Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Master Thesis

European Master in Computational Logics

Multiple sequence alignment correction using constraints

Luciano M. Guasco[1]
EMCL[2]
FCT/UNL[3]
lucianoguasco@gmail.com ()

Lisboa
(2010)

[1] supervised By Dr. Ludwig Krippahl.
[2] European Master in Computational Logics
[3] Faculdade de Ciências e Tecnologia / Universidade Nova de Lisboa, Caparica, Portugal

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Master Thesis

Multiple sequence alignment correction using constraints

Luciano M. Guasco[4]
EMCL[5]
FCT/UNL[6]
luciano.guasco@gmail.com ()

Orientador: Dr. Ludwig Krippahl

[4] supervised By Dr. Ludwig Krippahl.

[5] European Master in Computational Logics

[6] Faculdade de Ciências e Tecnologia / Universidade Nova de Lisboa, Caparica, Portugal

Trabalho apresentado no âmbito do European Master in Computational Logics, como requisito parcial para obtenção do grau de Mestre em Computational Logics.

Lisboa
(2010)

To my grandfather, who introduced me in the wonderful world of science.

Acknowledgements

First of all, I would like to thank my supervisor Dr. Ludwig Krippahl for his guidance, advises, continue feedback and enthusiasm to answer my doubts about many topics.

I want to express my thanks to all the CENTRIA people, that accompanied me during my studies in Portugal, specially when we shared many chats at the incredible open houses.

My thanks to the EMCL consortium, for the scholarship I got to complete this studies, money without which it would have been impossible for me to study in Europe.

I would like to give thanks to my family and my friends for their continuous support and the strength transmitted to commit this work.

I also would like to thank my girlfriend. I am really grateful for her continuous moral support and advises.

I would like to give a special thank to my friend MSc. David Buezas, for all our brainstormings while tasting the Portuguese pastry.

Abstract

One of the most important fields in bioinformatics has been the study of protein sequence alignments. The study of homologous proteins, related by evolution, shows the conservation of many amino acids because of their functional and structural importance. One particular relationship between the amino acid sites in the same sequence or between different sequences, is protein-coevolution, interest in which has increased as a consequence of mathematical and computational methods used to understand the spatial, functional and evolutionary dependencies between amino acid sites. The principle of coevolution means that some amino acids are related through evolution because mutations in one site can create evolutionary pressures to select compensatory mutations in other sites that are functionally or structurally related.

With the actual methods to detect coevolution, specifically mutual information techniques from the information theory field, we show in this work that much of the information between coevolved sites is lost because of mistakes in the multiple sequence alignment of variable regions. Moreover, we show that using these statistical methods to detect coevolved sites in multiple sequence alignments results in a high rate of false positives.

Due to the amount of errors in the detection of coevolved site from multiple sequence alignments, we propose in this work a method to improve the detection efficacy of coevolved sites and we implement an algorithm to fix such sites correcting the misalignment produced in those specific locations.

The detection part of our work is based on the mutual information between sites that are guessed as having coevolved, due to their high statistical correlation score. With this information we search for possible misalignments on those regions due to the incorrect matching of amino acids during the alignment. The re-alignment part is based on constraint programming techniques, to avoid the combinatorial complexity when one amino acid can be aligned with many others and to avoid inconsistencies in the alignments.

In this work, we present a framework to impose constraints over the sequences, and we show how it is possible to compute alignments based on different criteria just by setting constraint between the amino acids. This framework can be applied not only for improving the alignment and detection of coevolved regions, but also to any desired constraints that may be used to express functional or structural relations among the amino acids in multiple sequences. We show also that after we fix these misalignments, using constraints based techniques, the correlation between coevolved sites increases and, in general, the new alignment is closer to the correct alignment than the MSA alignment.

Finally, we show possible future research lines with the objective of overcoming some drawbacks detected during this work.

Keywords: Multiple Sequence Alignment, Molecular Coevolution, Mutual Information, Constraints, Amino Acids Correlation

Contents

List of Figures

List of Tables

1

Introduction

Many methods have been proposed to understand the evolutionary dynamics of organisms using Multiple Sequence Alignments as the main tool.

Of special interest to this work is the coevolutionary relationships among amino acids in different position of protein sequences, because this relationship shows an evolutionary dependency between these regions.

The importance of correct sequences alignment that include co-evolved sites, stems from the fact that conserved correlation between this sites through evolution from a common ancestor shows a functional or structural relation between them.

During this work we show how the multiple sequence alignment produces mistakes when aligning the more variable regions where coevolution might have occurred, because alignment algorithm tries to optimize the conservation of sequences, based on static substitution weight matrices, and much of the correlation information is lost after the alignment.

In this work we also show some techniques to discover coevolution, and we propose some improved methods to do so.

Based on constraint techniques we also propose some methods to set amino acid's position restrictions in the sequences and between different sequences, with the objective of aligning some parts of the sequences for recover the lost coevolution information. To do this, we propose several models to represent the information in the sequences and constraints to reduce the search space, resulting in an efficient tool for correcting the alignment errors and search possible solutions that increase the mutual information between coevolving sites.

Constraint programming is a good technique to solve combinatorial problems, since constraint propagation can greatly reduce the search space. In our specific problem, we use constraints to solve many alignment conflicts, pruning the variables' domains and reducing the search space of allowed alignments. In this work, a framework for handling protein sequences and alignment constraints is presented, and it can be used in the future in any problem involving constraints on sequences.

1.1 Layout of this work

In Chapter 2, we give an introduction to the basis and definitions that we use in the next chapters. We explain some problems and previous works on protein sequence alignments, multiple sequence alignment, coevolution, constraint programming, and we also introduce our protein sequences simulator.

In Chapter 3 we present the objectives of this work.

In Chapter 4 we present the mutual information concept, some techniques already explored, and we present our tool for mutual information visualization and some results.

In Chapter 5 we model the sequences in a constraint framework, where we propose some methods for setting constraint and adding dynamic alignment. We also test the methods and show the results.

In Chapter 6 we present a method to disambiguate the uncertainty of which sites are coevolved and we specify a technique for using the method of chapter 5 to fix the alignment. We also test the different algorithms and show the results.

In Chapter 7 we present the conclusions and future work.

2

Preliminaries

2.1 Sequence Alignments

A protein sequence is an ordered list of amino acids. The amino acids are biological entities, that we represent as a set of 20 characters. Each amino acid occupies a position, called a site in the sequence. In this work, we focus only on the primary structure level of the proteins, working directly with the amino acids sequences.

Protein and DNA sequencing has become an essential tool in the study of the structures and functions of proteins in any kind of living organisms, to understand cellular processes, and even to study such organisms in many scientific areas such as ecology and species conservation. There are many methods to determine protein sequences, whether from DNA sequencing or working directly with proteins as is the case of *Mass spectrometry* and *Edman degradation*.

Once the protein sequences are determined, a *Sequence Alignment* is a way of comparing and aligning the sequences according to the amino acids present in the correspondent sites, from which some information can be inferred, such as the functional and structural importance of conserved regions and relations between sequences.

An alignment is the match of the amino acids of a sequence to corresponding amino acids in the other sequence or sequences. For this, gaps can be inserted when a change in position is desired. For instance, in table 2.1 we can see how two sequences, s1=KLYECNERSA and s2=ECNEERSKA are aligned.

K	L	Y	E	C	N	E	-	R	S	-	A
-	-	-	E	C	N	E	E	R	S	K	A

Table 2.1: Two sequences aligned.

In many representations of alignments, also in table 2.1, the sequences are arranged in rows so that the aligned residues, i.e. aligned positions, appear in successive columns. The columns with a total match in the amino acid can correspond to a conserved region in the sequences, information that never changed in the evolution from the common ancestor.

A mismatch in a column can correspond to a mutation, a replacement of one amino acid by other during the evolutionary process through the generations of the species, or it can be a mismatch that can be aligned with the correspondent successive amino acid in the sequences adding a gap.

We can identify the conserved regions as part of the sequences that are perfectly conserved through the linages in the evolution history of the common ancestor. These conserved regions, as proposed in [NH03], suggest that such regions have a structural or functional importance.

The reason why *sequence alignments* are so important in bioinformatics, is due to the identification of sequences similarity that many times is assumed to reflect a degree of evolutionary change from a common ancestor. So, in this case, if we align sequences we can see the differences among them as changes in the evolutionary lineage.

The alignments are important, because they can tell how related two sequences are, and which regions of the sequences they share, generally related to a functional or structural similarity in two sequences. Biologists usually use these alignments to compare sequences that have a common ancestor from which they inherit this functionality or structural sections.

Sequence Alignments is a widely used tool in many fields, and many algorithms have been proposed to solve this problem. In this work, we use them to realize some experiments.

2.2 Algorithms for Sequence Alignment

Given two protein sequences, which we can suppose share a common ancestor, they can be aligned according to their amino acid similarity. If a site contains the same amino acid, it means a match, we can say that the amino acid is conserved through the linage. Mismatches can be interpreted as mutations, if two amino acid differ in the same site, or as an indel introduced in some linage, if one of the sequences contains an insertion or a deletion mutation.

In this section we mention the basic algorithms for two sequence alignment and in the next section we analyze what happens when we want to align more than two sequences (Multiple Sequence Alignment).

Finding the alignment between two protein sequences is a complex task that can be tackled with many techniques. These approaches in general in two categories: *global alignment* and *local alignment*. *Global alignments* attempt to align the whole sequences, a form of global optimization that accounts for all regions wether they match or not. *Local alignments* instead, align those regions which are similar in both sequences and ignore regions that are too divergent.

We mention two approaches based on dynamic programming. One is a global alignment algorithm, Needleman-Wunsch [NW70], and the other a local alignment algorithm, Smith-Waterman [SW81]. We invite the reader to follow the correspondent papers to read more about the algorithms, but in this work it is enough to take into account that these methods are useful when pairwise alignment is needed, in some optimized version of quadratic time.

For both algorithms we need the *substitution matrix* concept. A *substitution matrix* is a matrix that, for every pair of amino acids, shows a measurement of how probable is that one of the amino acid changes to the other. These matrices are useful when we are aligning, because we should determine if its convenient to proceed over a mismatch with the insertion of a gap or consider it as a point mutation. The substitution matrices can be constructed from statistical analysis of many well known sequences where point mutations occur. Actually, many matrices already exist and can be used with this purpose, as it is the case of BLOSUM matrices [HH92]

2.3 Algorithms for Multiple Sequence Alignment

Given three or more protein sequences or DNA sequences, a multiple sequence alignment algorithm is a method to search the best alignment of the sequences. It is usually assumed that such sequences share an evolutionary relationship and are descendant of a common ancestor. Also these algorithms can produce a phylogenetic tree with an evolutionary lineage hypothesis.

These algorithms differ in the way they search for the alignment. Some of them are global alignment, local alignment or heuristic based search. The complexity of these algorithms is always bigger than the complexity of pairwise alignment, and that is why many approaches are based on heuristic rather than in the complete search of the optimal alignment.

There exists a classification of these algorithms according to the way they proceed over the sequences. In the next section we review the most important techniques for multiple protein sequence alignment, pinpointing the advantages and drawbacks of each one.

2.3.1 Dynamic Programming

Dynamic programming technique can be used to produce the globally optimal alignment. The algorithm receives a set of gap penalties and a substitution matrix assigning scores according to the alignment of each possible pair of amino acids, based on the similarity and the probability of change of them.

For n sequences, the algorithm constructs a n-dimensional matrix, equivalent to the 2-dimensional constructed for 2 sequences as described in previous section. The complexity of the search space increases exponentially when a new sequence is added, as this means a new matrix dimension. The multiple sequence alignment algorithm with dynamic programming technique, tries to minimize the sum of the scores of the matrix giving a way to traverse backward the matrix and determines which sites align with each other, according to the direction of the path, that can fall in a decision of add gap to some of the sequences or leave the partial alignment as it is in that position.

This algorithm has a complexity, governed by the matrix creation step, of $O(S^N)$, where S is the length of the sequences and N the number of sequences. As has been shown in [WJ94], the problem of finding optimum global alignment for n sequences using this method, is a NP-complete problem.

2.3.2 Progressive Alignment

The progressive alignment method is a technique in which the main steps can be clearly identified: First, a *guide tree* is constructed, representing a hierarchical or topological order according to the similarity of the sequences. This tree is usually constructed with a cluster algorithm such as neighbor-joining, as in [SN87], using a distance matrix that can be constructed with the pairwise alignment using dynamic programming. Second, once the *guide tree* is constructed, it provides an order in which the sequences should be aligned. Thus the two most closely related sequences, at the root of the tree, are aligned fixing an alignment which now will be aligned with the next sequence in the tree. The algorithm continues aligning each sequence, according to the tree constructed in the previous step, with the accumulated alignment until all the sequences are aligned.

In the way this method works, the alignment may not be globally optimal because the two most similar sequences are aligned and then this alignment is aligned with the next sequence, and so on. This process is susceptible to propagating the mistakes made in the previous steps to all future alignments with the remaining sequences. This is why in some studies, as [DWFG10], they conclude that the errors in the final result increase for more dissimilar sequences.

One of the most popular MSA application based on the progressive method is Clustal[LBB+07]. Clustal works by calculating all the pairwise distance for the sequences and constructing the tree from such distances. The pairwise distances can be calculated with a fast and not so precise method, or with one slower but more accurate based on dynamic programming algorithms. With this distance information, the tree is constructed with the cluster algorithms and stored in a dendrogram (a specific guide tree). Then, with the guide tree defining the order, the progressive alignment strategy is applied to obtain the final alignment. The performance of Clustal was determined in [TPP99] and motivate us to test the alignment with the coevolution sequences as we describe in the objectives section.

2.3.3 Iterative Alignment

If we think in the previous method, the progressive one, but now we relax the concept that every intermediate alignment is fixed, in the sense that we can adjust it in future alignment steps, we have an iterative method. Mainly we lose efficiency, lacking the straightforward steps of the progressive method, but we gain accuracy because a correction procedure is accepted after an intermediate alignment is done.

There are many iterative algorithms, as explained in [Bat05], and they have different approaches in the way they select the sequence to align and also the correction procedure after the intermediate alignments.

The iterative methods are also heuristic in the search of the best multiple sequence alignment. The idea is that many refinements are done when possible to improve the alignment in each iteration, computing a solution of the problem as a modification of an existing sub-optimal solution, driving to a good but sometimes not optimal solution [Bax05]. Anyway, the final alignments obtained by this methods are robust.

Some applications developed using this techniques are DIALIGN [MFDW98] and MUSCLE [Edg04].

2.4 Coevolved sites

In a general biological view, coevolution is the change of some state of a biological entity triggered or pressured by the change of a related entity. In nature this coevolution appear at many levels, for example in the evolution of two species where a particular change in one of them makes the other evolve and adapt to that change. It is the case of many related species that have coevolved for long time, as *hummingbirds* and *ornithophilous* flowers, or *angracoid orchids* and *African moths*. Although coevolution was also part of Darwin's work *The Origing of the Species* [Dar95], where there are many examples of pair of species evolving and adapting in a race against each other, the definition of the concept is attributed to Ehrlich and Raven, in a study on butterflies and plants evolution [ER64].

Many techniques have been used to understand the evolutionary dynamics of organisms through the analysis of multiple sequence alignments. Through these studies, the importance of conserved residues and correlated residues started to be a popular topic since the studies focused on protein function and structure.

When performing sequence alignment it is often assumed that a mutation in one site is independent of the remaining sites. In many cases, however, sites are related to others and their evolution is dependent on those. For instance, this can occur when two residues are close in the protein spatial structure and some interaction between them is established. In this case, the evolution of these sites will depend on each other. Thus, a mutation on one of them will create a selection pressure for mutants on the other site, leading to a correlation in their evolution.

In the last few years many studies aimed to uncover coevolutionary relationships in amino acids in protein sequences and how they were arose because of their evolutionary importance. For instance, in [FT06] we can see many examples of how the correlations between amino acids are related with the functional role of these residues and why these correlations appear in the evolution of protein sequences.

	i	j			m			n
$Seq_1:$	P	E	V	A	L	P	L	G
$Seq_2:$	P	E	A	A	L	P	L	G
$Seq_3:$	R	E	V	A	L	P	L	A
$Seq_4:$	P	F	V	A	G	P	L	G
$Seq_5:$	R	F	V	A	G	P	L	A
$Seq_6:$	P	F	V	A	G	P	L	G

Table 2.2: Coevolution in pairs of sites: (i,n) and (j,m).

To clarify the concept of coevolution, we can see the table 2.2 that there exist correlations between sites i and n, and j and m. We can see how a mutation at site i in Seq_3 and seq_5 mutate from amino acid P to R, and force also a mutation in site n for those sequences, mutating from amino acid G to A. In the site j the mutation produces a correlated mutation at site m, such that every time an amino acid in the site j mutates, in this case from E to F, the correlated site m mutates from L to G.

Coevolution appears when some amino acids are related through evolution because mutations in some sites can cause evolutionary pressures to select compensatory mutations in other sites that are functionally or structurally related.

Prediction of coevolution in protein sequences has been studied in many works as [GOP07, WP05, NGR10, FHWE04, FT06]. Some of them work directly on the phylogenetic analysis, instead of on the alignment itself. Phylogenetic techniques give an idea about the ancestral relation between the sequences, but this approach of detecting coevolution based on the phylogenetic trees has the problem of distinguishing between what is properly coevolution and what is just the inheritance of conserved amino acids that differ in different branches of the phylogenetic tree.

Most alignment algorithms use the mutation matrix probabilities and scoring functions to align the sites of the sequences. During this work, we examine how coevolution information is not taken into account at the moment of the alignment and in many cases losing this information in the final alignment.

In this work we address the calculation of the coevolution using the approach of calculating the mutual information for every pair of sites. We estimate the correlation of sites based on the statistical information in the sequences in a multiple sequences alignment and we consider these sites as the guide sites to detect mistakes in the alignments.

2.5 Simulated data sets

Since we need protein sequences for which we know the correct alignment and the coevolved sites to do our experiments with, we developed a protein sequence simulator.

Following the previous definition of a protein sequence as an ordered list of amino acids, we constructed an application to generate these sequences and provide an evolutionary engine simulator to imitate as best as possible the evolution of sequences in nature.

