Formalization of Block Pruning: Reducing the Number of Cells Computed in Exact Biological Sequence Comparison Algorithms

Edans F. O. Sandes[1], George L. M. Teodoro[1], Maria Emilia M. T. Walter[1], Xavier Martorell[2], Eduard Ayguade[2] and Alba C. M. A. Melo[1]

[1]*Department of Computer Science, University of Brasilia (UnB)*
Predio CIC-EST, Campus UNB, Asa Norte, 70910-900, Brasilia, Brazil
[2]*Department of Computer Architecture, Universitat Politecnica de Catalunya (UPC)*
C. Jordi Girona 1-3, Universitat Politecnica de Catalunya (UPC), 08034 Barcelona, Spain
Email: edans.sandes@gmail.com, {teodoro, mia}@cic.unb.br, {eduard, xavim}@ac.upc.edu, albamm@cic.unb.br

Biological sequence comparison algorithms that compute the optimal local and global alignments calculate a dynamic programming (DP) matrix with quadratic time complexity. The DP matrix H is calculated with a recurrence relation in which the value of each cell $H_{i,j}$ is the result of a maximum operation on the cells' values $H_{i-1,j-1}$, $H_{i-1,j}$ and $H_{i,j-1}$ added or subtracted by a constant value. Therefore, it can be noticed that the difference between the value of cell $H_{i,j}$ being calculated and the values of direct neighbor cells previously computed respect well-defined upper and lower bounds. Using these bounds, we can show that it is possible to determine the maximum and minimum value of every cell in H, for a given reference cell. We use this result to define a generic pruning method which determines the cells that can pruned (i.e. no need to be computed since they will not contribute to the final solution), accelerating the computation but keeping the guarantee that the optimal result will be produced. The goal of this paper is thus to investigate and formalize properties of the DP matrix in order to estimate and increase the pruning method efficiency. We also show that the pruning efficiency depends mainly on three characteristics: (a) the order in which the cells of H are calculated, (b) the values of the parameters used in the recurrence relation and (c) the contents of the sequences compared.

Keywords: Dynamic Progamming, Biological Sequence Comparison Algorithms

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

Pairwise Biological Sequence Comparison is a core operation in Bioinformatics, executed several times daily in research and industrial laboratories all over the world. Sequence comparison algorithms generate alignments which indicate regions of similarity between the sequences. These similarity regions are very important since they are often used by biologists to infer functional, structural or evolutionary relationships between the organisms.

The exact algorithms that compute optimal global and local sequence alignments were proposed by Needleman-Wunsh (NW) [1] and Smith-Waterman (SW) [2]. Both algorithms receive as input sequences S_0 and S_1 and compute a two-dimensional dynamic programming matrix H in which the value of cell $H_{i,j}$ depends on the value of three previously calculated adjacent cells: $H_{i-1,j-1}$, $H_{i-1,j}$ and $H_{i,j-1}$. NW and SW consider three possibilities of pairing (matches, mismatches, gaps) and they both execute in two phases. In phase 1, the whole matrix H is calculated and the optimal score is obtained whereas in phase 2 the optimal alignment is retrieved. The time and space complexity of both algorithms is $O(mn)$, where m and n are the sizes of sequences S_0 and S_1, respectively. NW and SW assume a linear gap model in which each gap receives the same penalty.

Gotoh [3] proposed an optimal global alignment algorithm that takes into account a more refined gap model called affine gap, using three DP matrices. In the literature, there are algorithms such as Hirschberg [4] and Myers-Miller (MM) [5] that compute optimal alignments in linear space by recursively obtaining the points that belong to the optimal alignment. The data dependency of the Gotoh, Hirshberg and MM algorithms is the same as the one expressed by NW and SW.

In the literature, there have been many efforts to reduce the number of DP cells calculated in biological sequence comparison problems. Most of these efforts were applied to the edit distance problem (ED) and its maximization counterpart, Longest Common Subsequence (LCS). These problems are formulated in such a way that, in the minimization problem (ED), the penalties for gaps and mismatches is +1 and the value for matches is +0. Therefore, the score computed for each DP cell $H_{i,j}$ differs from its direct neighbours $H_{i-1,j-1}$, $H_{i-1,j}$ and $H_{i,j-1}$ in at most +1 and the values in each diagonal never decrease. Based on this observation, Ukkonnen [6] defined properties for the ED and LCS DP matrices and these properties were later used by Landau et al. [7] to define a pruning strategy for the ED or LCS matrices with a bounded number of differences between the sequences. The ideas of [7] were then used in [8], [9] and [10]. Even though the work of Landau et al. [7] is able to prune the DP matrix in an interesting way, it relies on the edit distance formulation, which imposes severe restrictions on the values that a DP cell may assume. In the present paper, we assume that the value of a cell $H_{i,j}$ can be added or subtracted to/from the values of its neighbor DP cells and, thus, the monotonicity property explored by Landau et al. does not hold. Consequently, Landau's pruning strategy [7] and its further developments [8, 9, 10] cannot be applied to the NW or SW DP matrices.

Defining pruning strategies for the NW and SW recurrence relations is a more challenging problem and there are only a few pruning strategies proposed in the literature that can be applied to these cases. Fickett [11] proposed an algorithm that computes only a band of size k that encompasses the main diagonal of H and some diagonals nearby, with time complexity of $O(kn)$. The size of band k is defined empirically and the band must be enlarged if k diagonals do not contain the whole optimal alignment [11]. In this case, the DP matrix H is recomputed with the new band.

The LBD-Align algorithm [12] calculates optimal global alignments in linear space with pruning capabilities using a modified version of Hirschberg's [4]. In LBD-Align, the DP matrix is calculated diagonal by diagonal and two pruning tests are made in each diagonal, considering lower and upper bounds estimates. The prunability tests are made only in the first and last cells on each diagonal (pruning frontier), producing a window that is adjusted during the computation. The DP cells that are outside this window are not calculated, accelerating the computation. The lower bound is a score which is less than or equal to the optimal score. The author proposes two heuristics to compute reasonable lower bounds: diagonal and greedy-triangular. It is also mentioned that an heuristic local alignment algorithm such as BLAST [13] may be used to obtain the lower bound. The upper bound can be found by assuming that the characters of the sequences are the same, i.e., a perfect match case.

Block pruning is a pruning strategy proposed in CUDAlign 2.1 [14] and further used in SW# [15] and [16]. It calculates optimal local sequence alignments with the affine gap model using GPU (Graphics Processing Unit). The DP matrix is processed by blocks of diagonals and, instead of testing each diagonal as in LBD-Align, CUDAlign makes the pruning test for a block of diagonals, with very low overhead. Block pruning also defines lower and upper bounds, which are adjusted each time a block of diagonals is calculated. Block pruning was modified in MASA [17] to work with different ways of processing the DP matrix and with different devices (GPU, CPU and Intel Phi). We noticed in this work that the order in which the cells of the DP matrix are processed has a significant impact on the pruning results. Nevertheless, these two previous papers on block pruning did not provide a detailed formalization nor a detailed analysis of the block pruning technique.

Investigating thoroughly the problem, we observed that the value of cell $H_{i,j}$ differs from the values of its adjacent cells by at most and at least a constant value, due to the recurrence relations. Using this observation and having the value of a reference cell, we noticed that it is possible to determine superior and inferior bounds to the values of cells in H which not have been calculated yet in a quite accurate way and this is the first contribution of this paper.

The second contribution of this paper is a generic pruning method based on the superior and inferior bounds given by the first contribution. With the generic pruning, cells that cannot contribute to the optimal alignment are not calculated. We show that, even though many blocks of cells of H may be pruned, it is guaranteed that the optimal alignment will be produced.

Analyzing in depth the proposed pruning method, we observed that three characteristics have significant influence on the pruning ratio: (a) the contents of the sequences compared, (b) the order in which the cells of the DP matrix are calculated, and (c) the values of the parameters used in the recurrence relation. This is the third contribution of our paper, where we present several simulation studies showing the impact of the variation on each characteristic on the pruning ratio.

Finally, we present experimental results with real DNA sequences retrieved from NCBI

(*https://www.ncbi.nlm.nih.gov*), whose sizes vary from 50,999 BP (Base Pairs) to 1,043,007 BP. With these experiments, we show that, for the sequences compared, real pruning results are very close to the predicted ones, with a maximum error of 1.09 percentage point. We also show that the reduction in the area processed of the DP matrix corresponds to almost the same reduction in execution time.

The remainder of this paper is organized as follows. In section 2 we present the pairwise sequence comparison algorithms considered in this paper. Section 3 proposes the upper and lower bounds for the values of cells in the DP matrix H, using a given reference cell. In section 4, we propose our generic block pruning method and section 5 presents the impact of some characteristics of the problem/sequences on the pruning ratio. In section 6, five different pruning scenarios are discussed. Section 7 presents experimental results with real DNA sequences. Finally, section 8 concludes the paper and outlines future work.

2. EXACT BIOLOGICAL SEQUENCE COMPARISON

Biological sequences are DNA, RNA or protein sequences which are treated as strings composed of elements of the alphabets $\Sigma = \{A, T, G, C\}$, $\Sigma = \{A, U, G, C\}$ and $\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\}$, respectively. Protein and RNA sequences are rather small and their sizes range from hundreds to tens of thousands of characters. On the other hand, DNA sequences can be very long, often composed of millions of characters (Millions of Base Pairs - MBP). Without loss of generality, we assume in this paper that DNA sequences are compared.

The goal of pairwise biological sequence comparison is to find an alignment between the sequences, placing one sequence above the other and making clear the correspondence between the characters [18]. In the alignment, spaces (*gaps*) may be introduced in one of the sequences, in order to improve the alignment quality. Each alignment has a score, which measures the similarity between the sequences. The goal of exact pairwise biological sequence comparison algorithms is to obtain the optimal alignment, which is the alignment with the highest score.

There are two basic types of comparisons: (a) global, where all the characters of the sequences belong to the alignment; and (b) local, where a subset of the characters belongs to the alignment. Depending on the analysis, the biologists may choose among the types of the sequences (DNA, RNA, protein), the comparison type (local, global) and the output produced (score, score and alignment).

2.1. Algorithms

Needeleman-Wunsh (NW) [1] proposed a Dynamic Programming (DP) algorithm to obtain the optimal global alignment in quadratic time and space. The algorithm receives sequences S_0 and S_1, with sizes $|S_0| = m$ and $|S_1| = n$, respectively, and executes in two phases: computation of the DP matrix H and alignment retrieval (traceback).

In phase 1, the DP matrix H is computed. Each matrix cell $H_{i,j}$ contains the score of the alignment of prefixes $S[0..i]$ and $S[0..j]$. The first row and column of H are filled with $-G * i$ and $-G * j$ for every $H_{i,0}, 0 \leq i \leq n$ and $H_{0,j}, 0 \leq j \leq m$, respectively, where G is the penalty for one gap. The remaining cells $H_{i,j}$ are computed with Equation 1, in which $p(i,j)$ is calculated as follows. If $S_0[i] = S_1[j]$ then ma else $-mi$, in which ma and mi are respectively the values assigned for match and mismatch. Each cell keeps an indication of the cell that was used to produce the value (arrows in Figure 1). The optimal global score is the value contained in $H_{m,n}$.

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + p(i,j) \\ H_{i,j-1} - \mathrm{G} \\ H_{i-1,j} - \mathrm{G} \end{cases} \quad (1)$$

The second phase is responsible for retrieving the alignment. A traceback procedure is executed from the bottom right cell, following the arrows until the top left cell is reached, as shown in Figure 1 (a). In the figure, the DP matrix is shown on top and the alignment is shown at the bottom.

The algorithm Smith-Waterman (SW) [2] is used to obtain the optimal local alignment. It is similar to NW, with three differences. First, in phase 1, the first row and column are filled with zeroes. Second, also in phase 1, Equation 2 is used to compute the cells. Third, in phase 2, the traceback begins in the cell that has the optimal local score (highest value in H) and stops when a zero-valued cell is reached (Figure 1 (b)).

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + p(i,j) \\ H_{i,j-1} - \mathrm{G} \\ H_{i-1,j} - \mathrm{G} \\ 0 \end{cases} \quad (2)$$

NW and SW assign a constant cost G to gaps. However, gaps tend to occur in groups. So, a different approach called affine-gap model [19] associates a higher penalty to the first gap and a lower penalty to the remaining ones. Gotoh [3] proposed a DP-based algorithm to compute alignments with the affine-gap model. In this algorithm, 3 DP matrices are calculated: H, E and F (Equations 3, 4 and 5), where E and F keep track of gaps in each sequence.

$$H_{i,j} = \max \begin{cases} 0 \\ E_{i,j} \\ F_{i,j} \\ H_{i-1,j-1} + p(i,j) \end{cases} \quad (3)$$

$$E_{i,j} = \max \begin{cases} E_{i,j-1} - G_{ext} \\ H_{i,j-1} - G_{first} \end{cases} \quad (4)$$

$$F_{i,j} = \max \begin{cases} F_{i-1,j} - G_{ext} \\ H_{i-1,j} - G_{first} \end{cases} \quad (5)$$

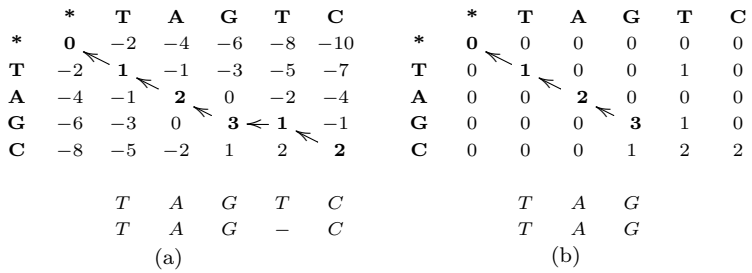


FIGURE 1. DP matrices and alignments for sequences S_0 and S_1 (mi=1, ma=1, G=2).

TABLE 1. Notations used in this paper

Symbol	Description
S_0,S_1	Input Sequences
m, n	Sizes of Sequences S_0 and S_1
$H_{i,j}$	DP cell
$H_{i-1,j-1}$, $H_{i-1,j}$, $H_{i,j-1}$	Adjacent DP Cells
$\mathcal{G}$	Traceback Graph
(*diag*), (*up*), (*left*)	Dependency Cases
$H^{\star}_{i,j}$	Reference DP Cell
Δ_i and Δ_j	Vertical and Horizontal Distances
$H_{i+\Delta_i,j+\Delta_j}$	Displaced DP Cell
$\bar{H}_{i+\Delta_i,j+\Delta_j}$	Connected DP Cell
$\delta_{i,j}$	Difference Between DP Cells
$\bar{\delta}_{i,j}$	Difference Between Connected DP Cells
$H^{max}_{i,j}$	Maximum Derived Score
$H^{min}_{i,j}$	Minimum Derived Score
$best_{i,j}$	Best Provisory Score
$\phi_{i,j}$	Iteration

The data dependency of these algorithms is the same, i. e., the value of cell (i, j) depends on previously calculated values $(i-1, j-1)$, $(i-1, j)$ and/or $(i, j-1)$.

3. UPPER AND LOWER BOUNDS FOR CELL $H_{I,J}$

The goal of this section is to define upper and lower bounds for the values of a cell in the DP matrix, using the value and coordinates of a given reference cell. In subsection 3.1, we will present the notations used in this paper and provide some definitions. In subsection 3.2, we present the lower and upper bounds on the values of two connected DP cells in H, computed with the NW equation (Equation 1). Subsection 3.3 presents the bound for any two cells, connected or not. In subsection 3.4 we show how the bounds defined for NW can be adjusted to the SW (Equation 2) and Gotoh equations (Equations 3, 4 and 5). Finally, subsection 3.5 presents the maximum and minimum derived scores, which are obtained with the bounds previously defined.

3.1. Notations and Definitions

Table 1 presents the notations used in this section. Without loss of generality, we will first consider the NW equation and then the SW and Gotoh equations.

In the following paragraphs, we will explain the 16 symbols shown in Table 1, representing each symbol in italic.

