Faster Algorithms for Optimal Multiple Sequence Alignment Based on Pairwise Comparisons

Yonatan Bilu, Pankaj K. Agarwal, and 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, many heuristics were suggested for the problem. We consider a version of the MSA problem where the goal is to find an optimal alignment in which matches are restricted to positions in predefined matching segments. We present several techniques for making the dynamic programming algorithm more efficient, while still finding an *optimal* solution under these restrictions. We prove that it suffices to find an optimal alignment of the predefined 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. Empirical study shows that, taken together, these observations lead to an improved running time over the basic dynamic programming algorithm by 4 to 12 orders of magnitude, while still obtaining an optimal solution. Under the additional assumption that matches between segments are transitive, we further improve the running time for finding the optimal solution by restricting the search space of the dynamic programming algorithm.

Index Terms—Multiple Sequence Alignment, algorithms, dynamic programming, shortest path.

◆

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 [15], [21], [33].

By extending the dynamic programming algorithm by Needleman and Wunsch [32] for pairwise sequence alignment (see also [26]), a dynamic programming (DP) algorithm can compute in $O(n^k)$ time an optimal alignment of k sequences for a wide range of scoring functions [30]. However, this is not plausible when k is large and, indeed, the MSA problem is known to be NP-hard for many natural scoring functions [3], [10], [18], [19], [25].

The intractability results shifted the focus on developing heuristics, including MACAW [35], DIALIGN [27], ClustalW [39], T-Coffee [34], MUSCLE [8], [9], ProbCons [6], SPEM [43], and POA [22]. 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. [22] argued that the *only* information in MSA is the aligned segments and their relative positions. Indeed, many methods (e.g., [27], [35], [34], [8]) 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 small in comparison with the potential $\Omega(n^k)$ running time of the basic DP algorithm and can even be reduced if good suboptimal alignments are used ([8], [9]). In progressive methods, this observation percolates to the order of adding the sequences to the alignment [13], [34], [17], [39], [5], [8]. Other methods assemble an alignment by combining segments in an order dictated by their similarity [35], [27]. The Carrillo-Lipman method restricts the full DP according to the pairwise similarities [4]. Lermen and Reinert [23] use the so-called A^* algorithm to restrict the full DP search by precomputing heuristic bounds on the score of the optimal alignment. Another expression of this observation is scoring, and then matching, of full segments rather than single residues [29], [27], [24], [34], [42]. Unfortunately, none of these methods guarantees finding an optimal alignment. See [21], [33] for recent results on MSA.

Researchers have also studied how to expedite the optimal (mostly pairwise) sequence-alignment algorithms by constructing only a part of the full DP table. Eppstein et al. [11], [12] modify the objective function for speedup. Landau et al. [20] devised an alignment algorithm that runs in subquadratic time by exploiting the compressibility of typical input sequences. Wilbur and Lipman [41], [42] designed a pairwise alignment algorithm that offers a trade-off between accuracy and running time by considering only matches between identical "fragments." Myers and Miller [31] and Morgenstern [28] designed efficient solutions for

---

- *Y. Bilu is with the Department of Molecular Genetics, Weizmann Institute of Science, 76100 Rehovot, Israel. E-mail: yonatan.bilu@weizmann.ac.il.*
- *P.K. Agarwal is with the Department of Computer Science, Levine Science Research Center D315, Box 90129, Duke University, Durham, NC 27708-0129. E-mail: pankaj@cs.duke.edu.*
- *R. Kolodny is with the Department of Biochemistry and Molecular Biophysics, Columbia University, 1130 St. Nicholas Ave., ICRB, Mail Box 200, New York, NY 10032. E-mail: rachel.kolodny@columbia.edu.*

Manuscript received 15 Feb. 2006; revised 19 June 2006; accepted 1 Aug. 2006; published online 31 Oct. 2006.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBBSI-0016-0206.

special cases of the segment matching problem. In particular, the case considered by Myers and Miller can be solved in polynomial time [31], while the general problem is NP-hard. Recently, Sze et al. [38] considered the problem of an MSA that preserves $k-1$ of the pairwise alignments among k sequences and showed an efficient solution to this variation.

Our work is motivated by the observation that the encoding sequences used in the NP-hardness proof are not representative of protein and DNA sequences abundant in nature and the alignments in these proofs are not reminiscent of ones studied in practice. For example, the fact that MSA is NP-hard is shown by reduction from the max-cut problem [14]—any graph can be encoded as a set of sequences such that computing an optimal alignment of these sequences reveals the maximal cut in the graph. However, such encoding sequences are rarely biologically relevant and, hence, the fact that the problem is hard on these sequences may have limited bearing on its tractability for biological sequences.

We propose algorithms that find an *optimal* alignment but take advantage of the biological nature of the input sequences to expedite the running time. This is in contrast to most heuristics, which offer an efficient computation that hopefully leads to a good alignment, but (unlike our approach) cannot guarantee the optimality.

We define and study the *multiple sequence alignment from segments* (MSAS) problem, a generalization of MSA that accounts for assumptions regarding the pairwise characteristics of the optimal MSA. In MSAS, the input also includes a segmentation of the input 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 do not match or clearly match. Utilizing these assumptions, we develop a more efficient DP algorithm.

We prove that, under some reasonable assumptions on the scoring function, segments match in their entirety in an optimal alignment and, hence, it suffices to match segments, rather than individual positions. In particular, the complexity of the DP algorithm for MSAS and, indeed, *any* algorithm for MSAS, 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 the five human proteins GBAS, GBI1, GBT1, GB11, and GB12 requires 4.3×10^8 rather than 6.6×10^{12} table updates. We can make the algorithm even faster, while still guaranteeing the optimal solution, by further decoupling the subproblems 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. Nonetheless, we prove that, in general, the segment matching problem is NP-hard.

Next, 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 a (near) identity: 1) Segment matches have a transitive structure, i.e., if segment A matches segment B and B matches C, then A necessarily matches C. 2) The objective function is an alignment 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. Namely, only alignments that utilize the so-called "special vertices" in the DP graph need to be considered.

The contributions of this paper are, to a large extent, theoretical: We identify assumptions on the structure of the input sequences that make the problem of computing an optimal alignment tractable. Nonetheless, as noted above, the algorithm can be also of practical use when these assumptions plausibly hold, namely, when the sequences can be reliably partitioned into matching segments and the objective is to find how the segments align within the data set.

The paper is organized as follows: In Section 2, we define the MSA problem, cast it into a graph-theoretic framework, and describe 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 3.4. 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. Last, 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 $\mathcal{S} = \{\sigma_1, \ldots, \sigma_k\}$ 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 $\mathcal{S}$ is the sum of scores over all columns. The objective in the MSA problem is to find an alignment of $\mathcal{S}$ of optimal score. Without loss of generality, we assume the objective is maximizing the scoring function, but, importantly, f can be any scoring function over $(\Sigma \cup \{-\})^k$ (e.g., the commonly used sum-of-pairs, but also scores that are based on the whole sequence set, such as that in [6]). Other formulations of MSA which have been suggested (e.g., [22], [29]) are beyond the scope of this work.

We first define our notation: Let $I \subseteq [k]$, where $[k]$ denotes the set $\{1, \ldots, k\}$. We denote by $e_i \in \{0,1\}^k$ the

```
FindOptimalPath($x$)     (Version 0)
  1) If $x = \vec{0}$ return $(0, \vec{0})$
  2) For all $\emptyset \neq I \subseteq [k]$
      2.1 If $\mu_{x-e_I}$ is undefined,
          $(\mu_{x-e_I}, p_{x-e_I}) =$ FindOptimalPath($x - e_I$)
  3) $I^* = \arg\max_{I \subseteq [k]} \mu(x, x - e_I) + \mu_{x-e_I}$
  4) Return $(\mu(x, x - e_{I^*}) + \mu_{x-e_{I^*}}, p_{x-e_{I^*}} \circ x)$.
```

Fig. 1. Basic DP MSA algorithm. μ_x denotes the score of the path p_x.

vector that is zero in all coordinates except the ith, where it is 1, and $e_I = \sum_{i \in I} e_i$.[1] 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 define 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 (from right to left) of the input sequences. 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 for sequences not in I (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 $\mu : E(\mathbb{G}_0) \rightarrow \mathbb{R}$ to be a *score function* over the edges of $\mathbb{G}_0$, based on the score function f over the columns of the alignment: μ assigns to an edge the value which f assigns to the corresponding column. We also extend μ to paths or sets of edges $E' \subseteq E(\mathbb{G}_0)$: $\mu(E') = \sum_{e \in E'} \mu(e)$. 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 (with the same score).

