Chapter 1
Chaining Multiple-Alignment Fragments in Sub-Quadratic Time

Gene Myers* Webb Miller†

Abstract

We describe a multiple-sequence alignment algorithm for determining the highest-scoring alignment that can be obtained by chaining together non-overlapping subalignments selected from a given collection of such "fragments". For a given set of K sequences, a problem instance consists of a set of F precomputed fragments, an alignment score for each fragment, and a "gap" penalty function that assigns a cost for chaining two fragments together. When interpreted as a maximum weight path problem in a directed acyclic graph, it is computable in $\Theta(F^2)$ time. Here we instead interpret the problem as "K-dimensional sparse dynamic programming" and take advantage of its underlying geometric nature. Assuming $K < \log F$, our algorithm runs in time $O(F \log^K F)$ and space $O(KF \log^{K-1} F)$, making it the first sub-quadratic sparse dynamic programming algorithm for the case $K > 2$.

1 Introduction

Consider sequences $A^0, A^1, \ldots, A^{K-1}$, where $A^i = a^i_1 a^i_2 \ldots a^i_{N_i}$ has length N_i. Intuitively, a "fragment subalignment" is an alignment of a highly conserved region between the sequences that we assume has been precomputed by some other method. The literature from computational molecular biology contains dozens of papers describing methods to compute various kinds of fragment subalignments (e.g., [10, 7]). Our goal is to find an optimum subset of such a collection of fragments that can be linked together to form a longer alignment of maximum score.

Proceeding formally, let a *point* be a K-tuple $\vec{p} = (p_0, p_1, \ldots, p_{K-1})$ of integers, where $p_i \in [0, N_i]$ for all $i \in [0, K-1]$. The partial order $\vec{p} \leq \vec{q}$ orders point $\vec{p}$ before point $\vec{q} = (q_0, q_1, \ldots, q_{K-1})$ if and only if $p_i \leq q_i$ for all $i \in [0, K-1]$. For our purposes, a *fragment* can then be summarized as a triple $< f.\vec{beg}, f.\vec{end}, f.score >$ where $f.\vec{beg}$ and $f.\vec{end}$ are points satisfying $f.\vec{beg} \leq f.\vec{end}$, and $f.score$ is a real number. The two end-points specify the substring of each sequence that is aligned by the fragment f, i.e., $a^i_{f.beg_i+1} a^i_{f.beg_i+2} \ldots a^i_{f.end_i}$ for each i, and $f.score$ is the score attributed to the precomputed multi-alignment of f's substrings.

The other component of our problem is a real-valued function $gap\_cost(\vec{p}, \vec{q})$ that gives the penalty for linking a fragment that ends at the point $\vec{p}$ with a fragment that begins at the point $\vec{q}$. The partial order $e \leq f$ iff $e.\vec{end} \leq f.\vec{beg}$ on fragments asserts that the substrings aligned by e precede those of f in their respective sequences, so e and f can be linked together. A *chain* of fragments is a sequence $\{f_t\}_{t=1}^{L}$ such that $f_t \leq f_{t+1}$ for $t \in [1, L-1]$, and the *score* of the chain is:

$$\sum_{t=1}^{L} f_t.score \; - \; \sum_{t=1}^{L-1} gap\_cost(f_t.\vec{end}, f_{t+1}.\vec{beg})$$

The *fragment-chaining problem* is to determine a chain of maximum score, given a set of fragments and a *gap_cost* function. The straightforward method of fragment-chaining is, in essence, a special case of the classic optimum-path algorithm for directed acyclic graphs, which works for any definition of *gap_cost*. Let $score\_at(f)$ be defined as the maximum score of all chains that end with f and immediately observe the recurrence:

$$score\_at(f) = f.score + max\{0, \mu\}$$

where

$$\mu = max\{score\_at(e) - gap\_cost(e.\vec{end}, f.\vec{beg}) : e \leq f\}$$

A dynamic programming algorithm applies this recurrence in any topological order of the fragments with respect to the partial order $\leq$ on fragments. This simple algorithm requires time $O(F^2 G)$, where F is the number of fragments and computing *gap_cost* takes $O(G)$ time. Its space requirement is $O(KF)$, i.e., the space needed for the fragments. This algorithm has been proposed as a practical approach for aligning biological sequences, first for $K = 2$ by Wilbur and Lipman [13], then for $K > 2$ by Sobel and Martinez [11].

For $K = 2$ sequences, fragment-chaining algorithms that run in time $O(N_0 + N_1 + F \log F)$ or faster were

*Department of Computer Science, University of Arizona, Tucson, AZ 85721. Supported by grant LM04960 from the National Library of Medicine and by the Aspen Center for Physics.

†Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802. Supported by grant LM05110 from the National Library of Medicine.

found independently by Eppstein *et al.* [4] and by Myers and Huang [8]. Eppstein *et al.* observed that in the case of sequence comparison the coordinates of points are *integers* between 0 and $N = max(N_i)$, and so the running time can be reduced to $O(F \log\log F)$, using a data structure of Johnson [5]. However, Myers and Huang were considering the problem of comparing restriction maps, where the points have to be modeled as K-tuples of *real* numbers, a context in which the speedup does not apply. In order to make the central algorithm of practical value to biologists, Chao and Miller [2] extended it simultaneously in two directions. First, space consumption was reduced to $O(N)$, assuming that fragments can be generated at will and discarded as the algorithm proceeds. Second, the generalized algorithm delivers any desired number of highest-scoring, non-intersecting chains at a slight increase in asymptotic time complexity. Though these subquadratic fragment-chaining algorithms require $gap\_cost(\vec{p}, \vec{q})$ to satisfy certain restrictions, they are sufficiently general to appropriately handle biological data, and they have been used in a practical program for aligning very long DNA sequences [3].

For potential application to biology, it is natural to seek generalizations of these algorithms to $K > 2$ sequences. Unfortunately, the above earlier methods, which are based on the sweep-line paradigm from two-dimensional computational geometry, do not readily generalize in that direction. Zhang *et al.* [14] gave a practical algorithm based on kD-trees to compute optimum fragment chains for more than two sequences. However, no rigorous analysis of running time was provided, as is typical with kD-tree methods. Moreover, the paper left open the problem of reducing space requirement below $\Theta(KF)$.

This paper develops a fragment-chaining algorithm whose worst-case time breaks the $O(F^2)$ barrier. As with earlier time-efficient algorithms, this requires specifying restrictions on gap_cost, specifying a particular topological order for fragments, dividing the computation of $score\_at$ into cases and developing data structures that allow each case to be solved efficiently. In effect, for a given point $\vec{q}$ these "cases" partition the set of fragments e such that $e.\vec{end} \leq \vec{q}$ into classes E_π for $\pi \in [1, C_K]$. Each of these classes has associated data structures supporting a procedure $select\_best_\pi(\vec{q})$ that returns the maximum value of $score\_at(e) - gap\_cost(e.\vec{end}, \vec{q})$ taken over all $e \in E_\pi$. The recurrence for $score\_at$ then operationally becomes:

$$score\_at(f) = f.score + max\{0, \mu\}$$

where

$$\mu = max\{select\_best_\pi(f.\vec{beg}) : \pi \in [1, C_K]\}$$

If $select\_best_\pi(\vec{q})$ runs in time $O(T_{K,F})$ then, ignoring the time needed to maintain the data structures, the fragment-chaining algorithm runs in time $O(FC_KT_{K,F})$. In the algorithm that follows, there are $C_K = K!$ cases, each of which can be solved in time $T_{K,F} = (\log^K F)/K!$. Our task is thus to describe the partitioning of fragments into classes and for each class to design an efficient procedure for $select\_best_\pi$. The next section presents our results for $K = 2$, Section 3 sketches some geometric intuition in the case $K = 3$, and the general algorithm is described in Section 4.

