Programming Research Group

INFORMATION CATEGORISATION IN BIOLOGICAL SEQUENCE ALIGNMENTS

Sumedha Gunewardena

PRG-RR-04-24

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Abstract

This is a two-part report. In the first part we introduce the reader to biological sequence alignment. We discus dynamic programming as is used in sequence alignment, first in the case of two sequences and later, how it is adopted for multiple sequence alignment. Several references are given to the different sequence alignment strategies reported in the literature used to enhance the standard dynamic programming algorithm for sequence alignment to suit biological sequences. A short discussion on how alignments are scored is given. Finally, some of the existing sequence alignment tools are described.

The second part of this report presents a critical analysis of information as it relates to biological sequence alignment. Information relating to the sequences being aligned form the basis on which any alignment is built. In its basic form this information might quantify how individual residues are scored when aligned with each other or how gaps are scored when introduced between two residues. Every biological sequence has if not explicit, at least some form of implicit information relating to its residues that form distinguishing markers along the sequence. There are many ways of extracting this information such as from databases of the relevant sequences, from the literature, prior processing etc. It is reasonable to assume that the more sequence information we use in an alignment, the more confidant we can be of the resulting alignment, and hence make better hypothesis of the unknown sequences. The aim of this part of the report is to build a framework on how to represent this information in such a way as to facilitate the dynamic and flexible incorporation of it to facilitate sequence alignments.

Part I

1 Sequence Alignment

1.1 Introduction

One common way to extract information from biological sequence data is to construct an alignment of a group of similar sequences. These alignments are often computed using dynamic programming techniques [21, 41, 49]. As the number of sequences to be aligned grows, the computational challenge of efficiently calculating useful alignments also increases.

It is possible to solve this multiple sequence alignment (MSA) problem using a direct extension of the standard two-dimensional dynamic programming algorithm initially introduced to the biological sequence alignment literature by Needleman and Wunsch [41] for aligning two sequences. However, this approach is generally considered impractical due to its $O(2^k N^k)$ time complexity (for k sequences of length N). The inherent computational difficulty of this problem has been demonstrated by Wang and Jiang [57], who proved that the optimal sum of pairs alignment problem is NP-complete. As a result many sub-optimal strategies have been suggested for the MSA problem [12, 10, 54], the most common of which is the progressive multiple sequence alignment technique [6, 17, 30, 58], which works by constructing a succession of pair-wise alignments. The progressive alignment technique works as follows: initially, two sequences are chosen and aligned using the standard two-dimensional dynamic programming algorithm. This alignment is then fixed and a third sequence is aligned with it. This process is iterated until all the sequences have been aligned. In general, one can extend the process by calculating a pair-wise alignment between two sequences, between a sequence and an alignment, or between two alignments. This method of aligning multiple sequences is not guaranteed to be mathematically optimal in the alignments that it produces, but can efficiently align a large number of sequences.

Various different strategies have been proposed to improve the sensitivity of the progressive MSA technique. Initial sequence ordering [52, 54, 6, 19], sensitive scoring schemes [33], iterative alignment strategies [6, 8, 22], sequence weighting [54], position specific gap penalties [54] and dynamic selection of residue comparison matrices [54] are some of the features found in existing alignment tools to improve the quality of alignments they produce. Non-conventional strategies such as those used in the MATCH-BOX multiple sequence alignment program by Depiereux et al [14] which builds alignments by recursively joining regions or boxes of complete matches common to all sequences; methods that use segment to segment comparisons instead of single residue comparisons [38]; alignments constructed from sub-alignments extracted from identified common points in several different alignments of the same sequences [11]; and genetic algorithm based multiple sequence alignment [42], have also been proposed as different means of incorporating local context in multiple sequence alignments to improve their quality.

Most current sequence alignment algorithms, though inspired by natural evolutionary models of genetic formation, do not take into account any additional specific information, which is available for the sequences being aligned. This deficiency is now beginning to be addressed by a number of authors. For example, Notredame et al. [43] describes an algorithm implemented in the program T-Coffee that uses a library of global and local pair-wise alignment information of the sequences to drive the progressive multiple sequence alignment. Lenhof et al [36] tackles the problem of optimally aligning a given RNA sequence of unknown structure to one of known sequence and structure (i.e., with known base pairings). Jennings et al [32] describe a method which attempts to improve the alignment produced by CLUSTALW and HMMer [16] by iteratively refining it using the secondary structure prediction from DSC [34]. The resulting alignment is reported to be more accurate and to yield higher prediction accuracy than the initial CLUSTALW / HMMer alignments. The multiple sequence alignment tool DbClustal [55] modifies the global alignment program CLUSTALW to incorporate local alignment data in the form of a list of anchor points between pairs of sequences. The program encourages the alignment of conserved motifs within the global alignment by weighting the global alignment towards these anchor points. Sander and Schneider [48] describes a way of incorporating secondary structure information in multiple sequence alignment by increasing the gap penalties within the α-helical and β-strand regions to discourage the introduction of gaps within these core secondary structure elements. The multiple sequence alignment tool PRALINE [27] combines secondary structure induced multiple alignments with position dependent amino acid exchange matrices to build a hierarchical multiple sequence alignment. Taylor

[53] describes a method of pair-wise protein sequence alignment that tries to capture the structural knowledge in multiple sequences by emphasizing matches of a preferred length through the use of a product-bias.

The approach to sequence alignment described in this report exploits the fact that many different functional and structural elements (e.g. secondary structure, binding sites, hypersensitive sites, etc.) are now known and well documented from detailed studies of DNA, RNA and protein sequences. This prior knowledge is often readily available to the sequence analyst, and tends to be incorporated by manually adjusting the alignments after they are computed, or else hard coding a very restricted amount of this knowledge into the alignment algorithms. We describe a critical analysis of the different forms of features present and how they can be viewed independent of the sequences they operate on. Such independent perspective on information acting on biological sequences we believe would lead to more flexible and dynamic sequence alignment algorithms.

1.2 Sequence Alignment

A sequence alignment can be seen as the problem of aligning two or more sequences using a given set of operations subject to constraints intrinsic to the organisation of the final alignment. An optimal alignment will have the added constraint that the final alignment weighted by a given scoring scheme will be optimal. For example, the Levenshtein distance (or edit distance) formulated by Levenshtein [37] in 1965, sets out the problem of sequence alignment as a similarity measure between two strings, measured by the number of deletions, insertions, or substitutions required to transform one string to the other. The idea is to perform this transformation with least cost where the cost is measured by assigning a cost for each one of these operations and taking the aggregate score over all operations performed.

1.2.1 Pair-wise Sequence Alignment

Pair-wise sequence alignment forms the foundation for many of the existing sequence alignment algorithms. There are many different variants to pair-wise alignments based on the expected properties of the final sequences, and computationally optimal algorithms for solving them.

Definition 1.1 *Let $S = S_1, \dots, S_k$ be a set of sequences over some finite alphabet Φ, of lengths $l_{S_1}, \dots, l_{S_k}$.*

Given two sequences $S_i, S_j \in S$, a pair-wise sequence alignment *of these two sequences is a function that maps the two sequences S_i and S_j to the new sequences S_i' and S_j' both of fixed length L. The sequences S_i' and S_j' are obtained by inserting* gaps *or* spaces *(represented by the 'gap' or 'space' character $\Delta \notin \Phi$) between certain elements of these sequences.*

The alignment A_2, of these two sequences can be specified by a $2 \times L$ matrix whose rows represent the new sequences S_i' and S_j'. Note that alignments are constrained so that the associated matrix does not contain any columns containing only gaps.

The i^{th} character of a sequence $S_j \in S$ will be denoted by $S_j[i]$. The i^{th} column in an alignment A_2 will be denoted $A_2[i]$. A sequence can also be referred to as a *string* and we will use these two terms interchangeably. We call a sequence S_i a $sub-sequence$ of sequence S_j if we can transform S_j to S_i by deleting zero or more characters from S_j while preserving the order of the remaining characters in S_j. We call a sequence S_i a $sub-string$ of sequence S_j if S_i is a contiguous sub-sequence of S_j i.e. $S_j \in \{\Phi^* S_i \Phi^*\}$. The alphabet Φ for biological sequences is a set of bases. For example, $\Phi = \{A, C, G, T\}$ for DNA sequences, $\Phi = \{A, C, G, U\}$ for RNA and Φ consists of the set of 20 amino acids in the case of protein sequences.

Definition 1.2 *Given two aligned sequences $S_i', S_j' \in A_2$, and an index x, $1 \leq x \leq L$, we define the following terms.*

- ***Match,*** *if $S_i'[x] \in \Phi$ and $S_i'[x] = S_j'[x]$,*
- ***Mismatch,*** *if $S_i'[x], S_j'[x] \in \Phi$ and $S_i'[x] \neq S_j'[x]$,*
- ***Insertion/Deletion,*** *if $S_i'[x] = \Delta$ and $S_j'[x] \in \Phi$ or $S_j'[x] = \Delta$ and $S_i'[x] \in \Phi$.*

A scoring function $\sigma(S_i'[x], S_j'[x])$ is a function that assigns a value to any one of the above events. The total score of a pair-wise alignment is given by some aggregate function $\mathcal{A}$ by taking the aggregate of the individual σ scores of the aligned residue pairs, $\mathcal{A}_{k=1}^{L} \sigma(S_i'[k], S_j'[k])$. There are many different scoring schemes available, some of which will be discussed in more detail in section 1.3. In general, a scoring scheme can be characterised as either a *similarity measure* or a *distance measure*. A similarity measure is a value assigned to an alignment by an aggregate function such that the higher the value, greater the similarity between the sequences, and vice versa. Alignment algorithms that use a similarity measure try to maximise their objective function. A distance measure on the other hand is a positive value assigned to an alignment by an aggregate function such that greater the value, the greater the distance between the sequences, and vice versa. Identical sequences will have zero distance. Alignment algorithms that use a distance measure try to minimise their objective function.

Example 1.3 Figure 1 shows a sub-optimal alignment of two sequences S_1 and S_2. This alignment is scored using two scoring functions, σ_1 and σ_2. The function σ_1 is defined as a similarity measure while the function σ_2 is defined as a distance measure. The aggregate function for both cases is the summation function. □

Figure 1 is an example of a sub-optimal alignment where the aggregate score of the alignment is not optimal for the given scoring function. Our objective is to produce an optimal alignment for a given pair of sequences i.e. to maximize or minimize as appropriate the aggregate function $\mathcal{A}$ over all possible alignments of the given sequences. One way of achieving this is to try all possible alignments and select an alignment with the maximum score. There are $\binom{2n}{n} \approx \frac{2^{2n}}{\sqrt{\pi n}}$ possible global alignments between two sequences of length n [15]. This makes this method far too slow to the extent of being unpractical except for very short sequences.

Event			Score Function	
			σ_1	σ_2
Match	$\sigma(a,a)$		2	0
Mismatch	$\sigma(a,b)$	$= \sigma(b,a),\ a \neq b$	-1	1
Insertion/Deletion	$\sigma(a,\Delta)$	$= \sigma(\Delta,a)$	-2	2

S_1 = TAGGCTACT, S_2 = TGGCTACGAG

Alignment:	S_1'	T	A	G	G	C	T	A	C	T	Δ	
	S_2'	T	G	G	C	T	A	C	G	A	G	
Score:	σ_1	2	-1	2	-1	-1	-1	-1	-1	-1	-2	= -5
	σ_2	0	1	0	1	1	1	1	1	1	2	= 9

Figure 1: A sub-optimal pair-wise alignment.

1.2.2 Global Alignment

Initially proposed by Needleman and Wunsch [41] and later refined by others [21], dynamic programming is an efficient computational technique in producing optimal alignments for long pairs of sequences. It achieves this by building up an optimal alignment by using previous solutions for optimal alignments of smaller sub-sequences. The result of this process is what is known as a dynamic programming matrix. The optimal alignment can be obtained by tracing back a path in this matrix with the optimal solution.

Consider two sequences S_1 and S_2 of lengths l_{S_1} and l_{S_2} respectively.

Definition 1.4 *Let $V(i,j)$ be the value of an optimal alignment of the sub-strings $S_1[1]$, $\ldots, S_1[i]$ and $S_2[1], \ldots, S_2[j]$.*

$V(i,j)$ corresponds to the value in the cell (i,j) of the dynamic programming matrix. Note, that the dynamic programming matrix is indexed from $(0,0)$. The score of the optimal alignment of the two sequences S_1 and S_2 is given by $V(l_{S_1}, l_{S_2})$. We build the dynamic programming matrix recursively by starting from a basis and building on it to compute all values $V(i,j)$, where $0 \leq i \leq l_{S_1}$ and $0 \leq j \leq l_{S_2}$, in order of increasing i and j.

$$\begin{aligned}
&\textit{Basis:} \\
V(0,0) &= 0 \\
V(i,0) &= V(i-1,0) + \sigma(S_1[i], \Delta),\ \textit{for } i > 0 \\
V(0,j) &= V(0,j-1) + \sigma(\Delta, S_2[j]),\ \textit{for } j > 0
\end{aligned}$$

The basis says that if i characters of S_1 are to be aligned with 0 characters of S_2, then they must all be matched with spaces and vice versa for S_2.

$$\textit{Recurrence: } \forall\, i,j > 0$$

$$V(i,j) = max \begin{cases} V(i-1,j-1) + \sigma(S_1[i], S_2[j]), \\ V(i-1,j) + \sigma(S_1[i], \Delta), \\ V(i,j-1) + \sigma(\Delta, S_2[j]). \end{cases}$$

The recurrence assigns a value for $V(i,j)$ by taking the maximum value of

1. aligning the characters $S_1[i]$ and $S_2[j]$, considering the alignment of the sub-strings $S_1[1], \ldots, S_1[i-1]$ with $S_2[1], \ldots, S_2[j-1]$ or

2. aligning the character $S_1[i]$ with a space considering the alignment of the sub-strings $S_1[1], \ldots, S_1[i-1]$ with $S_2[1], \ldots, S_2[j]$ or

3. aligning the character $S_2[j]$ with a space considering the alignment of the sub-strings $S_1[1], \ldots, S_1[i]$ with $S_2[1], \ldots, S_2[j-1]$.

If the optimal alignment is to be retrieved, it is important to keep track of the direction in which $V(i, j)$ is computed for each cell in the dynamic programming matrix. This direction will be one of the following.

1. diagonally from cell $(i-1, j-1)$ to cell (i, j) of the matrix

2. horizontally from cell $(i-1, j)$ to cell (i, j) of the matrix

3. vertically from cell $(i, j-1)$ to cell (i, j) of the matrix

These directions correspond to the three possibilities mentioned above respectively as shown in Figure 2.

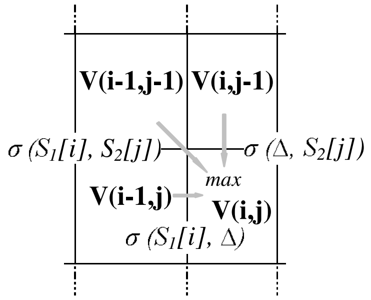

Figure 2: Computing the value of a dynamic programming matrix cell.

Example 1.5 Figure 3(a) shows the full dynamic programming matrix produced using the above algorithm and scoring function σ_1 for the two sequences S_1 an S_2 given in example 1.3. The first row and column of the matrix is initialised according to the basis function. The recursion function then fills in the matrix from top to bottom and left to right. For example, the entry in row 3 column 4 is obtained by computing $max\{(2+2), (0-2), (3-2)\} = 4$. The optimal alignment can be obtained by retracing the dynamic programming steps from the $(10, 9)$ cell entry to the $(0, 0)$ cell entry of the matrix along the path determined on the basis on, which preceding entry of a cell, was responsible for the current one. One such optimal path is highlighted in Figure 3(a). The arrows indicate the trace back path. The final alignment of the two sequences S_1 and S_2 obtained from this trace back is shown in Figure 3(b). □

			T	A	G	G	C	T	A	C	T
		0	1	2	3	4	5	6	7	8	9
	0	**0**	-2	-4	-6	-8	-10	-12	-14	-16	-18
T	1	-2	**2**	**0**	-2	-4	-6	-8	-10	-12	-14
G	2	-4	0	1	**2**	0	-2	-4	-6	-8	-10
G	3	-6	-2	-1	3	**4**	2	0	-2	-4	-6
C	4	-8	-4	-3	1	2	**6**	4	2	0	-2
T	5	-10	-6	-5	-1	0	4	**8**	6	4	2
A	6	-12	-8	-4	-3	-2	2	6	**10**	8	6
C	7	-14	-10	-6	-5	-4	0	4	8	**12**	10
G	8	-16	-12	-8	-4	-3	-2	2	6	**10**	11
A	9	-18	-14	-10	-6	-5	-4	0	4	**8**	9
G	10	-20	-16	-12	-8	-4	-6	-2	2	6	**7**

(a)

Alignment:	S_1'	T	A	G	G	C	T	A	C	Δ	Δ	T	
	S_2'	T	Δ	G	G	C	T	A	C	G	A	G	
Score:	σ_1	2	-2	2	2	2	2	2	2	-2	-2	-1	= 7

(b)

Figure 3: (a) The complete dynamic programming matrix for aligning the two sequences given in example 1.3. The arrows indicate the traceback path of an optimal alignment. (b) The optimal alignment given by the traceback path.

The dynamic programming algorithm has to compute a $(l_{S_1} + 1) \times (l_{S_2} + 1)$ matrix of entries. Any one entry in this matrix can be computed in a constant time c, with c being greatest for non initialisation cells of the matrix where 6 table lookups, 3 additions, and a three way maximum needs to be computed. Thus the complexity of the algorithm is given by $(l_{S_1} + 1)(l_{S_2} + 1)c = \bigcirc(l_{S_1} l_{S_2})$. The length of the longest possible alignment of S_1 and S_2 is the distance $l_{S_1} + l_{S_2}$ (this is when both sequences S_1 and S_2 are aligned with spaces). Therefore, the reconstruction of the optimal alignment can be done in a time of $\bigcirc(l_{S_1} + l_{S_2})$.

The dynamic programming algorithm as given above needs to produce a $(l_{S_1} + 1) \times (l_{S_2} + 1)$ matrix which requires $\bigcirc(l_{S_1} l_{S_2})$ space. There are modifications to the dynamic programming algorithm that can perform in $\bigcirc(l_{S_1} l_{S_2})$ time but $\bigcirc(l_{S_1} + l_{S_2})$ space [29].

1.2.3 Local Alignment

Unlike a global alignment algorithm, which aligns sequences optimally over their entire length, a local alignment algorithm only looks to align optimally sub-sequences of those sequences. Local alignments are good in instances where sequences show isolated regions of similarity, such as comparing multi-domain proteins where not all domains show signs of homology. Local alignments are better suited for aligning very highly diverged sequences and non-coding regions. In these otherwise weakly related sequences, it is more likely to find short conserved regions rather than conservation throughout the entire sequences.

Maximal Segment Pairs (MSP)

A maximal segment pair between two sequences is an un-gapped local alignment of a conserved region of the two sequences with optimal score.

