Albert-Ludwigs-Universität Freiburg

Diploma Thesis

Efficient solving of alignment-problems with side conditions using constraint techniques

Sebastian Barth
May 10, 2007

Fakultät für Angewandte Wissenschaften
Institut für Informatik
Lehrstuhl für Bioinformatik

Gutachter Prof. Dr. Rolf Backofen
Prof. Dr. Christoph Scholl

ERKLÄRUNG

Hiermit erkläre ich, dass ich diese Abschlussarbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Abschlussarbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Ort, Datum Unterschrift

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre fachliche und persönliche Unterstützung zum Gelingen dieser Diplomarbeit beigetragen haben.

Prof. Dr. Rolf Backofen danke ich für die Vergabe einer herausfordernden und überaus interessanten Themenstellung, die mir einen tieferen Einblick in die Bioinformatik ermöglichte.

Prof. Dr. Christoph Scholl danke ich für seine Bemühungen.

Meinem Betreuer Dr. Sebastian Will danke ich ganz herzlich für zahlreiche fruchtbare Diskussionen und viele Anregungen.

Große Dankbarkeit empfinde ich gegenüber meiner geliebten Frau, meiner Mutter und meinem Großvater für ihre liebevolle Unterstützung.

Contents

1 Introduction **7**
- 1.1 Motivation . . . 7
- 1.2 Related Works . . . 9
- 1.3 Overview . . . 9

2 Fundamentals **11**
- 2.1 Definition of an alignment . . . 11
- 2.2 Further constraints for alignments . . . 12
  - 2.2.1 Anchor constraints . . . 12
  - 2.2.2 Precedence constraints . . . 12
  - 2.2.3 Aligned segments . . . 13
  - 2.2.4 Sequence structure alignment . . . 13

3 The approach of dynamic programming **15**
- 3.1 Alignment using dynamic programming . . . 15
- 3.2 Extended alignment problems . . . 19
  - 3.2.1 Anchor constraints . . . 19
  - 3.2.2 Precedence constraints . . . 20
  - 3.2.3 Aligned segments . . . 21

4 Cluster tree decomposition, elimination **27**
- 4.1 A constraint model for sequence alignment . . . 27
- 4.2 Definition of a cluster tree . . . 29
- 4.3 Cluster tree elimination . . . 29
- 4.4 Using CTD/CTE for solving alignments . . . 30
- 4.5 Extended alignment problems . . . 33
  - 4.5.1 Anchor constraints . . . 34
  - 4.5.2 Precedence constraints . . . 34
  - 4.5.3 Aligned segments . . . 34
- 4.6 Sequence structure alignment . . . 36

4.6.1 Cluster-tree decomposition for a sequence structure alignment . . . 37
4.6.2 Attributes of the decomposition . . . 39
4.6.3 Cluster-tree elimination for sequence structure alignments . . . 39
4.6.4 Construction of a cluster tree for sequence structure alignment . . . 41
4.7 Combining several constraints . . . 42
4.7.1 Multiple constraints of one type . . . 42
4.7.2 Different multiple constraints combined . . . 45

5 Progressive alignment 47
5.1 Aligning three sequences . . . 47
5.1.1 Aligning an alignment of two sequences with a sequence 48
5.2 Progressive alignment . . . 49
5.2.1 How to align alignments using CTD and CTE . . . 50
5.3 Combining with constraints . . . 51
5.3.1 Induced constraints . . . 52
5.3.2 Following the guide tree . . . 54

6 Multiple alignments using CTD/CTE 55
6.1 Aligning three sequences . . . 55
6.1.1 A constraint model for three sequences alignment . . . 55
6.1.2 Cluster tree decomposition . . . 58
6.1.3 Cluster tree elimination . . . 58
6.2 Aligning m sequences . . . 59
6.3 Adding constraints . . . 60
6.3.1 Anchor constraints . . . 60
6.3.2 Precedence constraints . . . 60

7 Conclusion 61
7.1 Further perspectives . . . 62

Chapter 1

Introduction

1.1 Motivation

Since the discovery of the structure of the genome, RNA and DNA, we try to understand its code. A DNA consists of two sequences bound together by base pairs of the nitrogenous bases cytosine, guanine, adenine and thymine forming hydrogen bounds. RNA is a single sequence of the nitrogenous bases cytosine, guanine, adenine and uracil (see fig. 1.1[1]). A RNA sequence can form hydrogen bounds with itself. This forms the secondary structure, a general three dimensional structure. Proteins also exist as sequences.

Many problems and open questions of modern microbiology demand the comparison of RNA, DNA or protein sequences. The computational effort in performing these calculations is high and microbiology urges the development of efficient tools. An important requirement is the capability of these tools to deal with certain constraints. Examples might be a specified number of matches of two compared sequences within certain regions or a consideration of the secondary structure of a proteine [3, 13, 15, 22, 19].

Subject of this thesis is the efficient alignment of sequences, where we focus especially on different constraints on the alignment and how to combine these constraints, even in multiple alignments.

[1] taken from *http://de.wikipedia.org/wiki/Bild:RNA-comparedto-DNA.PNG*

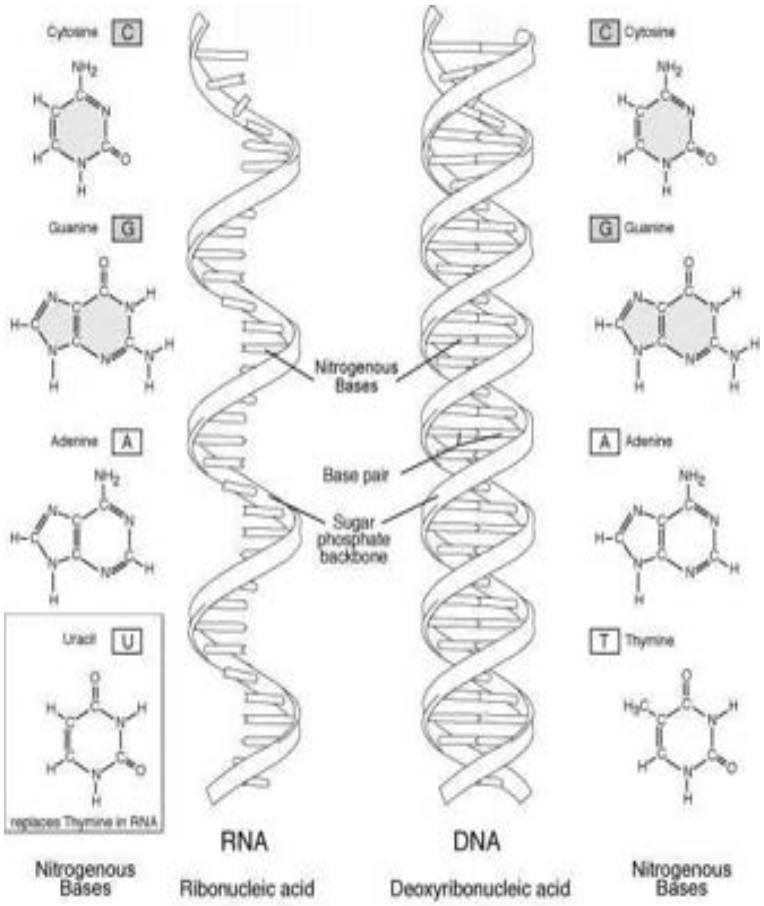

Figure 1.1: RNA compared to DNA

1.2 Related Works

A short overview of a computation of an alignment using dynamic programming can be found at [8]. The method of cluster tree decomposition and elimination is described in detail by [17]. [21] extends this approach to solve alignments and to handle with constraints. Multiple alignments are discussed in [7, 4, 14, 18, 20, 6], many of them use progressive alignment.

1.3 Overview

With a short theoretical introduction in chapter 2, we will start with standard dynamic programming approach for computing alignments inspired by [8] (chapter 3). Even though dynamic programming is an important method in bioinformatics it is hard to find an enlargement that can deal with constraints. We will show some possibilities for computing alignments with anchor, precedence and aligned segments constraints by using dynamic programming.

Chapter 4 is about cluster tree decomposition and elimination. The chapters 4.1 to 4.5 are based a lot on the the work [21]. In these chapters a constraint model for sequence alignments and the cluster tree decomposition, elimination are developed, as well as anchor, precedence and aligned segments constraints are added. [21] describes briefly how to deal with secondary structure alignments. This description is extended, formalised and a method to construct a decomposition for secondary structure alignment of a cluster tree decomposition for an alignment without any secondary structure is shown. Then we will concentrate on how to deal with combinations of several constraints, even if these are of a different type.

Chapter 5 gives an heuristic approach for solving multiple alignments using cluster tree decomposition and elimination. At the beginning we will compute an alignment on three sequences. After a short introduction on progressive alignment we show how to compute a multiple alignment using progressive alignment and continue by handling with anchor and precedence constraints on multiple alignments.

After computing the optimal alignment between three sequences, we show in chapter 6 an approach for finding an optimal multiple alignment. This approach is exponential in the number of sequences that should be aligned

and uses cluster tree decomposition and elimination. We close this work on adding anchor and precedence constraints on multiple alignments.

Chapter 2

Fundamentals

2.1 Definition of an alignment

Considering two strings we are interested in the degree of similarity if we compare them. An important term in the comparison of strings is the alignment.

Definition 2.1.1 *An alignment $\mathcal{A}$ of two strings $\mathcal{S}_1$ and $\mathcal{S}_2$ over the alphabet Σ is a subset of $\{1,\ldots,n\}\cup\{-\}\times\{1,\ldots,m\}\cup\{-\}$, where n is the number of characters in $\mathcal{S}_1$ and m the one in $\mathcal{S}_2$ (see [1]). $\mathcal{A}$ has to satisfy the following conditions:*

- $(-,-)\notin\mathcal{A}$
- $\forall(i,j)\in\mathcal{A}\cap\{1,\ldots,n\}\times\{1,\ldots,m\}$*: i and j appear only once in $\mathcal{A}$.*
- $\forall(i',j'),(i,j)\in\mathcal{A}\cap\{1,\ldots,n\}\times\{1,\ldots,m\} : i'<i\Leftrightarrow j'<j$*. Any pairs in $\mathcal{A}$ are non crossing.*

In an alignment each character of both strings is allocated to an other character in the other string or to a gap, represented by $'-'$. If aligning DNA-, resp. RNA-sequences, we can define Σ as: $\Sigma=\{a,c,g,t\}\cup\{-\}$, resp. $\Sigma=\{a,c,t,u\}\cup\{-\}$.

In order to be able to measure the similarity we need a *scoring scheme*: Let $\delta(x)\rightarrow\mathbb{R}$ be a cost-function, where $x=(x_1,x_2)\in\Sigma\times\Sigma$. δ measures the quality of aligning an element of $\mathcal{S}_1$ with an element of $\mathcal{S}_2$[1]. The score of $\mathcal{A}$ is:

[1] if aligning RNA, it is a measure for the quality of aligning two unpaired nucleotides

Definition 2.1.2 *Score of an alignment*

$$score(\mathcal{A}) = \sum_{x \in \Sigma \times \Sigma} \delta(x)$$

For example, we can allocate a positive value for δ if the two characters differ and 0 otherwise. This way we receive score$(\mathcal{A}) = 0$ for equivalent strings a, low value for similar strings and a high value for different ones.

Example 2.1.1 *Editing distance*
As an other interpretation, we consider $\min score(\mathcal{A})$*, where* $\mathcal{A}$ *is any alignment of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$*, equal to the editing distance. If we determine* $\delta(s_1, s_2) = 1$*, where* s_1 *is a character of* $\mathcal{S}_1$ *and* s_2 *one of* $\mathcal{S}_2$*, as the costs of a conversion of* s_1 *into* s_2 *if* $s_1 \neq s_2$ *and* $\delta(s_1, s_2) = 0$ *if* $s_1 = s_1$ *and* $\delta(s_1, -) = 1$ *resp.* $\delta(-, s_2) = 1$ *as the costs of deletion of* s_1 *in* $\mathcal{S}_1$ *resp. a insertion of* s_2 *into* $\mathcal{S}_1$*, the editing distance of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *expresses the minimal costs, with respect to* δ *and the number of operations to transform* $\mathcal{S}_1$ *into* $\mathcal{S}_2$*.*

With a high value of δ for matchings and lower, possibly negative, for mismatches, we can formulate the following definition.

Definition 2.1.3 *Optimal alignments*
Let $\mathcal{A}^*$*, resp* $\mathcal{A}^*(\mathcal{S}_1, \mathcal{S}_2)$ *denote the optimal alignments of two strings regarding the score. I.e. the alignments* $\mathcal{A}^*$ *have to fulfill the following equation:*

$$score(\mathcal{A}^*) = \max_{\mathcal{A} \text{ of } \mathcal{S}_1, \mathcal{S}_2} score(\mathcal{A})$$

Definition 2.1.4 *Optimal alignment of subsequences*
$\mathcal{A}^*(x_i, y_j)$*, where* $x_1 \dots x_i$ *and* $y_1 \dots y_j$ *are substrings of* $\mathcal{S}_1$ *and* $\mathcal{S}_2$ *beginning at the first position and ending at position* i*, resp.* j*, denote the optimal alignments of the substrings* $x_1 \dots x_i$ *and* $y_1 \dots y_j$*.*

2.2 Further constraints for alignments

2.2.1 Anchor constraints

This simple constraint forces the element at position i in the first sequence to be aligned with an element at a position less or equal to j in the second one.