2 The Case of K = 2 Sequences

The efficient algorithms discussed above all require that $gap\_cost(e.\vec{end}, f.\vec{beg})$ (i.e., the cost of placing fragments e and f consecutively within a longer alignment) be defined in such a way that no accounting is made for the particular sequence symbols that occur in the region between e and f. That is, the gap cost must be *symbol independent*. In the definition adopted for this paper, two parameters, ε and λ, are used to define gap_cost. Intuitively, $\varepsilon > 0$ is the cost of aligning two anonymous symbols, while $\lambda > 0$ is the cost of aligning an anonymous symbol with a gap position in the other sequence. We require that δ, defined as $2\lambda - \varepsilon$, satisfy $\delta \geq 0$; otherwise it would always be best to connect fragments entirely by gaps (i.e., insertions and deletions, but never replacements). Thus, the best way to connect two fragments is to perform substitutions until the entries in one of the sequences are exhausted, then insert the remaining elements.

The precise definition of $gap\_cost(\vec{p}, \vec{q})$ for $\vec{p} \leq \vec{q}$ is as follows. Define $\Delta_i = q_i - p_i$ for $i = 0$ and 1. Thus, Δ_0 and Δ_1 are the numbers of symbols in each sequence that would lie between a fragment ending at $\vec{p}$ and one beginning at $\vec{q}$. Note that $\vec{p}$ can be linked to $\vec{q}$ (i.e., $\vec{p} \leq \vec{q}$) if and only if $\Delta_0, \Delta_1 \geq 0$. If $\Delta_0 \geq \Delta_1$, then $gap\_cost(\vec{p}, \vec{q}) = \Delta_1\varepsilon + (\Delta_0 - \Delta_1)\lambda$. See Figure 1. Otherwise, $gap\_cost(\vec{p}, \vec{q}) = \Delta_0\varepsilon + (\Delta_1 - \Delta_0)\lambda$. These two cases, i.e., (1) $\Delta_0 \geq \Delta_1$ and (2) $\Delta_0 \leq \Delta_1$, constitute the aforementioned $2! = 2$ cases that will become $K!$ in the general case.

A point $\vec{p} = (s, t)$ is said to be on antidiagonal $antidiag(\vec{p}) = s + t$ and on diagonal $diag(\vec{p}) = s - t$. These two quantities provide an alternate, diagonal-based coordinate system for naming points. Specifically, the Cartesian point $\vec{p}$ can be described and written as the *antidiagonal point* $[x]_a$, where $x = diag(\vec{p})$ and $a = antidiag(\vec{p})$. Note that the mapping between these two frameworks is isomorphic; each Cartesian point maps to an antidiagonal point and vice versa.

Suppose $\vec{p} \equiv [x]_a$ is the end-point $e.\vec{end}$ of some fragment e. For points $\vec{q} \equiv [y]_b$ on a future antidiagonal

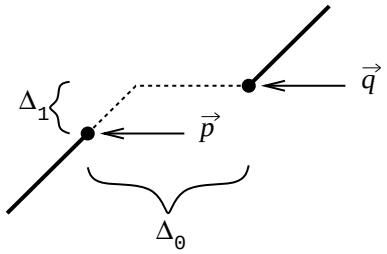

Figure 1: If $\Delta_0 \geq \Delta_1$, then $gap\_cost(\vec{p}, \vec{q}) = \Delta_1\varepsilon + (\Delta_0 - \Delta_1)\lambda$.

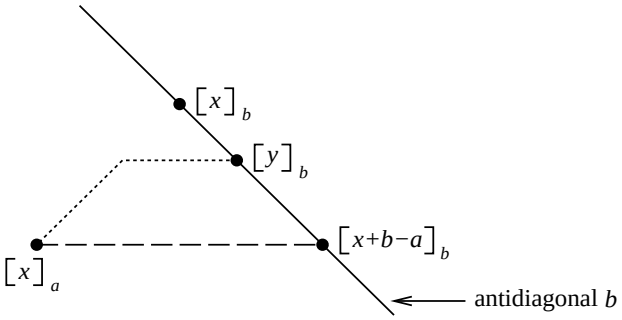

Figure 2: If $x \leq y \leq x+b-a$, then $gap\_cost([x]_a, [y]_b) = \frac{1}{2}\varepsilon(b-a) + \delta(y-x)$.

$b \geq a$, we wish to know in antidiagonal coordinate terms (1) which of Cases 1 or 2 apply, (2) whether $\vec{q} \geq \vec{p}$ (i.e. the points can be linked), and (3) an explicit formula for $gap\_cost(\vec{p}, \vec{q})$. It is readily seen that $\vec{q}$ satisfies Case 1 (i.e., $\Delta_0 \geq \Delta_1$) with respect to $\vec{p}$ if and only if $y \geq x$, and assuming that, then $\vec{q} \geq \vec{p}$ (i.e., $\Delta_1 \geq 0$) if and only if $y \leq x + (b-a)$. Moreover, in this case it also follows algebraically that $gap\_cost([x]_a, [y]_b) = \frac{1}{2}\varepsilon(b-a) + \delta(y-x)$, recalling that $\delta = 2\lambda - \varepsilon$. See Figure 2. For Case 2, the conditions are $y \leq x$ and $y \geq x + (a-b)$, and the gap cost is $\frac{1}{2}\varepsilon(b-a) + \delta(x-y)$.

In terms of the more general formulation that will follow, note that in both cases the conditions corresponding to (1) and (2) define a domain of points that is the intersection of two half-lines, where one half-line has the form $C_0 y \geq c_0$ and the other has the form $C_1 y \leq c_1 + \alpha b$, where C_0, C_1 and α are dependent *only on the case* in question, whereas c_0 and c_1 also depend on $e.e\vec{n}d$. In particular, for Case 1, $C_0 = C_1 = \alpha = 1$, $c_0 = x$ and $c_1 = x - a$; for Case 2, $C_0 = C_1 = \alpha = -1$, $c_0 = -x$ and $c_1 = -x - a$. It is further true that the gap cost in each case is expressible as $Sy + s + \beta b$, where S and β depend only on the case, whereas s depends also on $e.e\vec{n}d$. Specifically, $S = \delta$, $\beta = \frac{1}{2}\varepsilon$, and $s = -\frac{1}{2}a\varepsilon - \delta x$ for Case 1, and $S = -\delta$, $\beta = \frac{1}{2}\varepsilon$, and $s = -\frac{1}{2}a\varepsilon + \delta x$ for Case 2.

Our fragment-chaining algorithm processes the fragments in increasing order of the antidiagonal of their beginning points. For a fragment f, let $f.b\vec{e}g = [y]_b$, and let F_b be the set of fragments e with $antidiag(e.e\vec{n}d) \leq b$. For each of the two cases, $\pi \in \{1, 2\}$, we need a data structure that permits us to efficiently determine $select\_best_\pi(f.b\vec{e}g)$, i.e., the largest value of $score\_at(e) - gap\_cost(e.e\vec{n}d, f.b\vec{e}g)$ over those $e \in F_b$ for which $e \leq f$.