Definition 1.6 *Consider the two sequences* $S_1 = S_1[1], \dots, S_1[l_{S_1}]$ *and* $S_2 = S_2[1], \dots, S_2[l_{S_2}]$ *of length* l_{S_1} *and* l_{S_2} *respectively. A compatibility matrix* $C = \| c(i,j) \|_{l_{S_1} \times l_{S_2}}$*, for* S_1*,* S_2 *is defined as the* $l_{S_1} \times l_{S_2}$ *matrix whose elements* $c(i,j) = \sigma(S_1[i], S_2[j])$ *are given by the scoring function* σ*.*

Definition 1.7 *We define a diagonal of* C *of length* $l+1$ *as a consecutive sequence of pairs* $d = (S_1[i], S_2[j]), (S_1[i+1], S_2[j+1]), \dots, (S_1[i+l], S_2[j+l])$*,* $1 \leq i \leq l_{S_1}$*,* $1 \leq j \leq l_{S_2}$ *and* $1 \leq l \leq min(l_{S_1} - i, l_{S_2} - j)$ *where* $S_1[i], S_1[i+1], \dots, S_1[i+l]$ *is a sub-string of* S_1 *and* $S_2[j], S_2[j+1], \dots, S_2[j+l]$ *a sub-string of* S_2*. The score of a diagonal is given by the aggregate function* $\mathcal{A}$ *taken over the scores of the individual elements of the diagonal,* $\mathcal{A}_{k=0}^{l} C(i+k, j+k)$*. A diagonal is said to be locally optimal if and only if its score cannot be improved by either extending or shortening it.*

Definition 1.8 *A maximal segment pair (MSP) is the un-gapped alignment of the two sub-strings associated with a locally optimal diagonal and is considered a local alignment of the two sequences* S_1 *and* S_2*.*

Definition 1.9 *We define a diagonal line of* C *as a diagonal with either* ($i = 0$ *and* $j + l = l_{S_2}$) *and/or* ($j = 0$ *and* $i + l = l_{S_1}$)*. The density of a diagonal line is defined as the sum of the scores of all the locally optimal diagonals on the diagonal line.*

MSP is a very simple and fast method of finding un-gapped locally optimal sub-string alignments. In some instances, one may be interested in all locally optimal alignments in which case all MSPs above a given threshold value can be selected. The *BLAST* program is a good example of the use of MSPs. The aggregate function used for computing MSPs is the summation function. We call the compatibility matrix C a *dot plot* if the scoring function is defined as $\sigma(a,b) = \begin{cases} 1 & \textit{if } a = b \\ 0 & \textit{otherwise} \end{cases}$.

Optimal Local Alignment

An optimal local alignment algorithm computes a local alignment between two sequences but unlike a MSP, it allows for insertions and deletions between the aligned sub-sequences. The algorithm for finding optimal local alignments is similar to the global alignment algorithm described above, and was first proposed by Smith and Waterman [49]. The dynamic programming matrix for the local alignment algorithm is centred at zero. The basis function therefore initialises the 0^{th} row and column of the dynamic programming matrix to zero.

$$
\begin{array}{lcl}
\textit{Basis:} & & \\
V(0,0) & = & 0 \\
V(i,0) & = & 0 \; \textit{for} \; i > 0 \\
V(0,j) & = & 0 \; \textit{for} \; j > 0
\end{array}
$$

The recurrence equations have the extra possibility of the value zero added to it.

$$
\textit{Recurrence:} \; \forall \, i,j > 0
$$

$$
V(i,j) = max \begin{cases} V(i-1,j-1) + \sigma(S_1[i], S_2[j]), \\ V(i-1,j) + \sigma(S_1[i], \Delta), \\ V(i,j-1) + \sigma(\Delta, S_2[j])), \\ 0 \end{cases}
$$

The option zero corresponds to starting a new sub-alignment. The maximum local alignment score for the two sequences is given by the maximum score in the dynamic programming matrix, i.e. the highest value of $V(i,j)$ over the whole matrix. The optimum local alignment is given by the traceback starting from this cell to the first cell along the path with value zero.

For the local alignment algorithms to produce sensible results, there are two considerations to the scoring function σ that must be taken into account. The expected score for a random match must be negative and at least one score $\sigma(a,b), \exists \, a, b \in \Phi$, must be positive. This point holds for both MSP and optimal local alignment.

1.2.4 Scoring Insertions and Deletions

	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0
Sequence S_1 :	L	N	D	P	N	A	F	L	K	K	F	L	T	D	K	G	K	A	D	Q	A	V	G	V	T	K	M	T	F	K
Sequence S_2 :	I	R	D	P	K	A	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ	K	I	P	G	T	K	M	I	F	A

Figure 4: Alignment of two sequences S_1 and S_2. It is not possible to say whether S_1 has undergone an insertion or whether sequence S_2 has undergone a deletion without knowing the ancestral sequence of S_1 and S_2.

A gap in an alignment is the result of an *insertion* or *deletion* event. The two events are symmetric, i.e. given two sequences S_1 and S_2 (see Figure 4) with at least one of these events, we cannot determine whether sequence S_1 has undergone an insertion or whether sequence S_2 has undergone a deletion unless we know their ancestral sequence, therefore both insertion and deletion events are treated alike. The length of a gap ($\#gap$) is the number of consecutive *insertion/deletion* events. For the alignment in Figure 4, $\#gap = 14$. When gaps occur in an alignment, we usually expect them to be longer than a single residue. So far our gap model does not discriminate between different gap lengths. This type of model tends to create a scatter of short gaps throughout an alignment, which may not be ideal for biological sequences. Knight and Myers [35] introduced a concave gap scheme which penalizes gaps based on its length. With such gap function, the modified recurrence relations of the dynamic programming algorithm will be as follows:

$$\textit{Recurrence: } \forall\, i,j > 0$$
$$V(i,j) = max \begin{cases} V(i-1,j-1) + \sigma(S_1[i], S_2[j]), \\ V(k,j) + \gamma(i-k),\; k = 0,\dots,i-1, \\ V(i,k) + \gamma(j-k),\; k = 0,\dots,j-1.) \end{cases}$$

For some gap function $\gamma(\#gap)$.

This procedure requires $\bigcirc(l^3)$ operations to align two sequences of length l. As this can be a prohibitively costly increase in computational time, we resort to an alternative gap model which uses an *affine* gap cost structure and performs in $\bigcirc(l^2)$ time.

In the standard affine gap model, a penalty for a gap consists of a *gap initiation cost* and a *gap extension cost* which is a penalty that depends linearly on the length of the gap. The cost of a gap now becomes $W_g + (\#gap)\ W_s$, where W_g is the gap initiation cost and W_s, the gap extension cost. They are both negative constants.

The goal is now to maximize

$$\sum_{i=1}^{L} \sigma(S_1'[i], S_2'[i]) - W_g \times (number\ of\ gaps\ initiated) - W_s \times (total\ number\ of\ gaps)$$

The dynamic programming algorithm with an affine gap model will be as follows:
The algorithm, as in section 1.2.2 proceeds by aligning $S_1[1],\dots,S_1[i]$ with $S_2[1],\dots,$ $S_2[j]$ for $0 < i \le l_{S_1}$ and $0 < j \le l_{S_2}$, in order of increasing i and j.

Definition 1.10 *Let $G(i,j)$ be the maximum score of aligning $S_1[1],\dots,S_1[i]$ with $S_2[1],$ $\dots,S_2[j]$ given $S_1[i]$ is aligned with $S_2[j]$.*

Definition 1.11 *Let $F(i,j)$ be the maximum score of aligning $S_1[1],\dots,S_1[i]$ with $S_2[1],$ $\dots,S_2[j]$ given $S_1[i]$ is aligned with a gap.*

Definition 1.12 *Let $E(i,j)$ be the maximum score of aligning $S_1[1],\dots,S_1[i]$ with $S_2[1],$ $\dots,S_2[j]$ given $S_2[j]$ is aligned with a gap.*

$$\begin{array}{lcl} \textit{Basis:} & & \\ V(0,0) & = & 0, \\ V(i,0) & = & -W_g - i\ W_s,\ for\ i > 0, \\ V(0,j) & = & -W_g - j\ W_s,\ for\ j > 0, \\ F(0,j) & = & -\infty,\ for\ j > 0, \\ E(i,0) & = & -\infty,\ for\ i > 0. \end{array}$$

$$\begin{array}{rcl} \multicolumn{3}{l}{\textit{Recurrence: } \forall\, i,j > 0} \\ V(i,j) & = & max(G(i,j), F(i,j), E(i,j)), \\ G(i,j) & = & V(i-1,j-1) + \sigma(S_1[i], S_2[j]), \\ F(i,j) & = & max(F(i-1,j) - W_s, V(i-1,j) - W_g - W_s), \\ E(i,j) & = & max(E(i,j-1) - W_s, V(i,j-1) - W_g - W_s). \end{array}$$

As before, we can find the optimal alignment using the traceback procedure.

We will need the equivalent of four matrices to be filled in simultaneously, each with a size of $(l_{S_1}+1)\times(l_{S_2}+1)$. This can be computed in time $\bigcirc(l_{S_1}l_{S_2})$. The trace-back can be done in time $\bigcirc(l_{S_1}+l_{S_2})$. The space utilisation would be $\bigcirc(l_{S_1}l_{S_2})$.

1.3 The Structure of Substitution Matrices

The premise behind sequence alignment is to identify regions in sequences that are evolutionarily related to each other. In this sense, the substitution scores used in an underlying alignment model provide a means of quantifying this relationship. The scoring scheme used in such a model should therefore distinguish biologically apposite substitutions from those that might arise simply by chance. Biological sequences evolve in time. Different sequences reflect different divergence times from a common ancestor. It is reasonable to expect sequences that have diverged recently in time from a common ancestor to display more identical residues in their sequences than those that have diverged at a much longer distance in time. In these circumstances one would expect to see the probability P_{ab} of aligning two identical residues (i.e $a=b$) to be large while the probability of aligning non-identical residues (i.e $a\neq b$) to be small in the case of sequences that have diverged recently, and for those sequences that have diverged long ago in time one would accept the probability P_{ab} to approach the background frequency $(q_a q_b)$ of the residues. It is therefore important to select a scoring scheme that reflects the amount of divergence of the sequences being compared.

A random probability model R is a simple model based on the assumption that individual residues occur independently from each other. Let p_a be the frequency at which some residue a occur in this model. The probability of aligning two residues a, b according to this model will be simply the product of the probabilities of the individual residues.

$$P(a, b \mid R) = p_a p_b$$

A match model M assumes a dependence between aligned residues i.e. they are related by some evolutionary distance t. This dependence can be expressed as a joint probability p_{ab} for aligning residue pairs a and b.

Intuitively it is reasonable to assume a substitution score to reflect the likelihood that two residues are aligned because they are evolutionary related as opposed to a random match. A *likelihood ratio* expresses the comparative difference between two models. Using the random model as a benchmark, one can express the likelihood of a residue pair occurring as an aligned pair as opposed to an unaligned pair. One can define an additive scoring scheme by taking the logarithm of the likelihood ratio to give a log-likelihood ratio.

$$\sigma(a, b) = s(a, b) = log\left(\frac{p_{ab}}{p_a p_b}\right)$$

The score of an alignment can be calculated by taking the sum of the log-likelihood ratio over all the aligned pairs of residues.

$$S = \sum_i s(S_1'[i], S_2'[i])$$

This is a Markovian model of evolution based on the assumption that residues mutate randomly according to some probability intrinsic to itself but independent from each other. It does not model influences on the mutation of an individual residue by neighbouring residues. It is also a time-reversible model i.e. $s(a,b) = s(b,a)$.

Sequence divergences are usually modelled using a substitution matrix M. They describe the likelihood of residue mutations over a given period of evolution.

$$Pr\{residue\ i \longrightarrow\ residue\ j\} = M_{ij}$$

The *PAM* matrices [13] and the *BLOSUM* matrices [25] are two of the most widely used substitution matrix families for amino acids in bioinformatics. They differ from each other in the approach taken to computing the joint probability p_{ab}.

1.4 Multiple Sequence Alignment

The probability of a region being homologous and therefore having functional and/or structural significance increases when conservation is observed over a whole set of sequences. Pair-wise sequence alignments are locally optimal and therefore may not show subtle similarities that may only be prominent when observed on a multiplicity of sequences. Multiple sequence alignments help in different areas of sequence analysis including the characterisation of protein families, determination of consensus sequences, identification of highly conserved residues or buried hydrophobic residues, prediction of secondary and tertiary structures and the construction of phylogenetic trees.

Definition 1.13 *Let $S = S_1, \ldots, S_k$ be k sequences over some finite alphabet Φ, of lengths $l_{S_1}, \ldots, l_{S_k}$.*

A multiple sequence alignment *of these k sequences is a function that maps each sequence S_i to a sequence S_i' of fixed length l_A, for $i = 1, 2, \ldots, k$. Each sequence S_i' is obtained by inserting* gaps *between certain elements of sequence S_i.*

An alignment A, can be specified by a $k \times l_A$ matrix whose rows represent the new sequences $S_1' \ldots S_k'$. Note that alignments are constrained so that the associated matrix does not contain any columns containing only gaps.

The i^{th} sequence S_i' contributing to the alignment i.e. the i^{th} row of the alignment matrix will be denoted as A^i. The jth column in an alignment A will be denoted $A[j]$. The length of alignment A will be denoted $\#A$. An alignment will sometimes be subscripted by the number of rows in the alignment matrix, for example, an alignment A of k sequences can also be denoted as A_k.

Definition 1.14 *An* optimal *multiple sequence alignment is an alignment that optimises the positional similarity over all columns of the resulting matrix according to a given scoring and aggregate function.*

Given a cost function β on Φ^k, an optimal alignment is one that optimises the aggregate of the costs taken over all columns of the alignment matrix i.e. optimise $\mathcal{A}_j\beta(A^1[j],\ldots,A^k[j])+G$. The aggregate function $\mathcal{A}$ is usually the summation function. G is a function for scoring gaps in the alignment. This general scoring function assumes statistical independence between columns in the alignment There is no unique way of defining the cost function β. The most common definition of β is the *sum of pairs (SP)* score given by $\beta(A^1[r],\ldots,A^k[r]) = \sum_{1\leq i<j\leq k} \sigma(A^i[r],A^j[r])$. An optimal alignment measured using this scoring function emphasises the pair-wise alignment of the individual sequences being aligned. It tends to maximise the agreeability of residues at a given aligned position. One difficulty with this method is the question of scoring gaps. It is not possible to estimate the number of evolutionary events responsible for the gaps without knowing the phylogeny responsible for the alignment. The method usually adopted for scoring gaps in a SP scores is to treat insertion/deletion events according to what can be seen locally in the alignment.

Given k sequences $S_1,\ldots,S_k$ each with length $l_{S_1},\ldots,l_{S_k}$, a generalisation to the dynamic programming algorithm can compute the optimal alignment of the k sequences by filling in a k dimensional matrix $(l_{S_1}+1)\times(l_{S_2}+1)\ldots\times(l_{S_k}+1)$. If all the sequences were of the same length l, there would be $(l+1)^k$ entries to fill. Each one of these entries would depend on 2^k-1 adjacent entries corresponding to all possible combinations of aligning with gaps, except for the case in which a column of gaps are aligned with each other. As $(l+1)^k$ entries can be computed in a time proportional to 2^k, the running time of the general dynamic programming algorithm is $\bigcirc((2l)^k)$.

The general dynamic programming algorithm will need a k dimensional matrix $(l_{S_1}+1)\times(l_{S_2}+1)\ldots\times(l_{S_k}+1)$. If we consider all the sequences to be of the same length l, this would give us a space complexity of $\bigcirc(l^k)$.

These costs are prohibitive even for a moderate set of sequences. Wang and Jiang [57] demonstrated that the optimal sum of pairs alignment problem is NP-complete. In light of these observations we look into other more computationally efficient alignment techniques, which may not produce the optimal alignment but give a good estimate of it. This approximation may have some justification from the fact that a mathematically optimal alignment may not necessarily be a biologically optimal alignment for the given set of sequences.

1.4.1 Progressive Multiple Sequence Alignment

The basic progressive alignment algorithm proceeds by successively applying the pair-wise dynamic programming algorithm to two sequences, a sequence and an alignment or two alignments. An alignment, once produced is fixed and does not change in successive alignments. This is the basis of the progressive multiple sequence alignment algorithm proposed by Feng and Doolittle [17]. Their algorithm starts off by constructing a matrix of distances between all $k(k-1)/2$ sequence pairs by converting raw alignment scores to pair-wise distances. The order in which sequences are aligned is then determined by a guide tree constructed using the Fitch and Margoliash [18] clustering algorithm on these distances. A sequence is added to an alignment by aligning the sequence to the sequence in the alignment that gives optimal pair-wise score. When two alignments are aligned, the resulting alignment is governed by the two sequences from the two alignments with optimal pair-wise score.

The *centre star* alignment algorithm proposed by Gusfield [23] is a heuristic iterative alignment algorithm with fixed bounds that states that, the sum of pairs score of a centre star alignment is at most twice as bad as that of the optimum alignment. The algorithm starts by first finding the sequence $S_i \in S$ that optimises $\sum\limits_{S_j \in S \setminus \{S_i\}} \nu(S_i, S_j)$ where the function $\nu(S_1, S_2)$ returns the value of the optimal global alignment between sequences S_1 and S_2. The sequences $S \setminus \{S_i\}$ are then aligned against S_i one at a time. Gaps introduced in the process will effect all the sequences already aligned to S_i. If the scores used were distances the centre star algorithm becomes an approximation algorithm to minimise the sum of pair-wise distances between the sequences in the alignment.

Many progressive multiple sequence alignment algorithms use a guide tree to direct the chronology of building an alignment. A guide tree can be built using a clustering algorithm such as UPGMA [50] or the neighbour-joining [47] method. Leaves of a guide tree represent the individual sequences while the internal nodes represent alignments. The nodes are traversed in their order of addition to the tree, progressively aligning the sequences. The final alignment is given by the alignment at the root of the tree.

Definition 1.15 *A partial-alignment is the alignment of any proper sub set of the sequences that contributes to the final alignment.*

At each traversal of the tree either two sequences are aligned or a sequence is aligned to a partial-alignment, or two partial-alignments are aligned. Given an alignment A_{m+n} of two partial-alignments A_m and A_n of m and n sequences respectively, if the column $A_{m+m}[r]$, $1 \leq r \leq \#A_{m+n}$ was the result of aligning columns $A_m[p]$, $\exists\, p : 1 \leq p \leq \#A_m$ and $A_n[q]$, $\exists\, q : 1 \leq q \leq \#A_n$, the score of aligning the two columns $A_m[p]$, $A_n[q]$ is given by $\frac{1}{mn} \sum\limits_i \sum\limits_j \sigma(A^i_m[p], A^j_n[q])$. A single sequence can be considered as a partial-alignment with just one sequence.

A problem with progressive alignment techniques is the dependence of successive alignments on the initial alignments produced, which are fixed. Iterative alignment strategies try to address this problem by successively removing and realigning sequences for a given number of iterations or until the alignment score converges. One such strategy proposed by Barton and Sternberg [6] is as follows: Given a multiple sequence alignment of $A_k = S'_1, \dots, S'_k$, remove sequence S'_1 and realign it to the profile $A_k \setminus \{S'_1\}$. Repeat this process for the remaining sequences $S'_2, \dots, S'_k$. These steps are iterated for a given number of times or until the alignment score converges. Note that when a sequence is removed from an alignment both the sequence and the alignment must be pruned to remove any column with only gaps.

1.5 Existing Sequence Alignment Programs

1.5.1 FASTA

FASTA is a heuristic alignment algorithm developed by Pearson and Lipman [44] for fast database search. The algorithm first identifies all short words (*k-tuples*) common to both the query and database sequence (k is usually set between $1-3$ for amino acids and $1-6$ for nucleotides). This can be done using a dot-plot of the two sequences. Next, the ten densest diagonal lines are identified. The diagonal regions in these diagonal lines are then re-scored using a substitution matrix allowing for substitutions but not insertions or deletions. Dynamic programming is used to produce the final alignment (which may include gaps) taken from the regions with high scoring diagonals. The larger the k value, the smaller will be the number of exact matches and as a consequence the program will be faster but less sensitive.

1.5.2 BLAST

The BLAST algorithm was developed by Altschul et al.[1]. It is also a heuristic based fast database search algorithm. The first step in the BLAST algorithm is to extract a list of fixed length (usually 3 for amino acids and 11 for nucleotides) words from the query sequence and using that, build a list of all un-gapped words that can score above a given threshold value when compared with the extracted word list using a given substitution matrix. Next, the database is searched using this word list. Each time a hit is encountered, it is extended in both directions to form a MSP. All such MSPs that score above a given threshold value are reported.