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 and serves as a stepping stone for the new algorithms we develop here. In Fig. 1, we sketch the routine $\mathrm{FindOptimalPath}(x)$, which computes the optimal path from vertex x to the origin, denoted p_x, by considering the optimal paths from all its neighbors that are closer to the origin (as in Dijkstra's algorithm). The optimal MSA is calculated by calling $\mathrm{FindOptimalPath}((n_1, \ldots, n_k))$. The optimal scores of subproblems that have been solved recursively are stored to avoid recomputing them later. The algorithm returns the score of the optimal path and a sequence of vertices realizing it. In practical implementations, it is enough to store, at each node, the edge leading to it in an optimal path, rather than the entire path. 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 the Multiple Sequence Alignment from Segments (MSAS) problem—a generalization of MSA that is more suitable for biologically meaningful alignments. We assume a preprocessing step, which partitions the sequences into segments, matches pairs of these segments, and assigns a score to each match. This information is represented by the so-called segment matching graph. We use the segment matching graph to construct a graph $\mathbb{G}_1$, a subgraph of $\mathbb{G}_0$, which includes only those edges that correspond to the letters in the matched pairs of the segment matching graph. The MSAS problem asks for computing an optimal path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1$. Clearly, MSAS is a generalization of the MSA problem: Keeping in the preprocessing step all the edges of $\mathbb{G}_0$ (by segmenting the sequences into letters) gives $\mathbb{G}_1 = \mathbb{G}_0$ and we get the original MSA problem. We make this formulation precise in Section 3.1. The main property of $\mathbb{G}_1$ is that there exists an optimal path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1$ in which segments are matched in their entirety. Using this observation, we describe an algorithm in Section 3.2 that compresses $\mathbb{G}_1$ into a smaller graph $\mathbb{G}_2$ whose vertices correspond to the segments of the input sequences computed in the preprocessing step so that an optimal path in $\mathbb{G}_2$ corresponds to an optimal path in $\mathbb{G}_1$. We prove the above property of $\mathbb{G}_1$ in Section 3.3. Next, we show that, for computing the optimal path at a vertex in $\mathbb{G}_2$, it suffices to consider a subset of directions—the so-called relevant directions (Section 3.4). Since the number of edges in the graph governs the running time of the dynamic programming algorithm, considering fewer edges improves the running time.

3.1 Preliminaries

Definition 1. *For a sequence σ of length n, a* segmentation *of σ is a sequence of* breakpoints $0 = b_0 < b_1 < \ldots < b_l = n$. *The interval $[b_{i-1} + 1, b_i]$ is called the ith segment or* segment i *of σ. The breakpoint b_i is called the* entry point *into segment i (for $i = 1, \ldots, l$) and the* exit point *from segment $i + 1$ (for $i = 0, \ldots, l - 1$).*

Suppose we have a segmentation of the sequences in $\mathcal{S}$. Let l_j denote the number of segments in sequence σ_j.

Definition 2. *A* segment matching graph *(SMG), $\mathbb{M}$, over the given segmentation of $\mathcal{S}$ is an undirected weighted graph over vertex set $\{(j, i) : j \in [k], i \in [l_j]\}$. Every vertex represents a segment of an input sequence. Its edges are of two forms: 1) For every vertex (j, i), there is an edge $((j, i), (j, i))$, i.e., a self-loop, and 2) edges of the form $((j_1, i_1), (j_2, i_2))$, where $j_1 \neq j_2$ and the segment i_1 of sequence σ_{j_1} has the same length as the segment i_2 of σ_{j_2}; the edges of the type 2 form a k-partite graph. The edges of $\mathbb{M}$ have a scoring function $f : E(\mathbb{M}) \rightarrow \mathbb{R}$.*

An edge $e = ((j_1, i_1), (j_2, i_2))$ in the SMG signifies a match between segment i_1 in sequence σ_{j_1} and segment i_2 in sequence σ_{j_2}. Let l be the (same) length of these segments,

---

[1] This is a sum of vectors, i.e., the sum is component-wise.

and x_1 and x_2 their exit points on sequences σ_{j_1} and σ_{j_2}, respectively; then, for $t = 1, \dots, l$, the edge e implies that we allow a *match* between position $x_1 + t$ of sequence σ_{j_1} and position $x_2 + t$ of sequence σ_{j_2}.

The input to the MSAS problem is a set of segmented sequences, and a list of pairs of matching segments along with their scores, described by an SMG $\mathbb{M}$. The objective is still finding the highest scoring sequence alignment, but with the following two constraints:

C1. Two sequence positions may be aligned together only if they appear in matching segments and in the same relative position therein.

C2. The score of a multiple match depends only on the scores 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 restricts the algorithm to them. Equivalently, letters inside segments that cannot possibly match are not specified in the SMG and are thus disallowed. Furthermore, the algorithm assigns a "confidence level" (or 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 are nonoverlapping. In practice, we derive the segments from aligned portions of two sequences, which may overlap. This is resolved by splitting the overlapping segments to smaller nonoverlapping ones, as we discuss in Section 4.

Formally, given a set of segmented sequences and an SMG $\mathbb{M}$ over these segmented sequences, we define a graph $\mathbb{G}_1(\mathbb{M})$, a subgraph of $\mathbb{G}_0(\mathbb{M})$, as follows: The vertices of $\mathbb{G}_1(\mathbb{M})$ are the same as those of $\mathbb{G}_0$; an edge $(x, x - e_I)$ is in $\mathbb{G}_1(\mathbb{M})$ if and only if, for all $i, j \in I$, there is an edge $\gamma_{ij} \in E(\mathbb{M})$ such that position x_i on σ_i is matched to position x_j on σ_j in $\mathbb{M}$. In this case, we say that I is an *allowed direction* at x and that γ_{ij} is a match *defining* the edge $(x, x - e_I)$. The score of such an edge is a function of the corresponding edges in $\mathbb{M}$ (e.g., the sum-of-pairs scoring function, $\mu(x, x - e_I) = \sum_{i,j \in I, i<j} f(\gamma_{ij})$). 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 $\mathbb{M}$ have self-loops for all directions $i \in [k]$, $\{i\}$ is an allowed direction at all vertices x such that $x_i > 0$; this means that, as long as the beginning of the sequence has not been reached, we can always "align" a segment of one sequence against gaps in all the others.

The goal of the MSAS problem is to find a highest scoring path from $(n_1, \dots, n_k)$ to $(0, \dots, 0)$ in $\mathbb{G}_1$. Clearly, the algorithm depicted in Fig. 1 can be trivially modified to compute such a path. However, we improve the running using the following theorem, which is the main result of this section.

Theorem 1 (Segment Matching Theorem). *There is an optimal path from* $(n_1, \dots, n_k)$ *to* $(0, \dots, 0)$ *in* $\mathbb{G}_1$ *in which segments are either matched in their entirety or not matched at all.*

While the theorem is intuitively clear, the proof is deferred to Section 3.3 since it is somewhat technical and involved.

Definition 3. *A vertex* $x = (x_1, \dots, x_k)$ *in* $\mathbb{G}_1$ *is called a* breakpoint *if* x_i *is a breakpoint of sequence* σ_i *for all* $i \in [k]$. *We call a vertex* y *of* $\mathbb{G}_1$ *a* breakpoint with respect to x *if there exists an allowed direction* I *so that* y *is the first breakpoint reached when starting at* x *and repeatedly going in direction* I. *Let*

$$X(x) = \{y \in V(\mathbb{G}_1(\mathbb{M})) : y \text{ is a breakpoint w.r.t. } x\}.$$

Theorem 1 suggests that an optimal path from $(n_1, \dots, n_k)$ to $(0, \dots, 0)$ in $\mathbb{G}_1$ passes through a sequence of vertices $x_1, x_2, \dots, x_u$, where each x_i is a breakpoint and $x_{i+1} \in X(x_i)$. Using this observation, we next show how we can solve the MSAS problem faster by compressing the graph $\mathbb{G}_1$ to a smaller graph and computing an optimal path on this smaller graph.

3.2 Compressing $\mathbb{G}_1$ and Segment Matching

We define a graph $\mathbb{G}_2$, a "compressed" version of $\mathbb{G}_1$, whose vertices correspond to k-tuples of *segments* in the given segmentation of $\mathcal{S}$. That is, its vertex set is $[l_1] \cup \{0\} \times [l_2] \cup \{0\} \times \dots \times [l_k] \cup \{0\}$, where l_i is the number of segments in sequence σ_i. There is a directed edge from $z = (z_1, \dots, z_k)$ to $z - e_I$ in $\mathbb{G}_2$ if, for all $i, j \in I$, the z_ith segment of σ_i matches the z_jth segment of σ_j. Define $x = (x_1, \dots, x_k) \in V(\mathbb{G}_1)$ so that x_i is the entry point into the z_ith segment of σ_i. Let $x = (x_1, \dots, x_k)$ be the vertex in G_1 and let l be the length of the segments defining the edge $(z, z - e_I)$ (recall that two segments match only if they are of the same length). Then, $(z, z - e_I) \in E(\mathbb{G}_2)$ implies that $(x, x - e_I), (x - e_I, x - 2e_I), \dots, (x - (l-1)e_I, x - l \cdot e_I)$ are all edges in $\mathbb{G}_1$. In this sense, $(z, z - e_I)$ is a "compression" of the edges

$$\{(x, x - e_I), (x - e_I, x - 2e_I), \dots, (x - (l-1)e_I, x - l \cdot e_I)\}.$$

For example, in Fig. 2, the matching of the first segment of the upper sequence with the third segment of the lower sequence represents the fact that the six letters in these segments (NERMAL) match each other in order. The score of the edge $(z, z - e_I)$, denoted by $\mu(z, z - e_I)$, is the sum of the scores of all the edges in $\mathbb{G}_1$ that it represents. Since the score of these edges in $\mathbb{G}_1$ depends only on the matched segments, $\mu(z, z - e_I) = l \cdot \mu(x, x - e_I)$. Fig. 2 shows an example of two sequences, their SMG, and the graphs $\mathbb{G}_1, \mathbb{G}_2$.

The *segment matching* problem is to find a highest-scoring path from $(l_1, \dots, l_k)$ to $(0, \dots, 0)$ in $\mathbb{G}_2$. The segment matching theorem implies that such a path gives a highest-scoring path from $(n_1, \dots, n_k)$ to $(0, \dots, 0)$ in $\mathbb{G}_1$.