Consider a specific case (the other is handled symmetrically) that is uniquely characterized by the values C_0, C_1, S, α, and β described above. Further recall that for each fragment e, we will have constants c_0, c_1, and s that complete the specification of the region and cost of a gap from $e.e\vec{n}d$. We will denote these as $e.c_0$, $e.c_1$ and $e.s$ to model their dependence on e. In these terms the problem is to compute the maximum of $score\_at(e) - e.s - Sy - \beta b$ over those $e \in F_b$ for which $C_0 y \geq e.c_0$ and $C_1 y \leq e.c_1 + \alpha b$.

Our efficient data structure is a small variation of the traditional "range tree" [1], which is discussed in a number of books on algorithms or computational geometry, e.g., pp. 83-88 of Preparata and Shamos [9]. Essentially what is needed is a maximum function value over those fragments e for which the pair $(e.c_0, e.c_1)$ is in the semi-infinite range $(-\infty, C_0 y] \times [C_1 y - \alpha b, \infty)$. Note that all the $e.$-quantities are known at the outset and hence static. The only quantities that change during the course of the algorithm proper are the set of active fragments F_b and values $score\_at(e)$.

Recall that a 2-level range tree is a binary tree of binary trees. Specifically, we let T be a complete binary tree whose leaves are the set of all fragments e ordered by increasing value of $e.c_0$. In order to perform binary searches over T, each interior vertex, v, is labeled with the c_0-value of the rightmost leaf in its subtree. In addition, v has a pointer to a tree T_v whose shape and leaves are the shape and leaves of v's subtree, but which is ordered by increasing value $e.c_1$. Each interior vertex, w, of one of these nested binary trees T_v has a label to direct binary searches over c_1-values, analogous to T. In addition w has a dynamically maintained pointer, $w.max$, such that at the time the algorithm is about to process antidiagonal b, $w.max$ points to the leaf e in w's subtree maximizing $score\_at(e) - e.s$ subject to the constraint that $e \in F_b$. See Figure 3.

For each of the two cases, the following procedure evaluates $select\_best(\vec{q})$ for any $\vec{q} \equiv [y]_b$. ¿From T's root, traverse the path to the fragment with the smallest $e.c_0 > C_0 y$ using the search labels. Define a *left child of a path* as all left children of a vertex on the path that are not themselves on the path. For all left children v of the path in T just traversed, in T_v traverse the

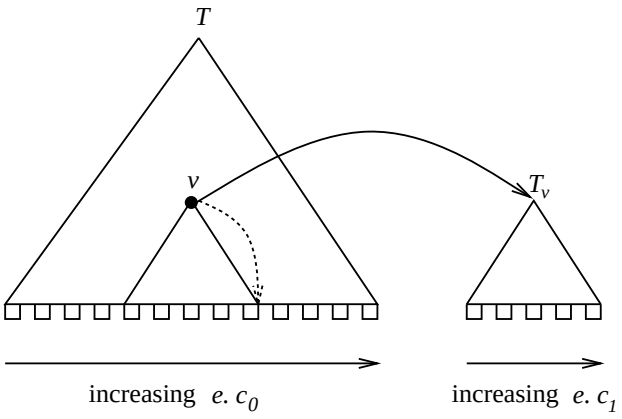

Figure 3: Data structure for a given case when $K = 2$.

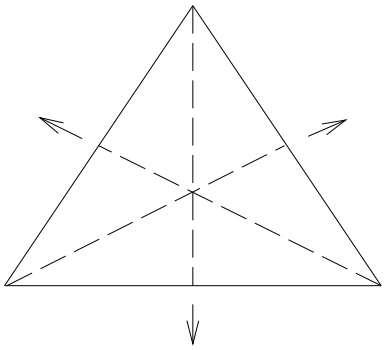

Figure 4: Intersection of the sweep plane in three dimensions with the set of $\vec{q} \geq \vec{p}$ for fixed $\vec{p}$. The six subtriangles are regions within which the "sum-of-pairs" *gap_cost* function (see Section 4) is affine. Arrows indicate directions of triangle expansion as the sweep plane recedes from $\vec{p}$.

path to the fragment with the largest $e.c_1 < C_1 y - \alpha b$ using the search labels. Over all the right children of all these paths, accumulate the maximum of the $w.max$'s associated value $score\_at(e) - e.s$. Return this maximum minus the quantity $Sy + \beta b$. The required time is $O(\log^2 F)$ per evaluation of *select_best*. Observe that this is just a 2-dimensional range query except that the nested trees are being used to efficiently compute the desired maximum over the entries in the range of the query.

The range tree structure for each of the two cases is completely static save for the *.max* fields, and so they are constructed in a preprocessing step. The leaves of T_v are obtained by merging the leaves of $T_{v.left}$ and $T_{v.right}$, so we can order the leaves of all T_v trees (for all v in T) in time $O(F \log F)$. Note that a given fragment e occurs in $O(\log F)$ T_v trees (corresponding to the vertices on a path from T's root to e). Given the leaf orders, the interiors of the trees can be built in time linear in the number of leaves. Thus all the trees can be built in a total of $O(F \log F)$ time and space.

Before beginning the antidiagonal ordered loop over the fragments, all the values $w.max$ are set to **nil**. When the "sweep antidiagonal" passes $e.\vec{end}$, the entry for e is activated by the following procedure. In each of the $O(\log F)$ T_v trees where e occurs, update the $w.max$ values in an upward pass from e to the root. The time required is at most $O(\log^2 F)$.

In summary, for $K = 2$ sequences, we do $O(F \log F)$ preprocessing, then process F fragments, each of which takes time $O(\log^2 F)$ for each of two cases used to evaluate $score\_at(f)$ plus time $O(\log^2 F)$ to activate f in the range trees for the two cases. Thus, the total time is $O(F \log^2 F)$ and the space requirement is $O(F \log F)$. The knowledgeable reader might ask whether the Willard-Lueker variation [12, 6] on range trees that shaves a $\log F$ factor from the complexity of queries could be applied here. Unfortunately, the answer appears to be negative, as we must not only divine the set of entries in the range, but also determine a maximum function value over a dynamic subset of them.

3 A Glance at K = 3 Sequences

Development of geometric intuition for three sequences was a key step for us in discovering the general algorithm, and we expect that it will be useful for the reader. Of course, geometric intuition is a highly individual. We hope that the following remarks will provide a useful guide. These ideas are formalized below as Theorem 4.1.

Consider a fixed point, $\vec{p}$, in Cartesian 3-space. The set of all $\vec{q} \geq \vec{p}$ forms a semi-infinite "rectangle" with its corner at $\vec{p}$. Let $\mathcal{L}$ be the line in 3-space that passes through the origin and the point $(1, 1, 1)$, and consider the intersection of the semi-infinite rectangle with a "sweep plane" that is perpendicular to $\mathcal{L}$. That intersection forms an equilateral triangle in the plane. As the plane recedes from p, the triangle expands, though the center remains fixed relative the the plane's intersection with $\mathcal{L}$. See Figure 4.

Assuming that $\vec{p} = e.\vec{end}$ for some fragment e, we seek to divide the triangle (i.e., the set of points on the plane that can be chained after $\vec{p}$) into regions within which the function $h(\vec{q}) = score\_at(e) - gap\_cost(\vec{p}, \vec{q})$ is affine. In other words, if the function's value in one of these regions is interpreted as the height of a surface over the region, then the surface is planar. The number and nature of the subregions depends on how

we define *gap_cost* for more than two sequences. With the definition adopted in the next section, the triangle is divided into $6 = K!$ right triangles. Each of the smaller triangles can be interpreted as the intersection of three half-planes; two of the half-planes stay fixed as the sweep plane recedes from $\vec{p}$, while the other moves in a regular fashion. Moreover, the orientations of the sides of this family of triangles are independent of $\vec{p}$.