2.2.2 Precedence constraints

Here an element at position i in the first sequence has to be aligned left (resp. right) to the position of j'th element in the second sequence.

2.2.3 Aligned segments

The aligned segments extend the normal alignments by the following constraint: Given two sequences a and b and the segments $k, \ldots, k'$ of sequence a, $l, \ldots, l'$ of sequence b. A certain percentage of the elements in segment $k, \ldots, k'$ have to be matched with elements of segment $l, \ldots, l'$.

2.2.4 Sequence structure alignment

In contrast to the other problems observed so far, we now also consider the secondary structure of our sequences [21].

Let two structures P_a and P_b be given with $P_a \subset \{1, \ldots, n\} \times \{1, \ldots, n\}$ over a sequence $A = \{a_1, \ldots, a_n\}$ and $P_b \subset \{1, \ldots, m\} \times \{1, \ldots, m\}$ over the sequence $B = \{b_1, \ldots, b_m\}$. A pair $(i_l, i_r) \in P_a$ (resp. $(j_l, j_r) \in P_b$), $l < r$, expresses dependencies in sequence A between the bases a_l and a_r (resp. b_l, b_r in B), as for instance in the case of base pairings. For any two pairs $(i_{l_1}, i_{r_1}), (i_{l_2}, i_{r_2}) \in P_a$ (resp. $(j_{l_1}, j_{r_1}), (j_{l_2}, j_{r_2}) \in P_b$) it holds: $i_{l_1} \neq i_{l_2}$ and $i_{r_1} \neq i_{r_2}$ (resp. $j_{l_1} \neq j_{l_2}, j_{r_1} \neq j_{r_2}$)

Let $\omega : \{1, \ldots, n\} \times \{1, \ldots, n\} \times \{1, \ldots, m\} \times \{1, \ldots, m\} \to \mathbb{R}$ be a function that weights a structural element of P_a in comparison to one of P_b.

In extension to def. 2.1.2 we can define the score of two aligned structures as following:

Definition 2.2.1 *The score of two aligned sequence structures*
Let P_a and P_b be two structures of the sequences A and B. Let $\mathcal{A}$ be an alignment of A and B. The score of an alignment $\mathcal{A}_{P_a,P_b}$ of two structures is defined as

$$score(\mathcal{A}_{P_a,P_b}) = score(\mathcal{A}) + \sum_{\substack{(i_l,i_r)\in P_a,(j_l,j_r)\in P_b \\ (i_l,j_l)\in\mathcal{A},(i_r,j_r)\in\mathcal{A}}} \omega(i_l, i_r; j_l, j_r)$$

A special type of secondary structures are pseudoknots and pseudoknot-free structures.

Definition 2.2.2 *Pseudoknot-free structure [5]*
A structure P over a sequence is called pseudoknot-free iff for any two pairs $(i_l, i_r), (j_l, j_r) \in P$, with $i_l < j_l$, holds

- $i_l < i_r < j_l < j_r$ *or*

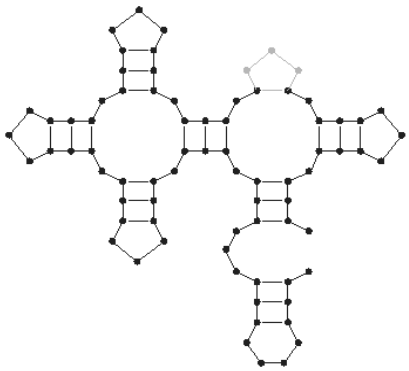

Figure 2.1: Example of a pseudoknot-free structure

- $i_l < j_l < j_r < i_r$

A visualisation of a pseudoknot-free structure is given in figure 2.1[2].

[2]taken from [5]

Chapter 3

The approach of dynamic programming

Dynamic programming is one of the very basic programming principles in computer science. It often allows to solve combinatorial optimisation problems over a search space of exponential size in polynomial space and time [11]. Dynamic programming was formalised in the early 1950s by the mathematician Richard Bellman.

Definition 3.0.3 *Dynamic algorithm*
A dynamic algorithm is distinguished by four different parts [8]:

- *A recursive definition of the optimal score of the corresponding optimisation problem.*
- *A dynamic programming matrix for remembering the optimal scores of subproblems in order to avoid multiple computation of the same values.*
- *A bottom-up approach for filling the matrix which contains the partial solutions. The smallest subproblems will be solved first, the others not until the needed values are accessible.*
- *A traceback algorithm that traverses the filled matrix in order to find the structure of the optimal solution that gave the optimal score.*

3.1 Alignment using dynamic programming

If we want to receive an optimal alignment of the two segments $\mathcal{S}_1$ and $\mathcal{S}_2$ we have to consider how we can alter our four parts adequately. Let us assume,

we want to transform $\mathcal{S}_1$ into $\mathcal{S}_2$.

Let x_i, $1 \leq i \leq m$, be a character[1] at position i of $\mathcal{S}_1$ and y_j, $1 \leq j \leq n$, one of $\mathcal{S}_2$, where m, n are the number of characters in $\mathcal{S}_1$ resp. $\mathcal{S}_2$. If we want to receive the optimal score, $score(\mathcal{A}^*(x_i, y_j))$ of an alignment of the subsequences $x_1 \dots x_i$ of $\mathcal{S}_1$ and $y_1 \dots y_j$ of $\mathcal{S}_2$ there are three different cases for a recursive definition to consider:

- x_i and y_j will be aligned, i.e. $x_i = y_j$ or we will replace x_i by x_j. The pair (x_i, y_j) will appear in our alignment of the subsequences.
- x_i will be aligned with $'-'$, i.e. x_i will be deleted in the subsequence of $\mathcal{S}_1$.The pair $(x_i, -)$ will appear in our alignment of the subsequences.
- y_j will be aligned with $'-'$, i.e. y_j will be inserted into the subsequence of $\mathcal{S}_1$ The pair $(-, y_j)$ will appear in our alignment of the subsequences.

That leads to the following recursive equation

$$score(\mathcal{A}^*(x_i, y_j)) = \max \begin{cases} score(\mathcal{A}^*(x_{i-1}, y_{j-1})) + \delta(x_i, y_j) \\ \text{for replacing } x_i \text{ by } y_j \\ score(\mathcal{A}^*(x_{i-1}, y_j)) + \delta(x_i, -) \\ \text{for deleting } x_i \text{ in } \mathcal{S}_1 \\ score(\mathcal{A}^*(x_i, y_{j-1})) + \delta(-, y_j) \\ \text{for inserting } y_j \text{ in } \mathcal{S}_1 \end{cases} . \tag{3.1}$$

If we introduce constant gap costs γ, we receive

$$score(\mathcal{A}^*(x_i, y_j)) = \max \begin{cases} score(\mathcal{A}^*(x_{i-1}, y_{j-1})) + \delta(x_i, y_j) \\ score(\mathcal{A}^*(x_{i-1}, y_j)) + \gamma \\ score(\mathcal{A}^*(x_i, y_{j-1})) + \gamma \end{cases} . \tag{3.2}$$

Our dynamic programming matrix $\mathcal{M}$ will store the scores. At position $\mathcal{M}(i, j)$ the $score(\mathcal{A}^*(x_i, y_j))$ will be stored . $\mathcal{M}$ is of size $O(nm)$ since we need entries for every combination $i \times j$. In addition we need the column $\mathcal{M}(0, j)$ and the row $\mathcal{M}(i, 0)$. The matrix will be initialised with the values $\mathcal{M}(0, 0) = 0$ and $\mathcal{M}(i, 0) = \sum_{k=1}^{i} \delta(x_i, -)$, $\mathcal{M}(0, j) = \sum_{l=1}^{j} \delta(-, y_j)$ respectively $\mathcal{M}(i, 0) = i\gamma$, $\mathcal{M}(0, j) = j\gamma$ in case of constant gap costs.

[1] A protein base for RNA resp. DNA

Once we initialised $\mathcal{M}$ we can fill the matrix bottom up by examining for each $\mathcal{M}(i,j)$ the three adjacent entries $\mathcal{M}(i-1,j-1)$, $\mathcal{M}(i-1,j)$, $\mathcal{M}(i,j-1)$ and choosing the maximal value. We can begin with $\mathcal{M}(1,1)$ and complete the first row, afterwards we do the same with the second one and so on.

Since each cell $\mathcal{M}(i,j)$ contains the optimal alignment score of the subsequences $x_1 \dots x_i$ and $y_1 \dots y_j$ of $\mathcal{S}_1$ and $\mathcal{S}_2$, the score of the optimal alignment $\mathcal{A}^*$ of $\mathcal{S}_1$ and $\mathcal{S}_2$ is $score(\mathcal{A}^*) = \mathcal{M}(n,m)$.

Now we know the score of an optimal alignment, but we are interested in the structure, too. Due to our recursive equation, we can be sure that $\mathcal{M}(m,n)$ refers to at least one of the adjacent cells $\mathcal{M}(m-1,n)$, $\mathcal{M}(m,n-1)$ or $\mathcal{M}(m-1,n-1)$. As seen in equation 3.1 these entries can be interpreted as a replacement, insertion or deletion of a character resp. a protein base in our sequence $\mathcal{S}_1$. Since $\forall i,j$, where $0 < i,j \leq n,m$ the entry in $\mathcal{M}(i,j)$ is, due to equation 3.1, only influenced by one of the cells $\mathcal{M}(i-1,j)$, $\mathcal{M}(i,j-1)$ or $\mathcal{M}(i-1,j-1)$[2] we only need to consider this tree adjacent cells to find the structure of the optimal alignment via traceback. If we recompute the entry in $(M)(i,j)$ we know which cell is responsible for $\mathcal{M}(i,j)$. This gives the traceback path. The corresponding structure of the alignment and the adequate action for transforming $\mathcal{S}_1$ into $\mathcal{S}_2$ is given by the traceback path. Figure 3.1[3] shows an example. In this example a scoring scheme of $+5$ for a match, -2 for a mismatch and -6 for each insertion or deletion is used. The traceback path is shown red and the arrowheads are 'traceback pointers' indicating which of the three cases were optimal for reaching each cell.

The storage capacity needed for this algorithm is $O(mn)$ since this is the dimension of our dynamic programming matrix $\mathcal{M}$. The filling of one cell takes constant time, therefore the filling of the complete matrix is also in $O(mn)$ time. Since we do the same computation for the traceback as before when we filled $\mathcal{M}$ and we consider each cell at most 2 times[4] it is also in the time complexity $O(mn)$. Therefore the time complexity of the complete algorithm is in $O(mn)$.

[2] Of corse, the optimal score can be obtained by one, two or even all three adjacent cells, but an algorithm would only take one and we are only interested in one representative of all optimal alignments.

[3] taken from [8]

[4] $\mathcal{M}(i-1,j-1)$ will be considered twice if we chose $\mathcal{M}(i-1,j)$ or $\mathcal{M}(i,j-1)$.

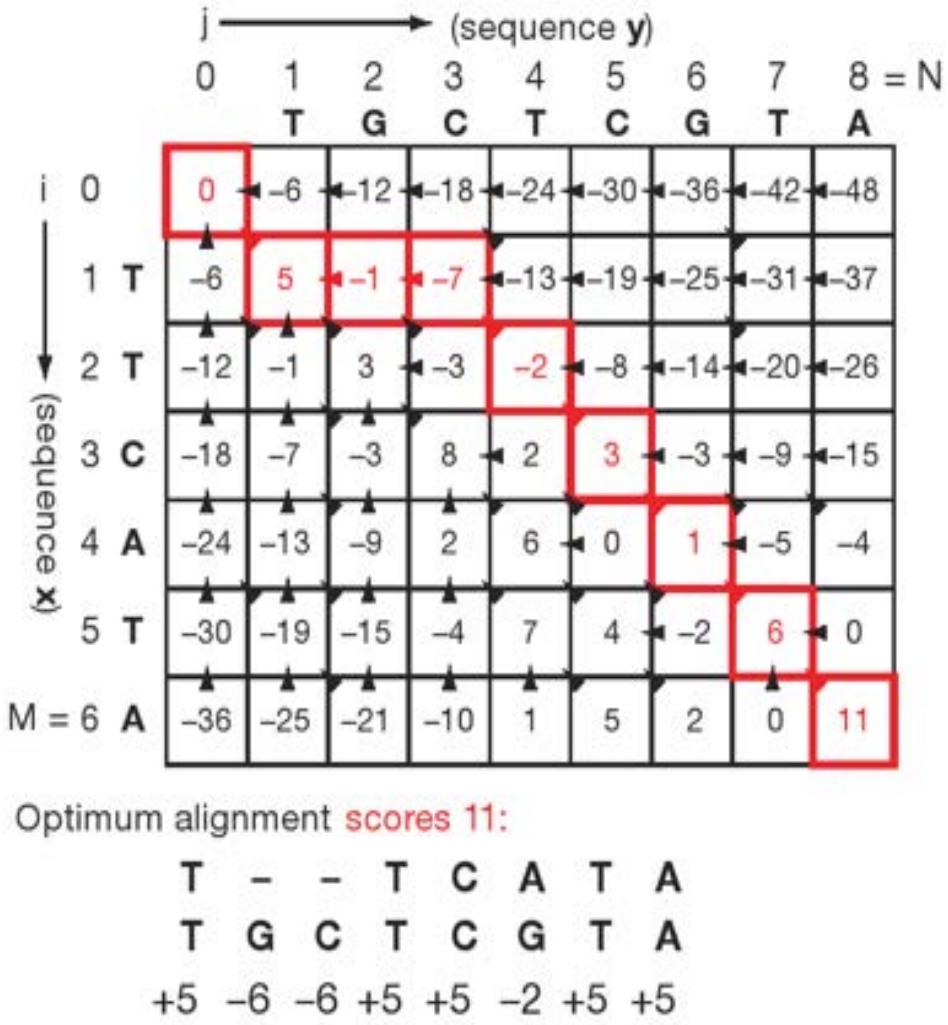

Figure 3.1: Dynamic programming matrix of an alignment

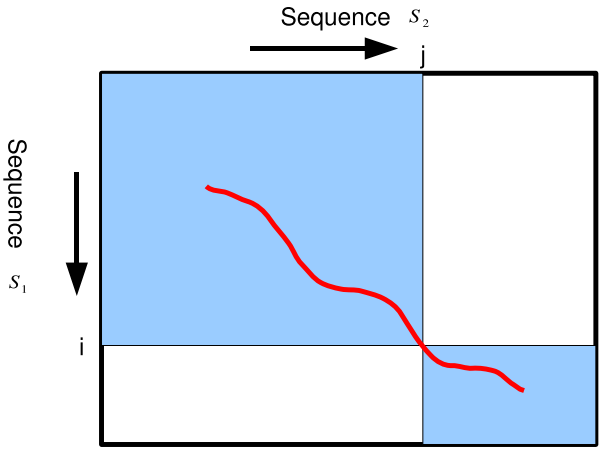

Figure 3.2: Dynamic programming matrix with one anchor constraint
Traceback path is marked red.