Fig. 3 sketches the revision of the basic DP algorithm to compute an optimal path in $\mathbb{G}_2$. The running time of the algorithm is $\Theta(2^k \prod_{j=1}^{k} l_j)$. Hence, if the sequences in $\mathcal{S}$ are long but consist of a small number of segments, DP may be appropriate for solving the segment matching problem but not for solving the MSA problem.

3.3 Proof of the Segment Matching Theorem

We prove the segment matching theorem by showing that, if the positions in a segment s in a given alignment are

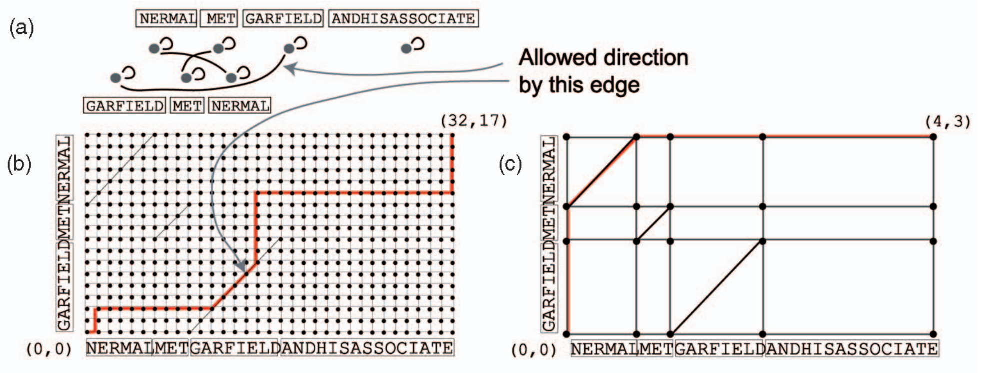

(a)
NERMAL MET GARFIELD ANDHISASSOCIATE
GARFIELD MET NERMAL
Allowed direction by this edge
(b)
(32,17)
GARFIELD MET NERMAL
(0,0)
NERMAL MET GARFIELD ANDHISASSOCIATE
(c)
(4,3)
GARFIELD MET NERMAL
(0,0)
NERMAL MET GARFIELD ANDHISASSOCIATE

Fig. 2. Example of an SMG for two sequences: (a) shows the sequences, their partitioning, and the SMG where each segment corresponds to a (gray) node. (b) shows $\mathbb{G}_1$. Unlike $\mathbb{G}_0$, which has all diagonals, the diagonals in $\mathbb{G}_1$ are defined by the SMG. An allowed path in $\mathbb{G}_1$ is also shown. (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 toward the origin.

matched to positions in several other segments, $s_1, \ldots, s_t$ (in other sequences), we can replace this alignment by another without decreasing its score, where, in the new alignment, all positions in s are either matched to the same segment (from among $s_1, \ldots, s_t$) or are not matched at all. For example, for the alignment shown in Fig. 4b, we show that one of the options in Fig. 4c does not reduce the score. Intuitively, we would like s to match in its entirety to one of the segments from $s_1, \ldots, s_t$ with the highest score. However, there is a caveat here: Suppose the best match of s is to s_i. Matching s to s_i might prevent matching s_i to some other segment, which, in total, improves the overall score of the alignment. Thus, we have to use a global argument to prove the theorem (see Fig. 4d), which makes the proof involved.

We define the *clique hypergraph* of the SMG $\mathbb{M}$, denoted by $\mathbb{H}(\mathbb{M})$: The vertex set of $\mathbb{H}(\mathbb{M})$ is $V(\mathbb{M})$. The hyperedges are defined by the cliques in $\mathbb{M}$. That is, $\{(i, z_i)\}_{i \in I}$ is a hyperedge if, for all $i, j \in I$, there is an edge $((i, z_i), (j, z_j)) \in E(\mathbb{M})$. If $(x, x - e_I)$ is an edge in $\mathbb{G}_1$, then, by construction, there is a unique hyperedge in $\mathbb{H}(\mathbb{M})$ corresponding to this edge; many edges of $\mathbb{G}_1$ might map to the same hyperedge of $\mathbb{H}(\mathbb{M})$.

The *score* of a hyperedge is the same as that of the corresponding edge in $\mathbb{G}_1(\mathbb{M})$. Observe that this is well defined because, by C2, the score of an edge in $\mathbb{G}_1$ depends only on the match and not on the specific position therein. We associate with each hyperedge $\{(i, z_i)\}_{i \in I}$ in $\mathbb{H}(\mathbb{M})$ 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.

Definition 4. *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 sequence of entries* $v_i^1, \ldots, v_i^r$ *is monotonically nonincreasing (excluding those entries which are "$-$"). A sequence of hyperedges is* monotone *if the sequence of associated vectors is monotone.*

For example, the sequence of vectors $v^1 = (3, -, -); v^2 = (2, 2, -); v^3 = (-, -, 2); v^4 = (1, 1, 1)$ is monotone. They can be thought of as describing a multiple alignment of three

```
FINDOPTIMALPATH($x$)     (Version 1)
    1) If $x = \vec{0}$ return $(0, \vec{0})$.
    2) $\forall y = x - l \cdot e_I \in X(x)$
        2.1 If $\mu_y$ is undefined, $(\mu_y, p_y)$ = FINDOPTIMALPATH($y$)
        2.2 $d_y = l \cdot \mu(x, x - e_I)$
    3) $y^* = \arg\max \mu_y + d_y$
    4) Return $(\mu_{y^*} + d_{y^*}, p_{y^*} \circ x)$.
```

Fig. 3. Segment-based DP MSA algorithm.

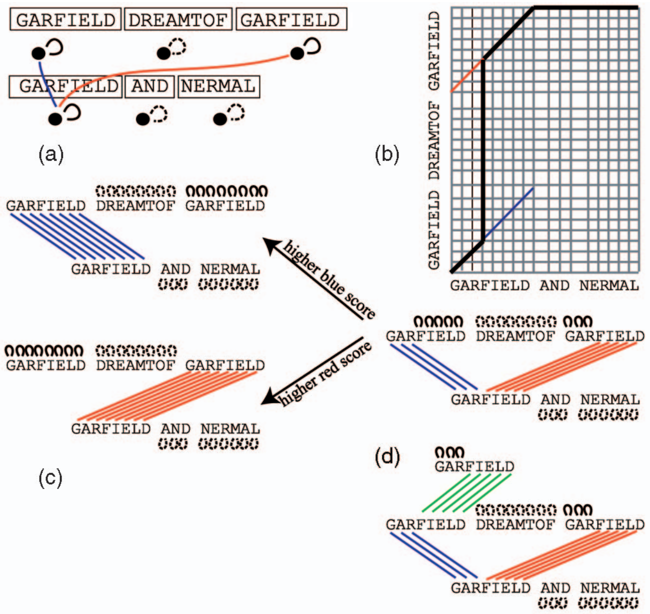


Fig. 4. Depiction of concepts used in the proof of Theorem 1. (a) The SMG of two sequences. (b) An optimal path for aligning these two sequences which does not match entire segments. (c) The hypergraph restricted to the path in (b) (right) and the two possible alignments which match whole segments (left). The choice between them depends on whether the score of the blue hyperedge is greater than that of the red hyperedge or not. Note that the parts in the alignment that are not in this connected component are unaffected. (d) Local considerations are not enough in modifying the path to match whole segments since, within a connected component, each change may effect other matches—choosing the blue edges implies losing the green edges.

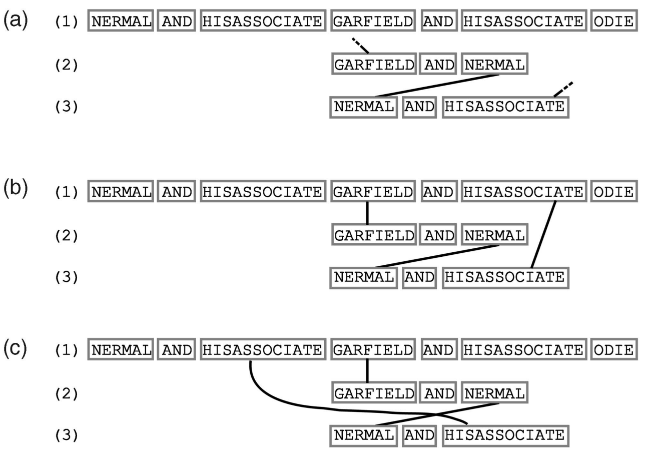

(a)
(1) NERMAL AND HISASSOCIATE GARFIELD AND HISASSOCIATE ODIE
(2) GARFIELD AND NERMAL
(3) NERMAL AND HISASSOCIATE
(b)
(1) NERMAL AND HISASSOCIATE GARFIELD AND HISASSOCIATE ODIE
(2) GARFIELD AND NERMAL
(3) NERMAL AND HISASSOCIATE
(c)
(1) NERMAL AND HISASSOCIATE GARFIELD AND HISASSOCIATE ODIE
(2) GARFIELD AND NERMAL
(3) NERMAL AND HISASSOCIATE

Fig. 5. The "cycle-structure" of the hypergraph induced by a path. A set of segments which constitute a cycle in the sense of (a), where the dashed edges are known to connect to segments on the same sequence, can only be of the structure (b). The match depicted in (c) is illegal since it violates the monotone property of the edges, i.e., the edges cross each other.

sequences. The first sequence has three segments and the other two have two. The second segments of the first and second sequences are aligned together and the first segments of all three sequences are aligned together.

