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

Pankaj K. Agarwal† Yonatan Bilu ‡ Rachel Kolodny§

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, several heuristics were suggested for the problem. We present several techniques for making the dynamic programming algorithm more efficient, while still finding an *optimal* solution. We solve 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 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. We also identify "shortcuts" that expedite the dynamic programming scheme. Under some more assumptions, namely, that matches between segments are transitive, we show how to further improve the running time for finding the optimal solution by restricting the search space of the dynamic programming algorithm.

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 [10, 16, 27].

Dynamic programming (DP) algorithms compute an optimal Multiple Sequence Alignment for a wide range of scoring functions. In 1970, Needleman and Wunsch [26] proposed a DP algorithm for pairwise alignment, which was later improved by Masek and Paterson [20]. Murata *et al.* [24] extended this algorithm to aligning k sequences (each of length n). It constructs a k-dimensional grid of size $O(n^k)$, with 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 [13, 3, 14, 19]. This is shown by reduction from Max-Cut and Vertex Cover [9], that is, instances to 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 in NP-hard, many heuristics were devised, including MACAW [29], DIALIGN [21], ClustalW [32], T-Coffee [28], and POA [17]. 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.* [17] argued that the *only* information in MSA is the aligned sub-sequences and their relative positions. Indeed, many methods (e.g., [21, 29, 28]) align all

---

* 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 Department of Computer Science, Stanford University and visiting Duke University.

† Department of Computer Science, Box 90129, Duke University, Durham NC 27708-0129, USA; e-mail:pankaj@cs.duke.edu

‡ Department of Molecular Genetics, Weizmann Institute, 76100 Rehovot,Israel; email:johnblue@cs.huji.ac.il

§ Department of Biochemistry and Molecular Biophysics, Columbia University; e-mail:rachel.kolodny@columbia.edu

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 [8, 28, 12, 32, 5]. Other methods assemble an alignment by combining segments in an order dictated by their similarity [29, 21]. The Carrillo-Lipman method restricts the full DP according to the pairwise similarities [4]. Unfortunately, none of these methods guarantees finding an optimal alignment. Another expression of this observation is scoring, and then matching, of full segments rather than single residues [23, 21, 18, 28, 34]. See in [16, 27] 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. Crochemore *et al.* [15] devised a sub-quadratic time alignment by exploiting the compressibility of typical input sequences. Wilbur and Lipman [33, 34] designed a pairwise alignment algorithm that offers a tradeoff between accuracy and running time, by considering only matches between identical "fragments". Myer and Miller [25] and Morgenstern [22] designed efficient solutions for special cases of the segment matching problem.

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 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 formalize the assumption that MSA relies on a preliminary alignment of all pairs in the set of sequences by studying *Multiple Sequence Alignment from segments* (MSAS). 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 which optimizes the objective score. However, only positions that are in matching segments may be aligned.

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-Prot [2] identifiers) 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 the 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: the MSA of the example mentioned above requires only 1.5×10^5 table updates.

When aligning DNA sequences, we consider two more biologically reasonable assumptions, which in turn allow more performance improvements. In MSA of DNA sequences, a match indicates (near) identity. We thus 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. Thus, we can more efficiently search only among alignments of this structure (rather than the entire space).

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 3.3. 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 discuss a transitive version of MSA, as well as an even faster algorithm for finding an optimal solution to it.

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 $f : (\Sigma \cup \{-\})^* \rightarrow \mathbb{R}$ (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 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 optimal is a maximum.

We first define our notation: Let $I \subseteq [k]$. 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 $x = (x_1, \ldots, x_k) \in \mathbb{N}^k$, $x|_I$ is 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 the 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 $\mathbb{G}_0$ iff $x > y$ 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.

FindOptimalPath(x) (Version 0)

1. If $x = \vec{0}$ return $\vec{0}$
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}.

Figure 1. Basic DP MSA algorithm.

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 $s : E(\mathbb{G}_0) \rightarrow \mathbb{R}$ be a score function over the edges of $\mathbb{G}_0$, based on the score function f over the columns of the alignment. s assigns to an edge the value which 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 an *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 $\mathbb{G}_0$: $\Theta(2^k \cdot \prod_{j=1}^{k} n_j)$.

3. MSA from segments

In this section we formulate Multiple Sequence Alignment from Segments (MSAS) — a variant of MSA. Intuitively, we assume a preprocessing step in which the sequences are partitioned into segments and pairwise matches between them are found. These define a subgraph $\mathbb{G}_1 \subseteq \mathbb{G}_0$, and we then consider the restricted problem of finding an optimal path in $\mathbb{G}_1$ (Section 3.1). 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* (Section 3.2). 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 (Section 3.3). We elaborate the preprocessing step in section 4.1, where we discuss the implementation of the algorithm.