3.2 Extended alignment problems

In this chapter we will discuss, how we can deal with further constraints using algorithms based on the dynamic programming approach discussed before.

3.2.1 Anchor constraints

If we want to force character x_i in sequence $\mathcal{S}_1$ to align with y_j in $\mathcal{S}_2$ it suffices to have a look at the alignments of the subsequences $x_1, \ldots, x_i$, $y_1, \ldots, y_j$ and $x_{i+1}, \ldots, x_m$, $y_{j+1}, \ldots, y_n$.

Since the traceback path of our alignment has to pass $\mathcal{M}(i,j)$ (see figure 3.2) we fill our dynamic programming matrix as mentioned before for all cells $\mathcal{M}(k,l)$, with $0 \leq k \leq i$ and $0 \leq l \leq j$. We consider $\mathcal{M}(i,j)$ as if it would be a start point of a normal traceback path. In this way we obtain the dynamic programming matrix for the alignment of the first subsequences. For $i \leq k \leq m$ and $j \leq l \leq n$ we fill $\mathcal{M}$ in the following way:

- $\mathcal{M}(i,l) = \mathcal{M}(i,j) + \sum_{o=i+1}^{l} \delta(-, y_o),\ j \leq l \leq n$
- $\mathcal{M}(k,j) = \mathcal{M}(i,j) + \sum_{p=j+1}^{k} \delta(x_p, -),\ i \leq k \leq m$
- $\mathcal{M}(k,l)$, where $k \neq i$ and $l \neq j$, will be filled using the same bottom up mechanism like the standard alignment (equation 3.1).

For $0 \leq k \leq i$ and $j < l \leq n$ or $i < k \leq m$ and $0 \leq l \leq j$ we can fill $\mathcal{M}(k,l)$ with $-\infty$ because these areas are not of interest since our traceback path won't pass them.

Once we received $\mathcal{M}(m,n)$, we can compute our traceback path as we did in chapter 3.1. The time and space complexity of this approach is in $O(mn)$.

3.2.2 Precedence constraints

Here we distinguish two different cases depending on if the i'th element in segment $\mathcal{S}_1$ has to be aligned to an element in segment $\mathcal{S}_2$ at a position smaller or bigger position j. Let x be the mentioned position of the element in $\mathcal{S}_2$.

1. **Element i in $\mathcal{S}_1$ is aligned *left* to element j in $\mathcal{S}_2$.**
   I.e. $0 \leq x < j$ (see figure 3.3). In this case, the traceback path can't pass the cells $\mathcal{M}(k,l)$ with $0 \leq k < i$ and $j \leq l \leq n$. These cells can be filled with $-\infty$. The rest of $\mathcal{M}(k,l)$ will be filled as following:

   - For $0 \leq k \leq m$ and $0 \leq l < j$:
     The cells of $\mathcal{M}$ can be filled as if we compute a standard alignment for the subsequences $x_1, \ldots, x_m$ of $\mathcal{S}_1$ and $y_1, \ldots, y_m$ of $\mathcal{S}_2$.
   - For $k = i$ and $j \leq l \leq n$:
     $\mathcal{M}(k,l) = \mathcal{M}(i, j-1) + \sum_{p=j}^{l} \delta(x_p, -)$
   - For $i < k \leq m$ and $j \leq l \leq n$:
     $\mathcal{M}(k,l)$ can be computed convenient (formula 3.1) by using the boundary values $\mathcal{M}(k, j-1)$ and $\mathcal{M}(i,l)$.

2. **Element i in $\mathcal{S}_1$ is aligned *right* to element j in $\mathcal{S}_2$.**
   I.e. $j < x \leq n$ (see figure 3.4). Like above, we can fill certain cells of $\mathcal{M}$ with $-\infty$. In this case they are the cells $\mathcal{M}(k,l)$ with $i < k \leq m$ and $0 \leq l \leq j$. For filling the rest of $\mathcal{M}(k,l)$ we have to consider the following cases:

   - For $0 \leq k \leq i$ and $0 \leq l \leq n$:
     $\mathcal{M}(k,l)$ can be computed as if we consider an alignment for the subsegments $x_1, \ldots, x_i$ of $\mathcal{S}_1$ and $y_1, \ldots, y_n$ of $\mathcal{S}_2$.
   - For $i < k \leq m$ and $l = j + 1$:
     $\mathcal{M}(k,l) = \mathcal{M}(i, j+1) + \sum_{o=i+1}^{k} \delta(-, y_o)$

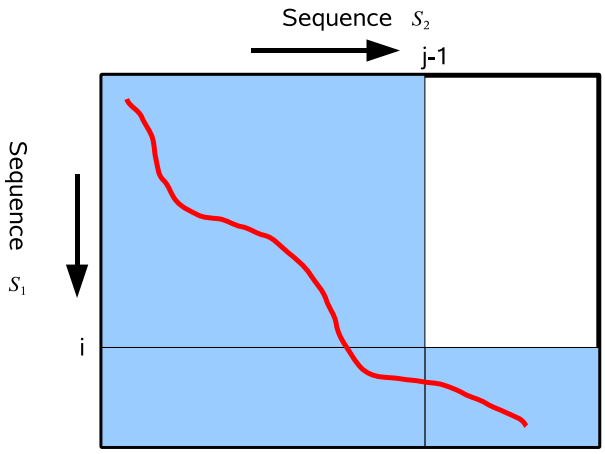

Figure 3.3: Dynamic programming matrix with one precedence constraint, 'left'

Traceback path is marked red.

- For $i < k \leq m$ and $j + 1 < l \leq n$:
  In this case we can compute $\mathcal{M}(k, l)$ using the recursive formula 3.1 and the starting values at the boundary given by the two cases above.

It is quite obvious that these approaches take $O(mn)$ time and space complexity.

3.2.3 Aligned segments

Let $|k|$ be the number of elements in the segment $x_k, \ldots, x_{k'}$ of $\mathcal{S}_1$ and $|l|$ be the number of elements in $y_l, \ldots, y_{l'}$ of $\mathcal{S}_2$. Let x be the number of characters in the segment $x_k, \ldots, x_{k'}$ of $\mathcal{S}_1$ which have to be aligned with characters in the segment $y_l, \ldots, y_{l'}$ of $\mathcal{S}_2$ in order to satisfy the constraint in chapter 2.2.3. Let $|l| \geq x$, i.e. $y_l, \ldots, y_{l'}$ contains enough elements to fulfill the constraint. Due to the recursive equations 3.1 we need at least x diagonal steps in the traceback path. A closer look on the square $\mathcal{M}' = (\mathcal{M}(k, l), \mathcal{M}(k', l), \mathcal{M}(k', l'), \mathcal{M}(k, l'))$ shows that a possible traceback path can't pass all cells of $\mathcal{M}$ within this square. We can only legally enter the square at positions in $\{\mathcal{M}(k, l), \mathcal{M}(k + 1, l), \ldots, \mathcal{M}(k' - x, l)\}$ or $\{\mathcal{M}(k, l), \mathcal{M}(k, l + 1), \ldots, \mathcal{M}(k, l' - x)\}$ and leave it at $\{\mathcal{M}(k + x, l'), \mathcal{M}(k + x + 1, l'), \ldots, \mathcal{M}(k', l')\}$ or $\{\mathcal{M}(k', l + x), \mathcal{M}(k', l + x + 1), \ldots, \mathcal{M}(k', l')\}$ (see

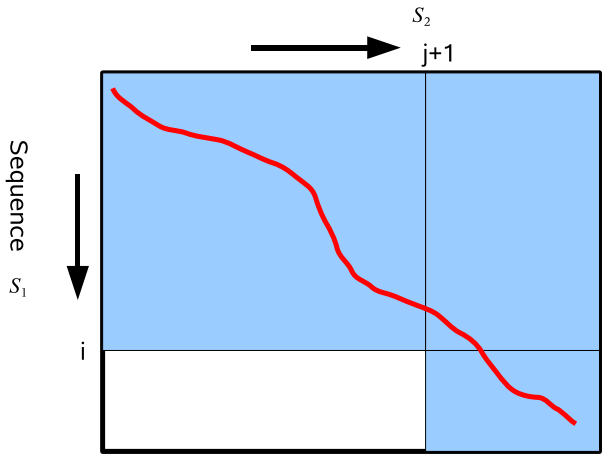

Figure 3.4: Dynamic programming matrix with one precedence constraint, 'right'

Traceback path is marked red.

figure 3.5). Let $\mathcal{I}'_1, \ldots \mathcal{I}'_4$ be coordinates of cells of $\mathcal{M}$ lying on the border of $\mathcal{M}'$. Let them be defined as following.

- $\mathcal{I}'_1 = (k' - x, l)$
- $\mathcal{I}'_2 = (k', l + x)$
- $\mathcal{I}'_3 = (k + x, l')$
- $\mathcal{I}'_4 = (k, l' - x)$

Let $\mathcal{L}_1$ be the straight line passing $\mathcal{M}(\mathcal{I}'_1), \mathcal{M}(\mathcal{I}'_2)$ and $\mathcal{L}_2$ the one passing $\mathcal{M}(\mathcal{I}'_3), \mathcal{M}(\mathcal{I}'_4)$. It holds that $\mathcal{L}_1$ and $\mathcal{L}_2$ pass exactly x cells within the square $\mathcal{M}'$. If $\mathcal{L}_1$ or $\mathcal{L}_2$ is part of the traceback path, this path has exactly x parts on the diagonal and no part parallel to the borders of $\mathcal{M}$. I.e. exactly x elements of the considered segments are aligned. A legal traceback path can't pass cells within $\mathcal{M}'$ but below $\mathcal{L}_1$ or above $\mathcal{L}_2$: such a path wouldn't consist of enough diagonal elements.

Let $\mathcal{I}_1, \ldots, \mathcal{I}_4$ be the projection of $\mathcal{I}'_1, \ldots, \mathcal{I}'_4$ on the margin of $\mathcal{M}$. $\mathcal{I}_1, \ldots, \mathcal{I}_4$ represent the following coordinates:

- $\mathcal{I}_1 = (k' - x, 0)$
- $\mathcal{I}_2 = (m, l + x)$

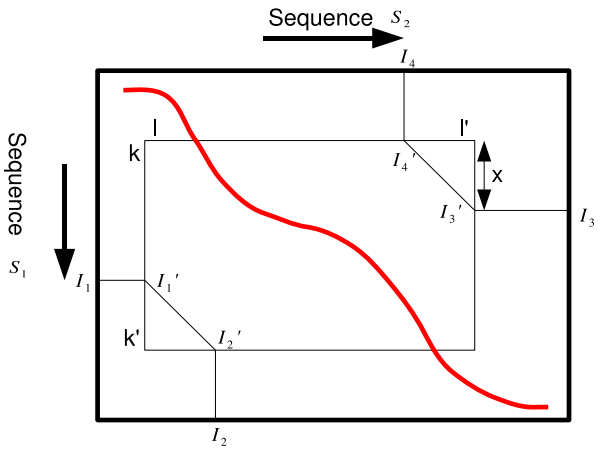

Figure 3.5: Dynamic programming matrix with aligned segment constraint Traceback path is marked red.

- $\mathcal{I}_3 = (k + x, n)$
- $\mathcal{I}_4 = (0, l' - x)$

Let $\mathcal{P}$ be the area limited by the polygon

$$((0,0), \mathcal{I}_1, \mathcal{I}_1', \mathcal{I}_2', \mathcal{I}_2, (m,n), \mathcal{I}_3, \mathcal{I}_3', \mathcal{I}_4', \mathcal{I}_4).$$

For geometrical reasons, due to our recursive definition (eq. 3.1), a possible tracebackpath can't pass cells outside $\mathcal{P}$. The value of these cells can be set to $-\infty$.

We will initialise $\mathcal{M}(i,j)$ as following.