Let p be a path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1(\mathbb{M})$. It is not hard to verify that the sequence of hyperedges associated with the edges of p (in order) is monotone. We denote by $\mathbb{H}_{|p}$ the hypergraph with vertex set of $\mathbb{H}(\mathbb{M})$ and hyperedges corresponding to the edges of p. We show in the next lemma that $\mathbb{H}_{|p}$ has a certain "cycle-structure," depicted in Fig. 5.

Lemma 1 (Cycle Structure Lemma). *Let p be a path from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1(\mathbb{M})$, and let $f_0, \ldots, f_{t-1}$ be a set of hyperedges in $E(\mathbb{H}_{|p})$ with $|f_i| > 1$ for all $i \le t$. Let $s_0, s_1, \ldots, s_{2t}, s_{2t-1}$ be a sequence of segments, where s_i is a segment of sequence σ_{j_i}, that form a cycle in the following sense:*

1. *For all $1 \le i < t$, $j_{2i-1} = j_{2i}$ and $s_{2i-1} < s_{2i}$, i.e., the segments s_{2i-1} and s_{2i} lie in the same sequence and the segment s_{2i-1} appears before s_{2i}.*
2. *s_{2t-1} and s_0 lie in the same sequence.*
3. *For all $0 \le i < t$, $(j_{2i}, s_{2i}), (j_{2i+1}, s_{2i+1}) \in f_i$, i.e., p matches segment s_{2i} with segment s_{2i+1}.*

Then, it must be the case that $s_0 < s_{2t-1}$.

Proof. For $0 \le i < t$, let $v^i \in ([l_1] \cup \{-\}) \times \ldots \times ([l_k] \cup \{-\})$ be the vector associated with f_i, as above. Since there is a monotone ordering of the edges of $\mathbb{H}_{|p}$, there is also one of $f_1, \ldots, f_t$. Since $v^{i-1}_{j_{2i-1}} = s_{2i-1} < s_{2i} = v^{i}_{j_{2i-1}}$ for all $1 \le i < t$, the monotone ordering of the f_is must be precisely $f_{t-1}, f_{t-2}, \ldots, f_0$. In particular, the numbers in coordinate j_0 must also be monotone. Hence, $s_0 = v^0_{j_0} < v^{t-1}_{j_0} = s_{2t-1}$. □

We are now ready to prove the theorem. Let p be an optimal path in $\mathbb{G}_1$ and let $\mathcal{C}$ denote the set of connected components of $\mathbb{H}_{|p}$. Observe that all segments in the same connected component of $\mathbb{H}_{|p}$ have the same length. Consider a component $H \in \mathcal{C}$ and let $l = l(H)$ be the length of the segments in H. Let $1 < i \le l$ be an integer and (j_1, s_{j_1}) a vertex in H. Suppose the ith position of segment s_{j_1} on sequence σ_{j_1} is matched by p to the ith position of segments $s_{j_2}, \ldots, s_{j_r}$ on sequences $\sigma_{j_2}, \ldots, \sigma_{j_r}$ (respectively). So, $\{(j_1, s_{j_1}), \ldots, (j_r, s_{j_r})\}$ is a hyperedge of H. Define $E_i(H)$ to be the set of all such hyperedges, which correspond to the matches of position i among the segments of H. Let $\mu(E_i(H)) = \sum_{\gamma \in E_i(H)} \mu(\gamma)$. Note that each hyperedge of $E_i(H)$ corresponds to a unique edge in p and vice versa. Therefore,

$$\mu(p) = \sum_{H \in \mathcal{C}} \sum_{i=1}^{l(H)} \mu(E_i(H)). \tag{1}$$

For each connected component H, let

$$\mathcal{E}(H) = \arg\max_{1 \le i \le l(H)} \mu(E_i(H)). \tag{2}$$

Set $\mathcal{E} = \bigcup_{H \in \mathcal{C}} \mathcal{E}(H)$. It can be checked that each segment of an input sequence appears in exactly one hyperedge of $\mathcal{E}$.

We claim that there is a path p' from $(n_1, \ldots, n_k)$ to $(0, \ldots, 0)$ in $\mathbb{G}_1$ such that, for each hyperedge $\{(j_1, s_{j_1}), \ldots, (j_1, s_{j_r})\} \in \mathcal{E}$, the path 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 $\mathcal{E}$. This will prove the theorem since

$$\mu(p') = \sum_{H \in \mathcal{C}} l(H) \cdot \mu(\mathcal{E}(H)) \ge \sum_{H \in \mathcal{C}} \sum_{i=1}^{l(H)} \mu(E_i(H)) = \mu(p).$$

It remains to show that such a path p' exists. Suppose we have constructed p' up to a vertex $x = (x_1, \ldots, x_k)$ and have chosen a subset $\mathcal{E}' \subseteq \mathcal{E}$ of hyperedges (corresponding to the edges of the path up to x). Without loss of generality assume that $x_i \ge 1$ for all $i \le k$. Denote by s_i the segment in sequence σ_i that contains position $x_i - 1$. Let $\mathcal{F} = \{f_1, \ldots, f_r\}$ be the subset of hyperedges in $\mathcal{E} \backslash \mathcal{E}'$ in which the segments $s_1, \ldots, s_k$ appear. Recall that there is a unique hyperedge of $\mathcal{E}$ in which each s_i appears.

Lemma 2. *There is a hyperedge $f \in \mathcal{F}$ so that $f \subseteq \{(1, s_1), \ldots, (k, s_k)\}$.*

Proof. If there exists a hyperedge $f \in F$ of size 1, i.e., of the form $\{(i, s_i)\}$, then the lemma is obviously true. So, assume that $|f| > 1$ for all $f \in \mathcal{F}$. For the sake of contradiction, suppose that each $f_t \in \mathcal{F}$ contains a pair $(q_t, s(f_t))$ so that $s(f_t) \ne s_{q_t}$, i.e., the position $x_{q_t} - 1$ does not lie in $s(f_t)$. Since all positions of σ_{q_t} up to x_{q_t} have already been matched using the hyperedges of $\mathcal{E}$ (by the choice of x) and the hyperedge $f_t \notin \mathcal{E}'$ has not been chosen so far, it follows that

$$s(f_t) < s_{q_t}. \tag{3}$$

We construct an ordered subsequence $f_{i_0}, f_{i_1}, \ldots$ of $\mathcal{F}$ as follows: Start with $f_{i_0} = f_1$. Suppose we have constructed $f_{i_0}, \ldots, f_{i_j}$. We then set $f_{i_{j+1}}$ to be the hyperedge of $\mathcal{F}$ that contains the pair $(q_{i_j}, s_{q_{i_j}})$. We stop the process when we reach a sequence that we have already chosen, say, f_{i_0}. Suppose we construct the

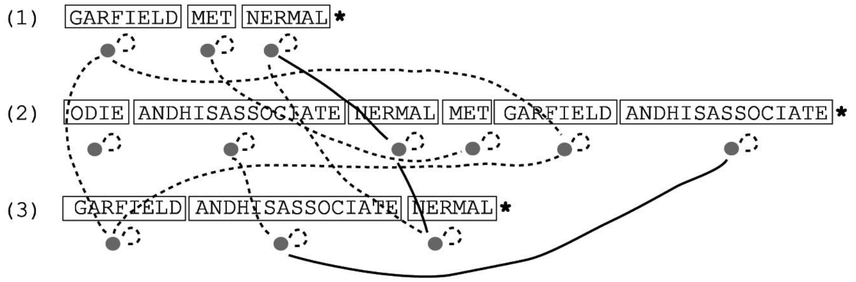

(1) GARFIELD MET NERMAL *
(2) ODIE ANDHISASSOCIATE NERMAL MET GARFIELD ANDHISASSOCIATE *
(3) GARFIELD ANDHISASSOCIATE NERMAL *

Fig. 6. An example SMG on three sequences, demonstrating relevant directions. We consider the vertex x which is the end of the three sequences: These positions are marked by * in the figure. The relevant matches for x in coordinates [1], [2], [3] are shown in bold and the other matches in this SMG are in dashed lines. The allowed directions for x are $\{[1], [2], [3], [1,3]\}$. The subset $S = [3, 2]$ is relevant at x. By Theorem 2, it suffices to consider only allowed directions that intersect S or, equivalently, there is no need to consider direction $[1]$.

subsequence $f_{i_0}, \dots, f_{i_{m-1}}$. We then set $\tilde{s}_0 = s_{q_{i_{m-1}}}$, $\tilde{s}_{2j-1} = s(f_{i_{j-1}})$, and $\tilde{s}_{2j} = s_{q_{i_{j-1}}}$ for $1 \leq j < m$, and $\tilde{s}_{2m-1} = s(f_{i_{m-1}})$. The hyperedges $f_{i_0}, \dots, f_{i_{m-1}}$ and the segments $\tilde{s}_0, \dots, \tilde{s}_{2m-1}$ satisfy conditions 1-3 of Lemma 1, but $\tilde{s}_0 > \tilde{s}_{2m-1}$ (by (3)), which contradicts the Cycle Structure Lemma (Lemma 1). Hence, there exists a hyperedge $f \in \mathcal{F}$ that is a subset of $\{(1, s_1), \dots (k, s_k)\}$. □

By the above lemma, let $f = \{(i_1, s_{i_1}), \dots, (i_r, s_{i_r})\} \subseteq \{(1, s_1), \dots, (k, s_k)\}$ be a hyperedge of $\mathcal{E} \backslash \mathcal{E}'$ and let l be the length of any segment in f. We choose f and we set $I = \{i_1, \dots, i_r\}$ and $x = x - l \cdot e_I$. By repeating this step until we reach $(0, \dots, 0)$, we have a path p' that matches segments in their entirety, as desired. This completes the proof of the theorem. It is easy to see that these matches can be reordered so that p' proceeds from each breakpoint vertex x to a breakpoint vertex y such that $y \in X(x)$.

