 2 we define the MSA problem and cast it into a graph-theoretic framework; for completeness, we mention the straightforward DP solution. In Section 3 we present the MSAS problem and prove its equivalence to the segment matching problem, leading to a faster algorithm. We improve the running time even more by considering only "relevant directions" in Section 3. We describe our implementation in Section 4, including the conversion of pairwise alignments to the input format of MSAS, and give several examples of the performance when aligning human proteins. Lastly, in Section 5 we show that a transitivity assumption on the matches leads to further improved efficiency.

2 Multiple Sequence Alignment

The input of a *Multiple Sequence Alignment* (MSA) problem is a set $\mathbb{S} = \{\sigma_1, \ldots, \sigma_k\}$ of k sequences of lengths $n_1, \ldots, n_k$ over an alphabet Σ and a scoring function $f : (\Sigma \cup \{-\})^* \to \mathbb{R}$ (where the gap sign, "$-$", is not in Σ). A multiple alignment of the sequences is a $k \times n$ matrix with entries from $\Sigma \cup \{-\}$. In the ith row the letters of the ith sequence appear in order, possibly with gap signs between them. The score of a column of the matrix is the value of f on the k-tuple that appears in that column. The

FindOptimalPath(x) (Version 0)

1. If $x = \mathbf{0}$ return $\mathbf{0}$
2. For all $\emptyset \neq I \subseteq [k]$
   2.1 If p_{x-e_I} is undefined, compute $p_{x-e_I} = \text{FindOptimalPath}(x - e_I)$
3. $I = \arg\max_{J \subseteq [k]} s(x, x - e_J) + s(p_{x-e_J})$
4. Return the path x, p_{x-e_I}.

Fig. 1. Basic DP MSA algorithm

score of a multiple alignment of $\mathbb{S}$ is the sum of scores over all columns. The objective in the MSA problem is to find an alignment of $\mathbb{S}$ with optimal score. Without loss of generality, we consider scoring functions whose optimum is a maximum, rather than a minimum. Other formulations of MSA, which have been suggested (e.g. [12,16]), are beyond the scope of this work.

We first define our notation: Let $I \subseteq [k]$, where $[k] := \{1, \ldots, k\}$. We denote by $e_i \in \{0,1\}^k$ the vector that is zero in all coordinates except the ith, where it is 1, and $e_I = \sum_{i \in I} e_i$. For a vector $x = (x_1, \ldots, x_k) \in \mathbb{N}^k$ let $x|_I$ be the projection of x onto the subspace spanned by $\{e_i\}_{i \in I}$, i.e., the ith coordinate of $x|_I$ is x_i if $i \in I$, and 0 otherwise. For two vectors, $x, y \in \mathbb{N}^k$ we say that x *dominates* y, and write $x > y$ if $x_i \geq y_i$ for $i = 1, \ldots, k$. We study the directed graph $\mathbb{G}_0$ — its vertex set is $[n_1] \cup \{0\} \times [n_2] \cup \{0\} \times \ldots \times [n_k] \cup \{0\}$, and there is an edge (x, y) in $\mathbb{G}_0$ if and only if $x > y$ and $x - y = e_I$ for some $\emptyset \neq I \subset [k]$; in this case we call I the *direction* that leads from x to y.

The paths from the vertex $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_0$ correspond to alignments of the input sequence. Let p be such a path. Consider $(x, x - e_I)$, the jth edge that the path transverses: In the corresponding sequence alignment, the jth column is a k-tuple that aligns positions x_i of sequences $i \in I$, and has a gap in the rest (in this case we say that the path matches position x_i of sequence i and position $x_{i'}$ of sequence i', for all $i, i' \in I$). We define $s : E(\mathbb{G}_0) \to \mathbb{R}$ to be a scoring function over the edges of $\mathbb{G}_0$, based on the scoring function f over the columns of the alignment. The function s assigns to an edge the value that f assigns to the corresponding column. We also extend s to paths, or sets of edges $E' \subseteq E(\mathbb{G}_0)$: $s(E') = \sum_{e \in E'} s(e)$. It is not hard to see that every such path defines a multiple alignment, and that every multiple alignment can be described by such a path.