We implemented the pairs of coevolving sites as described by Pollock et al. [WP05]. Every pair is in the form *(A,B)*, where A, B are sites and A is the driver and B the driven. The concept of driver and driven can be easy defined such that for every state on the driver, there is a favored state in the driven that will pressure the transformation process.

For the representation of the sequence of amino acids we used the List Prolog constructor. Each position in the list corresponds to a site, and each element of the list will correspond to a pair (Amino,Pos), where Amino is one of the possible amino acids, and Pos corresponds to an identifier of the site. In the beginning of the simulation, Pos has the value of the position, the site, of the amino acid. But practically this identifier is maintained for future work on the sequences, specifically in the alignment step and while dealing with insertions.

A protein sequence will be a triplet *(ID,List,Coevolved)*, where ID represents the sequence identifier, List represents the list of pairs *(Amino, Pos)* in the sequence, and Coevolved correspond to a list of pairs *(P1,P2)*, such that for this given sequence the sites P1, P2 are coevolved. We assume that every site in List that doesn't belong to any pair in Coevolved, corresponds to an independent site.

The evolution process of the simulator is an iterative transformation of the lists. In the beginning we create one sequence, the root sequence, and then, from this one, we create two offspring through a transformation step. In each step of the iteration, one transformation - as it will be explained later, mutation, insertion or deletion - is applied to each of the two offspring that will replace the parent sequence.

To generate the first sequence, we compute a random assignment of amino acids for each site, following a probability distribution corresponding to the relative frequency of each amino acid, data which is easily available [MB99].

A List of sequences will represent the complete offspring in a certain generation. For each step, to create the next generation, we transform each sequence on the list replacing it by their two offspring. Each generation containing N sequences will generate in an iteration a population of 2*N sequences.

The insertion in a given position N, corresponds to the insertion of a new amino acid in that place. For this, we shift all the sites, after the N position, to the right on the sequence. We use uniform distribution probability for the selection of the new amino acid to be inserted.

The insertion can cause an incompatibility between the coevolved sites, because now the sequence has a different size and the amino acids were shifted. So, we should normalize the coevolved sites as explained:

- Ff N<=A, for every pair (A,B) in the coevolved list for this sequence add 1 to A and B.

- If N>A, N<=B, for every pair (A,B) in the coevolved list add 1 to B.

- Do nothing otherwise.

The deletion in a given position N, corresponds to a deletion of the amino acid in the site N, so it is a renaming of the amino acid in such position by a '-' character (indel). If the deletion will be done on a coevolved site, it means there exists a pair (A,B) in the coevolved site list of the actual sequence and A=N, or B=N, then the deletion will not have any effect on the sequence.

We simulate the mutation used in the evolution along the offspring with changes in random sites following a BLOSUM62 matrix of transition probabilities [HH92]. BLOSUM62 transition probabilities is a matrix that contains for row I, column J, the probability of I mutating to J.

We handle the mutation of a coevolved site for a given sequence like other researchers do in [GOP07]. As we know, the coevolved sites are pairs (A,B) such that A is the driver, B is the driven. This means that the driver site will press on the mutation of the dependent site, and this pressure will correspond to a selection of favored state in the driven given the state of the driver, with a certain level of pressure corresponding to a coevolution factor.

We implemented the coevolution method in three different ways. In the Most Probable Cluster method, as specified in the work [GOP07], the amino acids were grouped into 7 different clusters, corresponding to the evolutionarily most meaningful division of them. Each site will evolve according to BLOSUM62 matrix index with the actual amino in the site to mutate (the first element in the pairs of blosum buckets), but the choice won't be done over all the possible mutations for this amino acid, but only for those which are in the same cluster as the driver amino acid. The clusters were obtained using a non-hierarchical clustering method, k-means, on the matrix BLOSUM62 and yielding the evolutionary most meaningful division of amino acids. The next method, Blosum Coevolution, will be a mutation guided by the Blosum matrix. The idea, is to get from the Blosum distributions the mutation correspondent to the driver and also the new amino acid distribution that will pressure on the correspondent coevolved site mutation. The third one is just a strict mapping, from the most probable evolution of one amino acid to other one, purely experimental for our future work but that we construct from the probabilistic mutation matrix adding biological meaning.

As the sequences are generated by simulation, we can postulate that there exists a way of doing a correct alignment based on evolutionary relations, using the information stored during the simulation. This correct alignment is used during this work as a benchmark for testing the results of other alignment methods and detect some properties on this result that will conduct our research during the whole work.

2.6 Constraint Programming

A *constraint* can be seen as a logical relation among many entities, each of them taking some values from their specific domain. The constraints are responsible for restricting the values the entities can take. One of the flexibilities of the constraints is their heterogeneous application to different domains for different type of entities, restricting and relating many domains of a problem.

In the last years, Constraint Programming has been a really focused research area in Computer Science and other fields, because it works in a declarative way, providing tools for specifying the relationship that should be maintained in the system, through relation over the variables that take values from their domain, without specifying the computational process through which the relations are assured.

In the same way as we use constraints daily, for instance when we are reasoning or planning: "I will take the metro. In Cais do Sodre station I have to leave before 1pm, and then I have to take the boat before 1.30pm", Constraint Programming is a tool for integrate constraint relating objects in computational systems, assign values to these objects according to the rules that the constraints define and a search for a respective solution.

2.6.1 Constraint Satisfaction

A constraint satisfaction problem (CSP) can be defined as:

- a set of variables $X = \{x_1, x_2, ..., x_n\}$,
- each variable is associated with a domain, so for each x_i, D_i is a finite set of possible values x_i can take. Depending the framework, the domain of each variable can be of one of many types as *integer,boolean,range,etc.*, and
- a set of constraints, C, logical relations, that restrict the values the variables can simultaneously take.

In this definition a constraint can be defined as a subset of the cartesian product of the domain of the variables that are involved in that constraint.

A solution to this CSP, is an assignment of a value from its domain to every variable in X, always such that this assignment satisfies all the constraints belonging to set C in the CSP. We can wish to find all the solutions, any of them, or may be just one with an optimal condition, or at least one that satisfies an objective function over all or some variables.

The solution can be found by searching systematically, traversing all the possible assignments of values in the domain to the correspondent variable. Since this is not the most efficient way of obtaining a solution, many techniques were proposed during the last years, as many variants of backtracking, consistency techniques, constraint propagation and optimizations among others.

The classic CSP as described above, defines the satisfaction of each constraint as a necessary condition to the solution of the CSP. This restriction means that each constraint is imperative, each solution satisfies them, and in a total assignment for all the variables, they must be completely satisfied. But these strict assumptions can be relaxed, as in Flexible CSPs, allowing solutions that do not satisfy all the constraints but only a subset of all constraints.

In this work we work with *Max-CSP*, where some constraints are allowed to be violated, and the solution is searched through a qualification over the amount of constraints satisfied. Also *Weigthed-CSP* is considered in this work, a *Max-CSP* where the violations of the constraints are weighted according to some criteria and those with more weight are the ones preferred to satisfy.

2.6.1.1 Consistency techniques

Consistency techniques are approaches to speed up the solution of a CSP, based on the principle of removing inconsistent values from the variables' domains, reducing the search for consistent labellings. The inconsistencies can appear in the labelling of a variable because the value assigned is inconsistent with such variable, or with the logical relation over the variables, or with some more complex connection.

This techniques are deterministically computations through which values are removed from the domain of the variables, and any non-deterministic computation is done once there is no more propagation to do with the consistency techniques.

We will define the different techniques based on unary and binary constraints, due to the fact that n-ary constraints can be transformed to equivalent binary CSP [BvB98]. So, given the CSP we can define:

- Node-Consistency: It removes values from the variables' domain that are inconsistent with unary constraints defined over such variables. So, for every value $d \in D_i$, such that X_i taking the value d make unsatisfiable an unary constraint U, $D_i = D_i - \{d\}$. For instance a constraint *positive(X)* that states that values in the domain that are below 0 are removed.

- Arc-Consistency: Given a binary constraint arc(X_i,X_j), is arc-consistent if and only if for every value v_i in the current domain of X_i which satisfy the unary constraints on X_i, there is some value, called a support for v_i, in the domain of X_j, such that $X_i = v_i$ and $X_j = v_j$ is not unsatisfying the constraint *arc*. The idea of this technique is that if arc-consistency is entailed by the binary constraint, then the values of all variables' domains that have not support on other variables related by the constraint can be removed from their respective domain. As an example, we can see figure 2.1 where the nodes represent the variables, labeled with the domains, and the edges are the constraints and we can see the steps for enforce the arc-consistency.

Several arc-consistency algorithms were proposed as in [BR97], from AC-1 to AC-7. They are all based on the same principle, repeated revisions of the arcs, until a fixed point is reached with a consistent state, or some domain become empty making the constraint unsatisfiable.

Arc-consistency removes many inconsistencies from the constraint graph, but it doesn't mean that if all the constraint in the CSP are arc-consistent, there exist a solution for such CSP. The idea behind arc-consistency is the reduction of the domains on those values that will never be part of a solution. This can be seen in the example of the figure 2.1, where arc-consistency is achieved but there is no solution for this CSP.

- Path-consistency: Beyond the values removed by node-consistency and arc-consistency, more values can be remove from the variables' domain with path-consistency. It requires for every pair of values, of two variables X_i, X_j, satisfying the respective binary constraint, that should exist some value for each variable along a path from X_i to X_j, such that all binary constraint in the path are satisfied.

  There are many algorithms proposed to enforce path-consistency, like revisited in [Sin95], but in many situations dependent of the problem, the computational cost to enforce this consistency is not justified against the gain in the speed of the search for a solution.

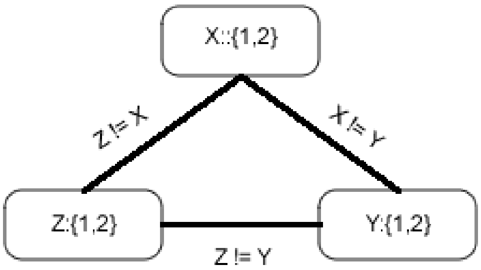


Figure 2.1: Arc consistent graph, but no solution.

Most consistency techniques are not complete, so there is no completeness boundaries to increase the efficiency of the algorithms losing some power in the value removal. Some cases of directional arc-consistency, for instance, require that the variables be ordered for working with efficiency and efficacy for constraints that previously enforced this criteria.

2.6.1.2 Constraint Propagation

The consistency techniques explained before, are not always sufficient for finding a solution or proving any CSP unsatisfiable. Instead these are methods that use each constraint or set of constraints to satisfy a local consistency between the variables they are associated with. Instead of using isolated techniques, a combination of local consistency and systematic search is in general used to search for the solution.

There exist many variations of the backtracking algorithms, as *backmarking* [Pro95] that reduce the number of redundant checks by remembering previous incompatible labellings.

Also there are many variations of look ahead approaches, methods that prevent future conflicts analyzing the actual status and checking possible new states. As explained in [RBW$^+$] *Forward Checking*, and *Partial Look Ahead* are the most common look ahead techniques and they combine arc-consistency, or directed arc-consistency, during the search enforcing that there exist possible values for future variables to be explored.

Many techniques for increasing the efficiency of constraint propagation are based on the search strategy, in which the order in which the variables are labeled is defined, i.e. a deterministic way is imposed. Also, the domain of the variables can be explored by the labeler in different ways, according to the problem, and extracting the values to assign with a specific criteria, as the maximum value, minimal value, etc. These techniques are really dependent on the problem, and how its modeled with constraints, but they add some efficiency when the solver is searching for a solution.

2.6.2 Constraint Optimization

The goal behind constraint optimization, is not just to find a solution but to find a good solution, measuring its quality by an application dependent function called *objective function*. The solution is searched satisfying the constraints, and also minimizing or maximizing the objective function.

The common technique for this kind of optimization is branch and bound [RBW$^+$], which works over a heuristic function that maps the partial labeling to a representative value. It is an under estimation (in case of minimization) of the objective function for the best solution (complete labelling) that can be obtained from the actual partial labelling. The algorithm proceeds reducing the space of possible solutions, and proceed in a deep search on them. As soon as a value is assigned to a variable, the value of the objective function is calculated for the new labelling, and if its value exceeds the bound, then this subtree is pruned immediately. The bound is started as a big number (in case of minimization), and stores the objective function value for the best solution found so far. So the main idea is to discard all branches that, from the partial state, we know will not generate a solution better than one obtained before.

The efficiency of branch and bound is based on the quality of the objective function, and also on whether a good bound is found early in the search process.

2.6.3 Constraints

There are many type of constraints, and in general, they are classified by the variables they relate. As explained in the node-consistency, the unary constraint relate one variable. Binary constraints relate pairs of variables. Many binary constraint are already define with their respective consistency and propagation techniques and in this work mainly we use:

- eq: $x = y$. Equality constraint, that forces the two variables to take the same value. Enforce Arc-Consistency.

- lt: $x < y$. Strictly less than, forcing variable x to take a value that is strictly smaller than y. Enforce Arc-Consistency.

.

Global constraints is the class of constraints that relate more than two variables. In general these constraints enforce arc-consistency, and path-consistency could be enforced losing some efficiency. In this work we make reference to three arc-consistent global constraints:

- $allDifferent(X_1,...,X_n)$: states that the arguments have pairwise distinct values: $X_1 \neq X_j;\ \forall i \neq j$, This version of the global constraint, is equivalent to pairwise comparison but considering all variables at same time, giving rise to better propagation, with specific efficient algorithms. Further in this work we analyze this constraint for future research in Chapter 7.

- $globalCardinality(\overrightarrow{X};low;up)$ states bounds on the occurrence numbers of any value v in the vector of variables X (here,offset min is the minimum value over all variables in x) : $low[v-min] \leq |\{i|X_i = v\}| \leq up[v-min], \ \forall value \ \ v$. low and up are the lower and upper occurrence value arrays for each possible value of the variables in the vector X.

- *increasingNValue*$(X_1,...,X_n,N)$: states that the variables $X_1,..X_n$ take increasing values, and moreover it forces N to be the number of different values taken by the variables. So, if we impose N equal to n, then $X_1 < X_2,..,< X_3 < X_n$. Otherwise, some consecutive variables can take the same values.

Finally, many user defined constraints can be created using the previously classification, depending on the framework for the constraint programming. In this work, the framework permits to create expression over variables, as boolean or arithmetic expressions, structures, and also relate them through predefined and user defined constraints.

3

Objectives

Our objective is to conceive and implement techniques, integrated as a single tool, to help in the identification of coevolved sites in Multiple Sequence Alignments (MSA), improving the detection of these sites and the correction of errors in the MSA, in those sites that lost the coevolution correlation.

We start by showing that the alignment algorithm produces mistakes on the coevolved sites. To do so we need a framework for measuring the information of sites that a priori we know have coevolved in multiple sequences to which we apply a standar MSA tool(Clustal). Due to the size and number of sequences and the lack of a tool to visualize correlation values calculated from the alignment, we need to develop a framework to systematically analyze where and how the mistakes occur to proceed to achieve our goals.

We want to detect this lost coevolution information as consequence of the alignment process. To do this, a protein sequence simulator was developed, which can create a data set with different types of mutations and coevolution schemes, based on the ideas proposed in [GOP07], and it can also indicate the correct alignment for the sequences generated.

Once we can pinpoint the impact of misalignments in the Mutual Information calculation, we estimate that there is a need to fix mistakes in the multiple sequence alignment due to the difficulty of aligning variable regions. Many applications, in particular Clutalx which we use during this work, show an average of 40-60% of accuracy when aligning [TPP99], and most of these mistakes occur in variable regions, precisely those that contain coevolution information, since conserved regions cannot show evidence of coevolution.

Based on our tool for detection of possible coevolved sites using a statistical method, we want a better approach to distinguish those sites that have really coevolved from those that have not but are detected as possible candidates due to correlation that arose by chance or by misalignment produced by the MSA algorithm during the alignment. Then, we need a constraint framework that give us the flexibility to set some relations among the amino acids in the sequences, adding a dynamic method to define our criteria of alignment. This framework can be our tool to test techniques for the correction of misalignments in variable regions that have coevolved.

Our objective is to find a way to correct the misalignment that hide the coevolution in the original sequences. With this tool, we want to increase performance in the detection of coevolved sites in multiple sequences alignments with good computational efficiency, and also produce a better alignment for these regions.

4

Mutual Information in Sequence Alignments

In this chapter we cover the concept of Mutual Information. We use it specifically to detect apparently coevolved sites, due to the property that indicates coevolution: correlation. Since we are searching for coevolved sites in multiple sequence alignment, mutual information is a good measure to realize this task.

During this chapter, we will explain how this measure closely relates the information in each site to the coevolution with other sites. Given the difficulties raised bythe errors in the alignment of variable regions, we introduce in this chapter also some techniques to overcome these drawbacks and we show their benefits.

This measure is of central importance for the work described in the next chapters, which focuses on the correction of misalignments to improve the mutual information between sites that coevolved but which lost correaltion during the alignment process. We do not test the result of each type of detection over the simulated data set, because during the next chapters we do not work over the mutual information values themselves. Rather, we use them as a tool for filtering some sites. Anyway, many tests of the normalizations were proposed in [GOP07] and we invite the reader to analyze those interesting results.

4.1 Mutual Information Definition

Mutual Information measures the amount of information related between one random variable to another one. It appeared first as part of Shannon's work [Sha01] and has become an important tool inside the Information Theory field.

Mutual Information, over two random variables *I,J*, *MI(I;J)*, measures in some way the information that the two variables share. It is an uncertainty measure between what we know about one variable and the information that is in the other.