3.1. Preliminaries

DEFINITION 3.1 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*

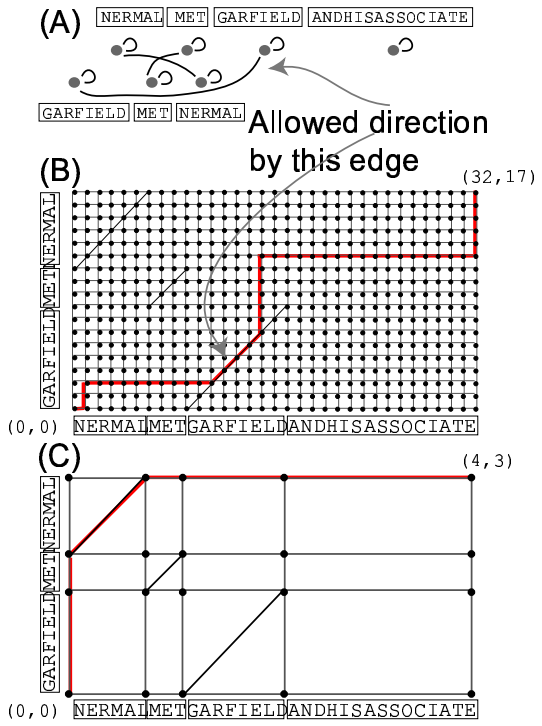

Figure 2. **Example of an SMG for two sequences: Panel (A) shows the sequences, their partitioning and the SMG where each segment corresponds to a (gray) node. Panel (B) shows $\mathbb{G}_1$. Unlike $\mathbb{G}_0$ that has all diagonals, the diagonals in $\mathbb{G}_1$ are defined by the SMG. Panel (C)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

point from segment $i+1$ (for $i = 0, \ldots, l-1$). Denote by l_j the number of segments in the jth sequence.

DEFINITION 3.2 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 $t = 1, \ldots, l$, 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 score 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.

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. $(x, x - e_I)$ is an edge in $\mathbb{G}_1(M)$ iff 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 y. Note also, the because all vertices in M have edges 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.

3.2. 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 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 se-

quence 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 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 $\Theta(2^k \cdot \prod_{j=1}^{k} l_j)$. 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 a couple of additional definitions:

DEFINITION 3.3 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 its the first extremal vertex reached when starting at x an 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 3.1 *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 3.1 is intuitively clear, the proof is somewhat involved, and appears in Appendix A. Figure 3 sketches the revision of the DP algorithm based on Theorem 3.1.

FindOptimalPath(x) (Version 1)

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

Figure 3. Segment based DP MSA algorithm.

3.3. 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 roughly the case when there are some matches between the two subsets, but very few of them. We make this notion explicit in this subsection. Again, we start with some definitions:

DEFINITION 3.4 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 $i \in S$ $((i, y_i), (j, y_j))$ being relevant for x at coordinate i implies $j \in S$.

THEOREM 3.2 *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'll 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's 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 $h < t$, $I_h \cap I_t = \emptyset$. 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's 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's 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 following DP algorithm (this time think of z as a vertex in $\mathbb{G}_2$), sketched in Figure 4:

FindOptimalPath(z) (Version 2)

1. If $z = \vec{0}$ return $\vec{0}$.
2. Let D be a 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}.

Figure 4. Version 2 of the MSA algorithm. Details on how D is computed will be given in the full version.

Note that in order to implement this algorithm there's 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. Using our implementation of the algorithm, we investigate in section 4.2 its complexity (measured in the number of vertices it visits, or table updates) on real biological sequences.

4. Implementing the algorithm

4.1. Generating a Sequence Matching Graph (SMG)

Existing tools, such as BLAST [1] or DIALIGN [23], provide local alignments rather than the input format which we assumed previously. Thus, we must convert the pairwise alignments 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(s) 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(s). 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).

4.2. Example MSAs

We demonstrate the properties of our algorithm by several examples of aligning human protein sequences (denoted by their Swiss-Prot [2] identifiers). 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 10e-2 to find local alignments.

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 ungapped). The second column lists the number of segments in each sequence in the SMG, which was calculated from the BLAST un/gapped 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. In all cases, it yields a feasible computation. Other alignments that we studied had similar properties to the ones shown.