1.5.3 CLUSTAL W

CLUSTALW developed by Thompson et al. [54] is one of the most widely used multiple sequence alignment algorithms. It is a global alignment algorithm and works with both amino acids and nucleotides. It succeeded the CLUSTALV program [28]. The basic structure of the CLUSTALW algorithm is as follows; The algorithm first computes the pair-wise distances between all the $k(k-1)/2$ sequences. It then builds a guide tree using these distances. Sequences are aligned progressively according to the branching order of the guide tree. CLUSTALW achieves its enhanced performance from a few inbuilt modifications to this basic algorithm. Sequence weighting is performed to correct for any bias that may occur due to unequal sampling of sequences with different divergences. Weights are calculated directly from the branch lengths in the initial guide tree. Substitution matrices are selected, at different alignment stages, according to the divergence between the sequences. CLUSTALW uses residue specific gap penalties. Gaps are encouraged in hydrophilic regions rather than in regular secondary structure regions. Gaps are also more encouraged at positions when aligned to positions in early alignments where gaps have already been opened.

1.5.4 Lalign

Lalign is a pair-wise local alignment algorithm developed by William Pearson. It adopts the algorithm developed by Huang and Miller [31] in the SIM program. Lalign calculates the N-best non-intersecting local alignments between two sequences reporting these alignments and their scores. The N-best alignments do not share any aligned pairs. The algorithm is space efficient utilising space proportional to the sum of the input sequence lengths and output alignment lengths. The algorithm uses affine weights where a gap penalty is given by $W_g + (\#gap - 1)\ W_s$.

1.5.5 T-Coffee

T-Coffee is a progressive multiple sequence alignment tool developed by Notredame et al. [43]. It tries to address the pitfalls of the greedy nature of the progressive alignment strategy for multiple sequence alignment. The algorithm starts by building a library of alignment information for all $k(k-1)/2$ sequence pairs. It uses CLUSTALW [54] to compute the best global alignments and Lalign [31] to compute the 10 best local alignments between pairs of sequences. Information is stored on each aligned residue pair of these alignments as a list of pair-wise residue matches to form the library. These residue pairs are weighted based on their significance computed from the individual pair-wise alignments. Significance of an individual residue pair is measured as the level of sequence identity of its alignment. This library information is enhanced by examining the consistency, and upgrading the weights accordingly of each pair of residues with residue pairs of all other alignments. T-Coffee then, integrates this information. This is done by simple addition. For every residue pair in the library, all scores pertaining to it (obtained from the different alignment sources) are added to give it a grand total score. The result of all this is a list of weighted residue pairs. T-Coffee uses these weights instead of a weight matrix such as PAM or BLOSUM to match residues during alignment. No special treatment is made to score gaps during the alignment, as this information is taken care of during the pair-wise alignment stage and reflected in the library. The progressive multiple sequence alignment strategy of T-Coffee is similar to that of CLUSTALW.

1.5.6 DIALIGN

DIALIGN is a multiple sequence alignment program developed by Morgenstern et al. [39]. It is based on segment-to-segment comparisons rather than residue-to-residue comparisons. Alignments are constructed from gap free equal length segments called 'diagonals'. The first step of the algorithm is to identify significant diagonals from all $k(k-1)/2$ pair-wise alignments obtained from a dot-matrix. The significance of a diagonal is measured by its weight which is given by $-\ln P(l,m)$ if this value is greater than a given threshold or 0 otherwise. The value $P(l,m)$ is the probability of a diagonal of length l to consist of at least m matches $P(l,m) = \sum_{i=m}^{l} \binom{l}{i} p^i (1-p)^{l-i}$ where p is the probability of a single point in the dot-matrix to represent a match. In this step all diagonals greater than the threshold value are selected. In the next step a consistent set of diagonals for all pairs of sequences are obtained from a maximal alignment of the diagonals for the pair of sequences computed from a modified version of the standard pair-wise dynamic programming algorithm [38]. A multiple sequence alignment is produced using these diagonals. First all diagonals are sorted according to their weight irrespective of the initial pair-wise alignment they originated from. Diagonals are added in a greedy way to the multiple sequence alignment in descending order of their weight preserving the consistency of the alignment as they are added. Any diagonal that is inconsistent with the alignment constructed to the point of its addition is rejected. The final alignment is obtained by iteratively applying the above procedure to the parts of the sequences that have not yet being aligned keeping with the consistency of the existing alignment.

1.5.7 DbClustal

DbClustal is a progressive multiple sequence alignment program developed by Thompson et al. [56]. The program is designed to produce automatic multiple sequence alignments of the top scoring full-length sequences detected by database homology searches. Homologous sequences are obtained using a BlastP [2] database search with a query sequence. DbClustl works on a modified ClustalW platform, modified to accommodate local alignment data in the form of anchor points between pairs of sequences. The anchor points are generated using Ballast [45], a Blast post-processing program that identifies high scoring segment pairs in the sequences detected by Blast and creates a file containing a self-consistent set of anchors between the query sequence and the database hits. Each anchor point is given a score as described in Plewniak et al. [45]. The global alignment as computed by the ClustalW algorithm in DbClustal is encouraged towards these anchor points by a modification to the scoring function for scoring the alignment of two residues or columns to include the anchor score computed for the two residues or columns by simple addition.

Part II

2 Information

2.1 Information Categorisation

Almost all modern sequence alignment programs use knowledge encoded in a fixed substitution matrix (e.g. PAM, BLOSUM) in accounting for amino acid or nucleotide substitutions in an alignment. However we argue that incorporating knowledge to a sequence alignment algorithm should not be restricted to the substitution matrix. With the growth of our understanding of biological sequences, their structure and features, it is important to find ways of incorporating this knowledge into the alignment process. Lenhof et al. [36], for example, showed how knowledge of RNA sequence structure can be used to improve RNA sequence alignment. The popular sequence alignment program CLUSTALW uses knowledge of hydrophilic regions in a sequence to control gaps introduced in an alignment. Other alignment programs such as DbClustal and T-Coffee combine knowledge of alignment information obtained from different alignment sources such as BLAST searches, CLUSTALW global alignments, Lalign local alignments and other available alignment information of the sequences to produce alignments of possibly higher quality than the individual alignment programs.

Information on, or prior knowledge of, the sequences can be obtained from many different sources. It can come from experimental results of wet lab tests, or from the literature on the sequences being aligned. It can be obtained from databases, such as EPD[46], TRANSFAC[24], PDB[9] or can be knowledge gathered from annotation information obtained from databases such as Swiss-Prot[4], PIR[20], EMBL[40] and GenBank[7]. It can be information obtained from secondary pattern databases such as PRINTS[3], PROSITE[5], BLOCKS[26] and Pfam[51]. Alignment information can be obtained progressively for successive alignments from initial alignments of the sequences, or it can be obtained from initial alignments of the sequences produced by other alignment programs, such as CLUSTALW and Lalign. This knowledge can come from any one or a combination of these, or any other information source.

A common means of incorporating such knowledge in a program is to code the relevant details within its algorithm. Though this may have an advantage on speed, it lacks the flexibility of dynamically incorporating different forms of knowledge and their profiles appropriate to different contexts. With the rapid pace in which new knowledge on genetic sequences is being generated, the need for such dynamic systems capable of incorporating knowledge on demand and making inferences from that information is becoming ever greater. **We describe a formalism to address this problem in which information is modelled as standalone objects expressed independently of the sequences they operate on**. Objects are equipped with a knowledge base and are associated with families that define and monitor their behaviour. In this formalism, a sequence by itself is just a pattern with no inference value. Processing is done on induced sequences (formally defined later) which are sequences embedded with different objects imparting their behaviour and properties at the regions of embedding on the sequences. A diagrammatic description of this process is shown in Figure 5. To allow information to be defined in this form, we make the following definitions. We start with the basic object of a *sequence-factor.The definitions that follow are described in 'Z' notation [59].*

2.1.1 Sequence-Factor

Formally, a sequence-factor is defined as follows:

Definition 2.1 ***Sequence-factor*** $\mathbb{f}$ **:** $\mathcal{P} \times \mathcal{P} \times \operatorname{seq} \mathbb{N}^{+} \times \operatorname{seq} \mathbb{N} \times \mathbb{N}$

$$\mathbb{f} == (s, p, l, t, i) : \mathcal{P} \times \mathcal{P} \times \operatorname{seq} \mathbb{N}^{+} \times \operatorname{seq} \mathbb{N} \times \mathbb{N} \quad | \quad \begin{array}{ll} (\#p = \#l = \#t\,) & \wedge \\ s[l] = p & \wedge \\ s(1) = \ltimes & \wedge \\ s(\#s) = \rtimes & \end{array}$$

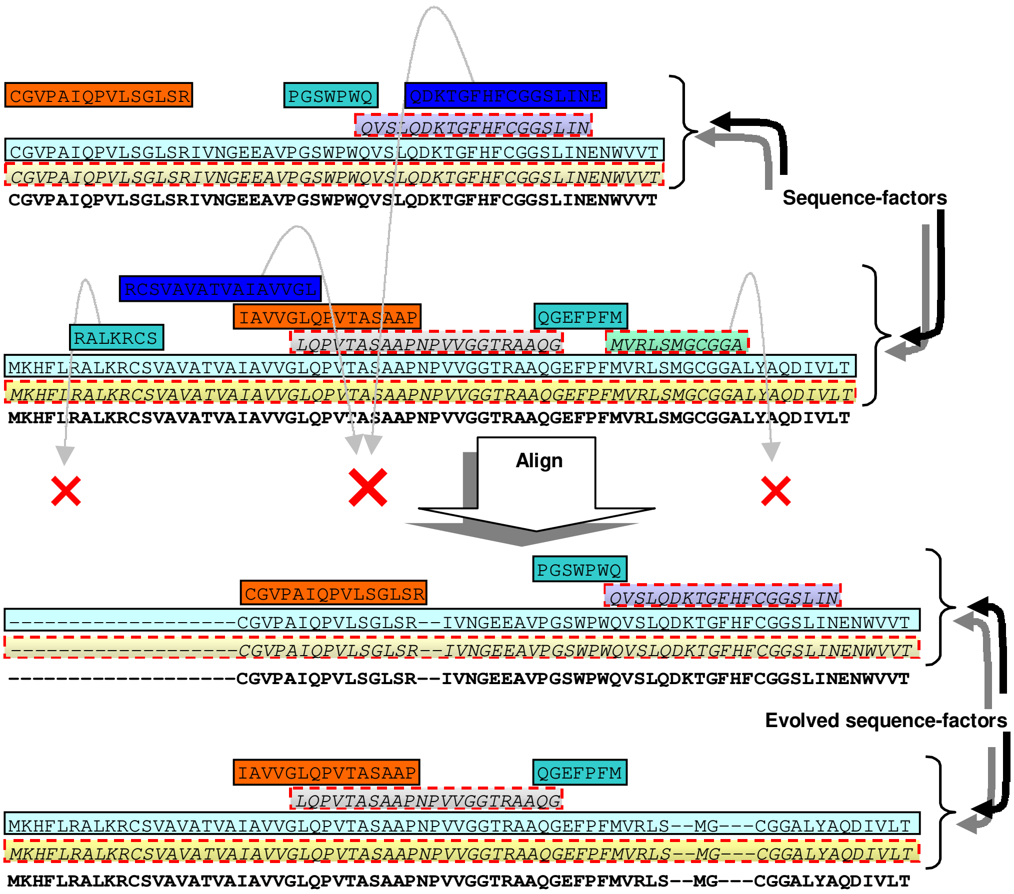

Figure 5: The figure shows two *induced sequences* before and after an *alignment*. The sequences are shown in bold characters. The coloured boxes represent *sequence-factors*. Sequence-factors *induce* information at their regions of interaction with the sequence. The boxes with perforated borders represent *homointeractive* sequence-factors. They act locally to influence the behaviour of the sequence within their region of influence i.e. the induced region. The boxes with solid borders represent *heterointeractive* sequence-factors. They interact with sequence-factors from the same *motif set* (shown by a distinct colour) on opposite sequences. An alignment causes sequence-factors to *evolve*. Some sequence-factors are destroyed in the process (as shown by the crosses) while others evolve to describe the resulting alignment. Sequence factors can be quantified individually (homointeractive) or in pairs (heterointeractive). Taking the aggregate over all its sequence-factors scores an alignment. An optimal alignment will maximise this value. There are two basic sequence-factors spanning the entire length of a sequence (shown by the first two sequence-factors induced on the sequences in the diagram) that are usually associated with an induced sequence. One is a homointeractive sequence factor that models the global gap penalty and the other is a heterointeractive sequence-factor that models the global substitution scores of the individual residues of the sequences.

$\mathcal{P}$ is the set of all valid 'sequences' or 'patterns' (see notations and basic definitions 15 on page 51).

In words, a *sequence-factor* is a quintuplet $\mathbb{f} == (s, p, l, t, i)$, where.

- $s \in \mathcal{P}$ is a target sequence where the first character of s is the *begin sequence* character $\ltimes$ and the last character of s is the *end sequence* character $\rtimes$.
- $p \in \mathcal{P}$ is a pattern.
- $l \in \operatorname{seq} \mathbb{N}^{+}$ is a location list.
- $t \in \operatorname{seq} \mathbb{N}$ is a indicator list.
- $i \in \mathbb{N}$ is the unique identifier of the motif set of the sequence-factor.

Satisfying the invariants $\#p = \#l = \#t$ and $s[l] = p$ where $\#s$ for a set s gives the number of elements in s, and $a[b]$ for two sequences a and b where b is a sequence of integers within the domain of a, returns the sub-sequence of s indexed by the range of b *(see notations and basic definitions 14 on page 51).* Figure 6 shows a fully instantiated sequence-factor. The target sequence s is the sequence onto which a sequence-factor binds.

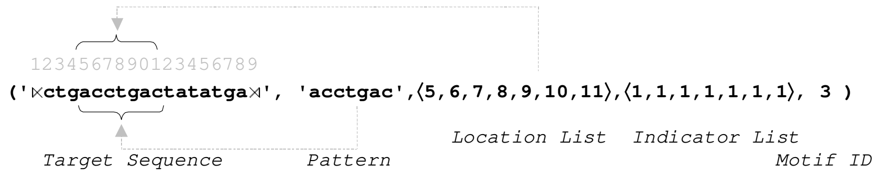

Figure 6: Components of a fully instantiated sequence-factor.

It starts with the begin sequence character $\ltimes$ and ends with the end sequence character $\rtimes$. The location list l describes the region of influence of the sequence-factor within the target sequence. The target sequence and the location list together can describe a complete instantiation of a sequence-factor. The pattern p is the sub-sequence of the target sequence indexed by the location list. As can be seen from the invariant $s[l] = p$, this information is implicit once the location list is provided for a given target sequence. The main function of the pattern is to give a more robust and expressive input structure to the sequence-factor. The indicator list t is used to describe the internal structure of the sequence-factor and is closely related to the scoring function associated with it. Sequence factors exist in sets. The identifier i is the **key** to associating a sequence-factor with the set it belongs to. It is unique to the sequence-factors of that set. The identifier plays an important part in determining the associations of different sequence-factors on induced sequences and helps in relating them to their sets of origin, which defines and monitors their behaviour. These sets are known as *motif sets*.

2.1.2 Motif-Set

A motif set is a collection of sequence-factors that exhibit a particular behaviour. Formally, a *motif set* $\mathcal{M}$ is defined as follows:

Definition 2.2 ***Motif Set*** $\mathcal{M} : \mathbb{P}\ \mathbb{P}\ \mathbb{f}$

$$\begin{array}{|l}
\mathcal{M} : \mathbb{P}\ \mathbb{P}\ \mathbb{f} \\
\hline
\forall f_1 : \mathbb{P}\ \mathbb{f} \bullet \\
\qquad f_1 \in \mathcal{M} \implies \forall f_2 \in \mathcal{M} \bullet \quad \begin{array}{l} f_1.5 = f_2.5 \quad \wedge \\ \left(\begin{array}{l} f_1.1 = f_2.1 \wedge \\ f_1.3 = f_2.3 \end{array} \right) \implies f_1 = f_2 \end{array}
\end{array}$$

The declaration of this definition states that a motif set is a collection of sequence-factors ($\mathbb{P}\ \mathbb{f}$) (i.e. an element of a collection of collections of sequence-factors ($\mathbb{P}\ \mathbb{P}\ \mathbb{f}$)). There are two constraints to this definition that apply to all sequence-factors in a motif set. The first constraint ensures that all sequence-factors in a motif set have the same identity. The identity of a sequence-factor associates it with the motif set. The second constraint ensures that every sequence-factor in a motif set is different and is fully identifiable by the target sequence, location list pair. This ensures that there are no structural conflicts between different sequence-factors in the set.

Example 2.3 Figure 7 shows 7 motif sets, labelled A to G. Each of these motif sets contain different sequence-factors corresponding to a particular β-chain of the Immunoglobulin domain [6] across a variety of proteins. The pattern of each sequence-factor represents the β-chain and each motif set represents a homologous family. They have been obtained experimentally by determining the 3-dimensional structure of these proteins. □

	PATTERN					
A	B	C	D	E	F	G
(⊛,'VFG', ⊛,⟨1, 2, 3⟩,⊛)	TISCTG	NVKWY	SVSKS	TSATLAI	YYCQSY	VLTQPP
(⊛,'VFG', ⊛,⟨1, 2, 3⟩,⊛)	TISCSG	TVNWY	SGSKS	ASASLAI	YYCAAW	QLVQSG
(⊛,'YWG', ⊛,⟨1, 2, 3⟩,⊛)	RLSCSS	AMYWV	TISRN	NTLFLQM	YFCARD	QLEQSG
(⊛,'VWG', ⊛,⟨1, 2, 3⟩,⊛)	SLTCTV	YWTWV	TMLVN	NQFSLRL	YYCARN	SVFLFP
(⊛,'IEK', ⊛,⟨1, 2, 3⟩,⊛)	EVTCVV	KFNWY	KTKPR	VVSVLTV	YKCKVS	SVTLFP
(⊛,'VEK', ⊛,⟨1, 2, 3⟩,⊛)	TLVCLI	TVAWK	GVETT	ASSYLSL	YSCQVT	SVFPLA
(⊛,'VDK', ⊛,⟨1, 2, 3⟩,⊛)	ALGCLV	TVSWN	GVHTF	LSSVVTV	YICNVN	QVYTLP
(⊛,'TQK', ⊛,⟨1, 2, 3⟩,⊛)	SLTCLV	AVEWE	NYKTT	LYSKLTV	FSCSVM	

Figure 7: 7 *position independent homogeneous* motif sets for immunoglobulins. All sequence-factors should be of the form $(\circledast, PATTERN, \circledast, \circledast, \circledast)$, as in motif set A. For brevity, only the pattern is shown for motif sets B-G. *The meta character* ⊛ *is defined in notations and basic definitions 9 on page 50.*

Note that we are not assuming that any of the sequence-factors in a motif set are necessarily instantiated with any of the given set of sequences for an alignment. In some cases it will be convenient to simply specify an entire database of properly defined sequence-factors and let the instantiation process determine which of the motifs actually occur in the given sequences. For example, if we are dealing with regulatory regions of DNA sequences, then it may be useful to specify a motif set for each known transcription factor, containing the possible binding sites for that factor. Note also that the definition of a motif set does not place any constraints on the length of its elements or assume any length relationship between different sequence-factors in the set. In some cases it may be useful to include sequence-factors with quite long patterns in a motif set in order to identify large sequence features.

