On the Complexity of Sparse Exon Assembly

Carmel Kent* University of Haifa

Gad M. Landau† University of Haifa & Polytechnic University

Michal Ziv-Ukelson‡ Technion - Israel Institute of Technology

Abstract

Gene structure prediction is one of the most important problems in computational molecular biology. It involves two steps: the first is finding the evidence (e.g. predicting splice sites) and the second is, interpreting the evidence, that is, trying to determine the whole gene structure by assembling its pieces. In this paper we suggest a combinatorial solution to the second step, which is also referred to as the *"Exon Assembly Problem"*. We use a similarity based approach which aims to produce a single gene structure based on similarities to a known homologous sequence.

We target the sparse case, where filtering has been applied to the data, resulting in a set of $O(n)$ candidate exon blocks. Our algorithm yields an $O(n^2\sqrt{n})$ solution.

1 Introduction

Prediction of genes in eukaryotic DNA is seriously complicated by noisy regions (*introns*) that interrupt the coding regions (*exons*) of genes. A combinatorial approach, due to Gelfand, Mironov and Pevzner [7, 20, 27] incorporates similarity analysis into gene prediction by attempting to find a set of potential exons in a genomic sequence whose concatenation is highly similar to one of the already known gene sequences in the database.

The task of gene prediction is generally divided into two stages. The first task is that of finding *candidate exons* in a long DNA sequence believed to contain a gene. A candidate exon is a sequence fragment whose left boundary is an *acceptor* site or a start codon, and the right boundary is a *donor* site or a stop codon. The nucleotide sequence in Figure 1 contains marked sites where a candidate exon may begin and end. Uppercase A-E mark identified sites where an exon is likely to begin (start/acceptor sites), and lowercase f-j mark sites where exons are likely to end (stop/donor sites). Candidate exons are A-f, A-g, A-h, A-i, A-j, B-f, B-g, B-h, B-i, B-j, C-g, etc.

The second task, which is referred to as *"The Exon Assembly Problem"* and which is the main concern of this paper, is that of selecting the best subset of non overlapping candidate exons to

*Department of Computer Science, University of Haifa, Haifa 31905, Israel; email: ckent@cs.haifa.ac.il; partially supported by the Israel Science Foundation grant 282/01.

†Department of Computer Science, University of Haifa, Haifa 31905, Israel, phone: (972-4) 824-0103, FAX: (972-4) 824-9331; Department of Computer and Information Science, Polytechnic University, Six MetroTech Center, Brooklyn, NY 11201-3840; email: landau@poly.edu; partially supported by the Israel Science Foundation grant 282/01.

‡Department of Computer Science, Technion-Israel Institute of Technology, Technion City, Haifa 32000, Israel; Phone: (972-4) 829-4883 ; email: michalz@cs.technion.ac.il; partially supported by the Aly Kaufman Post Doc-toral Fellowship.

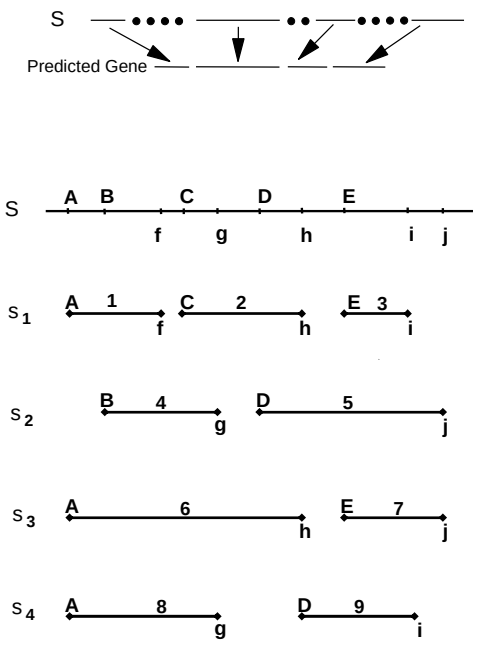


Figure 1: A nucleotide sequence (S) and four of its derived candidate genes (s_1, s_2, s_3, s_4).

cover the sequence of the predicted gene. (Four of the many possible assemblies of candidate exons as candidate genes are shown in the figure: $s_1 = \{A-f, C-h, E-i\}$, $s_2 = \{B-g, D-j\}$, $s_3 = \{A-h, E-j\}$ and $s_4 = \{A-g, D-i\}$.) Each candidate gene (a concatenation of non-intersecting candidate exons which satisfy some natural consistency conditions [25]) is compared against the target sequence, which is an already known gene from a homologous species. (For a survey on the subject of Gene Prediction, the interested reader is referred to [18] and to *http://www.nslij-genetics.org/gene/*).

Good filtration is crucial for exon assembly, especially if targets from distant taxa are used. In [20], the authors studied the performance of the exon assembly algorithm for different targets on a complete set of human genomic sequences with known relatives and demonstrated that the average performance of the method remains high even for distant targets. The authors analyzed several filtration procedures of varying strength and demonstrated that weak filtration provides better results with mammalian targets. In the case of distant targets, the stronger filtering was more useful than in the case of close targets, and sometimes it was shown to significantly improve the prediction quality. Furthermore, it was also explicitly shown (see *http://www-hto.usc.edu/software/procrustes-/#salign)* that the number of candidate exons generated in either weak or moderate filtration modes is linear in the size of the genomic sequence: the default (weak) filter retained approximately 1 exon per 14 nucleotides on the average, while the moderate filter retained approximately 1 exon per 33 nucleotides.

Therefore, we re-address the Exon Assembly problem for the linear-sized candidate exon-set case. More formally, for any two substrings B and B' of a genomic sequence S, we write $B \prec B'$ if B ends before B' starts. A sequence $\Gamma = (B_1 \ldots, B_n)$ of substrings of S is a *chain* if $B_1 \prec B_2 \prec \ldots \prec B_n$. We denote the concatenation of strings from the chain Γ by $\Gamma^* = B_1 * B_2 * \ldots * B_n$.

Definition 1 *Given two sequences, S and T, of size $O(n)$ each, and a set $\beta = \{B_1, \ldots B_b\}$ con-*

taining $O(n)$ blocks (substrings of S), of size $O(n)$ each. The ***Sparse Exon Assembly Problem*** *is to find a chain Γ of strings from β such that score(Γ^*, T) is maximum among all chains of blocks from β.*

We suggest a new algorithm for *Sparse Exon Assembly*. Our new approach is based in efficient elimination of redundancy due to candidate-exon overlaps. Note that dominant portions of each of the competing candidate gene assemblies in Figure 1 are segments common to other candidates, since the candidate exons overlap in the genomic source sequence. (For example, the two source strings S_1 and S_2 share the substrings B-f, C-g, D-h and E-i.) The authors of [7] indeed noticed this redundancy and utilized it to speed up the naive dynamic programming algorithm. When no filtration is applied, and the number of candidate exons generated is $O(n^2)$, their approach reduces the Exon Assembly complexity from the naive $O(n^4)$ down to $O(n^3)$. However, when filtration is applied, and the number of candidate exons is $O(n)$, their algorithm still yields the same $O(n^3)$ result.

A thorough examination of the problem leads to the conclusion that it is the dependency of traditional dynamic programming on the direction in which it is applied, as well as the fact that values in each dynamic programming table are sensitive to the content of preceding (or symmetrically following) chained blocks in the candidate solution, which stand in the way of isolating the common denominator between blocks and exploiting the overlap redundancy.