Figure 5 shows an excerpt from the SMG generated based on the local pairwise alignments of three proteins: MATK, SRC, ABL1, and the subset of edges that are the optimal alignment. We show only three proteins to allow a clear picture in the limited space

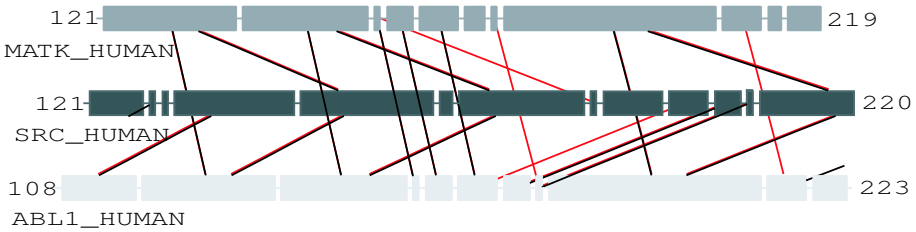

Figure 5. **An excerpt from the SMG and the alignment edges of three human proteins MATK_HUMAN, SRC_HUMAN, and ABL1_HUMAN, starting from residues 121, 121, and 108 respectively. The edges of the SMG are shown in red, and the subset chosen in the alignment in black (overlaying the red). The width of each node is proportional to the number of residues in the segment; the most thin nodes correspond to segments of one residue. The alignments were found with un-gapped BLAST and E-value better than 10e-2.**

Human proteins	full DP	gapped BLAST		un-gapped BLAST	
		DP	actual	DP	actual
MATK,SRC, ABL1,GRB2	$66,511,986,450$	$91 \cdot 98 \cdot 99 \cdot 89$ $=78,576,498$	$7,199,547$	$77 \cdot 84 \cdot 81 \cdot 74$ $=38,769,192$	$1,994,813$
PTK6,PTK7, RET, SRMS, DDR1	2.395168×10^{14}	$92 \cdot 96 \cdot 106 \cdot 88 \cdot 125$ $=10,298,112,000$	$281,752$	$60 \cdot 53 \cdot 66 \cdot 57 \cdot 58$ $=3,736,260$	$2,980$
GBAS, GBI1, GBT1, GB11, GB12	6.621774×10^{12}	$148 \cdot 116 \cdot 113 \cdot 115 \cdot 120$ $= 2.67717792 \times 10^{10}$	$270,289$	$61 \cdot 72 \cdot 68 \cdot 71 \cdot 70$ $= 430,938,144$	$145,366$

Table 1. **Number of table updates for three sets of human proteins. We compare full DP, full DP on the Sequence Matching Graph (SMG), and the actual number of table updates when considering only relevant directions; 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.**

available in a printed format. The width of the nodes in the graph is proportional to the length of their corresponding segments. Complete figures of the cases listed in Table 1, are available in the supplementary material[1] in a format which allows zooming for exploring the details.

We see that there are many instances in which the relevant directions strategy is beneficial (e.g., segments with only self-edges). Also, we notice that there is a high incidence of singletons, that account for a significant fraction of the work, but are not necessarily as influential on the score. This perhaps calls for an (approximation) algorithm (or heuristic) which exploits this property.

1 at http://trantor.bioc.columbia.edu/~kolodny/MSA/

5. The Transitive MSAS

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

ASSUMPTION 1 The score 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 $\mathbb{G}_0$.

ASSUMPTION 2 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 1 is achieved by setting the score function (over the edges of $\mathbb{G}_1$) to be $s(x, x-e_I) = |I|-1$: 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, the one that "saves" the most steps, is

equivalent to the highest scoring one. Since this score 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*.

5.1. Maximal directions

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

DEFINITION 5.1 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 $J \supsetneq I$, $(x, x - e_J)$ 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 5.1 *There is an optimal path in $\mathbb{G}_1(M)$ that is also maximal.*

Proof: Let p be an optimal path. Let $(x, x - e_I)$ be the first edge in the path which is not maximal. We shall construct a path p' that is identical to p up to x; from x it will proceed in direction J such that $I \subsetneq J$ and its length will be at most that of p. This proves the lemma, because by repeating this argument presents a path which is at least as good as p, and consists only of maximal edges (at each iteration a non-maximal direction is replaced by one with strictly more coordinates).