2.1.3 Universal Motif Set Collection

The collection of all motif sets is called the *universal motif set collection* $\mathbb{M}$ and is defined as follows:

Definition 2.4 ***Universal Motif Set Collection*** $\mathbb{M} : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,\mathbb{f}$

$$
\begin{array}{|l}
\mathbb{M} : \mathbb{P}\,\mathbb{P}\,\mathbb{P}\,\mathbb{f} \\
\hline
\forall M : \mathbb{P}\,\mathbb{P}\,\mathbb{f} \bullet \\
\qquad M \in \mathbb{M} \Longrightarrow \left(\begin{array}{lll} \forall f \in M & \bullet & f.5 = idm(M) \quad \wedge \\ \forall f_1, f_2 \in M & \bullet & \left(\begin{array}{l} f_1.1 = f_2.1 \wedge \\ f_1.3 = f_2.3 \end{array} \right) \Longrightarrow f_1 = f_2 \end{array} \right)
\end{array}
$$

The universal motif set collection is the collection of all the collections of sequence factors that form a motif set ($\mathbb{P}\,\mathbb{P}\,\mathbb{f}$), i.e., an element of $\mathbb{P}\,\mathbb{P}\,\mathbb{P}\,\mathbb{f}$. Every motif set collection referred to will be a subset of $\mathbb{M}$, and every motif set an element of $\mathbb{M}$. Every motif set in $\mathbb{M}$ is assigned a unique identity described by the function $idm\_$ (the generic function $idm\_$ takes a set and assigns a unique number to each element of that set. See *notations and basic definitions 16 on page 51).* This is the identity assigned to all sequence-factors in that set and can be used to uniquely distinguish one motif set from another. **The behaviour and computational properties of a sequence factor is defined by its inclusion in a motif set**.

In this formalism, motif sets are classified into collections of mutually exclusive sets, with each collection describing a particular behavioural trait of its members (see Table 1).

The first distinction we make is between motif sets that model behavioural properties **within** a sequence and those that model behavioural properties **between** different sequences. We call the prior class of sets *homointeractive motif sets* and the latter class of sets *heterointeractive motif sets.*

Motif-Set Type		Description
Homointeractive	$\mathtt{M}_{[o]}$	These are Motif sets that describe the interaction of residues **within a region** of a sequence.
Heterointeractive†	$\mathtt{M}_{[e]}$	These are motif sets that describe the interaction between residues **between regions** of different sequences.
Position Independent†	$\mathtt{M}_{[i]}$	These are motif sets whose input sequence-factors have an instantiated pattern and a un-instantiated location list.
Position Dependent	$\mathtt{M}_{[d]}$	These are motif sets whose input sequence-factors have an instantiated location list.
Hybrid	$\mathtt{M}_{[h]}$	These are motif sets whose input sequence-factors are a mix of sequences-factors with an instantiated pattern and a un-instantiated location list and those with an instantiated location list.
Homogeneous†	$\mathtt{M}_{[s]}$	These are motif sets whose sequence-factors are not internally partitioned i.e. they have identical value indicator lists.
Heterogeneous	$\mathtt{M}_{[n]}$	These are motif sets whose sequence factors are partitioned by the indicator list.
Unbounded†	$\mathtt{M}_{[r]}$	These are motif sets whose sequence factors are free to evolve with a changing pattern i.e. their pattern can acquire gaps during evolution.
Bounded	$\mathtt{M}_{[t]}$	These are motif sets whose sequence factors can only evolve with a constant pattern i.e. their pattern cannot acquire gaps during evolution.
Weakly Aligning†	$\mathtt{M}_{[w]}$	These are heterointeractive motif sets with no hard alignment constraints for the quantification of their contribution to an alignment of sequences induced by their sequence-factors.
Strongly Aligning	$\mathtt{M}_{[p]}$	These are heterointeractive motif sets that require encompassing regions of influence of their sequence factors in an alignment in order to quantify their contribution to an alignment of sequences induced by them.
Constrained Strongly Aligning	$\mathtt{M}_{[b]}$	These are strongly aligning motif sets that have an additional constraint that the induced regions that align do not have any gaps in order to quantify their contribution to an alignment of sequences induced by them.

Table 1: Given in the table are the different types of motif-sets. Motif-sets in each category are mutually exclusive from each other. A '†' indicates the default type of each category.

2.1.4 Homointeractive and Heterointeractive Motif Sets

Homointeractive motif sets describe characteristic features of regions (which can include the whole sequence as a single region) local to the individual sequences. Their behaviour is defined internal to the sequence factor. These can be for example the sequence itself, expressing the property of cohesion between individual residues quantified by the gap opening and extension costs between them. They can be regions with distinct biochemical properties such as charge and hydrophobicity, physical properties such as molecular weight and shape, kinetic properties such as turnover rates, secondary structures such as α-helices and β-strands. They can be regions of GC isochroes, CpG islands, transposons, retrotransposons or any region with some significant characteristic distinguishing it from the rest of the sequence and capable of being quantified in terms of its individual residues (see Figure 8 (a) and (c)). This is expressed in the function $\mathit{Property}_1\_\_\_$ which is defined as follows:

Definition 2.5 ***function*** $\mathit{Property}_1\_\_\_$ **:** $\mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
\mathit{Property}_1\_\_\_ : \mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f : \mathcal{M};\ n_1, n_2 : \mathbb{N} \bullet \exists R : \mathbb{R} \bullet \\
\qquad \left(\begin{array}{ll} \neg\ (n_1 = n_2) & \wedge \\ n_1 \in \operatorname{ran}(f.3) \wedge n_2 \in \operatorname{ran}(f.3) & \wedge \\ \Im_1(f, n_1, n_2) = R & \end{array} \right) \iff \mathit{Property}_1(f, n_1, n_2) = R
\end{array}
$$

Where the function $\Im_1$ is the kernel function of the motif set. For example, the kernel function for the affine gap model can be defined as follows:

$$
\begin{array}{|l}
\Im_1[\text{affine gap model}]\_\_\_ : \mathcal{M} \times \mathbb{N} \times \mathbb{N} \mathrel{+\!\!\!\!\to} \mathbb{R} \\
\hline
\forall f : \mathcal{M};\ n_1, n_2 : \operatorname{ran}(f.3) \bullet \\
\qquad (\neg\ (n_2 = n_1 + 1) \wedge \Im_1(f, n_1, n_2) = 0) \qquad \vee \\
\qquad \exists\, n : \mathbb{N} \mid n_1 = n \wedge n_2 = n + 1 \bullet \\
\qquad\qquad \left(\begin{array}{ll} f.2(n) \in \Phi & \wedge \\ f.2(n+1) = \Delta & \end{array} \right) \implies \Im_1(f, n, n+1) = \text{gap opening cost} \quad \wedge \\
\qquad\qquad \left(\begin{array}{ll} f.2(n) = \Delta & \wedge \\ f.2(n+1) = \Delta & \end{array} \right) \implies \Im_1(f, n, n+1) = \text{gap extension cost} \quad \wedge \\
\qquad\qquad \left(\begin{array}{ll} f.2(n) = \Delta & \wedge \\ f.2(n+1) \in \Phi & \end{array} \right) \implies \Im_1(f, n, n+1) = \text{gap closing cost} \quad \wedge \\
\qquad\qquad \left(\begin{array}{ll} f.2(n) \in \Phi & \wedge \\ f.2(n+1) \in \Phi & \end{array} \right) \implies \Im_1(f, n, n+1) = 0
\end{array}
$$

Heterointeractive motif sets model the behaviour between sequence-factors on different induced sequences i.e. sequence-factors having different target sequences. The behaviour of such sequence-factors is quantified by the function $\mathit{Property}_2\_\_\_\_$ which is defined as follows:

PLEASE TURN OVER

Definition 2.6 ***function*** *Property*$_2$ ____ **:** $\mathcal{M} \times \mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$\begin{array}{|l}
Property_2 ____ : \mathcal{M} \times \mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f_1, f_2 : \mathcal{M};\ n_1, n_2 : \mathbb{N} \bullet \exists R : \mathbb{R} \bullet \\
\quad \left(\begin{array}{ll} \neg\ (\ idm(f_1.1) = idm(f_2.1)\ ) & \wedge \\ n_1 \in \operatorname{ran}(f_1.3) \wedge n_2 \in \operatorname{ran}(f_2.3) & \wedge \\ \Im_2(f_1, f_2, n_1, n_2) = R & \end{array} \right) \iff Property_2(f_1, f_2, n_1, n_2) = R
\end{array}$$

Where the function $\Im_2$ is the kernel function of the motif set. For example, this function can be defining the substitution scores (BLOSUM, PAM matrices) between individual residues in the regions of influence of the sequence-factors as shown bellow for the $BLOSUM60$ matrix.

$$\begin{array}{|l}
\Im_2[BLOSUM60] ____ : \mathcal{M} \times \mathcal{M} \times \mathbb{N} \times \mathbb{N} \twoheadrightarrow \mathbb{R} \\
\hline
\forall f_1, f_2 : \mathcal{M};\ n_1 : \operatorname{ran}(f_1.3);\ n_2 : \operatorname{ran}(f_2.3) \bullet \\
\quad \left(\begin{array}{l} f_1.2(n_1) \in \Phi \wedge \\ f_2.2(n_2) \in \Phi \end{array} \right) \Longrightarrow \begin{array}{l} \Im_2(f_1, f_2, n_1, n_2) = \\ \qquad BLOSUM60(\,f_1.2(n_1),\ f_2.2(n_2)\,) \end{array} \quad \wedge \\
\quad \left(\begin{array}{l} f_1.2(n_1) = \Delta \wedge \\ f_2.2(n_2) = \Phi \end{array} \right) \Longrightarrow \begin{array}{l} \Im_2(f_1, f_2, n_1, n_2) = \\ \qquad \text{Score for aligning with a gap} \end{array} \quad \wedge \\
\quad \left(\begin{array}{l} f_1.2(n_1) = \Phi \wedge \\ f_2.2(n_2) = \Delta \end{array} \right) \Longrightarrow \begin{array}{l} \Im_2(f_1, f_2, n_1, n_2) = \\ \qquad \text{Score for aligning with a gap} \end{array} \quad \wedge \\
\quad \left(\begin{array}{l} f_1.2(n_1) = \Delta \wedge \\ f_2.2(n_2) = \Delta \end{array} \right) \Longrightarrow \begin{array}{l} \Im_2(f, n, n+1) = \\ \qquad \text{Score for aligning two gaps} \end{array}
\end{array}$$

Heterointeractive motif sets can model the alignment of regions on sequences with some special characteristic, distinct to but in conjunction with the rest of the alignment. For example, we might wish to give special alignment consideration to regions identified (if possible) as coming from a family of transcription factor binding sites or regions of hypersensitive sites, which are regions that are biologically distinguishable from the rest of the sequence and would favour aligning among themselves. Designating such regions using motif sets enables for example, distinct residue interaction matrices (through the kernel function of the motif set) to be specified for them. Likewise, heterointeractive motif sets can model known homologous regions (see Example 2.3) or a collection of regions within the sequences defined by some common feature such as codons (for coding regions of DNA sequences) or regions of *CpG* islands. They can model alignment information between different sequences obtained from initial alignments or other alignment programs (see Figure 8 (b) and (d)).

Using the *Property* functions, we can now formally define the two classes of motif sets described above.

Definition 2.7 ***Homointeractive Motif Set*** $\mathbb{M}_{[o]}$ **:** $\mathbb{P}\,\mathbb{M}$
Homointeractive motif sets are defined as:

$$\mathbb{M}_{[o]} == \{ M : \mathbb{M} \mid \forall f : M \bullet \begin{array}{ll} (\exists\, r \in \mathbb{R};\ n_1, n_2 \in \mathbb{N}^+ \bullet Property_1(f, n_1, n_2) = r\,) & \wedge \\ (\forall\, f' \in M;\ n_1, n_2 \in \mathbb{N}^+ \bullet Property_2(f, f', n_1, n_2) = \odot\,) & \end{array} \}$$

.

For a description of the meta character $\odot$ refer to notations and basic definitions 10 on page 50. The above definition states that the set of homointeractive motif sets are a sub-set of the universal motif set collection. Sequence factors of these sets define the interaction properties between individual residues on a sequence given by the $Property_1$ function. These sequence factors do not define the interaction properties between residues of different sequence factors. This is indicated by the fact that the $Property_2$ function always returns the dead value.

Definition 2.8 ***Heterointeractive Motif Sets*** $\mathbb{M}_{[e]}$ ***:*** $\mathbb{P}\,\mathbb{M}$
Heterointeractive motif sets are defined as:

$$\begin{aligned}\mathbb{M}_{[e]} == \{M : \mathbb{M} \mid\ & (\forall f_1, f_2 \in M \mid \neg\ (idm(f_1.1) = idm(f_2.1)) \bullet \\ & \qquad \exists\, r \in \mathbb{R};\ n_1, n_2 \in \mathbb{N}^+ \bullet \\ & \qquad\qquad Property_2(f_1, f_2, n_1, n_2) = r\,) \quad \wedge \\ & (\forall f \in M;\ n_1, n_2 \in \mathbb{N}^+ \bullet\ Property_1(f, n_1, n_2) = \odot\,) \qquad \}\end{aligned}$$

.

The above definition states that the set of heterointeractive motif sets are a sub-set of the universal motif set collection. Sequence factors of these sets define the interaction properties of individual residues between sequences given by the $Property_2$ function. They do not define the interaction properties between residues of the same sequence. This is indicated by the fact that the $Property_1$ function always returns the dead value.

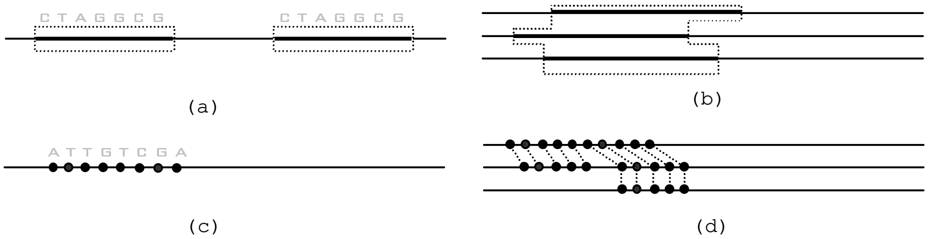

Figure 8: The different representations of information. (a) Position independent homointeractive sequence-factors. (b) Position independent heterointeractive sequence-factors. (c) Position dependent homointeractive sequence-factors. (d) Position dependent heterointeractive sequence-factors.

2.1.5 Position Independent, Position Dependent and Hybrid Motif Sets

The second distinction on motif sets that we make relates to the form in which the initial or input sequence-factors are specified. This is a distinction made on the input specification of the motif sets. Initial sequence factors can be specified by their pattern independent of the location they appear on the target sequence, or they can be specified by the position they appear on the target sequence, given by the location list. If all the initial sequence-factors in a motif set are specified only by their pattern leaving the location list uninstantiated, such a motif set is called a *position independent* motif set. Formally, a position independent motif set is a motif set whose initial elements are specified as follows:

Definition 2.9 ***Position Independent Motif Sets*** $\mathbb{M}_{[i]}$ **:** $\mathbb{P}\,\mathbb{M}$
$\mathbb{M}_{[i]} == \{M : \mathbb{M} \mid \forall f : M \bullet (\mu\, p : \mathcal{P} \bullet f.2 = p) \wedge (f.3 = \circledast)\}$.

The definite description μ and the meta character $\circledast$ are described in *notations and basic definitions 12 on page 51* and *notations and basic definitions 9 on page 50.* What this definition states is that if an initial motif set is specified as a position independent motif set, then the pattern element of all its sequence-factors must be instantiated with a definite value while their location list must be uninstantiated. It should be noted that this is only a requirement on the initial or input motif set as specified by the user.

If all the initial sequence-factors in a motif set are specified using the location list irrespective of the pattern being instantiated or not, such a motif set is called a *position dependent* motif set. Formally, a position dependent motif set is a motif set whose initial elements are specified as follows:

Definition 2.10 ***Position Dependent Motif Sets*** $\mathbb{M}_{[d]}$ **:** $\mathbb{P}\,\mathbb{M}$
$\mathbb{M}_{[d]} == \{M : \mathbb{M} \mid \forall f : M \bullet (\mu\, l : \operatorname{seq} \mathbb{N}^{+} \bullet f.3 = l)\}$.

What this definition states is that if an initial motif set is specified as a position dependent motif set, then the location list of all its sequence-factors must be instantiated to a definite value. Note that there are no requirements on how the pattern is declared in such motif sets. This too is only a reflection on the initial motif set as entered by the user. If a motif set has a mix of sequence factors, some specified only by their pattern and others by their location list, such a motif set is referred to as a *hybrid* motif set. Formally, a hybrid motif set is a motif set whose initial elements are specified as follows:

Definition 2.11 ***Hybrid Motif Sets*** $\mathbb{M}_{[h]}$ **:** $\mathbb{P}\,\mathbb{M}$

$$\begin{array}{llclr}
\mathbb{M}_{[h]} == \{M : \mathbb{M} \mid & (\exists f_1, f_2 : M & \bullet & (\mu\, p : \mathcal{P} \bullet f_1.2 = p \wedge f_1.3 = \circledast) \;\wedge & \\
& & & (\mu\, l : \operatorname{seq} \mathbb{N}^{+} \bullet f_2.3 = l) &) \;\wedge \\
& (\forall f : M & \bullet & (\mu\, p : \mathcal{P} \bullet f.2 = p) \;\vee & \\
& & & (\mu\, l : \operatorname{seq} \mathbb{N}^{+} \bullet f.3 = l) &) \quad \}
\end{array}$$

.

Position independent motif sets specify regions of a target sequence only by its pattern. It is possible for the same pattern to occur at several different locations in the sequence or in different sequences if the target sequence is not instantiated (see Figure 8 (a) and (b)). When an initial sequence-factor is specified only by its pattern, it will create copies of itself at all matches of that pattern on the sequence and instantiate the location list to match the pattern on the target sequence. If the target sequence is left uninstantiated, this pattern matching will be performed on all initial sequences. Information obtained on many of the sites described above fall into this category. For example GC isochroes could be found in many parts of a sequence and in many different sequences, transcription factor binding sites can exist in different parts of the sequences. By restricting the initial instantiated input of the sequence-factors to their pattern, specified directly or as a regular expression, one can leave the induction process (that is where a sequence factor is matched with a target sequence) to instantiate sequence-factors for all occurrences of the pattern along the target sequence or sequences.

Position dependent motif sets on the other hand specify a region in a sequence not by the pattern of its residues but by the exact positions within the sequence. The pattern is implicit rather than explicit in this information (see Figure 8 (c) and (d)). Alignment information of the sequences, obtained from initial alignments or secondary alignment sources (as in T-Coffee) is an example of position dependent information. In these cases the regions of residue interactions in sequence factors are specified explicitly rather than through a pattern instantiation process. The kernel function of position dependent motif sets can be defined explicitly over the individual positions of the region encapsulated by the sequence factor.

Example 2.12 For example, given two position dependent sequence-factors f_1 and f_2 with location lists l_1 and l_2, you can define a kernel function of the form:

$$\Im_2(f_1, f_2, i, j) = \begin{array}{c c} i/j & \begin{array}{cccc} l_1(1) & l_1(2) & .. & l_1(\#l_1) \end{array} \\ \begin{array}{c} l_2(1) \\ l_2(2) \\ \cdots \\ l_2(\#l_2) \end{array} & \begin{pmatrix} r_{1,1} & r_{1,2} & .. & r_{1,l_1} \\ r_{2,1} & r_{2,2} & .. & r_{2,l_1} \\ \cdots & \cdots & .. & \cdots \\ r_{l_2,1} & r_{l_2,2} & .. & r_{l_2,l_1} \end{pmatrix} \end{array}$$

The above function is defined on the location lists of the two sequence factors. It explicitly specifies the degree of interaction $r_{i,j}$ between any two positions on the location list. □

2.1.6 Homogeneous and Heterogeneous Motif Sets

The third distinction we make is between *homogeneous* and *heterogeneous* motif sets described bellow. The location list of a sequence-factor specifies its region of influence on its target sequence. Motif sets in which the interaction of residues specified by their sequence factors are not constrained by regions within their pattern are called *homogeneous* motif sets. A homogeneous motif set is defined as follows:

Definition 2.13 ***Homogeneous Motif Set*** $\mathbb{M}_{[s]}$ ***:*** $\mathbb{P}\,\mathbb{M}$

$$\begin{array}{|l} \mathbb{M}_{[s]} : \mathbb{M} \\ \hline \forall\, M : \mathbb{M} \bullet \\ \qquad M \in \mathbb{M}_{[s]} \implies \left(\begin{array}{l} \forall\, f_1, f_2 : M \bullet \\ \qquad \forall\, i_1 : f_1.4;\ i_2 : f_2.4 \bullet i_1 = i_2) \end{array} \right) \end{array}$$

This definition states that a homogeneous motif set is a motif set whose sequence-factors are **not** partitioned internally ($f_1 = f_2$) or externally ($f_1 \neq f_2$) via the indicator list. Any residue specified by a sequence factor in a homogeneous motif set has the freedom to interact with any other residue in that sequence factor (for homointeractive motif sets) or with any residue specified by another sequence factor (for heterointeractive motif sets) of the same set.