1.1 The Results of This Paper

We describe an $O(n^2\sqrt{n})$ solution for Sparse Exon Assembly, which improves upon the previous results by $O(\sqrt{n})$. Furthermore, the new algorithm degrades gracefully, so that even in the case when no filtration is applied to the exon set, and the number of candidate exons may be $O(n^2)$, our algorithm will yield an $O(n^3)$ complexity which is no worse than previous results. The efficiency improvement is based in a tighter, more sophisticated elimination of overlap redundancy among the candidate exons. The crux of the new algorithm is thus based in three components.

1. A Rectilinear Steiner Minimal Arborescence technique [8, 9, 17, 24] is applied to the analysis of the overlapping candidate exons. The Steiner technique is a well-practiced tool in both Computational Geometry and Graph Theory problems. Its importance stems from the fact that it has applications in as diverse areas as VLSI-layout and the study of phylogenetic trees. The application of the Steiner tool here to the parsing of overlapping candidate exons leads to efficient re-use of the substrings shared by multiple exon candidates and is key to the resulting efficiency gain. (See section 4.)

2. We replace the dynamic programming tables in the network graph with alternative $DIST$ data structures. (See Section 3.) This will allow us to focus on the local common denominator between overlapping blocks, and mask out the global information which varies for each considered chain in which the blocks are embedded.

3. In order to achieve direction flexibility in the alignment computation, a new key operation is introduced in this paper which supports both left-to-right and right-to-left computations. (See Section 5.) Note that related work on dynamic programming "against the grain" without the I/O independence feature can be found in [13, 14, 15].

The combination of the above three components enables the new algorithm to eliminate redundancy by applying the major part of the alignment work to multiple blocks in tandem, thus reducing the work associated with re-computation of segments shared by several blocks.

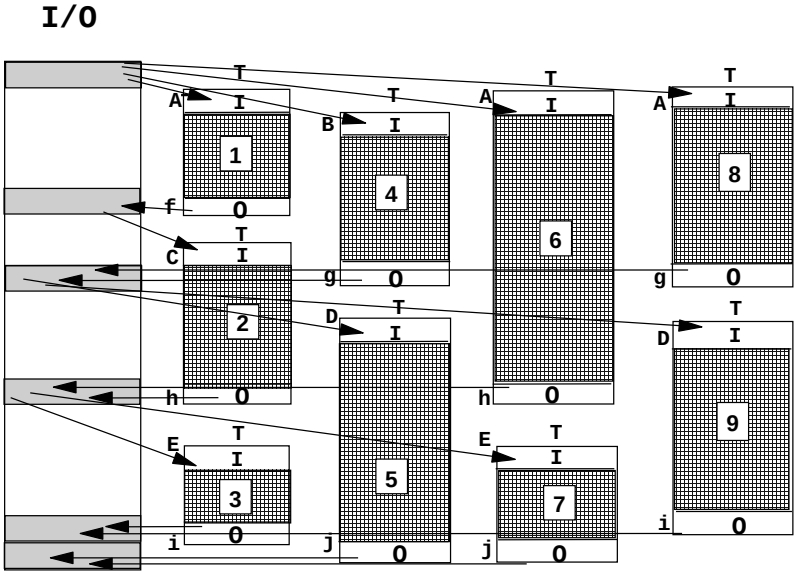


Figure 2: A example of the basic network dynamic programming algorithm, showing the candidate exon block dynamic programming tables. Consecutive "state-shots" of the I/O vector at the stages when it is accessed also shown. This figure continues the example of Figure 1.

The results of this paper apply to all distance or similarity scoring schemes which use a scoring table with rational number values and employ a linear gap penalty. In the next section we will briefly describe Gelfand, Mironov and Pevzner's algorithm [7] since our algorithm uses a similar framework.

2 Gelfand, Mironov and Pevzner's Algorithm

The Exon Assembly problem is first reduced to a search for the optimal path in a network graph. Nodes in this graph correspond to candidate-exon blocks, arcs correspond to potential transitions between blocks, and the path weight is defined as the weight of the optimal alignment between the concatenated blocks of this path and the target sequence.

Figure 2 demonstrates the network alignment graph. Each of the rectangles corresponds to the dynamic programming table for the alignment of a single candidate-exon substring S_i^j of S versus T. Clearly, each such table is of size $O(n^2)$. The first row in each such dynamic programming table is denoted I and the last row is denoted O. The only special feature of the computation is the fact that the I row is being set from a global vector named *the I/O vector* which will be defined below. All other computations are done in the standard way based on the penalty matrix.

The algorithm uses the I/O vector to maintain global information regarding the various possible chainings of the dynamic programming tables for the exon-blocks.

Definition 2 *At step k of the algorithm, $I/O[\ell]$, for $\ell = 1 \ldots n$, holds the best alignment score of the prefix T_1^l with any possible chaining of a set of complete, non-overlapping candidate exons which ends at an index no higher then k.*

To summarize, the work done at step k of the algorithm consists of the following three tasks:

1. First, all dynamic programming tables whose I row is at index k will be initialized with the values of the I/O vector.

2. Then, the values of all rows internal to blocks (neither I nor O) which correspond to the character at index k of S, will be computed from their preceding row via dynamic programming computation.

3. Finally, for each dynamic programming table DP_i^k (corresponding to the alignment of candidate exon S_i^k versus T) the I/O vector will be updated with the information from row O of DP_i^k: $I/O[\ell] = max\{DP_i^k[k, \ell], I/O[\ell]\}$, for $\ell = 1 \ldots n$.

Time Complexity

In both the Dense and the Sparse case the time and space complexity of the algorithm is $O(n^3)$. The interested reader is referred to [7] for more details.

3 An Alternative Approach to Exon Assembly

Our new algorithm for Sparse Exon Assembly will maintain the main frame of the original algorithm and will similarly run a sweep-line, top-to-bottom traversal over the network graph of candidate exon blocks. However, the new approach will be based in replacing the dynamic programming tables for the blocks with an alternative data structure which analyzes the DP graph (see Figure 3,6 and 9), denoted $DIST$ [1, 4, 12, 23].

Definition 3 *$DIST_B^A[0 \ldots n][0 \ldots n]$- given the DP Graph for the alignment of a sequence A of size m versus a sequence B of size n, $DIST_B^A[i, j]$ stores the weight of the highest scoring path from vertex $(0, i)$ in the first row of the DP graph to vertex (m, j) in the last row of the DP graph.*

In other words, $DIST_B^A[i, j]$ stores $score(A, B_i^j)$ (see Figure 3).

Note that since all substrings of B may be considered as candidate exons, this problem can also be referred to as the *All Substrings Alignment Problem* [10, 11, 2, 6].

We will use an $O(n)$ representation of $DIST$, which replaces the full $O(n^2)$ $DIST$, and which is explained in Section 5.1.

Consider the DP graph for the alignment of any candidate exon S_i^j with T in Figure 2. By replacing the dynamic programming table with a $DIST$ data structure, the alignment work involved in the computation of column O from column I can be greatly reduced, as follows. Given $DIST_{S_i^j}^T$ and input row I, the output row O can be computed in $O(n)$ time instead of $O(n^2)$, by employing the techniques from [5, 16].