- $\mathcal{M}(0,0) = 0$
- $\mathcal{M}(i,0) = i\delta(x_i,-),\ 0 < i \leq k' - x$
- $\mathcal{M}(0,j) = j\delta(-,y_j),\ j < j \leq l' - x$
- $\mathcal{M}(k'-x,j) = \mathcal{M}(k'-x,0) + \sum_j \delta(-,y_j),\ 0 < j \leq l$
- $\mathcal{M}(i,l'-x) = \mathcal{M}(0,l'-x) + \sum_i \delta(x_i,-),\ 0 < i \leq k$
- $\mathcal{M}(k'-x+a,l+a) = \mathcal{M}(k'-x+a-1,l+a-1) + \delta(k'-x+a,l+a) = \mathcal{M}(k'-x,l) + \sum_{b=1}^{a} \delta(k'-x+b,l+b),\ 0 < a \leq x$

- $\mathcal{M}(k+a, l'-x+a) = \mathcal{M}(k+a-1, l'-x+a-1)+\delta(k+a, l'-x+a) = \mathcal{M}(k'-x, l) + \sum_{b=1}^{a} \delta(k+b, l'-x+b),\ 0 < a \leq x$
- $\mathcal{M}(i, l+x) = \mathcal{M}(k', l+x) + \sum_i \delta(x_i, -),\ k' < i \leq m$
- $\mathcal{M}(k+x, j) = \mathcal{M}(k+x, l') + \sum_j \delta(-, y_j),\ l' < j \leq n$

The initialisation can be done in linear time according to the size of the sequences.

Let $\mathcal{P}'$ be the area limited by the polygon

$$((0,0), \mathcal{I}_1, \mathcal{I}_1', \mathcal{I}_2', (k', l'), \mathcal{I}_3', \mathcal{I}_4', \mathcal{I}_4).$$

Initially we can compute the cells of $\mathcal{M}$ that are covered by $\mathcal{P}'$ as we did before for the simple alignment. Due to the constraints we have to fulfill, we have to take a closer look on the values of the cells in $\mathcal{M}'$.

Let $\mathcal{M}'$ be represented by an additional matrix. We keep the coordinates of $\mathcal{M}$ to identify the cells of $\mathcal{M}'$. We are interested in the scores saved at the margins $\mathcal{M}'(i, j)$, where $i = k',\ j > l + x$ or $i > k + x,\ j = l'$, and the corresponding traceback paths. Let $\mathcal{T}$ be an adequate datastructure for storing all traceback paths, like a list of lists. Let P be a cell on the mentioned area on the margin (fig. 3.6). We will compute for each P a tracebackpath. Let d be the diagonal straight line passing P within $\mathcal{M}'$, let $|d|$ be the number of cells within $\mathcal{M}'$ passed by d. Let $g = |d| - x$ be the number of gaps allowed in an alignment aligning the elements corresponding to P in $\mathcal{S}_1$ and $\mathcal{S}_2$, resp. the maximal number non-diagonal steps in the traceback path in order to fulfill our constraint. $d - g$ and $d + g$ denote parallels of d in such a distance to d that $d - g,\ d + g$ can be reached from d in g horizontal or vertical steps. It holds that all possible traceback paths through P in $\mathcal{M}'$ lie in the area limited by the margin of $\mathcal{M}'$, $d-g$ and $d+g$ containing d. Let $\mathcal{A}$ denote that area.

The initial values of the considered area can be taken from $\mathcal{M}$ which values are already computed for the cells covered by $\mathcal{P}'$. All other cells of $\mathcal{M}'$ can be initialised by $-\infty$. The cells within $\mathcal{A}$ have to store g values, e.g in an array, since we need to know how many non-diagonal steps are contained in a traceback path passing the corresponding cells. The value stored at position 0 of one cell corresponds to a traceback path with 0 non-diagonal elements and the value stored at position h with h non-diagonal elements. Let $\mathcal{M}'_h(i, j)$ denote the hth value stored at position (i, j) in the matrix $\mathcal{M}'$.

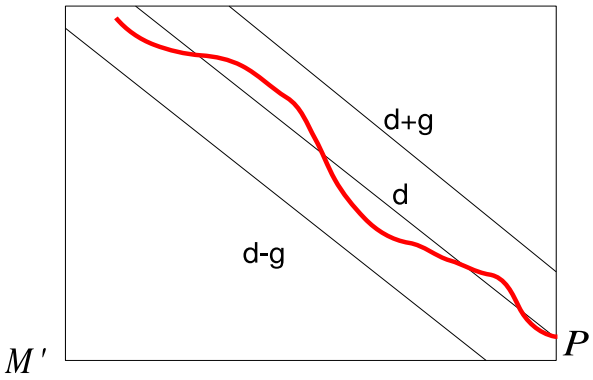

Figure 3.6: Closer look on $\mathcal{M}'$ for segment constraint
Traceback path is marked red.

The recursive formula (eq. 3.1) will alter as following:

$$\mathcal{M}'_h(i,j) = \max \begin{cases} \mathcal{M}'_h(i-1,j-1) + \delta(x_i, y_j), \text{ for } 0 \leq h \leq g \\ \mathcal{M}'_{h-1}(i-1,j) + \delta(x_i, -), \text{ for } 0 < h \leq g \\ \mathcal{M}'_{h-1}(i,j-1) + \delta(-, y_j), \text{ for } 0 < h \leq g \end{cases} \tag{3.3}$$

We begin our traceback path at the cell corresponding to P. When we reached the cell $\mathcal{M}_h(i,j)'$ the next cell on our path will be:

- $\mathcal{M}_0(i,j)'$ if $h = 0$.
- The cell of $\mathcal{M}'$ with the according h corresponding to the value which was used for the computation of $\mathcal{M}_h(i,j)'$.

The traceback path will be stored in $\mathcal{T}$.

For filling $\mathcal{M}$ at the area $\mathcal{P} \backslash \mathcal{P}'$ we use the already computed initial values and the maximal values stored in $\mathcal{M}'_h$ limiting $\mathcal{P} \backslash \mathcal{P}'$. Beside these initial values, we can continue using the standard algorithm (eq. 3.1).

The complete traceback path will start at $\mathcal{M}(m,n)$. It will be computed as used until it enters the cells corresponding to $\mathcal{M}'$ at the cell named P. The next part of the path will looked up in $\mathcal{P}$. When leaving $\mathcal{M}'$, we can compute the traceback path using the standard approach and the values stored in $\mathcal{M}$.

$\mathcal{M}$ requires $O(mn)$ storage and $\mathcal{M}'$ needs $O(|k||l|g)$. $\mathcal{T}$ stores $O(|k|+|l|)$ traceback paths of the length $O(|d|+g)$, i.e. $\mathcal{T}$ needs $O((|k|+|l|)(|d|+g))$

storage. For the worst case we will assume $g = |d| = O(|k|)^5$, without any restrictions, that leads to $O(mn + |k|^2|l|)$ total storage complexity. Since we have to consider at most three adjacent cells for filling our matrices and computing the traceback paths the time complexity is also $O(mn + |k|^2|l|)$.

[5] We could also assume $g = |d| = O(|l|)$.

Chapter 4

Cluster tree decomposition, elimination

4.1 A constraint model for sequence alignment

In contrast to chapter 2.1 we treat the gaps in an alignment separately and introduce a new scoring scheme.

We are interested in aligning two sequences $a = a_1, \ldots, a_n$ and $b = b_1 \ldots b_m$, both consist of letters of the alphabet Σ. According to [21] an alignment $\mathcal{A}$ of a and b can be defined as an ordered matching of positions in a and b, i.e. as a subset of $\{1, \ldots, n\} \times \{1, \ldots, m\}$, with the following constraints:

1. $i = i'$ if and only if $j = j'$ and
2. $i < i'$ implies $j < j'$

for all $(i, j), (i', j') \in \mathcal{A}$.
i and j are called matched by $\mathcal{A}$ if and only if $(i, j) \in \mathcal{A}$.

With the similarity function $\sigma : \{1, \ldots, n\} \times \{1, \ldots, m\} \Rightarrow \mathbb{R}$ and the gap costs γ one can define the score of an alignment $\mathcal{A}$:

Definition 4.1.1 *The scoring scheme for cluster tree decomposition*

$$score(\mathcal{A}) = (n + m - 2|\mathcal{A}|)\gamma + \sum_{(i,j)\in\mathcal{A}} \sigma(i, j)$$

The first part of this equation are the total gap costs, the second one represents the similarity of all matches in an alignment. By maximising score($\mathcal{A}$) we receive a measure for the similarity of the sequences a and b according to σ and γ. Definition 4.1.1 is equivalent to def. 2.1.2 considering

$$\delta = \begin{cases} \sigma & \text{for aligning two bases} \\ \gamma & \text{for a gap} \end{cases}$$

Definition 4.1.2 *Representation of an alignment as valuation.*
Here an alignment of a and b will be represented as a valuation of finite domain variables X_i, $1 \leq i \leq n$ with the domains $dom(X_i) = \{0, \ldots, m\}$. X_0 and X_{n+1}, being fixed values, are introduced for technical reasons: $X_0 = 0$ and $X_{n+1} = m + 1$. σ is extended by $\sigma(n+1, m+1) = 0$. Now an alignment can be uniquely represented by a valuation $(X_0 = 0, \ldots, X_{n+1} = X_n + 1)$ of the variables $X_0, \ldots, X_{n+1}$. Furthermore is:

1. *$x_i = j$ if $(i, j) \in \mathcal{A}$ and*
2. *$x_i = x_{i-1}$, for every i that is not matched in $\mathcal{A}$*

It holds, that i and j are matched if and only if $x_i = j$ and $x_i > x_{i-1}$.

Example 4.1.1 *The valuation $\mathbf{x} = (0, 1, 2, 5, 6, 6, 6, 7, 8)$ of $X_0, \ldots, X_8$ corresponds to the alignment $\{(1, 1), (2, 2), (3, 5), (4, 6), (7, 7)\}$ with, as already mentioned $X_0 = 0$ and $X_8 = 8$. The same alignment can also be represented by aligning the elements of pairs one upon the other. Gaps are represented by a bar aligned with an element of one sequence.*

$$\begin{array}{c} a_1\ a_2\ -\ -\ a_3\ a_4\ a_5\ a_6\ a_7 \\ b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ -\ -\ b_7 \end{array}$$

We have to take care of the following hard constraints on the the variables; it holds, that $X_{i-1} \leq X_i$ for $1 \leq i \leq n+1$. They are modeled by the functions:

$$leq_i : \mathrm{dom}(X_{i-1}) \times \mathrm{dom}(X_i) \to \{-\infty, 0\} \tag{4.1}$$

If these constraints are broken leq_i assigns $-\infty$ otherwise 0.

We can express the score by functions $f_i(X_{i-1}, X_i)$ for $1 \leq i \leq n+1$ that considers two adjacent variables in our variable valuation by the following scoring scheme.

$$f_i(j', j) = \begin{cases} \sigma(i, j) + (j - j' - 1)\gamma & \text{if } j' < j \\ \gamma & \text{otherwise} \end{cases} \tag{4.2}$$

Where $(j - j' - 1)\gamma$ in the first case counts the gaps in sequence a caused by the alignment and weights them by γ directly in front of a_i . The second case occurs if $j' = j$ i.e. there is a gap in sequence b at the position of a_j in alignment $\mathcal{A}$. Note that f_i correctly models the score of an alignment. The valuation $(X_0 = x_0, \ldots, X_{n+1} = x_{n+1})$ represents an alignment $\mathcal{A}$ of a and b if and only if $\sum_{1 \leq i \leq n+1} f_i(x_{i-1}, x_i) + leq_i(x_{i-1}, x_i)$ is not $-\infty$. In that case

$$\sum_{1 \leq i \leq n+1} f_i(x_{i-1}, x_i) = score(\mathcal{A}) \tag{4.3}$$

(see Def. 4.1.1) [21].

4.2 Definition of a cluster tree

Definition 4.2.1 *Cluster tree decomposition*
According to [16, 17, 21] a cluster tree decomposition (CTD) consists of a triple $< T, \chi, \psi >$, with $T = (V, E)$ with vertices V and edges $E = \{(v_i, v_j) | v_i, v_j \in V\}$. χ and ψ are labeling functions. Each vertex $v \in V$ can be classified to the two sets $\chi(v) \subseteq X$ and $\psi(v) \subseteq F$. X is the set of variables and F the set of functions of a reasoning problem. For a cluster tree the following conditions must be satisfied:

1. *For each function $f_i \in F$, there is exactly one vertex $v \in V$ such that $f_i \in \psi(v)$.*
2. *If $f_i \in \psi(v)$ then for every variable x of F it holds, that $x \in \chi(v)$*
3. *For each variable $x \in X$, the set of vertices labeled by x $\{v \in V | x \in \chi(v)\}$ induces a connected subtree.*
4. *$\forall i$ $Z_i \subseteq \chi(v)$ for some $v \in T$.*

According to [21] vertices are called cluster. Variables which are shared by two clusters that are connected by an edge, are called separator variables.

4.3 Cluster tree elimination

A reasoning problem represented by a CTD is solved by using the method of cluster tree elimination (CTE). First the functions of each cluster are combined to one function that marginalise them to the separator variables.