Homogeneous motif sets are good to represent sequence-factors defining regions with global interactions. For example, the global substitution scores of a set of sequences can be represented by a heterointeractive homogeneous motif set. Each sequence in this case will be represented by a sequence-factor in which the location-list indexes the entire length of its target sequence (i.e. the pattern is equal to the target sequence). The kernel function of the set will define the global substitution scores between residues (e.g. PAMx scores). The global gap penalties can be specified by a homointeractive homogenious motif set in which the location list of each sequence-factor indexes the entire length of its target sequence and the kernel function of the set defines the gap model (e.g. an affine gap model). Together, the above two motif sets form the basis for a standard alignment.

In some instances, it may be necessary to explicitly distinguish the locations of the region of influence on the target sequence by a sequence factor. This is achieved via the indicator list. A variable indicator list explicitly distinguishes between different positions in the region specified by the location list. Such demarcations can be used to specify, selective interaction properties between residues of the same sequence factor (homointeractive) or different sequence factors (heterointeractive). Motif sets in which the region of influence of its sequence factors are explicitly demarcated using the indicator list are called *heterogeneous* motif sets and are defined as follows:

Definition 2.14 ***Heterogeneous Motif Set*** $\mathbb{M}_{[n]}$ **:** $\mathbb{P}\,\mathbb{M}$

$$
\begin{array}{|l}
\mathbb{M}_{[n]} : \mathbb{M} \\
\hline
\forall\, M : \mathbb{M} \bullet \\
\qquad M \in \mathbb{M}_{[n]} \implies \left(\begin{array}{l} \exists\, f_1, f_2 : M \bullet \\ \qquad \exists\, i_1 : f_1.4;\ i_2 : f_2.4 \bullet \neg\, (\, i_1 = i_2\,) \end{array} \right)
\end{array}
$$

This definition states that a heterogeneous motif set is a motif set in which its sequence-factors are partitioned internally (i.e. $f_1 = f_2$ in the first predicate of the implication) or externally (i.e. $f_1 \neq f_2$ in the first predicate of the implication) via the indicator list. Residues specified by sequence-factors of heterogeneous motif sets have the property that they can selectively interact with a specified set of one or more residues within the sequence factor (for homointeractive motif sets) or with a specified set of residues of any other sequence-factor (for heterointeractive motif sets) of that set (see Figure 9). The regions of interaction are governed by the indicator list. Sequence-factors encoding homologous patterns as those described in Figure 7 where their alignments have been determined experimentally and residue interactions confirmed are best expressed in heterogeneous motif sets with each corresponding location having a unique indicator. Constraining the alignment of residues to those in locations with like indicators will facilitate the alignment of such regions to conform with that determined experimentally. Codons (for aligning coding regions in DNA sequences) and segmented sequences with different distinct regions (to the degree that they can be identified) are some other candidates for

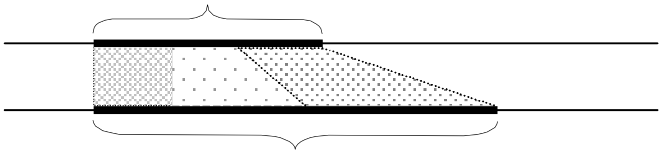

Figure 9: The figure shows a region in the two sequences induced by a heterogeneous motif set. The highlighted areas indicate the scope of a valid interaction between the two sequence-factors t_1 and t_2 of the motif set s.

this type of motif set. Heterogeneous motif sets also provide a powerful framework to formulate innovative alignment strategies. *Delphian* motif sets are a good example of this. A Delphian motif set is a heterointeractive, heterogeneous motif set used to encode future interaction information to make current interaction decisions (see Chapter **??**).

2.1.7 Unbounded and Bounded Motif sets

Any two motif sets are said to be *compatible* if the behaviour and computational properties of the sequence factors of the union of the two sets is defined. This is expressed by the *compatibility relation* $\_ \leftrightsquigarrow \_$ which is an equivalence relation defined as follows:

Definition 2.15 ***relation*** $\_ \leftrightsquigarrow \_ : \mathbb{M} \leftrightarrow \mathbb{M}$

$$
\begin{array}{|l}
\_ \leftrightsquigarrow \_ : \mathbb{M} \leftrightarrow \mathbb{M} \\
\hline
\forall M : \mathbb{M} \quad \bullet \quad M \leftrightsquigarrow M \quad \wedge \\
\forall M_1, M_2 : \mathbb{M} \quad \bullet \quad M_1 \leftrightsquigarrow M_2 \iff M_2 \leftrightsquigarrow M_1 \quad \wedge \\
\forall M_1, M_2, M_3 : \mathbb{M} \quad \bullet \quad M_1 \leftrightsquigarrow M_2 \wedge M_2 \leftrightsquigarrow M_3 \implies M_1 \leftrightsquigarrow M_3 \quad \wedge \\
\forall M_1, M_2 : \mathbb{M} \quad \bullet \\
\quad M_1 \leftrightsquigarrow M_2 \implies \\
\quad \left(\begin{array}{l}
\left(\begin{array}{ll}
\forall f : M_1 \cup M_2;\ n_1, n_2 : \mathbb{N} \mid & \neg\ (n_1 = n_2) \quad \wedge \\
 & n_1 \in \operatorname{ran}(f.3) \quad \wedge \\
 & n_2 \in \operatorname{ran}(f.3) \quad \bullet \\
 & \quad \mathit{Property}_1(f, n_1, n_2) = *
\end{array} \right) \quad \vee \\
\left(\begin{array}{ll}
\forall f_1, f_2 : M_1 \cup M_2;\ n_1, n_2 : \mathbb{N} \mid & \neg\ (\mathit{idm}(f_1.1) = \mathit{idm}(f_2.1)) \quad \wedge \\
 & n_1 \in \operatorname{ran}(f_1.3) \quad \wedge \\
 & n_2 \in \operatorname{ran}(f_2.3) \quad \bullet \\
 & \quad \mathit{Property}_2(f_1, f_2, n_1, n_2) = *
\end{array} \right)
\end{array} \right)
\end{array}
$$

For any two motif sets M_1, M_2, the relation $M_1 \leftrightsquigarrow M_2$ means that the motif set M_1 is compatible with the motif set M_2. When two motif sets are compatible with each other, their sequence factors can be merged to form a single set, which is the union of the two. For the compatibility relation to hold between two motif sets, additional kernel functions quantifying the behaviour between the two sets of sequence-factors in the enlarged motif set may need to be defined.

Sequence factors are not static objects. They can *evolve* into new sequence-factors subject to constraints specified by their motif set membership. The evolution of a sequence-factor takes it from one state to another. This process can lead to structural changes in its target sequence s and as a result changes in its pattern p, location-list l and indicator-list t. A change in the structure of a sequence-factor is **only** attributed to the **insertion** of one or more gaps in its target sequence. For example, a sequence-factor can evolve as a result of it participating in an alignment (as a part of an *induced* sequence). The evolution of a sequence-factor need not change its structure if no gaps are introduces in its target sequence. It is possible that the result of an evolutionary process on a sequence factor (f) leads to its death ($f = \odot$). This occurs as a result of the sequence-factor infringing constraint set out by its motif set.

We introduce the term *motif set extension* to describe the motif set with the same properties as a given motif set and comprising of all and only those sequence-factors that are a result of a valid evolutionary process on the sequence factors of that set. A motif set will always be a subset known as the *base set* of its motif set extension. The two sets will always be compatible. We define two functions $\mathit{MotifSetExtension}_1$ and $\mathit{MotifSetExtension}_2$ both describing a particular type of motif set extension, but first, we define the function $\mathit{GapsInserted}\_\_\_$ which returns the number of gaps introduced from the origin to a given point of a sequence as a result of transforming it into a new sequence with the insertion of gaps. The *function* $\mathit{GapsInserted}\_\_\_$ is defined as follows:

Definition 2.16 ***function*** $\mathit{GapsInserted}\_\_\_$ ***:*** $\mathcal{P} \times \mathcal{P} \times \mathbb{N} \longrightarrow \mathbb{Z} \cup \{\odot\}$

$$
\begin{array}{|l}
\mathit{GapsInserted}\_\_\_ : \mathcal{P} \times \mathcal{P} \times \mathbb{N} \longrightarrow \mathbb{Z} \cup \{\odot\} \\
\hline
\forall\, s_1, s_2 : \mathcal{P};\ n : \mathbb{N}^{+} \bullet \\
\qquad \left(\begin{array}{ll} s_1 \mathrel{\underline{\subset\!\!\!{\scriptstyle +}}} s_2 & \wedge \\ s_1 \upharpoonright \Phi = s_2 \upharpoonright \Phi & \wedge \\ n \in \{1 \,..\, \# s_1\} & \end{array} \right) \iff \\
\qquad\qquad \left(\begin{array}{lr} \exists\, \alpha_{1_1}, \alpha_{1_2}, \alpha_{2_1}, \alpha_{2_2} : \mathcal{P} \mid & \\ \qquad \alpha_{1_1} \mathrel{\underline{\subset\!\!\!{\scriptstyle +}}} \alpha_{2_1} & \wedge \\ \qquad \alpha_{1_2} \mathrel{\underline{\subset\!\!\!{\scriptstyle +}}} \alpha_{2_2} & \wedge \\ \qquad s_1 = \alpha_{1_1} \frown s_1(i) \frown \alpha_{1_2} & \wedge \\ \qquad s_2 = \alpha_{2_1} \frown s_1(i) \frown \alpha_{2_2} & \bullet \\ \qquad\qquad \mathit{GapsInserted}(s_1, s_2, i) = & \\ \qquad\qquad\qquad \#(\alpha_{2_1} \upharpoonright \{\Delta\}) - \#(\alpha_{1_1} \upharpoonright \{\Delta\}) & \end{array} \right)
\end{array}
$$

The operator $\_ \frown \_$ is the sequence concatenation operator. The operator $\_ \mathrel{\underline{\subset\!\!\!{\scriptstyle +}}} \_$ is the sub-sequence operator and is described in *notations and basic definitions 13 on page 51.* Figure 10 shows an example application of the function *GapsInserted* applied to a sequence S_1 and its extension S_2.

The *unbounded motif set extension* function $\mathit{MotifSetExtension}_1\,\_$ takes a motif set and returns its unrestricted motif set extension. This set consists of all the sequence-factors

```
     123456789012345678901234
S1 = 'WAPVYSDYETSGVDIL'
S2 = 'WAPΔΔΔVYSDYΔETSGVΔΔΔΔDIL'

     i       =  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
gi(s1,s2,i) =   0 0 0 3 3 3 3 3 4 4  4  4  4  8  8  8
```

Figure 10: Application of the function $GapsInserted\_\_\_$.

that are a result of an evolutionary process of the sequence factors of the original set, applied on either or both of their target sequence **and** pattern without any constraints on the introduction of gaps on either sequence. Formally, the unbounded motif set extension function $MotifSetExtension_1$ is defined as follows:

Definition 2.17 ***function*** $MotifSetExtension_1\_$ **:** $\mathbb{M} \longrightarrow \mathbb{M}$

$$
\begin{array}{|l}
MotifSetExtension_1\_ : \mathbb{M} \longrightarrow \mathbb{M} \\
\hline
\forall M : \mathbb{M} \bullet \\
\quad MotifSetExtension_1(M) = \{ f_2 : \mathcal{M} \mid \\
\quad\quad \exists f_1 : M \bullet \\
\quad\quad\quad f_1.1 \Subset f_2.1 \hfill \wedge \\
\quad\quad\quad f_1.1 \upharpoonright \Phi = f_2.1 \upharpoonright \Phi \hfill \wedge \\
\quad\quad\quad f_1.2 \Subset f_2.2 \hfill \wedge \\
\quad\quad\quad \forall i : \{1 \mathinner{..} \#f_1.4\} \bullet f_2.4.(i + GapsInserted(f_1.2, f_2.2, i)) = f_1.4(i) \quad \wedge \\
\quad\quad\quad f_2.3 = \langle f_1.3.1 + GapsInserted(f_1.1, f_2.1, f_1.3.1) \mathinner{..} f_1.3.\#f_1.3 + \\
\quad\quad\quad\quad\quad GapsInserted(f_1.1, f_2.1, f_1.3.\#f_1.3) \rangle \quad \}
\end{array}
$$

The *bounded motif set extension* function $MotifSetExtension_2\_$ takes a motif set and returns its restricted motif set extension. This set consists of all the sequence-factors that are a result of an evolutionary process on the sequence factors of the original set with the restriction that their pattern stays unaltered. There are no restrictions on the insertion of gaps in the target sequence except for the region defined by the location list representing the pattern. The pattern itself is constrained from change by the introduction of gaps. Formally, the bounded motif set extension function $MotifSetExtension_2$ is defined as follows:

PLEASE TURN OVER

Definition 2.18 ***function*** *MotifSetExtension*$_2$ _ **:** $\mathbb{M} \longrightarrow \mathbb{M}$

$$
\begin{array}{|l}
\mathit{MotifSetExtension}_2\,\_ : \mathbb{M} \longrightarrow \mathbb{M} \\
\hline
\forall\, M : \mathbb{M} \bullet \\
\quad \mathit{MotifSetExtension}_2(M) = \{ f_2 : \mathcal{M} \mid \\
\qquad \exists\, f_1 : M \bullet \\
\qquad\quad f_1.1 \mathrel{\underline{\Subset}}_{+} f_2.1 \hfill \wedge \\
\qquad\quad f_1.1 \upharpoonright \Phi = f_2.1 \upharpoonright \Phi \hfill \wedge \\
\qquad\quad f_1.2 = f_2.2 \hfill \wedge \\
\qquad\quad f_2.4 = f_1.4 \hfill \wedge \\
\qquad\quad f_2.3 = \langle f_1.3.1 + \mathit{GapsInserted}(f_1.1, f_2.1, f_1.3.1) \mathrel{..} f_1.3.\#f_1.3 + \\
\qquad\qquad\qquad\qquad \mathit{GapsInserted}(f_1.1, f_2.1, f_1.3.\#f_1.3) \rangle \quad \}
\end{array}
$$

When an extension is created for a sequence-factor, the location list of the original sequence factor is transformed to take into account the displacement of the pattern from its original location in the initial target sequence to its new location in the extended target sequence. The transformation of the location list of an unbounded extension may introduce new indices to take into account any gaps added to the pattern. On the other hand, the transformation of the location list of a bounded extension will only involve a fixed displacement of its indices corresponding to any gaps added to the target sequence before the point of the pattern. An unbounded extension may expand the indicator list to correspond with any expansion of the pattern. The resultant indicator list will have the same indicators as the initial sequence factor but shifted relative to the shift in residues of the pattern. New indicators will be added at positions corresponding to inserted gaps in the extended pattern. Their value will be decided by the properties (e.g. homogeneous or heterogeneous and if heterogeneous, how the indicator list is distributed etc.) of the motif set. Figure 11 shows an example of a sequence factor and one of its unbounded extensions. There will be no change to the indicator list of a sequence factor as a result of a bounded extension.

```
f  = (... 'acgtact',⟨ 11, 12, 13, 14, 15, 16, 17 ⟩,⟨ 1, 1, 1, 2,2,3,3 ⟩ ,...)

f' = (... 'ac∆∆gta∆ct ',⟨ 31, 32, 33, 34, 35, 36,37,38,39,40 ⟩,⟨ 1,1,*,*,1,2,2,*,3,3 ⟩,...)
```

Figure 11: The figure shows a sequence factor f and one of its unbounded extensions f'. The start of the location list has shifted from 11 in f to 31 in f' due to gaps inserted in the extended target sequence (not shown) up to the point of the pattern. The three gaps added to the extended pattern are at locations $33, 34$ and 38. The locations corresponding to these gaps in the indicator list of f' are denoted by a $'*'$ to mean that their instantiation is governed by the properties of the motif set. The indicators of the rest of the positions are shifted relative to their new positions in the extended pattern but unchanged in value.

This brings us to our fourth distinction of motif sets. We can associate every motif set with either its unbounded motif set extension or its bounded motif set extension. The two classes of motif sets, *unbounded motif sets* and *bounded motif sets* are characterised based on this association. Formally, an unbounded motif set is defined as follows:

Definition 2.19 ***Unbounded Motif Sets*** $\mathbb{M}_{[r]}$ ***:*** $\mathbb{P}\,\mathbb{M}$
$\mathbb{M}_{[r]} == \{\, M : \mathbb{M} \mid M \leftrightsquigarrow \mathit{MotifSetExtension}_1(\, M\,)\,\}$.

What this definition states is that every motif set that is an element of $\mathbb{M}_{[r]}$ is compatible with its unbounded motif set extension. In other words, there is a kernel function in the extended set that can quantify all the valid interactions of its sequence factors. In the case of a homointeractive motif set, this will be the interaction between individual residues in the pattern of each sequence-factor defined by the location list and governed by the indicator list. In the case of a heterointeractive motif set, this will be the interaction between residues in the patterns defined by the location list and governed by the indicator list in sequence factors with distinct target sequences. Usually, the kernel function is *fully defined* in the initial set and inherited by the extended set. A 'fully defined' kernel function of a motif set must have provisions to deal with all possible valid interactions of its sequence-factors. This may mean that it should have provisions for quantifying interactions between residues and newly inserted gaps and also the interaction between two gaps.

There are instances in which we would like to prevent the alteration of the pattern of a sequence factor during evolution. The evolution of such sequence factors can be described in a bounded motif set. Formally, A bounded motif set is defined as follows.

Definition 2.20 ***Bounded Motif Sets*** $\mathbb{M}_{[t]}$ ***:*** $\mathbb{P}\,\mathbb{M}$
$\mathbb{M}_{[t]} == \{\, M : \mathbb{M} \mid M \leftrightsquigarrow \mathit{MotifSetExtension}_2(\, M\,) \wedge \neg\,(\, M \leftrightsquigarrow \mathit{MotifSetExtension}_1(M)\,)\,\}$.

What this definition states is that all motif sets in $\mathbb{M}_{[t]}$ are compatible with their bounded motif set extension but **not** with their unbounded motif set extension. In other words, the interaction of sequence factors in a bounded motif set, after an evolution, can only be computed if their patterns are conserved in the process. Bounded motif sets are good to express structures that we would like to consider only as fixed units. For example, sequence segments defining conserved diagonals as used in the alignment algorithm *DiAlign* or codons defined in the alignment of protein coding regions of DNA sequences needs to be considered as inseparable units in order to make sense of their alignment. There may be sequence segments we would like to tightly conserve due to some reason (for example, they may have a well defined structure) during the alignment process in which case they can be included in a bounded motif set. In general though, many regions of genomic sequence will be unbounded and free to inherit gaps as they evolve.