Intuitively, the savings can be viewed as follows: for each block, the algorithm will eliminate the work which is invested by the original exon assembly algorithm in the computation of all dynamic programming table rows between I and O (highlighted with dark squares in Figure 2). Clearly such an approach could lead to an $O(n^2)$ time complexity for the computation of the optimal path

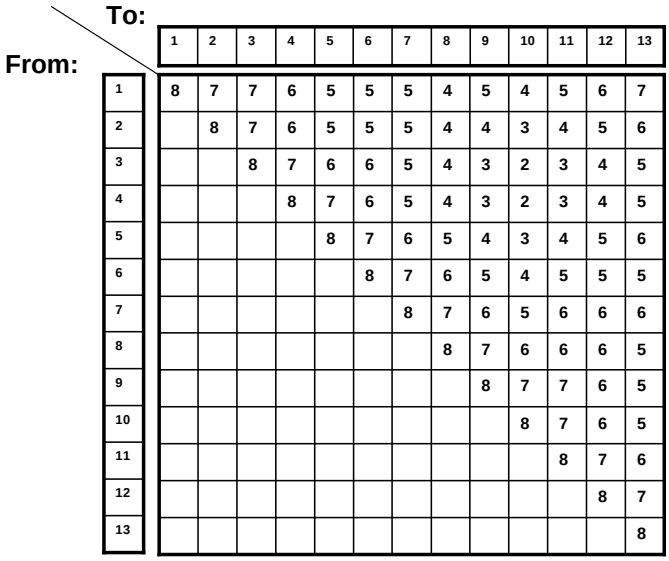


Figure 3: The full $DIST_B^A$ where A = "$ACCADBAB$", B = "$BBCAACABABCCB$", for the Edit Distance metric. Note that $\psi = 3$.

through the exon blocks, assuming that the $DIST$s for the comparison of each block with T are available.

However, such an assumption does not hold in this case and therefore $O(n)$ $DIST$ tables need to be constructed. Computing a single $DIST$ for the comparison of two $O(n)$-sized sequences would naively take $O(n^3)$ time. Schmidt [23] shows how to compute such a $DIST$ in $O(n^2)$ time. Thus, the computation of $O(n)$ $DIST$s, where each $DIST$ represents two strings of size $O(n)$ each, can be done in $O(n^3)$ time. This creates a complexity bottleneck which overrides the speed-up which was obtained in reducing the work for the network dynamic programming graph to $O(n^2)$.

Therefore, in the rest of this paper we address the challenge of how to compute the $DIST$s for all blocks in an efficient manner that would reduce the $O(n^3)$ $DIST$-construction bottleneck to $O(n^2\sqrt{n})$.

4 Breaking the $O(n^3)$ bottleneck with "Append-Prepend" Parsing

In Section 5, the following two incremental $DIST$ construction operations will be described. Both operations cost $O(n)$ time each, and will be used for efficient computations of all $DIST$s for β (Definition 1).

Definition 4 ***DISTAppend*** *- given a single input character a and $DIST_B^A$, computes $DIST_B^{Aa}$, where Aa is the concatenation of the character a to the end of string A.*

This operation is based on the work of [23].

Definition 5 ***DISTPrepend*** *- given a single input character a and $DIST_B^A$, computes $DIST_B^{aA}$, where aA is the concatenation of the character a to the beginning of string A.*

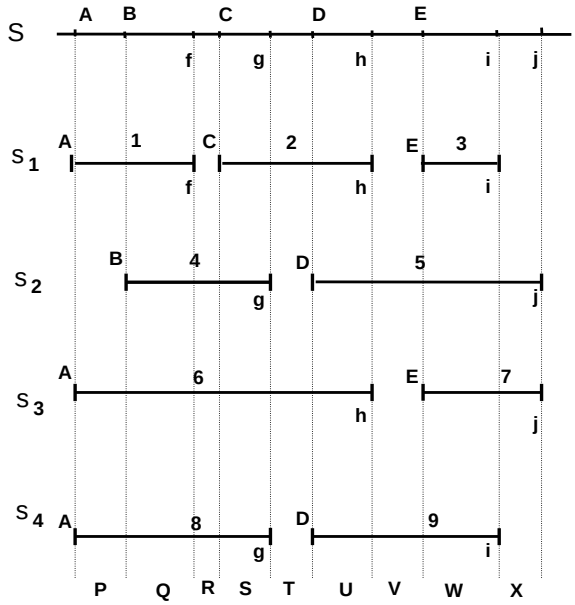


Figure 4: An example of a set of candidate exon blocks. This figure continues the example of Figure 1. Let $|P| = 1$, $|Q| = 4$, $|R| = 1$, $|S| = 2$, $|T| = 3$, $|U| = 3$, $|V| = 3$, $|W| = 4$ and $|X| = 2$.

This operation will be described in section 5.3.

Consider the candidate exons $1 \ldots 9$ which are shown in Figure 4. Dotted vertical lines are used in this example to mark all start and end positions of candidate-exons. Clearly, each such dotted line can be translated to at least one I/O-vector update. The letters $P, Q, \ldots T$ are used to mark *unit substrings* of S, which are uninterrupted by a beginning or end of a candidate exon.

Naively, the $DIST$ for each candidate-exon could be generated by a series of $O(n)$ $DISTAppend$ operations, or alternatively by a series of $O(n)$ consecutive $DISTPrepend$ operations. This would yield a total of $O(n^2)$ $DISTAppend$ and $DISTPrepend$ operations. Since the complexity of each $DISTAppend$ and $DISTPrepend$ operation is $O(n)$, the total work would amount to $O(n^3)$. However, a more careful construction scheme would try to order the $DIST$ increment operations in such a way as to minimize re-generation of unit-substrings that appear in more than one exon. For example, generating the two exons "PQ" (block 1 in Figure 4) and "QRS" (block 4) by a series of $DISTAppend$ operations would yield a total of 12 operations. Alternatively, suppose that the substring "Q", which is common to both exons, is first computed at the cost of 4 $DISTAppend$ operations, and then "PQ" is obtained from "Q" via a single $DISTPrepend$ while "QRS" is obtained from "Q" via 3 $DISTAppend$ operations. The total cost in this case would be 8 instead of 12, since "Q" is only generated once. From this example we conclude that both the bi-directional nature of $DIST$ and the fact that any subset of exons may share substrings could lead to an efficient construction of the associated set of $DIST$s. However, some order of $DISTAppend$ and $DISTPrepend$ operations should be determined that will minimize the number of required operations.

For example, lets concentrate on the construction of $DIST$ tables for exons 1, 4, 6 and 8 (see figure 4). Suppose we choose to start by generating the $DIST$ for substring Q. What should be the next step? If the $DIST$ table for Q is extended to the right to QR, exons 4, 6, and 8 will benefit as opposed to exon 1. The consequence of this construction order is that the left extension for P will have to be done twice: once for exon 1 (the creation of substring PQ) and once for exons 6 and 8

(the creation of substring PQRS). On the other hand, if we choose to extend Q to the left first by creating the $DIST$ table for substring PQ the symmetric result is that of building the extension to R twice. This leads to the following optimization problem.

Definition 6 *The **Append-Prepend Parsing optimization problem: (APPOP)** Compute the minimal-cost series of DISTAppend and DISTPrepend operations that will generate the DISTs for all the candidate exons in β.*

4.1 $APPOP$ Reduced to RSMA

In this section we show how to solve $APPOP$ by reducing it to a Steiner-Tree problem [9]. Furthermore, the special features of the problem will allow us to tightly map it to a special case of directed Steiner trees on a rectilinear grid. Such trees have the great advantage that their size has a proven bound - a feature which is key to the efficiency gain suggested by our algorithm. The Rectilinear Steiner Minimal Arborescence (RSMA) is defined as follows [24].