Sequences S_0 and S_1, with sizes m and n, respectively, will be compared with the NW algorithm, generating a DP matrix. Each matrix entry is a DP cell $H_{i,j}$ which has three adjacent DP cells: $H_{i-1,j-1}$, $H_{i-1,j}$ and $H_{i,j-1}$. These cells represent the dependency pattern expressed by Equation 1.

The traceback graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where each node $\mathcal{V}$ represents some DP cell and the edges $\mathcal{E}$ indicate which adjacent cell(s) produced the *max* clause results in the recurrence relation (Equation 1). Figure 2 presents the traceback graph for sequences TTACACACTT and TGCACACAGG, with $G = 2$, $ma = 1$ and $mi = 1$. It can be noticed that each cell has 1, 2 or 3 edges, with the exception of cell $H_{0,0}$, which has no dependency. The edges in bold represent the optimal global alignments.

Since the value of cell $H_{i,j}$ is the maximum value calculated with one or more adjacent cells, there are three dependency cases *(diag, up, left)* as shown in Figure 3. Each dependency case which generates the cell value is an edge in the traceback graph. It must be noted that more than one dependency case can generate the same cell value.

Now, we will distinguish some types of cells that will be used in the explanation. A reference DP cell $H^{\star}_{i,j}$ is a DP cell for which the value has already been calculated and will be used as a reference. The vertical and horizontal distances (Δ_i and Δ_j, respectively) between a given Displaced DP cell $H_{i+\Delta_i,j+\Delta_j}$ and the reference cell $H^{\star}_{i,j}$ are the minimum number of nodes which will be traversed from $H^{\star}_{i,j}$ to $H_{i+\Delta_i,j+\Delta_j}$, in the horizontal or vertical directions, respectively. We say that a displaced cell $\bar{H}_{i+\Delta_i,j+\Delta_j}$ is connected with a reference cell $H^{\star}_{i,j}$ if there is a directed path between them in the traceback graph. Figure 2 illustrates with a rectangle the cells $H_{0,0}$ and $H_{m,n}$, which are connected cells.

Given a reference cell $H^{\star}_{i,j}$ and a displaced DP cell $H_{i+\Delta_i,j+\Delta_j}$, $\delta_{i+\Delta_i,j+\Delta_j}$ is the difference between the values of these DP cells, as shown in Equation 6. Similarly, $\bar{\delta}_{i+\Delta_i,j+\Delta_j}$ is the difference between the values of these DP cells if $H^{\star}_{i,j}$ and $\bar{H}_{i+\Delta_i,j+\Delta_j}$ are connected.

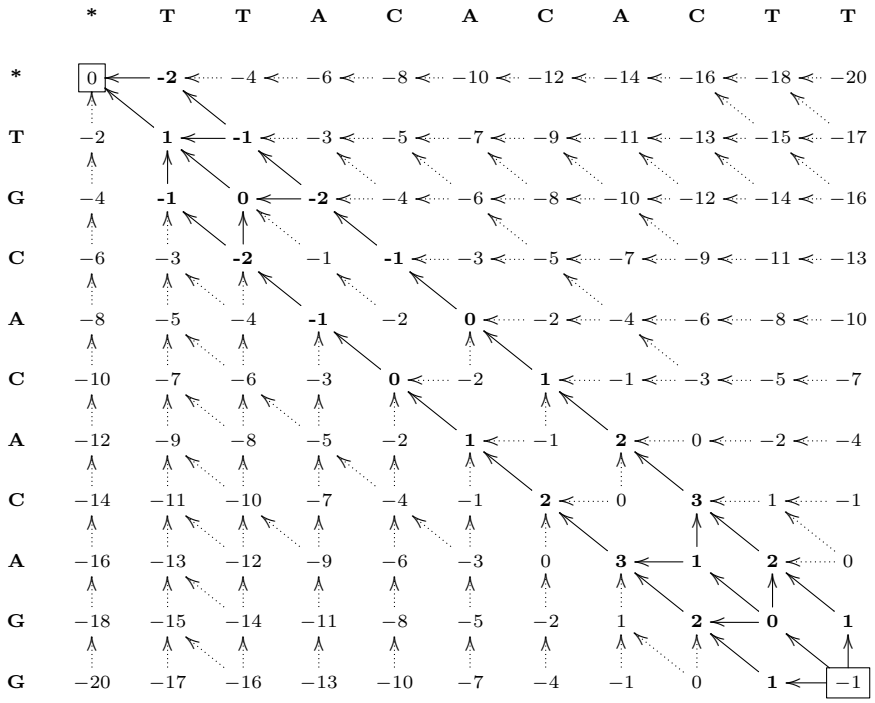


FIGURE 2. Traceback graph.

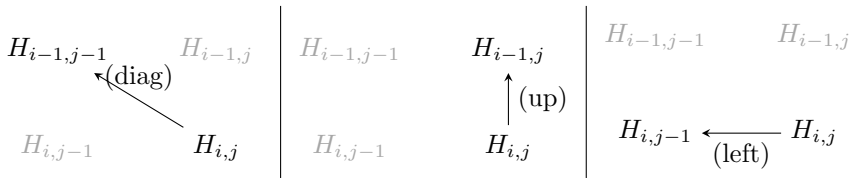


FIGURE 3. Dependency cases for $H_{i,j}$.

3.2. Bounds on the Difference Between Values of Connected Cells ($\bar{\delta}$)

$$\delta_{i+\Delta_i,j+\Delta_j} = H_{i+\Delta_i,j+\Delta_j} - H^{\star}_{i,j} \qquad (6)$$

Assuming that the value of a given cell $H_{i,j}$ is known, the maximum derived score $H^{max}_{i,j}$ is the highest possible score for the optimal alignment if it crosses cell $H_{i,j}$. Analogously, the minimum derived score $H^{min}_{i,j}$is the lowest possible score for the optimal alignment if it crosses cell $H_{i,j}$. The best provisory score is defined as $best_{i,j}$ and it is the highest value among the minimum derived scores $H^{min}_{i',j'}$ of all cells $H_{i',j'}$, which were calculated before calculating $H_{i,j}$. At the end of the matrix computation, the value $best_{m,n}$ is the best real score (optimal score).

Finally, assuming that the DP matrix is calculated iteratively, we will call $\phi_{i,j}$ the iteration in which cell $H_{i,j}$ was calculated. Therefore, $\phi_{i',j'} < \phi_{i,j}$ if $H_{i',j'}$ was calculated in a previous iteration than the iteration in which $H_{i,j}$ was calculated.

For adjacent cells which are connected, we calculate the difference $\bar{\delta}$ between the values of the cells using Equation 1, i.e., $\bar{\delta}_{i-1,j-1} = +mi$ or $-ma$, $\bar{\delta}_{i-1,j} = +G$ and $\bar{\delta}_{i,j-1} = +G$ (Figure 4).

In order to calculate the maximum difference between DP connected cells which are not adjacent, we may consider all paths in the traceback graph that connect these cells. Due to the characteristics of Equation 1, a path between two DP cells $H_{i,j}$ and $H_{i',j'}$ only exists if ($i \leq i'$ and $j \leq j'$) or ($i \geq i'$ and $j \geq j'$). Note that the case in which ($i = i'$ and $j = j'$) must be excluded. Table 2 presents the maximum values for $\bar{\delta}$ for the neighborhood of a reference cell $H^{\star}_{i,j}$, marked in the table by the symbol $\star$.

Inequality 7 generalizes the maximum difference $\bar{\delta}_{i+\Delta_i,j+\Delta_j}$ between the values of reference DP cell $H^{\star}_{i,j}$ and a given DP Cell $H_{i+\Delta_i,j+\Delta_j}$ (Section 3.1).

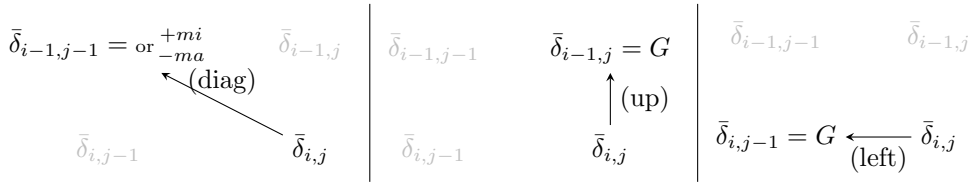


FIGURE 4. Difference $\bar{\delta}$ between adjacent DP connected cells.

TABLE 2. Maximum Difference $\bar{\delta}$ Between Connected Cells.

		$\leftarrow \Delta_j \rightarrow$						
	$\cdots$	-3	-2	-1	0	$+1$	$+2$	$+3$
	-3	$+3mi$	$+2mi{+}1G$	$+1mi{+}2G$	$+3G$	$--$	$--$	$--$
	-2	$+2mi{+}1G$	$+2mi$	$+1mi{+}1G$	$+2G$	$--$	$--$	$--$
$\uparrow$	-1	$+1mi{+}2G$	$+1mi{+}1G$	$+1mi$	$+1G$	$--$	$--$	$--$
Δ_i	0	$+3G$	$+2G$	$+1G$	$0\star$	$-1G$	$-2G$	$-3G$
$\downarrow$	$+1$	$--$	$--$	$--$	$-1G$	$+1ma$	$+1ma{-}1G$	$+1ma{-}2G$
	$+2$	$--$	$--$	$--$	$-2G$	$+1ma{-}1G$	$+2ma$	$+2ma{-}1G$
	$+3$	$--$	$--$	$--$	$-3G$	$+1ma{-}2G$	$+2ma{-}1G$	$+3ma$

$$\bar{\delta}_{i+\Delta_i,j+\Delta_j} \leq \begin{cases} -max(\Delta_i,\Delta_j)mi + |\Delta_i - \Delta_j|G\text{: if } \Delta_i \leq 0 \text{ and } \Delta_j \leq 0 \\ min(\Delta_i,\Delta_j)ma - |\Delta_i - \Delta_j|G\text{: if } \Delta_i \geq 0 \text{ and } \Delta_j \geq 0 \end{cases} \quad (7)$$

By symmetry, the minimum value of $\bar{\delta}_{i+\Delta_i,j+\Delta_j}$ is equal to the minimum value of $-\bar{\delta}_{i-\Delta_i,j-\Delta_j}$. Thus, the value $\bar{\delta}_{i+\Delta_i,j+\Delta_j}$ is defined by Inequality 8.

$$\bar{\delta}_{i+\Delta_i,j+\Delta_j} \geq \begin{cases} -min(\Delta_i,\Delta_j)mi - |\Delta_i - \Delta_j|G\text{: if } \Delta_i \geq 0 \text{ and } \Delta_j \geq 0 \\ max(\Delta_i,\Delta_j)ma + |\Delta_i - \Delta_j|G\text{: if } \Delta_i \leq 0 \text{ and } \Delta_j \leq 0 \end{cases} \quad (8)$$

As an example, suppose that $ma = 1$, $mi = 3$ and $G = 5$. In this case, a given DP cell $H_{i+\Delta_i,j+\Delta_j}$, connected to a reference cell $H^{\star}_{i,j}$ and placed $\Delta_i = 100$ rows below and $\Delta_j = 70$ columns to the right of $H^{\star}_{i,j}$ will have a difference of values in $-360 \leq \bar{\delta}_{i+\Delta_i,j+\Delta_j} \leq -80$. If we consider that the reference cell is $H^{\star}_{i,j} = 1000$, then the range of possible vaues lies in $640 \leq \bar{H}_{i+\Delta_i,j+\Delta_j} \leq 920$.

3.3. Bounds on the Difference Between Values of Any Two DP Cells

In this subsection, we determine inferior and superior bounds for the values of any DP cell, connected or not to a given reference cell. In order to do this, we first define the bounds for adjacent cells and then we present the limits for any two DP cells.

3.3.1. Superior Bound for Adjacent Cells

Applying directly the NW Equation (Equation 1), Inequalities 9, 10 and 11 are valid for the dependency cases (diag), (up) and (left).

$$H_{i,j} \geq H_{i-1,j-1} - mi \quad (9)$$
$$H_{i,j} \geq H_{i-1,j} - G \quad (10)$$
$$H_{i,j} \geq H_{i,j-1} - G \quad (11)$$

Considering Equation 6, the superior bounds for the difference between neighbor cells are thus $\delta_{i-1,j-1} \leq +mi$, $\delta_{i-1,j} \leq +G$ and $\delta_{i,j-1} \leq +G$ (Figure 5).

$\delta_{i-1,j-1} \leq +mi$	$\delta_{i-1,j} \leq G$
$\delta_{i,j-1} \leq G$	$\delta_{i,j} = 0$

FIGURE 5. Superior Bounds for Adjacent Cells

3.3.2. Inferior Bound for Adjacent Cells

In order to determine the inferior bounds, we rely on Theorem 3.1. It must be noted that, by definition, G, mi and ma have positive values. We will also assume that $mi < 2G$. If this condition does not hold, all *mismatches* would be replaced by two *gaps*, making therefore the *mismatch* case useless.

Theorem 3.1.
For any $H_{i,j}$ $(0 \leq i \leq m, 0 \leq j \leq n)$ in the DP matrix, the following 3 cases must hold:

$$\begin{array}{lllll} (a) & H_{i,j} & \leq & H_{i-1,j} & +ma + G \\ (b) & H_{i,j} & \leq & H_{i,j-1} & +ma + G \\ (c) & H_{i,j} & \leq & H_{i-1,j-1} & +ma \end{array}$$

The proof by induction of Theorem 3.1 is provided in Appendix A.

By Theorem 3.1 and Equation 6, we get Corollary 3.1.

Corollary 3.1. *The inferior bounds on the differences of adjacent cells are: $\delta_{i-1,j-1} \geq -ma$, $\delta_{i-1,j} \geq -ma - G$ and $\delta_{i,j-1} \geq -ma - G$.*

Figure 6 illustrates the inferior bounds of adjacent cells.

$\delta_{i-1,j-1} \geq -ma$	$\delta_{i-1,j} \geq -ma - G$
$\delta_{i,j-1} \geq -ma - G$	$\delta_{i,j} = 0$

FIGURE 6. Inferior Bounds of Adjacent Cells from Corollary (1)

3.3.3. Bounds on the differences on non-adjacent cells Using the bounds illustrated in Figures 5 and 6, we can find the maximum difference between two non-adjacent cells of the DP matrix. Table 3 presents the upper bounds on the differences of DP cells (adjacent or not), where $\star$ indicates the reference cell $H^{\star}_{i,j}$.

Inequality 12 presents the maximum difference $\delta_{i+\Delta_i,j+\Delta_j}$ between the values of the reference DP cell $H^{\star}_{i,j}$ and a given DP cell $H_{i+\Delta_i,j+\Delta_j}$, in which Δ_i and Δ_j are the vertical and horizontal distances to $H^{\star}_{i,j}$.

$$\delta_{i+\Delta_i,j+\Delta_j} \leq \begin{cases} -max(\Delta_i,\Delta_j)mi + |\Delta_i - \Delta_j|G\text{: if } \Delta_i < 0 \text{ and } \Delta_j < 0 \\ max(\Delta_i,\Delta_j)ma + |\Delta_i - \Delta_j|G\text{: if } \Delta_i \geq 0 \text{ or } \Delta_j \geq 0 \end{cases} \tag{12}$$

By symmetry, the minimum value $\delta_{i+\Delta_i,j+\Delta_j}$ is equal to $-\delta_{i-\Delta_i,j-\Delta_j}$. Thus, we express the minimum value $\delta_{i+\Delta_i,j+\Delta_j}$ with Inequality 13.

$$\delta_{i+\Delta_i,j+\Delta_j} \geq \begin{cases} -min(\Delta_i,\Delta_j)mi - |\Delta_i - \Delta_j|G\text{: if } \Delta_i > 0 \text{ and } \Delta_j > 0 \\ min(\Delta_i,\Delta_j)ma - |\Delta_i - \Delta_j|G\text{: if } \Delta_i \leq 0 \text{ or } \Delta_j \leq 0 \end{cases} \tag{13}$$

As an example, suppose that $ma = 1$, $mi = 3$ and $G = 5$. We can state that a DP cell $H_{i+\Delta_i,j+\Delta_j}$ which is $\Delta_i = 100$ rows below and $\Delta_j = 70$ columns to the right of a reference cell $H^{\star}_{i,j}$ will have a value whose difference is limited $-360 \leq \delta_{i+\Delta_i,j+\Delta_j} \leq 250$. If we consider that $H^{\star}_{i,j} = 1000$, then $640 \leq H_{i+\Delta_i,j+\Delta_j} \leq 1250$.