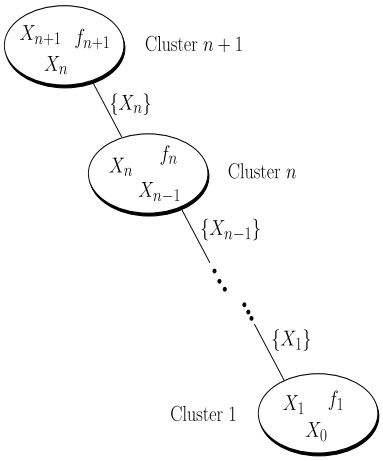

Figure 4.1: Cluster tree of a sequence alignment

Secondly the values of these functions are exchanged repeatedly by messages[1] between two clusters that share a common edge. Each message becomes a new message of the receiving cluster. According to the problem, the CTE maximises or minimises the functions. [21]

4.4 Using cluster tree decomposition and elimination for solving the alignment problem

For a simple sequence alignment the cluster tree degenerates to a linear list (Figure 4.1[2]). The cluster containing f_i, X_{i-1}, X_i and leq_i (equation 4.2 and 4.1) is called cluster i. The clusters i and $i-1$ share the separator variable X_{i-1}. The messages sent between cluster i and cluster $i+1$ are called g_i, where g_i are functions over the separator variables X_i.

Beginning with cluster 1 (the leave of our degenerated tree), the messages are sent upwards to the root. Cluster i can only send its message g_i to cluster $i+1$ when it received g_{i-1} since g_i depends on the results of g_{i-1}. The procedure finishes when cluster $n+1$ receives its message g_n.

The marginalisation of the functions in cluster $n+1$ to the empty set of variables is $\max_{1 \leq j \leq m}(g_n(j) + f_{n+1}(j, m+1))$. According to [21] it can be

[1] different values of one message caused by different valuations can be expressed in tabular form e.g. by using arrays

[2] taken from [21]

shown, that this marginalisation is the maximal alignment score due to the correctness of the method of cluster-tree decomposition and elimination

Here we show an alternative proof that $\max_{1\leq j\leq m}(g_n + f_{n+1}(j, m+1))$ is equal to the maximal alignment score of the two sequences $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_m)$. Let $\max score(\mathcal{A}_{i,j})$ denote the maximal alignment score of the sequences $(a_1, \ldots, a_i)$ and $(b_1, \ldots, b_j)$. We only consider legal alignments, i.e. $leq_i(j', j) = 0$ since illegal alignments would lead to a values equal to $-\infty$ and won't be the maximal value chosen by our maximisations. This proof uses induction.

- We suppose

$$\max_{1\leq j\leq m} \big(g_i(j) + f_{n+1}(j, m+1)\big) = \max score(\mathcal{A}_{n,m})$$

  holds for all $0 \leq i \leq n$.

- It holds

$$\begin{aligned}
&\max_{1\leq j\leq m} \big(g_1(j) + f_2(j, m+1)\big) \\
&\quad = \quad \max_{1\leq j\leq m} \left(\max_{0\leq j'\leq m} \big(g_0(j') + f_1(j', j)\big) + f_2(j, m+1) \right) \\
&\quad = \quad \max_{1\leq j\leq m} \left(\max_{0\leq j'<j} \big(f_i(j', j)\big) + f_{i+1}(j, m+1) \right) \\
&\quad \overset{\text{eq. 4.3}}{=} \quad \max score(\mathcal{A}_{1,m})
\end{aligned}$$

  since $X_{n+1} = m + 1$ in our valuation.

- $i - 1 \to i$

$$\begin{aligned}
&\max_{1\leq j\leq m} \big(g_i(j) + f_{i+1}(j, m+1)\big) \\
&\quad = \max_{1\leq j\leq m} \left(\max_{0\leq j'\leq m} \big(g_{i-1}(j') + f_i(j', j)\big) + f_{i+1}(j, m+1) \right) \\
&\quad = \max_{1\leq j\leq m} \big(\max score(\mathcal{A}_{i-1,m}) + f_{i+1}(j, m+1)\big) \\
&\quad = \max score(\mathcal{A}_{i,m})
\end{aligned}$$

  Note that $0 \leq j' \leq m$ can be replaced by $1 \leq j' < j$ due to our constraints and since $g_{i-1}(0) = 0$.

According to the CTE algorithm the message g_i, $0 \leq j \leq m$ is defined as:

$$g_i(j) = \max_{0 \leq j' \leq m} (g_{i-1}(j') + f_i(j', j) + leq_i(j', j)) \tag{4.4}$$

$g_i(j)$ is the maximal alignment score of the subsequences $a_1, \ldots, a_i$ and $b_1, \ldots, b_j$ of the two sequences a and b. The initial values $g_0(j)$ and $g_i(0)$ are set to 0. For computing g_i this approach takes $O(m^2)$ time and space. Since $O(n)$ messages are sent while traversing the tree this approach takes $O(nm^2)$.

In the following way Will, Busch and Backofen improved this complexity in their paper [21] to $O(nm)$, the same complexity as the dynamic programming approach:

By inserting the equation 4.2 for the f_i and using the definition 4.1 of leq_i we receive

$$g_i(j) = \max_{0 \leq j' \leq j} \left(g_{i-1}(j') + \begin{cases} \sigma(i,j) + (j - j' - 1)\gamma & \text{if } j' < j \\ \gamma & \text{otherwise} \end{cases} \right). \tag{4.5}$$

Since leq_i is mapped to $-\infty$ iff $j' > j$, we can adjust the index of max to $0 \leq j' \leq j$.

Now, we can resolve the case distinction of f_i and move the constant $\sigma(i,j)$ out of the maximisation.

$$g_i(j) = \max \begin{cases} \sigma(i,j) + \max_{0 \leq j' < j}(g_{i-1}(j') + (j - j' - 1)\gamma) \\ g_{i-1}(j) + \gamma \end{cases} \tag{4.6}$$

Now we only have to select the maximum of both equations.

The inner maximisation can be replaced by a helper function

$$g^m(j) = \max_{0 \leq j' < j} (g_{i-1}(j') + (j - j' - 1)\gamma), \tag{4.7}$$

which is the maximal alignment score of all subsequences $a_1, \ldots, a_{i-1}$ and $b_1, \ldots, b_{j'}$ with $0 \leq j' < j$ to sum with the gap costs in the subsequence of a. $g^m(j)$ can be defined recursively by

$$g^m(0) = -\infty, \; g^m(1) = g_{i-1}(0),$$

and for

$$j > 1 : \; g^m(j) = \max \begin{cases} g^m(j-1) + \gamma \\ g_{i-1}(j-1) \end{cases} .$$

This definition makes sense, since $g^m(0)$ is the beginning of the recursive function and the constraint $0 \leq j' < j = 0$ can't be fulfilled, furthermore

$$\begin{aligned} g^m(j-1) &= \max_{0 \leq j' < j-1} (g_{i-2}(j') + (j - j' - 2)\gamma) \\ &\Rightarrow \\ g^m(j) &= \max_{0 \leq j' < j} \begin{cases} (g_{i-2}(j') + (j - j' - 2)\gamma) + \gamma & \text{for } j' < j-1 \\ g_{i-1}(j-1) & \text{for } j' = j-1 \end{cases} \\ &= \max \begin{cases} g^m(j-1) + \gamma \\ g_{i-1}(j-1) \end{cases} \end{aligned}$$

therefore

$$g^m(1) = \max \begin{cases} g^m(0) + \gamma \\ g_{i-1}(0) \end{cases} = \max \begin{cases} -\infty \\ g_{i-1}(0) \end{cases} = g_{i-1}(0).$$

Finally the $g_i(j)$ can be evaluated with the following function:

$$g_i(j) = \max \begin{cases} \sigma(i,j) + g^m(j) \\ g_{i-1}(j) + \gamma \end{cases} \tag{4.8}$$

With its simple recursive definition of the helper function $g^m(j)$, $g_i(j)$ can be calculated in $O(m)$. Using this improved approach the algorithm for computation of the alignment score finishes in time $O(nm)$ like the approaches using dynamic programming.

4.5 Extended alignment problems

We regarded possibilities to extend alignments in dynamic programming to compute alignments with anchor, precedence or aligned segments constraints. This chapter focuses on the question if the approach of modeling sequence alignments with cluster trees, as described above, can be extended easily to further problems and how these extensions look like. The simple anchor and precedence constraints as well as alignment segments were discussed by Will, Busch and Backofen in their paper [21].

For example, if we want to force that position k in sequence a to be aligned to position l_1, or l_2, or $\ldots$, or l_i, in sequence b, where $i \leq m$ with m being the number of elements in b, it is sufficient to adjust the domains of our valuation (4.1, page 27):

$$X_k \in \{l_1, l_2, \ldots, l_i\} \text{ and } X_{k-1} < X_k$$

4.5.1 Anchor constraints

If we want to apply anchor constraints[3] to our model, it is sufficient to add the following constraints to our valuation:

$$X_{i-1} < j, X_{i+1} > j, \text{ and } X_i = j \vee X_i = X_{i-1} \tag{4.9}$$

In this case we want to align a_i in sequence a with b_j in sequence b. The first constraints imply that the elements left (resp. right) to a_i in sequence a have to be matched with elements left (resp. right) to b_j in sequence b. The second ones ensure, that a_i is matched with b_j or with a gap.

4.5.2 Precedence constraints

Similar to the anchor constraints problem we can reduce the precedence constraints[4] to an adjustment of the domains of our cluster tree decomposition.

The following constraints are adequate for adjusting the domains:

$$X_i < j \text{ (resp. } X_i > j) \tag{4.10}$$

This ensures that a_i in sequence a will be matched left (resp. right) to b_i in sequence b.

4.5.3 Aligned segments

For this problem it is not enough to alter the domains, we will need to introduce new variables and functions which have to be added to our cluster tree decomposition. We also have to adjust the functions g_i. A graphical representation is shown in Figure 4.2[5].

[3] as described in 2.2.1, page 12
[4] as described in 2.2.2, page 12
[5] taken from [21]

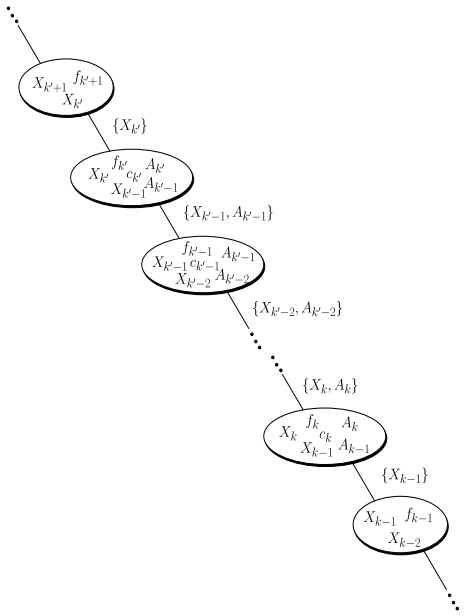

Figure 4.2: CTD of an alignment with segment constraints

For ensuring that at least $x\%$ of the positions in the segment $\{k, \ldots, k'\}$ in sequence a match with the positions $\{l, \ldots, l'\}$ in sequence b, we add the variables $A_{k-1}, \ldots, A_{k'}$. The additional functions $c_i(A_{i-1}, A_i, X_{i-1}, X_i)$ encode the hard constraint

$$A_i = A_{i-1} + \begin{cases} 1 & \text{if } X_{i-1} < X_i \text{ and } l \leq X_i \leq l' \\ 0 & \text{otherwise.} \end{cases} \tag{4.11}$$

A_{k-1} is fixed to 0. It holds, that A_i counts the number of proper matches in the segment $\{k, \ldots i\}$. Now we restrict the domains of A_i to $\{\max(0, \lceil \frac{x}{100}(k' - k + 1) \rceil - (k' - i)), \ldots, i - k + 1\}$ for expressing our constraint. $i - k + 1$ is the maximal number of matches at position i. The minimal number of this interval is the maximum of 0 and the minimal number of matches needed $(\frac{x}{100}(k' - k + 1))$ without the matches we can still achieve $(k' - i)$, if positive.

c_i, with $k \leq i \leq k'$, A_{i-1} and A_i will be added to cluster i in our cluster tree decomposition. We now have to adjust our equations g_i to our new constraints. For $g_i(X_i, A_i)$ we receive according to the CTE algorithm and the definition of A_i:

$$g_i(j, a) = \max_{0 \leq j' \leq j} \begin{cases} g_{i-1}(j', a-1) + f_i(j', j) & \text{if } j' < j \text{ and } l \leq j \leq l' \\ g_{i-1}(j', a) + f_i(j', j) & \text{otherwise} \end{cases} \tag{4.12}$$

Like before our functions leq_i disappear due to the index of our maximisation function. Using the definition of our scoring scheme 4.2 we receive

$$g_i(j,a) = \max \begin{cases} \sigma(i,j) + g^m(j,a) \\ \gamma + g_{i-1}(j) \end{cases} .$$

Where

$$g^m(j,a) = \max_{0 \le j' < j} ((j - j' - 1)\gamma + g_{i-1}(j', a'))$$

with

$$a' = \begin{cases} a & \text{if } l \le j \le l' \\ a - 1 & \text{otherwise} \end{cases}$$

Finally, as in the previous section, we define g^m recursively like in the previous section.

$$\begin{aligned} g^m(0,a) &= -\infty \\ g^m(1,a) &= g(1,a') \\ \text{and for } j > 1,\ g^m(j,a) &= \max \begin{cases} g^m(j-1,a'') + \gamma \\ g_{i-1}(j-1,a'') \end{cases} \end{aligned}$$

with $a'' = a$ if $l \le j \le j'$ and $a'' = a - 1$ otherwise. One can derive this equation equivalent to the derivation of 4.7. Again, this approach can be computed in $O(nm)$.

4.6 Sequence structure alignment

The sequence structure problem, as described in chapter 2.2.4, can be solved very elegant using cluster tree decomposition and elimination [21].