4 The General Case

The algorithm of Section 2, though it incurs an additional $\log F$ factor in time and space over previous results for $K = 2$, generalizes to $K > 2$, as described in this section. While it is clear how to define *gap_cost* when $K = 2$, there are a number of ways to extend it to the multi-dimensional case. We explored several, as discussed in the paper's Epilog. The most difficult of these is the so-called "sum-of-pairs" model, where the cost of a multi-gap is the sum of the pairwise gap costs of the $\frac{1}{2}K(K-1)$ pairs of unaligned segments, and so we focus on just it here. Specifically, if $\vec{p} = (p_0, p_1, \ldots, p_{K-1})$ and $\vec{q} = (q_0, q_1, \ldots, q_{K-1})$ then we define:

$$gap\_cost(\vec{p}, \vec{q}) = \sum_{0 \le i < j < K} gap\_cost((p_i, p_j), (q_i, q_j))$$

For this definition of *gap_cost*, our approach to fragment chaining divides the computation of *gap_cost* into $K!$ cases, each of which is determined by one of the possible orderings of the components, $\Delta_i = q_i - p_i$, of the vector $\vec{\Delta} = \vec{q} - \vec{p}$ between $\vec{p}$ and $\vec{q}$. Specifically, for each permutation π of $[0, K-1]$ the condition $\Delta_{\pi(0)} \ge \Delta_{\pi(1)} \ge \ldots \ge \Delta_{\pi(K-1)}$ defines one of the cases, and we say π sorts $\vec{\Delta}$. In what follows, π^{-1} will denote the inverse permuation of π, i.e. $\pi^{-1}(\pi(i)) = i$ for all i. Each case is handled identically, so we find that in an initial reading of this section it helps to focus on the particular case where π is the identity permutation, i.e., $\pi(i) = i$ for all i. Summation notation is simplified by assuming that all sums are over the index i which always begins at 0. For example, $x_0 + x_1 + \ldots + x_N$ will be written as $\sum^N x_i$.

Hence forward each case is denoted by a permutation π, and a pair of points $\vec{p}$ and $\vec{q}$ are said to be in the domain of the case if π sorts $\vec{q} - \vec{p}$. Certainly, every pair of points is in the domain of at least one case. For any case π, Lemma 4.1 below gives the condition under which two Cartesian points in the domain of π can be linked, as well as a formula for the *gap_cost* between them.

LEMMA 4.1. *Let π be a permutation of $[0, K-1]$. If π sorts $\vec{\Delta} = \vec{p} - \vec{q}$ then (1) $\vec{p}$ can be linked to $\vec{q}$ iff $\Delta_{\pi(K-1)} \ge 0$ and (2) $gap\_cost(\vec{p}, \vec{q}) = \frac{1}{2}(K-1)\varepsilon \cdot \sum^{K-1} \Delta_i + \delta \cdot \sum^{K-1} (\frac{1}{2}(K-1) - \pi^{-1}(i))\Delta_i$.*

Proof. Part (1) is easily proved by noting that within the domain of π, $\Delta_{\pi(K-1)} \ge 0$ is equivalent to $\Delta_i \ge 0$ for all i, which in turn is equivalent to asserting that $\vec{p} \le \vec{q}$. The proof of part (2) begins by observing that simply permuting terms in the summations by π gives the equivalent identity:

$$\begin{aligned} gap\_cost(\vec{p}, \vec{q}) &= \tfrac{1}{2}(K-1)\varepsilon \cdot \textstyle\sum^{K-1} \Delta_{\pi(i)} \qquad (\dagger) \\ &\quad + \delta \cdot \textstyle\sum^{K-1} (\tfrac{1}{2}(K-1) - i)\Delta_{\pi(i)}. \end{aligned}$$

which we prove by induction on K.

(BASIS) When $K = 2$, equation (†) asserts that $gap\_cost(\vec{p}, \vec{q}) = \frac{1}{2}\varepsilon \cdot (\Delta_{\pi(0)} + \Delta_{\pi(1)}) + \delta \cdot (\frac{1}{2}\Delta_{\pi(0)} - \frac{1}{2}\Delta_{\pi(1)})$, whereas the definition of $gap\_cost(\vec{p}, \vec{q})$ is $\varepsilon \cdot \Delta_{\pi(1)} + \lambda \cdot (\Delta_{\pi(0)} - \Delta_{\pi(1)})$. Substituting the definition $\delta = 2\lambda - \varepsilon$ and rearranging terms proves the identity of these formulas.

(INDUCTION) Assume equation (†) for fixed $K \ge 2$. Adding another sequence, where $\Delta_{\pi(K)} \le \Delta_{\pi(i)}$ for $0 \le i < K$, adds K terms of the form:

$$\begin{aligned} &gap\_cost(< p_{\pi(i)}, p_{\pi(K)} >, < q_{\pi(i)}, q_{\pi(K)} >) \\ &\quad = \varepsilon\Delta_{\pi(K)} + \lambda(\Delta_{\pi(i)} - \Delta_{\pi(K)}). \end{aligned}$$

Thus when $\vec{p}$ and $\vec{q}$ are $(K+1)$-vectors:

$$\begin{aligned} gap\_cost(\vec{p}, \vec{q}) &= \tfrac{1}{2}(K-1)\varepsilon \cdot \textstyle\sum^{K-1} \Delta_{\pi(i)} \\ &\quad + \delta \cdot \textstyle\sum^{K-1} (\tfrac{1}{2}(K-1) - i)\Delta_{\pi(i)} \\ &\quad + \textstyle\sum^{K-1} (\varepsilon\Delta_{\pi(K)} + \lambda(\Delta_{\pi(i)} - \Delta_{\pi(K)})) \\ &= [\tfrac{1}{2}K\varepsilon \cdot \textstyle\sum^{K} \Delta_{\pi(i)} - \tfrac{1}{2}\varepsilon \textstyle\sum^{K-1} \Delta_{\pi(i)} - \tfrac{1}{2}K\varepsilon\Delta_{\pi(K)}] \\ &\quad + [\delta \cdot \textstyle\sum^{K} (\tfrac{1}{2}K - i)\Delta_{\pi(i)} - \tfrac{1}{2}\delta \textstyle\sum^{K-1} \Delta_{\pi(i)} \\ &\qquad + \tfrac{1}{2}K\delta\Delta_{\pi(K)}] \\ &\quad + \textstyle\sum^{K-1} (\varepsilon\Delta_{\pi(K)} + \lambda(\Delta_{\pi(i)} - \Delta_{\pi(K)})) \\ &= \tfrac{1}{2}K\varepsilon \textstyle\sum^{K} \Delta_{\pi(i)} + \delta \textstyle\sum^{K} (\tfrac{1}{2}K - i)\Delta_{\pi(i)} \\ &\quad + (-\tfrac{1}{2}\varepsilon - \tfrac{1}{2}\delta + \lambda) \textstyle\sum^{K-1} \Delta_{\pi(i)} \\ &\quad + K(-\tfrac{1}{2}\varepsilon + \tfrac{1}{2}\delta + \varepsilon - \lambda)\Delta_{\pi(K)} \\ &= \tfrac{1}{2}K\varepsilon \textstyle\sum^{K} \Delta_{\pi(i)} + \delta \textstyle\sum^{K} (\tfrac{1}{2}K - i)\Delta_{\pi(i)} \end{aligned}$$