3.4. Bounds for SW and Gotoh

In Sections 3.2 and 3.3, we analyzed the maximum and minimum differences between cells of the DP matrix calculated with the NW equation (Equation 1). For the SW and Gotoh algorithms (Section 2.1), the analysis is very similar.

The SW equation (Equation 2) does not allow negative values ($H_{i,j} \geq 0$). Therefore, the difference between the reference cell $H^{\star}_{i,j}$ and a given DP cell $H_{i+\Delta_i,j+\Delta_j}$ will always be less or equal than the value of $H^{\star}_{i,j}$, as shown in Inequality 14.

$$\begin{aligned} \delta_{i+\Delta_i,j+\Delta_j} &= \underbrace{H_{i+\Delta_i,j+\Delta_j}}_{\geq 0} - H^{\star}_{i,j} \\ \delta_{i+\Delta_i,j+\Delta_j} &\geq -H^{\star}_{i,j} \end{aligned} \tag{14}$$

In order to apply the maximum and minimum differences to the Gotoh algorithm (Equations 3, 4 and 5), we must notice that the penalty for the first gap $\gamma(1) = G_{first}$ is greater than the penalty for the remaining gaps (i.e. $G_{first} > G_{ext}$). Thus, equations 7, 8, 12 and 13 are also valid for the Gotoh algorithm if we replace G by G_{first}. It must be noted that, in this case, the Gotoh bounds are less strict than the NW bounds.

3.5. Maximum and Minimum Derived Scores ($H^{max}_{i,j}$ and $H^{min}_{i,j}$)

In order to find the superior and inferior bounds on the scores of a given DP cell (Table 1), we will consider global and local alignments separately.

Global Alignment The optimal global alignment ends, by definition, in the last DP cell $H_{m,n}$. Using Equation 7, $H^{max}_{i,j}$ is defined for the global alignment by Equation 15.

$$\begin{aligned} H_{m,n} &= H_{i,j} + \bar{\delta}(m,n) \\ H_{m,n} &\leq \underbrace{H_{i,j} + min(m-i, n-j) \cdot ma - |(m-i) - (n-j)| \cdot G}_{=H^{max}_{i,j}} \end{aligned} \tag{15}$$

Analogously, with Inequality 8, $H^{min}_{i,j}$ is defined by Equation 16.

$$\begin{aligned} H_{m,n} &= H_{i,j} + \bar{\delta}(m,n) \\ H_{m,n} &\geq \underbrace{H_{i,j} - min(m-i, n-j) \cdot mi - |(m-i) - (n-j)| \cdot G}_{=H^{min}_{i,j}} \end{aligned} \tag{16}$$

Local Alignment By definition, the optimal local alignment can end in any DP cell. In this case, we need to consider the maximum possible value of each DP cell which can be connected to $H_{i,j}$. Using Inequality 7, $H^{max}_{i,j}$ is defined for the local alignment by Equation 17.

$$\begin{aligned} H^{max}_{i,j} &= H_{i,j} + \max_{(i,j) \leq (i',j') \leq (m,n)} \bar{\delta}(i',j') \\ H^{max}_{i,j} &= H_{i,j} + min(m-i, n-j) \cdot ma \end{aligned} \tag{17}$$

Since, in the local alignment, the optimal alignment may end in the DP cell $H_{i,j}$ which is being considered, then $H^{min}_{i,j}$ of an alignment that passes through this cell is $H_{i,j}$, as shown in Equation 18.

$$H^{min}_{i,j} = H_{i,j} \tag{18}$$

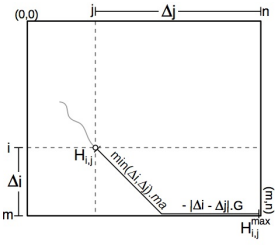
Figure 7 illustrates geometrically the values of $H^{max}_{i,j}$ and $H^{min}_{i,j}$ for local and global alignments. The lines leaving cell $H_{i,j}$ represent the best (Figures 7(a) and 7(c)) and worst (Figures 7(b) and 7(d)) alignment scenarios, with matches or mismatches between all the remaining characters of the sequences.

4. PRUNING METHOD

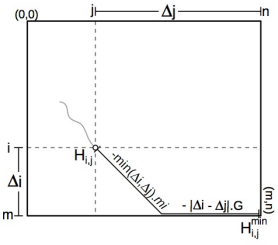
The pruning method proposed in this paper defines the conditions that determine if an entry in the DP matrix

TABLE 3. Maximum difference $\delta_{i+\Delta_i,j+\Delta_j}$ between DP cells.

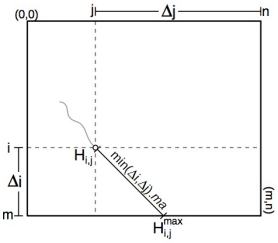
		$\leftarrow \Delta_j \rightarrow$						
	$\cdots$	-3	-2	-1	0	$+1$	$+2$	$+3$
	-3	$+3mi$	$+2mi{+}1G$	$+1mi{+}2G$	$+3G$	$+1ma{+}4G$	$+2ma{+}5G$	$+3ma{+}6G$
	-2	$+2mi{+}1G$	$+2mi$	$+1mi{+}1G$	$+2G$	$+1ma{+}3G$	$+2ma{+}4G$	$+3ma{+}5G$
$\uparrow$	-1	$+1mi{+}2G$	$+1mi{+}1G$	$+1mi$	$+1G$	$+1ma{+}2G$	$+2ma{+}3G$	$+3ma{+}4G$
Δ_i	0	$+3G$	$+2G$	$+1G$	$0\star$	$+1ma{+}1G$	$+2ma{+}2G$	$+3ma{+}3G$
$\downarrow$	$+1$	$+1ma{+}4G$	$+1ma{+}3G$	$+1ma{+}2G$	$+1ma{+}1G$	$+1ma$	$+2ma{+}1G$	$+3ma{+}2G$
	$+2$	$+2ma{+}5G$	$+2ma{+}4G$	$+2ma{+}3G$	$+2ma{+}2G$	$+2ma{+}1G$	$+2ma$	$+3ma{+}1G$
	$+3$	$+3ma{+}6G$	$+3ma{+}5G$	$+3ma{+}4G$	$+3ma{+}3G$	$+3ma{+}2G$	$+2ma{+}2G$	$+3ma$



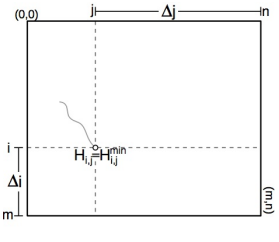
(a) $H^{max}_{i,j}$ for Global Alignment



(b) $H^{min}_{i,j}$ for Global Alignment



(c) $H^{max}_{i,j}$ for Local Alignment



(d) $H^{min}_{i,j}$ for Local Alignment

FIGURE 7. Geometric Representations for $H^{max}_{i,j}$ and $H^{min}_{i,j}$.

needs to be computed or can be skipped since it will not contribute to obtain the optimal alignment. In other words, a DP cell can be discarded if its maximum derived score $H^{max}_{i,j}$ (Section 3.5) is less than the inferior bound on the optimal alignment (*bound*), shown in Inequality 19, which is called **pruning condition**.

$$H^{max}_{i,j} \leq bound \tag{19}$$

The pruning condition can potentially reduce the number of DP cells calculated, reducing thus the execution time of the exact biological sequence comparison algorithms. In this section, we will first provide some definitions and then the algorithms that use the pruning condition. Finally, we present some formulae to estimate the effectiveness of the pruning algorithms.

4.1. Definitions

Definition 1 (Prunable Cell). If Inequality 19 holds for $H_{i,j}$, then $H_{i,j}$ is a prunable cell. The computation of a prunable cell could have been discarded since no optimal alignment will pass through it.

Definition 2 (Pruned Cell). A pruned cell $H_{i,j}$ is a cell which has been identified as prunable before its computation.

The verification of a pruned cell is made by analyzing its adjacent DP cells. If all the adjacent cells ($H_{i-1,j-1}$, $H_{i-1,j}$ and $H_{i,j-1}$) are prunable, then cell $H_{i,j}$ is also prunable and, in this case, it can be pruned. A special case occurs when the prunable cells are in the first row or column. If cell $H_{y,0}$ is prunable, all the remaining cells in which $H_{i>y,0}$ are also prunable and can be pruned. Analogously, if cell $H_{0,x}$ is prunable, all the remaining cells in which $H_{0,j>x}$ are also prunable. Figure 8 illustrates the pruning condition.

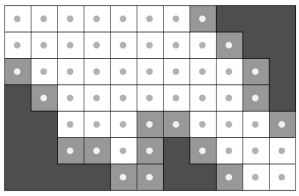


FIGURE 8. Pruning condition. The light gray cells represent the prunable cells. The dark gray cells represent the pruned cells. Cells with a circle in the center were calculated and the other cells (pruned cells) were discarded.

Definition 3 (Inferior Bound *bound*). The inferior bound is used in Equation 19. In our method, we will ignore alignments whose scores are less than *bound*.

Definition 4 (Initial Inferior Bound $bound_0$). The inferior *bound* has an initial value $bound_0$, which may be defined in the following ways:

- **Unrestricted**: In this case, $bound_0 = -\infty$. With this, all the possible alignments are considered in the beginning of the computation.
- **Manual**: The user defines that $bound_0$ is greater than a specific value. In this case, only alignments with score higher than this manually defined $bound_0$ are considered.
- **Heuristic**: An heuristic algorithm is used to find an alignment and the score of this alignment is set to $bound_0$.

- **Oracle**: The optimal score is known before the beginning of the computation and $bound_0$ is set to the optimal score.

4.2. Updating the inferior bound

The inferior bound (*bound*) can be updated with new values $bound_1, bound_2, \cdots$, in the following ways:

- **by cell**: The value of *bound* is updated with the highest minimum derived score $H_{i,j}^{min}$ (Equations 16 or 18) immediately after the computation of each cell.
- **by block**: This kind of update uses the same equations as in the cell update. However, the computation of the highest minimum derived score is applied only to the cell which has the highest score in the block. With this, we are able to considerably reduce the computation overhead.
- **by period of time**: The value of *bound* is updated by time intervals. The highest minimum derived score $H_{i,j}^{min}$ is calculated with the highest score found up to the moment.
- **by execution**: This type of update is applied when $bound_0$ is manually defined since, depending on the value specified by the user, the first execution of the algorithm may not find an optimal alignment. Thus, the DP matrix is computed again in possibly more than one execution until an alignment with a score greater than *bound* is found. In each execution, the value of *bound* is set to successively lower values ($bound_0 > bound_1 > bound_2 > \cdots$).

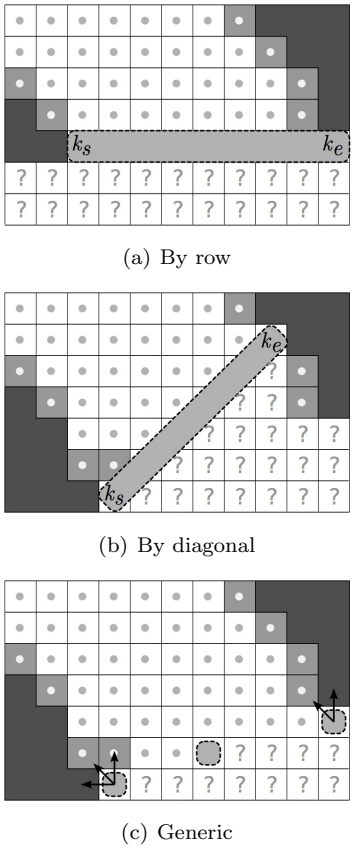
4.3. Pruning Algorithms

In this section we present three pruning algorithms (Algorithms 2 to 4). In the explanation, we will consider local alignments. The small modifications needed for global alignments were discussed in Section 3.4. The pruning algorithms are:

- (Algorithm 2) **Specific by row**: applied when the DP matrix is computed row by row or column by column. Memory complexity for this algorithm is constant $O(1)$.
- (Algorithm 3) **Specific by diagonal**: applied when the DP matrix is computed diagonal by diagonal. Memory complexity is constant $O(1)$.
- (Algorithm 4) **Generic linear**: applied to all possible ways of computing the DP matrix. Memory complexity is linear $O(m+n)$.

Figure 9 illustrates these three ways of computing the DP matrix.

In the beginning of the explanation, we will present a basic skeleton with characteristics which are common to the three algorithms. Then, the characteristics specific to each algorithm are presented.



(a) By row

(b) By diagonal

(c) Generic

FIGURE 9. Ways of Computing the DP Matrix with Pruning Algorithms

Basic Skeleton : Algorithm 1 presents the skeleton which will be used by all pruning algorithms (Algorithms 2 to 4). This skeleton is divided in four parts: a) initialization; b) function IsPrunable; c) function IsPruned; and d) function PruningUpdate.

Algorithm 1 Pruning Skeleton

```
1: bound ← −∞
2: ... Specific Initializations ...

3: function IsPrunable(i, j, H_{i,j}, bound)
4:     H^{max}_{i,j} ← H_{i,j} + min(m − i, n − j) · ma
5:     return H^{max}_{i,j} ≤ bound
6: end function

7: function isPruned(i, j)
8:     ... specific code ...
9: end function

10: procedure pruningUpdate(i, j, H_{i,j})
11:     ... specific code ...
12: end procedure
```

The initializations are made in the first lines of the algorithm. We assume, without loss of generality, that the initial inferior bound $bound_0$ (Definition 4) is initialized as $bound_0 = -\infty$. In algorithms 2

to 4 we can observe initializations of variables which occur exclusively in each algorithm. It is clear from these initializations that the memory complexity of the specific algorithms is $O(1)$ whereas the generic algorithm has linear memory complexity $O(m+n)$.

Function IsPrunable (lines 3-6), identifies if a cell is prunable or not (Definition 1). This function is used in all pruning algorithms and it implements Equations 17 and 19.

Function IsPruned is called before the computation of each cell and it identifies if the computation can be discarded (pruned) without compromising the optimal result (Definition 2). Each pruning algorithm will identify the pruned cells differently.

Function PruningUpdate is called after the computation of each cell. It updates variables used for pruning, including the variable *bound* (Definition 3). Each pruning algorithm has a different way of updating its variables.

4.3.1. Algorithm Specific by Row

Assuming that the DP matrix is calculated row by row (or analogously column by column), the pruning algorithm acts on pruning windows. We call non-prunable window the interval $[k_s..k_e]$ of columns which need to be processed in a given row. Initially, $(k_s, k_e) = (0, n)$. For each row i, the algorithm computes all cells in the non-prunable window $(i, j \in [k_s..k_e])$. Then, it updates the non-prunable window with values $[k'_s..k'_e]$, in which k'_s and k'_e are, respectively, the first and last non-prunable cells in this row.

Algorithm 2 Algorithm Specific by Row

```
1: (k_s, k_e) ← (0, n)

2: procedure PruningUpdate(i, H_{i,[0..n]})
3:     bound ← max(bound, H_{i,[0..n]})
4:     while k_s < n and IsPrunable(i, k_s, H_{i,k_s}, bound) do
5:         k_s ← k_s + 1        ▷ enlarging pruning area to the left
6:     end while
7:     if k_e < n and ¬IsPrunable(i, k_e, H_{i,k_e}, bound) then
8:         k_e ← k_e + 1
9:         while k_s < B and ¬IsPrunable(i, k_e, H_{i,k_e}, bound)
   do
10:            k_e ← k_e + 1  ▷ reducing pruning area to the right
11:        end while
12:    else
13:        k_e ← k_e − 1;
14:        while k_e ≥ k_s and IsPrunable(i, k_e, H_{i,k_e}, bound) do
15:            k_e ← k_e − 1 ▷ enlarging pruning area to the right
16:        end while
17:        k_e ← k_e + 1;
18:    end if
19: end procedure

20: function IsPruned(j)
21:    return j < k_s or j > k_e
22: end function
```

If cells $(i-1, j \in [0..k_s])$ are prunable, then all cells in $(i, j \in [0..k_s])$ are prunable and, by induction, all cells $(i' \geq i, j \in [0..k_s])$ are also prunable. If all cells $(i-1, j \in [k_e..n])$ are prunable and cell $(i, x \in [k_e..n])$ is prunable, then all cells $(i, j \in [x+1..m])$ will also be prunable. If cell (i, k_e+1) is prunable, then the new value for k'_e is the last non-prunable cell in interval $(i, j \in [k_s..k_e])$, in which $k'_e \leq k_e$. Otherwise, we need to compute the remaining cells of row i until the first prunable cell $(i, x \in [k_e+2..n])$ is found, in such a way that the new value of k'_e will be $x-1$. Cells in the interval $(i, j \in [0..k'_s])$ are then left-prunable cells and cells in the interval $(i, j \in [k'_e+1..n])$ are called right-prunable cells.