The *polymorphic motif set extension* function $\mathit{MotifSetExtension}\_$ is defined using $\mathbb{M}_{[r]}$ and $\mathbb{M}_{[t]}$ as follows:

PLEASE TURN OVER

Definition 2.21 ***function*** *MotifSetExtension* $\_$ *:* $\mathbb{M} \longrightarrow \mathbb{M} \cup \{\odot\}$

$$
\begin{array}{|l}
\mathit{MotifSetExtension}\,\_ : \mathbb{M} \longrightarrow \mathbb{M} \cup \{\odot\} \\
\hline
\forall\, M : \mathbb{M} \bullet \\
\quad M \in M_{[r]} \quad \Longrightarrow \quad \mathit{MotifSetExtension}(M) = \mathit{MotifSetExtension}_1(M) \quad \wedge \\
\quad M \in M_{[t]} \quad \Longrightarrow \quad \mathit{MotifSetExtension}(M) = \mathit{MotifSetExtension}_2(M) \quad \wedge \\
\quad \neg\,(M \in M_{[r \cup t]}) \quad \Longrightarrow \quad \mathit{MotifSetExtension}(M) = \odot
\end{array}
$$

The set $M_{[r\cup t]}$ refers to the union of the two sets $M_{[r]}$ and $M_{[t]}$ (*see notations and basic definitions 6 on page 49*).

2.1.8 Weakly Aligning, Strongly Aligning and Constrained Strongly Aligning Motif Sets

When considering the alignment of regions of two sequences, in some instances, it is necessary to align particular regions exclusively with each other in order to gain any inference value from their alignment. For example, when aligning nucleotide codons in protein coding regions or sequence segments as described in *DiAlign* or blocks as used in the Match-Box server, the alignment of such segments (i.e. a pair of codons or two sequence segments to form a diagonal or a block of conserved amino acids) apart from the regions of alignment having to be bounded must also align totally to form a valid alignment (see Figure 12). Given two aligned sequence factors, we can quantify their

```
...eagakafnqcqtchvivddsgt...
...akgekefnkcktchsiiapdgt...
```

(a)

```
...eagakafΔΔnqcqtchvivddsgt...
...akΔΔgekefnkcktchsiiapdgt...
```

(b)

```
      ...eagakafnqcqtchvivddsgt...
...akgΔΔΔekefnkcktchsiiapdgt...
```

(c)

```
  ...eagakafnqcΔΔΔΔqtchvivddsgt...
...akgΔΔekefnkcktchsiiapdgt...
```

(d)

Figure 12: The figure shows four alignments of the two sequence segments s_1 = *'eagakafnqcqtchvivddsgt'* and s_2 = *'akgekefnkcktchsiiapdgt'* defined in the motif set $M_{[t\cap p]}$ i.e. a bounded and strongly aligned motif set. The adjacent table shows the conformation of the sequence segments and alignments with the characteristic properties of each of the motif sets $M_{[t]}$, $M_{[p]}$ and $M_{[t\cap p]}$.

		(a)	(b)	(c)	(d)
$M_{[t]}$	s_1	✓	✗	✓	✗
	s_2	✓	✗	✗	✗
$M_{[p]}$		✓	✓	✗	✗
$M_{[t\cap p]}$		✓	✗	✗	✗

alignment using one of the three functions *weak alignment of sequence factors score function*, *AlignmentFactorScore*$_1$, *strong alignment of sequence factors score function*, *AlignmentFactorScore*$_2$ or *constrained strong alignment of sequence factors score function*, *AlignmentFactorScore*$_3$ based on this requirement. To qualify as a pair of aligned sequence-factors, they must at least satisfy the two conditions that they both have different target sequences and that both target sequences are of the same length. Formally, the weak alignment of sequence factors score function $AlignmentFactorScore_1$ _ _ is defined as follows:

Definition 2.22 ***function*** $AlignmentFactorScore_1\,\_\,\_$ **:** $\mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AlignmentFactorScore_1\,\_\,\_ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f_1, f_2 : \mathbb{P}\,\mathbb{f} \bullet \\
\qquad \left(\begin{array}{ll} idm^{\sim}[\mathbb{M}](f_1.5) \leftrightsquigarrow idm^{\sim}[\mathbb{M}](f_2.5) & \wedge \\ \#f_1.1 = \#f_2.1 & \wedge \\ \neg\,(\,idm(f_1.1) = idm(f_2.1)\,) & \wedge \\ \neg\,(\,\mathrm{ran}(f_1.3) \cap \mathrm{ran}(f_2.3) \;=\; \varnothing\,) & \end{array} \right) \Longleftrightarrow \\
\qquad\qquad\qquad\qquad \left(\begin{array}{ll} \exists R : \mathbb{R} \bullet & \\ \quad \exists B : \mathrm{seq}\,\mathbb{N}^+ \bullet & \\ \qquad B = \mathrm{ran}(f_1.3) \cap \mathrm{ran}(f_2.3) & \wedge \\ \quad R = \sum\limits_{n \in B} Property_2(f_1, f_2, n, n) & \wedge \\ \quad AlignmentFactorScore_1(f_1, f_2) = R & \end{array} \right)
\end{array}
$$

.

The function $AlignmentFactorScore_1$ returns a value equal to the sum of all valid interaction scores taken over the aligned residues of the two sequence factors if and only if the two sequence factors come from compatible motif sets and they qualify to be an alignment and there is at least one aligned interaction between residues of the two sequence factors as indicated by their location lists i.e. $\neg\,(\mathrm{ran}(f_1.3) \cap \mathrm{ran}(f_2.3) \;=\; \varnothing)$. In all other cases, the function $AlignmentFactorScore_1$ returns the dead character $\odot$. The strong alignment of sequence factors score function $AlignmentFactorScore_2$ _ _ is defined as follows:

Definition 2.23 ***function*** $AlignmentFactorScore_2\,\_\,\_$ **:** $\mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AlignmentFactorScore_2\,\_\,\_ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f_1, f_2 : \mathbb{P}\,\mathbb{f} \bullet \\
\qquad \left(\begin{array}{ll} idm^{\sim}[\mathbb{M}](f_1.5) \leftrightsquigarrow idm^{\sim}[\mathbb{M}](f_2.5) & \wedge \\ \#f_1.1 = \#f_2.1 & \wedge \\ \neg\,(\,idm(f_1.1) \;=\; idm(f_2.1)\,) & \wedge \\ (f_1.3 \trianglelefteq f_2.3 \vee f_2.3 \trianglelefteq f_1.3) & \end{array} \right) \Longleftrightarrow \\
\qquad\qquad\qquad\qquad \left(\begin{array}{ll} \exists R : \mathbb{R} \bullet & \\ \quad \exists B : \mathrm{seq}\,\mathbb{N}^+ \bullet & \\ \qquad B = \mathrm{ran}(f_1.3) \cap \mathrm{ran}(f_2.3) & \wedge \\ \quad R = \sum\limits_{n \in B} Property_2(f_1, f_2, n, n) & \wedge \\ \quad AlignmentFactorScore_2(f_1, f_2) = R & \end{array} \right)
\end{array}
$$

.

The function *AlignmentFactorScore*$_2$ returns a value equal to the sum taken over the aligned residues of two sequence factors if and only if the two sequence factors come from compatible motif sets, they qualify to be an alignment and the region defined by one sequence factor aligns completely within the region defined by the other sequence factor as indicated by their location lists i.e. $f_1.3 \Subset f_2.3 \vee f_2.3 \Subset f_1.3$. In all other cases, the function *AlignmentFactorScore*$_2$ returns the dead character $\odot$. The constrained strong alignment of sequence factors score function *AlignmentFactorScore*$_3$ __ is defined as follows:

Definition 2.24 ***function*** *AlignmentFactorScore*$_3$ __ **:** $\mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AlignmentFactorScore_3 \_\_ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f_1, f_2 : \mathbb{P}\, \mathbb{f} \bullet \\
\qquad \left(\begin{array}{ll}
idm^{\sim}[\mathbb{M}](f_1.5) \rightsquigarrow idm^{\sim}[\mathbb{M}](f_2.5) & \wedge \\
\# f_1.1 = \# f_2.1 & \wedge \\
\neg\, (\, idm(f_1.1) \;=\; idm(f_2.1)\,) & \wedge \\
\left(\begin{array}{l}
\left(\begin{array}{ll}
f_1.3 \Subset f_2.3 & \wedge \\
f_1.2 \upharpoonright \{\Delta\} = \varnothing & \wedge \\
f_2.2[f_1.3] \upharpoonright \{\Delta\} = \varnothing &
\end{array} \right) \vee \\
\left(\begin{array}{ll}
f_2.3 \Subset f_1.3 & \wedge \\
f_2.2 \upharpoonright \{\Delta\} = \varnothing & \wedge \\
f_1.2[f_2.3] \upharpoonright \{\Delta\} = \varnothing &
\end{array} \right)
\end{array} \right) &
\end{array} \right) \iff \\
\qquad\qquad\qquad\qquad \left(\begin{array}{l}
\exists R : \mathbb{R} \bullet \\
\qquad \exists B : \operatorname{seq} \mathbb{N}^+ \bullet \\
\qquad\qquad B = \operatorname{ran}(f_1.3) \cap \operatorname{ran}(f_2.3) \qquad \wedge \\
\qquad R = \sum_{n \in B} Property_2(f_1, f_2, n, n) \qquad \wedge \\
\qquad AlignmentFactorScore_3(f_1, f_2) = R
\end{array} \right)
\end{array}
$$

.

The function *AlignmentFactorScore*$_3$ returns a value equal to the sum taken over the aligned residues of two sequence factors if and only if the two sequence factors come from compatible motif sets, they qualify to be an alignment and the region defined by one sequence factor aligns completely within the region defined by the other sequence factor as indicated by their location lists and the regions that align of the two sequence-factors is gap free. In all other cases, the function *AlignmentFactorScore*$_3$ returns the dead character $\odot$.

All three functions described above quantify the interaction between two sequence-factors by taking the aggregate over the individual interactions at matching locations in their location list i.e. they return the value of aligning the two regions represented by the two sequence-factors, with each other. The function *AlignmentFactorScore*$_1$ quantifies this value for both total and partial alignments of the two regions while the function *AlignmentFactorScore*$_2$ only returns a value if the two regions are totally aligned. A total alignment of two regions is described as an alignment in which one region is totally encompassed by the other region or they both align totally onto each other as indicated by their location lists i.e. if l_1 and l_2 are the location lists of the two regions, then $\mathrm{ran}(l_1) \cap \mathrm{ran}(l_2) = \mathrm{ran}(l_1)$ or $\mathrm{ran}(l_1) \cap \mathrm{ran}(l_2) = \mathrm{ran}(l_2)$. The function $AlignmentFactorScore_3$ in addition to requiring a total alignment requires that the regions that align i.e. $\mathrm{ran}(l_1) \cap \mathrm{ran}(l_2)$ of the two sequence factors be gap free in order to return a value. We can now make the fifth distinction between motif sets based on the properties of their alignment as described by the three functions $AlignmentFactorScore_1$, $AlignmentFactorScore_2$ and $AlignmentFactorScore_3$. We call these three classes of motif sets, *weakly aligning motif sets*, *strongly aligning motif sets* and *constrained strongly aligning motif sets*. Formally, *weakly aligning* motif sets are defined as follows:

Definition 2.25 ***Weakly Aligning Motif Sets*** $\mathbb{M}_{[w]}$ **:** $\mathbb{P}\,\mathbb{M}$

$$
\begin{array}{l}
\mathbb{M}_{[w]} == \{ \quad M : \mathbb{M} \mid \forall f_1, f_2 : M \mid \neg\ (\,idm(f_1.1) = idm(f_2.1)\,) \bullet \\
\quad (\,(\exists f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_1(f_1', f_2') = *\ ) \quad \wedge \\
\quad (\,(\#f_1.2 \geq 2 \wedge \#f_2.2 \geq 2) \implies \\
\qquad \exists f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_1(f_1', f_2') = * \wedge AlignmentFactorScore_2(f_1', f_2') = \odot\ ) \quad) \ \}
\end{array}
$$

A weakly aligning motif set is a collection of sequence factors that have, both quantifiable total and partial alignments between them. A total alignment is an implicit result rather than an explicit requirement of a weakly aligning motif set unlike a strongly aligning motif set which has an explicit requirement that its sequence factors align totally with each other. Formally, *strongly aligning* motif sets are defined as follows:

Definition 2.26 ***Strongly Aligning Motif Sets*** $\mathbb{M}_{[p]}$ **:** $\mathbb{P}\,\mathbb{M}$

$$
\begin{array}{l}
\mathbb{M}_{[p]} == \{ \quad M : \mathbb{M} \mid \forall f_1, f_2 : M \mid \neg\ (\,idm(f_1.1) = idm(f_2.1)\,) \bullet \\
\quad (\,(\exists f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_2(f_1', f_2') = *\ ) \quad \wedge \\
\quad (\forall f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_1(f_1', f_2') = * \iff AlignmentFactorScore_2(f_1', f_2') = *\ )\ ) \ \}
\end{array}
$$

A strongly aligning motif set is a collection of sequence factors that have quantifiable alignment scores only for total alignments between them. Partial alignments of such sequence factors are not defined. If we add the constraint that the regions that align between a pair of strongly aligning sequence factors be gap free, we get a constrained strongly aligning motif set. Formally, *constrained strongly aligning* motif sets are defined as follows:

Definition 2.27 ***Constrained Strongly Aligning Motif Sets*** $\mathbb{M}_{[b]}$ **:** $\mathbb{P}\,\mathbb{M}$

$$
\begin{array}{l}
\mathbb{M}_{[p]} == \{ \quad M : \mathbb{M} \mid \forall f_1, f_2 : M \mid \neg\ (\,idm(f_1.1) = idm(f_2.1)\,) \bullet \\
\quad (\,(\exists f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_3(f_1', f_2') = *\ ) \quad \wedge \\
\quad (\forall f_1' : MotifSetExtension(\{f_1\});\ f_2' : MotifSetExtension(\{f_2\}) \bullet \\
\qquad AlignmentFactorScore_2(f_1', f_2') = * \iff AlignmentFactorScore_3(f_1', f_2') = *\ )\ ) \ \}
\end{array}
$$

A constrained strongly aligning motif set consists of sequence factors that have quantifiable alignment scores only if their induced regions totally align with each other and the region that so align do not have gaps. This is an added constraint on strongly aligning motif sets. A majority of practical applications use constrained strongly aligning motif sets to define information e.g. diagonals, codons, etc. Processing constrained strongly aligning sequence-factors can be done much more efficiently than sequence-factors that are only strongly aligning but not constrained strongly aligning or those that are bounded and strongly aligning.

We can now define a *polymorphic alignment of factors Score* function $AlignmentFactorScore\_\_$ using the three sets $\mathbb{M}_{[w]}$, $\mathbb{M}_{[p]}$ and $\mathbb{M}_{[b]}$ as follows:

Definition 2.28 ***function*** $AlignmentFactorScore\_\_$ ***:*** $\mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$AlignmentFactorScore\_\_ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$$

$$\left(\begin{array}{l} \forall f_1, f_2 : \mathbb{P}\,\mathbb{f} \mid \neg\ (\,idm(f_1.1) \ = \ idm(f_2.1)\,) \ \wedge \\ \qquad\qquad (idm^{\sim}[\mathbb{M}](f_1.5) \leftrightsquigarrow idm^{\sim}[\mathbb{M}](f_2.5)) \bullet \\ \quad \left(\begin{array}{l} (\ idm^{\sim}[\mathbb{M}](f_1.5) \in \mathbb{M}_{[w]} \ \wedge \ idm^{\sim}[\mathbb{M}](f_2.5) \in \mathbb{M}_{[w]} \ ) \iff \\ \qquad AlignmentFactorScore(f_1, f_2) = AlignmentFactorScore_1(f_1, f_2) \end{array} \right) \wedge \\ \quad \left(\begin{array}{l} (\ idm^{\sim}[\mathbb{M}](f_1.5) \in \mathbb{M}_{[p]} \ \wedge \ idm^{\sim}[\mathbb{M}](f_2.5) \in \mathbb{M}_{[p]} \ ) \iff \\ \qquad AlignmentFactorScore(f_1, f_2) = AlignmentFactorScore_2(f_1, f_2) \end{array} \right) \wedge \\ \quad \left(\begin{array}{l} (\ idm^{\sim}[\mathbb{M}](f_1.5) \in \mathbb{M}_{[b]} \ \wedge \ idm^{\sim}[\mathbb{M}](f_2.5) \in \mathbb{M}_{[b]} \ ) \iff \\ \qquad AlignmentFactorScore(f_1, f_2) = AlignmentFactorScore_3(f_1, f_2) \end{array} \right) \end{array} \right) \wedge$$

$$\left(\begin{array}{l} \forall f_1, f_2 : \mathbb{P}\,\mathbb{f} \bullet \\ \quad \left(\begin{array}{l} \neg\ (\,AlignmentFactorScore(f_1, f_2) \ = \ AlignmentFactorScore_1(f_1, f_2)\,) \ \wedge \\ \neg\ (\,AlignmentFactorScore(f_1, f_2) \ = \ AlignmentFactorScore_2(f_1, f_2)\,) \ \wedge \\ \neg\ (\,AlignmentFactorScore(f_1, f_2) \ = \ AlignmentFactorScore_3(f_1, f_2)\,) \end{array} \right) \\ \qquad\qquad\qquad\qquad \implies AlignmentFactorScore(f_1, f_2) = \odot \end{array} \right)$$

.

2.2 Information Association

In this formalism, a sequence obtains its inference capabilities by associating itself with one or more sequence factors. This is called an association by induction. A sequence so associated is referred to as an *induced sequence*. An induced sequence can be thought of as a string that attaches a set of sequence factors along its length and in doing so, implicitly transforming the behavioural properties of regions along its length according to the behavioural properties inherited from those sequence factors at the point of their region of influence on it. A sequence factor associates itself with a sequence by instantiating its target sequence to the sequence in concern. When this association if fully instantiated i.e. when all the variables of the sequence factor are given appropriate values we say that the sequence factor *induces* information on the sequence at its region of influence.

Given a sequence and a motif set, we can create a *motif set sequence induction list* to represent the subset of all fully instantiated sequence factors of the motif set that induces information on the sequence. Formally, a motif set sequence induction list is defined as follows:

Definition 2.29 ***Motif-Set Sequence Induction List*** $\mathbb{I}$ ***:*** $\mathbb{N} \times (\mathbb{M} \cup \{\odot\})$

$$
\begin{array}{|l}
\mathbb{I} : \mathbb{N} \times \mathbb{M} \cup \{\odot\} \\
\hline
\neg\,(\mathbb{I} = \circledast) \quad \wedge \\
\mathbb{I}.2 = \varnothing \iff \mathbb{I} = \odot \quad \wedge \\
\forall f : \mathbb{I}.2 \bullet f.5 = \mathbb{I}.1 \quad \wedge \\
\forall f_1, f_2 : \mathbb{I}.2 \bullet \quad idm(f_1.1) = idm(f_2.1) \quad \wedge \\
\qquad\qquad f_1.3 = f_2.3 \implies f_1 = f_2
\end{array}
$$

In words, a *motif set sequence induction list* which is not dead ($\neg\,(\mathbb{I} = \odot)$) is a pair (n, F) which holds a subset of sequence factors from a motif set $M \in \mathbb{M}$ where.

- $n \in \mathbb{N} \bullet n = idm(M)$ is the unique identity of the motif set M.
- $F \in \mathbb{M} \bullet F \subseteq M$ is a subset of sequence factors from M.