and thus equation (†) holds for all $K \ge 2$ by induction. □

Our chaining algorithm proceeds in order of antidiagonals of the fragment endpoints, in accordance with the data-dependencies of the recurrence for *score_at* given in the Introduction. Thus Lemma 4.1 must be expressed in terms of an antidiagonal coordinate system. As for 2 dimensions, a K-dimensional point $\vec{p}$

is defined to be on the antidiagonal $antidiag(\vec{p}) = \sum^{K-1} p_i$, i.e., the sum of p's coordinates. However, the idea of p's diagonal must generalize from a scalar to a $(K-1)$-vector. Namely, $\vec{p}$ is said to be on diagonal $diag(\vec{p}) = (p_0 - a/K, p_1 - a/K, \ldots, p_{K-2} - a/K)$ where $a = antidiag(\vec{p})$. Once again there is an isomorphism between a Cartesian point $\vec{p}$ and its antidiagonal point $[\vec{x}]_a$ where $\vec{x} = diag(\vec{p})$ and $a = antidiag(\vec{p})$. Be wary in what follows that $\vec{x}$ is a $(K-1)$-vector and $\vec{p}$ is a K-vector. Lemma 4.2 below gives the equivalences needed to express Lemma 4.1 in the framework of antidiagonals.

LEMMA 4.2. *Suppose $\vec{p} \equiv [\vec{x}]_a$ and $\vec{q} \equiv [\vec{y}]_b$. Let $[\vec{\Gamma}]_\Lambda = [\vec{y}]_b - [\vec{x}]_a$, i.e. $\Gamma_i = y_i - x_i$ for $0 \le i < K-1$ and $\Lambda = b - a$. The conditions for sorting $\vec{\Delta}$ and linking $\vec{p}$ and $\vec{q}$ may be directly translated as follows. For i, $j < K-1$:*

$$\begin{array}{lll} \Delta_i \ge \Delta_j & \textit{iff} & \Gamma_i - \Gamma_j \ge 0. \\ \Delta_i \ge \Delta_{K-1} & \textit{iff} & (\sum^{K-2} \Gamma_i) + \Gamma_i \ge 0. \\ \Delta_i \ge 0 & \textit{iff} & -\Gamma_i \le \Lambda/K. \\ \Delta_{K-1} \ge 0 & \textit{iff} & \sum^{K-2} \Gamma_i \le \Lambda/K. \end{array}$$

Finally, if π sorts $\vec{\Delta}$ then:

$$\begin{aligned} gap\_cost([\vec{x}]_a, [\vec{y}]_b) &= \tfrac{1}{2}(K-1)\varepsilon \cdot \Lambda \\ &\quad + \delta \cdot \textstyle\sum^{K-2} (\pi^{-1}(K-1) - \pi^{-1}(i))\Gamma_i \end{aligned}$$

Proof. It follows from the definitions of antidiagonal coordinates that if $0 \le i < K-1$, then $\Delta_i = \Gamma_i + \Lambda/K$, whereas $\Delta_{K-1} = \Lambda/K - \sum^{K-2} \Gamma_i$. The stated equivalences between inequalities follow immediately from these facts.

To see the equivalence of the formula for *gap_cost* with the formula given in the statement of Lemma 4.1, first note the straightforward identity of the coefficients of ε, i.e., that $\frac{1}{2}(K-1)\sum^{K-1} \Delta_i = \frac{1}{2}(K-1)\Lambda$. For the coefficients of δ we have:

$$\begin{aligned} &\textstyle\sum^{K-1} (\frac{1}{2}(K-1) - \pi^{-1}(i))\Delta_i \\ &= \textstyle\sum^{K-2} (\frac{1}{2}(K-1) - \pi^{-1}(i))\Delta_i \\ &\quad + (\tfrac{1}{2}(K-1) - \pi^{-1}(K-1))\Delta_{K-1} \\ &= \textstyle\sum^{K-2} (\frac{1}{2}(K-1) - \pi^{-1}(i))(\Gamma_i + \Lambda/K) \\ &\quad + (\tfrac{1}{2}(K-1) - \pi^{-1}(K-1))(\Lambda/K - \textstyle\sum^{K-2} \Gamma_i) \\ &= \textstyle\sum^{K-2} [(\frac{1}{2}(K-1) - \pi^{-1}(i)) \\ &\qquad - (\tfrac{1}{2}(K-1) - \pi^{-1}(K-1))]\Gamma_i \\ &\quad + [(\textstyle\sum^{K-2} (\frac{1}{2}(K-1) - \pi^{-1}(i)) \\ &\qquad + (\tfrac{1}{2}(K-1) - \pi^{-1}(K-1))]\Lambda/K \\ &= \textstyle\sum^{K-2} (\pi^{-1}(K-1) - \pi^{-1}(i))\Gamma_i \\ &\quad + [\textstyle\sum^{K-1} (\frac{1}{2}(K-1) - \pi^{-1}(i))]\Lambda/K \\ &= \textstyle\sum^{K-2} (\pi^{-1}(K-1) - \pi^{-1}(i))\Gamma_i \\ &\quad + [\tfrac{1}{2}K(K-1) - \textstyle\sum^{K-1} i]\Lambda/K \\ &= \textstyle\sum^{K-2} (\pi^{-1}(K-1) - \pi^{-1}(i))\Gamma_i \end{aligned}$$

□

Our last mathematical step, before proceeding to algorithmic development, is to recapitulate Lemmas 4.1 and 4.2 in vector algebraic terms in order to abstract the relevant structure of the domain of the cases and of the associated gap costs.

THEOREM 4.1. *Let π be a fixed permutation of $[0, K-1]$. For a given fragment end, $e.\vec{end} \equiv [\vec{x}]_a$, there exists $(K-1)$-vectors $\vec{C}_0$, $\vec{C}_1$, ..., $\vec{C}_{K-1}$, and $\vec{S}$, and scalars c_0, c_1, ..., c_{K-1}, s, α, and β that have the following properties.*

(a) For a fixed *antidiagonal $b \ge a$, the set of all points $[\vec{y}]_b$ that are in the domain of π and can be linked to $e.\vec{end}$ (i.e., satisfy $\Delta_{\pi(0)} \ge \Delta_{\pi(1)} \ge \ldots \ge \Delta_{\pi(K-1)} \ge 0$) are within a non-empty, bounded, and convex polytope delimited by K hyperplanes (of dimension $K-1$).*

(b) $K-1$ of the inequalities defining the convex polytope have the form $\vec{C}_j \cdot \vec{y} \ge c_j$ for $j = 0, \ldots, K-2$ where "$\cdot$" denotes inner product. Note that these hyperplanes are independent of b.

(c) The remaining bounding hyperplane depends on b and is given by the inequality $\vec{C}_{K-1} \cdot \vec{y} \le c_{K-1} + \alpha b$.

(d) The value of $gap\_cost([\vec{x}]_a, [\vec{y}]_b)$ is $\vec{S} \cdot \vec{y} + s + \beta b$.

(e) Finally, the vectors $\vec{C}_i$, $\vec{S}$, and scalars α, β are independent of e. Only the K scalars c_i and s depend on e.