In Algorithm 2, function isPruned (lines 20-22) returns a boolean value indicating if column j is inside or outside the non-prunable window $[k_s..k_e]$. Function PruningUpdate (lines 2-19) updates the non-prunable window. Line 5 updates the value of k_s and lines 10 and 15 update the values of k_e.

4.3.2. Algorithm Specific by Diagonal

Algorithm 3 is very similar to Algorithm 2. The main difference is that the non-prunable window $[k_s..k_e]$ indicates the cells which belong to the current diagonal and will be computed. Function isPruned (lines 17-19) is the same as in Algorithm 2. Function PruningUpdate (lines 2-19) differs from Function PruningUpdate in Algorithm 2 in two aspects. First,$H_{d-k,k}$ (in which $k \in [0..n]$) contains cells which belong to diagonal d in the DP matrix. Second, the reduction in the pruning area to the right (line 8) is calculated in such a way that the reduction is made by 1 in each update.

Algorithm 3 Algorithm Specific by Diagonal

```
1: (k_s, k_e) ← (0, n)

2: procedure PruningUpdate(d, H_{d−k,k})        ▷ k ∈ [0..n]
3:     bound ← max(bound, H_{d−k,k})
4:     while k_s < n and IsPrunable(d−k_s, k_s, H_{d−k_s,k_s}, bound)
   do
5:         k_s ← k_s + 1        ▷ enlarging pruning area to the left
6:     end while
7:     if k_e < n and ¬IsPrunable(d − k_e, k_e, H_{d−k_e,k_e}, bound)
   then
8:         k_e ← k_e + 1;     ▷ reducing pruning area to the right
9:     else
10:        k_e ← k_e − 1;
11:        while k_e ≥ k_s and IsPrunable(d −
   k_e, k_e, H_{d−k_e,k_e}, bound) do
12:            k_e ← k_e − 1 ▷ enlarging pruning area to the right
13:        end while
14:        k_e ← k_e + 1;
15:    end if
16: end procedure

17: function IsPruned(j)
18:    return j < k_e or j > k_e
19: end function
```

4.3.3. Algorithm Generic Linear

To our knowledge, in the literature, the implementations of the algorithms presented in Section 2.1 either compute the DP cells by row, by column or by diagonal. However, the DP cells of these algorithms can

also be computed in a dataflow-based or generic way. Algorithm Generic Linear (Algorithm 4) is a new algorithm, which allows the computation of DP cells in any order, as long as the data dependencies are respected. Instead of the non-prunable window, it maintains two linear vectors: k_h (horizontal) and k_v (vertical), which indicate the last prunable cell in a given column or row. Thus, memory complexity is $O(m+n)$.

In order to identify if the computation of $H_{i,j}$ may be discarded, function isPruned (lines 3-11) needs to identify if $H_{i-1,j}$, $H_{i,j-1}$ and $H_{i-1,j-1}$ are all prunable (line 4). At this moment, if $k_h[j] = i-1$ then the last prunable cell of column j is cell $H_{i-1,j}$. Analogously, if $k_v[i] = j-1$ then the last prunable cell of row i is $H_{i,j-1}$.

However, it must be noted that the immediate cell in the diagonal ($H_{i-1,j-1}$) must also be considered. In order to do this, we add the adjustment factor $-ma-G$ (line 14) which takes into account the difference in values of adjacent cells (Figure 6). With this, vectors k_h and k_v are updated in such a way that condition $k_h[j] = i-1$ occurs only if $H_{i-1,j}$ and $H_{i-1,j-1}$ are prunable. Analogously, $k_w[i] = j-1$ happens only if $H_{i,j-1}$ and $H_{i-1,j-1}$ are prunable. The pruning test made in line 4 also considers cell $H_{i-1,j-1}$.

In Algorithm 4, function pruningUpdate (lines 12-18) updates positions $k_v[i]$ and $k_h[j]$ if $H_{i,j}$ is identified as prunable by the function isPrunable. Besides, *bound* is updated if the value computed for cell $H_{i,j}$ is greater than its actual value.

Algorithm 4 Algorithm Generic Linear

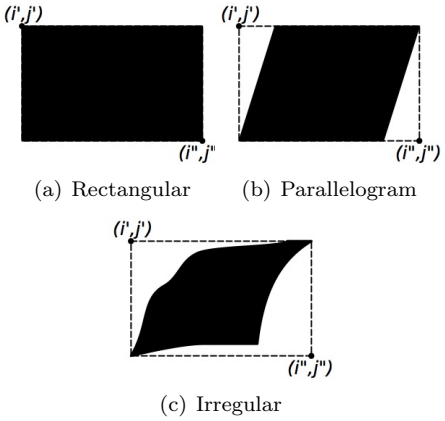
```
1: k_v[0..m] ← {−∞, −1, −1, ..., −1}
2: k_h[0..n] ← {−∞, −1, −1, ..., −1}

3: function IsPruned(i, j)
4:     if k_v[i] = j − 1 and k_h[j] = i − 1 then
5:         k_v[i] ← b_j
6:         k_h[j] ← b_i
7:         return true;
8:     else
9:         return false;
10:    end if
11: end function

12: procedure PruningUpdate(i, j, H_{i,j})
13:    bound ← max(bound, H_{i,j})
14:    if IsPrunable(i, j, H_{i,j} − G − ma, bound) then
15:        k_v[i] ← b_j
16:        k_h[j] ← b_i
17:    end if
18: end procedure
```

4.4. Block Pruning

A block is defined as a set of contiguous cells. In order to eliminate cyclic dependencies among blocks, the following rule must be respected: if cells $H_{i,j}$ and $H_{i-1,j-1}$ belong to a given block, then cells $H_{i-1,j}$ and $H_{i,j-1}$ also belong to this block. Figure 10 presents 3 shapes of blocks which satisfy this property.



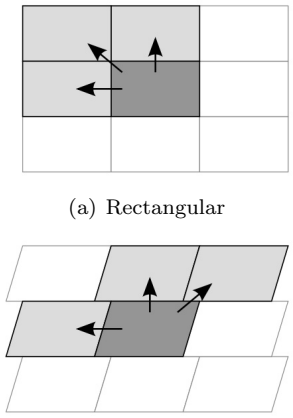
(a) Rectangular (b) Parallelogram

(c) Irregular

FIGURE 10. Different shapes of blocks and their maximum and minimum coordinates.

A *grid* is a matrix $B_h \times B_w$ of blocks placed in h rows and w columns. Each block $B_{x,y}$ has an (i', j') minimum coordinate and an (i'', j'') maximum coordinate, as illustrated in Figure 10.

The dependency among the blocks of a grid depends directly on the shape of the blocks. In Figure 11(a) block dependencies are the same as the cell dependencies discussed previously. Nevertheless, blocks arranged as a parallelogram (Figure 11(b)) have distinct dependencies, in which block $B_{x,y}$ depends on blocks$B_{x-1,y}$, $B_{x,y-1}$ and $B_{x+1,y-1}$. We will call *differentiated dependency* the dependency on $B_{x+1,y-1}$.



(a) Rectangular

(b) Parallelogram

FIGURE 11. Dependencies Among the Blocks of a Grid

The maximum derived score of an alignment that passes through a block $B_{x,y}$ may be defined by Equation 20, which modifies Equation 17.

$$H^{max}_{Bx,y} = \max_{(i,j)\in Bx,y} [H^{max}_{i,j}]$$
$$H^{max}_{Bx,y} = \max_{(i,j)\in Bx,y} [H_{i,j} + min(m-i, n-j)\cdot ma] \quad (20)$$

However, in order to avoid the computation of $H^{max}_{i,j}$ for all the cells which belong to a block, we propose Equation 21, which contains a less strict computation of

the maximum derived score of block $B_{x,y}$, where $H_{Bx,y}$ is the highest score for block $B_{x,y}$.

$$H^{max}_{Bx,y} = \underbrace{\max_{(i,j)\in Bx,y}[H_{i,j}]}_{H_{Bx,y}} + \max_{(i,j)\in Bx,y}[min(m-i, n-j)\cdot ma] \tag{21}$$

Even though the value $H_{Bx,y}$ is obtained after the full block computation, there is an additional overhead to compute $min(m-i, n-j)\cdot ma$ (Equation 21) for all cells which belong to the block. To avoid this, we will consider the minimum coordinate (i', j') for the block and the maximum score of block $H_{Bx,y}$. Equation 22 computes $H^{max}_{Bx,y}$ in a less strict way than Equation 20 but it can be done in constant time, provided that $H_{Bx,y}$ is already known.

$$H^{max}_{Bx,y} = H_{Bx,y} + min(m-i', n-j')\cdot ma \tag{22}$$

Algorithm 5 presents function isPrunable for blocks, implementing Equation 22.

Algorithm 5 Algorithm Block Pruning - Function isPrunable

```
1: function isPrunable(x, y, H_{Bx,y}, bound)
2:     (i', j') := GetMinCoord(B_{x,y})
3:     H^{max}_{Bx,y} = H_{Bx,y} + min(m - i', n - j') · ma
4:     return H^{max}_{Bx,y} ≤ bound
5: end function
```

5. METHODOLOGY FOR THEORETICAL EVALUATION

In this section, we propose a methodology for the theoretical evaluation of the pruning method, with a focus on its effectiveness. In this section, we will only consider local alignments and our methodology will take into account three characteristics: (a) the contents of sequences S_0 and S_1, (b) the order in which the DP cells are computed and (c) the values of the match (ma), mismatch(mi) and gap (G) parameters.

Even though the DP cells have integer values, our methodology uses real values for coordinates and DP values. With this approach, we can analyze and simulate the pruning method with mathematical formulae, instead of computing recurrence equations for each DP cell.

We will consider three special cases for the contents of sequences S_0 and S_1, in order to define the values $H_{i,j}$:

- ***Perfect Match* with repeated characters ($\mathbf{PM}_r$)**: In this scenario, the alphabet has a single character $\Sigma = \{c_1\}$ and both sequences are formed by a repetition of this unique character (i.e., $S_0 = c_1c_1c_1...c_1$ and $S_1 = c_1c_1c_1...c_1$). Thus, $H_{i,j}$ is defined by Equation 23.

$$H_{i,j} = min(i, j).ma \tag{23}$$

- ***Perfect Match* with distinct characters ($\mathbf{PM}^{1.0}$)**: We suppose in this scenario that the alphabet has an infinite number of characters $\Sigma = \{c_1, c_2, c_3, ...\}$ and that the sequences are formed by distinct characters (i.e., $S_0 = c_1c_2c_3...c_m$ and $S_1 = c_1c_2c_3...c_n$), in which $m = n$. In this case, $H_{i,j}$ is defined by Equation 24. Scenario *Perfect Match* with distinct letters is a generalization of scenario *Perfect Match* with similar and distanct characters (PM_r) if we consider $G = 0$.

$$H_{i,j} = max(0, min(i, j).ma - |i-j|G) \tag{24}$$

- ***Partial Match* ($\mathbf{PM}^p$)**: In this scenario, we define *Partial Match* as a case similar to *Perfect Match* with distinct letters, replacing some characters of sequence S_0 by a character that does not exist in sequence S_1, in order to generate mismatches in these positions. Considering that the replacements are uniformly distributed along the sequence and the probability of keeping a nucleotide is $\psi \in [0, 1]$ (i.e., the probability of replacing a nucleotide is 1-ψ), we define the values of the cells in an approximate way by Equation 25, in which $p = \psi - \frac{mi}{ma}(1-\psi)$ and $p \in [0, 1]$. Scenario *Partial Match* is a generalization of scenario $\mathrm{PM}^{1.0}$ if we consider $p = 1$.

$$\begin{aligned} H_{i,j} &= max(0, min(i, j)\cdot \underbrace{(ma.\psi - mi(1-\psi))}_{ma\cdot p} - |i-j|G) \\ &= max(0, min(i, j)\cdot ma\cdot p - |i-j|G) \end{aligned} \tag{25}$$

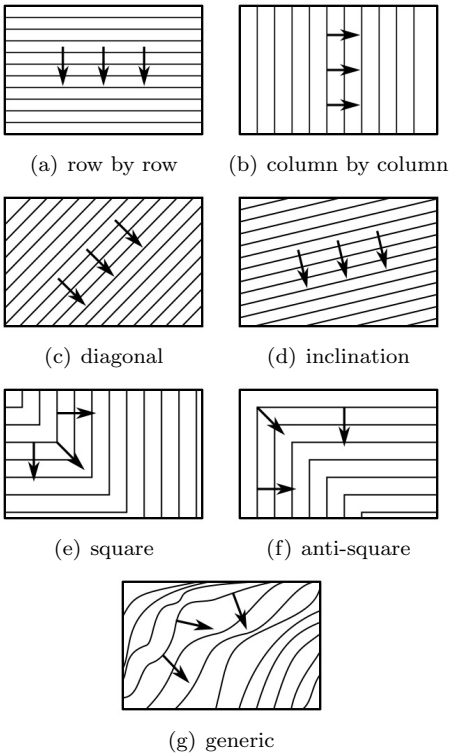
The second aspect considered in the proposed methodology is the order in which the DP cells are computed. Figure 12 shows the ways of processing the DP matrix considered in this paper: row by row (Figure 12(a)), column by column (Figure 12(b)), by diagonal (Figure 12(c)), with inclination (Figure 12(d)), square (Figure 12(e)), anti-square (Figure 12(f)) and generic (Figure 12(g)). Computing by inclination (considering an angle θ) generalizes computing row by row ($\theta = 0°$), column by column ($\theta = 90°$) and by diagonal ($\theta = 45°$). Computing in a generic way will not be used in our analysis since the processing order may vary in each execution.

The order in which the DP cells are computed is directly related to the best provisory score $best_{i,j}$ (Table 1) which is produced when cell $H_{i,j}$ is computed. If we assume that $H_{i,j}$ is known, the score $best_{i,j}$ can be defined by a formula. The analysis of $best_{i,j}$ is simpler for the local alignment since $H^{min}_{i',j'} = H_{i',j'}$ for all DP cells (Equation 18).

Assuming that the optimal alignment lies on the main diagonal of the DP matrix for scenarios PM_r, $PM^{1.0}$ and PM^p, the value of $best_{i,j}$ may be defined for iteration $\phi_{i,j}$ (Table 1) using Equation 26.

$$best_{i,j} = H_{\phi_{i,j},\phi_{i,j}} = \phi_{i,j}.ma.p \tag{26}$$

Figure 13 presents geometrically the position of



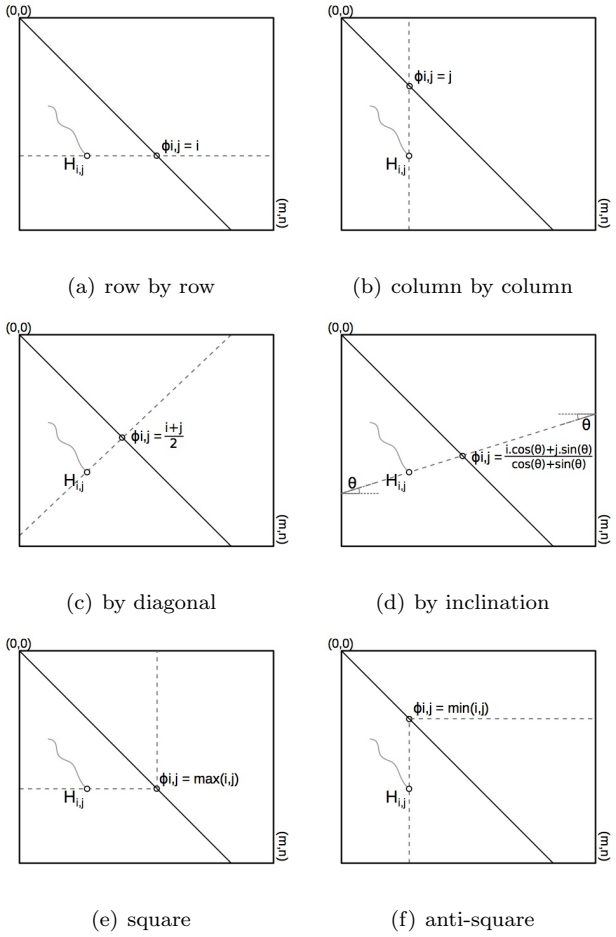
(a) row by row (b) column by column

(c) diagonal (d) inclination

(e) square (f) anti-square

(g) generic

FIGURE 12. Ways of computing the DP matrix.