Subject to the following constraints:

- $\neg\,(\mathbb{I} = \circledast)$ % $\mathbb{I}$ *is fully instantiated (see notations and basic definitions 9 on page 50).* **and**
- $F = \varnothing \implies \mathbb{I} = \odot$ % *If there are no sequence factors in* F*,* $\mathbb{I}$ *is considered 'dead'.* **and**
- $\forall f : F \bullet f.5 = n$ % *All sequence factors should be from the same motif set.* **and**
- $\forall f_1, f_2 \in F \bullet idm(f_1.1) = idm(f_2.1)$ % *They should all operate on the same sequence.* **and**
- $\forall f_1, f_2 \in F \bullet f_1.3 = f_2.3 \implies f_1 = f_2$ % *All sequence factors are different (removes redundancy).*

A motif set sequence induction list describes the overall association between a sequence and a motif set. An *induced sequence* describes the association between a sequence and all motif sets. Formally, an induced sequence is defined as follows:

Definition 2.30 ***Induced Sequence*** $\mathbb{s}$ ***:*** $\mathcal{P} \times \mathbb{P}\,\mathbb{P}\,\mathbb{I}$

$$
\begin{array}{|l}
\mathbb{s} : \mathcal{P} \times \mathbb{P}\,\mathbb{P}\,\mathbb{I} \\
\hline
\neg\,(\mathbb{s} = \circledast) \quad \wedge \\
\forall l_1, l_2 : \mathbb{s}.2 \bullet l_1.1 = l_2.1 \implies l_1 = l_2 \quad \wedge \\
\forall l : \mathbb{s}.2 \bullet \\
\qquad \forall f \in l.2 \bullet idm(f.1) = idm(s)
\end{array}
$$

In words, an *induced sequence* is a pair $\mathbb{s} == (s, L)$ which must always be instantiated to some value $(\neg\ (\mathbb{s} = \circledast))$, where.

- $s \in \mathcal{P}$ is a target sequence.
- $L \in \mathbb{P}\,\mathbb{P}\,\mathbb{l}$ is a list of motif set sequence induction lists.

Subjected to the following constraints:

- $\forall\, l_1, l_2 \in L \bullet l_1.1 = l_2.1 \implies l_1 = l_2$ % *All motif sets contributing to the list of motif set sequence induction lists must be different (removes duplicated processing).* **and**
- $\forall\, l \in L \bullet \forall f \in l.2 \bullet idm(f.1) = idm(s)$ % *All sequence factors must operate on the same sequence i.e. the induced sequence.*

Any initial sequence can be seen as an induced sequence, induced by the *null* sequence-factor. A null sequence factor is a sequence factor whose region of influence is the sequence in concern and whose kernel function is the *null function.* The null function has no overall effect on the *Property*( ) functions. It is defined as follows:

$$
\begin{array}{|l}
\Im_1[null]\_\_\_ : \mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \\
\Im_2[null]\_\_\_\_ : \mathcal{M} \times \mathcal{M} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{R} \\
\hline
\forall f : \mathcal{M};\ n_1, n_2 : \mathrm{ran}(f.3) \bullet \\
\qquad \Im_1(f, n_1, n_2) = \text{null value} \qquad\qquad \text{\% \textit{usually 0}} \\
\forall f_1, f_2 : \mathcal{M};\ n_1 : \mathrm{ran}(f_1.3);\ n_2 : \mathrm{ran}(f_2.3) \bullet \\
\qquad \Im_2(f_1, f_2, n_1, n_2) = \text{null value} \qquad\qquad \text{\% \textit{usually 0}}
\end{array}
$$

We define the function *Induce* __, which describes the induction process between an induced sequence and a motif set. We consider an initial sequence as an induced sequence induced by the null sequence factor, hence every sequence is treated as an induced sequence. The function *Induce* __ takes an induced sequence s and a motif set M and returns a new induced sequence s' which is the induced-sequence s induced by the sequence-factors of M. It is defined as follows:

Definition 2.31 ***function*** *Induce* __ *:* $\mathbb{P}\,\mathbb{s} \times \mathbb{M} \longrightarrow \mathbb{P}\,\mathbb{s}$

$$
\begin{array}{|l}
Induce\,\_\_ : \mathbb{P}\,\mathbb{s} \times \mathbb{M} \longrightarrow \mathbb{P}\,\mathbb{s} \\
\hline
\forall\, s : \mathbb{P}\,\mathbb{s};\ M : \mathbb{M} \bullet \\
\qquad \exists\, s' : \mathbb{P}\,\mathbb{s} \bullet \\
\qquad\qquad Induce(s, M) = s' \iff \\
\qquad\qquad\quad \left(\begin{array}{ll}
s'.1 = s.1 & \wedge \\
\left(\begin{array}{ll}
\exists\, l : s.2 \bullet l.1 = idm(M) \implies & \\
\qquad s'.2 = s.2 \oplus \{(l.1, l.2 \cup \{f : M \mid f.1 = s.1\})\} & \vee \\
\forall\, l : s.2 \bullet \neg\ (l.1 = idm(M)) \implies & \\
\qquad s'.2 = s.2 \cup \{(idm(M), \{f : M \mid f.1 = s.1\})\} &
\end{array} \right) &
\end{array} \right)
\end{array}
$$

.

This definition states that the function $Induce\_\_$ takes as input an induced sequence s and a motif set M and returns an induced sequence s' such that both s and s' have the same target sequence and, the list of motif set sequence induction lists of s' is the list of motif set sequence induction lists of s appended to take into account the additional new sequence factors of M if the motif set M is already represented in this list or concatenated with the new motif set sequence induction list created by M on the target sequence of s.

Table 1 summarises the different categories of motif sets described in the previous section. The types defined in each of these categories are mutually exclusive. A motif set can be defined only by one type from each category. For example, a motif set that is both 'bounded' and 'unbounded' cannot exist, it must either be a bounded motif set or an unbounded motif set but not both. In order to be a valid motif set also known as a *fully defined motif set*, a motif set must be defined in terms of all these categories except for the 'weakly aligning' and 'strongly aligning' types which are only applicable to heterointeractive motif sets. For example, we can define a heterointeractive, position dependent, homogeneous, bounded and strongly aligning motif set $M_{[e \cap d \cap s \cap t \cap p]}$ (*refer to notations and basic definitions 6 on page 49 for more information on the notation used for naming motif sets*) which is a fully defined motif set with the properties of each one of these types. Where the defining type or types of a motif set are not specified for any one or more of these categories, the default types (these are shown in Table 1) are assumed if it cannot be inferred from the context.

2.3 Information Inheritance

A sequence-factor has a knowledge base (also known as its descriptive base) that is explicitly defined by its target sequence, pattern and location list, and an evaluation base that it inherits from its motif set. The evaluation base is internally structured by its indicator list. A sequence-factor also has a defined behaviour that it inherits from its motif-set membership. This behaviour is monitored as the sequence-factor evolves.

Sequence-factors induce information on initial sequences. Once this happens, the region on the sequence subject to the influence of the sequence-factor implicitly inherits the evaluation and behavioural properties of it. Such a region on a sequence can be associated with the type of sequence factor inducing information on it. For example, a region on a sequence induced by a bounded sequence factor will be a bounded region on that sequence in relation to that sequence-factor.

This action of induction ties the sequence-factors to their target sequences and implicitly acts as constraints governing the alignment process. During this process, a sequence-factor evolves with its target sequence as it transforms from an initial sequence to an aligned sequence. A sequence-factor can evolve into a new sequence factor or succumb in the transformation. The function $Evolve\_\_$ captures this process. It is defined as follows:

PLEASE TURN OVER

Definition 2.32 ***function*** $Evolve\_\_$ **:** $\mathbb{P}\,\mathbb{f} \times \mathcal{P} \longrightarrow \mathbb{P}\,\mathbb{f} \cup \{\odot\}$

$$
\begin{array}{|l}
Evolve\_\_ : \mathbb{P}\,\mathbb{f} \times \mathcal{P} \longrightarrow \mathbb{P}\,\mathbb{f} \cup \{\odot\} \\
\hline
\forall f : \mathbb{P}\,\mathbb{f};\ s : \mathcal{P} \bullet \\
\qquad \left(\begin{array}{ll} \neg\ (f = \circledast) & \wedge \\ \exists_1 f' : \mathit{MotifSetExtension}(\{f\}) \bullet f'.1 = s & \end{array} \right) \iff Evolve(f, s) = f'
\end{array}
$$

.

Where $\exists_1$ is the unique existential quantifier *(see notations and basic definitions 11 on page 50)*. The function $Evolve\_\_$ takes as input a sequence factor and a pattern. The input pattern would usually describe the state of the induced sequence after an alignment. The result of this function is a new sequence-factor or the dead value. A dead value results if the sequence factor cannot alter its descriptive base to accommodate the new target sequence given by the input pattern. This alteration is dictated by the behavioural properties of the sequence factor. In other words, for the alteration to prevail, the new sequence factor must be an element of the motif set extension of the old sequence-factor. In such a case, the new sequence factor will inherit the rest of the properties (evaluation base with an altered indicator list if necessary and the motif set type membership) from the old sequence factor. The result of this process can either be used to compute an overall alignment score (discussed below) or used to ensure a succession of well-defined sequence factors for successive alignments. The above definition requires that the input sequence-factor be fully instantiated. Given an evolution of a fully instantiated sequence-factor, there can only be one sequence factor in its motif set extension that conforms to a given resultant target sequence. The antecedent to the survival of a sequence factor after an evolution is the deformation of its pattern. The defining attributes of this behaviour are inherited by a sequence-factor from its membership in either an unbounded or bounded motif-set. Sequence-factors of an unbounded motif set are free to acquire gaps in their pattern as they evolve into new sequence-factors. Sequence-factors of bounded motif sets on the other hand must preserve their pattern in order to survive an evolution. Accordingly, a sequence-factor survives an evolution if and only if it preserves its pattern during the process or is unbounded. This leads to an important design consideration: if the survival of a sequence-factor is important under all circumstances, as is the case for example in the basic sequence factors described in Figure 5, then it must be defined under an unbounded motif set.

2.4 Alignment

An alignment is carried out on a set of induced sequences and scored over all its surviving sequence factors. The process entails altering the target sequence of the induced-sequences by inserting zero or more gaps in them. The resulting sequences will be of equal length. As the target sequence of an induced sequence is synchronised with the target sequences of its inducing sequence-factors, the alteration of the target sequence of the induced sequence effectively alters the target sequence of those sequence-factors.

As mentioned above this process can lead to some sequence-factors dying and others evolving into new sequence factors. The end result can be seen as a collection of aligned sequence-factors. The process **does not** introduce any new sequence factors not already in the list of motif set sequence induction lists, even though there may be such members with the potential of instantiating with the new target sequence in one or more of the original motif-sets. The mapping between a set of initial induced sequences and their alignment is a total bijection described by the following generic definition.

Definition 2.33 ***function*** $AlignedSeq\_ : \mathbb{P}\,\mathbb{s} \rightarrowtail\!\!\!\!\twoheadrightarrow \mathbb{P}\,\mathbb{s}$

$$
\begin{array}{l}
[S_1, S_2] \\
\hline
AlignedSeq\_ : S_1 \rightarrowtail\!\!\!\!\twoheadrightarrow S_2
\end{array}
$$

We can now define the relationship between a set of initial induced sequences and their alignments. The relation $\_align\_$ is defined as follows:

Definition 2.34 ***relation*** $\_align\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \leftrightarrow \mathbb{P}\,\mathbb{P}\,\mathbb{s}$

$$
\begin{array}{l}
\_align\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \leftrightarrow \mathbb{P}\,\mathbb{P}\,\mathbb{s} \\
\hline
\forall\, S, S' : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \bullet \\
\quad S \; align \; S' \iff \\
\quad\quad \left(\begin{array}{lll}
AlignedSeq(S) = S' & & \wedge \\
\forall\, s_1', s_2' : S' \bullet \#s_1'.1 = \#s_2'.1 & & \wedge \\
\forall\, s : S \bullet \exists\, s' : S' \bullet & AlignedSeq[S, S'](s) = s' \quad \wedge & \\
 & s.1 \Subset s'.1 \quad \wedge & \\
 & s.1 \upharpoonright \Phi = s'.1 \upharpoonright \Phi \quad \wedge & \\
 & s'.2 = \bigcup\limits_{\forall\, l : s.2} \{(l.1, \{f : l.2 \bullet Evolve(f, s'.1)\})\} &
\end{array}\right)
\end{array}
$$

.

Given two sets of induced sequences, an initial set S and an aligned set S', the relation $S \; align \; S'$ holds if S' is a valid alignment of S. According to the above definition, for S' to be a valid alignment of S the following conditions must hold:

- The mapping between S and S' must be a total bijection.
- The length of the target sequence of all the induced sequences of S' must be the same.
- For every induced sequence s in the set S, there must be a unique induced sequence s' in the set S' such that the target sequence of s' is the result of an insertion of zero or more gaps in the target sequence of s and the list of motif set sequence induction lists of s' is the list of motif set sequence induction lists s with all its sequence factors evolved to accommodate the new target sequence.

The value of an alignment can be obtained by scoring the set of aligned induced sequences S'.

2.4.1 Scoring an Alignment

Given a set of **aligned** induced sequences, we can score it by taking the aggregate over all its induced sequence factors. The scoring model has two main genres described by the homointeractive and heterointeractive motif sets. Sequence-factors belonging to homointeractive motif sets are scored individually while those belonging to heterointeractive motif sets are scored in pairs. We have already introduced the alignment of factors score function $\mathit{AlignmentFactorScore}\_\_$ for scoring a pair of heterointeractive sequence-factors. We now introduce the *internal sequence-factor score function*, $\mathit{InternalFactorScore}\_$ for scoring homointeractive sequence-factors. The function $\mathit{InternalFactorScore}\_$ is defined as follows:

Definition 2.35 ***function*** $\mathit{InternalFactorScore}\_$ **:** $\mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
\mathit{InternalFactorScore}\_ : \mathcal{M} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall f : \mathbb{P}\,\mathbb{f} \bullet \\
\quad \exists R : \mathbb{R} \bullet \\
\qquad R = \sum_{\substack{n_1, n_2 \in f.3 \\ n_1 < n_2}} \mathit{Property}_1(f, n_1, n_2) \iff \mathit{InternalFactorScore}(f) = R
\end{array}
$$

The function $\mathit{InternalFactorScore}$ returns the aggregate location interaction score taken over all location pairs of a homointeractive sequence-factor. The location information is given by its location list. The scoring scheme is defined by the kernel function of its motif set. The location interaction score need not be directly computed from the location information. The location information can be tied to the pattern and/or indicator list of the sequence-factor. For example, the location interaction of the affine gap model (introduced with the $\mathit{Property}_1$ function) is tied to the pattern of the sequence factor. In this case, the location interaction is scored between two residues, a residue and a gap, two gaps or a gap and a residue whose values are taken from the corresponding locations of the pattern.

The score of an aligned set of induced sequences is the sum of the, *aggregate internal sequence factor scores* and the *aggregate alignment of sequence factor scores* taken over all homointeractive sequence factors for the first case and all pairs (taken between different induced sequences) of heterointeractive sequence-factors for the second case. In its simplest from, this score can be calculated using the two functions $\mathit{AlignmentFactorScore}$ and $\mathit{InternalFactorScore}$. We introduce the two aggregate functions $\mathit{AggregateInternalFactorScore}\_$ and $\mathit{AggregateAlignmentFactorScore}\_$ for computing the *aggregate internal sequence factor score* and the *aggregate alignment of sequence factors score.* The function $\mathit{AggregateInternalFactorScore}\_$ is defined as follows:

PLEASE TURN OVER

Definition 2.36 ***function*** $AggregateInternalFactorScore\,\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AggregateInternalFactorScore\,\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall S : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \bullet \\
\quad \exists F : \mathbb{P}\,\mathbb{P}\,\mathbb{f} \bullet \\
\qquad \left(\begin{array}{ll} S \; align \; S & \wedge \\ F = \bigcup \{ s : S \bullet \bigcup \{ l : s.2 \mid idm^{\sim}[\mathbb{M}](l.1) \in \mathbb{M}_{[o]} \bullet l.2 \} \} & \wedge \\ \neg\,(F = \varnothing) & \end{array} \right) \iff \\
\qquad\qquad AggregateInternalFactorScore(S) = \sum_{f \in F} InternalFactorScore(f)
\end{array}
$$

The function $AggregateInternalFactorScore$ takes a set of aligned induced sequences and returns the aggregate internal sequence-factor score of all its homointeractive sequence-factors. For a set of induced sequences, the predicate $S \; align \; S$ is true if and only if S is a valid alignment of itself. The set F in the above definition is the collection of all homointeractive sequence factors inducing information on the set of induced sequences S. Every homointeractive sequence factor has a kernel function defined over its location list which ensures that the function $InternalFactorScore$ returns a value. If the set F is not empty then a value is guaranteed from this function. The function $AggregateAlignmentFactorScore\,\_$ is defined as follows:

Definition 2.37 ***function*** $AggregateAlignmentFactorScore\,\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AggregateAlignmentFactorScore\,\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall S : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \bullet \exists F : \mathbb{P}\,\mathbb{P}\,\mathbb{f} \bullet \\
\left(\begin{array}{ll} S \; align \; S & \wedge \\ F = \{ s : S \bullet \bigcup \{ l : s.2 \mid idm^{\sim}[\mathbb{M}](l.1) \in \mathbb{M}_{[e]} \bullet l.2 \} \} & \wedge \\ \exists M_1, M_2 : F \bullet \exists f_1 : M_1, f_2 : M_2 \bullet \neg\,(AlignmentFactorScore(f_1, f_2) = \odot) & \end{array} \right) \\
\iff \\
\quad AggregateAlignmentFactorScore(S) = \sum_{\substack{M_1, M_2 \in F \\ M_1 \neq M_2}} \sum_{\substack{f_1 \in M_1 \\ f_2 \in M_2}} AlignmentFactorScore(f_1, f_2)
\end{array}
$$

The function $AggregateAlignmentFactorScore$ takes a set of aligned induced sequences and returns the aggregate alignment of sequence-factors score between all its heterointeractive sequence-factors. Note that, for a pair of sequence-factors f_1 and f_2 the function $AlignmentFactorScore(f_1, f_2)$ returns the dead value $\odot$ if f_1 and f_2 do not form a valid pair of aligned sequence-factors. This value does not affect the summation *(see notations and basic definitions 10 on page 50)*. For a set of aligned induced sequences, the total score of the alignment is given by the function $AggregateAlignmentScore\,\_$ which is defined as follows:

PLEASE TURN OVER

Definition 2.38 ***function*** $AggregateAlignmentScore\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\}$

$$
\begin{array}{|l}
AggregateAlignmentScore\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \longrightarrow \mathbb{R} \cup \{\odot\} \\
\hline
\forall S : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \bullet \\
\qquad AggregateAlignmentScore(S) = \; AggregateInternalFactorScore(S) \quad + \\
\qquad \qquad \qquad \qquad \qquad \qquad AggregateAlignmentFactorScore(S)
\end{array}
$$

We use the term *interaction unit* to refer to a homointeractive sequence-factor or a pair of heterointerative sequence-factors when used for scoring. A *valid interaction unit* is an interaction unit that has a valid score defined over its location-list (i.e. not $\odot$).

There can be may variations to the *aggregatealignmentscore* described above and the appropriate method selected for scoring sequence factors will depend on the nature of the information provided and will be at the discretion of the sequence aligner. For example, the aggregate alignment score returned by the above function favours positive information (i.e. information with a positive *InternalFactorScore* or *AlignmentFactorScore* score) in that it does not weight the final score with the total number of valid interaction units (i.e. the number of homointeractive and heterointerative pairs, of sequence factors scored) the score is based on. This can encourage the provision of indiscriminate information in order to improve the alignment score. A better score would be the *average aggregate alignment score*. The average aggregate alignment score is computed by taking the sum average of the $AggregateInternalFactorScore\_$ and $AggregateAlignmentFactorScore\_$ taken over all the sequence-factors in the prior case and sequence-factor pairs in the latter case that have a **valid** score. Another alternative to these scores is the unit scores, which are scores normalised to the value of a single interaction in a sequence factor or between a pair sequence factors. They are the average value of a single interaction in an interaction unit, which is usually proportional to the length of the location-list of the sequence-factor(s). This normalisation enables the comparison of sequence-factors with location-lists of varying length.

The method used to score a set of aligned induced sequences is a choice for the sequence aligner. Depending on the circumstances, one method may be more appropriate than the other. For example, if one was judicious in the provision of information and attribute valid information units to the re-enforcement of the quality of an alignment then, the *aggregate alignment score* which scores an alignment in addition to its quality, in proportion to the amount of information provided will be better than the other normalised scores. On the other hand, if information was lax with possible redundancies, then a normalised Score will give a better picture of the quality of the alignment. Given a scoring scheme, an optimal alignment of a set of induced-sequences is the alignment with the maximum score for that scoring scheme. The method in which a final alignment is scored will affect the computational process of an optimal alignment.

