Edit Distance with Duplications and Contractions Revisited - Supplementary Materials

Tamar Pinhas, Dekel Tsur, Shay Zakov, and Michal Ziv-Ukelson
Department of Computer Science Ben-Gurion University of the Negev, Israel
{matuskat, dekelts, zakovs, michaluz}@cs.bgu.ac.il

1 Correctness of the recursive computation

In this section we show that the basic recursion given, in Section 2.2 in the paper, is correct. Call an edit script which contains only mutations, duplications, and insertions, a generating edit script, and an edit script which contains only mutations, contractions, and deletions, a reducing edit script. Consider the case of a generating edit script \( s = u^0 \to u^1 \to \ldots \to u^l = t \). For each letter \( u^k \) in some intermediate string \( u^k \), it is possible to define a letter \( u^k_{k-1} \) in \( u^{k-1} \) which generated it: \( u^k_{k-1} \) was either obtained by taking some letter \( u^r_{k-1} \) as is or by mutating it, or by duplicating some letter \( u^r_{k-1} \), or obtained by an insertion next to some letter \( u^r_{k-1} \) (in which case the generating letter \( u^r_{k-1} \) is arbitrarily chosen to be the letter before or after the insertion position). Thus, in a transitive manner, it is possible to define for every letter in \( t \) its unique generating letter in \( s \). It is straightforward to observe that the set of letters in \( t \) generated by some letter \( s_i \) in \( s \) corresponds to some consecutive substring \( t' \) of \( t \). Also, it is clear that if the edit script is an optimal script from \( s \) into \( t \), then the sub-sequence of edit operations which generated \( t' \) from \( s_i \) is an optimal (generating) script from \( s_i \) into \( t' \). The case of a reducing edit script from \( s \) into \( t \) is symmetric, where in this case we may think of every letter \( t_i \) in \( t \) as the outcome of reducing some substring \( s' \) of \( s \).

![Fig. 1: An edit of two strings via a common ancestral string \( w \).](image)

The following lemma restates a lemma presented and proven in [2].

**Lemma 1.1** [Lemma 2 in [2]] For every pair of strings \( s \) and \( t \), there is an optimal edit script from \( s \) into \( t \) in which all deletions and contractions are performed prior to all insertions and duplications.

Let \( s \) and \( t \) be two strings, and consider an optimal edit script \( s = u^0 \to u^1 \to \ldots \to u^l = t \) of the form implied by the above Lemma. Let \( w = u^k \) be the intermediate string which is obtained right after performing all deletions and contractions, and before performing any insertion or duplication. Note that the prefix of the script \( s = u^0 \to \ldots \to u^k = w \) is a reducing edit script, where the suffix of the script \( w = u^k \to \ldots \to u^l = t \) is a generating edit script. Thus, each letter \( w_i \) in \( w \) is obtained by applying a reducing script on some substring \( s' \) of \( s \), and is transformed to some substring \( t_i \) of \( t \) by applying a generating script (see Fig. 1). Due to the optimality of the script from \( s \) to \( t \), it is clear that the costs of these substring-letter edit scripts are of the forms \( ed(s', w_i) \) and \( ed(w_i, t_i) \).

Consider a prefix \( s_{0,i} \) of \( s \) and a prefix \( t_{0,j} \) of \( t \). As shown in Fig 1, \( ed(s_{0,i}, t_{0,j}) \) can be computed recursively by finding the best possible right-most letter \( \alpha \) in an optimal intermediate string \( w \). This letter is found by examining all letters in the alphabet and all partitions of \( s_{0,i} \) and \( t_{0,j} \) at indices \( k \) and \( l \), correspondingly, such that both \( s_{k,i} \) and \( t_{l,j} \) are reduced (or generated, correspondingly) from \( \alpha \). This yields the computation exhibited by Eq. 2.1 in the paper.
2 Finite number of steps for discrete costs

In this section, we show that if the set of costs is discrete then the matrices \( T^\alpha, S^\alpha, TD \) and \( TD^\alpha \) are \( D \)-discrete matrices. This requirement was discussed in Section 5.1 in the paper.