Suppose $J' \supsetneq I$, is an allowed direction at x. Denote $L' = J' \backslash I$. Let y be the first point in p such that $y|_{L'} \neq x|_{L'}$. Let $L \subseteq L'$ be the subset of coordinates in L by which $x|_{L'}$ and $y|_{L'}$ differ. Let $J = I \cup L$.
We now describe p'. It is identical to p up to vertex x and from vertex y onwards. Let $x = p_1, \ldots, p_r = y$ be the vertices p visits (in order) when going from x to y. Replace them in p' by $p'_t = p_t - e_L$, for $t = 2, \ldots, r$. We need to show that the direction taken at x indeed strictly contains I, and that this path is indeed a legal path in $\mathbb{G}_1$.

Observe that $p'_2 = p_2 - e_L = x - e_I - e_L = x - e_J$. Hence, the direction taken at x is J. Indeed $J \subset J'$ is allowed at x, and $I \subsetneq J$. It remains to show that if K is an allowed direction chosen at p_t then it also allowed at p'_t. By the choice of y, for $t < r - 1$, $K \cap L = \emptyset$. By induction, p_t and p'_t are identical on coordinates outside of L, so for $t < r - 1$, K is an allowed direction at p'_t. For $t = r - 1$ the same argument shows that $K \backslash L$ is an allowed direction at p'_{r-1}, which leads to the desired destination, y. ∎

Henceforth, by "optimal path" we refer to a maximal optimal path. As a corollary of Lemma 5.1, there is no need to check *all* directions (or all those that intersect a subset of independent relevance), just maximal ones, in the DP algorithm for the transitive MSA problem. 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.

5.2. Obvious directions

The notion of "relevant directions" discussed in section 3.3 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 5.2 Let x be a vertex in $\mathbb{G}_1(M)$ and I a maximal direction at x. 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 5.2 *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 goes to $x - e_I$ from x.*

Proof: Since I is allowed at x, for $i \in I$ $x_i > 0$, and thus at some point p moves in a direction that includes i, for some $i \in I$. Let x' be the first point when this occurs. Let I' be the direction in which p proceeds from x'. Clearly $x'|_I = x|_I$, thus, as I is obvious at x, and $x' < x$, I is maximal at x', and therefore $I = I'$.
Now denote by $I_1, \ldots, I_r$ the sequence of directions p moves along from x to x'. Define p' as identical to p up to x. From x it goes to $x - e_I$. It then proceeds in order along directions $I_1, \ldots, I_r$ (it is easy to see that since all edges in the path are maximal, they are indeed allowed). Clearly this leads to $x' - e_I$, and from this vertex p' proceeds as p does. ∎

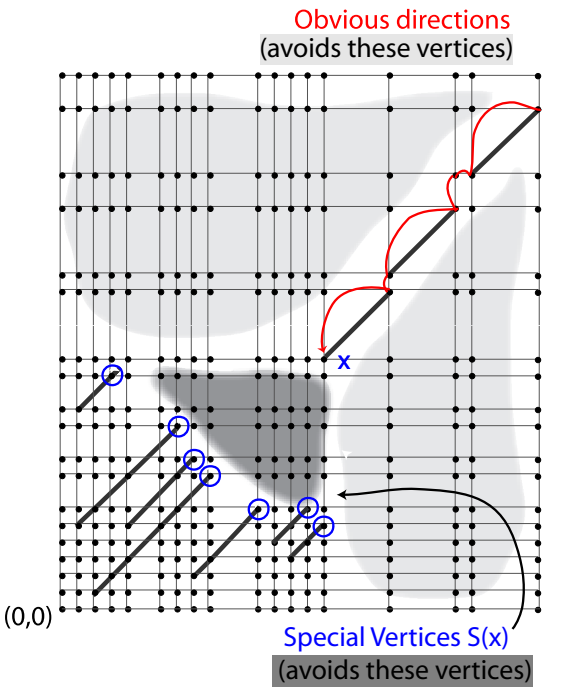

Figure 6. **Advantage of considering obvious and special vertices: Entries in the DP table corresponding to the lightly shaded vertices need not updated by the algorithm, since it moves in obvious directions (marked red). By considering only special vertices, there's no need to update entries corresponding to the darker-shaded vertices.**

COROLLARY 5.1 *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.*

Hence, as for relevant directions, the DP algorithm can be revised to immediately move to an obvious vertex, without needing to recurse over all extremal vertices (see Figure 6).