2.4.2 Optimal Alignment

The goal of an alignment is to provide a biologically significant match between a given set of sequences. It is reasonable to assume that a good correlation exists between the certainty of an inference made from an alignment and the affinity of the alignment to already known information. Sequences are induced with information from sequence-factors to form induced sequences. The value of information so induced will vary from sequence-factor to sequence-factor. An alignment of a set of induced sequences usually requires the alteration of one or more of its target sequences. Such alterations lead to the destruction of some interaction units and the creation of new ones. The goal is to obtain an alignment with a configuration that maximises the total score of its interaction units. Such an alignment is called an *optimal* alignment. We define a relation $\_\,OptimalAlign\,\_$ which relates a set of induced sequences and its optimal alignments as follows:

Definition 2.39 ***relation*** $\_\,OptimalAlign\,\_$ ***:*** $\mathbb{P}\,\mathbb{P}\,\mathbb{s} \leftrightarrow \mathbb{P}\,\mathbb{P}\,\mathbb{s}$

$$
\begin{array}{|l}
\_\,OptimalAlign\,\_ : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \leftrightarrow \mathbb{P}\,\mathbb{P}\,\mathbb{s} \\
\hline
\forall\, S, S_1 : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \bullet \\
\quad S \; OptimalAlign \; S_1 \iff \\
\qquad\qquad \left(\begin{array}{ll} S \; align \; S_1 & \wedge \\ \forall\, S_2 : \mathbb{P}\,\mathbb{P}\,\mathbb{s} \mid S \; align \; S_2 \bullet Score(S_1) \geq Score(S_2) & \end{array} \right)
\end{array}
$$

The function $Score\,\_$ in the above definition can be any of the scoring functions described above or any other valid scoring function. There can be more than one optimal alignment for a given set of induced sequences. Any alignment that scores less than the optimal alignment is called a *sub-optimal* alignment. There are many instances in which the production of an optimal alignment can be computationally impractical. The alignment of multiple sequences is one such instance. In this formalism, there are two main determinants to the computational complexity of computing an alignment. The first is the computational complexity of estimating scores attributable to the participating interaction units at their regions of influence. The second is the computational complexity involved in optimally combining these individual scores to obtain the overall alignment. Computational complexities that exceed quadratic time are generally not feasible. This is a measure that is taken relative to the number of individual location pairs need to be considered in scoring an interaction unit. In situations where an optimal alignment is impractical, we would strive to obtain a sub-optimal alignment with a score that comes as close as possible to the optimal.

A Notations and Basic Definitions

The notations used here follow the Z specification language. For further details see Woodcock & Davis [59].

1. Variables that represent individual objects (e.g. number, character, sequence, sequence factor (defined in the text), etc.) are written in simple letters (e.g. n, c, s, f, etc.)

2. Variables representing sets or sequences of individual objects (e.g. sequence of numbers, sequence of characters, set of sequences, set of sequence factors, etc.) are written in capital letters (e.g. N, C, S, F, etc.).

3. Data types defining individual objects (e.g. sequence factor, induced sequence (defined in the text), etc.) are written in simple letters and in 'mahpzc' LaTeXfont (e.g. $\mathbb{f}$, $\mathbb{s}$, etc.).

4. Data types defining sets of individual objects (e.g. motif set (defined in the text), set of induced sequences, etc.) are written in capital letters and in 'mathcal' LaTeXfont (e.g. $\mathcal{M}$, $\mathcal{S}$, etc.).

5. Data types defining sets of sets of individual objects (e.g. motif set collection, set of sets of induced sequences, etc.) are written in capital letters and in 'mahcal' LaTeXfont (e.g. $\mathbb{M}$, $\mathbb{S}$, etc.).

6. To describe a set of a particular kind of objects, the variable or data type identifier is subscripted within square brackets by an expression consisting of simple letters identifying the particular kind of set. For example, we can represent a *heterointeractive motif set* (defined in the text) as $M_{[e]}$ where the subscript e identifies the set to be a set with the properties of a heterointeractive motif set. Then, the data type identifying the set of all heterointeractive motif sets will be represented as $\mathbb{M}_{[e]}$ and $M_{[e]} \in \mathbb{M}_{[e]}$. As another example, we can represent the set of all *bounded motif sets* (defined in the text) as $\mathbb{M}_{[t]}$ where the subscript t identifies these sets to be sets with the properties defined for a bounded motif set. We can now combine different types of sets to form new sets. For example, we can represent a motif set that is both heterointeractive and bounded as $M_{[e\cap t]}$ i.e. $M_{[e\cap t]} \in \mathbb{M}_{[e]} \wedge M_{[e\cap t]} \in \mathbb{M}_{[t]}$. As the set intersection comes up in many places we will write a representation like $M_{[e\cap t]}$, simply as $M_{[et]}$.

7. ***Prefix 'initial−' :*** We use the prefix *initial* before an object to distinguish the object as a part of the primary input to a given process which can involve one or more sub-process and should be considered in context. In other words, they are the inputs before the process in question does any processing on them. These can be data provided by the user or data gathered from a pre-processing stage. For example the set of sequences given as input to the program will be referred to as the initial-sequences to the program. The set of motif sets after the second iteration

of a recursion will be the initial-motif-sets to the third iteration of the recursion but **not** to the recursion itself.

8. ***Metacharacter $'*'$:*** The generic metacharacter '$*$' is used to represent any *valid* and *defined* entry at the point it is specified. This character can be used when we do not know the value that instantiates a variable or the variable is not yet instantiated or we do not care about this value but we know that it is a well-defined value from the set of values the variable can take. For example if $n \in \mathbb{Z}^+$ and $n = *$ means $n = 1$ or $n = 2$ or $n = 3$ etc or n has not been allocated a value yet. In another example of the use of the character '$*$', if we have a function $afs\_\_$ (defined in the text) which takes two sequence factors and returns a real value or the *undefined* value '$\odot$' (described below), the statement $afs(f_1, f_2) = *$ says that the function afs given the two sequence factors f_1 and f_2 produces a valid output which can be 1.0 or 15.344 or any other real number but it **cannot** be the undefined value '$\odot$'. The presence of the character '$*$' does not make a statement as to whether or not the variable is instantiated to a particular value at this point. For example, if the variable $n = 5$ but we are not interested in the value of this variable, we can write $n = *$ but n will still be instantiated to the value 5 which we are not concerned with at this point

9. ***Metacharacter '$\circledast$' :*** The generic metacharacter '$\circledast$', like the character '$*$', is used to represent any *valid* and *defined* entry at the point it is specified but it also makes the additional statement that the variable is not instantiated to a particular value at the point it is specified. For example, if we say that the integer variable $n = \circledast$, we are making the statement that no particular integer value has been allocated to n at this point but n is free to take any integer value later on. For any $n - tuple$ $T == (t_1, t_2, \cdots, t_n)$ if $\exists\, i : \{1, \cdots, n\} \bullet (\, t_i = \circledast\,) \implies (\, T = \circledast\,)$, accordingly, the statement $\neg\ (T = \circledast)$ means that every variable in the tuple T is uniquely instantiated.

10. ***Metacharacter '$\odot$' :*** The generic metacharacter '$\odot$' is used to specify an *undefined* or *dead* value. The union of the value $\odot$ is valid for any set. For any set X the following is always true $\forall\, x \in X \cup \{\odot\} \bullet x \circledast \odot = x$ where $\circledast$ is any binary operator defined on X and for any function $\mathbf{F}$, $\mathbf{F}(\odot) = \odot$. For example if $n \in \mathbb{Z}$ then $n + \odot = n$, $n - \odot = n$, $n * \odot = n$, $n/\odot = n$, etc. In another example, if we have a function $mse\_$ (defined in the text) which takes as input a motif set and returns a motif set or $\odot$, the statement $mse(M) = \odot$ says that the function mse given the motif set $M \in \mathbb{M}$ returns an undefined of dead output. Given a motif set $M' \in \mathbb{M}$, the result of an operation like $M' \cup mse(M)$ will be the motif set M' i.e. $M' \cup mse(M) = M'$. This is true for all operators e.g. $M' \cap mse(M) = M'$, $M' \setminus mse(M) = M'$, etc. because $mse(M) = \odot$.

11. ***Unique Quantifier $\exists_1$:*** The unique existential quantifier specifies that there is

exactly one object with a given property in a set (see [59]). It is defined as follows:

$$\begin{aligned}\exists_1\, x : a \mid p \bullet q \quad \iff \quad & \exists\, x : a \mid p \bullet q \\ & \wedge \\ & \forall\, y : a \bullet p[y/x] \wedge q[y/x] \implies y = x\end{aligned}$$

12. ***Definite Description 'μ' :*** The μ-notation is used for the definite description of an object. The statement $\mu\ x : a \mid p$ denotes the **unique** object x from a satisfying the predicate p. For further information on the μ-notation, refer to Woodcock and Davies 1996 [59].

13. ***Subsequence $\_ \Subset \_$:*** seq $X \times$ seq X
    The subsequence relation $\_ \Subset \_$ is defined as follows:

$$\begin{array}{|l}
[X] \\ \hline
_ \Subset _ : \text{seq } X \leftrightarrow \text{seq } X \\ \hline
\forall\, s_1, s_2 : \text{seq } X \bullet \\
\quad s_1 \Subset s_2 \iff \exists\, I : \mathbb{P}\,\mathbb{N} \bullet \mathit{squash}(I \lhd s_2) = s_1 \\ \hline
\end{array}$$

14. ***Indexed sequence $\_[\_]$:*** seq $X \times$ seq $\mathbb{N} \longrightarrow$ seq X
    The indexed sequence operator $\_[\_]$ is defined as follows:

$$\begin{array}{|l}
[X] \\ \hline
_[_] : \text{seq } X \times \text{seq } \mathbb{N} \longrightarrow \text{seq } X \\ \hline
\forall\, s : \text{seq } X;\ l : \text{seq } \mathbb{N} \bullet \\
\quad \forall\, i : \text{ran}(l) \bullet 1 \leq i \leq \#\, s \wedge \\
\qquad s[l] = \{R : \mathbb{N} \times X \mid x \mapsto y \in R \iff x \in \text{dom}(l) \wedge y = s(l(x))\} \\ \hline
\end{array}$$

15. ***Pattern $\mathcal{P}$:*** $\mathbb{P}\,\text{seq}(\Phi \cup \{\Delta, \ltimes, \rtimes\})$
    A *pattern* $\mathcal{P} == \{p : \text{seq}(\Phi \cup \{\Delta\}) \mid p \upharpoonright \Phi \neq \langle\rangle\} \cup \{p : \langle\ltimes\rangle \frown \text{seq}(\Phi \cup \{\Delta\}) \frown \langle\rtimes\rangle \mid p \upharpoonright \Phi \neq \langle\rangle\}$
    Where:

    - Φ is an alphabet.
    - $\Delta \notin \Phi$ is the gap character.
    - $\ltimes \notin \Phi$ is the sequence start character.
    - $\rtimes \notin \Phi$ is the sequence end character.

16. ***function*** $idm\_\_ : X \longrightarrow \mathbb{N}$
The generic *injective function* $idm\_\_$ that uniquely identifies each element of a set X with a natural number is defined as follows:

$$
\begin{array}{|l}
\hline
[X] \\
idm\_\_ : X \rightarrowtail \mathbb{N} \\
\hline
\end{array}
$$

References

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

[2] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Journal of Molecular Biology*, 25:3389–3402, 1997.

[3] T. K. Attwood, M. E. Beck, D. R. Flower, P. Scordis, and J. N. Selley. The PRINTS protein fingerprint database in its fifth year. *Nucleic Acids Res.*, 26:304–308, 1998.

[4] A. Bairoch and R. Apweiler. The SWISS-PROT protein sequence data bank and its supplement TrEMBL in 1998. *Bioinformatics*, 26(2):38–42, 1998.

[5] A. Bairoch, P. Bucher, and K. Hofmann. The PROSITE database, its status in 1997. *Nucleic Acids Res.*, 26(1):217–221, 1997.

[6] G. J. Barton and M. J. E. Sternberg. A strategy for the rapid multiple alignment of protein sequences. *Journal of Molecular Biology*, 198:327–337, 1987.

[7] D. A. Benson, M. S. Boguski, D. J. Lipman, J. Ostell, and B. F. Ouellette. GenBank. *Nucleic Acids Res.*, 26(1):1–7, 1998.

[8] M. P. Berger and P. J. Munson. A novel randomised iterative strategy for aligning multiple protein sequences. *Computer Applications in the Biosciences*, 7:479–484, 1991.

[9] H. M. Berman, J. Westbrook, E. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28:235–242, 2000.

[10] K. Binder and D. W. Heerman. *Monte Carlo simulation in Statistical Physics*. Springer-Verlag, 1988.

[11] K. Bucka-Lassen, O. Caprani, and J. Hein. Combining many multiple alignments in one improved alignment. *Bioinformatics*, 15(2):122–130, 1999.

[12] G. A. Churchill. Stochastic models for heterogeneous DNA sequences. *Bulletin of Mathematical Biology*, 51:79–94, 1989.

[13] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in protein. In *In Dayhoff, M. O, ed., Atlas of Protein Sequence and Structure*, volume 5 of *3*. National Biomedical Research Foundation, Washington D.C, 1978.

[14] E. Depiereux and E. Feytmans. MATCH-BOX - A fundamentally new algorithm for the simultaneous alignment of several protein sequences. *Comput. Appl. Biosci.*, 8(5):501–509, 1992.

[15] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis (Probabilistic models of protein and nucleic acids).* Cambridge University Press, 1998.

[16] S. R. Eddy. Profile hidden Markov models. *Bioinformatics*, 14:755–763, 1998.

[17] D. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360, 1987.

[18] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[19] Genetics Computer Group. Wisconsin Package v.8. *Madison, WI.*

[20] D. George, W. Barker, H. Mewes, F. Pfeiffer, and A. Tsugita. The pir-international protein sequence database. *Nucl. Acids. Res.*, 24:17–20, 1996.

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

[22] O. Gotoh. Significant improvement in accuracy of multiple protein alignments by iterative refinement as assessed by reference to structural alignments. *Journal of Molecular Biology*, 264:823–838, 1996.

[23] Gusfield. Efficient methods for multiple sequence alignment with guaranteed error bounds. *Bull. Math. Biol.*, 55(1):141–154, 1993.

[24] T. Heinemeyer, E. Wingender, I. Reuter, H. Hermjakob, A. E. Kel, O. V. Kel, E. V. Ignatieva, E. A. Ananko, O. A. Podkolodnaya, F. A. Kolpakov, Podkolodny N. L., and N. A. Kolchanov. Databases on Transcriptional Regulation: TRANSFAC, TRRD, and COMPEL. *Nucleic Acids Res.*, 26:364–370, 1998.

[25] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA.*, 89:10915–10919, 1992.

[26] S. Henikoff, S. Pietrokovski, and J. G. Henikoff. Superior performance in protein homology detection with the Blocks Database servers. *Nucleic Acids Res.*, 26(1):309–312, 1998.

[27] J. Heringa. Two strategies for sequence comaprison: profile-preprocessed and secondary structure-induced multiple alignment. *Computers and Chemistry*, 23:341–364, 1999.

[28] D.G. Higgins, A.J. Bleasby, and R. Fuchs. CLUSTAL V: improved software for multiple sequence alignment. *Comput Appl Biosci*, 8(2):189–191, 1992.

[29] D. S. Hirschberg. A linear-space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18:341–343, 1975.

[30] P. Hogeweg and B. Hesper. The alignment of sets of sequences and the construction of phyletic trees: an integrated method. *Journal of Molecular Evolution*, 20:175–186, 1984.

[31] X. Huang and W. Miller. A time–efficient, linear-space local similarity algorithm. *Adv Appl Math*, 12:337–357, 1991.

[32] A. J. Jennings, C. M. Edge, and M. J. Sternberg. An approach to improve multiple alignments of protein sequences using predicted secondary structure. *Prot. Engin.*, 4:227–231, 2001.

[33] J. Kececioglu and W. Zhang. Aligning alignments. *Proceedings of the 9th Symposium on Combinatorial Pattern Matching, Springer-Verlag Lecture Notes in Computer Science*, 1448:189–208, 1998.

[34] R. D. King and M. J. Sternberg. Identification and application of the concepts important for accurate and reliable protein secondary structure prediction. *Prot. Sci.*, 5:2298–2310, 1996.

[35] J. R. Knight and E. W. Myers. Approximate Regular Expression Pattern Matching with Concave Gap Penalties. *Proceedings of the Third Symposium on Combinatorial*, pages 67–78, 1992.

[36] H. P. Lenhof, K. Reinert, and M. Vingron. A Polyhedral Approach to RNA Sequence Structure Alignment . *Journal of Computational Biology*, 5:517–530, 1998.

[37] V. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Daklady*, 10:707–710, 1966.

[38] B. Morgenstern, A. Dress, and T. Werner. Multiple DNA and protein sequence alignment based on segment-to-segment comparison. *Proc. Natl. Acad. Sci. USA*, 93:12098–12103, 1996.

[39] B. Morgenstern, K. Frech, A. Dress, and T. Werner. DIALIGN: Finding local similarities by multiple sequence alignment. *Bioinformatics*, 14:290–294, 1998.

[40] A. M. Moseley, J. Sleep, M. McGowran, M. Garcia-Pastor, and P. Sterk. The EMBL Nucleotide Sequence Database. *Nucleic Acids Research*, 26(1):8–15, 1998.

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

[42] C. Notredame and D. G. Higgins. SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*, 24:1515–1524, 1996.

[43] C. Notredame, D. G. Higgins, and J. Heringa. T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302:205–217, 2000.

[44] W. R. Pearson and D. J. Lipman. Improved tools for biological sequence comparison. *Proc. Natl. Acad. Sci. USA*, 85:2444–2448, 1988.

[45] F. Plewniak, J. D. Thompson, and O. Poch. Ballast: Blast post-processing based on locally conserved segments. *Bioinformatics*, 16(9):750–759, 2000.

[46] R. C. Prier, T. Junier, and Bucher P. The Eukaryotic Promoter Database EPD. *Nucleic Acids Research*, 26(1):353–357, 1998.

[47] N. Saitou and M. Nei. The neighbour-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[48] C. Sander and R. Schneider. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9(1):56–68, 1991.

[49] T. F. Smith and M. S. Waterman. Identification of common molecular sequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[50] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

[51] E. Sonnhammer, S. Eddy, E. Birney, A. Bateman, and R. Durbin. Pfam: multiple sequence alignments and HMM-profiles of protein domains. *Nucleic Acids Research*, 26(1):320–322, 1998.

[52] W. R. Taylor. A flexible method to align large numbers of biological sequences. *Journal of Molecular Evolution*, 28:161–169, 1988.

[53] W. R. Taylor. Motif-biased protein sequence alignment. *Journal of Computational Biology*, 1(4):297–310, 1994.

[54] J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680, 1994.

[55] J. D. Thompson, F. Plewnial, J. C. Thierry, and O. Poch. Rapid and reliable global multiple alignments of protein sequences detected by database searches. *Nucleic Acid Research*, 28(15):2919–2926, 2000.

[56] J. D. Thompson, F. Plewnial, J. C. Thierry, and O. Poch. Rapid and reliable global multiple alignments of protein sequences detected by database searches. *Bioinformatics*, 28(15):2919–2926, 2000.

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

[58] M. S. Waterman and M. D. Perlwitz. Line geometries for sequence comparisons. *Bulletin of Mathematical Biology*, 46:567–577, 1984.

[59] Jim Woodcock and Jim Davies. *Using Z: Specification, Refinement, and Proof.* Prentice Hall International Series in Computer Science. Prentice Hall, London, 1996.