(a) row by row (b) column by column

(c) by diagonal (d) by inclination

(e) square (f) anti-square

FIGURE 13. Best provisory score ($best_{i,j}$) in six ways of DP computation.

$best_{i,j}$ for different ways of processing the DP matrix, which may be applied to scenarios PM_r, $PM^{1.0}$ or PM^p.

Then, the value of iteration $\phi_{i,j}$ (Table 1) is defined for each DP processing way with Equations 27 to 32. We recall that the value $\phi_{i,j}$ belongs to the set of real numbers.

$$\text{row by row:} \quad \phi_{i,j} = i \tag{27}$$

$$\text{column by column:} \quad \phi_{i,j} = j \tag{28}$$

$$\text{by diagonal:} \quad \phi_{i,j} = \frac{i+j}{2} \tag{29}$$

$$\text{by inclination } \theta\text{:} \quad \phi_{i,j} = \frac{i.cos(\theta) + j.sin(\theta)}{cos(\theta) + sin(\theta)} \tag{30}$$

$$\text{square:} \quad \phi_{i,j} = max(i,j) \tag{31}$$

$$\text{anti-square:} \quad \phi_{i,j} = min(i,j) \tag{32}$$

5.1. Effectiveness of the Pruning Method

We define Effectiveness of Pruning the ratio between the number of prunable cells and the total number of cells in the DP matrix, as shown in Equation 33.

$$\text{effectiveness} = \frac{prunable\_cells}{|S_0| \times |S_1|} \tag{33}$$

We can also state that the pruning effectiveness is geometrically defined by the ratio between the area containing prunable cells and the total area of the DP matrix. The pruning area is the area in which the condition $H^{max}_{i,j} < best_{i,j}$ is true. This condition is called the *prunable area condition.*

In the remainder of this section, we define the size of the pruning area for scenarios PM_r, $PM^{1.0}$ and PM^p (in which $H_{i,j}$ is defined by Equations 23, 24 and 25, respectively) and the different ways of processing the matrix (where the value $\phi_{i,j}$ is defined by Equations 27 to 32). Only local alignments will be considered and we assume that the sequences have the same length ($|S_0| = |S_1| = m$). The values $H^{max}_{i,j}$ and $H^{min}_{i,j}$ are defined by Equations 17 and 18.

Figure 5.1 presents the geometrical definitions of the pruning areas. We can distinguish two pruning areas, which are joined by the point (m, m) at the bottom right of the matrix. We will call A_1 and A_2 the areas which are respectively to the right and to the left of the main diagonal. These areas have 4 sides at most, limited by lines expressed by functions f_1, f_2, f_3, f_4 and by the borders of the matrix. The definition of functions f_1, f_2, f_3, f_4 is based on the prunable area condition and some premisses which will be defined in Equations 37. The vertices of the areas are defined as $A_1 = (p11, p12, p13, p14)$ and $A_2 = (p21, p22, p23, p24)$ and their coordinates are expressed in Equations 34. Vertices p_{14} and p_{24} are the intersection of lines $f_1 \cap f_2$ and $f_3 \cap f_4$, respectively.

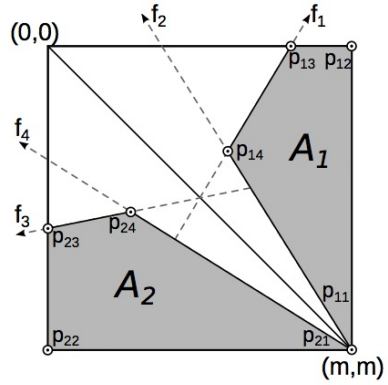


FIGURE 14. Geometrical representation of the pruning areas.

$$\begin{array}{llll} p_{11} & = (0, f1(0)) & p_{21} & = (f3(0), 0) \\ p_{12} & = (0, m) & p_{22} & = (m, 0) \\ p_{13} & = (m, m) & p_{23} & = (m, m) \\ p_{14} & = (i', f1(i')) = (i', f2(i')) & p_{24} & = (f3(j'), j') = (f4(i'), j') \end{array} \tag{34}$$

With the points defined by Equations 34, we can calculate the size of areas A_1 and A_2 with Equation 35. This generic equation calculates the area of a polygon formed by four points.

$$A = \frac{1}{2} \cdot abs[\begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{vmatrix}] = abs\left[\frac{(x_1y_2 + x_2y_3 + x_3y_4 + x_4y_1) - (y_1x_2 + y_2x_3 + y_3x_4 + y_4x_1)}{2}\right] \tag{35}$$

To define the equations of functions f_1, f_2, f_3 and f_4, we will consider the prunable area condition $H_{i,j}^{max} \leq best_{i,j}$ using the values for $H_{i,j}$ (Equation 25), $\phi_{i,j}$ (Equations 27 to 32), $best_{i,j} = \phi_{i,j}.ma.p$ (Equations 26) and $H_{i,j}^{max}$ (Equation 17), obtaining Inequality 36:

$$\begin{aligned} H_{i,j}^{max} &\leq best_{i,j} \\ H_{i,j} + min(m-i, m-j).ma &\leq \phi_{i,j}.ma.p \\ max\,(0, min(i,j)ma.p - |i-j|G) + & \\ min(m-i, m-j).ma &\leq \phi_{i,j}.ma.p \end{aligned} \tag{36}$$

In the definition of functions f_1, f_2, f_3 and f_4, we will use a set of premises to select operands of functions max and min from Inequality 36. The premises used in each function are defined in (37):

$$\begin{array}{ll} f1: & j > i \text{ and } H_{i,j} = 0 \\ f2: & j > i \text{ and } H_{i,j} \neq 0 \\ f3: & j \leq i \text{ and } H_{i,j} = 0 \\ f4: & j \leq i \text{ and } H_{i,j} \neq 0 \end{array} \tag{37}$$

We then transform Inequalities 36 in four equations (Equations 38 to 41) based in the four premises defined in (37).

case f_1: $j > i$ and $H_{i,j} = 0$

$$\begin{aligned} H_{i,j} + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ 0 + (m-j).ma &= \phi_{i,j}.ma.p \\ j &= m - \phi_{i,j}.p \end{aligned} \tag{38}$$

case f_2: $j > i$ and $H_{i,j} \neq 0$

$$\begin{aligned} H_{i,j} + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ min(i,j)ma.p - |i-j|G + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ i.ma.p - (j-i)G + (m-j).ma &= \phi_{i,j}.ma.p \\ j(G+ma) &= i(ma.p+G) + m.ma - \phi_{i,j}.ma.p \\ j &= \frac{i(ma.p+G) + m.ma - \phi_{i,j}.ma.p}{G+ma} \end{aligned} \tag{39}$$

case f_3: $j \leq i$ and $H_{i,j} = 0$

$$\begin{aligned} H_{i,j} + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ 0 + (m-i).ma &= \phi_{i,j}.ma.p \\ i &= m - \phi_{i,j}.p \end{aligned} \tag{40}$$

case f_4: $j \leq i$ and $H_{i,j} \neq 0$

$$\begin{aligned} H_{i,j} + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ min(i,j)ma.p - |i-j|G + min(m-i, m-j).ma &= \phi_{i,j}.ma.p \\ j.ma.p - (i-j)G + (m-i).ma &= \phi_{i,j}.ma.p \\ i(G+ma) &= j(ma.p+G) + m.ma - \phi_{i,j}.ma.p \\ i &= \frac{j(ma.p+G) + m.ma - \phi_{i,j}.ma.p}{G+ma} \end{aligned} \tag{41}$$

For the four cases listed in Equations 38 to 41, we must define values for $\phi_{i,j}$. For instance, if the way of processing the matrix is row by row then $\phi_{i,j} = i$ (Equation 27). Using this, functions f_1, f_2, f_3 and f_4 are defined as:

case f_1: (Equation 38)

$$\begin{aligned} j &= m - \phi_{i,j}.p \\ j &= m - i.p \\ f_1(i) &= m - i.p \end{aligned}$$

case f_2: (Equation 39)

$$\begin{aligned} j &= \frac{i(ma.p+G) + m.ma - \phi_{i,j}.ma.p}{G+ma} \\ j &= \frac{i(ma.p+G) + m.ma - i.ma.p}{G+ma} \\ f_2(i) &= \frac{i.G + m.ma}{G+ma} \end{aligned}$$

case f_3: (Equation 40)

$$\begin{aligned} i &= m - \phi_{i,j}.p \\ i &= m - i.p \\ f_3(j) &= \frac{m}{p+1} \end{aligned}$$

case f_4: (Equation 41)

$$\begin{aligned} i &= \frac{j(ma.p+G) + m.ma - \phi_{i,j}.ma.p}{G+ma} \\ i &= \frac{j(ma.p+G) + m.ma - i.ma.p}{G+ma} \\ f_4(j) &= \frac{j(ma.p+G) + m.ma}{G+ma.(p+1)} \end{aligned}$$

Assuming that $p = 1$, $ma = 1$ and $G = 3$, we find the following coordinates of the pruning area:

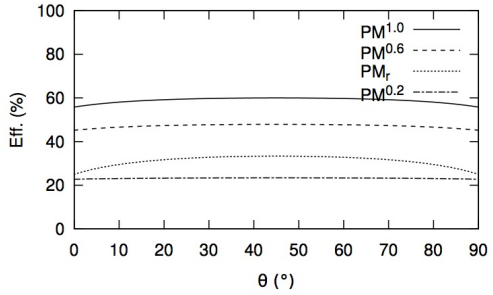


FIGURE 15. Pruning Effectiveness vs. Processing Angle

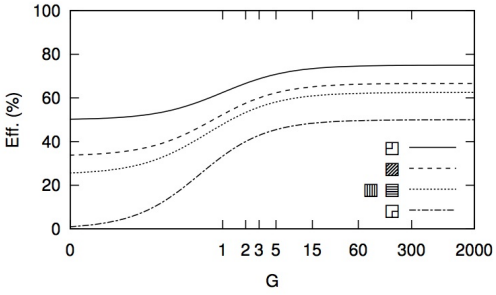


FIGURE 16. Pruning Effectiveness vs. Gap Penalty (Scenario $PM^{1.0}$)

$$\begin{array}{llll} p_{11} & = (0, m) & p_{21} & = (m\frac{1}{2}, 0) \\ p_{12} & = (0, m) & p_{22} & = (m, 0) \\ p_{13} & = (m, m) & p_{23} & = (m, m) \\ p_{14} & = (m\frac{3}{7}, m\frac{4}{7}) & p_{24} & = (m\frac{1}{2}, m\frac{3}{8}) \end{array} \quad (42)$$

In order to solve our equations in all analyzed scenarios, we used the *Maxima* tool [20] as presented in Appendix (Appendix B).

Table 4 presents the 18 equations using variables ma, mi, G, p and θ. With these equations, it can be seen that, as expected, processing by row or by column presents exactly the same pruning effectiveness. An interesting result is that the pruning effectiveness obtained when processing by row/column corresponds to the median between the effectiveness of processing by square and anti-square (Figures 16, 17 and 18).

5.2. Impact on Pruning Effectiveness

With the equations listed in Table 4, we can analyze the impact of some variables on the pruning effectiveness. The graphics illustrated in Figures 15 to 18 present the effectiveness calculated when some parameters are varied. In these graphics, the following default values were used: $G = 3$, $ma = 1$, $mi = 3$.

Figure 15 shows a variation in the θ angle from 0° to 90° in scenarios $PM^{1.0}$, $PM^{0.6}$, PM_r and $PM^{0.2}$. We can notice that the best result for all scenarios considered occurs when $\theta = 45°$ (by diagonal), indicating that we discard more DP cells when the DP matrix is processed in a wavefront of 45°. It may also be noted that the variation in the effectiveness of pruning is relatively small in the PM^p scenarios. For instance, in the $PM^{1.0}$ scenario, effectiveness varies 4.20%, from 55.80% (0° and 90°) to 60.00% (45°). With $PM^{0.6}$, effectiveness varies 2.71%: 45.18% (0° and 90°) to 47.89% (45°). The variation in the effectiveness for the PM_r scenario (Figure 15) is a little bit higher (8.33%), between 25% (0°, 90°) and 33.33% (45°). Therefore, we may conclude that the variation on θ has more impact in scenario PM_r and that the lower the value for p, the lower the impact of θ on the effectiveness of pruning.

Figure 16 illustrates the pruning effectiveness in scenario $PM^{1.0}$ with the gap penalty G varying from 0 to 2000. It should be noted that the PM_r scenario occurs when $G = 0$. It can be seen that the effectiveness of pruning grows quickly in the interval defined by $G = 1$ and $G = 5$, tending to a constant value when $G \geq 60$. When G tends to infinity, the effectiveness is 75.00%, 66.67%, 62.50% and 50.00% for the square, diagonal ($\theta = 45°$), by row/column and anti-square processing ways, respectively. We therefore conclude that the value of the gap penalty has a high impact on the pruning effectiveness. Moreover, varying G in a small interval ([1,5]) produces a great variation on the pruning effectiveness.

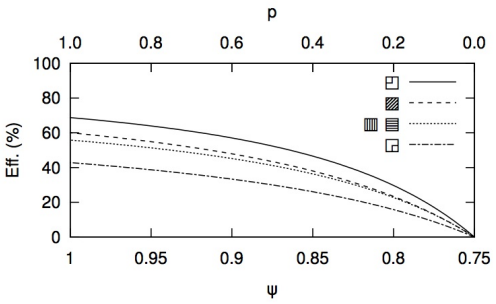
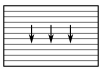
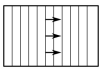
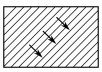
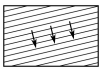
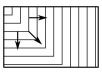
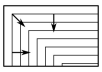


FIGURE 17. Pruning Effectiveness vs. the values of ψ and p (Scenario PM^p)

Figure 17 presents the pruning effectiveness in scenario PM^p when we vary the similarity between the sequences (ψ). The value of p depends on ψ and it is obtained with Equation 25 ($p = \psi - \frac{mi}{ma}(1 - \psi)$). It can be seen that, as expected, the effectiveness of pruning is zero when the sequences have no similarity at all ($p = 0$). We can also notice that the pruning effectiveness is always decrescent when we reduce the similarity ψ of the sequences, attaining zero when $\psi = \frac{mi}{ma+mi}$. Thus, we can also say that, as expected, the similarity between the sequences has a huge impact on the pruning effectiveness. For instance, when we process the DP matrix by square, the pruning effectiveness varies from 68.8% ($\psi = 1$) to 0% ($\psi = 0.75$).