If I and J are two independent random variables then our measure of mutual information is zero, because even if we know one of them, we do not have any information about the other. If both variables are identical, and so always take the same values, our mutual information measure is the uncertainty of one of them, because knowing I we know J and vice-versa. This, knowing one will give us the information about the other that is necessary to reduce the uncertainty to zero. As we will see below, the uncertainty of each random variable is measured by the entropy, so in this case the mutual information between X and Y will be determined by the entropy of any of these variables.

$$MI(I,J) = \sum_{a\in I}\sum_{b\in J} P_{IJ}(a,b).\log(\frac{P_{IJ}(a,b)}{P_I(a)P_J(b)}) \tag{4.1}$$

This equation is equivalent to one based on Shannon Entropy concept, and is part of the work in [Sha01]

$$MI(I,J) = E(I) + E(J) - E(I,J) \tag{4.2}$$

Where *E(I),E(J)* refers to the entropy of site I and J respectively, and *E(I,J)* the join entropy of *I* and *J*, and what formulated in terms of entropy is,

$$MI(I,J) = -\sum_{a\in I} p_I(a)log(p_I(a)) - \sum_{b\in J} p_J(b)log(p_J(b)) + \sum_{a,b\in I,J} p_{I,J}(a,b)log(p_{I,J}(a,b)) \tag{4.3}$$

In this equation 4.3, we can see the interpretation under the entropy concept. Intuitively we can say that the mutual information is relating the amount of uncertainty of knowing independently I and J, and the uncertainty in the pair I,J.

4.2 MI Calculation

In our work, mutual information is used to compare two sites in the multiple sequence alignments, and measure the related information between these sites. It means that two sites are correlated with a high mutual information value, if the existence of one specific amino acid in a site determines the existence of another specific amino acid in the other site, which suggests a functional or structural relation between the sites.

This measure of MI can give us a clue about which sites coevolved in a set of sequences that are evolutionary related. One mutation in one site will select for a compensatory mutation in other site and this correlation will be kept in the lineage of the sequences.

Intuitively we can calculate the probability of each amino acid occurring in a site *I*, as the frequency of such amino acid in *I* over the total amount of amino acids in *I* (equivalent to the total amount of sequences).

What we are trying to do in this work, is to detect those sites in the multiple sequence alignment that indicate a possible coevolution, and for this we do the $\frac{N(N-1)}{2}$ calculations of mutual information between the sites I,J.

Algorithm 1 MI Calculation

```
$Freq \leftarrow empty$
$Freq_{ij} \leftarrow empty$        ▷ Initialization of the HashMap to count amino acid and pairs
for $sitei = 0..LongSequence$ do
    for $j = 0..NumberSequences$ do
        $Freq(i) \leftarrow Amino[i][j]$    ▷ Freq count the aminos in sites i. Amino[i][j] is the amino of the sequence i, in position j.
    end for
end for
for $sitei = 0..LongSequence$ do
    for $j = i+1..LongSequences$ do
        for $k = 0..NumberSequences$ do
            $Freq(i,j) \leftarrow (Amino[i][k], Amino[j][k])$ ▷ Freq(i,j) count the pair of aminos in sites i,j
        end for
    end for
end for
for $sitei = 0..Longsequence$ do
    for $sitej = i+1...LongSequence$ do
        $MI[i][j] \leftarrow MI(i,j)$
    end for
end for
```

As we can see in algorithm 1, for the computation of this information over the sequences, we use a MultiHash structure to keep the amount of each amino acids in each site, and other one to keep the amount of amino acids pairs in each pair of sites. First, we traverse all the sites and for each sequence we extract the amino acid on such position, adding it to the structure. We do the same for all the pairs i,j of the $\frac{n(n-1)}{2}$ combinations of different pairs of sites.

With this mutual information matrix, we can guess the coevolved sites over the evolutionary lineage just by taking the values that are over a threshold T. To use of this information in the analysis of mutual sequence alignments we also developed a visualization tool that allows the matrix representation to be displayed on the screen with a scale of colors representing the amount of MI between each pair of sites and also a correspondence between each cell in the matrix and the sequence alignment result.

As we show in the section 4.5, the threshold T can be estimated after some experiments with simulated coevolved sites and analyzing the results of the algorithms. Moreover, we use several normalizations from [GOP07, NGR10] to tune the mutual information equation 4.1 and get better results in guessing coevolved pairs.

4.3 Normalization of the MI

To detect correlation on sites using the mutual information calculation we have to use some techniques to normalize equation 4.1. For instance, the wide range of possible value for the number of different aminoacids in a site, the logarithmic base we use in the formula, or the calculation of correlation for totally conserved sites, can produce mistakes of good possible candidates of coevolved sites when they are not.

Some solutions to the normalization problem has been proposed in [GOP07]. In the next sections we will explain some models to normalize the values, and we will conduct an analysis of which of the models better solves our problem, showing results that justify our choice according to our calculations.

4.3.1 Adaptable logarithmic base

In the equation 4.1 we can postulate that the MI value will fall in the range from zero to one, if we can guarantee three conditions each time we calculate MI between two sites. First, that there is perfect co-variation in each site, i.e. that every amino in one site is co-related with other amino in the other site, and there never exists a pair that does not follow this correlation. Second, the 20 amino acids are present in both sites, condition needed because of the base used in the logarithm of the formula 4.3. Third, the 20 different amino acids are present in equal frequencies in both sites.

So, we have the formula that states the values of the MI to be conditioned to the three previously defined conditions. As [GOP07] argues, the only condition that is important for the coevolution detection is the first one, and the other two, even though when present in the formula the detection of coevolution is still working, they also decrease the quality of the detection of the coevolution if it is not normalized.

One solution to counter the second condition, is known as Mutual Information of Adaptable Logarithmic Base (MI Adp):

$$MI(I,J) = \frac{\sum_{a \in I} \sum_{b \in J} P_{IJ}(a,b).\log(\frac{P_{IJ}(a,b)}{P_I(a)P_J(b)})}{log(sqrt\#i.\#j)} \tag{4.4}$$

where $\#i$ represents the number of different amino acids occurring in site A and $\#j$ in site B.

With this formula, we can calculate the MI of two sites obtaining results ranging from zero to one, but independent on the number of different amino acids in each site.

4.3.2 Simple Correlation

With the previous method we can calculate the MI independently of the number of different amino acid in each site. Now, to add independence of the third condition, in order to calculate Mutual Information without influence of the amino acid distribution in the correspondent sites, we apply the following approach.

The influence of the amino acid distribution is due to the entropy calculation, using the logarithm. As explained in [GOP07], the logarithm equation:

$$-\sum_{i}^{n}(xlog(x)) \tag{4.5}$$

has its maximum at $x = 1/n$ with $0 < x < 1$. In terms of amino acids and probabilities it has its maximum when they follow a uniform distribution on the site.

We can use a logarithm independent formula:

$$MI(I,J) = \sum_{a \in I} \sum_{b \in J} \frac{P_{IJ}(a,b)^2}{P_I(a)P_J(b)} \tag{4.6}$$

This measure has the property of being only dependent on the correlation of the sites. But it is not perfect at all because it has the drawback of calculating as correlated sites many that are strictly conserved (not mutating during the evolution). Conservation of the sites I and J, in terms of entropy means low values for *E(I)* and *E(J)*, we can't say too much about the uncertainty of the sites. Thus, if we use this measure to detect coevolved sites, we could detect all the coevolved sites plus all the conserved site pairs.

We want to discard the conserved sites, because as we proposed in the coevolution definition, conserved sites provide no evidence that mutations at one site selected for particular mutations at the other.

The solution proposed in [GOP07] is just to do the whole calculation of MI for every pair of sites, and after that check the existence of conserved sites to which the MI value is modified to zero.

4.3.3 Entropy division

We seek to remove the noise from the first formula 4.1, keeping the scores of coevolution detection as high as possible and expanding the model to remove the false positives. In some of the methods stated before we have to increase computation time to remove false positives and get a better accuracy.

We can still have a better coevolution accuracy without increasing computation time. We can observe on equation 4.2, that coevolved sites will have similar entropy individually and also similar to the joint entropy. So, one solution is to normalize the formula, dividing the equation 4.1 by the maximum entropy in one of the two sites, obtaining:

$$MI(I,J) = \frac{E(I)+E(J)-E(I,J)}{max(E(I),E(J))} \tag{4.7}$$

This normalization works well when we know that the number of different amino acids in the sites is not large. Evolutionarily, it means that the amino acids in such sites did not mutate many times to different amino acids, but instead they kept within a small group of amino acids. As proposed in the literature [GOP07, FHWE04, FT06], if we are doing a multiple sequence alignment with evolved sequences, this is the behavior we expect to find for coevolved sites.

When one site mutates at a higher rate than the other and changes to different amino acids, it has a higher entropy. Thus, the division by the maximum entropy of the two correspondent sites will maximize the MI when the sites have similar entropy and are correlated, and will decrease the MI when they have different entropy values. This method adds some noise when the entropies differ in an almost related coevolved site. We show in section 4.5 how this measure is used to classify pair of sites as possibly coevolved ones through a highest scores selection.

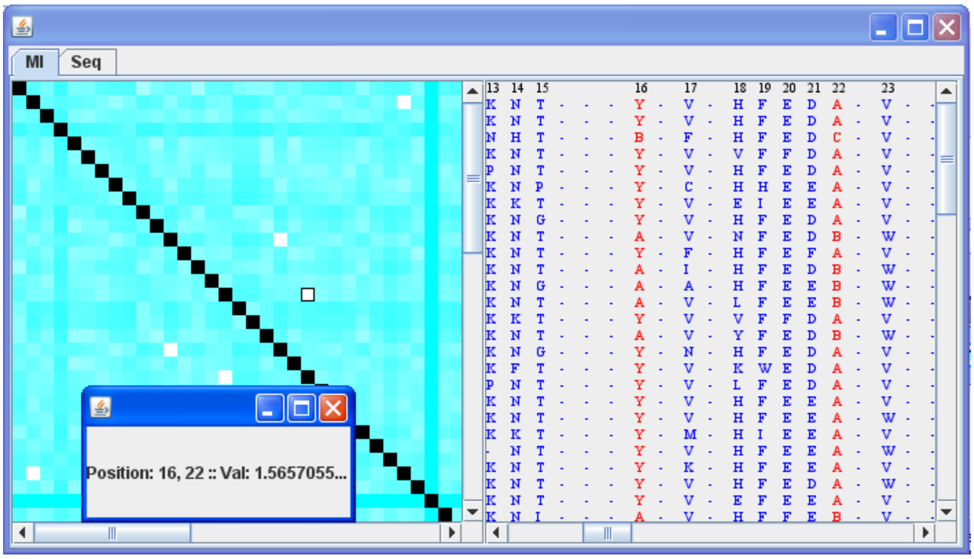
In terms of computation this normalization is cheap, because we need also the entropy for the MI's computation. Indeed, the entropy calculation for a site is needed when we want to know if the site is fully conserved because no mutations in a site result in zero entropy.

4.4 Visualization tool

Once we apply the Clustal alignment and we calculate the MI for all the sites in the MSA, we need a tool to visualize the MI score and the sequences alignment.

This motivated us to develop a visualizer during this work, to pinpoint characteristics in the misalignments over which we drive our work in next chapters.

As we show in figure 4.1, the tool shows the pair of sites in a symmetric matrix representation, where each *(i,j)* cell correspond to the MI value for the sites *i* and *j*. We calculate for any MSA, the $\frac{n*(n-1)}{2}$ different combination of sites *(i,j)* and we store in the matrix the MI(i,j) with any of the methods presented above. In the visualization we can see a value scale, from 0 to the maximum value the MI can take (according to the method). Also the tool provides a direct visualization of the sequences, and if one pair *(i,j)* is selected in the MI matrix, then in the sequences visualization these sites are also highlighted for fast identification.



Position: 16, 22 :: Val: 1.5657055...

```
13 14 15          16    17    18 19 20 21 22    23
K  N  T  -  -  -  Y  -  V  -  H  F  E  D  A  -  V  -  -
K  N  T  -  -  -  Y  -  V  -  H  F  E  D  A  -  V  -  -
N  H  T  -  -  -  B  -  F  -  H  F  E  D  C  -  V  -  -
K  N  T  -  -  -  Y  -  V  -  V  F  F  D  A  -  V  -  -
P  N  T  -  -  -  Y  -  V  -  H  F  E  D  A  -  V  -  -
K  N  P  -  -  -  Y  -  C  -  H  H  E  E  A  -  V  -  -
K  K  T  -  -  -  Y  -  V  -  E  I  E  E  A  -  V  -  -
K  N  G  -  -  -  Y  -  V  -  H  F  E  D  A  -  V  -  -
K  N  T  -  -  -  A  -  V  -  N  F  E  D  B  -  W  -  -
K  N  T  -  -  -  Y  -  F  -  H  F  E  F  A  -  V  -  -
K  N  T  -  -  -  A  -  I  -  H  F  E  D  B  -  W  -  -
K  N  G  -  -  -  A  -  A  -  H  F  E  E  B  -  W  -  -
K  N  T  -  -  -  A  -  V  -  L  F  E  E  B  -  W  -  -
K  K  T  -  -  -  Y  -  V  -  V  F  F  D  A  -  V  -  -
K  N  T  -  -  -  A  -  V  -  Y  F  E  D  B  -  W  -  -
K  N  G  -  -  -  Y  -  N  -  H  F  E  D  A  -  V  -  -
K  F  T  -  -  -  Y  -  V  -  K  W  E  D  A  -  V  -  -
P  N  T  -  -  -  Y  -  V  -  L  F  E  D  A  -  V  -  -
K  N  T  -  -  -  Y  -  V  -  H  F  E  E  A  -  V  -  -
K  N  T  -  -  -  Y  -  V  -  H  F  E  E  A  -  W  -  -
K  K  T  -  -  -  Y  -  M  -  H  I  E  E  A  -  V  -  -
-  N  T  -  -  -  Y  -  V  -  H  F  E  E  A  -  W  -  -
K  N  T  -  -  -  Y  -  K  -  H  F  E  E  A  -  V  -  -
K  N  T  -  -  -  Y  -  V  -  H  F  E  D  A  -  W  -  -
K  N  T  -  -  -  Y  -  V  -  E  F  E  E  A  -  V  -  -
K  N  I  -  -  -  A  -  V  -  H  F  F  E  B  -  V  -  -
```

Figure 4.1: Visualizer tool, Mutual Information and Alignment

4.5 Results

In this section we test the mutual information detection with multiple sequence alignments. We want to detect if there are mistakes in the coevolved sites as a consequence of the multiple sequence alignment tool. In this case, we generate random sequences with our simulator. We can see the description of these sequences in the table 4.1. The *Test Nr* correspond to the identifier of each test (each test is an average of 5 runs under same conditions.) Generations indicate number of iterations for the simulation, and *AVGsimilarity* the average similarity score correspondent to the Clustal pairwise score average, which represent the average sequences homology for each test. The generation was done with the coevolution method mapping, as described in Section 2.5. In each sequence we have 20 pairs of coevolved sites, each sequence is 100 amino acids long and we keep 70 sequences, randomly selected from the last generation of the simulation.

Test Nr	Generations	AVGsimilarity(%)
1	18	79
2	20	73
3	25	67
4	30	52
5	35	41

Table 4.1: Average Pairwise homology score given by Clustal.

We proceed by calculating the Mutual Information for all the pair of sites, in the correct alignment and in the Clustal alignment. Then for each pair of coevolved site in the test, we compare the MI score marking a match if in both alignments the score is the same and as a mismatch otherwise. Of course, in the correct alignment we get the original information, so if the sites are coevolved and this is reflected in the mutations, the site has a large MI score. If the sites are never mutated, and just conserved, the MI score is really low event if they were marked as coevolving sites in the simulation. But in this experiment we do not pay attention to the MI score itself but only on the differences between correct alignment and the Clustal alignment for the coevolved sites.

We can see in the figure 4.2 a graph with the respective average of the tests. It is clear that many misalignments occur in the Clustal on coevolved sites. If we see the results, even for homologous sequences the number of errors is large. This shows that Clustal is making many mistakes, and that some correction is needed. In the test with fewer generations, it is more likely that coevolving sites never mutate, i.e. they remain conserved along all the simulated evolution, or mutate very few times, producing a match in the result because Clustal tends to align conserved regions correctly. When we increase the number of generations, the probability of a mutation occurring in a coevolved pair of sites increases, producing less similar sequences and increasing the number of misalignments. The reader can also refer to the results in Section 6.2.2, where many tests over the Clustal alignment show a high number of False Positives

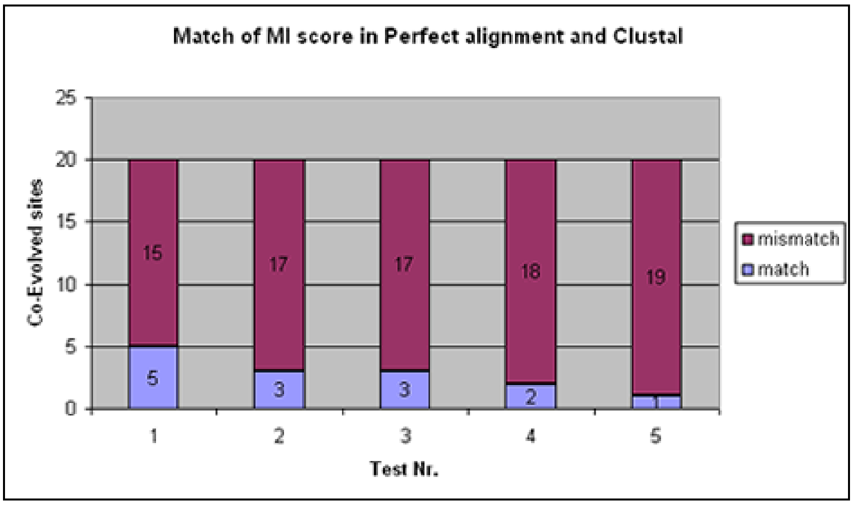


Figure 4.2: Ratio match, mismatch of coevolved sites in Correct and Clustal alignment.

and False Negatives for sites detected as coevolved sites, showing also that many mistakes are done over the alignment in such sites, and that the statistical method for coevolution detection are too sensitive to such noise to be used for detecting coevolution without further corrections.

With these results we can claim that misalignments appear on the coevolved sites when the multiple sequence alignment is done, if the coevolving pairs mutate significantly during evolution. In the next chapters we compare methods for deciding if a pair of sites are coevolved, and we show why a more accurate method than the MI alone is necessary, and how we can correct the alignment mistakes in the coevolved sites.