3.4 Narrowing the Search Space: Relevant Directions

Consider an input to the MSAS problem that consists of two subsets of k sequences each and suppose that none of the segments in the first subset match any of those in the second subset. Naively applying the DP algorithm requires building the entire table and a 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 the sequences can be partitioned into loosely connected subsets. We make this notion explicit in this section. Again, we start with some definitions:

Definition 5. *Let x be a vertex in $\mathbb{G}_2(\mathbb{M})$. An edge $((i, y_i), (j, y_j))$ in the SMG is* relevant *for x at coordinate i if $x_i = y_i$ and $x_j > y_j$. A subset of indices $S \subseteq [k]$ is* relevant *at x if, for all $i \in S$, the edge $((i, y_i), (j, y_j))$ being relevant for x at coordinate i implies $j \in S$.*

Theorem 2. *Let p be an optimal path in $\mathbb{G}_2$ and x be a vertex on it. Let S be a subset of indices that is relevant 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 onward. We will define a different set of allowed directions that lead from x to y. Let $I_1, \dots, I_t$ 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 $j \notin I_1, \dots I_{t-1}$. Clearly, if $y_j + 1 = x_j$, then, again, $j \notin I_1, \dots I_{t-1}$. In other words, for all $h < t$, we have $I_h \cap I_t = \emptyset$. Define p' to follow directions $I_t, I_1, \dots, I_{t-1}$ 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 there is no need to look in *all* directions in the DP algorithm. Let S be a subset that is relevant 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 $I \cap S \neq \emptyset$. See Fig. 6 for an example of relevant matches and directions. This suggests the DP algorithm sketched in Fig. 7 (this time think of z as a vertex in $\mathbb{G}_2$). Note that there is no need to keep a table of size $|V(\mathbb{G}_2)|$ to implement this algorithm—the vertices that are actually visited by the algorithm can be kept in a hash table.

3.5 Segment Matching Is NP-Complete

The NP-completeness of the segment-matching problem does not immediately follow from the known results on the complexity of multiple sequence alignment. First, MSA is known to be NP-complete only for certain types of scoring functions, while the scoring function can be arbitrary in the segment matching problem. Second, the input to the problem is different—in the segment matching problem, the scoring function is part of the input and may be stated explicitly for each match. Hence, the question of whether there is an algorithm that solves the problem in time polynomial in the input size is different.

Nonetheless, we show that segment matching is NP-complete. We do so by a reduction from the *minimal feedback vertex set* problem, which is known to be NP-complete [14]. The input to the minimal feedback vertex set problem is a weighted, directed graph and the objective is to find a set of vertices of maximal total weight that span an acyclic subgraph.

It will be convenient to think of the segment matching problem in the terminology of Definition 4. That is, the input is a set of weighted vectors in $(\{1, \dots, l\} \cup \{-\})^k$ (representing the matches between k sequences, each with

```
FINDOPTIMALPATH($z$)    (Version 2)
1) If $z = \vec{0}$ return $(0, \vec{0})$.
2) Let $D$ be a minimal set of directions that intersects a subset that is relevant. It is computed as follows:
    2.1 For each $i \in [k]$ such that $z_i \neq 0$, compute the minimal subset $S_i$ such that $i \in S_i$, and $S_i$ is relevant at $z$.
    2.2 For each $i \in [k]$ let $D_i$ be the set of directions $I$ that are allowed at $z$, and $I \cap S_i \neq \emptyset$.
    2.3 Let $D$ be the smallest set among the $|D_i|$.
3) For all $\emptyset \neq I \in D$
    3.1 If $p_{z-e_I}$ is undefined, $(\mu_{z-e_I}, p_{z-e_I})$ = FINDOPTIMALPATH($z - e_I$)
4) $I^* = \arg\max_{J \in D} \mu(z, z - e_J) + \mu_{z-e_J}$
5) Return the optimal path $(\mu(z, z - e_{I^*}) + \mu_{z-e_{I^*}}, p_{z-e_{I^*}} \circ z)$.
```

Fig. 7. Version 2 of the MSA algorithm.

l segments) and the goal is to find a monotone nonincreasing sequence of maximal weight.

Theorem 3. *The segment matching problem is NP-complete.*

Proof. The fact that segment matching is in NP is trivial: Given a multiple alignment, we can compute its score in polynomial time and decide whether it is bigger than some value. So, it remains to prove that the problem is NP-hard.

Let G be a weighted directed graph on n vertices. Define n integer vectors of length $|E(G)|$, that is, the vectors are associated with the vertices of G and are indexed by the directed edges. With a slight abuse of notation, we shall denote a vertex and the vector associated with it by the same letter. Define the $e\text{th}$ coordinate of the vector v as 1 if e is an edge going into v, 0 if e is an edge going out of v, and "-" otherwise. Define the weight of a vector as that of the vertex in G to which it is associated.

The construction of these vectors can clearly be done in polynomial time, so the reduction from the minimal feedback vertex set is polynomial. To show that segment matching is NP-hard, we need to prove that a set of monotone vectors of maximal weight defines an acyclic subgraph of G of maximal weight. Thus, it suffices to show a bijection between monotone vector sequences and acyclic spanned subgraphs (while keeping the score).

Letting $U = \{u^1, \ldots, u^t\}$ be a sequence of vertices spanning an acyclic subgraph, we show that the associated vectors can be ordered as a monotone sequence. Assume U is enumerated in topological order, that is, there may be an edge from u^i to u^j only if $i < j$ (this can be so since the spanned graph is acyclic). Assume, for contradiction, that the sequence of vectors associated with U is not monotone. Then, there are two vectors, u^i and u^j, with $i < j$, such that $u^j > u^i$ does not hold. That is, there is some coordinate e such that $u^i_e \geq u^j_e$. Clearly, $u^i_e \neq u^j_e$ since there is only one vector v for which $v_e = 0$ (corresponding to the vertex from which e originates) and only one vector v' such that $v'_e = 1$. Hence, since all vectors are in $\{0, 1, -\}$, it must be that $u^i_e = 1$ and $u^j_e = 0$. But, this implies that e is an edge from u^j to u^i, in contradiction of the assumption that U is enumerated in topological order.

Now, let $U = \{u^1, \ldots, u^t\}$ be a monotone sequence of vectors. We show that the subgraph spanned by the associated vertices is acyclic. It will suffice to show that the associated vertices are enumerated in topological order. Indeed, assume this is not the case, that is, that, for some $i < j$, there is an edge e from u^j to u^i. Then, the $e\text{th}$ coordinate of the associated vectors is defined as $u^i_e = 1$, $u^j_e = 0$. This is a contradiction of the assumption that the U is a monotone sequence of vectors. □

4 IMPLEMENTING THE ALGORITHM

We have implemented Version 2 of the algorithm, described in Fig. 7. Using the 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 in Section 4.1 the preprocessing step that constructs the SMG and then discuss in Section 4.2 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.

Our implementation differs from the described algorithm in one point—in Fig. 7, for the sake of a simpler presentation, the entire optimal path is stored in each node. In practice, we only store the optimal edge leading to each node.

4.1 Generating a Segment Matching Graph (SMG)

Existing tools, such as BLAST [1] or DIALIGN [29], provide local alignments rather than the input format which we assumed previously. In order to restrict the problem to only MSAs which 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

TABLE 1
Number of Table Updates for Three Sets of Human Proteins

		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}	$91 \cdot 98 \cdot 99 \cdot 89$ $=7.86 \times 10^{7}$	7.2×10^{6}	$77 \cdot 84 \cdot 81 \cdot 74$ $=3.88 \times 10^{7}$	2×10^{6}
PTK6,PTK7, RET, SRMS, DDR1	2.4×10^{14}	$92 \cdot 96 \cdot 106 \cdot 88 \cdot 125$ $=1.03 \times 10^{10}$	2.81×10^{5}	$60 \cdot 53 \cdot 66 \cdot 57 \cdot 58$ $=3.74 \times 10^{6}$	2.98×10^{3}
GBAS, GBI1, GBT1, GB11, GB12	6.62×10^{12}	$148 \cdot 116 \cdot 113 \cdot 115 \cdot 120$ $= 2.68 \times 10^{10}$	2.7×10^{5}	$61 \cdot 72 \cdot 68 \cdot 71 \cdot 70$ $= 4.3 \times 10^{8}$	1.45×10^{5}

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/ungapped BLAST alignments. We see that, in all cases, the actual work is several orders of magnitude faster than the DP calculation.