Definition 7 *Let N be a set of nodes lying in the first quadrant of E^2. The set of edges is composed solely of horizontal and vertical arcs oriented only from left to right and from bottom to top. A Rectilinear Steiner Tree for N is a tree rooted at the origin and containing all nodes in N. A Rectilinear Steiner Arborescence (RSA) is a directed Rectilinear Steiner Tree for N with the property that for each $n_i \in N$, the length of the unique path in the tree from the origin to n_i equals $x_i + y_i$. A **Rectilinear Steiner Minimum Arborescence (RSMA)** for N is an RSA for N that has the shortest possible total edge size.*

This problem, which is *NP*-Complete [26], was first studied by Ladeira de Matos [19]. He proposed an exponential time dynamic programming algorithm to solve the problem. Rao, Sadayappan, Hwang and Shor [24] have suggested a heuristic 2-approximation algorithm for the problem, with the time complexity of $O(n \log n)$, which finds an RSMA with a bounded size of $O(n\sqrt{n})$. Their algorithm scans the nodes in the grid using a diagonal sweep line approach, starting at the highest rightmost corner of the grid and progressing toward the root at the lowest leftmost corner of the grid. We use their algorithm while sweeplining only the diagonals in the upper right triangle. Additional related work can be found in [8, 9, 17]. In the next claim we prove that our problem reduces to a variant of RSMA. Meanwhile, we give a high level description of this reduction.

Consider the upper right triangle of the $n \times n$ directed grid graph G in figure 5, associated with the sequence $S = s_1, s_2, \ldots, s_n$ which continues the example in figures 1 – 4. The origin of this graph is point $z_1 = (0, 0)$ in the lowest, leftmost corner of the grid. Each column and row in G can be associated to a different character of S. Columns are associated to $s_n, \ldots, s_1$ from left to right, and the rows are associated to $s_1, \ldots, s_n$ from bottom to top. Let the diagonal which is just above the main diagonal of G, be called the *Characters Diagonal*. Nodes in the Characters Diagonal are associated with consecutive characters of S, such that each character s_i is associated to node $(i, n - i + 1)$ on the Characters Diagonal. Recall that S can also be considered as a sequence of the unit substrings $P, Q, \ldots, X$, which are substrings of S, uninterrupted by a beginning or end of a candidate exon. For convenience purposes only, we will gather columns, rows and nodes on the Characters Diagonal of G so that they can be associated to the unit substrings instead of to single characters (see figure 5).

For example, the unit substring $Q = s_2, \ldots, s_5$ is associated to rows $2, \ldots, 5$, to columns $22, \ldots, 19$ and thus to the nodes $(2, 22)$, $(3, 21)$, $(4, 20)$ and $(5, 19)$ respectively.

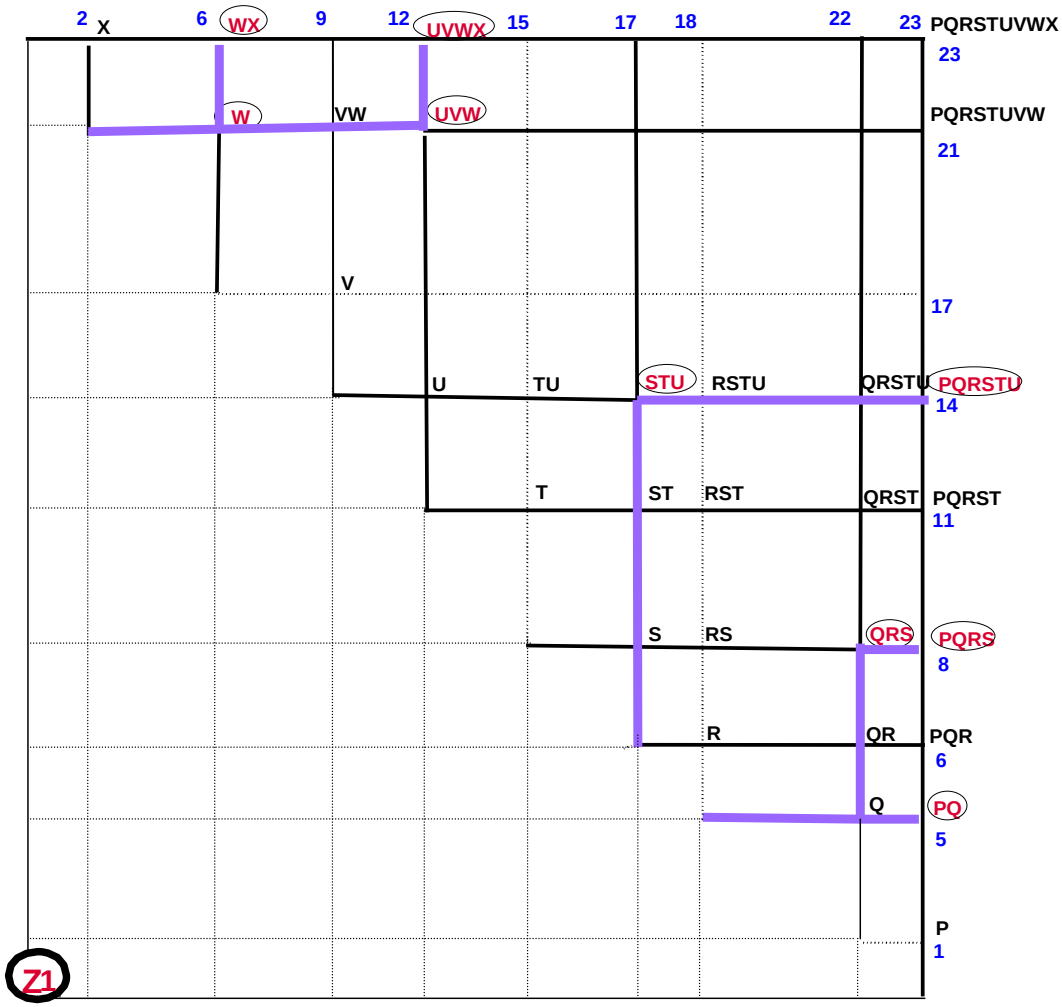


Figure 5: *APPOP* reduced to a Minimal Steiner Arborescence problem on the upper triangle in a Rectilinear Grid. The terminal nodes, representing the candidate-exon substrings, are circled. The designated root z_1 is located at the origin $(0, 0)$. The sequence of unit substrings of S is $PQRSTUVWX$. For the sake of the figure's clarity, the sizes have been omitted from the edges. Instead, the accumulative lengths of the unit substring is shown in the right and upper sides of the figure. The RSMA approximation, as computed by employing the algorithm of claim 2, is marked with thick bright lines. This figure continues the examples of Figures 1 and 4.

We will now show how the substrings of S are associated to nodes in G. Any substring S_i^j $(j \geq i)$ can be realized by concatenating the unit substrings associated to rows i to j or by concatenating the unit substrings associated to columns $n - j + 1$ to $n - i + 1$. The node which represents this substring in G is $(j, n-i+1)$, which is the upper right intersection point of these rows and columns. Among these nodes, the set of $O(n)$ terminal nodes N (corresponding to the set of candidate exons) are circled in figure 5.