5.3. Special vertices

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

DEFINITION 5.3 We say that a vertex y is *special* with respect to a vertex x if:

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 5.4 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's 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 5.3 *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.*

Proof: Let p' be a maximal path from x to y. We need to show that its length is the same as that of p between these vertices. Let $y' \neq y, x$ be any vertex in p'. Since y' lies on a path from x to y, we have that $x > y' > y$. In other words, y' dominates x. Since $y \in S(X)$, by requirement (4) of definition 5.3 $y' \notin S(X)$. However, x dominates y' and there's a maximal path (p') from x to y'. Since $y' \notin S(X)$ it must be the case that it does not fulfill requirement (3) of definition 5.3. Hence $D(y') = D(x)$. The same argument shows that for any vertex y'' that p visits between x and y, $D(y'') = D(x)$.

$D(x)$ is a partition of $[k]$. Both p and p' follow only these directions when moving from x to y. In particular, each direction in $D(x)$ is followed the same number of times in both these paths. This implies that the two paths 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 5.3 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 restrict the search space of the DP algorithm further. When computing the shortest path from a vertex x, rather than considering the relevant extremal vertices, it is enough to consider those which are special. 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 5.5 Let $x, y \in \mathbb{G}_1(M)$ be such that y is special with respect to x. Let I, I' be maximal directions at x. We say that y is a *break point 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 5.1 *Let y be a special vertex with respect to x. Then y is one of the types in definition 5.5. Furthermore, if x is an extremal vertex then so is y.*

The proof is somewhat involved, and appears in the appendix.

COROLLARY 5.2 *All special vertices are extremal vertices.*

This leads to the version of the DP algorithm given in Figure 7. By Corollary 5.2, it can actually be run on $\mathbb{G}_2$ rather than $\mathbb{G}_1$ (See Figure 6 for an illustration of how considering special vertices saves table updates.)

We defer a detailed analysis of the running time to the full version, and mention that it can be bounded by $O(\sum_{x \in S} d(x) \cdot k \cdot l)$, where k is the number of sequences, l the number of segments per sequence, and $d(x)$ is 1 if there's an obvious direction at x, and $|D(x)|^2$ otherwise. In other words, it is linear in the number of segments and in 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.

FindOptimalPath(x) (version 3)

1. If $x = \vec{0}$ return $\vec{0}$.
2. If $\exists y$ which is obvious w.r.t. x
   - 2.1 If p_y is undefined, compute $p_y = \text{FindOptimalPath}(y)$
   - 2.2 Return the path x, p_y
3. For all $y = x - c \cdot e_I$ that is special with respect to x
   - 3.1 If p_y is undefined, compute $p_y = \text{FindOptimalPath}(y)$
   - 3.2 $d_y = c \cdot s(x, x - e_I)$
4. $y^* = \arg\max s(p_y) + d_y$
5. Return the path x, p_{y^*}.

Figure 7. A DP algorithm for the transitive case. In step 2, if possible, we choose a vertex y for which p_y was already computed.

References

[1] 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.

[2] Bairoch, A. and Apweiler, R. The SWISS-PROT protein sequence data bank and its supplement TrEMBL. *Nucleic Acids Res.*, 25(1):31-6, 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] D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evolution*, (25):351–360, 1987.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability–A Guide to the Theory of Np-completeness*. Freeman, San Francisco, 1979.

[10] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology.* Cambridge University Press, 1997.

[11] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci.*, 89: 10915–10919, 1992.

[12] D. G. Higgins and P. M. Sharp. Clustal - a package for performing multiple sequence alignment on a microcomputer. *Gene*, 73(1):237–244, 1988.

[13] T. Jiang and L. Wang. On the complexity of multiple sequence alignment. *J. Comp. Biol.*, 1(4):337–48, 1994.

[14] W. Just. Computational complexity of multiple sequence alignment with sp-score. *J. Comput. Biol.*, 8(6):615–623, 2001.

[15] 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.

[16] T. Lassmann and E. L. L. Sonnhammer. Quality assessment of multiple alignment programs. *Febs Letters*, 529(1):126–130, 2002.

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

[18] HP. Lenhof, B. Morgenstern, and K. Reinert. An Exact Solution for the Segment-to-segment Multiple Sequence Alignment Problem *Bioinformatics*, 15(3):203–210, 1999.

[19] B. Manthey. Non-approximability of weighted multiple sequence alignment. *Theor. Comput. Sci.*, 296(1):179–192, 2003.

[20] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *J. Comput. Sys. Sci.*, 20(1):18–31, 1980.

[21] B. Morgenstern. Dialign 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, 15(3):211–218, 1999.

[22] 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.

[23] T. Morgenstern, B. Dress. and A. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Nat. Acad. Sci.*, 93(22):12098–12103, 1996.