5

Alignment using Constraints

In this chapter, we propose two models that can be used to align sequences with imposed constraints. These alignments are based on desired criteria, for instance, for every sequence each amino acid in a site should match, if possible, the same amino acid in the next sequence at the same site. Moreover, we establish an alignment framework to work with on the next chapter.

We explore the use of constraints to define possible situations on the alignment and discard unsatisfied states in the search for a solution. We also identify the problems each model has in the alignment process, extending some features to avoid these obstacles. Since many constraints can be unsatisfied due to conflict with other constraints, we adapt the models as MaxCSP problems obtaining a system that can handle violations on the constraints, i.e. in the alignment specifications, getting the best possible alignment corresponding to the maximum amount of satisfied constraints.

We also present some experiments with multiple sequences and we evaluate the models introduced in this chapter with the goal of detecting the drawbacks and advantages of each of them leading to our selection of a framework.

5.1 Motivation

We are not proposing a MSA algorithm, but a constraint based alignment over sequences as a tool for correcting alignment errors in a MSA. We need to propose a framework where we can align the sequences using constraints because with the existent algorithms for sequences alignment, we do not have the flexibility to relate the positions where coevolution occurs and force those positions to align not following the typical score matrix but a particular correlation instead. For instance, in Dynamic Programming MSA, the search for the best solution is done expanding the values in the table in an incremental manner, but if we find a state that conditions the change of some state previously calculated, we can not go back and then go forward again as often neded without paying a computational cost and without changing the algorithm to handle any specific optimization alignment.

Having a framework to set constraints in the alignment provides a flexible tool to experiment with the sequences adding dynamical properties that they have to maintain according to our desire. We need to add this flexibility because we want to deal with the correlations present in the coevolved sites.

5.2 Preliminary definitions

As we want to formalize the models and compare them, we need to establish a common language to talk about the amino acids, sequences, alignments, mismatches, and so forth. We discussed about many of these concepts in the chapter 2, but we need to organize them with more formal specific definitions.

Definition 1. *We call Σ to the alphabet, a finite set of characters. We can define also Σ_G, an extension such that $\Sigma_G = \Sigma \cup \{-\}$, where $-$ represents a gap in the alignment.*

In this work, and from now on, the alphabet corresponds to the set of amino acids because we work over protein sequences, but it can correspond, for instance, to the nucleotides in DNA sequencing.

Definition 2. *We define as* protein sequence s*, during this work also called just* sequence*, to a finite* string *or finite ordered list of elements over an alphabet Σ. More formally, $s = (\Sigma)^*$.*

Definition 3. *We define as* alignment sequence *s_A, to a finite* string *or finite ordered list of elements over an alphabet Σ_G. More formally, $s_A = (\Sigma_G)^*$. We define this as an extension of the protein sequences where gaps are added in order to align with other sequences.*

Definition 4. *Let* s *be a protein sequence, $s = a_1, a_2, ..., a_n$ where $\forall 0 < i < n,\ a_i \in \Sigma$. Let s_A be an alignment sequence, $s_A = b_1, b_2, ..., b_n$ where $0 < i < n,\ a_i \in \Sigma_G$ We can define the following operations over s, and over s_A:*

- *$s(i) = a_i$. With this operation we obtain the element of* s *at* ith *position, that belongs to Σ.*

- $s_A(i) = b_i$. *The* i-th *element for an alignment sequence, that belongs to* Σ_G.
- $lg : (\Sigma)^* \rightarrow \mathbb{N}$. *Given the protein sequence* s, lg(s) *calculate its length.*
- $lg : (\Sigma_G)^* \rightarrow \mathbb{N}$. *Given the alignment sequence* s_A*,* $lg(s_A)$ *calculate its length.*

Definition 5. *Given an alignment sequence* s_A*,* $s_A \in \Sigma_G^*$*, the projection of* s_A *w.r.t* Σ*, noted as* $s_A|_\Sigma$*, is defined as* s_A *with the deletion of all the occurrences of '-'.*

Definition 6. *Given two or more protein protein sequences,* s_1*,...,*s_n*, we can define a multiple sequence alignment as a* n-tuple *(*s_{1A}*,...,*s_{nA}*), such that:*

- $s_{1A}, ..., \ s_{nA} \ \in (\Sigma_G)^*$. *i.e., they are n-alignment sequences.*
- $s_{1A}|_\Sigma = s_1$*,...,* $s_{nA}|_\Sigma = s_n$
- $lg(s_{1A}) = ... = lg(s_{nA})$
- $\nexists i \ \ 0 < i < lg(s_{1A}) = ... = lg(s_{nA}) \ \ s.t. \ \ s_{1A}(i) = ... = s_{nA}(i) =' -'.$

Definition 7. *From now on, when we talk about multiple sequences (multiple sequence alignment), MS (MSA) of size* m*, we are making reference to a collection of m protein sequences (alignment sequences).* $MS = s_1, s_2, s_3, ..., s_m$

We have a formalization of the the multiple sequence alignment, that helps guide our description of the methods for alignments using constraints. Moreover, we need to formalize some concepts in each respective model that we present in this chapter, with a specific semantic.

Definition 8. *We can define a set of variables,* V*, whose elements are variables in the domain* $[0..N]$*. When we work on particular domain we specify the meaning of the value N.*

Definition 9. *A sequence of variables, or Variables, is defined as a succession of different variables of V. Formally:* $Variables = X_1, X_2, ..., X_n$ *such that,* $X_i \in V$.

Then, we need to map the multiple sequences onto a set of variables. We can do it in the following way:

Definition 10. *A mapping, Vars, from a protein sequence to a collection of variables, is defined as* $Vars : Sequence \rightarrow Variables$ *such that* $\forall i \ \ , 0 < i < n, \ \ if\, a_i \in Sequence$*, then* $\exists X_i \in Variables.$ *The idea is that for different sequences, we will have different set of variables, such that* $V = \bigcup_i Vi \ \ , \forall Vi \in Variables,$ *and* $\forall i \neq j \ \ , Vi \cap Vj = \emptyset$.

In other words, if we have a multiple sequence MS, we have as many sets of variables as sequences in the MS, and all the variables in all the sets are different. So, every variable represents univocally one amino acid in one sequence.

Actually what we define is a way of representing the elements of the sequences as variables over which we work in the next sections. In other words, we can have a mapping from the sequences to sequences of variables, as shown in the figure 5.1. This representation is useful for the next section where we give the correspondent semantic for such variables.

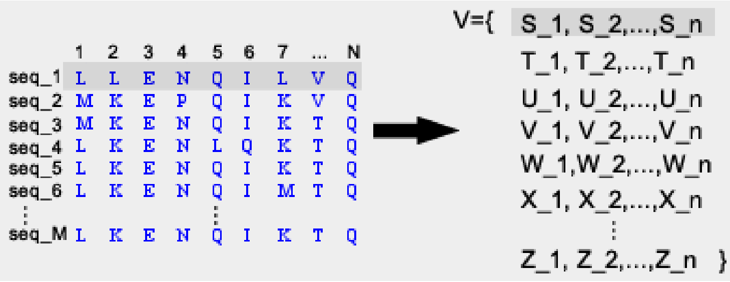


Figure 5.1: Variable based representation of sequences

5.3 Gaps Model

5.3.1 Semantic

The goal of this model is to assign the correspondent semantic to some elements defined above, with the objective of represent gaps between the amino acid of an alignment sequence and do an alignment based on constraint over such gaps.

First we need to add the semantic to the set of variables V. In this model we use the variables to represent the amount of gaps in a certain position. Given the set of variables, from now on, gap variables V with their respective domain *[0...N]*, we can represent gaps between any pair of amino acids a_{i-1} and a_i with the correspondent variable X_i, assigning values from the domain to this variable. N is a constant defining the maximum-length of gaps between consecutive amino acids. For the first element of the sequence, a_1, the meaning of the value of X_1 is the gap length in the beginning of the sequence, i.e. before the first amino acid.

To clarify these concepts, we can see again the figure 5.1, where the variable U_2 represents the gaps between the $1st$ and $2nd$ amino acids in the sequence seq_3, i.e. the gaps between M_1 and K_2.

Before we proceed with the definition of the constraints for the position of the elements in the sequence, we need to define the concept of position.

Definition 11. *We can define a position offset of an amino acid with respect to the beginning of a sequence* s*, as a function $Pos_s : \Sigma_G - > Expression$, i.e. a function with domain in all the possible amino acids and image in Expression, that corresponds to a constraint Expression, defined recursively as:*

- $Pos_s(a_1) = X_1$, $X_1 \in Vars(s)$, *i.e. a_i is the first amino acid in the sequences s and the equivalent expression is the first variable in Vars.*

- $Pos_s(a_j) = Pos_s(a_i) + X_j + 1$ *iff* $s(j) = a_j$, $s(i) = a_i$, $i < j$. *i.e., for any amino acid that is not the first one, we calculate the expression as the position of the previous amino acid plus the variable corresponding to this amino acid plus 1.*

Note that this function is defined only for the elements of the sequence s. The idea of this function is that, given an amino acid in a sequence, we can get an arithmetic expression that is based on some gaps variables and that locates an amino acids in a sequence, when the gaps variables take values. Any amino acid position in the sequence is in function of the gaps variables that identify such amino acid and the amount of previous variables and amino acids.

What motivates us to define this position function is the necessity to formalize the concept of the expressions, because this is what we want to constraint when we search for an alignment. When we want to align an amino acid with another located in other sequence, we can force their position expressions to be equal. As we explain later, we write all the alignment desires as a set of constraints.

Definition 12. *We define a Position Constraint, PC, over two amino acids at position* i *and* j *in two sequences respectively s_1 and s_2 as a constraint on the position of the amino acids. This Position Constraint relates with an operator the position of the two amino acids.*

$$PC(s_1(i), s_2(j)) : Pos_{s_1}(i) \quad = \quad Pos_{s_2}(j),$$

Now, we can use the amino acids in the multiple sequences to create *Position Constraints*, according to the alignment desired between any amino acids in any subsequence. Intuitively what we are doing with the Position Constraints is creating an expression that should be valid, relating the position of two amino acids, and forcing an assignment of natural numbers to the variables involved which corresponds to the gaps in different sites of the sequences.

Once we have defined all the constraints on the position variables corresponding to the desired alignment, we can solve the constraints over the variables' domains and get a solution for that alignment.

We can see graphically in Figure 5.2(a) some position constraints that we can require for an alignment. In this case we want to align the amino acid *P1* with *P2* or *P3*, and *P4* with *P2* or *P3*. We also show some possible alignments, in 5.2(b)(c), that satisfy our intended alignment defined by constraints. We can see the expressions obtained when applying $pos(P_i)$. In this figure we do not show how we solve the constraints nor the labelling process because these topics are explained further on.

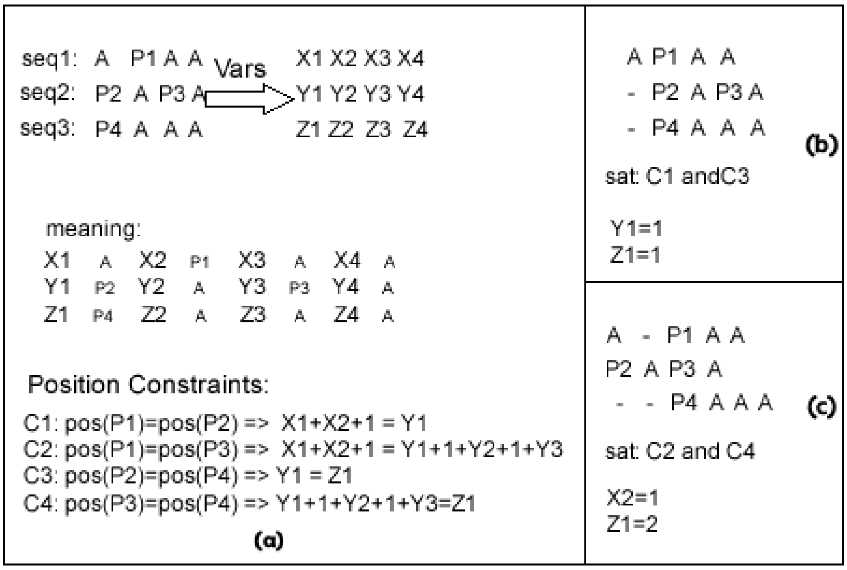


Figure 5.2: Gap Model: Position Constraint, variables and expressions.

To transform back the values of the variables into a multiple sequence alignment, we have to add the corresponding value for each variable to the position they represent. The main idea is that from protein sequences we create alignment sequences, i.e. the same protein sequences plus the gaps in the corresponding positions according to the value of the variables. To do so, we can define the following function:

Definition 13. *We define a function* $Transform : Sequence\ \ x\ \ V \rightarrow Alignment\ \ Sequence$*, such that:*

- $Transform(s,V) = s_A \quad s.t \quad s' = val(V_1)a_1...val(V_n)a_n\ \forall\ a_i \in s$*, where* V_i *is the variable associated with the amino acid* a_i*, and* $val(V_i)$ *is k times '-', if k is the value that the solver gives to this variable.*

In other words, what the last definition does is a replacement of each variable value by the specific amount of gaps between the amino acids the variable represents.

In the next sections we show an intuitive sketch of proof about the correctness of the model, we show the method in which we set the constraints and we propose a test for the models.

5.3.2 Correctness of the model

When we set the Position Constraints, and we relate the expressions that represent the amino acid positions to force such amino acids to be aligned, we are setting conditions that we try to solve according to the variables' domain values assignment using a solver.

The variables values add to the aligned sequences a defined amount of gaps. After we apply the Transform function, we get a new alignment sequence with the same amino acids as the original one, but with gaps ('-') added between some amino acids, and as many as the specific value each variable takes after solving the constraints. So, there is no adding of new amino acids nor any transformation of the original ones, but just a shift of the amino acid with gaps insertion.

5.3.3 Aligning with the Gaps Model

In this section we propose a way to approximate an alignment using the Gaps Model. The goal is just to set some constraints and to test the performance of this algorithm to detect advantages or drawbacks of using it. We have the bases of the model, and now we want to obtain an alignment using a specific criteria and setting some constraints to achieve it.

This criteria for instance, can be measured after the alignment with respect to the final position of the amino acids in the sequence, determining a quality of the alignment. But we do not focus now on optimizing the alignment but just on testing the tool to fulfill the basis for comparing it with other methods and choose one to use in the next chapter.

The criteria we test for is a successive matching, where for every site j, each amino acid in a sequence k matches with the same amino acid in the sequence $k+1$ whenever possible. With the models we can apply other criteria, such as for instance, one following a substitution matrix [HH92] as explained in chapter 2, aligning each amino acid in a site with another amino acid, such that this match corresponds to a meaningful mutation.

The idea is to traverse all the sequences and to define constraints over the gap variables. In this method, we just align the amino acids in a sequence with those in the next sequence, trying to get the best alignment possible under such criteria, as we show in the algorithm 2 where we set all the constraints. Position is the function described previously that returns the respective expression with gaps variables for an amino acid in sequence *i* and site *j*.

Algorithm 2 Setting alignmnet constraints

```
for $i = 0..NroSubsequence - 1$ do
    for $j = 0..LengthSubsequences$ do
        for $k = 0..LengthSubsequences$ do
            if $aminoacid[i, j] = aminoacid[i+1, k]$ then
                $newConstraint(eq(Position(i, j), Position(i+1, k)))$
            end if
        end for
    end for
end for
```

Before we show the results of the experiment, we introduce other model in the next section and finally we compare them.

5.4 Position Model

5.4.1 Semantic

For this model, the mapping of the amino acids to the variables is the same as in the previous model as shown in figure 5.1, to which we give in this section a different semantic.

The semantic for the variables and the values they can take from the domain, now is defined as the position of the respective amino acid in a sequence, taking values from the domain [0..NP], where *NP* is a constant meaning the maximum new position in the aligned sequence. The way in that we define the maximum new position, *NP*, determines the possible gaps between the position of the amino acids.

In this model, the gaps are implicit in the variables values in the sense that if two consecutive variables do not take consecutive values (i.e. consecutive positions), the difference between them corresponds to the gap length between their respective amino acids. So, the difference to the first model is that in this model the variables take values representing the positions themselves, instead of gaps that determine the final positions of the amino acids. Thus, to define the position of an amino acid in this model, we just assign the desired value to the corresponding variable.

For instance, we can see the figure 5.3 with the respective constraints for that sequences and where we define $NP = N + N * 3$, where N is the length of the original sequences.

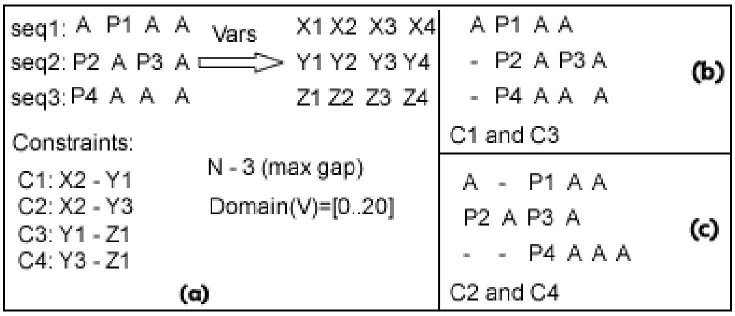


Figure 5.3: Position Model:Constraints, variables and expressions.

Under this model, we have to define constraints that restrict the order of the amino acids in a sequence. If we do not do this, we allow values in the variables that may switch the position of two amino acid in a sequence, a result that should be avoided. Also we set constraints to align, meaning that if an amino acid has to be aligned with another one in another sequence, we constraint both variables to take the same value over their domains.

The constraints to define the order in a sequence s, can be defined as:

$$X_i < X_{i+1} \;\; \forall 0 < i < n$$

where n is the length of s, and $X_i \in V_s$, the set of the variables for this sequence.

These constraints are necessary to maintain the semantic of the sequences. Formally, we should include this necessary constraint for all the sequences in any multiple sequence MS.

Once the sequences order constraints are defined, any other kind of constraint over the amino acids can be imposed. We use the equality binary constraint = (also *eq*) between two variables, X_i in a sequence and Y_j in another one, to force the value of both to be the same. The semantic of this is that the position of the amino acids that each of them represent in their respective sequences should be aligned, i.e. in the same site.