The edges demonstrate all the possible options to extend any of these substrings either by *DISTAppend* or by *DISTPrepend*. Therefore, all edges in the upper triangle of the grid graph are directed, either upward or to the right. An edge leaving a node upwards corresponds to an extension

of the substring represented by the node via a $DISTAppend$ operation. For example, consider the three consecutive edges starting at the node representing the substring RST and going upwards end at the node representing the substring $RSTU$ (at the row corresponding to the unit substring U). These edges represent the result of 3 consecutive $DISTAppend$ operations on RST which yields the target substring $RSTU$ (U's length is 3). Similarly, an edge leaving a node to the right corresponds to an extension of the substring represented by the node via a $DISTPrepend$ operation. For example, the four consecutive edges starting at the node representing the substring RST and going to the right end at the node representing the substring $QRST$ (at the column corresponding to the unit substring Q). This edge represents the result of 4 consecutive $DISTPrepend$ operations on RST which yields the target substring $QRST$ (Q's length is 4).

Consider some set of single characters of S (on the Characters Diagonal). The extension of each one of them up and to the right with a sequence of $DISTAppend$ and $DISTPrepend$ operations can finally result all the candidate exon substrings. Clearly, such a sequence can be mapped to a forest which spans all the terminals in N, such that each tree in the forest is rooted in the Characters Diagonal of G and grows in the upwards and right directions. A minimal-size such forest would therefore yield a solution to the $APPOP$.

The RSMA variant problem is then to find a set of paths from the Characters Diagonal to all terminals in N such that the total size of the edges in these paths is as small as possible (see Figure 5).

Claim 1 *$APPOP$ can be reduced to the RSMA problem on the upper triangle in a Rectilinear Grid.*

Proof: The proof is immediate from the previous discussion. ∎

Next we show that running the RSMA algorithm of Rao *et al.* [24] on G would yield a 2-Approximation to $APPOP$ with the time complexity of $O(n \log n)$.

In their algorithm the distance in L_1 of a point $p = (p_x, p_y)$ from the origin is defined as $p_x + p_y$. Clearly, all points lying on a certain diagonal share the same distance from the origin. Thus, this RSMA algorithm scans the nodes in the grid using a diagonal sweep line approach. Starting at the highest rightmost corner of the grid and progressing toward the root at the lowest leftmost corner of the grid, the algorithm does not leave a diagonal until it processes every node on it which is a candidate to participate in a minimal arborescence approximation. Theorem 7 in [24] states that for any diagonal processed during the execution of their algorithm, the size of the forest found by this step is no more then twice the size of the optimal forest spanning the upper triangular subgraph defined by this diagonal. Therefore we can run the standard RSMA algorithm of [24] on the upper triangle of G and halt once the Characters Diagonal is reached.

Claim 2 *A 2-Approximation to the $APPOP$ can be computed in $O(n \log n)$.*

Proof: The proof is immediate from the previous discussion. ∎

Let $\tau_G(N)$ denote the approximated RSMA obtained by running the algorithm of claim 2 on G. The next lemma, which is based on Rao *et al.* [24], defines a bound on the size of $\tau_G(N)$.

Lemma 1 $|\tau_G(N)| = O(n(\sqrt{n}))$

Proof: Rao *et al.* [24] proved that the size of an RSMA for n terminal points in a unit square is bounded by $\sqrt{2n} + 2$. If the problem is scaled up from a unit-grid with n^2 points to an $n \times n$ grid with n^2 points, both the size of each upward-edge and the size of each right-edge in the arborescence increases by a factor of n, which yields a scaling of the total size of the arborescence by a factor of n to $O(n\sqrt{n})$. ∎

4.2 The New Sparse Exon Assembly Algorithm

The new algorithm is given below.

Algorithm Sparse Exon Assembly:

Input: a set β of $O(n)$ candidate-exon substrings of S and the homologous gene sequence T.
Output: a chain Γ of strings from β such that the alignment score of the concatenation of Γ with T is maximum among all chains of blocks from β.

1. Compute an approximation $\tau_G(N)$ of the RSMA of G as described in section 4.1.

2. Using $\tau_G(N)$, compute the linear encodings of the $DIST$s for all candidate exons in β, using the $DISTAppend$ and $DISTPrepend$ operations described in section 5.

3. Using the linear encodings of the $DIST$s, scan the network graph in a top-to-bottom sweepline of increasing row index, and for each scanned block S_i^j:

3a. Initialize I with the I/O vector.

3b. Compute O from I, applying the technique from [16] to the linear encoding of $DIST_{S_i^j}^T$ which was computed in Step 2.

3c. Update the I/O vector with O, as described in section 2.

4. Upon completion, report the maximal value in the I/O vector and by tracebacking find the maximal value chain Γ.

Time Complexity Analysis

A 2-Approximation $\tau_G(N)$ of the RSMA for $O(n)$ points in an $n \times n$ grid can be computed in $O(n \log n)$ as explained in the proof of claim 2. By Lemma 1, the size of the obtained tree $\tau_G(N)$ is $O(n\sqrt{n})$.

Following the incremental order defined by $\tau_G(N)$, the linear encodings of the $DIST$s for all exons in B are computed in $O(n\sqrt{n})$ steps. Each step consists of either a single $DISTAppend$ or a single $DISTPrepend$ operation, each in $O(n)$ time (see Section 5). Altogether, the time invested in the construction of the $DIST$s is $O(n^2\sqrt{n})$.

Using the $O(n)$ linear encodings of the $DIST$s, the computation of I from O for each of the $O(n)$ blocks is done in $O(n)$ time using [16]. In addition, the I/O update work for initializing the I row for each block from the I/O vector and for updating the I/O vector with the resulting O row for each block is also $O(n)$ per block. Altogether, this stage contributes a term of $O(n^2)$ work to the total complexity.

Thus, the total time complexity of the new Sparse Exon Assembly algorithm is $O(n^2\sqrt{n})$.

We next show that even though the sparse case improved, the algorithm degrades gracefully in the dense case. When the number of candidate exons approaches $O(n^2)$, the 2-Approximation $\tau_G(N)$ of the RSMA for $O(n^2)$ points can be computed in $O(n^2 \log n)$ similarly to the way as with the sparse case. The size of an RSMA for n^2 terminal points in a unit square is bounded by

$\sqrt{2} \times n + 2$. The scaling to an $n \times n$ grid with n^2 points brings the total size of the arborescence to $O(n^2)$.

Altogether, the time invested in the construction of the $DIST$s in the dense case is $O(n^3)$.

In a similar manner to the sparse case, the network alignment algorithm contributes a term of $O(n^3)$ work to the total complexity, yielding the total time and space complexity of $O(n^3)$ in the dense case of our Exon Assembly algorithm.

5 A DP Toolkit to Support $APPOP$

In this section we describe the two $DIST$ construction operations, which enable us to utilize the RSMA approximation $\tau_G(N)$, which was computed as a solution to the $APPOP$ of the exon-candidate set β, to efficiently construct the $DIST$ data structures which represent the alignments of all blocks in β with T. We will first explain the compressed, linear encoding representation of $DIST$, and then describe the $DISTAppend$ and $DISTPrepend$ construction operations on this data structure.