In MSA we seek a *maximal (scoring) path* from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_0$. The well-known DP solution to this problem is straightforward; we sketch it in Figure 1. Most importantly, we store the optimal scores of subproblems that have been solved recursively to avoid recomputing them later. For each vertex $x \in \mathbb{G}_0$, we compute the optimal path from x to the origin, denoted p_x, by considering the optimal scores of all its neighbors that are closer to the origin. Thus, we calculate the optimal MSA by calling FindOptimalPath$(n_1, \ldots, n_k)$. The time complexity of the algorithm is the number of edges in $\mathbb{G}_0$, i.e., $\Theta(2^k \prod_{j=1}^{k} n_j)$.

3 MSA from Segments

In this section we formulate Multiple Sequence Alignment from Segments (MSAS) — a generalization of MSA. We assume a preprocessing step that partitions the sequences into segments and matches pairs of these segments. These define a subgraph $\mathbb{G}_1 \subseteq \mathbb{G}_0$, and we then consider the restricted problem of finding an optimal path in $\mathbb{G}_1$. Intuitively, $\mathbb{G}_1$ disallows some of the pairwise alignments in $\mathbb{G}_0$ and consequently in the optimal alignment; clearly, we can allow all the diagonals in $\mathbb{G}_0$ (by segmenting the sequences into letters), leaving the MSA problem unchanged. Next, we show that the vertices of $\mathbb{G}_1$ can be condensed, yielding an even smaller graph $\mathbb{G}_2$; the vertices in $\mathbb{G}_2$ correspond to the segments of input sequences computed in the preprocessing step. The problem is now reduced to computing an optimal path in $\mathbb{G}_2$, which we refer to as the *segment matching problem*. Finally, we show that for computing the optimal path at a vertex it suffices to consider a subset of directions – the so-called relevant directions. We discuss the implementation of the algorithm and elaborate on the preprocessing step in Section 4.

Preliminaries.

Definition 1. *For a sequence q of length n, a* segmentation *of q is a sequence of* extremal *points $0 = c_0 \leq c_1 \leq \ldots \leq c_l = n$. The interval $[c_{i-1}+1, c_i]$ is called the ith segment of q. The extremal point c_i is said to be the* entry point *into segment i (for $i = 1, \ldots, l$), and the* exit point *from segment $i+1$ (for $i = 0, \ldots, l-1$). Denote by l_j the number of segments in the jth sequence.*

Definition 2. *A* segment matching graph *(SMG) over k segmented sequences is a k-partite undirected weighted graph with vertex set $\{(j,i) : j \in [k], i \in [l_j]\}$. Each vertex has an edge connecting it to itself. In addition, vertices (j_1, i_1) and (j_2, i_2) may be connected if the i_1th segment of sequence j_1 has the same length as the i_2th segment of sequence j_2, and $j_1 \neq j_2$.*

An edge $e = ((j_1, i_1), (j_2, i_2))$ in the SMG signifies a match between segment i_1 in sequence j_1 and segment i_2 in sequence j_2. Let l be the (same) length of these segments, and x_1 and x_2 their exit points on sequences j_1 and j_2, respectively, then for $t = 1, \ldots, l$, the edge e implies a match between position $x_1 + t$ of sequence j_1 and position $x_2 + t$ of sequence j_2.

The input to the MSAS problem is a set of segmented sequences, and a list of matching segments, described by an SMG M. The objective is still finding the highest scoring sequence alignment, but with the following restrictions. First, two sequence positions may be aligned together only if they appear in matching segments, and in the same relative position therein. Second, the score of a multiple match depends only on the weights of the corresponding edges in the SMG (and not on the letters themselves). In other words, we can think of the domain of the scoring function as being k-tuples of segments, rather than positions.