After the constraints are solved, the variables keep the position for the respective amino acids and we have to transform such assignment in new sequences. We can define a Transform function for a sequence s and a set of variables V as follow:

Definition 14. *Transform is a function, $Transform : Sequence\ x\ V -> AlignmentSequence$, such that:*

- $Transform(s,V) = s'\ \ s.t.\ \forall a_i \in s$, *then* $a_i \in s'$ *and* $s'(V_i) = a_i$, *and for all* $V_j,\ V_i, if\ \ j > i$

and $\nexists V_k$. s.t. $i < k < j$, then $V_j - V_i - 1$ is the amount of '-' between amino acids a_i, a_j. i.e. every amino acid in s appears also in s' in the position that the variable associated with it indicates, and for every consecutive amino acids the amount of gaps between them is the distance of their positions minus 1 (if their positions are adjacent then no gaps are added).

5.4.2 Correctness of the model

In this model we directly set the constraints as a relation between the variables that represent the amino acid positions. In this model we do not have any expression, as in the previous one, but just a variable whose value represents the position in the sequence.

The correctness of this model can be justified as the satisfiability of the ordering constraints for all the amino acids in the sequence, forcing them to keep their order, and that only gaps are added. Any sequence of variables that represent a sequence of amino acids takes values for such variables in incremental order. Any variable takes a larger value than a previous one in the sequence.

When we apply the Transform function, we obtain from the original sequence of amino acids and the set of variables, a new alignment sequence that preserves the order in the amino acids as the original one and also conserves the same amino acids without introducing any new amino acid. Only some gaps can be added between the amino acids, as many as $X_j - X_i - 1$, for any consequent amino acids a_i, a_j.

5.4.3 Aligning with the Position Model

A simple alignment in this model can be done asserting binary equality constraint between the variables of different sequences, enforcing two amino acids to take the same site.

In the figure 5.4, we can see a simple alignment of the Qs of the sites 5 and 6.

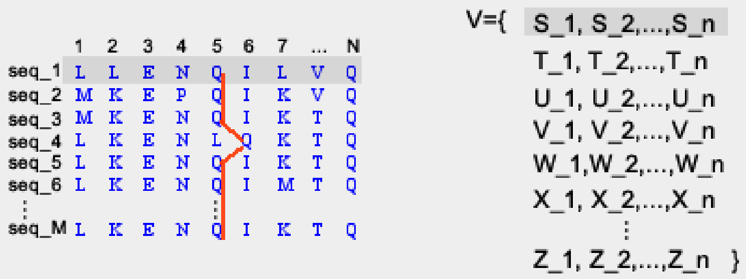


Figure 5.4: A simple alignment and order constraints

The constraints for the multiple sequences of the figure 5.4 can be generalized as:

- Fixed constraints for a sequence s: $\forall i, j 0 < i < n$, if $a_i, a_{i+1} \in s$, then $X_i < X_{i+1}$ is a constraint that should be satisfied.
- Alignment Constraints: $S_5 = T_5 = U_5 = V_6 = W_5 ... Z_5$ if we want to align all Qs with ohter Qs between sites 4 and 6.

To test the method, the general procedure to create the constraints is similar to the one presented in the previous model. We show in this part an equivalent algorithm that aligns an amino acid in a sequence *k*, with those equal amino acids in the sequence *k+1*.

In the algorithm 3, we show the creation of the alignment constraints. The variables are represented in the matrix *vars*, such that each row correspond to a sequence and each column to the sites, thus, $var[i][j]$ correspond to the position of the amino acid a_j in the sequence *i*.

Algorithm 3 Setting alignment position constraints

```
for $i = 0..NroSubsequence$ do
    for $j = 0..LengthSubsequences - 1$ do
        $newConstraint(vars[i,j] < vars[i,j+1]);$
    end for
end for
for $i = 0..NroSubsequence - 1$ do
    for $j = 0..LengthSubsequences$ do
        for $k = 0..LengthSubsequences$ do
            if $aminoacid[i,j] = aminoacid[i+1,k]$ then
                $newConstraint(vars[i,j] = vars[i+1,k])$
            end if
        end for
    end for
end for
```

5.5 Reified constraints

We have to extend the use of constraints, because in the models presented in the section 5.3 and section 5.4, we can easily detect that if two constraints are in conflict they can not be solved simultaneously and then the whole process fails without finding a solution. With those models we have not yet a robust solution to align multiple sequences because problems arise when two or more constraints are in conflict. In the alignment problem this means that there is no possible alignment, which must be solved as a search of the best feasible solution that allow an alignment.

We can think in the possible constraints present in the figure **??**, and how the conflicts can occur if not all constraints can be satisfied simultaneously. In the Gaps Model as well in the Position Model described before, we can not find any solution to this alignment.

seq1:	A	D
seq2:	D	A

Table 5.1: Conflict if we want to align A's, and B's

One solution is based on reified constraints and, before proceeding with it, we have to introduce the following definition:

Definition 15. *A Reified Constraint, for a boolean variable* b *and a Constraint* C, *states that* b *is true if and only if constraint* c *holds:*

- $b = 1 \Leftrightarrow C$.
- $b = 0 \Leftrightarrow \neg C$.

Specifically in the constraint solver package that we are using, Choco Library [Cho], the behavior of a reified constraint is the following:

- if b is instantiated to 1 (respectively to 0), then c (respectively $\neg c$) is propagated.
- otherwise:
  - if c is entailed, b is set to 1
  - else b is set to 0.

One approach to solve this inconsistency in the constraints system is to use reified constraints implementing a MaxCSP problem. We can try to maximize the amount of satisfied constraints, even though not all of them are satisfied. On the one hand, we have the model of gaps variables, and a solution to the inconsistency problem in this model can be obtained using reification for each constraint keeping its satisfiability status. On the other hand, we have the model of position variables, where the constraints to be reified are only those that refer to the alignment criteria and not those constraints that keep the sequence order. These last ones should always be satisfied because we can not lose the meaning of the amino acid order in the sequences.

So, in summary we have the following constraint in each model:

- Gaps Model: variables represent gaps between amino acids, relating them with position expressions.
  - Reified Constraints: $Expr1 \ = \ Expr2 \Leftrightarrow b_m$, where *Expr1* and *Expr2* are arithmetic expression relating some gap variables and defining the position of an amino acid, and $b_m \in \overrightarrow{B}$ is a boolean variable for this constraint stored in a vector of boolean variables $\overrightarrow{B}$. All the constraints of position over gap variables are reified.

- Position Model:
  - Normal Constraints: $X_i < X_{i+1} \ \forall i, \ 0 < i \leq n-1$ where n is the size of the sequences. This constraints are forced to be satisfied, we do not allow amino acid switching their position in the same sequence.
  - Reified Constraints: $X_i \ = \ Y_j \Leftrightarrow b_m$, where X_i represents amino acid a_i in a sequence s_1 and Y_j represents amino acid a_j in a sequence s_2, and $b_m \in \overrightarrow{B}$ is a boolean variable for this constraint, stored in a vector of boolean variables $\overrightarrow{B}$.

The idea of using Reified Constraints is to maximize the number of constraints that can be simultaneously satisfied. So, we can use a value function *sum*, $sum(\overrightarrow{B})$ that we try to maximize.

5.6 Defining the search strategy

One way to decrease, at least in the average case, the execution time for the solution of the constraints is to address the search of the solution for the constraint solver. The main advantage of guiding the exploration of the search space is that we can define a strategy in the order in which the variables are selected by the labeler and also the order in which the values from the domain are obtained.

When we use reified constraints, for instance, we can start labeling the boolean variables. Moreover, if the strategy is to maximize the sum of such variables, we can start to take the upper bound values from such domain. With this strategy, the reified variables will be all 1, meanwhile the constraints attached to such reified one can be simultaneously satisfied with the actual propagated constraints.

To proceed, we can add an strategy for the labeling of the variables in the alignment constraints. In both models we can define the strategies as starting from the variables that are used in the constraint positions, and then following to the others. In both cases the results show a better execution time when the variables are labeled by columns, i.e.

first all the variable in the first position of all the sequences, then the variables in the second position, and so on.

To do this in Choco, we can create a data structure, precisely an array to maintain the *static order* of the variables and then define the strategy as *static variable*, proceeding according to their appearance in such array. Also the increasing domain, or decreasing domain options are allowed when we define the strategy to select the values from the domain in certain order.

We describe now the importance of the search strategy showing some short examples for both models, that reflect the impact it has when mapped to a problem of bigger magnitude. To do this, we describe a little alignment for two sequences as follows:

$$\begin{array}{rccc} & 1 & 2 & 3 \\ seq_1: & A & D & A \\ seq_2: & D & A & D \end{array}$$

- Position Model:
  - Order Constraints:
    - $A_{11} < D_{12} < A_{13}$
    - $D_{21} < A_{22} < D_{23}$
    - We mention as $\overrightarrow{V}$ to this position variables.
  - Alignment reified Constraints:
    - $A_{11} = A_{22} \Leftrightarrow b1$
    - $D_{12} = D_{21} \Leftrightarrow b2$
    - $A_{13} = A_{22} \Leftrightarrow b3$
    - $D_{12} = D_{23} \Leftrightarrow b4$
    - We keep all the variables in a vector: $b_i \in \overrightarrow{B}$

So, let us analyze how the constraint propagation occurs if we consider different strategies.

We start labeling the boolean variables, $\overrightarrow{B}$, and we try to maximize the sum of $\overrightarrow{B}$, propagating the respective reified constraints. First of all, the labeling process takes a variable in $\overrightarrow{B}$, and propagated the corresponding reified constraint *Ci*, that in case of satisfied constraint this b_i boolean takes the value of 1. The propagation of *Ci*, causes a pruning of the domain of the variables that *Ci* relates to and also a propagation of the < constraints for those variables. Then the next variable in $\overrightarrow{B}$ is labeled and the consistency check is done. If the propagation of the constraint *Cj* for the selected variable *bj* is satisfied, it continues with the next one, otherwise *bj* takes the value of 0. At the end of the labeling of all $\overrightarrow{B}$, it has a solution with a value for sum ($sum = \sum_i b_i \in \overrightarrow{B}$). When we try to maximize the sum of $\overrightarrow{B}$, we can start labeling them with the maximum value of their domain (1), when it finds a solution it has the solution found so far, and the algorithm can start searching for an assignment that maximizes this value.

Following we can see the search tree for this strategy, using the Position Model representation. Here the boolean variables are labeled, with the respective constraint propagation, and the solution of sum equal to 2 is found in the first spanning, pruning instantly other search that does not lead to a better solution. Also the final domain for the variables is shown and the labeling for Vars and Boolean (position and boolean variables).

```
..[2] down branch b1:?==1 branch 0
...[3] down branch b2:?==1 branch 0
...[3] up branch b2:?==1 branch 0
...[3] down branch b2:0==0 branch 1
....[4] down branch b3:?==1 branch 0
....[4] up branch b3:?==1 branch 0
....[4] down branch b3:0==0 branch 1
.....[5] down branch b4:?==1 branch 0
......[6] down branch a11:?==1 branch 0
.......[7] down branch d12:?==2 branch 0
........[8] down branch a13:?==3 branch 0
- Solution #1 found. Objective: 2, 1 Solutions, 0 Time (ms), 8 Nodes, 2 Backtracks, 0 Restarts.
  a11:1, d12:2, a13:3, d21:0, a22:1, d23:2, b1:1, b2:0, b3:0, b4:1, sum:2,
- Restarting search - 1 Restarts
..[2] down branch b1:?==1 branch 0
...[3] down branch b2:?==1 branch 0
...[3] up branch b2:?==1 branch 0
..[2] up branch b1:?==1 branch 0
- Search completed
  Maximize: sum:2
  Solutions: 1
  Nodes: 10
  Backtracks: 4
  Restarts: 1

Vars:
a11: {1...1}
d12: {2...2}
a13: {3...3}
d21: {0...0}
a22: {1...1}
d23: {2...2}
Boolean
b1: [1,1]
b2: [0,0]
b3: [0,0]
b4: [1,1]
```

If instead of starting to label the variables from $\overrightarrow{B}$, we start labeling the position variables, the constraint *lt* (< constraint) is propagated first. This removes some values from the domain of the variables by enforcing arc-consistency, and when nothing else can be done the reified constraints of $\overrightarrow{B}$ are propagated and the possible value of the boolean variables calculated, resulting in the respective sum for these boolean variables, according to the labeling of the *lt* related variables. But as the $\overrightarrow{B}$ values are a consequence of the $\overrightarrow{V}$ propagation, we have to calculate all the possible labeling for $\overrightarrow{V}$ if we want to maximize the sum of $\overrightarrow{B}$. So, a strategy that starts labeling the $\overrightarrow{V}$, but that maximize the sum of $\overrightarrow{B}$, does not prune much of the search space because each time it finds a solution for a value *sum* all the possible labellings of $\overrightarrow{V}$ are explored searching for a larger value of *sum*, although possibly in vain.

In the next figure5.5, we can see the space search and the labeling process as described before. In this case, the whole labeling starts with the variables of the gaps, and then follows with the boolean variables. All the possible assignments are done first to the gap variables, and then the boolean varaibles to get the actual sum.

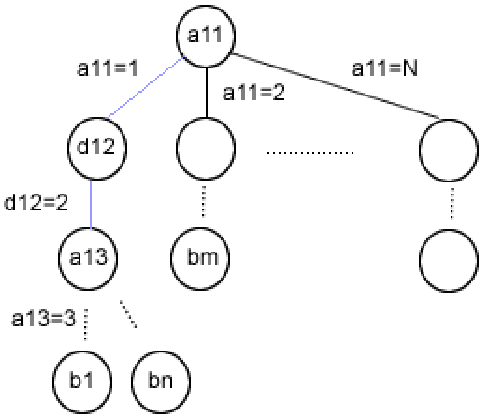


Figure 5.5: The search space is totally expanded for the gaps variables

The result size of the search space is the next one, with the respective solution:

```
Vars:
a11: {0...0}
d12: {1...1}
a13: {2...2}
d21: {1...1}
a22: {2...2}
d23: {3...3}
Boolean
b1: [0,0]
b2: [1,1]
b3: [1,1]
b4: [0,0]

- Search completed
  Maximize: sum:2
```

```
Solutions: 3
Time (ms): 63
Nodes: 53
Backtracks: 70
Restarts: 3
```

We can compare the amount of backtracks and restarts done by the solver and the nodes created for the tree. In the first search strategy that we define, the tree is smaller and the solution is instantly detected because all the violations are propagated and values removed after the boolean assignment of the reified variables.

In the second example instead, the whole search space is explored (we can see the number of nodes), and the larger number of backtracks and restarts when the partial labelling does not lead to a solution.

So, defining the search strategy according to the constraints that are set in the problem and the way these constraints are propagated can determine drastically the result of our algorithms.

5.7 Experimental Results and Analysis

To test the two models presented above we prepared some data sets, extracted from the protein sequence simulation, that contain different sizes and number of sequences and also different distributions of amino acids to ensure that many distinct and sometimes contradictory constraints appear as part of the alignment.

In the next chapter, we work over subsequences of small length. This determines our selection of the size of the sequences for the data sets to test our methods in this chapter, obtaining a result with an impact on the future chapters.

In the table 5.2 and table 5.3 we can see the results for the experiments with the Gaps Model and Position Model respectively. The columns are labeled as *Nr Seq*, meaning the number of sequences, *Length* meaning the amount of amino acids, *Constr* the number of constraints according to each model, *Conflicts* describe the number of constraints that are in conflict, *Set Up* is the time in milliseconds it takes to set the constraint and the creation of the model before the solution search, and finally *Solve* represent in milliseconds the execution time to find a solution.

The contradictory (Conflicts, called in the tables) are those constraints that cannot be satisfied simultaneously, which implies that some of them make the reified variable takes the value 0, and to generate some pruning in the domain of the variables. We show first the results, we explain them and finally we show some analysis on the advantages of using constraints.

All the algorithms were implemented in Java, using the Choco Solver library [Cho] and run on an Intel Pentium Core 2 duo 1.6Ghz, 2gb RAM.

Nr seq	Length	Constr	Conflicts	Set Up (ms)	Solve(ms)
10	8	39	5	200	941
8	6	20	6	178	532
7	5	24	3	160	604
20	5	54	12	184	1568
50	5	159	8	297	3458
50	6	45	14	260	1653

Table 5.2: Gaps Model, set up and execution times.

Nr seq	Length	Constr	Conflicts	Set Up (ms)	Solve(ms)
10	8	123	5	168	98
8	6	72	4	153	75
7	5	60	5	170	42
20	5	143	13	210	103
50	5	449	15	350	352
50	6	290	14	287	324

Table 5.3: Position Model, set up and execution times.

As we can observe in the tables 5.2 and 5.3, the Position Model shows a better performance compared with the Gaps Model in the search of the solution. Though it requires more constraints, the first advantage of the Position Model is its pruning power. The maximization of the boolean variables, specifying a *MaxCSP* problem, has the advantage of solving the contradictory constraints problem and always finding a solution to the alignments.

The number of constraints is linear with respect to the number of sequences and size, because we are just aligning each sequence with the next one. In the way we use this model in the next chapter, this type of alignment is enough. If it is the case of all sequences being aligned with all other sequences, we have a number of constraints that grow exponentially with the number of sequence and, in this case, the execution time increases exponentially (we did not test this case here). A timeout is set for the search, for 10 minutes, so if a criteria of setting the constraints impose an exponentially increase of their number, we can still handle the problem. Of course, if all the solutions to find involve an exponentially growing number of constraints, the solution found within the time limit will not be the best at all.

The execution times as we can see in the result tables, are linear for the position constraints, and they do not increase too much when the number of conflicts increases. The Position Model shows a better performance due to the pruning also when there are conflicting constraints, but is not the case of the Gaps Model. In the Gaps Model, as we show later, the pruning system is not as effective as in the Position Model and thats why the execution time is so susceptible to conflicts in the constraints.