Proof. In order to succinctly describe the various vectors of the construction that constitutes the proof, we define $[\cdots f(u) \cdots]_u$ as the $(K-1)$-vector $[f(0), f(1), f(2), \cdots, f(K-2)]$ where f is a function of the index u. In addition, let ι_{ui} be 1 if $u = i$ and 0 otherwise.

First consider the conditions for which the point $[\vec{y}]_b$ is in the domain of π. For $i, j < K-1$, $\Delta_i \ge \Delta_j$ is equivalent to $\Gamma_i - \Gamma_j \ge 0$, which is equivalent to $\vec{C} \cdot \vec{y} \ge c$ for $\vec{C} = [\cdots \iota_{ui} - \iota_{uj} \cdots]_u$ and $c = \vec{C} \cdot \vec{x}$. Similarly, $\Delta_i \ge \Delta_{K-1}$ for $i < K-1$ iff $\vec{C} \cdot \vec{y} \ge c$ for $\vec{C} = [\cdots 1 + \iota_{ui} \cdots]_u$ and $c = \vec{C} \cdot \vec{x}$; the converse, $\Delta_{K-1} \ge \Delta_i$, is simply obtained by multiplying $\vec{C}$ and

c by -1. Taken together, the $K-1$ inequalities that guarantee that π sorts Δ translate into the $K-1$ inequalities of part (b) of the theorem.

The linking inequality, $\Delta_{\pi(K-1)} \geq 0$ of Lemma 4.1 translates into the inequality of part (c) as follows. If $\pi(K-1) = K-1$, then the condition is equivalent to $\sum^{K-1} \Gamma_i \leq \Lambda/K$, which is equivalent to $\vec{C} \cdot \vec{y} \leq c + \alpha b$ for $\vec{C} = [\cdots 1 \cdots]_u$, $\alpha = 1/K$, and $c = \vec{C} \cdot \vec{x} - \alpha a$. On the other hand, if $\pi(K-1) < K-1$, then the link condition is equivalent to $\vec{C} \cdot \vec{y} \leq c + \alpha b$ for $\vec{C} = [\cdots -\iota_{ui} \cdots]_u$, $\alpha = 1/K$, and $c = \vec{C} \cdot \vec{x} - \alpha a$. Similarly, $gap\_cost([\vec{x}]_a, [\vec{y}]_b) = \vec{S} \cdot \vec{y} + s + \beta b$ for $\vec{S} = [\cdots \delta(\pi^{-1}(K-1) - \pi^{-1}(u)) \cdots]_u$, $\beta = \frac{1}{2}(K-1)\varepsilon$, and $s = -\vec{S} \cdot \vec{x} - \beta a$. This gives part (d), and part (e) follows by inspection of the construction to this point.

The final point to be proved is part (a). Certainly the polytope bounded by the K hyperplanes embodying the case and linking condition is convex as it is the intersection of these hyperplanes. The one issue of interest is to prove that the region is non-empty and bounded for any $b \geq a$. It is certainly non-empty as $[\vec{x}]_b$ always satisfies all K inequalities. To prove the region is bounded is rather lengthy, so we just sketch how it can be done here as this fact is not directly relevant to the algorithm that follows. It suffices to show that $\vec{0}$ is the only vector $\vec{q}$ for which $[\vec{x} + c\vec{q}]_b$ is in the region for all $c > 0$. In terms of the constraints, the condition is equivalent to $\vec{C}_i \cdot \vec{q} \geq 0$ for all $i < K-1$ and $\vec{C}_{K-1} \cdot \vec{q} \leq 0$. ¿From this point one can argue on a case-by-case basis that it is impossible for $\vec{q}$ to have any non-zero components by levering the special form of the C-vectors. □

To illustrate the construction of Theorem 4.1, we list the vectors and scalars for the case of the identity permutation with $K = 5$:

$$\begin{array}{lll}
\vec{C}_0 = [1,-1,0,0] & c_0 = \vec{C}_0 \cdot \vec{x} & \\
\vec{C}_1 = [0,1,-1,0] & c_1 = \vec{C}_1 \cdot \vec{x} & \\
\vec{C}_2 = [0,0,1,-1] & c_2 = \vec{C}_2 \cdot \vec{x} & \\
\vec{C}_3 = [1,1,1,2] & c_3 = \vec{C}_3 \cdot \vec{x} & \\
\vec{C}_4 = [1,1,1,1] & c_4 = \vec{C}_4 \cdot \vec{x} - \alpha a & \alpha = \frac{1}{5} \\
\vec{S} = [4\delta, 3\delta, 2\delta, 1\delta] & s = -\vec{S} \cdot \vec{x} - \beta a & \beta = 2\varepsilon
\end{array}$$

¿From here on, we will denote each item belong to a case π by superscripting with π, and a scalar item i depending on fragment e will be denoted by $e.i$. For example, for a point $[\vec{y}]_b$ in the domain of case π, we have that $gap\_cost(e.\vec{end}, [\vec{y}]_b) = \vec{S}^\pi \cdot \vec{y} + e.s^\pi + \beta b$. The quantities α and β are never qualified by a case superscript as they are the same for all cases. Note that there are only $O(K^2)$ distinct values of the vectors $\vec{C}_i^\pi$ over all choices of π and i, and the same is true for the set of all $e.c_i^\pi$. Unfortunately, there are $K!$ different S-vectors and $e.s$-scalars.

Theorem 4.1 gives us the appropriate generalization of the $K = 2$ case discussed in Section 2. While the Theorem identifies the range of fragment start-points $f.\vec{beg}$ that can be chained after a fixed $e.\vec{end}$, the derived inequalities serve equally well to characterize the set of fragment end-points that can be chained ahead of a fixed fragment start-point. In particular, the inequalities reduce the problem of locating all such permissible predecessors to a geometrical search problem.

As is widely known among computational geometers, the data structure of Figure 3 generalizes readily to a K-level range tree, as sketched in Figure 5. For a given case π and fixed $[\vec{y}]_b$, the data structure allows one to quickly determine a set of subtrees of T^{K-1} trees, the union of whose leaves is precisely the set of fragment end-points, $e.\vec{end}$, that satisfy the K inequalities given in parts (b) and (c) of Theorem 4.1. We augment the traditional range-tree construction so that each interior vertex of a T^{K-1} tree has a dynamically maintained pointer, *max*, to the leaf in the vertex's subtree that maximizes $score\_at(e) - e.s^\pi$ over all contained leaves (i.e., fragment end-points) that have already been passed by the algorithm's "sweep hyperplane."

In the following algorithm description, $\mathcal{P}$ denotes the set of all permutions π of $[0, K-1]$ and f_1, f_2, $\cdots$, f_F are the given fragments. The notes following the algorithm summary give expanded descriptions of individual lines of the algorithm.

```
1.  for each π ∈ P do
2.      Create a range tree for {f_t.end}_{t=1}^F.
3.  for p ∈ {f_t.beg, f_t.end}_{t=1}^F in antidiagonal order do
4.      if p = f_t.end then
5.          Activate p in each T^{K-1} tree containing it.
6.      else    /* p = f_t.beg */
7.          score_at(f_t) ← f_t.score + max{0, μ} where
8.              μ = max{select_best_π(p) : π ∈ P}
```

(2) A K-level range tree, as depicted in Figure 5, is created for each π. The *max* pointers discussed below in connection with line 5 are initialized to **nil**.

(3) The loop treats fragment begin- and end-points in order of increasing b. If one fragment's end-point coincides with another fragment's begin-point, the end-point is processed before the begin-point.