The intuition behind these restrictions is that the preprocessing stage identifies matching segments, and commits the algorithm to them. Furthermore, it assigns a "confidence level" (or the weight) to each match, and the objective is to find a highest-scoring alignment, with respect to these values. Here, the segments of each sequence

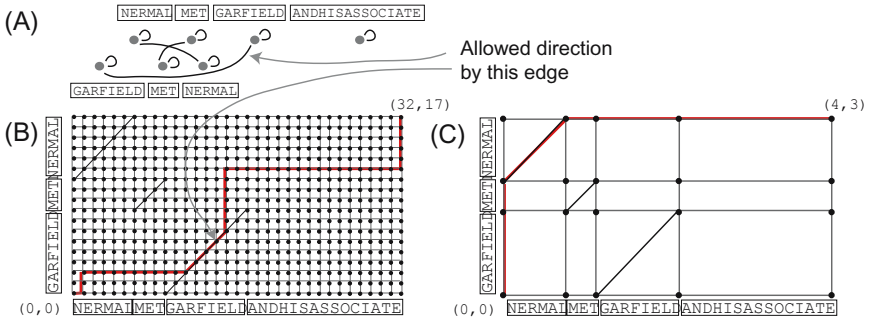

Fig. 2. Example of an SMG for two sequences: Panel (A) shows the sequences, their partitioning and the SMG where each segment corresponds to a (gray) node. Panel (B) shows $\mathbb{G}_1$. Unlike $\mathbb{G}_0$ that has all diagonals, the diagonals in $\mathbb{G}_1$ are defined by the SMG. An allowed path in $\mathbb{G}_1$ is also shown. Panel (C) shows $\mathbb{G}_2$, and an allowed path in it. The directions of the edges are omitted in the illustration for clarity, but are towards the origin.

are non-overlapping. In practice, we derive the segments from aligned portions of two sequences, and these may be overlapping. This is resolved by splitting the overlapping segments to smaller non-overlapping ones, as we discuss in Section 4.

Formally, given a set of segmented sequences, and an SMG M, we define $\mathbb{G}_1(M)$ as follows. It is a subgraph of $\mathbb{G}_0$, containing all vertices. The edge $(x, x - e_I)$ is in $\mathbb{G}_1(M)$ if and only if for all $i, j \in I$ there is an edge $m \in E(M)$, such the position x_i on sequence i is matched to position x_j on sequence j. In this case we say the I is an *allowed direction* at x, and that m is a match *defining* the edge $(x, x - e_I)$. The score of such an edge depends only on the weights of the corresponding edges in M (e.g., the sum-of-pairs scoring function). It is not hard to see that if x and y are vertices such that $x_I = y_I$ and I is allowed at x, then I is also allowed at y. Note also, that because all vertices in M have an edge connecting them to themselves, for $i \in [k]$, $\{i\}$ is an allowed direction at all vertices x such that $x_i > 0$.

As in the MSA problem, the goal in the MSAS problem is to find a highest scoring path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$. Clearly the previously mentioned DP algorithm solves this restricted MSA problem as well. In the following subsections we describe how it can be improved.

MSAS and Segment Matching. The vertices of $\mathbb{G}_1$ correspond to k-tuples of positions along the input sequences, one from each sequence. We now define the graph $\mathbb{G}_2$, a "condensed" version of $\mathbb{G}_1$, whose vertices correspond to k-tuples of *segments*. That is, its vertex set is $[l_1] \cup \{0\} \times [l_2] \cup \{0\} \times \ldots \times [l_k] \cup \{0\}$. There is a directed edge from $z = (z_1, \ldots, z_k)$ to $z - e_I$ in $\mathbb{G}_2$ if for all $i, j \in I$ the z_ith segment of sequence i matches the z_jth segment of sequence j. Define $x \in V(\mathbb{G}_1)$ by taking x_i to be the entry point into the z_ith segment of sequence i. Suppose the length of the segments defining the edge $(z, z - e_I)$ is l (recall that two segment match only if they are of the same length). Observe that $(z, z - e_I) \in E(\mathbb{G}_2)$ implies that $(x, x - e_I), (x - e_I, x - 2e_I), \ldots, (x - (l-1)e_I, x - l \cdot e_I)$ are all edges in $\mathbb{G}_1$. In this sense, $(z, z - e_I)$ is a "condensation" of all these edges. Define the score of the edge $(z, z - e_I)$ as the sum

FindOptimalPath(x) (Version 1)

1. If $x = \mathbf{0}$ return $\mathbf{0}$.
2. For all $y = x - l \cdot e_I$ that is extremal with respect to x
   2.1 If p_y is undefined, compute $p_y = \text{FindOptimalPath}(y)$
   2.2 $d_y = l \cdot s(x, x - e_I)$
3. $y^* = \arg\max s(p_y) + d_y$
4. Return the path x, p_{y^*}.