We are not proposing a MSA, because we focus on a tool to solve the eventual errors done by a MSA on variable regions that coevolve, and to re-aling those particular mistakes. The tractability is reasonable for sequences of considerable length according to the problem presented in the next chapter, and we can see in the results how this tool can help to solve our main goal.

In the Position Model, based on variables representing the amino acid positions, we propose two different ways for representing the sequence order constraints, and we choose the best one to realize the experiments:

- Solution 1: $\forall 0 < i < N-1$ Constraint: $vars[i] < vars[i+1]$.
- Solution 2: Global constraint *increasing_nvalue(N,vars)*. This global constraint enforces the increasing order for the positions, and also enforces that at least N values be distinct.

Solution 2, is not the best approach because this constraint is enforcing arc-consistency but also is checking N-Value. Choco does not provide a strictly increasing global constraint and although we fix N as a constant, to be the amount of variables to label incrementally without repetitions, the filter is still managing huge data structures and complex algorithms. Although we can indicate to the constraint *increasing_nvalue* to use an option that force the approximation from the lower bound, that can be computed in polynomial time, as explained in [Cho], using this constraint is not as efficient as solution 1.

Solution 1 shows the better approach to use in the order sequence constraints. This is due to the fact that the constraint '<', in Choco *lt*, enforce arc-consistency and since we relate all the variables with this constraint, $lt(X_i,X_{i+1})$, $lt(X_{i+1},X_{i+2})$,$lt(X_{i+2},X_{i+3})$ and so on, during the constraint propagation starting by *lt*(X_i,X_{i+1}), the values in the domain of the variable X_i that have no support in the variable X_j through this constraint are removed, and also the other way around. So, every time the labeler assigns a value to any of these variables and propagates the constraints that enforce arc-consistency, some pruning is done if it is possible.

Clearly the Position Variables model is the best, as our experiments show, and its due to the efficacy of pruning against the less effective pruning of the Gaps model as explained in the next section. We also searched for the best approach in the Position Variable, according to the possible constraints to use and we got a robust model after some adjustments. The efficiency of the alignment is reduced if we want to add the flexibility of having constraints. In the case of sequences, just for a pure alignment, or a best alignment, many algorithms were introduced as we explained in the preliminaries chapter. But when we need to link some positions and restrict them according to the values they take, strict algorithms like Dynamic programming based on the scoring matrix are not as flexible as a model with constraints. Solving an alignment with some criteria, such as our tests with the constraint framework is an upper bound with respect to the complexity of the task that we realize in the future to fix misalignment in coevolving regions, as shown in the next chapter. We can claim that this method works well to set constraint on the sequences and align under specific criteria.

Also we showed how the efficiency changes when a different search strategy is selected, and we proposed the best one according to our results. If any extension to the model is done, for instance adding external constraints to the variables, these strategies should be reconsidered again to obtain the most efficient method to find the solutions.

5.8 Limitations of the models

In the case of the gap variables model, we found a big problem in the pruning system. When the solver calls the pruning, initially we expect that all those reified constraint that are labeled with 1 and that force a constraint C to be satisfied, force the pruning of values in the domain of variables linked by C, and also variables related with the constraints that are in conflict with C.

Explicitly, we have $b1 \Leftrightarrow C1$, and $b2 \Leftrightarrow C2$, where b1 and b2 are two boolean variables, and C1 and C2 two constraint over the expression that define position for the amino acids.

And now, let us suppose that the constraints try to align are like this:

$$\begin{array}{cc} A & P \\ P & A \end{array}$$

So C1 align the A's and C2 align the P'2, which implies that $C1 : X1 = Y1 + Y2 + 1$ and $C2 : Y1 = X1 + X2 + 1$.

If we search for a labeling of the boolean variables that maximize the sum of the values (as explained before for the MaxCSP), we can get b1=1, and b2=0 because both constraints simultaneously result in an unsatisfiable problem. But b2=0 implies that C2 cannot be satisfied, which implies a restriction in the domain of the variables of such expression. The problem is that the constraint solver does not go farther with the propagation, and does not resolve this kind of algebraic equations. So to solve that equation the solver has to start the labeling of the variables and then decide which values are in a solution.

Clearly this limitation of the Gaps Model makes it the worst of the two models when we measure the execution time. Anyway, this model is more flexible than the position one when we care about the gaps between the amino acids, and we want to work directly on that information, i.e. the assignment of the values to the variables is directly reflected as gaps insertions. The Position Model instead, gives more flexibility when we want to work directly on the position of the amino acid, but the gap length are then a consequence of the amino acid positions.

6

Using constraints to fix the alignment

In this chapter we explore some heuristics to improve the identification coevolution sites, once we calculate all the sites containing more than a certain amount of correlation.

As we described in section 4.5, Multiple Sequence Alignments obscures the coevolution information because of mistakes in aligning variable regions, resulting on many misalignment in the locations of coevolved sites.

We propose a method for checking the information in the potentially coevolved sites and determine if they have actually coevolved according to the possible misalignments around such site produced by the aligner. During this chapter we present some approaches to proceed and decide, from the set of pairs of sites that are candidates as possibly coevolved, which of them give sufficient information to satisfy our claim.

Moreover, in this chapter we show the results to support the claim that the existent methods to detect candidates for coevolved sites present a high failure rate. These facts are the grounds of our motivation for improving the detection, reducing the failure rates and trying to fix mistakes in the alignment.

We also propose techniques for fixing the alignment using the method described in 5.4 and correct errors in the sites that we determine are actually coevolved. We also measure the impact of re-aligning such sites to correct the misalignments and we check the impact of this on the mutual information measures.

In the end of the chapter we also show experimental results on which we base our analysis of the advantages and drawbacks of this approach.

6.1 General problem and definitions

Since we detected a loss of coevolution information after the sequences are aligned, as described in section 4.5 because the aligner does not take coevolution into account or because the variation in coevolving sites produces scores that are not so good for the optimal alignment, we are searching now a method to detect these mistakes and correct them by aligning these regions with the model introduced in the previous chapter.

After we align the sequences with some of the multiple sequence alignment tools, we want to identify such coevolved sites. The first step is then, to exclude pairs of sites that have no correlation, which suggest they did not coevolve. To do this, we use the method of mutual information to determine those sites that are above certain threshold of mutual information.

When we talk about a naïve algorithm to detect coevolved sites, we are just defining the algorithm that checks the mutual information between a pair of sites *(i,j)* and concludes that the sites have coevolved if their mutual information, *MI(i,j)* is over a threshold *T*.

We consider the $\frac{n(n-1)}{2}$ pairs of sites, and we calculate the $MI(i,j)$ as described in previous sections. Keeping these sites above a given threshold *T* we can define:

- $MI(i,j) >= T$, (i,j) a pair of sites that we guess as coevolved sites.
  - If (i,j) are really coevolved sites, we can say that this pair is a true positive (TP).
  - If (i,j) are really not coevolved sites, we can say that this pair is a false positive (FP).
- $MI(i,j) < T$, a pair of sites that we discard as unlikely coevolved sites.
  - If (i,j) are really coevolved sites, we can say that this pair is a false negative (FN).
  - If (i,j) are really not coevolved sites, we can say that this pair is a true negative (TN).

Under this classification, we need a method to obtain a better fraction of true positives and true negatives than the naïve algorithm, decreasing the false positives and false negatives. With such method we expect to detect more precisely which sites are coevolved and extend the detection to the correction of the sequences if a misalignment was done in such site.

6.2 Map based detection

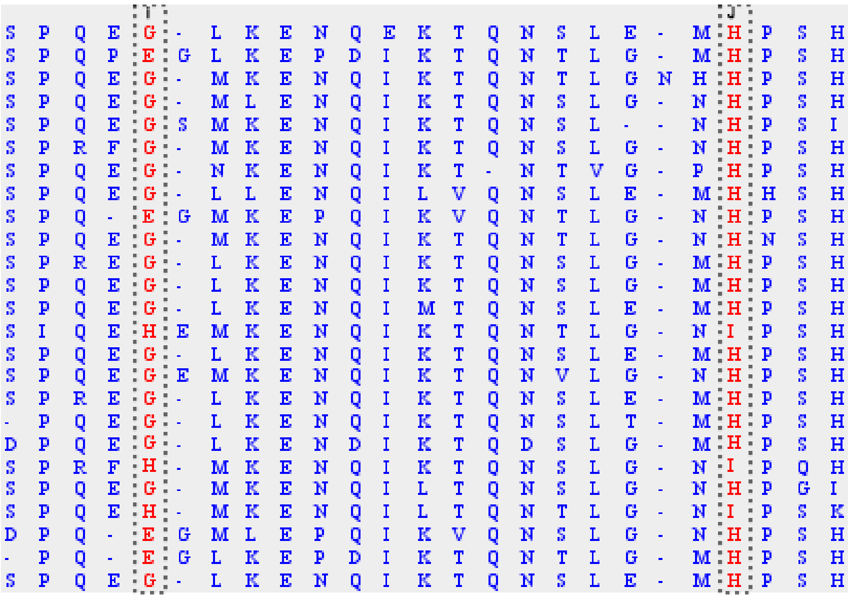
This approach focuses on the information present in a pair of sites (i,j), and tries to guess the coevolution mapping. This mapping is the selection pressure that one site generates on the other, as explained in previous sections and presented in [WP05]. In other words, the mapping are the instances of amino acids in the correlation from one site to the other along the multiple sequences.

We are proposing a way of detecting errors in the alignment of coevolved sites, extracting this mapping from the sites in question and checking along the sequences the amino acids present for those sites.

Once we assume a mapping for these sites, we can search the sequences where the mistakes occur for those sites according to the guessed mapping, and we analyze if in the alignment it is possible to increase the coevolution information, doing some correction to set in the coevolved sites a pair that belongs to such mapping.

This definition stems from the nature of coevolution. One change in one of the sites, forces the other to change and forces that the new amino acid to follow the previous one through a bijection.

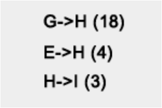
Definition 16. *A mapping between sites i and j, from now on Map(i,j), is a set of different pairs of amino acids present along the sequences for the sites i and j. We keep the appearance time of each pair.*



```
        I                                  J
S P Q E G - L K E N Q E K T Q N S L E - M H P S H
S P Q P E G L K E P D I K T Q N T L G - M H P S H
S P Q E G - M K E N Q I K T Q N T L G N H H P S H
S P Q E G - M L E N Q I K T Q N S L G - N H P S H
S P Q E G S M K E N Q I K T Q N S L - - N H P S I
S P R F G - M K E N Q I K T Q N S L G - N H P S H
S P Q E G - N K E N Q I K T - N T V G - P H P S H
S P Q E G - L L E N Q I L V Q N S L E - M H H S H
S P Q - E G M K E P Q I K V Q N T L G - N H P S H
S P Q E G - M K E N Q I K T Q N T L G - N H N S H
S P R E G - L K E N Q I K T Q N S L G - M H P S H
S P Q E G - L K E N Q I K T Q N S L G - M H P S H
S P Q E G - L K E N Q I M T Q N S L E - M H P S H
S I Q E H E M K E N Q I K T Q N T L G - N I P S H
S P Q E G - L K E N Q I K T Q N S L E - M H P S H
S P Q E G E M K E N Q I K T Q N V L G - N H P S H
S P R E G - L K E N Q I K T Q N S L E - M H P S H
- P Q E G - L K E N Q I K T Q N S L T - M H P S H
D P Q E G - L K E N D I K T Q D S L G - M H P S H
S P R F H - M K E N Q I K T Q N S L G - N I P Q H
S P Q E G - M K E N Q I L T Q N S L G - N H P G I
S P Q E H - M K E N Q I L T Q N T L G - N I P S K
D P Q - E G M L E P Q I K V Q N S L G - N H P S H
- P Q - E G L K E P D I K T Q N T L G - M H P S H
S P Q E G - L K E N Q I K T Q N S L E - M H P S H
```

Figure 6.1: Sequences and coevolution guessing

To understand this concept better, we can see the sequences in figure 6.1 from where we analyze the sites i and j and we extract the mapping show in the figure 6.2.



G->H (18)
E->H (4)
H->I (3)

Figure 6.2: Mapping for the sequences in sites I and J

The next step is to normalize the mapping, in a way that we obtain a bijection between keys of the mapping and values, because we base our approach on the next definition:

Definition 17. *Two sites* i,j, *are coevolved if and only if columns i and j satisfy a bijection over the amino acids they have along the sequences.*

The normalization we present, is a compression and grouping of the elements of the Map, in the sense that we group elements of the map with the following criteria:

- We sort the Map from the most to the least frequent elements. So, if A->B is the first element of the Map (A is the key, B the value), this means that is the most repeated pair in the sites i,j.

- We iterate for all the elements of Map, from the least frequent pair to the most frequent pair. For each iteration, we iterate again from the current pair, say M->N, to the most frequent pair, and if there exists a pair K->V that appears more times than the current pair M->N, and M==K or N==V, then the number of pairs M->N is added to the total of pairs K->V, and we remove the pair M->N (M->N are included are counted now as if they were K->V pairs). We repeat this process until no more compression can be done.

This normalization will give us a bijection in the mapping. For instance, and following the example of figure 6.1, the normalized mapping for the sites i,j of the example can be seen in the figure 6.3. With this bijection we obtain the most probable exact matching for different pairs, between one site and the other based on the information in the sites.

With this bijection, we can check if the pairs are always preserved in the corresponding sites along the sequences or in those in which the bijection does not hold, we cam check if there is a way to fix the misalignment and preserve this mapping. The method is an heuristic to proceed after the aforementioned MI detection, because the bijection is not perfect an can add mistakes because the misalignments, and also because we are assuming that in the coevolved sites there are no random mutations other than those in the coevolution mapping we assume.

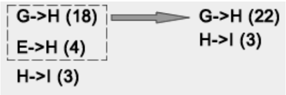


Figure 6.3: Normalization of the mapping: at left the Map, at right the normalized one

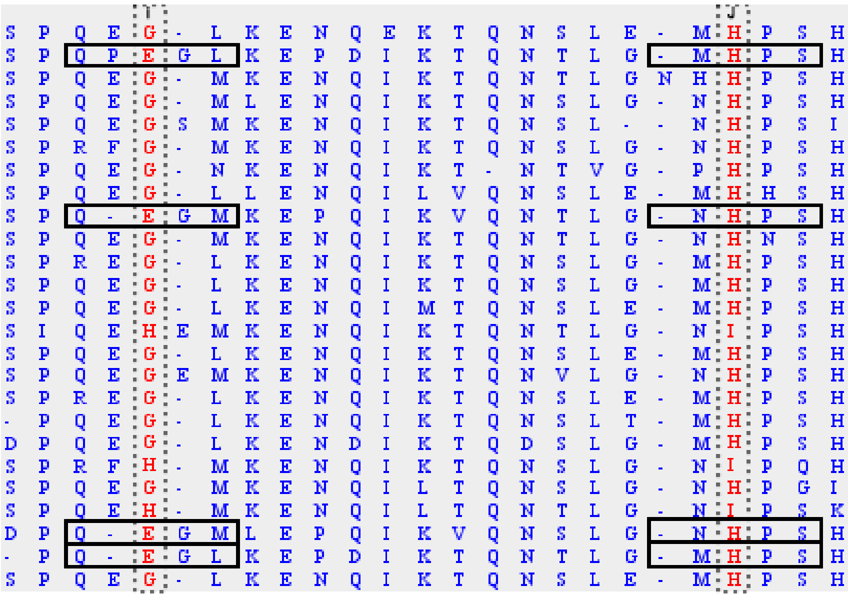
Algorithm 4 CoevolveDetection

```
for $i = 0..M$ do                                          ▷ M are the sites
    for $j = i+1..M$ do
        if $MI(i,j) = K$ then
            $Coev \leftarrow Coev \cup (i,j)$;
        else
            if $(MI(i,j) >= T$ then                        ▷ T is the Threshold
                $Map \leftarrow getMapping(i,j)$;
                $NMap \leftarrow Normalize(Map)$;
                if $cardinality(NMap) > 1$ then
                    $ListPos \leftarrow getMistakes((i,j), NMap)$;    ▷ ListPos get all sequences number where we detect mistakes based on NMap in sites (i,j).
                    $Can \leftarrow TRUE$
                    for all $pos \in ListPos$ do
                        if $notcanFix(pos, NMap, (i,j))$ then
                            $Can \leftarrow FALSE$  ▷ canFix check if we can shift obtaining in position i,j elements of the mapping
                            break;  ▷ break stop considering this site as a possible coevolution
                        end if
                    end for
                else
                    $Can \leftarrow FALSE$;                ▷ NMap only one element, Not Coevolution.
                end if
                if $Can = TRUE$ then $Coev \leftarrow Coev \cup \{(i,j)\}$;
                end if
            end if
        end if
    end for
end for
return Coev;
```

In the algorithm 4 we can see how to proceed for every pair of sites. First we start by guessing the coevolved sites, and discarding the others below the threshold. With the remaining ones, those above a threshold T, we continue to get the mapping and normalize as explained previously. Since the normalized mapping, NMap, can have only one element, we have to make a decision in such cases. If there is only one element, it means that we can correct mistakes and probably get the same amino acid a for all the sequences in site i and the same amino acids b for site j. In this case there is no evidence for coevolution at all, but only for simply conserved sites through all the evolutionary generations, so we can discard this pair of sites from the candidates for coevolved.