5.1 Properties of DIST that Enable Linear Encoding

Aggarwal and Park [3] observed that $DIST$ tables are Monge arrays [21].

Definition 8 *A matrix $M[0 \ldots m, 0 \ldots n]$ is a **Monge** array if either condition 1 or 2 below holds for all $a, b = 0 \ldots m;\ c, d = 0 \ldots n$:*

1. *$M[b,d] - M[b,c] \geq M[a,d] - M[a,c]$ for all $a < b$ and $c < d$.*
2. *$M[b,d] - M[a,d] \geq M[b,c] - M[a,c]$ for all $a < b$ and $c < d$.*

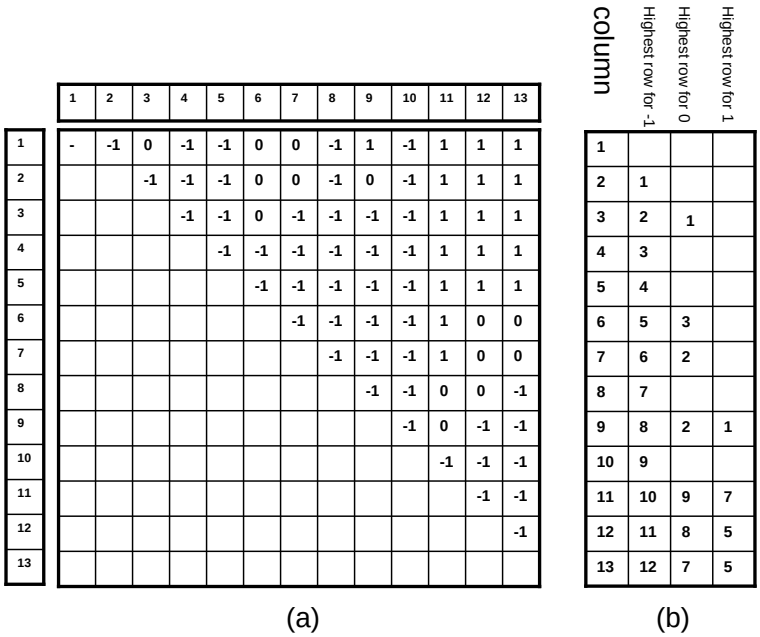
For discrete scoring schemes, a more efficient encoding of high scoring paths in the DP Graph can be achieved, by utilizing the fact that, due to the Monge property of $DIST$, the number of relevant changes, from one column to the next, is constant. This property, also discussed in [16, 23], allows for a representation of $DIST$ via an $O(n)$ number of "relevant" points.

The $HDIFF$ matrix on columns of $DIST$ is defined as follows (see Figure 6).

Definition 9 $HDIFF_{col}{}^A_B[i,j] = DIST^A_B[i,j] - DIST^A_B[i,j-1],\ for\ i,j = 1 \ldots n.$

The range of possible values for $HDIFF_{col}{}^A_B[i,j]$ depends on the scoring scheme which is used for the string comparison, and is actually the upper bound for the value difference between two consecutive elements in the dynamic programming table. We will use the term ψ to denote the range bound for $HDIFF[i,j]$ values. As an example, if the similarity metric used is LCS, the only possible values for $HDIFF$ will be either 1 or 0, and ψ assumes a value of 1. For the Edit Distance metric, on the other hand, ψ is 2, since $HDIFF$ can only assume one of the 3 values: -1, 0, 1 [28]. Our algorithm applies to all scoring scheme metrics for which ψ is a constant. Masek and Paterson [22] observed that ψ is a constant for discrete scoring schemes (*i.e.* when the values of the applied scoring matrices are restricted to rational numbers).

Schmidt [23] showed that since, for discrete scoring schemes, the number of "steps" (row indices in which the series of column entries increases in value) in each column of $HDIFF$ is constant, each column $j = 1 \ldots n$, can be encoded by a sorted list of $\psi = O(1)$ row indices i_k, where i_k is the



	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	-1	0	-1	-1	0	0	-1	1	-1	1	1	1
2			-1	-1	-1	0	0	-1	0	-1	1	1	1
3				-1	-1	0	-1	-1	-1	-1	1	1	1
4					-1	-1	-1	-1	-1	-1	1	1	1
5						-1	-1	-1	-1	-1	1	1	1
6							-1	-1	-1	-1	1	0	0
7								-1	-1	-1	1	0	0
8									-1	-1	0	0	-1
9										-1	0	-1	-1
10											-1	-1	-1
11												-1	-1
12													-1
13													

(a)

column	Highest row for -1	Highest row for 0	Highest row for 1
1			
2	1		
3	2	1	
4	3		
5	4		
6	5	3	
7	6	2	
8	7		
9	8	2	1
10	9		
11	10	9	7
12	11	8	5
13	12	7	5

(b)

Figure 6: The linear encoding compression of $DIST_B^A$ by columns, where $A =$ "$ACCADBAB$", $B =$ "$BBCAACABABCCB$", for the Edit Distance metric. (a) $HDIFF_{col}{}_B^A$. (b) $HLIST_{col}{}_B^A$. This Figure continues the example of Figure 3.

highest row index in column j, such that $HDIFF_{col}{}_B^A[i_k, j] = k$. Altogether, $HDIFF_{col}{}_B^A$ can be encoded by its $O(n)$ "steps", using the data structure defined below (See Figure 6).

Definition 10 *$HLIST_{col}{}_B^A$ is the linear encoding of $HDIFF_{col}{}_B^A$. i.e., $HLIST_{col}{}_B^A[j,k]$ holds the value of the **highest** row index i in which $HDIFF_{col}{}_B^A[i,j] = k$, for $j = 1 \ldots n$, $k = 1 \ldots \psi$.*

We have shown in [16] that the linear encoding of $DIST$ is sufficient to compute O from I, thus liberating the algorithm from the time and space overhead of maintaining the full $O(n^2)$ $DIST$ representation. In the following sections we will show that the linear representation of $DIST$ suffices for the computation of both $DISTAppend$ and $DISTPrepend$, thus allowing for the efficient $O(n)$ computation of these operations.

5.2 Schmidt's Algorithm for $DISTAppend$ on the Linear Encoding of DIST

In this section we will describe a general framework of the $DISTAppend$ operation of [23]. This algorithm is intended as a black box for the new $DISTPrepend$ operation which will be described in Section 5.3. Consider the DP graph for the alignment of Aa versus B. Note that column j in $DIST_B^A$ represents the j paths starting at row 0 of the DP graph for the alignment of A versus B, and ending at vertex (m, j). Thus, appending a single character to the end of sequence A corresponds to appending a new row to the last, bottom row of the DP graph (see Figure 7). The goal of the $DISTAppend$ operation is to compute the linear-encoded representation of $DIST_B^{Aa}$ from the linear-encoded representation of $DIST_B^A$, namely computing all the paths which end at vertices of row $m + 1$ of the DP graph.