Fig. 3. Segment based DP MSA algorithm

of the scores of all the edges in $\mathbb{G}_1$ that it represents. Since the score depends only on the segments, this is simply $l \cdot s(x, x - e_I)$.

The *segment matching* problem is to find a highest-scoring path from $(l_1, \ldots, l_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_2$. Clearly the same DP algorithm as above can be used to solve this problem in time $\Theta(2^k \prod_{j=1}^{k} l_j)$. Hence, when the sequences are long, but consist of a small number of segments, DP for solving the segment matching problem may be plausible, while solving the MSA problem might not.

In the sequel of this section we prove that in order to find an optimal solution to the MSAS problem, it is enough to solve the associated segment matching problem. To state this precisely, we need the following definition.

Definition 3. *Let x be a vertex in $\mathbb{G}_1(M)$. We say that x is an* extremal *vertex if for all $i \in [k]$, x_i is an extremal point of sequence i.*
We say that y is extremal with respect to *x, if it is the first extremal vertex reached when starting at x and repeatedly going in direction I, for some allowed direction I. Denote $X(x) = \{y \in V(\mathbb{G}_1(M)) : \ y \text{ is extremal w.r.t. } x\}$.*

Theorem 1. *There is an optimal path, $p = p_1, \ldots, p_v$, such that if $x^1, \ldots, x^u$ are the extremal points, in order, through which it passes, then $x^{i+1} \in X(x^i)$.*

Observe that in particular, the theorem says that segments are either matched in their entirety, or not matched at all. Hence, any solution to the segment matching problem defines an optimal solution of the MSAS problem. In other words, it suffices to solve the problem on the "condensed" graph $\mathbb{G}_2$. While Theorem 1 is intuitively clear, the proof is somewhat involved, and omitted from this version. Figure 3 sketches the revision of the DP algorithm based on Theorem 1.

Narrowing the Search Space: Relevant Directions. Consider an input to the MSAS problem that consists of two subsets of k sequences each. Suppose that none of the segments in the first subset match any of those in the second subset. Naively applying the algorithm above will require running time exponential in $2k$. Yet clearly the problem can be solved on each subset independently, in time exponential in k rather than $2k$. Intuitively, this is also the case when there are only few matches between the two subsets. We make this notion explicit in this subsection. Again, we start with some definitions:

FindOptimalPath(z) (Version 2)

1. If $z = \mathbf{0}$ return $\mathbf{0}$.
2. D = minimal set of directions that intersect a subset of independent relevance.
3. For all $\emptyset \neq I \in D$
   3.1 If p_{z-e_I} is undefined, compute $p_{z-e_I} = \text{FindOptimalPath}(z - e_I)$
4. $I = \arg\max_{J \in D} s(z, z - e_J) + s(p_{z-e_J})$
5. Return the path z, p_{z-e_I}.

Fig. 4. Version 2 of MSA algorithm. Details on how to compute D are given in the full version

Definition 4. *Let x be a vertex in $\mathbb{G}_2(M)$. Let $((i, y_i), (j, y_j))$ be a match in the SMG. We say that such a match is* relevant *for x at coordinate i, if $x_i = y_i$ and $x_j > y_j$.*
We say that a subset of indices $S \subset [k]$ is of independent relevance *at x if for all $i \in S$ the match $((i, y_i), (j, y_j))$ is relevant for x at coordinate i implies $j \in S$.*

Theorem 2. *Let p be an optimal path in $\mathbb{G}_2$, and x a vertex on it. Let S be a subset of indices of independent relevance at x. Then there is an optimal path p' that is identical to p up to x, and from x goes to $x - e_I$ for some $I \subset [k]$ such that $I \cap S \neq \emptyset$.*