[24] M. Murata, J. S. Richardson, and J. L. Sussman. Simultaneous comparison of 3 protein sequences. *Proc. Nat. Acad. Sci.*, 82(10):3073–3077, 1985.

[25] G. Myers and W. Miller. Chaining multiple-alignment fragments in sub-quadratic time. *Proc. 6th Annual ACM-SIAM Sympos. Discrete Algo.*, 38–47, 1995.

[26] 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.

[27] C. Notredame. Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, 3(1):131–144, 2002.

[28] 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.

[29] 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.

[30] 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.

[31] T. F. Smith and M. S. Waterman. Comparison of biosequences. *Adv. Applied Math.*, 2(4), 482–489, 1981.

[32] 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.

[33] 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.

[34] W. J. Wilbur and D. J. Lipman. The context dependent comparison of biological sequences. *SIAM J. Applied Math.*, 44(3):557–567, 1984.

Appendix A: Proof of Theorem 3.1

In proving Theorem 3.1 it will be useful to consider $H(M)$, the clique hypergraph of M. Formally, the vertex set of $H(M)$ is that of M. $\{(i,z_i)\}_{i\in I}$ is a hyper-edge in $H(M)$ if for all $i,j\in I$ there's an edge $((i,z_i),(j,z_j))\in E(M)$. Moreover, the hyper-edges have multiplicity. A hyper-edge $\{(i,z_i)\}_{i\in I}$ in $H(M)$ has multiplicity equal to the length of the z_ith segment on sequence I, for some $i\in I$. Note that by definition all matched segments are of equal length, so this definition does not depend on the choice of i.

Alternatively, the hyper-edges of $H(M)$ can be thought of as corresponding to the edges of $\mathbb{G}_1(M)$. Fix $x\in V(\mathbb{G}_1(M))$ and $\emptyset\neq I\subset[k]$. For $i\in I$, let z_i be the segment on sequence i in which position x_i lies. Observe that by the definition above, $(x,x-e_I)$ is an edge in $\mathbb{G}_1(M)$ iff $\{(i,z_i)\}_{i\in I}$ is a hyper-edge in $H(M)$. Hence, we define the score of a hyper-edge in $H(M)$ as that of the edge in $\mathbb{G}_1(M)$ corresponding to it. Note that although several edges in $\mathbb{G}_1(M)$ may correspond to the same hyper-edge, the score is well defined, since in the MSAS problem it depends only on the segments.

A key observation is that in the MSAS problem we are looking for a set of "monotone" edges in $H(M)$:

DEFINITION 5.6 A sequence of vectors $v^1,\ldots,v^r\in([l_1]\cup\{-\})\times\ldots\times([l_k]\cup\{-\})$ is *monotone*, if for each $i\in[k]$, the numbers in the sequence $v_i^1,\ldots,v_i^r$ are monotonic non-increasing.
Associate with each hyper-edge $\{(i,z_i)\}_{i\in I}$ in $H(M)$, a vector $v\in([l_1]\cup\{-\})\times\ldots\times([l_k]\cup\{-\})$, by taking $v_i=z_i$ for $i\in I$, and $v_i=$ "$-$" otherwise.
A sequence of hyper-edges is *monotone* if the sequence of associated vectors is monotone.

Let p be a path from $(n_1,\ldots,n_k)$ to $(0,\ldots,0)$ in $\mathbb{G}_1(M)$. It is not hard to verify that the sequence of hyper-edges associated with the edges of p (in order) is monotone. We denote by $H(p)$ the hypergraph with vertex set of $H(M)$, and hyper-edges corresponding to the edges p, with multiplicity equal to the number of times they appear in p.

LEMMA 5.4 *Let p be a path from $(n_1,\ldots,n_k)$ to $(0,\ldots,0)$ in $\mathbb{G}_1(M)$, and $f_1,\ldots,f_t\in E(H(p))$ be edges of size >1. Suppose that for $i=1,\ldots,t-1$, there are distinct segments $s_i\in f_i, s_i'\in f_{i+1}$, from sequence j_i, with $s_i<s_i'$. Suppose also that there are distinct segments $s\in f_t, s'\in f_1$ from sequence j. Then $s'<s$.*

Proof: Let $v^1,\ldots,v^t$ be the vectors in $([l_1]\cup\{-\})\times\ldots\times([l_k]\cup\{-\})$ associated with $f_1,\ldots,f_t$, as above. Since there's a monotone ordering of the edges of $H(p)$, there's also one of $f_1,\ldots,f_t$. Since $v^i_{j_i}=s_i<s_i'=v^{i+1}_{j_i}$, this monotone ordering must be precisely $f_t,\ldots,f_1$. Now, the numbers in coordinate j must also be monotone. Hence $s'=v^1_j<v^t_j=s$. ∎