We start with a short discussion on how \( \infty \) values are dealt with, in an otherwise \( D \)-discrete matrix. Recall that, according to the definitions of the matrices used by our algorithms, these matrices hold the initial value of \( \infty \) in some cells. Specifically, the matrices \( T^\alpha \) and \( S^\alpha \) hold the value \( \infty \) in the lower triangle of the matrix throughout the algorithm. The matrices \( TD^\alpha \) and \( TD \) hold the value \( \infty \) as an initial value in all cells. These values are replaced by real ones according to the execution of the algorithm. In practice, however, the value of \( \infty \) can be replaced in a matrix by sufficiently large integer values, that acts as placeholders for \( \infty \) and allow both an unaltered computation of the min value, while at the same time maintaining the discreteness of the matrix. Specifically, that placeholder may be set as follows. Let \( B \) be a matrix with some \( \infty \) values in an outer region (e.g. lower triangle of the matrix) and assume that the property of discreteness holds for all adjacent cells in \( B \) that have values which are not \( \infty \). Then, for adjacent cells \( B_{i,j} = \text{val} \neq \infty \) and \( B_{i',j'} = \infty \) we set \( B_{i',j'} = \text{val} + 2D \). For \( \infty \) valued cells in the column or row of \((i', j')\) the value is also set to \( \text{val} + 2D \).

We call \( B_{i',j'} - B_{i,j} \) for two adjacent cells a step in \( B \). In Lemma 2.1 we show that the steps of any edit distance matrix are bounded. Then, in Lemma 2.2 we show that if the set of costs is discrete and the steps in a matrix are bounded then the matrix is \( D \)-discrete. The proof is similar to the one given in Masek and Paterson’s paper [6] for simple edit distance.

Lemma 2.1

For any two strings \( s, t \) the set of steps in an edit distance matrix of \( s \) and \( t \) is bounded.

Proof. Let \( B \) be an edit distance matrix. We show that there exists a constant \( b \) such that for every pair of indices \( i, j \), \( |B[i+1,j] - B[i,j]| \leq b \) and \( |B[i, j+1] - B[i,j]| \leq b \). Our argument uses an optimal edit script for a pair of substrings (corresponding to a certain cell in \( B \)) to construct an edit script for a slightly different pair of substring (corresponding to the neighboring cell in \( B \)). Thus, the optimal value of the neighboring cell is bounded by the cost of the construct edit script.

The edit script transforming \( s_{0,i+1} \) into \( t_{0,j} \) can be composed by first deleting \( s_{i+1} \) and then transforming \( s_{0,i} \) into \( t_{0,j} \), hence \( B[i+1,j] \leq \text{del}(s_{i+1}) + B[i,j] \). Similarly, the edit script of \( s_{0,i} \) into \( t_{0,j} \) can be composed by first inserting \( s_{i+1} \) and then transforming \( s_{0,i+1} \) into \( t_{0,j} \), hence \( B[i,j] \leq \text{ins}(s_{i+1}) + B[i+1,j] \).

Thus, \( -\text{ins}(s_{i+1}) \leq B[i+1,j] - B[i,j] \leq |B[i+1,j] - B[i,j]| \leq \text{del}(s_{i+1}) \). A similar argument holds for \( B[i,j+1] - B[i,j] \). \( \square \)

Lemma 2.2

If the set of costs is discrete then the set of possible steps in edit distance matrices is finite.