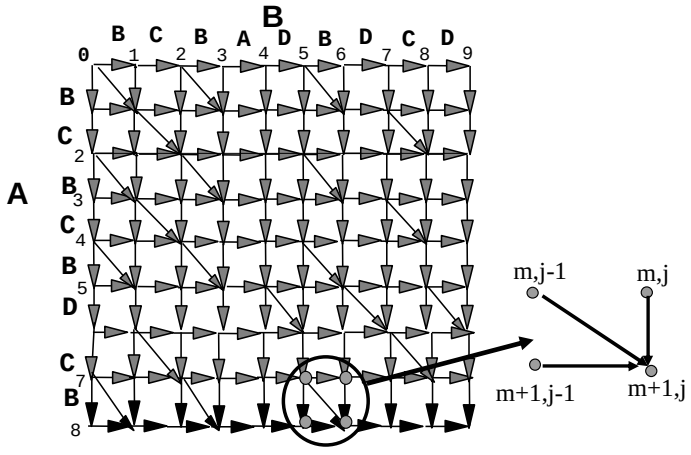


Figure 7: The DP graph for computing the similarity between A = "$BCBCBDCB$" and B = "$BCBADBDCD$". The highlighted edges demonstrate the extension of the prefix "BCBCBDC" of A by appending the character 'B'.

For each vertex in the newly appended row $(m+1, j)$, $j = 1 \ldots n$, given the difference: $DIST_B^A[i,j] - DIST_B^A[i,j-1]$ (*i.e.*, entry j of $HLIST_{col}{}_B^A$), the algorithm computes in $O(1)$ the difference: $DIST_B^{Aa}[i,j] - DIST_B^{Aa}[i,j-1]$ (entry j of $HLIST_{col}{}_B^{Aa}$)

We refer the interested reader to [23] for the detailed description of this algorithm.

Time and Space Complexity Analysis of the $DISTAppend$ Operation: Given a string B of size $O(n)$, a character a, a discrete scoring table, and the linear encoding of $DIST_B^A$ ($HLIST_{col}{}_B^A$) for a string A, the linear encoding of $DIST_B^{Aa}$ ($HLIST_{col}{}_B^{Aa}$), can be computed in $O(n)$ time and space complexity.

5.3 $DISTPrepend$ on the Linear Encoding of DIST

Given $HLIST_{col}{}_B^A$ and a single character a, in this section we will show how to compute the $DISTPrepend$ operation, *i.e.*, compute the linear encoding of $DIST_B^{aA}$ ($HLIST_{col}{}_B^{aA}$), where aA is the concatenation of the character a to the beginning of sequence A. Note that this simple technique is new.

Lets examine $DIST_B^A$'s rows and columns. Column j of this $DIST$ represents all the paths which originate in vertices 0 to j of the first row ℓ (note that we say "row ℓ" instead of "row 0" for the sake of notation clarity) of the DP graph for A versus B, and ending at vertex (m,j) of the graph (see Figure 8). Respectively, row i of $DIST_B^A$ represents all the paths originating in vertex (ℓ,i) and ending in vertices i to n of row m of the graph. Reversing the directed edges in the grid graph will enable us to view row i in $DIST_B^A$ as the paths ending (instead of originating) at vertex (ℓ,i) in the DP graph, while the values (the weights of the paths) remain the same. Therefore we can consider the paths as going now in the opposite direction: from right to left, bottom-up.

Appending a single character to the beginning of sequence A is equivalent to appending a new row (row $\ell - 1$ in Figure 8) to the top of the DP graph. In this case, in a symmetric manner to the way we viewed the *Append* operation in Section 5.2, every vertex (ℓ,i), which is the end-point of $n-i$ paths that originate the last row m of the DP graph, is represented as row i in $DIST_B^A$. Therefore, extending the DP graph by appending a row to its top end, requires extending some

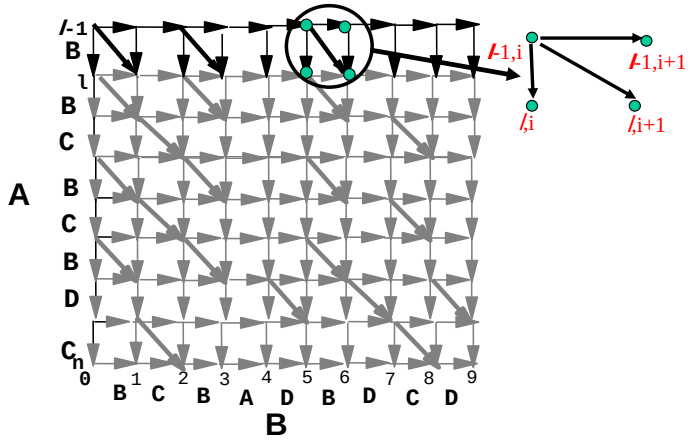


Figure 8: The DP graph for computing the similarity between $A = BCBCBDC$ and $B = BCBADBDCD$. The highlighted edges demonstrate the extension of the suffix "BCBCBDDC" of A by prepending the character 'B'.

representation of $DIST_B^A$ by the differences between its **rows** instead of its columns. These are defined, symmetrically to the Definitions of section 5.2, as $HDIFF_{row}{}_B^A = DIST_B^A[i,j] - DIST_B^A[i+1,j]$ and $HLIST_{row}{}_B^A$. Given $HLIST_{row}{}_B^A$, $HLIST_{row}{}_B^{aA}$ could be computed in $O(n)$ time by using a similar, reversed variant of the algorithm described in section 5.2.

Thus, what is left to show in order to compute $DISTPrepend$ in $O(n)$ time and space complexity, is how to compute $HLIST_{row}{}_B^A$ from a given $HLIST_{col}{}_B^A$ in $O(n)$ time and space complexity. We will next show a simple technique which does that. The same technique can be applied in the opposite direction as well, computing $HLIST_{col}{}_B^A$ from a given $HLIST_{row}{}_B^A$, a fact which will enable us to prepend and append characters to $DIST$ alternately. The next claim can be demonstrated by comparing Figures 6 and 9.

Claim 3 *Given any ordered pair (j,i) of $HLIST_{col}{}_B^A$, the reversed ordered pair (i,j) will be listed in $HLIST_{row}{}_B^A$ (i.e., index j will appear in one of the ψ "steps" at row i of $HLIST_{row}{}_B^A$).*

Proof: The existence of such ordered pair (j,i) in $HLIST_{col}{}_B^A$ implies the following 'local' structure in the full $DIST$: $DIST[i,j] - DIST[i,j-1] \neq DIST[i+1,j] - DIST[i+1,j-1]$. This simple algebraic formula implies that $DIST[i,j] - DIST[i+1,j] \neq DIST[i,j-1] - DIST[i+1,j-1]$, which immediately indicates the existence of the ordered pair (i,j) in $HLIST_{row}{}_B^A$ (*i.e.*, index j will appear in row i of $HLIST_{row}{}_B^A$). ∎

Using the above claim, $HLIST_{row}{}_B^A$ can easily be computed from $HLIST_{col}{}_B^A$ in $O(n)$ time and space via a single scan of its values.