Proof: Let y be the first vertex on p after x, such that $y_i = x_i - 1$ for some $i \in S$. Define p' to be the same as p up to x, and from y onwards. We will define a different set of allowed directions that lead from x to y. Let $I_1, \ldots, I_t$ be the directions followed from x to y. Let $i \in I_t \cap S$. For all $i \neq j \in I_t$, there is a match between (i, x_i) and $(j, y_j + 1)$. Hence, either $j \in S$, or $y_j + 1 = x_j$. Since y is the first vertex in p that differs from x on a coordinate in S, if $j \in S$, then $j \notin I_1, \ldots I_{t-1}$. Clearly, if $y_j + 1 = x_j$ then again $j \notin I_1, \ldots I_{t-1}$. In other words, for all $h < t$, we have $I_h \cap I_t = \emptyset$. Define p' to follow directions $I_t, I_1, \ldots, I_{t-1}$ from x. As I_t is disjoint from the other directions, this indeed defines an allowed path from x to y, and $i \in I_t \cap S$. ∎

The theorem implies that in the DP there is no need to look in *all* directions. Let S be a subset of independent relevance at a point x, then to compute the optimal path from x to the origin it is enough to consider paths from $x - e_I$ to the origin for $I \subset [k]$ such that $I \cap S \neq \emptyset$. This suggests the DP algorithm sketched in Figure 4 (this time think of z as a vertex in $\mathbb{G}_2$). Note that to implement this algorithm there is no need to keep a table of size $|V(\mathbb{G}_2)|$. The vertices that are actually visited by the algorithm can be kept in a hash table.

4 Implementing the Algorithm

We have implemented Version 2 of our algorithm, described in Figure 4. Using our implementation of the algorithm, we investigate its efficiency (measured in the number of vertices it visits, or table updates) on real biological sequences. We first describe the preprocessing step that constructs the SMG, and then discuss the performance of the algorithm on a few examples. We stress that *efficiency* is indeed the property of interest here, as the multiple alignment found is an *optimal* solution for the MSAS problem.

Generating a Segment Matching Graph (SMG). Existing tools, such as BLAST [2] or DIALIGN [16], provide local alignments rather than the input format that we assumed previously. In order to restrict the problem only to MSAs that conform to these local pairwise alignments, we must convert them to an SMG. In particular, we need to segment the sequences, and allow matches only between equal-length segments.

Starting with the set of sequences, we add breakpoints onto them based on the local alignments. This way, we progressively build the SMG, stopping when all local alignments are properly described. The ends of an alignment define breakpoints in the two aligned sequences. If the segments between those breakpoints have the same length, we simply add a connecting edge (or edges) to the SMG. However, the segments lengths may differ due to two reasons: First, gapped alignments match segments of unequal length; we solve this by adding breakpoints at the gap ends. Second, regions of the sequences corresponding to different alignments may overlap; we solve this by adding breakpoints at the ends of the overlapping region (or regions). Notice that if we add a breakpoint inside a segment that already has an edge associated with it, we must split the edge (and a corresponding breakpoint must be added to the connected segment).

Table 1. Number of table updates for three sets of human proteins. We compare full DP (Version 0), full DP on the Segment Matching Graph (Version 1), and the actual number of table updates when considering only relevant directions (Version 2); the SMG is generated using all significant gapped/un-gapped BLAST alignments. We see that in all cases, the actual work is several orders of magnitudes faster than the DP calculation.

		gapped BLAST		un-gapped BLAST	
Human proteins	full DP	Version 1	Version 2	Version 1	Version 2
MATK,SRC, ABL1,GRB2	6.65×10^{10}	$91 \cdot 98 \cdot 99 \cdot 89$ =78, 576, 498	7.20×10^{6}	$77 \cdot 84 \cdot 81 \cdot 74$ =38, 769, 192	1, 994, 813
PTK6,PTK7, RET, SRMS, DDR1	2.40×10^{14}	$92 \cdot 96 \cdot 106 \cdot 88 \cdot$ $\cdot 125 = 1.03 \times 10^{10}$	281, 752	$60 \cdot 53 \cdot 66 \cdot 57 \cdot$ $\cdot 58$=3, 736, 260	2, 980
GBAS, GBI1, GBT1, GB11, GB12	6.62×10^{12}	$148 \cdot 116 \cdot 113 \cdot 115 \cdot$ $\cdot 120 = 2.68 \times 10^{10}$	270, 289	$61 \cdot 72 \cdot 68 \cdot 71 \cdot$ $\cdot 70 = 4.3 \times 10^{8}$	145, 366