Proof. Let \( Q \) denote a discrete set of operation costs (cost functions mapping into the rational numbers are always discrete [6]). Any element of an edit matrix is the sum of the costs of a series of edit operations. Therefore, the steps are a linear integral combination of \( Q \). By Lemma 2.1, there exists a constant \( b \) such that \(|\Delta| < b \) for any possible step \( \Delta \). Since \( Q \) is discrete, there exists a real number \( r > 0 \) such that every step is a multiple of \( r \). Hence, there are at most \( 2[b/r] + 1 \) possible steps. \( \square \)

3 Omitted lemma proofs

In this section we give proofs for lemmas 1 appearing in Section 5.2 in the paper.

Lemma 1

Let \( X_{n \times k} \) and \( Y_{k \times m} \) be two \( D \)-discrete matrices. Then, \( Z = X \otimes Y \) is also a \( D \)-discrete matrix. Let \( X_{n \times m} \) and \( Y_{n \times m} \) be two \( D \)-discrete matrices. Then, \( Z = X \oplus Y \) is also a \( D \)-discrete matrix.

Proof. Part 1. Let \( D = [a, b] \). Consider a pair of adjacent entries of the form \( Z_{i,j} \) and \( Z_{i+1,j} \), and let \( r_1 \) and \( r_2 \) be indices such that \( Z_{i,j} = X_{i,r_1} + Y_{r_1,j} \) and \( Z_{i+1,j} = X_{i+1,r_2} + Y_{r_2,j} \). Then:

\[
Z_{i+1,j} - Z_{i,j} = X_{i+1,r_2} + Y_{r_2,j} - (X_{i,r_1} + Y_{r_1,j}) \\
\leq X_{i+1,r_1} + Y_{r_1,j} - (X_{i,r_1} + Y_{r_1,j}) \\
= X_{i+1,r_1} - X_{i,r_1} \leq b.
\]
Similarly, it can be shown that $Z_{i+1,j} - Z_{i,j} \geq a$, as well as the symmetric case of adjacent entries of the form $Z_{i,j}$ and $Z_{i,j+1}$.

Part 2. Let $D = [a, b]$, and consider a pair of adjacent entries $Z_{i,j}$ and $Z_{i+1,j}$ in $Z$. Then $Z_{i+1,j} - Z_{i,j} = \min(X_{i+1,j}, Y_{i+1,j}) - \min(X_{i,j}, Y_{i,j}) \leq \min(X_{i,j} + b, Y_{i,j} + b) - \min(X_{i,j}, Y_{i,j}) = b$. Similarly, it can be shown that $Z_{i+1,j} - Z_{i,j} \geq a$, as well as the symmetric case of adjacent entries of the form $Z_{i,j}$ and $Z_{i,j+1}$.

Lemma 2

Let $x = (0, \Delta(x))$ and $y = (y_0, \Delta(y))$ be two $D$-discrete vectors of length $q$. If $y_0 \geq (|D| - 1)(q - 1)$, then $x \oplus y = x$.

Proof. Let $D = [a, b]$. Since the difference between each pair of adjacent entries in $x$ is at least $a$ and at most $b$, and since the first entry $x_0$ in $x$ equals 0, each entry $x_i$ of $x$ satisfies $a \leq x_i \leq b$. Similarly, each entry $y_i$ of $y$ satisfies $y_0 + ai \leq y_i \leq y_0 + bi$. Therefore, $y_0 - ib + ia \leq y_i - x_i \leq y_0 + ib - ia$, and thus $y_0 - (q - 1)(|D| - 1) \leq y_i - x_i \leq y_0 + (q - 1)(|D| - 1)$. It follows that if $y_0 \geq (q - 1)(|D| - 1)$, then $y_i - x_i \geq 0$ for every $0 \leq i \leq q - 1$, and thus $x_i = \min(x_i, y_i)$ for every $0 \leq i \leq q - 1$, and $x \oplus y = x$. The case where $y_0 \leq -(q - 1)(|D| - 1)$ is symmetric.

4 Time complexity analysis of run-length encoded EDDC

In this section we elaborate the time complexity analysis of the algorithm presented in Section 4.2 in the paper.