For given structures P_a and P_b, with P_a is pseudoknot-free (def. 2.2.2), we first we extend our alignment model by adding for each (i_l, i_r) functions $h_{i_l,i_r}(X_{i_l-1}, X_{i_l}, X_{i_r-1}, X_{i_r})$ that are defined as following:

$$h_{i_l,i_r}(j'_l, j_l, j'_r, j_r) = \begin{cases} \omega(i_l, i_r; j_l, j_r) & \text{if } j'_l < j_l, j'_r < j_r \text{ and } (j_l, j_r) \in P_b \\ 0 & \text{otherwise.} \end{cases}$$

In contrast to the known cluster trees our decomposition won't degenerate to a linear list, this tree will contain unary and binary nodes. An example cluster tree structure for an sequence structure alignment with two structural bindings $(k_l, k_r), (l_l, l_r) \in P_a$ is shown in figure 4.3[6].

[6] taken from [21]

4.6.1 Cluster-tree decomposition for a sequence structure alignment

A decomposition for a sequence a with m bases and a given structure P_a with $p \in \mathbb{N}$ base pairings will look as following. Let j_l, j_r, $1 \leq j \leq p$ denote the left and the right position of the j'th structural element in sequence a and the corresponding index of the variable valuation. It holds that $0 < j_l < j_r \leq m$. As before, an unary cluster containing X_i, X_{i-i}, f_i, leq_i will be called cluster i, $0 < i \leq m$, a binary cluster containing $X_{k_r}, X_{k_r-1}, f_{k_r}, leq_{k_r}, X_{k_l}, X_{k_l-1}, leq_{k_l}, h_{k_l k_r}$ will be called cluster k_r. Binary clusters will refer the right endpoints of a base binding in sequence a in our variable valuation, unary clusters all other valuations except X_0 and X_{n+1}. Therefore the indeces of all nodes, unary or binary, are unique and for every index $0 < i \leq m$ exists exactly one cluster.

- Cluster i is a unary vertex iff it holds for all j_r that $i \neq j_r,\ 1 \leq j \leq p$.
- Cluster i is a binary vertex iff there exists a j_r that $i = j_r,\ 1 \leq j \leq p$.
- Let c_{i_1} and c_{i_2} be two clusters of the cluster tree. Let c_{i_1} be part of the path from c_{i_2} to the root of the cluster tree. It holds: $i_1 > i_2$.
- The left subtree of a binary cluster j_r contains the clusters $j_l, \ldots, j_{r-1}$.
- The right subtree of the cluster j_r contains the clusters $1, \ldots, j_{l-1}$ iff j_r is not part of a left subtree of another binary cluster.
- Let c_{a_r} denote the nearest binary cluster above j_r on the path to the root of the cluster tree under the condition that j_r is part of the left subtree of c_{a_r}.
  - If j_r is part of the left subtree of c_{a_r}, it holds that the right subtree of the cluster j_r contains the clusters $c_{a_l}, \ldots, j_{l-1}$. c_{a_l} is a leaf since it refers to the element at the smallest position.
  - Let c_{min} denote the cluster with the minimal index in the right subtree of c_{a_r} and j_r be element of this subtree. The right subtree of j_r contains the clusters $c_{min}, \ldots, j_{l-1}$.

  If no binary cluster above j_r exists, the right subtree contains the clusters $1, \ldots, j_{l-1}$.
- Let c_a be a binary cluster above a cluster c. Let c be element of the left subtree of c_a on the path from the leaf c_l. It holds: $X_{c_l} \in c$

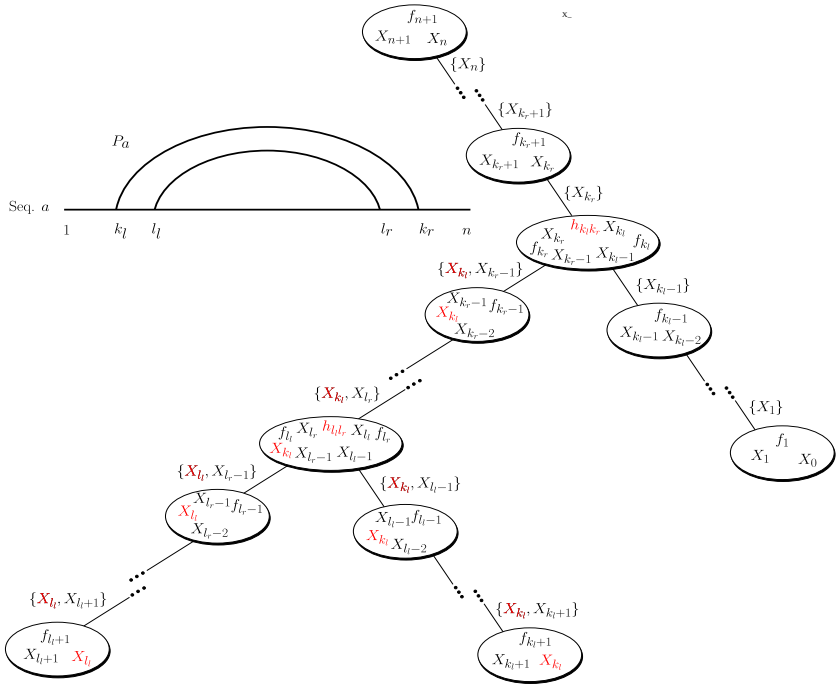

Figure 4.3: Sequence structure alignment CTD

- For an unary cluster c_u that is a leaf node it holds: $X_{u-1}, X_u, f_u, leq_u \in c_u$

- For a binary cluster c_{b_r} it holds:
  $h_{b_l b_r}, c_{b_r}, c_{b_r-1}, c_{b_l}, c_{b_l-1}, f_{b_r}, f_{b_l}, leq_{b_r}, leq_{b_l} \in c_{b_r}$

- A leaf node is either cluster 1 or a cluster referring to the 'left' base of a pairing in a since this position refers to the smallest index of the clusters of the subtrees, due to the construction of this tree.

As a small summary one could assume, that a binary cluster c_{i_r} stands for the right endpoint i_r of a binding in sequence a. Let T_s be a subtree representing the interval $T_{s_1}, \ldots, T_{s_2}$ where T_{s_1} is either 1 or the next smaller index than c_{i_l} in the structural binding[7] and T_{s_2} analogously the next bigger index of a binding or n. The right subtree of c_{i_r} contains the clusters $T_{s_1}, \ldots, c_{i_l-1}$, the left $c_{i_l}, \ldots, c_{i_r-1}$. Above c_{i_r} within T_s lie the clusters $c_{i_r}, \ldots, T_{s_2}$.

[7] if such an index exists

4.6.2 Attributes of the decomposition

The structure of the obtained cluster tree can be reduced to a binary tree by only considering the binary nodes, therefore a cluster tree for the sequence structure alignment with n binary clusters contains $n+1$ leaves. n leaves contain variables referring to the left endpoints of a binding, the last leaf is the cluster 0 due to the construction of our cluster tree. Each edge e contains only one or two separator variables since:

- If e is not part of the left subtree of a binary cluster and the only variable shared by the clusters x_i, x_{i+1} connected by e is x_i due to the construction of the cluster tree.

- If e is part of a left subtree of a binary cluster and c_{b_r} denotes the nearest binary cluster above e with e is part of the left subtree of c_{b_r}. Let c_1 and c_2 denote the clusters connected by e. c_1 refers to an element at position i_1 and c_2 at i_2, let $i_1 > i_2$. e lies either on the path from the leaf c_{b_l} to c_{b_r} or not.
  Let T_s denote the subtree below e with the root cluster c_2. Let T_s contain n binary clusters i.e. T_s contains $n+1$ leaves. Every binary cluster c_r in our subtree induces exactly one leaf cluster c_l in the same subtree.

  - If e isn't on the path, than it won't contain a separator variable X_{b_l} but it only contains X_{i_2}.
  - If e is part of the path, than n leaves of T_s belong to the different binary clusters in our subtree, the last leaf is c_{b_l}. All leaves except c_{b_l} belong to a binary cluster below e, therefore the only path between a leaf and a binary cluster, c_1 and c_2 can lie on, is the path between c_{b_l} and c_{b_r}, this means that the separator variables which c_1 and c_2 share are X_{i_2} and X_{b_l}.

4.6.3 Cluster-tree elimination for sequence structure alignments

Finally we have to define the functions g_{ssa} marginalising the functions of two connected clusters on the separator variables. Let $g_{i_{ssa}}$, $0 \leq i \leq m$ a function sent from cluster i to cluster $i+1$ over the separator variables. There are several different cases to be regarded, dependent on if edge $(i, i+1)$ contains one or two separator variables and if i is a unary or binary cluster. According to the CTE algorithm we receive:

- Edge $(i, i+1)$ contains one separator variable, $j \neq 0$, cluster i is no leaf:

$$g_{i_{ssa}}(j) = \begin{cases} \max_{0 \leq j' \leq m}(g_{i-1_{ssa}}(j') + f_i(j', j) + leq_i(j', j)) \\ \quad \text{if } i \text{ is an unary cluster} \\ \max_{0 \leq j'_l, j''_l, j'_r \leq m}(g_{i-1_{ssa}}(j'_l, j'_r) + f_i(j'_r, j) + leq_i(j'_r, j) + \\ g_{i_l-1}(j'_{l-1}) + f_{i_l}(j'_l, j''_l) + leq_{i_l}(j'_l, j''_l) + h_{i_l i}(j'_l, j''_l, j'_r, j)) \\ \quad \text{if } i \text{ is a binary cluster} \end{cases} \tag{4.13}$$

- Edge $(i, i+1)$ contains two separator variables, cluster i is no leaf:

$$g_{i_{ssa}}(j_l, j_r) = \begin{cases} \max_{j_l \leq j' \leq j_r}(g_{i-1_{ssa}}(j_l, j') + f_i(j', j) + leq_i(j', j)) \\ \quad \text{if } i \text{ is an unary cluster} \\ \max_{j_l \leq j'_l, j''_l, j'_r \leq j_r}(g_{i-1_{ssa}}(j'_l, j'_r) + f_i(j'_r, j_r) + leq_i(j'_r, j) + \\ g_{i_l-1}(j'_{l''}, j_l - 1') + f_{i_l}(j'_l, j''_l) + leq_{i_l}(j'_l, j''_l) + \\ h_{i_l i}(j'_l, j_l, j'_r, j)) \\ \quad \text{if } i \text{ is a binary cluster} \end{cases} \tag{4.14}$$

The initial values and the messages from leave nodes can be formulated as following:

- $g_{0_{ssa}} = 0$
- $g_{i_{ssa}}(0) = 0$
- Cluster i is a leaf
  Exploiting the fact that the initial values are 0 we can formulate $g_{i_{ssa}}$ over one separator variable as following:

$$g_{1_{ssa}}(j) = \max_{0 \leq j' \leq m} \big(f_1(j', j) + leq_1(j', j)\big)$$

  In the case $g_{i_{ssa}}$ is defined over two variables we omit the recursive call:

$$g_{i_{ssa}}(j_l, j_r) = \max_{j_l \leq j' \leq j_r} \big(f_i(j', j) + leq_i j', j\big)$$

It holds, that $\max_{1 \leq j \leq m} g_{n_{ssa}}(j) + f_{n+1}(j, m+1)$ is equal to the maximal alignment score, due to the correctness of the cluster tree elimination.

Supposing the size of the sequences is $O(n)$, we need $O(n^3)$ for computing one message $g_{i_{ssa}}(j_l, j_r)$. Since $O(n^2)$ messages are exchanged the total time complexity is in $O(n^5)$.

Note, if we use this method for aligning two sequences without any structural information, we receive the same algorithm as in chapter 4.4.

4.6.4 Construction of a cluster tree for sequence structure alignment

We can obtain a cluster tree for sequence structure alignments starting with the CTD of a standard alignment and changing its structure by adding the structural elements sequentially.

Let P_a denote the set of structural elements of sequence a, P_a is pseudoknot-free. Let T_i denote a cluster tree after adding the i'th structural element. Let $(l_{i+1}, r_{i+1}) \in P_a$ denote the $i+1$'th structural element. Note, since for any two structures of P_a it holds that the right and left endpoints are different (see 2.2.4), cluster r_{i+1} in T_i is unary. Let $T_{sub_i}(v)$ denote the subtree of an unary node with cluster $v-1$ as root after adding the i'th structural element. Let $mincluster(T)$ be the minimal index of a cluster of the cluster tree T. Let $T_{sub_{i_l}}(v), T_{sub_{i_r}}(v)$ denote the left and right subtree of a binary cluster v after adding the i'th structural element. Let $S_{T_i}(l, r)$ denote the subgraph of T containing the clusters $l, \ldots, r$ after adding the i'th structural element. Let $\chi_i(v)$ denote the set of variables of cluster v and $\psi_i(v)$ denote the set of functions of cluster v after adding the i'th structural element.