Example MSAs. We demonstrate the effectiveness of our algorithm by several examples of aligning human protein sequences. We align two sets of proteins from kinase cascades: (1) MATK, SRC, ABL1, and GRB2 of lengths 507, 535, 1130, and 217 respectively. (2) PTK6, PTK7, RET, SRMS, DDR1 of lengths 451, 1070, 1114, 488, and 913 respectively. We also align five heterotrimeric G-protein (subunits alpha) GBAS, GBI1, GBT1, GB11, GB12 of lengths 394, 353, 349, 359, and 380 respectively. We chose these (relatively long) proteins because their "mix-and-match" modular components characteristic highlights the strengths of our method. We use gapped and un-gapped BLAST with E-value threshold of 10^{-2} to find local alignments. Namely, in the optimal MSA two letters can be matched only if they are in a local BLAST alignment with E-value at most 10^{-2}.

Table 1 lists the number of table updates needed to find the optimal MSA for these alignments. The first column has the size of the full DP matrix, or the product of the

sequences lengths (same for gapped and un-gapped). The second column lists the number of segments in each sequence in the SMG, which was calculated from the BLAST gapped or ungapped alignments, and the size of their DP matrix. The last column has the actual number of vertices visited, or equivalently, the number of table updates. The number of updates drops dramatically, in the best case from 10^{14} to less than 3000. Other alignments that we studied had similar properties to the ones shown. Complete figures of the cases listed in Table 1 are available at [1] in a format that allows zooming for exploring the details.

5 The Transitive MSAS

In this section we further restrict the problem by making the following two assumptions, which allow for additional "shortcuts" in the DP algorithm.

ASSUMPTION 1: The matches are transitive, in the sense that if $\{i, j\}$ is an allowed direction at x, and $\{i, k\}$ is an allowed direction at x, then $\{j, k\}$ is also allowed at x (and hence, $\{i, j, k\}$ as well).
ASSUMPTION 2: The scoring function is such that we seek to find an alignment of minimal width, or equivalently, the shortest path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_0$.

The assumption of transitivity may be too restrictive in many biological relevant cases. We study it here for two main reasons: (1) The assumption holds in special cases of aligning nucleotide sequences, where a match indicates (near) identity; and (2) this analysis illuminates additional properties of the combinatorial structure of the problem, by further limiting the search space. The missing proofs appear in the full version.

Assumption 2 is achieved by setting the scoring function (over the edges of $\mathbb{G}_1$) as $s(x, x - e_I) = |I| - 1$: The longest possible path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ is of length $\sum n_i$. Each edge $(x, x - e_I)$ "saves" $|I| - 1$ steps in the path, exactly its score. Hence, a shortest path, or the one that "saves" the most steps, is the highest scoring one. Since this scoring function is so simple over $\mathbb{G}_1$, it is convenient to return the discussion from $\mathbb{G}_2$ to $\mathbb{G}_1$. At the end of this section we prove that the techniques developed here apply to $\mathbb{G}_2$ as well.

We call the problem of finding the highest scoring path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1(M)$, with s and M as above, the *Transitive MSAS Problem.*

Maximal Directions. The first observation is that an optimal solution to the Transitive MSAS proceeds in "maximal" steps.

Definition 5. *An edge $(x, x - e_I) \in E(\mathbb{G}_1(M))$ is called* maximal*, and the subset I a* maximal direction *(at x), if for all $J \supsetneq I$, the pair $(x, x - e_J)$ is not an edge. We denote by $D(x)$ the collection of maximal directions at vertex x (note that by transitivity, this is a partition of $[k]$). A path in $\mathbb{G}_1(M)$ is called a* maximal path *if it consists solely of maximal edges.*

Lemma 1. *There is an optimal path in $\mathbb{G}_1(M)$ that is also maximal.*

Henceforth, by "optimal path" we refer to a maximal optimal path. As a corollary of Lemma 1, the DP algorithm for the transitive MSA problem needs not check *all* directions (or all those that intersect a subset of independent relevance), only maximal ones.

This reduces the time complexity of the algorithm to $O(k \prod l_i)$, with a data structure that allows finding the maximal directions at a given vertex in $O(k)$. Details will be provided in the full version.