Theorem 4.1 Applying min-plus square matrix multiplication to run-length encoded EDDC yields an $O\left(|\Sigma| \left(n^2 + \frac{m n^2 (\log \log n)}{\log n} \right)\right)$ time algorithm.

Proof. Let $M(n)$ denote the time complexity of performing a min-plus multiplication of two $n \times n$ matrices. The currently fastest algorithm for min-plus matrix multiplication is due to Chan [3], with the running time of $O\left(\frac{n^2 (\log \log n)}{\log n}\right)$.

Let $T(n, m, n', m')$ denote the time complexity of the algorithm, described in Section 4.2 in the paper, when running on two strings of lengths $n$ and $m$, with $n'$ and $m'$ runs, respectively.

Consider the case when $n' > 1$ and $m' > 1$. At the first level of the recursion, the COMPUTE procedure partitions the region $[1, n] \times [1, m]$ into two subregions $[1, n_1] \times [1, m]$ and $[n_1 + 1, n] \times [1, m]$ for some integer $n_1$. In the next level of the recursion, each of these two subregions is partitioned horizontally: the first subregion is partitioned into the subregions $R_1 = [1, n_1] \times [1, m_1]$ and $R_2 = [1, n_1] \times [m_1 + 1, m]$, and the second subregion is partitioned to the subregions $R_3 = [n_1 + 1, n] \times [1, m_1]$ and $R_4 = [n_1 + 1, n] \times [m_1 + 1, m]$. Due to the definition of the partition stage, the recursive call to COMPUTE($R_i$) takes $T(n_1, m_1, n'/2, m'/2)$ time, and a similar expression holds for the recursive calls on $R_2, R_3,$ and $R_4$. We also need to account for the time of the update stages performed on the first two levels of the recursion: one update stage is performed on the first level, and two stages are performed on the second level. The update stage on the first level has two steps. The first step requires $|\Sigma|$ min-plus matrix multiplications, where each multiplication is of an $\frac{m}{2} \times n'$ matrix by an $n' \times m'$ matrix. Each such multiplication can be performed in $O(\frac{m'}{n'}) M(n'/2)$ time by partitioning the two matrices into $\frac{m}{2} \times \frac{m'}{2}$ submatrices and performing $O(\frac{m'}{n'})$ multiplications between the submatrices. The second step requires $|\Sigma|$ min-plus matrix multiplications, where each multiplication is of an $\frac{m'}{2} \times n'$ matrix by an $n'(n - m')$ matrix. Again, by partitioning the matrices into $\frac{m'}{2} \times \frac{m'}{2}$ submatrices, this step can be performed in $O(|\Sigma| m m'/2)$ time. Similarly, the updates of the second level of the recursion can be done in $O(|\Sigma| \frac{n}{m} M(m'/2))$ time by performing $O\left(|\Sigma| \frac{n}{m'} \right)$ multiplications of $\frac{m'}{2} \times \frac{m'}{2}$ matrices.
Similar analysis of the other cases gives the following recurrence.

$$T(n, m, n', m') \leq \begin{cases} 
\max_{n, n_1} \left( T(n_1, m_1, n'/2, m'/2) 
\begin{array}{l}
+ T(n - n_1, m_1, n'/2, m'/2) \\
+ T(n_1, m - m_1, n'/2, m'/2) \\
+ T(n - n_1, m - m_1, n'/2, m'/2)
\end{array}
\right) & \text{if } n' > 1, m' > 1 \\
+c|\Sigma| \left( \left\lceil \frac{m}{n} \right\rceil M(n'/2) + \left\lceil \frac{n}{m} \right\rceil M(m'/2) \right) & \text{otherwise}
\end{cases}$$

for some constant $c$. It is easy to show by induction that

$$T(n, m, n', m') = O \left( |\Sigma|nm + |\Sigma| \left( \left\lceil \frac{m}{n} \right\rceil M(n'/2) + \left\lceil \frac{n}{m} \right\rceil M(m'/2) \right) \right).$$

\[\square\]

5 Time complexity analysis of the fast $D$-discrete matrix-vector multiplication algorithm.

In this section we prove the time complexity analysis of the algorithm presented in Section 5.2 in the paper.