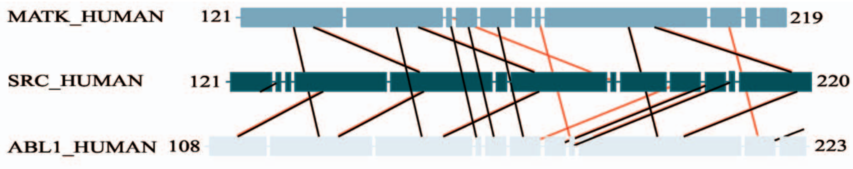


Fig. 8. An excerpt from the SMG and the alignment edges of three human proteins, MATK_HUMAN, SRC_HUMAN, and ABL1_HUMAN, starting from residues 121, 121, and 108, respectively. The edges of the SMG are shown in red and the subset chosen in the alignment in black (overlaying the red). The width of each node is proportional to the number of residues in the segment; the most thin nodes correspond to segments of one residue.

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. The segment 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).

4.2 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, 1,130, and 217, respectively. 2) PTK6, PTK7, RET, SRMS, DDR1 of lengths 451, 1,070, 1,114, 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 ungapped BLAST with an 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 an E-value of at most 10^{-2}. Importantly, this BLAST threshold is a parameter in practical uses of the algorithm, allowing a trade-off between sensitivity and running time. The lower the value, the greater the sensitivity of the algorithm as more potential matches are identified. However, numerous matches lead to an increased number of segments and, as a result, a longer running time.

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 (the 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 extreme case from 10^{14} to less than 3,000. In all cases, our algorithm finds an optimal MSAS solution in a reasonable number of steps.

We note that memory complexity is the appropriate measure in practice. When running the algorithm on a desktop computer with CPU speed of 3 GHz and 2 GB memory, it either terminated in less than 30 minutes or ran out of memory. The time complexity per table entry of our algorithm is roughly the same as that of the straightforward DP algorithm—the additional work of computing D (in Fig. 7, $O(k^2)$) is small in comparison with that required for going over all neighbors ($O(2^k)$), when $k > 4$.

Fig. 8 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 in the optimal alignment. We show only three proteins to allow a clear picture in the limited space 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 on our Web site (http://trantor.bioc.columbia.edu/~kolodny/MSA) 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). We also observed that there is a high occurrence of singletons in all of the examples we tried, which account for a large fraction of the work but are not necessarily as influential on the score. It might be possible to exploit this property in the following way: First, ignore singletons and match only longer segments with the MSAS algorithm. Then, once this "skeleton" of the alignment is established, run the algorithm independently for each region to match the singletons. This will improve the running time dramatically, but is not guaranteed to find an optimal solution. Hence, we leave this for future work.

For many practical applications of this algorithm, optimizing the preprocessing stage in which segments are identified is still required and lies beyond the scope of this work. Nonetheless, we also applied the algorithm (in the setting described above, but with a lax E-score cutoff of 100) to several test cases of the BaliBase benchmark [40].

Of the 19 sequences in reference 1 of medium length and at least 20 percent similarity, our algorithm was able to find an optimal solution based on the BLAST segmentation for 13 sequences in under 30 minutes. The BLAST segmentation, however, results in numerous short, unmatchable segments, which leads to a relatively low sum-of-pairs score for the alignment under common scoring schemes (see the supporting material on our Web site, http://trantor.bioc.columbia.edu/~kolodny/MSA/balibase/). If, however, gaps are not penalized—but given a score of 1 (which might be appropriate when only a small fraction of the sequences is expected to match many of the others, as in the examples discussed above)—then even this simplistic setting scores higher than CLUSTALW on 9 of these 13 test cases.

5 THE TRANSITIVE MSAS

In the previous section, we have seen how assumptions on the structure of the sought multiple alignment can be exploited to restrict the search space and improve running time. Namely, the assumption that the sequences are partitioned into segments and that only segments of equal length match reduces the problem to segment matching and allows consideration of only relevant directions.

In this section, we add two assumptions on the sought alignment, thus further restricting the structure of the optimal alignment, which, in turn, can be exploited to further speed the DP. Here, the relevant directions can be restricted to what we call "obvious directions," and only a subset of the vertices in $\mathbb{G}_2$, the so-called "special vertices," needs to be considered for an optimal path.

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 score function is such that we seek an alignment of minimal width or, equivalently, the shortest path from* $(n_1, \dots, n_k)$ *to* $(0, \dots, 0)$ *in* $\mathbb{G}_0$.

The assumption of transitivity may be too restrictive in many cases of aligning biological sequences. 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) analyzing this limited search space illuminates properties of the combinatorial structure of the problem.

Assumption 2 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 $(n_1, \dots, n_k)$ to $(0, \dots, 0)$ is of length $\sum_{i=1}^{k} n_i$. 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, 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 apply to $\mathbb{G}_2$ as well.

We call the problem of finding the highest scoring path from $(n_1, \dots, n_k)$ to $(0, \dots, 0)$ in $\mathbb{G}_1(\mathbb{M})$, under these two assumptions, 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 6. *An edge* $(x, x - e_I) \in E(\mathbb{G}_1(\mathbb{M}))$ *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* $\mathbb{G}_1(\mathbb{M})$ *is called a* maximal path *if it consists solely of maximal edges.*

Lemma 3. *There is an optimal path in* $\mathbb{G}_1(\mathbb{M})$ *that is also maximal.*

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

Suppose that $J' \supsetneq I$ 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 onward. Let $x = p_1, \dots, p_r = y$ 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 $t = 2, \dots, r$. 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 $I \supsetneq J$. It remains to show that if K is an allowed direction chosen at p_t, then it is 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 $t < r - 1$, the direction K is allowed at p'_t. For $t = r - 1$, the same argument shows that $K \backslash L$ is an

```
FINDMAXIMALDIRECTIONS($x$)
  1) Initialize $D = \emptyset$.
  2) For $i = 1, \ldots, k$, if coordinate $i$ is unmarked:
       a) Let $J \subseteq [k]$ be all coordinates $j$ such that position $x_j$ on sequence $j$ is matched to
          position $x_i$ on sequence $i$ (including $i$ itself).
       b) Add $J$ to $D$.
       c) Mark all coordinates in $J$
  3) Return $D$.
```

Fig. 9. Identifying maximal directions.

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 it is enough to consider only these when searching for an optimal alignment. That is, as a corollary of Lemma 3, the DP algorithm for the transitive MSA problem need not check *all* directions (or all of those that intersect a subset that is relevant), only maximal ones. This reduces the time complexity of the algorithm to $O(\prod l_i)$ times the time complexity of finding all maximal directions. Pseudocode for the latter is detailed in Fig. 9.

Step 2a can be realized in time linear in the number of retrieved coordinates by keeping a list of edges for each vertex in the SMG. The time complexity of this function is thus $O(k)$ and that of FINDOPTIMALPATH is $O(k \cdot \prod l_i)$. Recall that the bound for the general MSAS problem is $O(2^k \cdot \prod l_i)$.

5.2 Obvious Directions

The notion of "relevant directions" discussed in Section 3.4 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 7. *Let x be a vertex in $\mathbb{G}_1(\mathbb{M})$ and I a maximal direction at x. I is called an* obvious direction *(at x) if, for all $y \in \mathbb{G}_1(\mathbb{M})$ such that $y < x$ and $x|_I = y|_I$, I is a maximal direction at y. If $y = x - c \cdot e_I$ is a breakpoint 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 4. *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 from x to $x - e_I$.*

Proof. Since I is allowed at x, we have $x_i > 0$ for $i \in I$ 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 continues to $x - e_I$. It then proceeds in order along directions $I_1, \ldots, I_r$ (these directions are allowed since, being maximal, they do not intersect I). This leads to $x' - e_I$ and, from this vertex, p' proceeds as p does. Since p and p' have the same length and the same score, p' is optimal too. □

Corollary 1. *There is an optimal path p such that, if x is a breakpoint 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 trade-offs 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 breakpoint vertices (see Fig. 10).

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 8. *We say that a vertex y is* special *with respect to a vertex x if the following four conditions hold:*

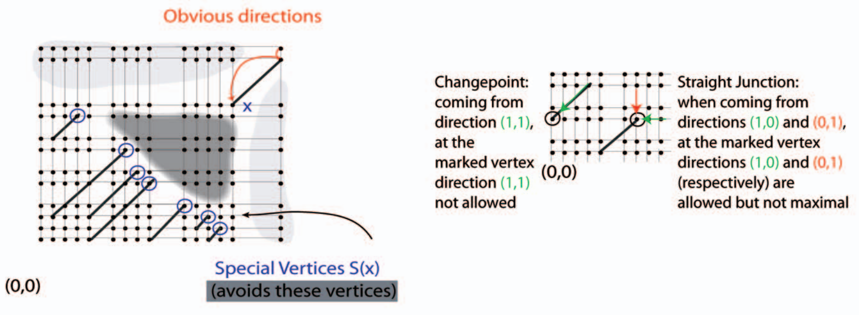
1. *x dominates y,*
2. *$D(x) \neq D(y)$,*
3. *there is a path from x to y consisting solely of maximal edges, and*
4. *no vertex y' satisfies all the above and dominates y.*

Denote by $S(x)$ the set of vertices that are special with respect to x.

We define the set of *special vertices* $S \subseteq \mathbb{G}_1(\mathbb{M})$ 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 breakpoint vertex to another, we can "leap" from one special vertex to another.