If it is not the case explained above for only one element in NMap, then the canFix function is called. This function considers two subsequences s_i,s_j of the sequence to be analyzed, width a length of window W and with center point the sites i, and j respectively. The main part of the check consist in comparing the amino acids in those subsequences, with those in the mapping, and determining if the sites have possibly coevolved if there exists a pair of sites (a_i, b_j), such that $a_i \in s_i$ and $b_j \in s_j$, and $(a_i, b_j) \in NMap$. This checking is explained in the algorithm 5

As an example of this checking, we can see for each mistake in the Figure 6.4 the window in the site i and j, which whom we check the condition explained before. In this case, all the elements in site i in the window considered and in the position where the mistakes occur have an element in the key of the normalized mapping. Of course, we check also for site j if the element belongs to the values of the normalize mapping, though in this case this is not necessary because there are no mistakes in j.



```
        i                                       j
S P Q E G - L K E N Q E K T Q N S L E - M H P S H
S P Q P E G L K E P D I K T Q N T L G - M H P S H
S P Q E G - M K E N Q I K T Q N T L G N H H P S H
S P Q E G - M L E N Q I K T Q N S L G - N H P S H
S P Q E G S M K E N Q I K T Q N S L - - N H P S I
S P R F G - M K E N Q I K T Q N S L G - N H P S H
S P Q E G - N K E N Q I K T - N T V G - P H P S H
S P Q E G - L L E N Q I L V Q N S L E - M H H S H
S P Q - E G M K E P Q I K V Q N T L G - N H P S H
S P Q E G - M K E N Q I K T Q N T L G - N H N S H
S P R E G - L K E N Q I K T Q N S L G - M H P S H
S P Q E G - L K E N Q I K T Q N S L G - M H P S H
S P Q E G - L K E N Q I M T Q N S L E - M H P S H
S I Q E H E M K E N Q I K T Q N T L G - N I P S H
S P Q E G - L K E N Q I K T Q N S L E - M H P S H
S P Q E G E M K E N Q I K T Q N V L G - N H P S H
S P R E G - L K E N Q I K T Q N S L E - M H P S H
- P Q E G - L K E N Q I K T Q N S L T - M H P S H
D P Q E G - L K E N D I K T Q D S L G - M H P S H
S P R F H - M K E N Q I K T Q N S L G - N I P Q H
S P Q E G - M K E N Q I L T Q N S L G - N H P G I
S P Q E H - M K E N Q I L T Q N T L G - N I P S K
D P Q - E G M L E P Q I K V Q N S L G - N H P S H
- P Q - E G L K E P D I K T Q N T L G - M H P S H
S P Q E G - L K E N Q I K T Q N S L E - M H P S H
```

Figure 6.4: Mistake identification and "canFix" checking

With the algorithm 4 we have a procedure for checking all the pair of sites guessed as coevolved using the MI threshold T, to determine which of them are most likely to

Algorithm 5 canFix

```
$Sub_i \leftarrow subsequence(pos, i, window)$
$Sub_j \leftarrow subsequence(pos, j, window)$
$Map \leftarrow getMapping((i, j));$
$NMap \leftarrow Normalize(Map);$                    ▷ NMap is the normalized mapping
$Can \leftarrow FALSE$
for all $AminoacidA_i \in Sub_i$ do
    for all $AminoacidA_j \in Sub_j$ do
        if $(A_i, A_j) \in NMap$ then
            $Can \leftarrow TRUE);$
            break;          ▷ break because we know we can align for some amino acids
        end if
    end for
end for
return $Can$
```

have coevolved based on this idea of the mapping. The method is totally based on the assumption that the normalized mapping for a pair of sites is the correct coevolution map for those sites. Of course, this assumption can be wrong, and can insert mistakes when the assumed mapping is not the correct one, but, in this approach, we are working under the assumption of coevolution with perfect matching as we generate the sequences. Furthermore, this feature can be extended considering a minimization of the mistakes in the mapping if other mutations can occur in the coevolved sites.

Once the coevolved sites are detected with this method, we can fix those sites which the normal aligner misaligned and lost the information about the coevolution. We proceed with the alignment correction, as explained in the next section, using the method of alignment with constraint explained previously in Chapter 5.

6.2.1 Fixing the alignment

Once we have the candidate sites identified as having coevolved, with their respective normalized mappings, we can start to fix the misalignments in these sites to get the correctly aligned sequences that show the coevolution for these sites.

As explained in Chapter 5, we can use variables for the gaps or variable that represent the positions of the amino acids. No matter which method we use to align, the important thing is to clarify how the constraints are created.

Given the pair of sites *(i,j)*, we consider a window W such that two regions are determined: $R1 = [i-W, i+W]$ and $R2 = [j-W, j+W]$. These regions define the window of sites over which we try to fix the alignment, and also correspond to the window used to disambiguate the coevolved sites with the mapping method as explained before.

We need to set a precondition, and it is that we fix the alignment of a pair of sites *(i,j)* if there exists other pair *(k,l)* that has been fixed, then *(i,j)*, *(k,l)* are not in conflict. We define two pair of sites to be in conflict if some of their regions intersect. With this precondition we avoid shifts in the amino acid to correct one site, that damage the coevolution for other pair of sites. Also we skip to fix a pair of sites *(i,j)* if the distance between *i* and *j* is smaller than two times the window *W*.

With the regions *R1,R2* we can create two matrices Mat_{posI} and Mat_{posJ}, where the columns are the sites in *R1* and *R2* respectively and the rows are the subsequences aforementioned. In these matrices we store the amino acids for such sequences in the respective sites.

We work now with the matrices Mat_{posI} and Mat_{posJ}, setting the constraints on them based on the mapping extracted previously. As we can see in the algorithm 6, the general procedure is, for each sequence we check the pairs (a_i, b_j) in the sites *i* and *j* iterating along the sequences. If $(a_i, b_j) \in NMap(i, j)$, then the sequence number is added to the list *ListCorrect*, otherwise the sequence number is added to the list *ListMistakes*, where $NMap(i, j)$ is the normalized mapping for the sites *i* and *j*.

During this search, we also set the constraints for those amino acids that can be aligned with amino acids in the next subsequence. So it can happen that one amino acid can be aligned to the coevolved site, or also with another amino acid from other site, giving two possibilities that are resolved using the reified constraints. Indeed, all the constraints that we set with the function *addConstraint* are reified constraints, and we try to maximize their satisfiability. To give a higher priority to the alignment with the coevolution, we can just increase the weight of those reified variables related to constraints with the anchor variables, by a constant factor, obtaining a maximized value if the constraints related to the anchor variables are set, plus all the other alignment constraints. The anchor variables keep the position of the coevolved sites when the constraint produce some shift.

For those positions such that the pairs $(a_i, b_j) \in NMap(i, j)$ that we get iterating the *ListCorrect*, we set the constraints to preserve the correct alignment of the amino acids in the coevolved site (i,j), and we force these positions to be some anchor variables I and J respectively. Iterating the list *ListMistakes* we get those pairs $(a_i, b_j) \notin Map(i, j)$, because a_i andor b_i are misaligned, and we search inside the window all the possible combinations of (a_i, b_j), $i \in R1$ and $jinR2$, such that $(a_i, b_j) \in NMap(posI, posJ)$ and we set the constraints such that if it is the case that a_i is in the position of the anchor variable *I*, then b_j is in the position of the anchor variable *J* and viceversa. With this constraint we force the variables to align according to the mapping, in the position of the coevolved sites. Note that we use a different *addConstraintWeighted* for the coevolved site, giving more priority to this by increasing the weight of the reification

Algorithm 6 setConstraints(siteI,siteJ)

```
$posI \leftarrow siteI$
$posJ \leftarrow siteJ$
$NMap(posI,posJ) \leftarrow previouslycomputednormalizedmappingfor(posI,posJ)$
$Mat_{p}osI \leftarrow subMatrix(posI,window)$
$Mat_{p}osJ \leftarrow subMatrix(posJ,window)$
$N \leftarrow amountofsequences$
for $seqK = 1..N$ do
    if $(Mat_{posI}[seqK][PosI], Mat_{posJ}[seqK][posJ]) \in NMap(posI,posJ)$ then
        $ListCorrect.insert(seqK)$
    else
        $ListMistake.insert(seqK)$
    end if
    for $i = 1..(Window * 2 + 1)$ do
        for $j = 1..(Window * 2 + 1)$ do
            if $(Mat_{posI}[seqK][i] == Mat_{posI}[seqK+1][j])$ then
                $addConstraint(eq(VAR(seqK,posI+i),VAR(seqK+1,posI+j)))$    ▷ alignment for any amino in the window of site I with the next sequence
            end if
        end for
    end for
    for $i = 1..(Window * 2 + 1)$ do
        for $j = 1..(Window * 2 + 1)$ do
            if $(Mat_{posJ}[seqK][i] == Mat_{posJ}[seqK+1][j])$ then
                $addConstraint(eq(VAR(seqK,posJ+i),VAR(seqK+1,posJ+j)))$    ▷ alignment for any amino in the window of site J with the next sequence
            end if
        end for
    end for
end for
for $k = 1..ListCorrect.size() - 1$ do
    $seqK = ListCorrect[k]$
    addConstraint(eq(VAR(seqK,posI),I)
    addConstraint(eq(VAR(seqK,posJ),J)    ▷ Anchor variables constraints.
end for
for $k = 1..ListMistake.size() - 1$ do
    $seqK = ListMistake[k]$
    for $i = 1..(Window * 2 + 1)$ do
        for $j = 1..(Window * 2 + 1)$ do
            if $(Mat_{posI}[seqK][i], Mat_{posJ}[seqK][j]) \in NMap(posI,posJ)$ then
                $addConstraintWeighted(eq(VAR(seqK,i),I) \Leftrightarrow eq(VAR(seqK,j),J))$    ▷ constraint to set a valid pair in the coevolved sites.
            end if
        end for
    end for
end for
return solve();    ▷ solve the constraint and return the matrices with the corrections.
```

variables corresponding to this constraint.

In the algorithm we use VAR(sequence,pos) to designate a variable corresponding to the amino acid that we are taking into account in a window W for a site I. This function just gives us the corresponding variable for the amino acid in question.

Once we extract the matrices for all sites that we guessed as coevolved, and we set these constraints, the alignment is done solving them iteratively. Then the whole alignment is reconstructed again, replacing the matrices in the correspondent places.

We can integrate the algorithms in a general procedure, that we test in future sections, as Fix algorithm and described in algorithm 7.

Algorithm 7 Fix

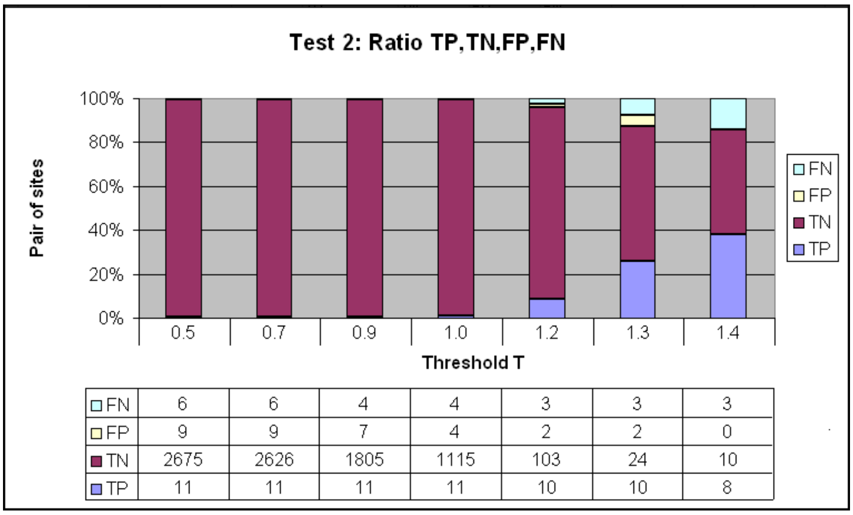
```
for all $(i, j) \in CoevolveDetection$ do    ▷ This algorithm returns the canFix checking for pair of sites over T.
    if $notConflict((i, j), listConflict)$ then
        $listConflict.Add((i, j))$
        $Matrixi, Matrixj \leftarrow SetConstraints(i, j)$
        $Alignment$ ←Replace Matrix i, Matrix j.
    end if
end for
```

6.2.2 Results

To test this approach, we generated a set of sequences, we aligned them and we compared these alignments with the coevolved sites generated. All the test are done taking a window of size 2 to each side of the site because it produce the best results in all the cases.

First we test our method, the mapping based one proposed in this chapter. We generate a set of sequences with different mutation, insertion and deletion ratios. For this set of sequences we run the algorithm that tries to detect, in the pool of pairs of sites with value MI over a threshold T, which of them are really coevolved and which ones can be discarded. To measure the effectiveness of this algorithm we identify the correct coevolved sites from the sequences generation step and we can compare that information against the result of this method.

In the figure 6.5, we can see the results for a constant threshold T. As we can see in this graphic, our method is discarding many sites as possible coevolved sites (the True Negatives,TN) that under the naïve algorithm fall in the assumption of coevolved. Also many True Positives (TP) are detected which means that there are candidates to be re-aligned increasing their MI. All the TN detected are False Positives (FP) in the naïve algorithm. This result shows that our mapping method to identify those possible pairs of sites works better than the naïve method and we can proceed to use it to determine which sites can be re-aligned.



□ FN	6	6	4	4	3	3	3
□ FP	9	9	7	4	2	2	0
■ TN	2675	2626	1805	1115	103	24	10
■ TP	11	11	11	11	10	10	8

Figure 6.5: Ratio of detections

First we showed how effective was our method, and now we can show how it helps fixing the sites that are detected as coevolved against the naïve algorithm. To realize these tests, we generated many sequences with different coevolution methods and different generation steps. As we can see in the table 6.1, we generated 5 tests of each type and we grouped them. Each test consists of 100 sequences obtained from the offspring set after the simulation process.

The generations are the times we evolve our simulation generating the child sequences. The different coevolution methods are those as explained in section 2.5. The Blosum method produces a mutation in the driver position (*i*-site) according to the blosum matrix. For this new amino acid there exists a favored state for the mutation of the driver, also following the Blosum probability matrix. The mapping method follow a strict mapping mutation and for each amino acid that mutate in a site, there is only one amino acid to which the driven site mutates. The mapping is constructed with a mutation probability matrix before the simulation, generating a presumed biologically significant correlation between the amino acids. The last method, Cluster, assigns for each amino acid a cluster of favored states. Each cluster contains a set of most probable amino acids to which the driven can mutate, producing a partition of the amino acid set and allowing mutation only within each subset.

The AVGsimilarity column shows the average of pairwise similarity score(%) for all the sequences in the correspondent group of tests, where higher values indicate more similarity, i.e. more homology of the sequences, and lower values more divergence in their similarity. The pairwise similarity values are calculated by Clustal when the alignment is done.

TestID Range	Mutation	Generations	AVGsimilarity(%)
1-5	blosum	20	71
6-10	blosum	25	62
11-15	blosum	30	52
16-20	mapping	20	72
21-25	mapping	25	61
26-30	mapping	30	54
31-35	cluster	20	68
36-40	cluster	25	55
41-45	cluster	30	45

Table 6.1: Tests description. Coevolution mutation: blosum, mapping, cluster.

In the figure 6.6, we can see a graphic of the tests for sequences generated with the Blosum(left) and mapping(right) coevolution methods. The horizontal axis correspond to the number of generations of the tests and the vertical axis to the amount of coevolved sites detected in the top 25% of MI value site calculation over the alignment. Each point represents the average of detection of 5 tests and also the standard deviation is shown for each point.

For generating the tests, we first create the sequences and then we calculate the MI over the sites to find those over a threshold T. We sort the results by their MI value in decreasing order and we get the top 25 percent of that list. From this list we count the number of coevolved sites detected (knowing them from the sequence generation). So, in this way we have the number of coevolved sites detected by the naïve algorithm. Also, with the same sequences and all the pairs of sites over the threshold T, we run our mapping algorithm to determine the possible coevolved sites and re-align the sequences using the constraint method. Then we measure again the MI for all the sites and from this new calculation we take the list of the top 25 percent and we count the number of coevolved sites on this list giving the result for the mapping algorithm.

We count in the 25 percent because with a smaller sample we do not have enough information to disambiguate between the two alogrithms results and more percentage we do not obtain any change in the results. So, 25 percent is a good amount to analyze the results and compare the two algorithms.

The result here shows that the Fix algorithm that we propose detects more coevolved sites than the naïve algorithm, in both coevolved method of generation. We

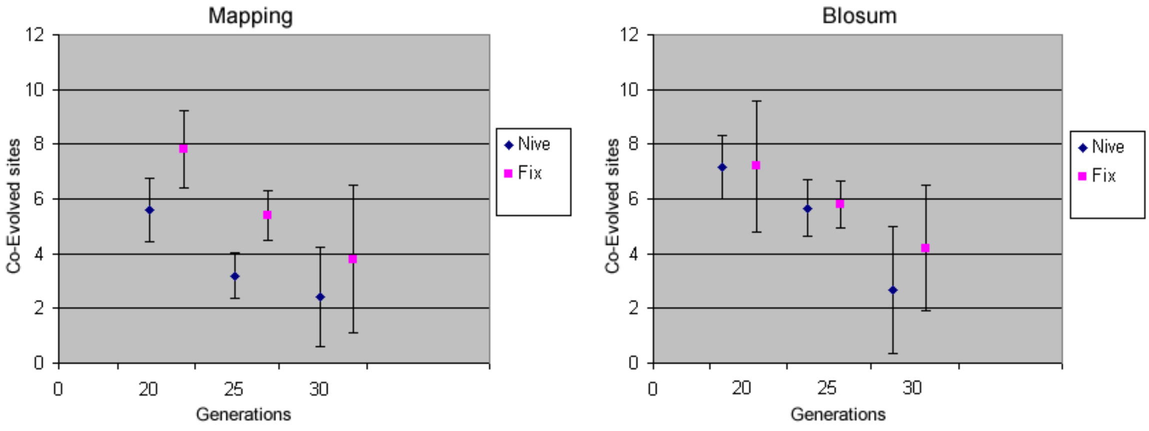


Figure 6.6: Detection using both algorithms for blosum and mapping coevolution.

can see that the differences of the result of both algorithms are similar in the different generations for the mapping methods, and this is because this method mutates the coevolved sites with a strict mapping. There are no noise associated to insertions during the simulation on the coevolved sites, and gaps appear only because of the Clustal alignment.

In the Blosum method instead, there are some mutations different from those in the mapping. During the generation, the driven site is taken under a certain probability (according to the Blosum matrix and as explained in the Chapter 2, and the reason why the naïve algorithm approximates the Fix algorithm in the results is due to the fact that the noise can cause mistakes in our canFix check because probably we can not find a possible bijection for a pair of sites that we detect as mistake and we discard it from the candidates to coevolved sites.

In both methods, the graph shows that the Fix algorithm has a better average efficacy for detection against the naïve algorithm.