Figures 18(a) and 18(b) illustrate the pruning effectiveness in the perfect match scenario ($PM^{1.0}$) when we vary, respectively, the values for match (ma) and mismatch(mi). In this scenario of perfect match, increasing the value of ma is equivalent to reduce, proportionally, the gap penalty G. When ma tends

TABLE 4. Pruning Effectiveness

Wave	$H_{i,j}$	Effectiveness (%)
	PM_r	$\frac{1}{4} = 25.0\%$
	$PM^{1.0}$	$\frac{1}{2}.\frac{G}{2G+ma} + \frac{1}{4}.\frac{3G+2ma}{2G+2ma}$
	PM^p	$\frac{1}{2}.\frac{p\,G}{G+p\,G+ma\,p} + \frac{1}{2}.\frac{p}{p+1}.\frac{2G+p\,G+ma\,p+ma\,p^2}{1G+p\,G+ma\,p+ma\,p^2}$
	PM_r	$\frac{1}{4} = 25.0\%$
	$PM^{1.0}$	$\frac{1}{4}.\frac{3G+2ma}{2G+2ma} + \frac{1}{2}.\frac{G}{2G+ma}$
	PM^p	$\frac{1}{2}.\frac{p}{p+1}.\frac{2G+p\,G+ma\,p+ma\,p^2}{1G+p\,G+ma\,p+ma\,p^2} + \frac{1}{2}.\frac{p\,G}{G+p\,G+ma\,p}$
	PM_r	$\frac{1}{3} = 33.3\%$
	$PM^{1.0}$	$\frac{1}{3}.\frac{8G+3ma}{4G+3ma}$
	PM^p	$\frac{p}{p+2}.\frac{6\,G+2\,p\,G+2\,ma\,p+ma\,p^2}{2\,G+2\,p\,G+2\,ma\,p+ma\,p^2}$
	PM_r	$\frac{2c_0-1}{5c_0-1}$
	$PM^{1.0}$	$\frac{1}{2}.\frac{c_0}{c_2+c_0}.\frac{(c_1+c_2)G+(c_2+c_3)ma}{(c_0+c_0)G+(c_0+c_2)ma} + \frac{1}{2}.\frac{c_0}{c_5+c_0}.\frac{(c_4+c_5)G+(c_5+c_6)ma}{(c_0+c_0)G+(c_0+c_5)ma}$
	PM^p	$\frac{1}{2}.\frac{c_0\,p}{c_2\,p+c_0}.\frac{c_1\,G+c_2\,p\,G+c_2\,ma\,p+c_3\,ma\,p^2}{c_0\,G+c_0\,p\,G+c_0\,ma\,p+c_2\,ma\,p^2} + \frac{1}{2}.\frac{c_0\,p}{c_5\,p+c_0}.\frac{c_4\,G+c_5\,p\,G+c_5\,ma\,p+c_6\,ma\,p^2}{c_0\,G+c_0\,p\,G+c_0\,ma\,p+c_5\,ma\,p^2}$
	$s = sen\theta$, $c = cos\theta$, $\alpha = (s+c)$, $c_0 = \alpha^2$, $c_1 = \alpha\,(\alpha+s)$, $c_2 = \alpha\,s$, $c_3 = s^2$, $c_4 = \alpha\,(\alpha+c)$, $c_5 = \alpha\,c$, $c_6 = c^2$	
	PM_r	$\frac{1}{2} = 50\%$
	$PM^{1.0}$	$\frac{1}{2}\cdot\frac{3G+2ma}{2G+2ma}$
	PM^p	$\frac{p}{p+1}\cdot\frac{2G+p\,G+ma\,p+ma\,p^2}{1G+p\,G+ma\,p+ma\,p^2}$
	PM_r	0
	$PM^{1.0}$	$\frac{G}{2G+ma}$
	PM^p	$\frac{p\,G}{G+p\,G+ma\,p}$

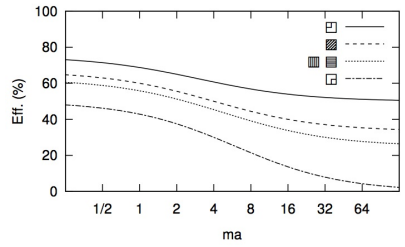
to infinity, the effectiveness is the same of the one presented by the scenario PM_r, in which $G = 0$. Thus, the shape of Figure 18(a) is similar to the one in Figure 16, even though in opposite directions. We therefore conclude that, if we increase the punctuation for matches, the effectiveness of pruning tends to be reduced, in the scenario of perfect match. On the other hand, Figure 18(b) shows that the punctuation of mismatches does not affect the pruning effectiveness in a perfect match scenario.

Figures 18(e) and 18(f) show the effectiveness of pruning on scenario PM^p with similarity $\psi = 0.9$ when we vary, respectively, the punctuations for matches (ma) and mismatches (mi). We must consider that $p = \psi - \frac{mi}{ma}(1-\psi)$ (Equation 25), thus the value p is also modified when ma or mi are modified. Here, we are considering a *partial match* scenario with $\psi = 0.9$. In this case, values for ma that are too small would make it impossible to obtain alignments, since the punctuation for mismatches (mi) would be proportionally much higher than the punctuation for matches (ma). For this reason, Figure 18(e) shows that the pruning effectiveness is zero for small values. Effectiveness then grows to a peak and decreases up to a constant value in infinity. The peak in the graphic depends on the way in which the matrix is processed and the values of parameters ma, mi and G. In addition, Figure 18(f) shows that the value of mi affects the effectiveness of pruning in the partial match scenario, since sequences which are not very similar will have scores close to zero when mi is much greater than ma. Figures 18(c) and 18(d) show the pruning effectiveness in an intermediate scenario of similarity ($\psi = 0.95$).

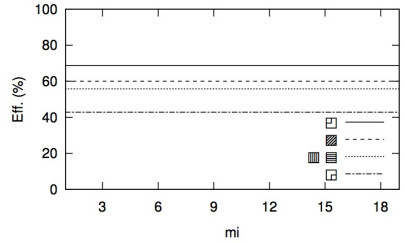
5.3. Block Pruning Effectiveness reduction

If the pruning is calculated in a per-block basis, a block will be prunable only if all its cells are prunable. In this scenario, the pruning effectiveness is reduced because there will exist prunable cells that will not be pruned because they are located in blocks that also contains non-prunable cells. The *effectiveness reduction ratio* is the number of such prunable cells divided by the matrix size. Figure 20 shows the pruning region with different grid sizes, where the gray blocks contains prunable and non-prunable cells. In the figure, the gray blocks are those that cross the edges of the pruning areas, and the area of gray blocks increases as the size of the blocks is increased.

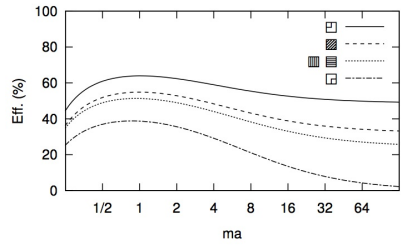
In order to estimate the number of the blocks that cross the edges of the pruning area, we must note that the number of gray blocks in Figure 20 is the same as the number of horizontal and vertical grid lines that cross



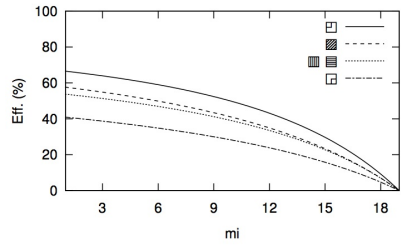
(a) *match* ($\psi = 1.0$)



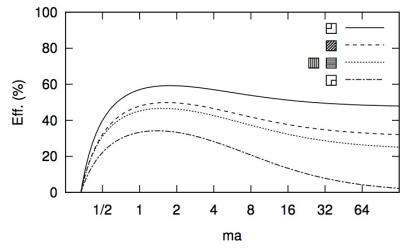
(b) *mismatch* ($\psi = 1.0$)



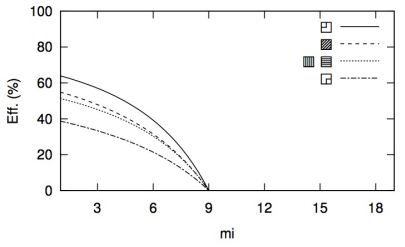
(c) *match* ($\psi = 0.95$)



(d) *mismatch* ($\psi = 0.95$)



(e) *match* ($\psi = 0.9$)



(f) *mismatch* ($\psi = 0.9$)

FIGURE 18. Pruning Effectiveness vs. punctuation for *matches* or *mismatches* (Scenario $PM^{1.0}$)

the edges of the pruning area. This number of blocks is at most $2(\frac{m+n}{B})$, where m and n are the sequence sizes and B is the size of the block (where B^2 is the number of cells in each block). Since the number of prunable cells inside these blocks may vary from 0 to B^2, the number of ignored prunable cells is at most $2B(m+n)$ cells, or proportionally to the matrix size, $\frac{2B(m+n)}{mn}$. If the sequences have the same size ($m = n$), the effectiveness reduction ratio can be estimated with Equation 43, where $\alpha \in [0, 4]$ is dependent on the pruning edges positions and *Grid* is the size of the grid (where $Grid^2$ is the number of blocks in the grid).

$$\text{eff. reduction} = \alpha\frac{B}{n} = \frac{\alpha}{Grid}, \text{ where } Grid = \frac{n}{B} \quad (43)$$

Thus, we can conclude that the block pruning effectiveness reduction is proportional to the block size (average block dimension), inversely proportional to the sequence size and inversely proportional to the grid size. For sufficiently large sequences ($n \gg B$), the effectiveness reduction tends to be negligible.

The best selection of the block size depends on the implementation strategy. For instance, MASA-CUDAlign [17] uses GPUs to accelerate the alignment computation and the GPU parallelism is based on blocks. The performance of MASA-CUDAlign is highly dependent on the size of the blocks and a study of this effect was already made in [14], where it was shown that using few large blocks reduces the parallelism whereas a great number of small blocks increases the synchronization overhead.

6. PRUNING SIMULATIONS

As shown in Sections 5.1 and 5.2, pruning effectiveness depends on characteristics of the sequences, on the way the DP matrix is processed and on the values used in some parameters. In this section, we present simulations in which the pruned area is plotted with the *Gnuplot* tool. With these simulations, we want to verify the appropriateness of the formulae proposed in Table 4 and the graphics shown in Section 5.2. Additionally, we present geometrically the effects that some parameters have on the pruning area. Like in Section 5.2, the default values used in the simulations are $G = 3$, $ma = 1$ and $mi = 3$.

6.1. Different Ways of Processing the DP Matrix

In the first simulation (Figure 19), we show the pruning area for different ways of DP processing in three different scenarios: perfect match with repeated characters PM_r (Equation 23), perfect match with distinct characters $PM^{1.0}$ (Equation 24) and partial match PM^p (Equation 25, with p=0.6). We can notice that scenario $PM^{1.0}$ with square processing yields the largest pruning area. On the other hand, the smaller

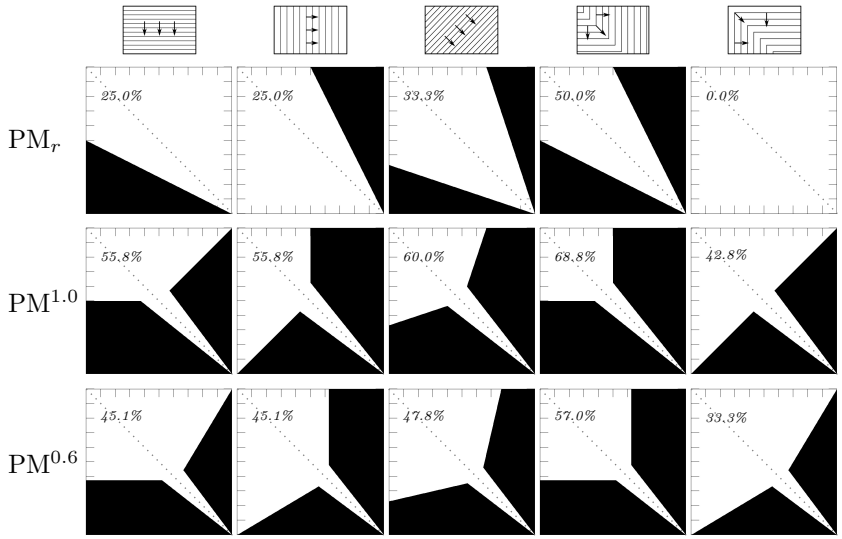


FIGURE 19. Pruning effectiveness at scenario PM_r, $PM^{1.0}$ and $PM^{0.6}$ with different ways of DP processing

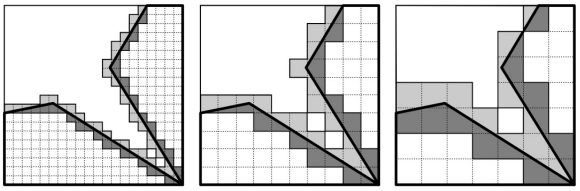


FIGURE 20. Highlighted blocks contains prunable (dark gray) and non-prunable (light gray) cells simultaneously.

pruning areas occur at scenario PM_r, since matches occur in all the DP cells and the values of each cell are higher than in other scenarios. We can also notice that the anti-square processing presents the smaller pruning area. That happens because this way of DP processing slows down the computation of the higher scores in the matrix. Notwithstanding, square processing produces higher values earlier and thus presents a very good pruning effectiveness.

6.2. DP Processing Angles

In the second simulation (Figure 21), we considered several θ angles when processing by inclination. At scenarios PM_r, $PM^{1.0}$ and PM^p, the pruning area is higher when $\theta = 45°$, with symmetric areas when the angle is higher or lower than 45°. For instance, the pruning area for angle $\theta = 15°$ is equal to the pruning area produced when $\theta = 75°$. Particularly, angle $\theta = 0°$ corresponds to processing row by row and angle $\theta = 90°$ corresponds to processing column by column.

6.3. Similarity between the Sequences

In the third simulation (Figure 22), we used different values of p in the Partial Match scenario (PM^p). As expected, the pruning area is reduced when the value of p is reduced, i.e., when the sequences become less similar. It is also worth noticing that, as in the other simulations, square processing leads to the best results in all scenarios considered. This happens because, in the cases studied in this paper, the optimal alignment lies at the main diagonal and that coincides with the position of the vertex of the square in each iteration. We can also notice that pruning is ineffective for sequences in which $p = 0$.

6.4. Gap Penalty

In the forth simulation (Figure 23), we considered several values for the gap penalty G. The effectiveness of Block Pruning is higher when the value of G is big. We can notice that the angle formed by lines f_2 and f_4 (Figure 5.1) is smaller when the value of G is higher, which means a higher decay in the values of the DP cells caused by the gap penalty. A scenario in which $G = 0$ is analogous to the perfect match with distinct characters scenario, where we can notice a smaller effectiveness of pruning.

6.5. Sequences with different lengths

Finally, in the fifth simulation (Figure 24), we consider sequences with different lengths, with a ratio $\frac{n}{m}$ varying from 1 to 3. In general, the effectiveness of pruning is higher when we augment the ratio $\frac{n}{m}$. The only case when there was a reduction in the effectiveness happened when the longest sequences lies in the horizontal and the matrix is processed row by row. Analogously, pruning effectiveness is also reduced when the longest sequence is placed in the vertical and the matrix is processed column by column.