When adding the structural element $i+1$ we have to:

1. Alter the elements of the clusters
   - $\chi_{i+1}(r_{i+1}) = \chi_i(r_{i+1}) \cup \chi_i(l_{i+1})$
   - $\psi_{i+1}(r_{i+1}) = \psi_i(r_{i+1}) \cup \psi_i(l_{i+1}) \cup h_{l_{i+1}r_{i+1}}$
   - $\chi_{i+1}(v) = \chi_i(v) \cup X_{l_{i+1}}$, where $v \in \{l_{i+1}+1, \ldots, r_{i+1}-1\}$
2. Change the structure of the CTD
   Let $T_{sub_i}(r_{i+1})$ be abbreviated by T_{sub}
   - Cluster r_{i+1} becomes a binary cluster
   - $T_{sub_{l_{i+1}}}(r_{i+1}) = T_{sub} \backslash S_{T_{sub}}(mincluster(T_{sub}), l_{i+1}-1)$
   - $T_{sub_{r_{i+1}}}(r_{i+1}) = T_{sub} \backslash S_{T_{sub}}(l_{i+1}+1, r_{i+1}-1)$

Let $m = |P_a|$ and the size of sequence a be n. Constructing a CTD for a sequence structure alignment has the time complexity $O(mn)$ since we have to change for each structural element the set of variables for $O(n)$ clusters.

4.7 Combining several constraints

This chapter deals with the question what happens if several constraints of the same or of different type are combined. Since anchor and precedence constraints in CTD and CTE only affect the domains of our values X_i in the valuation and add some exra functions and doesn't affect the structure of the cluster tree or the messages g_i exchanged by the clusters, they can be simply added to the sequence structure alignment. Only the domains of the X_i have to be changed in the way mentioned in 4.5.1 and 4.5.2.

Let $dom(X_i)$ denote the domain of the variable X_i in our valuation. Let a set C of constraints be called consistent according to two sequences iff an alignment of these two sequences exists, that fulfills all constraints of C.

4.7.1 Multiple constraints of one type

In this section we will discuss what happens if we consider several constraints but all of the same type.

Anchor constraints

Of corse, not every combination of different anchor constraints makes sense.

Example 4.7.1 *An example for two anchor constraints on two sequences a and b that makes aligning a and b impossible:*

Let c_1 be a constraint that forces element a_1 of sequence a to align with element b_1 of b and c_2 a constraint that a_2 has to be aligned with b_2. It is obvious, that if $a_1 < a_2$ and $b_1 > b_2$ it is impossible to fulfill both constraints, c_1 and c_2, when aligning a and b.[8]

Let $\{c_1, \ldots, c_n\}$ be a set C of n given constraints, that force elements in sequence a to be aligned with especial elements in sequence b. Each $c_i, c_j \in C$ force element a_i to be aligned with b_i and a_j to be aligned with b_j[9].

[8]Note: In this case a_1 does not mean the first element in sequence a but the element of sequence a according to constraint c_1.

[9]again a_i, b_i and a_j, b_j refer to the elements in the sequence according to c_i, c_j.

C is consistent iff one of the following constraints is fulfilled for each pair of different constraints $c_i, c_j \in C$:

- $a_i = a_j$ and $b_i = b_j$
- $a_i < a_j$ and $b_i < b_j$
- $a_i > a_j$ and $b_i > b_j$

A consistency check can be done while adjusting the domains. We can add successively all constraints. Let $dom_k(X_i)$ denote $dom(X_i)$ after adding constraint c_k. At the beginning $dom_0(X_i)$, $X_i \in \{X_0, \ldots, X_{m+1}\}$ is $\{0, \ldots, m+1\}$. When adding a constraint c_k which forces a_i to align with b_j[10] the domains will change according to eq. (4.9), page 34, like followed:

- $dom_k(X_l) = dom_{k-1}(X_l) \backslash \{j, \ldots, m+1\},\ l < i$
- $dom_k(X_l) = dom_{k-1}(X_l) \backslash \{0, \ldots, j\},\ l > i$

We still have to add a function $\alpha_k(X_i, X_{i-1})$ to cluster i encoding the constraint, that $X_i = X_{i-1}$.

$$\alpha_k(X_i, X_{i-1}) = \begin{cases} 0 & \text{if } X_i = X_{i-1} \vee X_i = j \\ -\infty & \text{else} \end{cases}$$

$\alpha_k(X_i)$ will be accumulated to the message g_i or $g_{i_{ssa}}$.

An inconsistent set of constraints c would lead for at least one $X_i \in \{X_0, \ldots, X_{m+1}\}$ that $dom(X_i) = \emptyset$.

Precedence constraints

Let c_r resp. c_l denote a precedence constraint where element a_i of sequence a has to be aligned right resp. left to element b_j of sequence b[11]. Let C_r be a set of precedence constraints 'right' and C_l a set of precedence constraints 'left'. Let $C = C_r \cup C_r$. C is consistent iff for each pair of different pair of constraints $c_i, c_j \in C$ one of the following conditions is fulfilled:

- $c_i \in C_r \vee c_j \in C_r$
- $c_i \in C_l \vee c_j \in C_l$

[10] In this case a_i, b_j refer to the i'th and j'th element in the sequences a, b

[11] Here a_i, b_i and a_j, b_j refer to the elements in the sequence according to c_i, c_j.

- $(c_i \in C_r \vee c_j \in C_l) \vee (a_i > a_j \vee b_i > b_j)$
- $(c_i \in C_l \vee c_j \in C_r) \vee (a_i < j_j \vee b_i < b_j)$

Again we iteratively restrict the domains of our variables $X_i,\ i \in \{0, \ldots, m+1\}$ by adding one constraint after the other. $dom_0(X_i) = 0, \ldots, m+1$, after adding the k'th constraint c_k the domains will be restricted like followed:

- If $c_k \in C_r$, i.e. a_i has to be aligned right to b_j[12]
  - $dom_k(X_l) = dom_{k-1}(X_l) \backslash \{j+1, \ldots, m+1\},\ l \leq i$
  - $dom_k(X_l) = dom_{k-1}(X_l) \backslash \{0, \ldots, j\},\ l > i$
- If $c_k \in C_l$, i.e. a_i has to be aligned left to b_j
  - $dom_k(X_l) = dom_{k-1(X_l)} \backslash \{j+1, \ldots, m+1\},\ l < i$
  - $dom_k(X_l) = dom_{k-1(X_l)} \backslash \{0, \ldots, j\},\ l \geq i$

If during the computation of the domains for any X_i, c_k the case occurs that $dom_k(X_i) = \emptyset$ than we can conclude that C is inconsistent.

Aligned segments

Let C be a set of aligned segments constraints. Let A_{z_i}, $c_{z_i}(A_{z_{i-1}}, A_{z_i}, X_{i-1}, X_i)$ encode the counter of the matches in segments of sequences a, b according to $c_z \in C$ referring the segments $\{k_{c_z}, \ldots, k'_{c_z}\}$, $\{l_{c_z}, \ldots, l'_{c_z}\}$ (see 4.5.3 eq. 4.11). Messages g_i that are only defined on the separator variable X_i are equal to eq. 4.4. Messages defined on X_i and one A_{k_i} are equal to eq. 4.12.

Let g_i be defined on $(X_i, A_{r_i}, \ldots, A_{s_i})$. Let $A_1, \ldots, A_h$ denote the counters of all aligned segments constraints $c_1, \ldots, c_h$ on cluster j. With $\Delta_c(j', j)$, $c \in C$ defined as

$$\Delta_c(j', j) = \begin{cases} 1 & \text{if } j' < j \text{ and } l_c \leq j \leq l'_c \\ 0 & \text{else} \end{cases}$$

our message g_i now modifies as following:

$$g_i(j, a_1 \ldots, a_h) = \max_{0 \leq j' \leq j} g_{i-1}(j', a_1 - \Delta c_1(j', j), \ldots, a_h - \Delta_{c_h}(j', j)) + f_i(j', j)$$

Again the complexity can be improved equivalent to eq. 4.12 in 4.5.3.

[12] Now a_i, b_j refer to the i'th and j'th element of a, b

4.7.2 Different multiple constraints combined

Anchor and precedence constraints restrict the domain of the possible values of the elements X_i of our valuation and add new variables and functions. Aligned segment constraints change the messages g_i on the affected segments and introduce new variables and equations but they don't affect the domains of the variables X_i. The new variables of the different constraints don't affect each other. Even adding the counters of the aligned segments constraints to the messages of the sequence structure alignment only imply marginal changes that doesn't affect the structure of the tree or the equations of the messages but admitting the counters to recursive call of $g_{i_{ssa}}$.

Since we can simply add sequence structure information by changing the structure and the messages of a cluster tree with or without constraints and since we can add the constraints mentioned above to any cluster tree with or without sequence structure information or any other constraints, it is possible to add constraints and structures in any order and mix them.

Chapter 5

Multiple alignments using progressive alignment

Here, we try to enlarge the concept of cluster tree elimination and decomposition in order to align more then two sequences. We will use heuristic approaches to solve multiple alignments. For these approaches we will need to align sequences with alignments or even alignments with alignments. As an example we will first try to enlarge the concept of CTE and CTD discussed so far to align three sequences. Later progressive alignment will be used to align multiple sequences and combine them with constraints.

5.1 Aligning three sequences

In this part an alignment of three sequences will be solved using the known mechanism of cluster tree decomposition and elimination.

Definition 5.1.1 *Alignment of three sequences*
An alignment $\mathcal{A}_3$ *of three sequences* $a = a_1, \ldots, a_n$, $b = b_1, \ldots, b_m$ *and* $c = c_1, \ldots, c_o$ *is a subset of* $\{1, \ldots, n\} \cup \{-\} \times \{1, \ldots, m\} \cup \{-\} \times \{1, \ldots, o\} \cup \{-\}$ *with the following conditions:*

- $(-, -, -) \notin \mathcal{A}_3$
- $\forall (i, j, k) \in \mathcal{A}_3$*:* i, j *and* k *appear exactly once in* $\mathcal{A}_3$
- *Any pairs in* $\mathcal{A}_3$ *are non crossing, i.e.*
  - $\forall (i', j', k'), (i, j, k) \in \mathcal{A}_3 \cap \{1, \ldots, n\} \times \{1, \ldots, m\} \times \{1, \ldots, o\}$*:* $i' < i \Leftrightarrow j' < j \Leftrightarrow k' < k$

- $\forall (i',j',k'),(i,j,k) \in \mathcal{A}_3 \cap \{1,\ldots,n\} \cup \{-\} \times \{1,\ldots,m\} \times \{1,\ldots,o\}$: $j' < j \Leftrightarrow k' < k$, *with* $i' =' -'$ *or* $i =' -'$
- $\forall (i',j',k'),(i,j,k) \in \mathcal{A}_3 \cap \{1,\ldots,n\} \times \{1,\ldots,m\} \cup \{-\} \times \{1,\ldots,o\}$: $i' < i \Leftrightarrow k' < k$, *with* $j' =' -'$ *or* $j =' -'$
- $\forall (i',j',k'),(i,j,k) \in \mathcal{A}_3 \cap \{1,\ldots,n\} \times \{1,\ldots,m\} \times \{1,\ldots,o\} \cup \{-\}$: $i' < i \Leftrightarrow j' < j$, *with* $k' =' -'$ *or* $k =' -'$

With the similarity function $\sigma : \{1,\ldots,n\} \cup \{-\} \times \{1,\ldots,m\} \cup \{-\} \times \{1,\ldots,o\} \cup \{-\} \Rightarrow \mathbb{R}$ we can define the score of an alignment:

$$score(\mathcal{A}) = \sum_{(i,j,k)\in\mathcal{A}} \sigma(i,j,k)$$

Let $\mathcal{A}(s_1,s_2)$ denote the alignment of sequences s_1, s_2. We will now pairwise align all pairs $(s_1,s_2) \in \{a,b,c\}$, where $s_1 \neq s_2$. Afterwards, we will compute for each pair (s_1,s_2) the alignment $\mathcal{A}_3(s_1,s_2,s_3)$ by aligning $\mathcal{A}(s_1,s_2)$ with s_3, where $s_3 \in \{a,b,c\} \backslash \{s_1,s_2\}$. We will choose the best alignment according to:

$$\max_{\substack{s_1,s_2,s_3\in\{a,b,c\} \\ \text{where } s_1\neq s_2\neq s_3\neq s_1}} score(\mathcal{A}_3(s_1,s_2,s_3))$$

Note that the received alignment mustn't be the optimal alignment of the sequences $s_1, s2, s_3$.

5.1.1 Aligning an alignment of two sequences with a sequence

Let $\mathcal{A}_2(s_1,s_2)$ denote an alignment of the sequences s_1, s_2 given by a set of pairs $A = \{(s_{1_1}, s_{2_1}),\ldots,(s_{1_\omega}, s_{2_\omega})\}$ where maximal one element of each pair can be a gap. Let i denote the i'th pair of A. Let j denote the j'th element of sequence s_3. Note that $\sigma(i,j) = \sigma(s_{1_i}, s_{2_i}, s_{3_j})$.

Definition 5.1.2 *Valuation*
An alignment of (s_1,s_2) *and* s_3 *will be represented as a valuation of finite domain variables* X_i, $1 \leq i \leq \omega$ *with the domains* $dom(X_i) = \{0,\ldots,o\}$. *Additionally* $X_0 = 0$, $X_{\omega+1} = o+1$. *Furthermore is:*

1. $x_i = j$ *if* $(i,j) \in \mathcal{A}_3(s_1,s_2,s_3)$ *and*
2. $x_i = x_{i-1}$ *for every* i *that is not matched in* $\mathcal{A}_3(s_1,s_2,s_3)$

Cluster tree decomposition

With the new definition of the valuation, we can maintain the functions $leq_i, f_i(j, j')$ of 4.1 with the marginal difference, that $\sigma(i, j)$ in f_i is defined on a triple where one element of the pair i can be a gap. We can obtain the decomposition on the very same way than for an alignment of two sequences.

Cluster tree elimination

Again, we can use the very same messages g_i as in 4.4 for computing the alignment by adjusting σ.