**Theorem 5.1** Given an $n \times m$ $D$-discrete matrix $A$ and an $m$-length $D$-discrete vector $x$, Fast $D$-discrete Matrix-Vector Multiplication $A \otimes x$ can be computed in $O\left(\frac{mn}{\log n}\right)$ time.

**Proof.** Choose $q = \frac{\log |D|}{n} = O(\log n)$ (note that $|D|$ is a constant which is independent of $n$). For simplicity of the presentation, we assume that $m = O(n)$, and that $q$ divides both $n$ and $m$. Note that the number $|D|^{q-1}$ of difference sequences $\Delta(x)$ of length $q-1$ satisfies $|D|^{q-1} = \sqrt{n}/|D|$. Under the RAM model assumptions, we may assume that representing an integer in the range $[0,O(n)]$ requires a constant amount of space, and that reading and writing an integer in this range, as well as accessing an entry in a table according to an index in this range, can be performed in constant time.

There are $nm/q^2$ blocks $B = A_{Q_i}Q_i$ in the decomposition of $A$. Each computation of a table $MUL_B$ requires to perform the multiplication of $B$ with all $O(\sqrt{n})$ canonical $q$-length vectors, where the time for computing each such multiplication is $O(q^2)$. Thus, the overall time for computing all lookup tables $MUL_B$ is $O(mn^{1.5})$.

In the lookup table $MIN$, there are $O(q|D|^{q^2}) = O(n\log n)$ entries, each is computed in $O(q) = O(\log n)$ time, thus its computation requires $O(n\log^2 n)$ time. Therefore, the overall processing time of $A$ is $O(mn^{1.5} + n\log^2 n) = O(mn^{1.5})$.

When computing a multiplication $A \otimes x$, the algorithm first computes all $\Delta$-encodings of $q$-length sub-vectors $x_{Q_i}$ of $x$. This can be done in $O(m)$ time, in a straightforward manner. Then, the algorithm computes independently $O(n/q)$ sub-vectors $y_{Q_i}$ in the result. In each computation of a sub-vector $y_{Q_i}$, there are $O(m/q)$ computations of the form $A_{Q_i}Q_i \otimes x_{Q_i}$, as well as $O(m/q)$ applications of the $\oplus$ operator over intermediate computed $D$-discrete $q$-length vectors. As we described, each such computation is implemented by performing a constant number of operations (including one lookup table query), and thus the overall time for computing $A \otimes x$ is $O(mn/q^2) = O\left(\frac{mn}{\log^2 n}\right)$. 

The algorithm has an additional valuable property: if $A$ is extended by adding columns (or rows), corresponding additional lookup tables $MUL_B$ can be computed in time proportional only to the amount of added data. After each addition of $q$ columns, the algorithm computes additional $n/q$ tables $MUL_B$, in $O(qn^{1.5})$ time. This time is less than the time spent on matrix-vector multiplications. This property allows for the incremental approach of the online EDDC algorithm described Section 5.1 in the paper. \[\square\]
6 Online weighted CFG parsing

In this section we give a detailed description of the algorithm outlined in Section 5.1 in the paper.

The standard algorithm for solving the Weighted CFG Parsing problem is a modification by Teitelbaum [7] to the well known CKY algorithm [4, 5, 9]. For an input string \( s \) of length \( n \), parsed according to a Chomsky normal form grammar with \( N \) non-terminals, the algorithm maintains \( N \) matrices, each of size \( n \times n \). Each non-terminal \( X \) in the grammar has a corresponding matrix \( \hat{X} \), where an entry \( \hat{X}_{i,j} \) reflects the minimum weight of a parse tree of \( s_{i,j} \) in the grammar, given that the root node in the tree is \( X \). Note that only cases where \( i \leq j \) correspond to valid substrings of the input, thus all matrices \( \hat{X} \) are upper-triangle matrices.