(5) Consider a fixed $\pi \in \mathcal{P}$. Only the T^{K-1} trees of Figure 5 contain chain-score information, i.e., the *max* pointers. In each such tree that contains the given $f.\vec{end}$, an upward walk from that leaf updates *max* pointers.

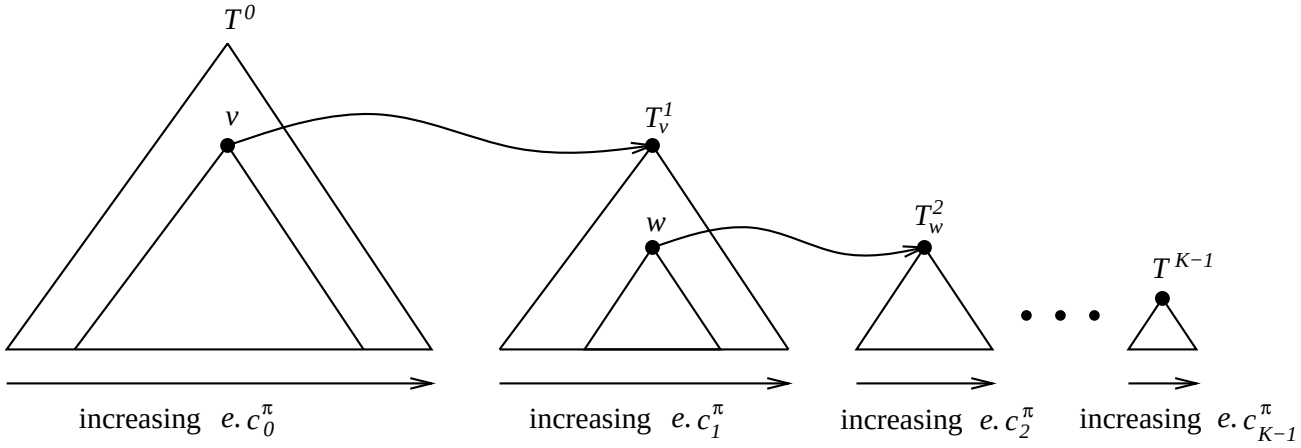

Figure 5: Data structure for a given case π when $K > 2$.

(8) The range tree for π, augmented with *max* pointers in all of its T^{K-1} trees, supports efficient computation of the maximum $score\_at(e) - e.s^\pi$ over those previously processed $f.\vec{end}$'s that satisfy inequalities (b) and (c) of Theorem 4.1. Subtracting $\vec{S}^\pi \cdot \vec{y} + \beta^\pi b$ from this maximum value gives $select\_best_\pi(f.b\vec{e}g)$ (part (d) of Theorem 4.1). In particular, the relevant fragments e are precisely those where $(e.c_0^\pi, e.c_1^\pi, \ldots, e.c_{K-1}^\pi)$ lies in the semi-infinite range $(-\infty, \vec{C}_0^\pi \cdot \vec{y}] \times (-\infty, \vec{C}_1^\pi \cdot \vec{y}] \times \cdots \times (-\infty, \vec{C}_{K-2}^\pi \cdot \vec{y}] \times [\vec{C}_{K-1}^\pi \cdot \vec{y} - \alpha b, \infty)$. An ordinary binary-tree search for the value $C_0^\pi \cdot \vec{y}$ in the T^0 tree determines a set of $O(\log F)$ subtrees (F is the number of fragments) whose leaves are the fragment end-points e satisfying the first inequality, $C_0^\pi \geq e.c_o^\pi$; those subtrees are just the "left children" of the search path. Each of the corresponding T^1 trees is searched with $C_1^\pi \cdot \vec{y}$ to identify a total of $O(\log^2 F)$ subtrees of T^1 trees containing fragments satisfying the first two inequalities, and so on. Finally, searches in T^{K-1} trees with the value $C_{K-1}^\pi \cdot \vec{y} - \alpha b$ locates a total of $O(\log^K F)$ subtrees of T^{K-1} trees containing precisely the set of end-points that satisfy all K inequalities. Use of each subtree's *max* pointers produces the maximum for $score\_at(e) - e.s^\pi$ in that subtree, and we take the maximum over all all those subtrees.

5 Analysis of the Algorithm

Analysis of the algorithm's time and space requirements can be broken down into analyses of the following quantities: (1) the time needed to construct the range trees, (2) the space needed for the range trees, (3) the time to evaluate $score\_at(f)$, and (4) the time to update *max* values in the T^{K-1} trees. Space constraints in these Proceedings limit us to considering quantity (3) in some detail.

5.1 Analysis of Search Time. Define $C(K, H)$ to be the maximum number of comparisons involved in a search of a K-level range tree of height H. Since our trees are balanced, H is $O(\log F)$. The time to search with given range values is $O(C(K, H))$, since the total number of RAM steps is proportional to the number of comparisons. $C(K, H)$ satisfies the recurrence relation:

$$C(K,H) = \begin{cases} \sigma + 1 & \text{if } K, H \geq 1 \\ K & \text{if } H = 0 \\ 0 & \text{if } K = 0 \end{cases}$$

where

$$\sigma = C(K, H-1) + C(K-1, H-1).$$

Explanation: When processing a vertex in the K-th level (considering the top level to be level K) at height H, one comparison operation might determine that all leaves of one of the subtrees lie in the range. In that case, inspection of the other subtree constitutes a level-K search of a node at height $H-1$ (hence the term $C(K, H-1)$), and a height-$(H-1)$ tree containing leaves of the first subtree must be searched at level $K-1$ (hence the term $C(K-1, H-1)$). The other case is that one of the subtrees is eliminated from the search, leaving only inspection of the other subtree, which constitutes a level-K search of a node at height $H-1$. Finally, note that in the boundary case of $H = 0$, the search may have to affirm that the fragment represented by the vertex is in each of the K ranges.

The remainder of this subsection utilizes the notation:

$$\prod^{k}[H] \text{ or } \prod^{k}[H] \text{ denotes } \prod_{i=1}^{k} \frac{H+1-i}{i}.$$

These terms are central to our analysis, and we need the following technical lemmas. The first three of them can be verified by straightforward algebraic manipulations.

LEMMA 5.1. $\prod^{k}[H-1] + \prod^{k-1}[H-1] = \prod^{k}[H]$.

LEMMA 5.2. $1 + \sum_{k=1}^{K}\prod^{k}[H-1] + \sum_{k=1}^{K-1}\prod^{k}[H-1] = \sum_{k=1}^{K}\prod^{k}[H]$.

LEMMA 5.3. $\sum_{k=0}^{K-1}(K-k)\prod^{k}[H-1] + \sum_{k=0}^{K-2}((K-1)-k)\prod^{k}[H-1] = \sum_{k=0}^{K-1}(K-k)\prod^{k}[H]$.

LEMMA 5.4. $C(K,H) \le \sum_{k=1}^{K}\prod^{k}[H] + \sum_{k=0}^{K-1}(K-k)\prod^{k}[H]$.

Proof. By induction on K. When $K = 0$, simply note that indeed $C(0,H) = 0$ (by the recurrence), while the two summations on the right-hand side in the statement of the Theorem are empty. It is also easy to verify that $C(K,0) = K = K\prod^{0}[0]$.