Obvious Directions. The notion of "relevant directions" discussed in Section 3 can be strengthened in the transitive setting. Indeed, there is a simple characterization of vertices in $\mathbb{G}_1$ for which the first step in an optimal path is obvious, and there is no need for recursion.

Definition 6. *Let x be a vertex in $\mathbb{G}_1(M)$ and I a maximal direction at x. The set I is called an* obvious direction *(at x) if for all $y \in \mathbb{G}_1(M)$, $y < x$, such that $x|_I = y|_I$, I is a maximal direction at y. If $y = x - c \cdot e_I$ is extremal with respect to x, and I is an obvious direction at x, we say that y is an* obvious vertex *with respect to x.*

Lemma 2. *Let p be an optimal path, x a vertex in p and I an obvious direction at x. Then there is an optimal path p' that is identical to p up to x, and that proceeds to $x - e_I$ from x.*

Corollary 1. *There is an optimal path p, such that if x is an extremal vertex in p, and y is obvious with respect to x, then p proceeds from x to y.*

Intuitively, obvious directions are cases where all benefits to the scoring function can be gained in the first step, or equivalently, there are no tradeoffs to consider. Hence, as for relevant directions, the DP algorithm can be revised to immediately move to an obvious vertex, avoiding the recursion over all extremal vertices.

Special Vertices. In this section we extend the "leaps" that the DP algorithm performs. Once more, we start with a few definitions.

Definition 7. *We say that a vertex y is* special *with respect to a vertex x if the following four conditions hold: (1) x dominates y; (2) $D(x) \neq D(y)$; (3) there is a path from x to y consisting solely of maximal edges; and (4) no vertex y' satisfies all the above, and dominates y. Denote by $S(x)$ the set of vertices that are special with respect to x.*

We define the set of *special vertices* $S \subseteq \mathbb{G}_1(M)$ as the smallest one such that $(n_1, \ldots, n_k) \in S$, and for every $x \in S$, $S(x) \subset S$. We first show that instead of "leaping" from one extremal vertex to another, we can "leap" from one special vertex to another.

Definition 8. *Let $p = (p_0, \ldots, p_r)$ and $p' = (p'_0, \ldots, p'_r)$ be two paths. Let $I_1, \ldots, I_r$ be the sequence of directions that p moves in, and $I'_1, \ldots, I'_r$ be the sequence of directions that p' does. We say that p and p' are equivalent if $p_0 = p'_0$, $p_r = p'_r$ and there is some permutation $\sigma \in S_r$ such that $I_i = I'_{\sigma(i)}$ for $i = 1, \ldots, r$.*

Note that equivalent paths have the same length, and hence the same score. We also observe:

Lemma 3. *Let p be an optimal path. Let x be a vertex in p, and let y be the first vertex in p that is also in $S(x)$. Then all maximal paths from x to y are equivalent.*

Let $p = (p_1, \ldots, p_t)$ be an optimal path. Define $x_1 = p_1$ and x_{i+1} to be the first vertex in p that is also in $S(x_i)$. Lemma 3 says that we only need to specify the vertices $\{x_i\}$ to describe an optimal path — all maximal paths connecting these vertices in order are equivalent.

As a corollary, we can further restrict the search space of the DP algorithm. When computing the shortest path from a vertex x, rather than considering the relevant extremal vertices, it is enough to consider the special ones. As we shall soon show, this is indeed a subset of the extremal vertices.

Before describing the modified algorithm in detail, let us observe that special points have a very specific structure.

Definition 9. *Let $x, y \in \mathbb{G}_1(M)$ be such that y is special with respect to x. Let I and I' be maximal directions at x. We say that y is a* breakpoint of direction I*, if $y = x - c \cdot e_I$ for some natural c, and I is not allowed at y.*
We say that y is a straight junction of direction I *if $y = x - c \cdot e_I$ for some natural c, and I is allowed, but not maximal, at y.*
We say that y is a corner junction of directions I and I' *if $y = x - c \cdot e_I - c' \cdot e_{I'}$ for some natural c and c', and I and I' are allowed, but not maximal, at y.*

Theorem 3. *Let y be a special vertex with respect to x. Then y is one of the types in definition 9. Furthermore, if x is an extremal vertex, then so is y.*

Corollary 2. *All special vertices are extremal vertices.*

This suggests a further improved DP algorithm that runs on $\mathbb{G}_2$. We defer the pseudo code listing and a detailed analysis of the running time of the algorithm to the full version. The analysis shows that the running time is linear in the number of segments and the number of special vertices, and at most cubic in the number of sequences.