As explained in [10], the computation of the best derivation of \( s_{i,j} \) from a rule of the form \( X \rightarrow YZ \) can be expressed as a vector multiplication of the form \( \hat{Y}_{i,[i+1,j-1]} \odot \hat{Z}_{[i+1,j-1],j} \). Following Valiant’s algorithm for the non-weighted version of the problem [8], Akutsu [1] has explained how to reduce the amortized time \( n \log(n^2) \) for enumerated \(\text{MIN}\) algorithm. As in Section 5.2 in the paper, the algorithm begins by creating the data structure necessary to directly apply the matrix-vector multiplication approach described there, due to the fact that the multiplied vectors consist of entries of column \( j \) in the matrices, that were not computed yet at the beginning of the stage in which the \( j \)th letter of \( t \) is added. Nevertheless, this difficulty can be overcome as follows.

We assume that the final length \( n \) of the input string is known at the beginning of the run of the algorithm. As in Section 5.2 in the paper, the algorithm begins by creating the data structure \( MIN \) (entry-wise minimum), for enumerated \( q \)-length vectors, where \( q = \frac{\log(n)}{2} \) (in \( O(n \log^2 n) \) time). It then receives the letters of the input one by one. Whenever a letter \( s_j \) is obtained, the algorithm computes the \( j \)th column in all matrices \( \hat{X} \). The algorithm partitions, in an online manner, the columns and rows of the matrices into \( q \) sized intervals denoted \( Q_0, Q_1, ..., Q_p \). In addition, whenever \( q \) computed columns of an interval \( Q_p \) accumulate, the algorithm computes lookup table \( MUL_B \), as described in Section 5.2 in the paper, for blocks \( B = \hat{X}_{Q_r,Q_p} \) such that \( r < p \) for all \( \hat{X} \).

Now, consider the computation of the \( j \)th column in all matrices \( \hat{X} \). For every \( 0 < i \leq j \) and every rule of the form \( X \rightarrow YZ \), the algorithm has to compute a vector multiplication of the form

\[
\hat{X}_{i,j} = \hat{Y}_{i,[i+1,j-1]} \odot \hat{Z}_{[i+1,j-1],j}.
\]  

We show how to use multiplication of blocks of size \( q \) for most of the computation of Eq. 6.1. Let \( p \) be the column block index of the block that includes \( j \) (i.e. \( p = \lfloor j/q \rfloor \)) and let \( r - 1 \) be the row block index of the block that includes \( i \) (i.e. \( r - 1 = \lfloor i/q \rfloor \)). Then the vector multiplication of Eq. 6.1 can be written as follows (Fig. 2):

\[
\hat{X}_{i,j} = \left( \hat{Y}_{i,[i+1,qr-1]} \odot \hat{Z}_{[i+1,qr-1],j} \right) \oplus \left( \hat{Y}_{i,[qr,qp-1]} \odot \hat{Z}_{[qr,qp-1],j} \right) \oplus \left( \hat{Y}_{i,[qp,j-1]} \odot \hat{Z}_{[qp,j-1],j} \right).
\]  

For a given interval \([qr, qp-1]\), it is possible to compute \( \left( \hat{Y}_{i,[qr,qp-1]} \odot \hat{Z}_{[qr,qp-1],j} \right) \) for all \( i \) in a given interval \( Q_{r-1} \) by using the following matrix-vector multiplication:

\[
\left( \hat{Y}_{Q_{r-1},[qr,qp-1]} \odot \hat{Z}_{[qr,qp-1],j} \right).
\]  