Time and Space Complexity Analysis of the $DISTPrepend$ Operation: Given string B of size $O(n)$, a character a, a discrete scoring table, and the linear encoding of $DIST_B^A$ ($HLIST_{col}{}_B^A$) for a string A, the linear encoding of $DIST_B^{aA}$ ($HLIST_{col}{}_B^{aA}$), can be computed as follows. First, $HLIST_{row}{}_B^A$ is computed in $O(n)$ time and space complexity from $HLIST_{col}{}_B^A$, as shown in this section. Then, an $O(n)$ "reversed" version of the $DISTAppend$ described in Subsection 5.2 is applied to $HLIST_{row}{}_B^A$, yielding $HLIST_{row}{}_B^{aA}$. $HLIST_{col}{}_B^{aA}$ can then be computed from $HLIST_{row}{}_B^{aA}$ in

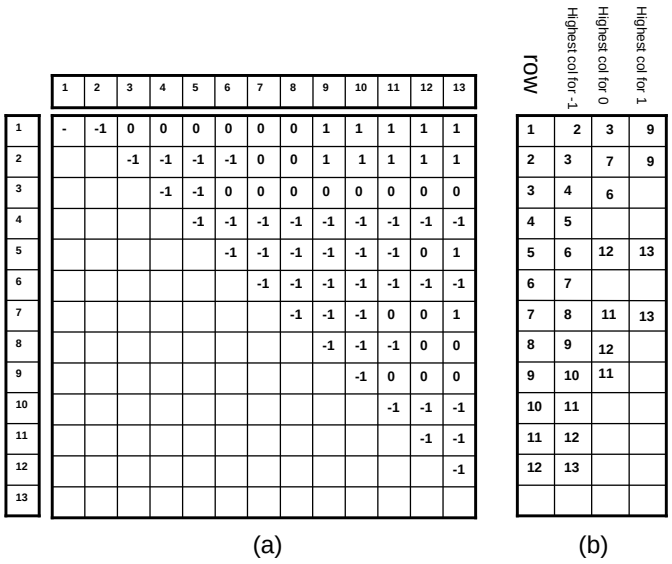


Figure 9: The linear encoding compression of $DIST_B^A$ by rows, where $A =$ "$ACCADBAB$", $B =$ "$BBCAACABABCCB$", for the Edit Distance metric. (a) $HDIFF_{row}{}_B^A$. (b) $HLIST_{row}{}_B^A$. This Figure continues the example of Figure 3.

$O(n)$ time, by a method which is symmetric to the one shown in this section. This yields a total of $O(n)$ time and space complexity for computing $HLIST_{col}{}_B^{aA}$ from $HLIST_{col}{}_B^A$.

Acknowledgments. Many thanks to Jeannette P. Schmidt for inspiration as well as very helpful discussions and comments, and in particular for pointing out to us that the number of filtered candidate exons is $O(n)$ in practice.

References

[1] Apostolico, A., M. Atallah, L. Larmore, and S. McFaddin, Efficient parallel algorithms for string editing problems. *SIAM J. Comput.*, **19**, 968-998 (1990).

[2] Alves, C. E. R., Cceres, E. N. and Song, S. W. Sequential and Parallel Algorithms for the All-Substrings Longest Common Subsequence Problem. *Technical Report RT-MAC-2003-03, Department of Computer Science, Institute of Mathematics and Statistics, University of So Paulo* (2003).

[3] Aggarawal, A., and J. Park, Notes on Searching in Multidimensional Monotone Arrays, *Proc. 29th IEEE Symp. on Foundations of Computer Science*, 497–512 (1988).

[4] Benson, G., A space efficient algorithm for finding the best nonoverlapping alignment score, *Theoretical Computer Science*, **145**, 357–369 (1995).

[5] M. Crochemore, G. M. Landau, B. Schieber, and M. Ziv-Ukelson, Re-Use Dynamic Programming for Sequence Alignment: An Algorithmic Toolkit, *String Algorithmices*, NATO Book series, KCL Press, 2004.

[6] W. Fu, W. Hon, W. Sung, On All-Substrings Alignment Problems, *COCOON* 80–89 (2003).

[7] Gelfand, M.S., A.A. Mironov, and P.A. Pevzner, Gene Recognition Via Spliced Sequence Alignment, *Proc. Natl. Acad. Sci. USA*, **93**, 9061–9066 (1996).

[8] M. Hanan, On Steiner's problem with rectiliniar distance, *SIAM J. Appl. Match.* **14**(1966), 255–265(1966).

[9] Hwang, F. K., D. S. Richards, and P. Winter, The Steiner Tree Problem, *Annals of Discrete Mathematics*, North-Holland Publisher (1992).

[10] Jiang, T., Lin, G., Ma, B., and K. Zhang, The Longest Common Subsequence Problem for Arc-Annotated Sequences, *Combinatorial Pattern Matching, 11th Annual Symposium (CPM)*, 1848 volume of Lecture Notes in Computer Science, 154–165,(2000).

[11] Jansson J., Ng S.-K., Sung W.-K.and H. Willy, A Faster and More Space-Efficient Algorithm for Inferring Arc-Annotations of RNA Sequences through Alignment. *Proceedings of the Fourth Workshop on Algorithms in Bioinformatics (WABI)*, (2004).

[12] Kannan, S. K., and E. W. Myers, An Algorithm For Locating Non-Overlapping Regions of Maximum Alignment Score, *SIAM J. Comput.*, **25**(3), 648–662 (1996).

[13] Kim, S., and K. Park, "A Dynamic Edit Distance Table.", *J. Discrete Algorithms*, **2**(2), 303–312 (2004).

[14] Landau, G.M., E.W. Myers, and J.P. Schmidt, Incremental String Comparison, *SIAM J. Comput.*, **27**, 2, 557–582 (1998).

[15] G.M Landau, E.W. Myers, and M. Ziv-Ukelson, Two Algorithms for LCS Consecutive Suffix Alignment, *15th Combinatorial Pattern Matching Conference*, 173–193 (2004).

[16] Landau, G.M., and M. Ziv-Ukelson, On the Common Substring Alignment Problem, *Journal of Algorithms*, **41**(2), 338–359 (2001).

[17] Bing Lu, Lu Ruan. Polynomial Time Approximation Scheme for the Rectilinear Steiner Arborescence Problem. *Journal of Combinatorial Optimization* **4** (3): 357-363 (2000).

[18] C. Math, MF. Sagot, T. Schiex and P. Rouz, Current methods of gene prediction, their strengths and weaknesses, *Nucleic Acid Res*, **30**, (19) (2002).

[19] Ladeira de Matos R. R., A Rectilinear Arborescence Problem, Dissertation, University of Alabama, 1979.

[20] Mironov, A.A., M.A. Roytberg, P.A. Pevzner, and M.S. Gelfand, Performance-Guarantee Gene Predictions Via Spliced Alignemnt, *Genomics 51 A.N. GE985251*, 332–339 (1998).

[21] Monge, G., Déblai et Remblai, *Mémoires de l'Academie des Sciences*, Paris (1781).

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

[23] Schmidt, J.P., All Highest Scoring Paths In Weighted Grid Graphs and Their Application To Finding All Approximate Repeats In Strings, *SIAM J. Comput*, **27**(4), 972–992 (1998).

[24] Rao, S. K., P. Sadayappan, F. K. Hwang, and P. W. Shor, The rectilinear Steiner arborescence problem. *Algorithmica*, **7**, 277–288 (1992).

[25] Roytberg, M.A., T.V. Astakhova, and M.S. Gelfand, Combinatorial Approaches to Gene Recognition, *Computers Chemistry*, **21**, 4, 229–235 (1997).

[26] W. Shi and C. Su, The Rectilinear Steiner Arborescence Problem is NP-complete, *Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, 780–787 (2000).

[27] Sze, S-H., and P.A. Pevzner, Las Vegas Algorithms for Gene Recognition: Suboptimal and Error-Tolerant Spliced Alignment, *J. Comp. Biol.* **4**, 3, 297–309 (1997).

[28] Ukkonen, E., Finding Approximate Patterns in Strings, *J. Algorithms* **6**, 132–137 (1985).