Acknowledgements

P.K.A is supported by National Science Foundation (NSF) grants CCR-00-86013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.–Israel Binational Science Foundation. Y.B. was supported by the Israeli Ministry of Science. R.K. is supported by NSF grant CCR-00-86013. We are grateful to Chris Lee and Nati Linial for enlightening discussions on these topics.

References

1. http://trantor.bioc.columbia.edu/~kolodny/MSA/
2. Altschul, F. Stephen, L.M. Thomas, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: A new generation of protein database search programs *Nucleic Acids Res.*, 25:3389–3402, 1997.
3. P. Bonizzoni and G. Della Vedova. The complexity of multiple sequence alignment with sp-score that is a metric. *Theoretical Computer Science*, 259(1-2):63–79, 2001.
4. H. Carrillo and D. Lipman. The multiple sequence alignment problem in biology. *SIAM J. Applied Math.*, 48(5):1073–1082, 1988.

5. F. Corpet. Multiple sequence alignment with hierarchical-clustering. *Nucleic Acids Research*, 16(22):10881–10890, 1988.
6. D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic-programming: I. linear cost-functions. *JACM*, 39(3):519–545, 1992.
7. D. Eppstein, Z. Galil, R. Giancarlo, and G. F. Italiano. Sparse dynamic-programming: II. convex and concave cost-functions. *JACM*, 39(3):546–567, 1992.
8. M. R. Garey and D. S. Johnson. *Computers and Intractability–A Guide to the Theory of NP-completeness*. Freeman, San Francisco, 1979.
9. S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.*, 89: 10915–10919, 1992.
10. T. Jiang and L. Wang. On the complexity of multiple sequence alignment. *J. Comp. Biol.*, 1(4):337–48, 1994.
11. G. M. Landau M. Crochemore and M. Ziv-Ukelson. A sub-quadratic sequence alignment algorithm for unrestricted cost matrices. *Proc. 13th Annual ACM-SIAM Sympos. Discrete Algo.*, 679–688, 2002.
12. C. Lee, C. Grasso, and M. F. Sharlow. Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464, 2002.
13. W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *J. Comput. Sys. Sci.*, 20(1):18–31, 1980.
14. B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999.
15. B. Morgenstern. A simple and space-efficient fragment-chaining algorithm for alignment of DNA and protein sequences *Applied Math. Lett.*, 15(1), 11–16, 2002.
16. B. Morgenstern, A. Dress. and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Nat. Acad. Sci.*, 93(22):12098–12103, 1996.
17. M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous comparison of 3 protein sequences. *Proc. Nat. Acad. Sci.*, 82(10):3073–3077, 1985.
18. G. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. *Proc. 6th Annual ACM-SIAM Sympos. Discrete Algo.*, 38–47, 1995.
19. S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *J. Mol. Biol.*, 48:443–453, 1970.
20. C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002.
21. C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: A novel method for fast and accurate multiple sequence alignment. *J. Mol. Biol.*, 302(1):205–217, 2000.
22. G. D. Schuler, S. F. Altschul, and D. J. Lipman. A workbench for multiple alignment construction and analysis. *Proteins-Structure Function And Genetics*, 9(3):180–190, 1991.
23. R. M. Schwartz and M. O. Dayhoff. Matrices for Detecting Distant Relationships. *Atlas of Protein Sequences and Structure, (M.O. Dayhoff, ed.)*, 5, Suppl. 3 (pp; 353-358), National Biomedical Research Foundation, Washington, D.C., USA.
24. T. F. Smith and M. S. Waterman. Comparison of biosequences. *Adv. Applied Math.*, 2(4), 482–489, 1981.
25. 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.
26. W. J. Wilbur and D. J. Lipman. Rapid similarity searches of nucleic-acid and protein data banks. *Proc. Nat. Acad. Sci.*, 80(3):726–730, 1983.
27. W. J. Wilbur and D. J. Lipman. The context dependent comparison of biological sequences. *SIAM J. Applied Math.*, 44(3):557–567, 1984.