Proof: (Theorem 3.1) Let p be an optimal path, and denote $H=H(p)$. Observe that segments of different lengths are in different connected components of H. Consider such a component H', with segments of length $l=l(H')$. Let $0<i\leq l$ be an integer and (j_1,s_{j_1}) a vertex in H'. Suppose the ith position of segment s_{j_1} on sequence j_1 is matched by p to the ith position of segments $s_{j_2},\ldots,s_{j_r}$ on sequences $j_2,\ldots,j_r$ (resp.). So $\{(j_1,s_{j_1}),\ldots,(j_r,s_{j_r})\}$ is an edge in H'. Define $E_i(H')$ as the set of all such edges.
Consider all pairs of the form $(E_i(H'),e)$, where H' is a connected component of H, and $e\in E_i(H')$. It is not hard to see that there is a $1:1$ correspondence between these pairs and the edges of p. Define $s(E_i(H'))$ to be the sum of the scores of the edges in $E_i(H')$. It follows that $s(p)=\sum_{H'\subset H}\sum_{i=1}^{l(H')}s(E_i(H'))$ (the first summation is over the connected components of H).
Now, for each connected component H', let $E(H')$ be the subset among $\{E_i(H')\}$ with the highest score. Note that each segment appears in exactly one edge of $E=\bigcup E(H')$.
We claim that there is a path p' such that for each hyper-edge $\{(j_1,s_{j_1}),\ldots,(j_1,s_{j_r})\}\in E$, p' matches segments $s_{j_1},\ldots,s_{j_r}$ to each other, completely. In other words, p' uses only edges corresponding to those in E. This will prove the lemma, since by the choice of $E(H')$,

$$\begin{aligned}s(p')&=\sum_{H'\subset H}l(H')\cdot s(E(H'))\\&\geq\sum_{H'\subset H}\sum_{i=1}^{l(H')}s(E_i(H')),\end{aligned}$$

and so $s(p')\geq s(p)$.

It remains to show that such a path p' exists. Suppose we've constructed p' up to a vertex x. We need to prove that there is an allowed direction I, such that the segments where positions $\{x_i\}_{i\in I}$ lie, consist an edge in E.
Denote by s_i the segment in sequence i which con-

tains position x_i. Denote by $f_1, \ldots, f_r$ the edges of E in which these segments appear. We need to show that there's a $1 \leq t \leq r$ such that $f_t \subset \{(1, s_1), \ldots, (k, s_k)\}$. Denote by I_t the set of sequences whose segments appear in f_t. Clearly, if for some t, $|I_t| = 1$, then it is an allowed direction. Following it indeed matches (trivially) all the segments in f_t, and we are done. Assume then that $|f_t| > 1$ for $t = 1, \ldots, r$, and (for contradiction) that for each t there's a segment $s(f_t) \in f_t$, on sequence q_t, such that $s(f_t) \neq s_{q_t}$. For each such segment, denote by $s'(f_t)$ the segment on sequence q_t containing position x_{q_t}. $s(f_t) < s'(f_t)$, since otherwise we would have matched it already, or would have been free to do so now.

Define an ordered subset of $f_1, \ldots, f_r$ as follows. Start with $f'_1 = f_1$. Let f'_2 be the hyper-edge containing $s'(f_1)$. Continue this way, f'_{i+1} being the hyper-edge containing $s'(f'_i)$, until reaching a hyper-edge that was already chosen (w.l.o.g. assume it's f'_1). Denote by r' the number of these edges. For $i = 1, \ldots, r'$ we have that $s(f'_i) \in f'_i, s'(f'_i) \in f'_{i+1}$, and that $s(f'_i) < s'(f'_i)$ (where by f'_{r+1} we refer to f'_1). This is a contradiction to Lemma 5.4.

We have shown that p' matches segments in their entirety. It is easy to see that these matches can be reordered so that p' proceeds from each extremal vertex x to an extremal vertex y, such that $y \in X(x)$, as claimed. ▮

Appendix B: Proof of Theorem 5.1

Proof: Suppose that y is special with respect to x. Let $p = (x = p_1, \ldots, p_t = y)$ be a maximal path from x to y. Each $I \in D(x) \backslash D(y)$ is either not allowed at y or not maximal at y.