The algorithm computes the \( j \)th column block by block, bottom-up. The common part for all entries in interval \( Q_{r-1} \) is computed once according to Eq. 6.3. Then, for cells within \( X_{Q_{r-1},j} \), the entries are computed one by one bottom-up, as follows. In addition to the common computation of Eq. 6.3, the remaining computation of Eq. 6.2, that is, the partial blocks multiplications \( \hat{Y}_{i,[i+1,qr-1]} \odot \hat{Z}_{[i+1,qr-1],j} \) and \( \hat{Y}_{i,[qp,j-1]} \odot \hat{Z}_{[qp,j-1],j} \) are directly computed in \( O(q) \) time. Finally, the algorithm updates the \( i \)th entries in interval \( Q_{r-1} \) of the \( j \)th column of matrices \( \hat{X} \) according to rules of the form \( X \rightarrow Y \), if such exist, by checking for a better value originating from an entry of matrix \( \hat{Y} \) with the same indices.
Fig. 2: Fast on-line weighted CFG parsing. For simplicity, the figure presents the computation of the best derivation of a substring $s_{i,j}$ according to a rule of the form $X \rightarrow XX$. For rules of the form $X \rightarrow YZ$, the entries which participate in the computation are taken from three different matrices, rather than from a single one.

**Time Complexity Analysis.** The multiplication of Eq. 6.3 can be computed in $O(n/q)$ time, as explained in Section 5.2 in the paper and in Section 5 here. Recall that there are $R$ such vector multiplications to perform, thus the computation of all $q$ entries in the $j$th column of all matrices $X$ and the row interval $Q_{r-1}$ can be implemented in $O(R(n/q + q^2)) = O(Rn/q)$ time. Starting from $r = p + 1$ and decreasing to $r = 1$, these intervals of the $j$th columns are computed, where the entries in each interval are computed by decreasing row index. Observe that all needed values are available once the algorithm computes some entry, thus, the computation can be conducted correctly. Since there are $O(n/q)$ such intervals, and there are $O(Nn)$ entries in the $j$th columns of all matrices, the time for computing a column in all the matrices is $O(Nn + Rn^2/q^2) = O(Nn + Rn^2/\log^2 n)$. The overall running time for computing all $n$ columns is therefore $O(Nn^2 + Rn^3/\log^2 n)$.

7 Additional figures

Fig. 3: Vector multiplication calculation of $TD[i,j]$, as described in Section 3.2 in the paper in Eq. 3.2.
Fig. 4: Matrix-vector multiplication calculation of the $j$th column of $TD^\alpha$, as described in Section 5.1 in the paper, in Eq. 5.1.

(a) Case 1: $i$ and $j$ are starts of runs in $s$ and in $t$.

(b) Cases 2 and 3: In Case 2, $i$ starts a run in $s$ and $j$ does not start a run in $t$. Case 3 is symmetric. The $j$th letter either mutates to $\alpha$ or is generated from the adjacent letter.

(c) Case 4: $i$ and $j$ are not starts of runs in $s$ and $t$. Either the $i$th letter of $s$ and the $j$th letter both mutate to $\alpha$ or are generated from their adjacent letters.

Fig. 5: An illustration of the cases handled in the run-length recurrence appearing in Eq. 4.2 given in Section 4.1 in the paper.
Fig. 6: An illustration of the matrix-vector multiplication procedure, described in Section 5.2 in the paper. Each one of the \( n/q \) intervals of length \( q \) in the result vector is computed separately. This computation involves the multiplication of \( m/q \) blocks of size \( q \times q \) with \( q \)-length vectors (in this example, \( m/q = 5 \)). Due to the pre-processed look-up tables, each block-vector multiplication takes a constant time, as well as the accumulation of its result to the current intermediate computed \( q \)-length vector via \( \oplus \) operations. Thus the computation of a \( q \)-length vector takes \( O(\frac{m}{q}) \) time, and the computation of all \( n/q \) such intervals of the result vector takes \( O(nm/q^2) = O(n^2/\log^2 n) \).

References