Definition 9. *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' moves in. We say that p and p' are* equivalent *if $p_0 = p'_0$, $p_r = p'_r$, and there is some permutation $\pi \in S_r$ such that $I_i = I'_{\pi(i)}$ for $i = 1, \ldots, r$.*

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



(a) (b)

Fig. 10. (a) demonstrates the advantage of considering obvious and special vertices: Entries in the DP table corresponding to the lightly shaded vertices do not need to be updated by the algorithm since it moves in obvious directions (marked by red leaps). By considering only special vertices, there is no need to update entries corresponding to the darker-shaded vertices. (b) shows examples of a changepoint and straight junction from Definition 10.

Lemma 5. *Let p be an optimal path. Let x be a vertex in p and let y be the first vertex in p that is 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 the length of this maximal path is the same as that of p between x and y. Let $y' \neq y, y' \neq x$ 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 y. Since $y \in S(X)$, by requirement 4 of Definition 8, we get $y' \notin S(x)$. However, x dominates y' and there is a maximal path (p') from x to y'. Since $y' \notin S(x)$, it must be the case that it does not fulfill requirement 2 of Definition 8, i.e., $D(y') = D(x)$. The same argument implies that any vertex y'' that p visits between x and y satisfies $D(y'') = D(x)$. Recall that $D(x)$ is a partition of $[k]$ and both p and p' follow only maximal directions from $D(x)$ 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 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 breakpoint vertices, it is enough to consider those which are also special. As we shall show soon, this is indeed a subset of the breakpoint vertices.

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

Definition 10. *Let $x, y \in \mathbb{G}_1(\mathbb{M})$ be such that y is special with respect to x, i.e., $y \in S(x)$. Let I and I' be maximal directions at x. We say that y is a: 1)* changepoint *of direction I if $y = x - c \cdot e_I$ for some natural c and I is not allowed at y; 2)* straight junction *of direction I if $y = x - c \cdot e_I$ for some $c \geq 0$ and I is allowed, but not maximal, at y. Finally, 3) y is a* corner junction *of directions I and I' if $y = x - c \cdot e_I - c' \cdot e_{I'}$ for some $c, c' \geq 0$ and I, I' are allowed, but not maximal, at y.*

Fig. 10 depicts a changepoint and a straight junction.

Theorem 4. *Let y be a special vertex with respect to x, i.e., $y \in S(x)$. Then, y is one of the types in Definition 10. Furthermore, if x is a breakpoint vertex, then so is y.*

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 is not maximal at y.

Case 1: Changepoint. Assume that I is not allowed at y (in particular, $|I| > 1$). Suppose that, from x, the path p proceeds in direction I to a point y' such that $y'|_I = y|_I$ (by Lemma 5, 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 breakpoint vertex, 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, so, since y is special with respect to x, we get $y' = y$.

Let $J \in D(y)$ be such that $I \cup \{i'\} \subseteq J$. For the second part of the lemma, we again 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 $x''|_{[k] \setminus I'} = x$. 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

```
FindOptimalPath($x$)   (version 3)
  1) If $x = \vec{0}$ return (0, $\vec{0}$).
  2) If $\exists y$ which is obvious w.r.t. $x$
      2.1 If $\mu_y$ is undefined, compute and store $(\mu_y, p_y)$ = FindOptimalPath($y$)
      2.2 Return the path $(\mu_y + \mu(x, y), p_y \circ x)$
  3) For all $y = x - c \cdot e_I$ that is special with respect to $x$
      3.1 If $\mu_y$ is undefined, compute and store $(\mu_y, p_y)$ = FindOptimalPath($y$)
      3.2 $d_y = c \cdot \mu(x, x - e_I)$
  4) For all $y = x - c \cdot e_I - c' \cdot e_{I'}$ that is special with respect to $x$
      4.1 If $\mu_y$ is undefined, compute and store $(\mu_y, p_y)$ = FindOptimalPath($y$)
      4.2 $d_y = c \cdot \mu(x, x - e_I) + c' \cdot \mu(x, x - e_{I'})$
  5) $y^* = \arg\max \mu_y + d_y$ (the maximum is taken over all $y$ which are special w.r.t. $x$).
  6) Return the optimal path $(\mu_{y^*} + d_{y^*}, p_{y^*} \circ x)$.
```

Fig. 11. A DP algorithm for the transitive case. In Step 2, if possible, we choose a vertex y for which p_y was already computed.

```
FindObviousVertex($x, I$)
  1) For all $I \in D(x)$:
      a) Let $i$ be the first element in $I$.
      b) If for all $j \in [k] \backslash I$, the least position on sequence $j$ that matches position $x_i$ in sequence
         $i$ is bigger than $x_j$, return $y$, the first breakpoint vertex after $x$ in direction $I$.
  2) Return $\emptyset$.
```

Fig. 12. Finding obvious neighbors.

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 and similarly for I'. Since $x|_{[k]\backslash(I \cup I')} = y|_{[k]\backslash(I \cup I')}$, we conclude that, if x is a breakpoint vertex, 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 exists $m \in E(\mathbb{M})$ matching y_i with $y_{i'}$. However, $y_{i'} = x_{i'}$. Assuming that x is a breakpoint vertex, $y_{i'}$ is too. This holds for all $i \in I$ and, since other coordinates are the same as in x, we have that y is also a breakpoint vertex.□

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

This leads to the version of the DP algorithm given in Fig. 11. By Corollary 2, it can run on $\mathbb{G}_2$ rather than $\mathbb{G}_1$ (see Fig. 10 for an illustration of how considering only special vertices saves table updates.)

As we describe below, the running time can be bounded by $O(|S| \cdot k^2 \cdot l)$ table updates, where k is the number of sequences and l is the maximal number of segments per sequence. Hence, it is fixed-parameter tractable (cf. [7]) in many of the parameters: It is linear in the number of segments and in the number of special vertices and squared in the number of sequences.

To see that the bound indeed holds, we need to describe how to find special vertices. Fig. 12 lists pseudocode for finding a vertex that is obvious with regard to an input vertex, Fig. 13 lists pseudocode for finding a changepoint or a straight junctions at a given direction, and Fig. 14 lists pseudocode for finding all corner junctions with regard to a pair of given directions.

To analyze the running time of the algorithm, observe that a recursive call to FindOptimalPath is only made for special vertices, so it suffices to bound the time complexity for finding all vertices which are special with regard to a given vertex. The analysis of the running time of these algorithms is straightforward:

1. In FindObviousVertex, going over all maximal directions at x requires $|D(x)|$ iterations and, at each one, at most k comparisons are made—of the least match in each sequence not in I. The total time complexity for this step is $|D(x)| \cdot k = O(k^2)$.
2. The loop in FindStraightVertex iterates at most l times. If breakpoints are stored in a linked list, each iteration, or finding the next breakpoint, takes constant time, a total of $O(l)$ for the execution of the loop. The function is called once for every maximal direction, for an overall contribution of $O(k \cdot l)$. Note that, in the first step of the algorithm, defining i as the first coordinate in I is arbitrary—by transitivity, choosing any element in I is equivalent.
3. In FindCornerVertex, computing z takes $O(l)$ time. To establish the number of iterations of the loop, observe that y is updated at most l times. Computing

```
FINDSTRAIGHTVERTEX($x, I$)
1) Let $i$ be the first coordinate in $I$.
2) Let $y = x$, $c = 0$.
3) While $D(x) = D(y)$:
    a) Retrieve $a$, the next breakpoint on sequence $i$ after $y_i$. Denote $g = y_i - a$.
    b) Update $y = y - g \cdot e_I, c = c + g$;
4) Return $(y, c)$.
```

Fig. 13. Finding changepoints and straight junctions (the same algorithm applies in both cases).

```
FINDCORNERVERTEX($x, I, I'$)
1) Initialize $Y = \emptyset$, $y = x$, $y' = 0$, $c = 0$.
2) Let $i, i'$ be the first coordinate in $I, I'$, respectively.
3) $z =$ FINDSTRAIGHTVERTEX($x, I$)
4) Loop:
    a) Retrieve $a$, the next entry point on sequence $i$ after $y_i$. Denote $g = y_i - a$.
    b) Update $y = y - e_I$, $c = c + g$.
    c) If $y < z$ return $Y$.
    d) Let $b$ be the maximal breakpoint on sequence $i'$ such that $y'_{i'} < b < x_{i'}$. If there is none,
       continue with the loop.
    e) Let $c' = x_{i'} - b$.
    f) Define $y' = y - c' \cdot e_{I'}$.
    g) Add $(y', c, c')$ to $Y$.
```

Fig. 14. Finding corner junctions.

b can done in constant time if breakpoints are kept in a linked list, for a total time complexity of $O(l)$. The function is called once for every pair of maximal directions, for an overall contribution of $O(k^2 \cdot l)$.

The time complexity of resolving each special vertex is therefore $O(k^2 \cdot l)$ and the time complexity of the entire algorithm is $O(|S| \cdot k^2 \cdot l)$, as claimed.

ACKNOWLEDGMENTS

The authors would like to thank Chris Lee and Nati Linial for enlightening discussions, Eylon Portugaly for his help in implementing the algorithm, and the referees for many helpful comments. Y. Bilu is supported by the Dewey David Stone Postdoctoral Fellowship and UniNet EC NEST consortium contract number 12990. P.K. Agarwal is supported by US National Science Foundation (NSF) grants CCR-00-86013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the US-Israel Binational Science Foundation. R. Kolodny is supported by NSF grant CCR-00-86013. Part of this work was done while Y. Bilu was at the School of Engineering and Computer Science, The Hebrew University of Jerusalem, and R. Kolodny was at the Department of Computer Science, Stanford University and visiting Duke University.

REFERENCES

[1] S. 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 Research,* vol. 25, pp. 3389-3402, 1997.