Case 1: Break point. Assume I is not allowed at y (in particular, $|I| > 1$). Suppose that from x, p proceeds in direction I to a point y' such $y'|_I = y|_I$ (by lemma 5.3 we may assume that it is so). Since, I is also not allowed at y', and y' dominates y, we conclude that $y' = y$. Furthermore, there are $i, j \in I$ such that $\{i, j\}$ is an allowed direction at every vertex in p except y. This means that there is a match $m = (i, j, y_i, y_j)$, and hence y_i and y_j are exit points. We conclude that if x was an extremal point, then so is y – for $i \in I$, y_i is an exit point; for $i \notin I$, $y_i = x_i$.

Case 2: Junctions. Assume I is not maximal at y. Let $i' \notin I$ be such that $I \cup \{i'\}$ is an allowed direction at y but not at x. In particular, $x|_{I \cup \{i'\}} \neq y|_{I \cup \{i'\}}$. Let I' be such that $i' \in I' \in D(x)$. Consider a maximal path that proceeds from x in direction I up to a point x' such that $x'|_I = y|_I$, and then in direction I' to a point y' such that $y'|_{I'} = y|_{I'}$. As $y'|_I = x'|_I$, we have $y'|_{I \cup I'} = y|_{I \cup I'}$. Thus $I \cup \{i'\}$ is an allowed direction at y'. But y' dominates y, thus, since y is special with respect to x, $y' = y$.

Let $J \in D(y)$ be such that $I \cup \{i'\} \subseteq J$. For the second part of the lemma, we need again to consider two cases.

Case 2.1: Corner junction. Assume that a positive number of steps was taken in both directions, I and I'. It follows that all vertices passed from x to x', including x', are not in $S(x)$. Thus $D(x) = D(x')$. Since $y_I = x'_I$, I is allowed at y. Let x'' be such that $x''|_{I'} = y$ and $x''|_{[k] \backslash I'} = x$. The same argument shows that I' is allowed at y, and hence $I \cup I' \subset J$. Now consider the point $y + e_{I \cup I'}$. It dominates y, and can be reached from x by taking one step less in both directions. Thus, it must be that J is not allowed there. Hence, for all $i \in I$, and all $j \in J \backslash I$ there is a match of sequences j and i, whose entry point on sequence i is y_i and on sequence j is y_j. Similarly for I'. Since $x|_{[k] \backslash (I \cup I')} = y|_{[k] \backslash (I \cup I')}$, we conclude that if x is extremal, so is y.

Case 2.2: Straight junction. Assume that y can be reached from x by following only one direction, say I. Consider an $i \in I$. Since $I \cup \{i'\}$ is allowed at y, there is a an $m \in E(M)$ matching y_i with $y_{i'}$. However, $y_{i'} = x_{i'}$. Assuming that x is extremal, $y_{i'}$ is an extremal point. This holds for all $i \in I$, and since other coordinates are the same as in x we have that y is also extremal. ▮

Appendix C: Segment matching is NP-hard

It is worthwhile noting that in general the segment matching problem is NP-hard, even if we define the input to the segment matching problem to be the clique hypergraph. Note that in this case NP-hardness does not follow immediately from the equivalence we've shown between MSAS and segment matching, since the number of edges in the hypergraph may be exponential in the number of sequences.

We show that segment matching is NP-hard by a reduction from the NP-complete *Minimal Feedback Vertex Set* problem (cf. [9]). In light of Definition 5.6, the segment matching problem can be reformulated as follows. The input is a set of weighted vectors in $([l_1] \cup \{-\}) \times \ldots \times ([l_k] \cup \{-\})$ - those associated with the hyper-edges $H(M)$. The goal is to find a monotone sequence of maximal weight.

Construct the following directed graph over the vectors. Put an edge from u to v, if there's some coordinate i such that $u_i > v_i$. Let $v^1, \ldots, v^r$ be a monotone sequence, then in the graph, there are no edges $v^i \to v^j$ for $i < j$. Conversely, if $v^1, \ldots, v^r$ is a sequence of vectors such that in the graph, there are no edges $v^i \to v^j$ for $i < j$, then it is a monotone sequence.

Note that a set of vertices in the graph which contains a directed cycle can not be ordered to form a "monotone" sequence. On the other hand, any set of vertices which does not contain a directed cycle, when ordered in reverse topological order, forms a monotone sequence. Hence, our problem is to find a set of vertices in the graph with maximal weight, such that the induced subgraph on them contains no directed cycles. Equivalently, the problem is to find a subset of vertices of minimal weight, whose removal from the graph eliminates all directed cycle. This is exactly the minimal feedback vertex set problem.