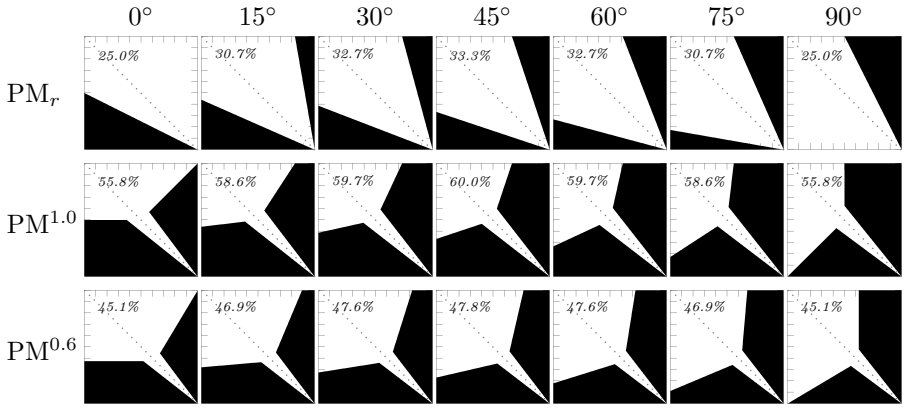


FIGURE 21. Pruning area for different processing angles

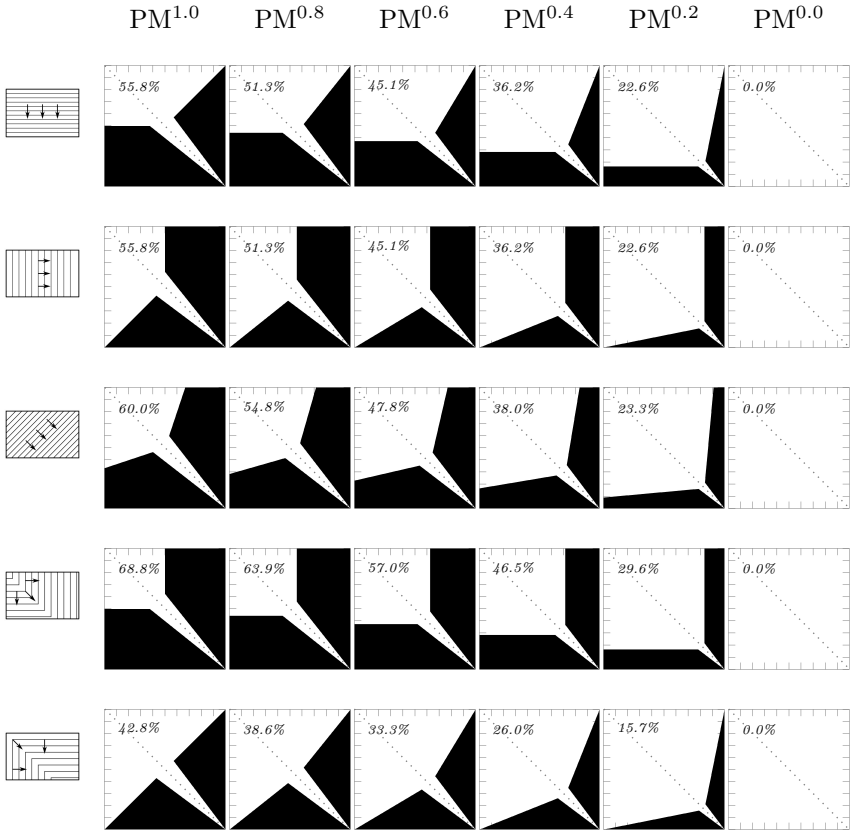


FIGURE 22. Variation on the similarity of the sequences compared (p)

7. EXPERIMENTAL RESULTS

In this section we present experimental results with real DNA sequences retrieved from the National Center for Biotechnology Information (NCBI) site (*www.ncbi.nlm.nih.gov*). Table 5 presents the accession number of the sequences, as well as their names and sizes, ranging from 51 Thousands Base Pairs (KBP) to 1 Million Base Pairs (MBP). Table 5 also shows four pairs of sequences with their optimal local score and an $adjusted_p = \frac{score}{min(n,m)}$. The $adjusted_p$ is proposed as an approximation of the p parameter in the definition of partial match (PM^p in Section 5) for the real cases. Comparison 50K×50K is a case of perfect match, with p=1.

We used the MASA-Serial and MASA-OpenMP tools from the MASA architecture [17] since they implement the block pruning algorithm, and they do not impose restrictions on the size of the sequences. MASA-Serial allows different ways of processing the DP matrix (by row, by column, by diagonal, with inclination, square and anti-square). MASA-OpenMP uses the OpenMP framework to process the blocks in parallel (by diagonals). The experiments were made in a machine with an Intel i5-2310 CPU (4 cores) and 8GB of RAM.

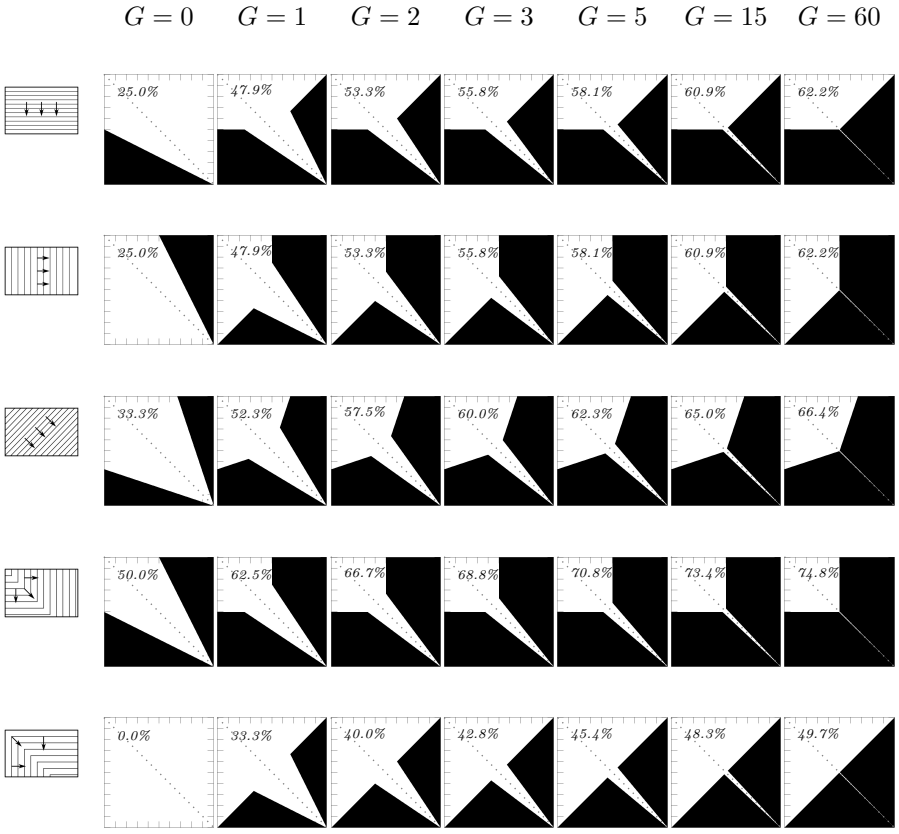


FIGURE 23. Variation on the gap penalty

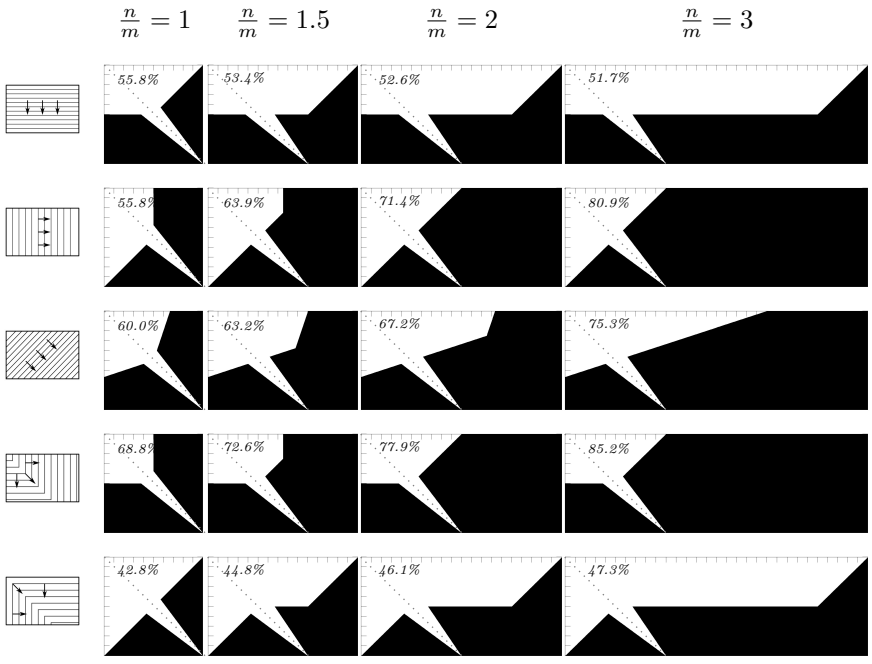


FIGURE 24. Variation on the ratio on the lengths of the sequences

Otherwise stated in the experiments, MASA-Serial was used.

To compute the matrix with inclination, we processed the blocks by diagonals composed by rectangular blocks with sides $(B{\cdot}c){\times}(\frac{B}{c})$, such that the block area is still B^2. To process the matrix with 20◦ inclination, we use $c = \sqrt{tan(20\circ)} = 1.65755$.

Comparison	Accession Number	Name	Real size	Score	$adjusted_p$
50K×50K	CP020060.1	*Escherichia coli* strain AR_0061 plasmid	50,999 BP	50,999	1.0000
	CP020060.1	*Escherichia coli* strain AR_0061 plasmid	50,999 BP		
100K×100K	AF270937.1	*Plutella xylostella* granulovirus genome	100,999 BP	99,361	0.9840
	KU529791.1	*Plutella xylostella* granulovirus isolate PxGV_C	100,980 BP		
200K×200K	KC813501.1	Cowpox virus strain RatAac09/1	212,814 BP	177,416	0.8490
	KC813504.1	Cowpox virus strain RatHei09/1	208,980 BP		
1M×1M	CP015296.1	*Chlamydia trachomatis* strain E-160	1,043,007 BP	930,481	0.9070
	CP018052.1	*Chlamydia trachomatis* strain QH111L	1,025,839 BP		

TABLE 5. Comparison for the real sequences used in tests. Sizes range from 50KBP to 1MBP.

7.1. Predicted vs Real Pruning Effectiveness

In this first experiment, we measured the difference between the predicted and real pruning effectiveness (absolute error). Each comparison was made using the following processing orders: by row, by diagonal, by square, and by anti-square. The block size was set in such way that the grid contains 1000×1000 blocks.

The comparison between the predicted and real pruning effectiveness is presented in Figure 25, in which the "Abs. Error" column presents the absolute error in percentage points. It can be noted that the highest differences occur in comparisons with less similar sequences (i.e. lower $adjusted_p$). For instance, comparison 200K×200K, which presents the lowest $adjusted_p$, presents errors ranging from 0.56 to 1.09 percentage point. Comparison 1M×1M presents an intermediate error variation, ranging from 0.03 to 0.54 percentage point. On the other hand, comparisons 50K×50K and 100K×100K, which are the comparisons with highest $adjusted_p$, presented much more stable errors as we vary the way of processing the DP matrix, ranging from 0.33 to 0.38 percentage point. In all cases, the absolute error of the predicted pruning effectiveness is very small (less than 1.1 percentage point), showing that our prediction is highly accurate.

Table 25 also presents the execution times of each comparison with and without block pruning. It can be noticed that the execution time reduction is almost the same as the real pruning effectiveness, showing that the block pruning mechanism has a very low overhead. For instance, the execution time of the 100K×100K comparison was reduced from 51.75s (without block pruning) to 17.87s (with block pruning) with the square processing order, a reduction of 65.47%, whereas the pruning effectiveness was 65.97%.

7.2. Pruning Effectiveness vs Sequence similarity

In this experiment, the original sequence CP020060.1 (50K) was transformed into similar sequences CP020060.1^x based on parameter $x \in [0..30]$ such that: a nucleotide is changed with x% probability; and an insertion or deletion is made with $0.02x$% probability. The length of each insertion and deletion of gaps is limited to range [1..200] and follows a power-law distribution with exponent -2.

Then, each sequence CP020060.1^x was compared with CP020060.1 and the optimal score was normalized as $p = \frac{score}{n}$, where n is the size of sequence CP020060.1. The pruning was made in blocks of size 16×16. Figure 26 shows the pruning effectiveness for each comparison, with varied similarities. Using the formulae developed in Section 5, the absolute error on pruning effectiveness was, at most, 0.56 percentage point (Figure 27).

7.3. Pruning Effectiveness vs Size of the Block

In this experiment, we measured the pruning effectiveness varying the block size B from 50 to 3000. We compared the sequence CP020060.1 (50K) with itself (perfect match) using the following processing orders: by square, by row, by diagonal (45∘), by inclination (20∘) and by anti-square.

Figure 28 presents the pruning effectiveness for this experiment. Along the x-axis, all the scenarios present an additional reduction of approximately 0.56 percentage points for every 100 cells increased in block size B, corresponding to the estimation of Equation 43 with $\alpha = 2.88$.

7.4. Pruning Performance

MASA-OpenMP was executed with 4 threads and we compared sequence CP020060.1 (50K) with itself (perfect match). The block size B was varied from 10 to 1000 and the times are presented in Figure 29, with and without block pruning, calculated as the average of 5 executions. In this figure, it is clear that block pruning can greatly reduce the execution time. For instance, using blocks of size $B = 100$, the comparison took 4.66 seconds without block pruning whereas it took 1.94 seconds with block pruning, with a reduction of 58.4% in the execution time.

We can see that the performance is very compromised when $B < 100$ due to the computation overhead of small blocks. For instance, the block pruning comparison with $B = 50$ was 27.7% slower compared with $B = 100$.

Comparison	Proc. Way	Prun. Effect. (%)		Abs. Error	Exec. Time (s)		Exec. Time
		Real	Predicted	(perc. pt.)	without BP	with BP	Reduction (%)
50K×50K		52.9507	53.3333	0.3826	13.32	6.41	51.88
		57.2079	57.5758	0.3679	13.37	5.90	55.87
		66.3004	66.6667	0.3663	13.35	4.71	64.72
		39.6010	40.0000	0.3990	13.42	8.19	38.97
100K×100K		52.6835	53.0466	0.3631	51.68	24.61	52.38
		56.8615	57.2401	0.3786	51.73	22.64	56.23
		65.9747	66.3517	0.3770	51.75	17.87	65.47
		39.4046	39.7415	0.3369	51.77	31.56	39.04
200K×200K		51.4368	50.3465	1.0903	222.12	108.51	51.15
		54.9974	54.0897	0.9077	222.43	100.73	54.71
		63.9093	63.3497	0.5596	222.42	80.90	63.63
		38.1638	37.3433	0.8205	222.40	137.94	37.96
1M×1M		52.1140	51.5723	0.5417	5316.40	2540.11	52.22
		55.7941	55.5172	0.2769	5295.04	2349.20	55.63
		64.6895	64.7206	0.0311	5298.34	1893.51	64.26
		38.9352	38.4241	0.5111	5296.14	3180.61	39.94

FIGURE 25. Comparison between real and predicted pruning effectiveness. Serial execution times are presented with block pruning and without block pruning, using MASA-Serial tool (no parallelism).

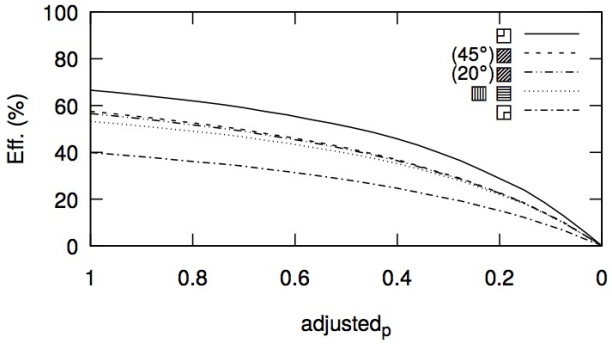


FIGURE 26. Pruning Effectiveness with varying sequence similarity.

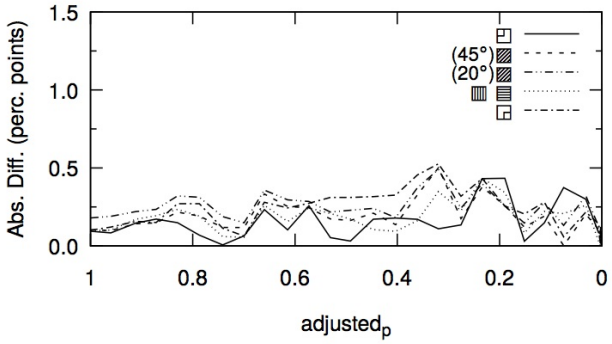


FIGURE 27. Absolute difference between predicted and real effectiveness.

Without Block Pruning, the best runtimes occurred when B is between 300 and 1000. The performance decreases a little when $B > 1000$ since there are fewer blocks and the parallelism is reduced.

With Block Pruning, the best execution times occurred when B is between 100 and 300. When $B > 300$, the performance starts to decrease due to the block pruning effectiveness reduction.