[2] A. Bairoch and R. Apweiler, "The SWISS-PROT Protein Sequence Data Bank and Its Supplement TrEMBL," *Nucleic Acids Research,* vol. 25, no. 1, pp. 31-36, 1997.

[3] P. Bonizzoni and G. Della Vedova, "The Complexity of Multiple Sequence Alignment with Sp-Score that Is a Metric," *Theoretical Computer Science,* vol. 259, nos. 1-2, pp. 63-79, 2001.

[4] H. Carrillo and D. Lipman, "The Multiple Sequence Alignment Problem in Biology," *SIAM J. Applied Math.,* vol. 48, no. 5, pp. 1073-1082, 1988.

[5] F. Corpet, "Multiple Sequence Alignment with Hierarchical-Clustering," *Nucleic Acids Research,* vol. 16, no. 22, pp. 10881-10890, 1988.

[6] C.B. Do, M.A. Mahabhashyam, M. Brudno, and S. Batzoglou, "ProbCons: Probabilistic Consistency-Based Multiple Sequence Alignment," *Genome Research,* vol. 15, no. 2, pp. 330-340, 2005.

[7] R.G. Downey and M. Fellows, *Parameterized Complexity.* Springer-Verlag, 1999.

[8] R.C. Edgar, "MUSCLE: Multiple Sequence Alignment with High Accuracy and High Throughput," *Nucleic Acids Research,* vol. 32, no. 5, pp. 1792-1797, 2004.

[9] R.C. Edgar, "MUSCLE: A Multiple Sequence Alignment Method with Reduced Time and Space Complexity," *BMC Bioinformatics,* vol. 5, p. 113, 2004.

[10] I. Elias, "Settling the Intractability of Multiple Alignment," *Proc. Ann. Int'l Symp. Algorithms and Computation (ISAAC),* pp. 352-363, 2003.

[11] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, “Sparse Dynamic-Programming: I. Linear Cost-Functions,” *J. ACM,* vol. 39, no. 3, pp. 519-545, 1992.

[12] D. Eppstein, Z. Galil, R. Giancarlo, and G.F. Italiano, “Sparse Dynamic-Programming: II. Convex and Concave Cost-Functions,” *J. ACM,* vol. 39, no. 3, pp. 546-567, 1992.

[13] D.F. Feng and R.F. Doolittle, “Progressive Sequence Alignment as a Prerequisite to Correct Phylogenetic Trees,” *J. Molecular Evolution,* vol. 25, pp. 351-360, 1987.

[14] M.R. Garey and D.S. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness.* Freeman, 1979.

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

[16] S. Henikoff and J.G. Henikoff, “Amino Acid Substitution Matrices from Protein Blocks,” *Proc. Nat'l Academy of Sciences,* vol. 89, pp. 10915-10919, 1992.

[17] D.G. Higgins and P.M. Sharp, “Clustal—A Package for Performing Multiple Sequence Alignment on a Microcomputer,” *Gene,* vol. 73, no. 1, pp. 237-244, 1988.

[18] T. Jiang and L. Wang, “On the Complexity of Multiple Sequence Alignment,” *J. Computer Biology,* vol. 1, no. 4, pp. 337-348, 1994.

[19] W. Just, “Computational Complexity of Multiple Sequence Alignment with Sp-Score,” *J. Computer Biology,* vol. 8, no. 6, pp. 615-623, 2001.

[20] G.M. Landau, M. Crochemore, and M. Ziv-Ukelson, “A Subquadratic Sequence Alignment Algorithm for Unrestricted Cost Matrices,” *Proc. 13th Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 679-688, 2002.

[21] T. Lassmann and E.L.L. Sonnhammer, “Quality Assessment of Multiple Alignment Programs,” *Febs Letters,* vol. 529, no. 1, pp. 126-130, 2002.

[22] C. Lee, C. Grasso, and M.F. Sharlow, “Multiple Sequence Alignment Using Partial Order Graphs,” *Bioinformatics,* vol. 18, no. 3, pp. 452-464, 2002.

[23] M. Lermen and K. Reinert, “The Practical Use of the A^* Algorithm for Exact Multiple Sequence Alignment,” *J. Computer Biology,* vol. 7, no. 5, pp. 655-672, 2000.

[24] H.P. Lenhof, B. Morgenstern, and K. Reinert, “An Exact Solution for the Segment-to-Segment Multiple Sequence Alignment Problem,” *Bioinformatics,* vol. 15, no. 3, pp. 203-210, 1999.

[25] B. Manthey, “Non-Approximability of Weighted Multiple Sequence Alignment,” *Theoretical Computer Science,* vol. 296, no. 1, pp. 179-192, 2003.

[26] W.J. Masek and M.S. Paterson, “A Faster Algorithm Computing String Edit Distances,” *J. Computer Systems Science,* vol. 20, no. 1, pp. 18-31, 1980.

[27] B. Morgenstern, “Dialign 2: Improvement of the Segment-to-Segment Approach to Multiple Sequence Alignment,” *Bioinformatics,* vol. 15, no. 3, pp. 211-218, 1999.

[28] B. Morgenstern, “A Simple and Space-Efficient Fragment-Chaining Algorithm for Alignment of DNA and Protein Sequences,” *Applied Math. Letters,* vol. 15, no. 1, pp. 11-16, 2002.

[29] B. Morgenstern, A. Dress, and T. Werner, “Multiple DNA and Protein Sequence Alignment Based on Segment-to-Segment Comparison,” *Proc. Nat'l Academy of Sciences,* vol. 93, no. 22, pp. 12098-12103, 1996.

[30] M. Murata, J.S. Richardson, and J.L. Sussman, “Simultaneous Comparison of 3 Protein Sequences,” *Proc. Nat'l Academy of Science,* vol. 82, no. 10, pp. 3073-3077, 1985.

[31] G. Myers and W. Miller, “Chaining Multiple-Alignment Fragments in Subquadratic Time,” *Proc. Sixth Ann. ACM-SIAM Symp. Discrete Algorithms,* pp. 38-47, 1995.

[32] 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. Molecular Biology,* vol. 48, pp. 443-453, 1970.

[33] C. Notredame, “Recent Progress in Multiple Sequence Alignment: A Survey,” *Pharmacogenomics,* vol. 3, no. 1, pp. 131-144, 2002.

[34] C. Notredame, D.G. Higgins, and J. Heringa, “T-Coffee: A Novel Method for Fast and Accurate Multiple Sequence Alignment,” *J. Molecular Biology,* vol. 302, no. 1, pp. 205-217, 2000.

[35] G.D. Schuler, S.F.F. Altschul, and D.J. Lipman, “A Workbench for Multiple Alignment Construction and Analysis,” *Proteins-Structure Function and Genetics,* vol. 9, no. 3, pp. 180-190, 1991.

[36] R.M. Schwartz and M.O. Dayhoff, “Matrices for Detecting Distant Relationships,” *Atlas of Protein Sequences and Structure,* pp. 353-358, 1979.

[37] T.F. Smith and M.S. Waterman, “Comparison of Biosequences,” *Advanced Applied Math.,* vol. 2, no. 4, pp. 482-489, 1981.

[38] S.-H. Sze, Y. Lu, and Q. Yang, “A Polynomial Time Solvable Formulation of Multiple Sequence Alignment,” *J. Computational Biology,* to appear.

[39] 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,* vol. 22, no. 22, pp. 4673-4680, 1994.

[40] J.D. Thompson, F. Plewniak, and O. Poch, “BAliBASE: A Benchmark Alignment Database for the Evaluation of Multiple Alignment Programs,” *Bioinformatics,* vol. 15, pp. 87-88, 1999.

[41] W.J. Wilbur and D.J. Lipman, “Rapid Similarity Searches of Nucleic-Acid and Protein Data Banks,” *Proc. Nat'l Academy of Sciences,* vol. 80, no. 3, pp. 726-730, 1983.

[42] W.J. Wilbur and D.J. Lipman, “The Context Dependent Comparison of Biological Sequences,” *SIAM J. Applied Math.,* vol. 44, no. 3, pp. 557-567, 1984.

[43] H. Zhou and Y. Zhou, “SPEM: Improving Multiple Sequence Alignment with Sequence Profiles and Predicted Secondary Structures,” *Bioinformatics,* vol. 21, no. 18, pp. 3615-3621, 2005.

Yonatan Bilu received the PhD degree in computer science in 2004 from the Hebrew University in Jerusalem. He is currently a postdoctoral fellow in the Depatment of Molecular Genetics at the Weizmann Institute of Science. His research interests include computational biology and systems biology.

Pankaj K. Agarwal received the PhD degree in computer science from the Courant Institute of Mathematical Sciences at New York University. He joined the Department of Computer Science of Duke University in 1989, where he is now the chair and a professor of computer science and mathematics. His research interests include geometric algorithms and data structures, computational molecular biology, spatial databases, global change, geographic information systems, sensor networks, and robotics. He has authored four books and more than 250 scholarly articles in various journals, edited volumes, and international conferences. He has received many awards, including National Young Investigator, Sloan Fellow, and ACM Fellow, and he serves on the editorial boards of a number of journals.

Rachel Kolodny received the PhD degree in computer science in 2004 from Stanford University. She is currently a computational associate in the HHMI (Howard Hughes Medical Institute) at the Department of Biochemistry and Molecular Biophysics at Columbia University. Her research interests are computational biology, focusing on the study of protein structure.