Now, with the same number of generations, we tested both algorithms for sequences simulated with the cluster coevolution method. The results are shown in the figure 6.7, and we can detect that often the naïve algorithm is better detecting coevolved sites against the Fix algorithm. The reason is that the cluster method of coevolution produce in the coevolved sites of the simulated sequences, frequent pairs of amino acids that violate a bijection. Under this type of simulation coevolved sites show a many-to-many correlation of amino acids according to the clusters. This increases the probability of a wrong mapping prediction in our method of detection of coevolved sites because this many frequent pairs of amino acids, that violates the bijection, are normalized obtaining a bijection that we use during this detection and increment the wrongly discarded pair of sites as not coevolved. This lack of a probable correlation of the coevolved sites, produce this false negatives that impact in the results. Moreover, when this coevolution method is used in the simulation, and we fix the coevolved sites, many misalignments appear in other sites prompting a poor whole alignment compared with the Clustal one as shown in the alignment results.

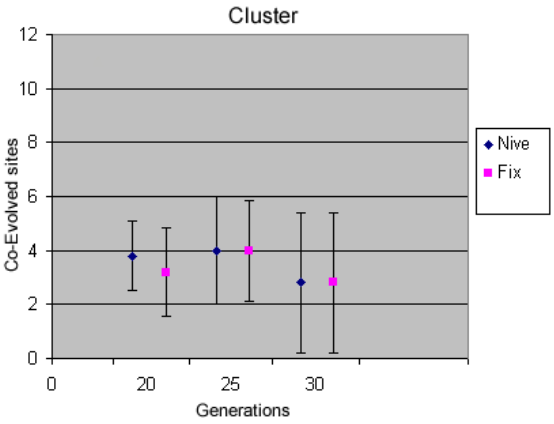


Figure 6.7: Detection using both algorithms for cluster coevolution.

We show in the table 6.2 a result of a re-alignment after fixing some of the coevolved sites detected. Here the fields Clustal and After Fix show the percentage correction of the alignment compared with the correct one.

The measure is done counting for the alignment fixed and for the Clustal alignment, how many amino acid are preserved in each site as in the correct alignment (that one that we can do from the generation of the sequences). In general after fixing, explained previously, we repair some of the coevolved sites and these sites are similar as in the correct alignment. Our method of constraints sometimes adds noise, in the sense that to obtain the mapping and the coevolved sites, it may result in switches that are not penalized by the distance or the amount of gaps to be introduced, producing an incorrect alignment for those sites close to the coevolved ones compared with the correct alignment.

In the last 15 tests, we can see how the similarity decreases, and also many times after fixing gives worst results. This is due to the fact that often for this type of tests we got many false positives and false negatives, and fixing those that are not coevolved sites or discarding those that are really coevolved sites and contain mistakes, we are not improving the alignment overall.

The good point is that we can work independently from the detection of coevolved sites to the correction of the alignment and, as proposed in the future work, improve each of these goals for a final integration of them in a whole technique that maximizes

Test Range	Mutation	Clustal	After Fix
1-5	blosum	65	72
6-10	blosum	61	63
11-15	blosum	57	59
16-20	mapping	64	68
21-25	mapping	61	66
26-30	mapping	58	59
31-35	cluster	49	44
36-40	cluster	42	45
41-45	cluster	38	36

Table 6.2: Alignment similarity with correct alignment

the detection given some alignment boundaries.

From the previous tests, we can conclude that the gain of using our method over the naïve algorithm sometimes is not so high with respect to re-aligned coevolved sites, but this gain with the combination of the power to discard True Negatives, makes the method a robust tool to detect coevolved sites with considerable efficacy and efficiency using the mutual information as a first step, the mappings to disambiguate and the constraints model presented in the previous chapter to produce the re-alignment. Also the correction of the misalignments show an improve over the Clustal alignment.

7

Conclusions and Future work

7.1 Conclusions

During this work we covered the backgrounds of an important field of bioinformatics, the sequence alignment tools, and we related the concept of coevolution with the alignment process.

We showed that often the multiple sequence alignment tools lose the information in coevolved sites and that the straightforward procedure for such alignments showed to not take into account the possible correlations between mutation on specific sites as consequence of natural pressure.

During the development of this work, the visualization tool implemented was very useful when we wanted to understand the impact of the MSA on the coevolved sites with respect to the original sequences and the correct alignment. Also this tool helped while testing different Mutual Information techniques for coevolved sites prediction and when we pinpointed how the dynamics of protein sequence evolution produce differences in the MI scores. One pending task is to analyze with real data, with previous tagged coevolved sites, if the multiple sequence alignment methods modify such sites. We expect that the results will be quite similar to the obtained in this work because we approximated the natural sequences with simulated ones and with the coevolution methods.

We think that the introduced constraint framework, including the two models for represent sequences with gap and position variables, is a great utility for future research on protein structural and functional studies, specifically when particular constraints have to be applied to preserve some relations between the amino acids at different sites and also in different sequences. Also the drawbacks presented for each of them in Chapter 5, should be taken into account as possible optimizations and perhaps as starting point of new methods.

The mapping method showed good results with the conditions established for the sequences. We believe that with more research time it can be tuned up and also a new method can be proposed to relate, using the same constraint system, all the coevolved sites integrating a detection and alignment process.

In this work we showed some improvements for the information theory techniques as a first step in the detection of coevolved sites, and also the use of constraint programming to proceed on the sequences alignments. We showed how the bioinformatics field can extend its resources to new methodologies towards the discovery of protein dynamics of evolution to finally understand the molecular process of biological organisms.

As possible future work, we would like to pinpoint some features that can be studied in deep to continue this research.

7.2 Future Work

One drawback of this work is that, during the process of detecting possible coevolved sites, we discard all those sites below the threshold T, so once we are checking the possible re-alignments we already discarded many pair of sites that may be involve some coevolved sites. This is because this model is based on the mutual information technique. Some other approach could be explored to avoid discard coevolved sites with correlated information lost because of misalignments and that present low MI score.

The main drawback of the mapping method deciding whether a pair of sites have coevolved or not is the possible uncorrect predicted mapping. Many times the frequent pairs of amino acid that looks like a perfect correlation are product of the misalignment of the alignment methods. When this is the case and we are checking for possible corrections on this sites according to this mapping, can occur that we decide that such re-alignment is not possible because a wrong predicted mapping, producing a False Negative if the sites were coevolved. This is the main reason why we say the method is not complete and is an heuristic that, as we showed in the result, is more efficient than the nïve method.

Other clear drawback of this method is the lack of re-alignment for those pairs of so close sites because, as we impose in the consistency checking, we do not re-align those sites that are not farther apart than a window W. The problem is that if we proceed re-aligning this sites, some mistakes can be introduced being detrimental to the final alignment. The same happens when two pairs of sites have some conflict in the alignment because some of the elements between the pairs are not enough distant. This is another drawback to overcome in future work, because this model just consider iteratively pairs of sites to re-align and when one site is the candidate we check for inconsistency into the window W with the already aligned pairs of sites, not re-aligning those that are in conflict. We think that the method should be extended to handle possible conflicts during the re-alignment because some pairs that are detected by our method as in conflict with other sites, are not sharing amino acids in the coevolved sites but in a not relevant site inside the window W.

Also some study is necessary to skip the limitations of analyzing just the primary protein structure, and try to use some information from the second and tertiary protein structure in the coevolution detection. With this information, we can get more features that can help to disambiguate with more precision those uncertain coevolved sites. To do this, a deeply study should be done in the biological field to understand the intramolecular relation and the amino acid evolution with respect to higher structures.

In the next subsection we pinpoint some ideas to address a possible solution to some of this drawbacks.

7.2.1 Constraint Matching

We want to address some solutions to the problem of the Chapter 6 with the wrong guessing of the mapping. Often the MSA algorithm switches amino acids from the coevolved sites to the neighbor sites and in the way our method work, it can happen that we do not consider such amino acids in the guessed mapping.

Lets consider an example in the table 7.1, as a result of a multiple alignment for sites I and J with a window of 2 sites.

And lets assume that the correct correlation coevolved amino acids for site i and j, are those pairs (P,D),(G,N). We see the Mat_i for sites (i-1,i, i+1) and Mat_j for sites (j-1,j,j+1).

If we proceed with the model of Chapter 6, the guess mapping is (P,M),(C,N),(E,D). When we search for the mapping in the 3rd sequence, we found (F,-,B) in Mat_i and (D,D,-) in Mat_j with no combination for any pair of the mapping, discarding as possible coevolved giving a false negative. Many of the mistakes in the result alignment of the

I-1	I	I+1	J-1	J	J+1
E	P	-	A	M	D
E	P	-	A	M	D
F	-	P	D	D	-
F	C	G	A	N	N
F	C	G	A	N	N
F	E	G	-	D	N

Table 7.1: Matching between Mat_i and Mat_j.

last chapter, are due to this fact, and in the detection of the coevolved sites are the false positive results, mostly appearing with the cluster coevolution method.

What we can do is to give a chance to those amino acids that were switched by the aligner in the same way we give the chance to those that actually appear in sites i and j.

One way to handle all the combinations of possible values, when variable size regions should be checked for a possible correlation, is to search for a matching between the amino acids in Mat_i and those in Mat_j. With this matching we have all the combinations of pairs between the two columns, i.e. all the tuples that can be aligned in sites I and J wether a corrections in the amino acid position is needed or not. So, from all different amino acid in the Mat_i we create a variable with a domain in the Mat_j, such that if a_i in a sequence in Mat_i and b_j in the same sequence in Mat_j then b_j is a value in the possible domain of a_i. Graphically, we can see this in figure 7.1(a) for the sequences described above, and we show the variables and the domains. In figure 7.1(b), we can see the bipartite graph, created from the variables to the domains and all the possible matchings.

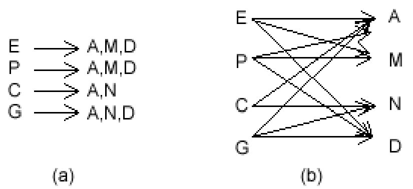


Figure 7.1: Variable, domains and matchings.

We have to relax a bit the matching according to our necessities. We can use weighted matchings to weight the distance between the elements in the matching, in the sense that if one amino acid in Mat_i matches the correlation with one in Mat_j, and we have to move them to the i and j position, we can measure the distance (in term of gapst) necessary to achieve a correct position of both of them in the coevolved sites. Also, we can have repetition of elements in the matching, in the sense that two amino acids can be matched with the same one, in case of mutations from the correlation.

This matchings can help, because for every amino acid in a variable regions close to a probable coevolved site, we can have a range of amino acids in the correlated site. In some way, we can think in a constraint system where the labelling of variables representing amino acids of the possible coevolved sites, takes value from the possible tuples of the matching, pruning values because each time a variable in a site is fixing an amino acid this condition the variable in the correlated site to take the correspondent value in the matching. Also we can measure the impact of one matching as the distance of the elements of the matching from the correlated sites.

The matchings can be generated with constraints. If we think about the amino acids in Mat_i as the variables, and those amino acids in Mat_j as the domain, we apply some constraint over the variables to get a matching. A common constraint to do this, is *AllDifferent* that we force all the variables to get a different value from the domain. For a relaxation of the strict matching, we can use the *soft AllDifferent* propagation algorithms, as proposed in [vH01]. The maximum matching using constraints, can be obtained if there exist a matching. But can be the case that the maximum perfect matching does not exist, and we have repeated elements in the domains, or many solutions to the matching. Many solutions means that we can choose a the value for that variable, and repetitions that we can give priority to some matchings. Many approaches with weighted matchings have been studied for combinatorial problems, for instance in [CL00]. The idea, is to have measure of the repetitions allowed in the matching, in our case a threshold in the amount of possible mistakes for each amino acid.

This technique can be also useful if we want to search for possible matchings in regions that are not at all conserved and that also the MI is too low. In this situations, with the method of Chapter 6 we discard them, obtaining some false negatives. But can be of interest to search possible coevolved sites in regions that are totally misaligned, and does not show any statistical signal of correlation, or in those site where the matching is spread among the neighbors sites.

More research work is needed to adapt this constraints with the alignment constraints, but we believe that this ideas can conduct to a better efficacy in the coevolution detection, working over the first protein structure.

7.2.2 Avoiding conflicts

For the conflict checking during the re-alignment process, when we can fix some coevolved site, we can try to increment the reliability of the system with a method that realize a complete re-alignment, in which we set implicit constraints that relate the different variable regions to re-align. In some way this constraints should not allow re-alignments in a site if such corrections are affecting negatively other pair of coevolved sites, and also this constraints should allow corrections only when the MI for the coevolved sites increase or a measure of the whole alignment is maximized.

We can try to get an objective function for the whole alignment and the correlation measure for the probable coevolved sites, such that we can take the decision of realign some pair of sites if we get an increment of that function. This, added to the conflict avoidance constraints, can produce a robust system that do a whole alignment maximizing the alignment and coevolved pair of sites, against our iteratively method of taking pair by pair.

One problem to work on, is an approximation of the coevolutionary best alignment we can get for a pair of sites. In the Chapter 6, we used the mapping as an heuristic of possible re-alignment in the sites to get more correlation. But now, a function measuring the partial solution for a pair of sites and the best possible correlation from that partial solution according to the amino acid present in the windows, should be calculated without exploring the whole search space.

If such function exists, we can measure the impact of re-aligning according to the constraint that we set and the impact on the whole alignment, deciding to discard such re-alignment if nothing better can be obtained from the partial correction in such sites. This technique, is a branch and bound approach, as explained in section 2.6.2 and requires the study of a good objective function that measure the impact both in the case of whole alignment and in the case of correlation among the sites.

Bibliography

[Bat05] Serafim Batzoglou. The many faces of sequence alignment. *Brief Bioinform*, 6(1):6–22, January 2005.

[Bax05] *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. Wiley-Interscience; 1 edition (July 14, 1998), ISBN-10: 0471191965, 3rd edition, 2005.

[BR97] Christian Bessiere and Jean-Charles Regin. Arc consistency for general constraint networks: Preliminary results, 1997.

[BvB98] Fahiem Bacchus and Peter van Beek. On the conversion between non-binary constraint satisfaction problems. In *Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, AAAI '98/IAAI '98, pages 311–318, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[Cho] Choco: an open source Java constraint programming library, version 2.1.1. `http://sourceforge.net/projects/choco/`.

[CL00] Yves Caseau and Francois Laburthe. Solving various weighted matching problems with constraints. *Constraints*, 5:141–160, January 2000.

[Dar95] Charles Darwin. *The Origin of Species*. Gramercy, May 1995.

[DWFG10] Russell J. Dickson, Lindi M. Wahl, Andrew D. Fernandes, and Gregory B. Gloor. Identifying and seeing beyond multiple sequence alignment errors using intra-molecular protein covariation. *PLoS ONE*, page e11082, 2010.

[Edg04] Robert C. Edgar. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5:113, 2004.

[ER64] Paul R. Ehrlich and Peter H. Raven. Butterflies and Plants: A Study in Coevolution. *Evolution*, 18(4):586–608, 1964.

[FHWE04] Hunter B. Fraser, Aaron E. Hirsh, Dennis P. Wall, and Michael B. Eisen. Coevolution of gene expression among interacting proteins. *Proceedings of the National Academy of Sciences of the United States of America*, 101(24):9033–9038, June 2004.

[FT06] Mario A. Fares and Simon A. A. Travers. A novel method for detecting intramolecular coevolution: Adding a further dimension to selective constraints analyses. *Genetics*, 173(1):9–23, May 2006.

[GOP07] Rodrigo Gouveia-Oliveira and Anders Pedersen. Finding coevolving amino acid residues using row and column weighting of mutual information and multi-dimensional amino acid representation. *Algorithms for Molecular Biology*, 2(1):12, 2007.

[HH92] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, 89(22):10915–10919, November 1992.

[LBB$^+$07] M. A. Larkin, G. Blackshields, N. P. Brown, R. Chenna, P. A. McGettigan, H. McWilliam, F. Valentin, I. M. Wallace, A. Wilm, R. Lopez, J. D. Thompson, T. J. Gibson, and D. G. Higgins. Clustal w and clustal x version 2.0. *Bioinformatics*, 23(21):2947–2948, November 2007.

[MB99] S. Harrell M. Beals, L. Gross. Amino Acid Distribution. http://www.tiem.utk.edu/bioed/webmodules/aminoacid.htm, 1999. [Online; accessed 2010].

[MFDW98] Burkhard Morgenstern, Kornelie Frech, Andreas W. M. Dress, and Thomas Werner. DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294, 1998.

[NGR10] Scott L. Nuismer, Richard Gomulkiewicz, and Benjamin J. Ridenhour. When is correlation coevolution? *The American Naturalist*, 175(5):525–537, May 2010.

[NH03] Pauline C. Ng and Steven Henikoff. SIFT: predicting amino acid changes that affect protein function. *Nucleic Acids Research*, 31(13):3812–3814, 2003.

[NW70] Saul Ben Needleman and Christian Dennis Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[Pro95] Patrick Prosser. Forward checking with backmarking. In *Constraint Processing, Selected Papers*, pages 185–204, London, UK, 1995. Springer-Verlag.

[RBW$^+$] Francesca Rossi, Peter Van Beek, Toby Walsh, Thom Fruhwirth, Laurent Michel, and Christian Schulte. Handbook of constraint programming.

[Sha01] C. E. Shannon. A mathematical theory of communication. *SIGMOBILE Mob. Comput. Commun. Rev.*, 5:3–55, 2001.

[Sin95] Moninder Singh. Path consistency revisited. In *In Proceedings of IEEE ICTAI-95, Washington D.C*, 1995.

[SN87] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, July 1987.

[SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, March 1981.

[TPP99] Julie D. Thompson, Frederic Plewniak, and Olivier Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic Acids Research*, 27(13):2682–2690, 1999.

[vH01] Willem-Jan van Hoeve. The all_different constraint: A survey. *CoRR*, cs.PL/0105015, 2001.

[WJ94] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology : a journal of computational molecular cell biology*, 1(4):337–348, 1994.

[WP05] Zhengyuan O. Wang and David D. Pollock. Context dependence and coevolution among amino acid residues in proteins. *Methods in enzymology*, 395:779–790, 2005.