8. CONCLUSION AND FUTURE WORK

In this paper we explored Block Pruning, which is a pruning optimization able to reduce significantly the execution time of algorithms that compute the optimal alignment of similar sequences. This optimization was originally developed for the CUDAlign 2.1 tool [14] using diagonal processing and, at that moment, it was already clear that it was able to attain high pruning efficiency. In a tool called MASA (*Multi-Platform Architecture for Sequence Aligners*) [17], Block Pruning was extended to the dataflow (generic) processing, attaining in this case higher efficiency than the diagonal processing. This result motivated us to formalize the properties of the DP matrix and study the Block Pruning optimization in detail.

In this paper, we investigated the properties of the DP matrices used to compute optimal local and global alignments, with linear and affine gap models. As a result of this investigation, we formalized the inferior and superior bounds on the difference of the values of any two DP matrix cells. Using these bounds, we extended the Block Pruning optimization and proposed a Block Pruning method that uses these bounds to reduce the number of matrix cells calculated. In our pruning method, we defined a basic skeleton and proposed three different pruning algorithms that use the basic skeleton, processing the DP matrices row by row, by diagonal or in a generic order. In order to evaluate theoretically the effectiveness of the pruning method, we considered different scenarios of perfect

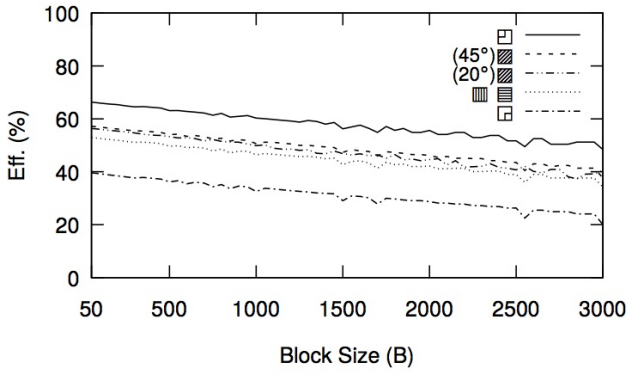


FIGURE 28. Pruning Effectiveness with varying block size (B).

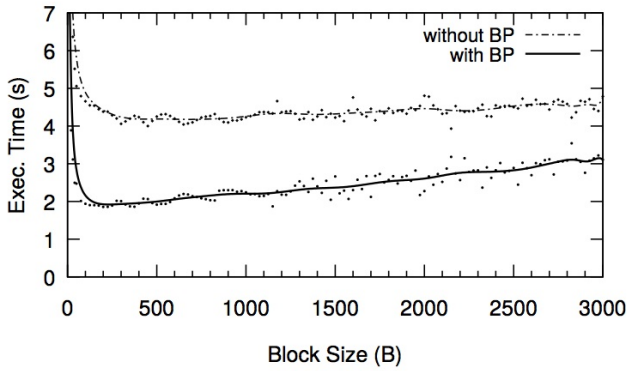


FIGURE 29. Execution times with MASA-OpenMP with varying block size (B), with and without block pruning.

match and partial match and six different processing orders, obtaining equations that calculate the area of a polygon defined by four lines. Solving the equations, we analyzed the impact of the similarity of the sequences, the punctuation of matches, mismatches, gaps and the processing order in the pruning effectiveness.

With the formulae and simulations, we observed that the square processing presented the best pruning results for cases in which the optimal alignment lies in the main diagonal. On the other hand, the anti-square processing presented the worst results for the same case. We also showed that the pruning results obtained when processing by rows/columns are on the median between the square and anti-square processing. In our results, we show that, when processing the DP matrix by square, up to 74.8% the DP matrices may be pruned when the gap penalties are high and up to 85% of pruning effectiveness may be reached when sequences are similar and have different lengths.

As future work, we plan to investigate other alignment patterns, such as: a) shifted alignments, which do not reside in the the main diagonal; b) overlapped alignments, in which two or more relevant alignments coexist; and c) truncated alignments, where the beginning/end of the alignment does not reside in the border of the DP matrix. Furthermore, we intend to investigate how to extend the block pruning method for the Multiple Sequence Alignment (MSA) problem, which was proven NP-hard when the alignments are scored with the sum-of-pairs function [21]. In this investigation, we will try to compare the pruning effectiveness of block pruning to other methods used to prune the search space of the MSA problem such as Carrillo-Lipman [22] and A-Star [23]. We also intend to adapt the block pruning method to problems usually solved with dynamic programming and have data dependency on the whole row and column such as RNA secondary structure prediction.

ACKNOWLEDGEMENTS

This work is supported by the Spanish Government through the Programa Hispano-Brasileño de Cooperación Interuniversitaria (PHBP14/00081), the Programa Severo Ochoa (SEV-2015-0493), by the Spanish Ministry of Science and Technology (TIN2015-65316-P) and by the Generalitat de Catalunya (2014-SGR-1051). We thankfully acknowledge the support of Capes/DGPU grant 306/2015 (Spain-Brazil joint research project) and CNPq/Brazil grants 313931/2013-5, 400493/2013-6, 305208/2014-4, 446297/2014-3 and 306705/2014-1.

REFERENCES

[1] Needleman, S. B. and Wunsch, C. D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, **48**, 443–453.

[2] Smith, T. F. and Waterman, M. S. (1981) Identification of common molecular subsequences. *Journal of Molecular Biology*, **147**, 195–197.

[3] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *Journal of Molecular Biology*, **162**, 705–708.

[4] Hirschberg, D. S. (1975) A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, **18**, 341–343.

[5] Myers, E. W. and Miller, W. (1988) Optimal alignments in linear space. *Computer Applications in the Biosciences*, **4**, 11–17.

[6] Ukkonen, E. (1985) Algorithms for approximate string matching. *Information and Control*, **64**, 100–118.

[7] G. M. Landau, J. P. S., E. W. Myers (1985) Incremental string comparison. *SIAM Journal of Computation*, **27**, 557–582.

[8] Barton, C., Flouri, T., Iliopoulos, C. S., and Pissis, S. P. (2013) GapsMis: flexible sequence alignment with a bounded number of gaps. *Int. Conf. on Bioinformatics, Computational Biology and Biomedical Informatics (BCB)*, New York, NY, USA, September 22–25, pp. 402–411. ACM.

[9] Hsu, P., Chen, K., and Chao, K. (2009) Finding All Approximate Gapped Palindromes. *Int. Symposium on Algorithms and Computation (ISAAC)*, Honolulu,

Hawaii, USA, December 16–18, pp. 1084–1093. Springer-Verlag, Berlin.

[10] Hyyro, H. and Inenaga, S. (2016) Compacting a Dynamic Edit Distance Table by RLE Compression. *Int. Conf. on Current Trends in Theory and Practice of Informatics (SOFSEM)*, Harrachov, Czech Republic, January 23–28, pp. 302–313. Springer-Verlag, Berlin.

[11] Fickett, J. W. (1984) Fast optimal alignment. *Nucleic Acids Research*, **12**, 175–179.

[12] Davidson, A. (2001) A fast pruning algorithm for optimal sequence alignment. *Bioinformatics and Bioengineering Conference (BIBE)*, Bethesda, Maryland, November 4–6, pp. 49–56. IEEE.

[13] Altschul, S. F., Gish, W., Miller, W., Myers, E. W., and Lipman, D. J. (1990) Basic local alignment search tool. *Journal of Molecular Biology*, **215**, 403–410.

[14] Sandes, E. F. O. and Melo, A. C. M. A. (2013) Retrieving smith-waterman alignments with optimizations for megabase biological sequences using gpu. *IEEE Trans Par Dist Syst*, **24**, 1009–1021.

[15] Korpar, M. and Sikic, M. (2013) SW# - GPU-enabled exact alignments on genome scale. *Bioinformatics*, **29**, 2494–2495.

[16] Okada, D., Ino, F., and Hagihara, K. (2015) Accelerating the Smith-Waterman algorithm with interpair pruning and band optimization for the all-pairs comparison of base sequences. *BMC Bioinformatics*, **16**.

[17] Sandes, E. F. O., Miranda, G., Martorell, X., Ayguade, E., Teodoro, G., and Melo, A. C. M. A. (2016) MASA: A Multiplatform Architecture for Sequence Aligners with block pruning. *ACM Trans Parallel Computing*, **2**.

[18] Mount, D. W. (2004) *Bioinformatics: sequence and genome analysis*. CSHL Press, Michigan.

[19] Durbin, R., Eddy, S., Krogh, A., and Mitchison, G. (1998) *Biological sequence analysis*. Cambridge University Press, Cambridge.

[20] Schelter, W. F. (2014). Maxima, a Computer Algebra System. Version 5.34.1. http://maxima.sourceforge.net/.

[21] Wang, L. and Jiang, T. (1994) On the complexity of multiple sequence alignment. *Journal of Computational Biology*, **1**, 337–348.

[22] Carrillo, H. and Lipman, D. (1988) The multiple sequence alignment problem in biology. *SIAM Journal of Applied Mathematics*, **48**, 1073–1082.

[23] Hart, P., Nilsson, N., and Raphael, B. (1968) A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science and Cybernetics*, **SSC-4(2)**, 100–107.

APPENDIX A. PROOF OF THEOREM 3.1

Proof. Theorem 3.1 will be proved by induction on the rows and columns of matrix H.

Base Case: As stated in Section 2.1, the first row ($i = 0$) and the first column ($j = 0$) of the DP matrix are initialized with $H_{i,0} = -G \cdot i$ and $H_{0,j} = -G \cdot j$ respectively. In addition, by Equations 7 and 8, we know that the cells of the second row ($i = 1$) and the second column ($j = 1$) respect the bounds defined by Equations A.1 and A.2, respectively.

$$(1-j)\cdot G - mi \leq H_{1,j} \leq (1-j)\cdot G + ma \quad \text{(A.1)}$$

$$(1-i)\cdot G - mi \leq H_{i,1} \leq (1-i)\cdot G + ma \quad \text{(A.2)}$$

The possible values of rows/columns 0 and 1 are shown in Figure A.1.

	j=0	j=1	j=2	j=3	···	-jG	···	-nG
i=0	0	-1G	-2G	-3G	···	-jG	···	-nG
i=1	-1G	+ma -mi	-1G+ma -1G-mi	-2G+ma -2G-mi	···	-(j-1)G+ma -(j-1)G-mi	···	-(n-1)G+ma -(n-1)G-mi
i=2	-2G	-1G+ma -1G-mi						
i=3	-3G	-2G+ma -2G-mi						
	···	···						
	-iG	-(i-1)G+ma -(i-1)G-mi						
	···	···						
i=m	-mG	-(m-1)G+ma -(m-1)G-mi						

FIGURE A.1. Possible values for rows/columns 0 and 1.

With these possible values for rows/columns 0 and 1, Figure A.2 presents, for any $1 \leq j \leq n$, the superior bound of $H_{1,j}$, the inferior bound of $H_{1,j-1}$ and the values for cells $H_{0,j-1}$ and $H_{0,j}$.

$H_{0,j-1} = (1-j)\cdot G$	$H_{0,j} = -j \cdot G$
$H_{1,j-1} \geq (2-j)\cdot G - mi$	$H_{1,j} \leq (1-j)\cdot G + ma$

FIGURE A.2. Bounds for cells adjacent to $H_{1,j}$

Using the values shown in Figure A.2, the superior bound of cell $H_{1,j}$ can be expressed by Equations A.3, A.4 and A.5 which validate the base case for DP cells that belong to the second row ($i = 1$).

$$H_{1,j} \leq H_{0,j-1} + ma \quad \text{(A.3)}$$

$$H_{1,j} \leq H_{0,j} + ma + G \quad \text{(A.4)}$$

$$H_{1,j} \leq H_{1,j-1} + ma - G + mi \leq H_{1,j-1} + ma + G \quad \text{(A.5)}$$

By symmetry, analogous results are obtained for the second column ($j = 1$).

Induction Hypothesis: For a given DP cell $H_{i,j}$, we assume that the theorem is true for its adjacent cells $H_{i-1,j}$ (Equation A.6), $H_{i,j-1}$ (Equation A.7) and $H_{i-1,j-1}$ (Equation A.8).

$$H_{i-1,j} \leq H_{i-1,j-1} + ma + G \quad \text{(A.6)}$$

$$H_{i,j-1} \leq H_{i-1,j-1} + ma + G \quad \text{(A.7)}$$

$$H_{i-1,j-1} \leq H_{i-2,j-2} + ma \quad \text{(A.8)}$$

Induction Step: The NW Equation (Equation 1) uses the maximum value of three terms. The value of function $p(i, j)$ is either $+ma$ or $-mi$ and, by the induction hypothesis, the maximum values of $H_{i-1,j}$

and $H_{i,j-1}$ are respectively defined by Equations A.6 and A.7. Hence, $H_{i,j} \leq H_{i-1,j-1}+ma$ (Equations A.9).

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} & + & sbt(S_0[i], S_1[j]) \\ H_{i-1,j} & - & G \\ H_{i,j-1} & - & G \end{cases}$$

$$H_{i,j} \leq \max \begin{cases} H_{i-1,j-1} & + & ma \\ (H_{i-1,j-1} + ma + G) & - & G \\ (H_{i-1,j-1} + ma + G) & - & G \end{cases}$$

$$H_{i,j} \leq H_{i-1,j-1} + ma \quad \text{(A.9)}$$

By the NW Equation, we obtain Inequalities A.10 and A.11.

$$H_{i-1,j-1} \leq H_{i,j-1} + G \quad \text{(A.10)}$$

$$H_{i-1,j-1} \leq H_{i-1,j} + G \quad \text{(A.11)}$$

Applying Inequalities A.10 and A.11 in Equation A.9, we get Equations A.12 and A.13.

$$H_{i,j} \leq H_{i-1,j-1} + ma \leq H_{i-1,j} + G + ma \quad \text{(A.12)}$$

$$H_{i,j} \leq H_{i-1,j-1} + ma \leq H_{i,j-1} + G + ma \quad \text{(A.13)}$$

Thus Theorem 3.1 holds for the remaining cells of the DP matrix and the proof of the induction step is complete.

Conclusion: By induction, it follows that Theorem 3.1 is true for all the DP cells $H_{i,j}$.

□

APPENDIX B. MAXIMA SCRIPT

In order to produce the equations presented in Table 4, we used the *Maxima* tool, which deals with resolution of algebraic equations. The source code shown in Figure B.1 was written in *Maxima* and it allows us to obtain the pruning effectiveness equations for the 6 ways of processing ($\phi_{i,j}$ - Equations 27 to 32), considering scenarios PM_r, $PM^{1.0}$ and PM^p.

```
/* definitions of phi */
phi(i,j):=i;                    /* row-by-row   */
phi(i,j):=j;                    /* col-by-col   */
phi(i,j):=(i+j)/2;              /* diagonal     */
phi(i,j):=(i*cos(th)+j*sin(th))/(cos(th)+sin(th));
    /*inc*/
phi(i,j):=max(i,j);             /* square       */
phi(i,j):=min(i,j);             /* anti-square */

/* solving equations f1, f2, f3 and f4 */
f1(i):=ev(rhs(solve(j=-phi(i,j)*ma*p/ma+m, j)[1]))
    ;
f2(i):=ev(rhs(solve(j=(-phi(i,j)*ma*p+m*ma+i*(G+ma
    *p))/(G+ma), j)[1]));
f3(j):=ev(rhs(solve(i=-phi(i,j)*ma*p/ma+m, i)[1]))
    ;
f4(j):=ev(rhs(solve(i=(-phi(i,j)*ma*p+m*ma+j*(G+ma
    *p))/(G+ma), i)[1]));

/* intersections s0:f1-f2 and s1:f3-f4 */
ii:ev(rhs(solve(f1(i)=f2(i), i)[1]));
jj:ev(rhs(solve(f3(j)=f4(j), j)[1]));

/* area computation */
area1:((-f1(0)*ii - f1(ii)*m + ii*m + m*m)/2);
area2:((-f3(0)*jj - f3(jj)*m + jj*m + m*m)/2);
area:area1+area2;

/* effectiveness computation */
factor(area/m/m);
```

FIGURE B.1. Code used to solve the area equations Maxima tool