For the induction step:

$$\begin{aligned}
C(K,H) &\le 1 + C(K,H-1) + C(K-1,H-1) \\
&\quad \text{(by the recurrence)} \\
&\le 1 + \sum_{k=1}^{K}\prod^{k}[H-1] + \sum_{k=0}^{K-1}(K-k)\prod^{k}[H-1] + \\
&\quad \sum_{k=1}^{K-1}\prod^{k}[H-1] + \sum_{k=0}^{K-2}((K-1)-k)\prod^{k}[H-1] \\
&\quad \text{(by induction)} \\
&= \sum_{k=1}^{K}\prod^{k}[H] + \sum_{k=0}^{K-1}(K-k)\prod^{k}[H] \\
&\quad \text{(by Lemmas 5.2 and 5.3).}
\end{aligned}$$

□

LEMMA 5.5. *If* $K < \log F$, *then* $C(H,K) \le \frac{H^K}{K!}\left(1-\frac{K}{H}\right)^{-2}$.

Proof. Relaxing $\prod^{k}[H]$ to $\frac{H^k}{k!}$ and applying Lemma 5.4 gives:

$$\begin{aligned}
C(K,H) &\le \left(H + \frac{H^2}{2!} + \cdots + \frac{H^K}{K!}\right) + \\
&\quad \left(K + (K-1)H + (K-2)\frac{H^2}{2!} + \cdots + \frac{H^{K-1}}{(K-1)!}\right) \\
&= \frac{H^K}{K!}\left(1 + 2\left(\frac{K}{H}\right) + 3\left(\frac{K}{H}\right)^2 + \cdots + K\left(\frac{K}{H}\right)^{K-1}\right) \\
&\le \frac{H^K}{K!}\left(1-\frac{K}{H}\right)^{-2}
\end{aligned}$$

where the last inequality requires that $K < H$. □

THEOREM 5.1. *Assume that* $K < \log F$, *where there are* K *sequences and* F *fragments. Then* $score\_at(f)$ *can be evaluated for fragment* f *in time* $O(\log^K F)$.

Proof. Lemma 5.5 shows that the time for a single range query is $O((\log^K F)/K!)$, and evaluating $score\_at(f)$ requires $K!$ such queries. Only $O(K^2)$ range values, $\vec{C}_i \cdot \vec{y}$, are needed over *all* these searches, and they are all computable in that amount of time since the vector inner products only take $O(1)$ time due to the special form of the vectors $\vec{C}_i$. Thus the time to do all the searching for a given fragment is $O(K^2 + \log^K F)$ $= O(\log^K F)$. □

5.2 Other Factors in the Analysis. An analysis that is not given here shows the algorithm's space requirement to be $O(K!F + KF(\log F + K)^{K-1})$. The term $KF(\log F + K)^{K-1}$ is for the $K!$ range trees, while the term $K!F$ comes from the information associated with each of the F fragment end-points. Specifically, for each fragment there are $O(K^2)$ distinct $e.c_i^{\pi}$ values and $K!$ different $e.s^{\pi}$ values. The time required to compute the information for each fragment and to build the range trees is proportional to the space requirement. The time for all updates of max-values in T^{K-1} trees is $O(F(\log F + K)^K$. Thus, assuming that $K < \log F$, the algorithm's total time requirement is $O(F\log^K F)$ and space is $O(KF\log^{K-1} F)$.

6 Epilog

We found it interesting and somewhat vexing that there is a log-factor gap between our result and existing work in the case where $K = 2$. We thought hard about trying to reduce this discrepancy but have been unable to do so, and the reasons appear to be fundamental. In the two-dimensional case, previous authors appealed to the fact that one could partition an antidiagonal into $O(F)$ convex domains for which a fragment would yield the best link. However, in three dimensions there can be $O(F^2)$ convex domains in the partitioning of an antidiagonal. Thus the generalization of earlier work would lead to an $\Omega(F^2)$ algorithm for $K = 3$, worse

than the simple algorithm. To improve upon our result appears to be a difficult open problem.

The result of the previous section is specific to the sum-of-pairs scoring scheme for *gap_cost*. Other common choices are the consensus scheme $gap\_cost(\vec{p},\vec{q}) = \lambda \sum_i \Delta_i$, and the column scheme $gap\_cost(\vec{p},\vec{q}) = \lambda \max_i \Delta_i$. For the consensus scheme there is only one case. For example, when $K = 3$ the equilateral triangle of Figure 4 is not subdivided. Only one range tree is required, and the $O(K)$ $e.c$ values and one $e.s$ value needed are computable in $O(K)$ time per e. Thus the time complexity is $O(KF + F\log^K F\,/\,K!)$ and space is $O(KF + F\log^{K-1} F\,/\,(K-1)!)$.

For the column scheme, things are a bit trickier. Technically the gapcost "projection" of a fragment can be decomposed into K planar faces, but each of these is bounded by $2(K-1)$ hyperplanes. $K-1$ of the hyperplanes move with the antidiagonal, while the others are fixed. One could model the situation with only K range trees but each would be of dimension $2(K-1)$. So while from one perspective there are only K cases, it is computationally more efficient to break the domain into $K!$ cases, giving an algorithm with the same time and space complexity as for sum-of-pairs scores. It appears that our approach is quite general and applicable to a wide range of symbol-independent choices of *gap_cost*.

References

[1] J. Bentley and H. Maurer, *Efficient worst-case data structures for range searching*, Acta Informatica, 13 (1980), pp. 155-168.

[2] K.-M. Chao and W. Miller, *Linear-space algorithms that build local alignments from fragments*, to appear in Algorithmica.

[3] K.-M. Chao, J. Zhang, J. Ostell and W. Miller, *A local alignment tool for very long DNA sequences*, to appear in CABIOS.

[4] D. Eppstein, Z. Galil, R. Giancarlo and G. F. Italiano, *Sparse dynamic programming. I: linear cost functions; II: convex and concave cost functions*, J. Assoc. Comput. Mach., 39 (1992), pp. 519-567.

[5] D. B. Johnson, *A priority queue in which initialization and queue operations take $O(\log\log D)$ time*, Math. Syst. Theory, 15 (1982), pp. 295-309.

[6] G. Lueker, *A data structure for orthogonal range queries*, Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science (1978), pp. 28-34.

[7] W. Miller, M. Boguski, B. Raghavachari, Z. Zheng and R. C. Hardison, *Constructing aligned sequence blocks*, Journal of Computational Biology, 1 (1994), pp. 51-64.

[8] E. Myers and X. Huang, *An $O(N^2\log N)$ restriction map comparison and search algorithm*, Bull. Math. Biol., 54 (1992), pp. 599-618.

[9] F. Preparata and M. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1988.

[10] G. Schuler, S. Altschul and D. Lipman, *A workbench for multiple alignment construction and analysis*, Proteins: Structure, Function and Genetics, 9 (1991), pp. 180-190.

[11] E. Sobel and H. Martinez, *A multiple alignment program*, Nucl. Acids Res., 14 (1986), pp. 363-374.

[12] D. E. Willard, *Predicate-oriented database search algorithms*, Ph. D. Thesis, Harvard University, and Aiken Computation Laboratory Report TR-20-78, 1978.

[13] W. Wilbur and D. Lipman, *Rapid similarity searches of nucleic acid and protein data banks*, Proc. Nat. Acad. Sci. USA, 80 (1983), pp. 726-730.

[14] Z. Zhang, B. Raghavachari, R. Hardison and W. Miller, *Chaining multiple-alignment blocks*, Journal of Computational Biology, 1 (1994), pp. 217-226.