It holds that $\mathcal{A}_2$ has maximal $2n$ pairs, since it can contain at most n gaps, therefore this approach is in $O(n^2)$ since we compute a constant number of alignments.

5.2 Progressive alignment

There exist several algorithms for computing an optimal alignment between two sequences. But finding an optimal alignment for multiple sequences is NP-hard [20]. Progressive alignment is an heuristic approach for finding a good multiple alignment. This will only be a rough description of progressive alignment according to [6]. It consists of three phases :

1. a pairwise alignment of all sequences
2. production of a guide tree using the alignment scores of step one
3. a progressive alignment of the sequences, guided by the tree

In phase one we align all sequences and remember their scores. We can use any preferred technique for aligning the sequences especially the one using cluster tree decomposition and elimination. According to the score, we construct the guide tree, hoping that close related sequences will be aligned at the beginning of the algorithm and that this leads to a nearly optimal solution. An example for an alignment of six sequences is shown in fig. 5.1. $s_1, \ldots, s_6$ represent the sequences, $a_1, \ldots, a_4$ intermediate result alignments and MA is the final multiple alignment. Phase three computes the progressive alignment starting with the sequences at the bottom of the guide tree, ending at the root by obtaining the multiple alignment.

The quality of the result of a progressive alignment depends heavily on the starting order, which is influenced by the scores of the pairwise alignments.

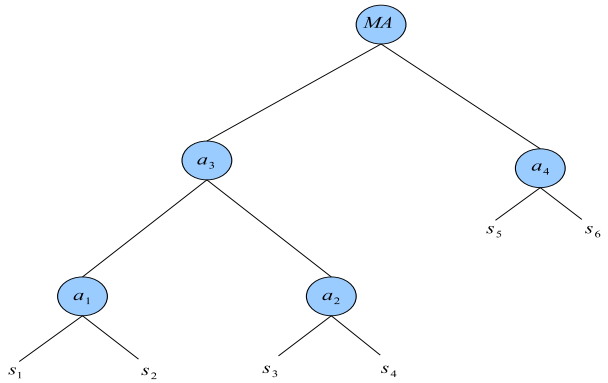

Figure 5.1: Guide tree of a multiple alignment

5.2.1 How to align alignments using CTD and CTE

Let two Alignments $\mathcal{A}_1, \mathcal{A}_2$ over n, m sequences be given. Let $\mathcal{A}_1$ be represented in the following way:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1x} \\ a_{21} & a_{22} & \cdots & a_{2x} \\ \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nx} \end{pmatrix}$$

Equivalently we will represent $\mathcal{A}_2$ as

$$\begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1y} \\ b_{21} & b_{22} & \cdots & b_{2x} \\ \vdots & \vdots & \vdots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mx} \end{pmatrix}$$

Where a_{ij}, b_{kl} are either elements of the sequences a_i, b_k or gaps. When $\mathcal{A}_1$ or $\mathcal{A}_2$ is only defined over one sequence, it will represent the sequence without any gaps.

Let $\mathcal{A}_{1i}, \mathcal{A}_{2j}$ denote the i, j'th column of $\mathcal{A}_1, \mathcal{A}_2$. With $\sigma(i, j)$ denoting the score for aligning $\mathcal{A}_{1i}$ and $\mathcal{A}_{2j}$ and an adequate γ, representing a row only filled with gaps, we can use the very same algorithm for cluster tree decomposition/elimination as used.

5.3 Combining multiple alignments with constraints

In this section we consider what happens if alignments with constraints are aligned with other alignments. Constraints between sequences can induce new constraints between other sequences in multiple alignments which we have to consider when using e.g. progressive alignment. We will only consider anchor and precedence constraints.

Example 5.3.1 *Constraints in multiple alignments*
Given three sequences A, B, C with an anchor constraint between a_i, b_j and a precedence constraint between B and C that force c_k to be aligned with a position bigger than b_l. If $b_j < b_l$, a precedence constraint between A and C is induced, namely c_k has to be aligned to a position bigger than a_i.

Let $\mathcal{S} = \{s_1, \ldots, s_g\}$ be a set of sequences and $\mathcal{C} = \{c_1, \ldots, c_h\}$ a set of constraints. Let $\sim_1$ be a relation symbol such that $s_i \sim_1 s_j \Leftrightarrow \exists c_k \in \mathcal{C} : c_k$ is a constraint on the sequences s_i, s_j, where $s_i, s_j \in \mathcal{S}$. Let $\sim$ be a transitive relation symbol with $s_i \sim s_j,\ s_i, s_j \in \mathcal{S}$ iff:

- $s_i \sim_1 s_j$ or
- $\exists s_{l_1}, \ldots, s_{l_e} \in \mathcal{S} : s_i \sim_1 s_{l_1} \sim_1 \cdots \sim_1 s_{l_e} \sim_1 s_j$

We will restrict the set of constraints $\mathcal{C}$ so far, that only constraints $c_j \in \mathcal{C}$ are allowed for all $s_i, s_k, s_l \in \mathcal{S}$ such that it is not possible to form a chain of relations $\sim_1$ as following: $s_i \sim_1 s_k \sim_1 \cdots \sim_1 s_l \sim_1 s_i$ with $s_k \neq s_l$[1]

Now we have to compute a transitive closure of the constraints on the sequences and consider what happens with the constraints on other levels of the guide tree during progressive alignment when aligning alignments with sequences or other alignments.

Let $\mathcal{G}$ be a graph with vertices $V = \mathcal{S}$ and undirected edges $E = \{(s_i, s_j) \in \mathcal{S} \times \mathcal{S} : s_i \sim_1 s_j\}$. For each $s_i \in \mathcal{S}$ the transitive closure of relation $\sim$ is given by the connectivity component of s_i. Due to the restriction, $\mathcal{G}$ is cycle free and a connectivity component is given by a spanning tree. We can now compute the transitive closure of each $s_i \in \mathcal{S}$ by using the spanning tree T_{s_i} with the root s_i.

[1] i.e. it is impossible for a constraint c on sequence s to induce further constraints on s

Let $\mathcal{C}_{ij}$ be the set of all constraints on s_i and s_j, with $s_i \sim_1 s_j$. Let $\mathcal{C}_h$ be the set of all constraints on s_h induced by any other sequence s_g below node s_i in the tree T_{s_i} , where $s_h \in T_{s_i}$. That means $s_h \sim s_g$.

Let $depth(T_{s_i})$ denote the depth of the tree T_{s_i} and $depth(T_{s_i}, n)$ the depth of node n in the tree T_{s_i}. With $nodes(T_{s_i}, j)$, the set of all nodes of the tree T_{s_i} at the depth j we can compute the set $\mathcal{C}_i$, i.e. all constraints on s_i. We begin computing $\mathcal{C}_j$ for all nodes $n_j \in nodes(T_{s_i}, depth(T_{s_i}) - 1)$. Next we will compute the set of all constraints of the nodes one level above n_j, successively until we reach the root s_i as below:

$$\mathcal{C}_n = \bigcup_{n' \in V} \mathcal{C}_{nn'} \bigcup_{\substack{n'' \in V, \text{ where} \\ n'' \text{ is a child of } n}} inducedconsts(n, n'')$$

With $inducedconsts(n, n'')$ gives the set of constraints that are induced by constraints $c_{n'',n'''}$ on the sequences $s_n, s_{n'''}$.

This gives us the set of all constraints on the sequence s_i. Let c_j be a constraint on the sequence s_i. Let s_i be already be aligned by the alignment $\mathcal{A}_k$. Let p be the set of positions that are affected by c_j. The set of rows of $\mathcal{A}_k$ that are affected by c_j are all rows, that contain an element of p. The case, that an alignment with constraints is aligned is equivalent.

5.3.1 Induced constraints

Here we discuss the function $inducedconsts(i, j)$ over $i, j \in V : \exists c_{i,j} \in \mathcal{C}$, where j is a child of i. Let $K = \{c_{j,k} \in \mathcal{C} | i \neq j \neq k\}$. Let $Result = \emptyset$. Let $\alpha, \beta, \gamma, \delta, \omega$ be positions in a sequence. For each pair $c_{ij} \in \mathcal{C}_{ij}, c_{jk} \in K$ we will do the following:

- If c_{ij} is an anchor constraint:
  Let c_{ij} force $s_{i\alpha}$ of sequence s_i to match with $s_{j\beta}$ in sequence s_j or with an element at a smaller position than the one of $s_{j\beta}$.
  - If c_{jk} is an anchor constraint:
    Let c_{jk} force $s_{j\gamma}$ of sequence s_j to match with $s_{k\delta}$ in sequence s_k or with an element at a smaller position then the one of $s_{k\delta}$.
    * If $\beta = \gamma$:
      $Result = Result \cup c_{ik}$, where c_{ik} is an anchor constraint between $s_{i\alpha}$ and $s_{k\delta}$.

* If $\beta > \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega > \delta$
* If $\beta < \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta$.

– If c_{jk} is a precedence constraint:
Without loss of generality let c_{jk} force $s_{j\gamma}$ with positions less than $s_{k\delta}$.

* If $\beta < \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta - 1$.
* If $\beta = \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta$.
* If $\beta > \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega > \delta - 1$.

- If c_{ij} is a precedence constraint:
Without loss of generality let c_{ij} force $s_{i\alpha}$ with positions less than $s_{j\beta}$.

– If c_{jk} is an anchor constraint:
Let c_{jk} force $s_{j\gamma}$ of sequence s_j to match with $s_{k\delta}$ in sequence s_k or with an element at a position smaller than the one of s_{k_δ}.

* If $\beta > \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega > \delta - 1$.
* If $\beta = \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta$.
* If $\beta < \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta - 1$.

– If c_{jk} is a precedence constraint:
Let c_{jk} be a precedence constraint, that forces $s_{j\gamma}$ to match with positions less $s_{k\delta}$.

* If $\beta > \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega > \delta - 1$.

* If $\beta = \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta$.
* If $\beta < \gamma$:
  $Result = Result \cup c_{ik}$, where c_{ik} is a precedence constraint, that forces $s_{i\alpha}$ to align with $s_{k\omega}$, where $\omega < \delta - 1$.

Note a precedence constraint that force $s_{i\alpha}$ to match with positions less that $s_{j\beta}$ is equal to a precedence constraint that force $s_{j\beta}$ to match with positions greater than $s_{i\alpha}$.

When computed the set $Result$ for all pairs $c_{ij} \in \mathcal{C}_{ij}, c_{jk} \in K$ mentioned above it holds: $inducedconsts(i, j) = Result$.

5.3.2 Following the guide tree

When we computed all constraints on all sequences $s_i \in \mathcal{S}$ we can start computing the multiple alignment aided by the guide tree.

Let b_i be either a sequence s_i or an alignment a_i. Let Γ_i denote the set of all constraints of b_i. Let

$$\Sigma_i := \begin{cases} \text{the set of all sequences that form the alignment } a_i \\ \{s_i\} \text{ for one sequence } s_i \end{cases}$$

After computing the alignment a_i of b_{i-1} and b_{i-2} we can compute Γ_i:

$$\Gamma_i = (\Gamma_{i-1} \cup \Gamma_{i-2}) \backslash \{c_{xy} \in \mathcal{C} | (s_x \in \Sigma_{i-1} \wedge s_y \in \Sigma_{i-2}) \vee (s_x \in \Sigma_{i-2} \wedge s_y \in \Sigma_{i-1})\}$$

Now we can add the constraints of Γ_i to a_i.

Chapter 6

Multiple alignments using the method of cluster tree decomposisiton and elimination

Finding an optimal multiple alignment is NP-hard [20]. This chapter gives an approach for computing alignments using the method of cluster tree decomposition and elimination. This approach is exponential in the number of sequences.

We will assume constant gap costs γ and discuss an alignment over three sequences first. The obtained results will be extended in order to solve a multiple alignment.

6.1 Aligning three sequences

Let three sequences $a = a_1a_2 \dots a_n, b = b_1b_2 \dots b_m$ and $c = c_1c_2 \dots c_o$ be given. Let $\mathcal{A}_3$ be an alignment on a, b, c according to definition 5.1.1. Let $\sigma(d)$ be a cost function, where $d \in \mathcal{A}_3$. Let $\sigma(-, x, y) = \sigma(x, y) + \gamma$. Let $\mathcal{A}_2(i, j, k, l)$ denote an alignment and $\mathcal{A}_2^*(i, j, k, l)$ denote the optimal alignment of the subsequences $b_i \dots b_j$ and $c_k \dots c_l$ with $1 \leq i < j \leq m$ and $1 \leq k < l \leq o$.

6.1.1 A constraint model for three sequences alignment

Definition 6.1.1 *Valuation for three sequences alignment*
The valuation of an alignment $\mathcal{A}_3$ of the sequences $a = a_1a_2 \dots a_n$, $b =$

$b_1 b_2 \dots b_m$ and $c = c_1 c_2 \dots c_o$ is represented by the vectors

$$X_i = \begin{pmatrix} X_{1_i} \\ X_{2_i} \end{pmatrix},\ 1 \leq i \leq n.$$

X_{1_i} and X_{2_i} are finite domain variables with $dom(X_{1_i}) = \{0, \dots, m\}$ and $dom(X_{2_i}) = \{0, \dots, o\}$. Again X_0, X_{n+1} are additional fixed values, introduced for technical reasons.

$$X_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix},\ X_{n+1} = \begin{pmatrix} m+1 \\ o+1 \end{pmatrix}$$

For the variables $X_{1_i}, X_{2_i}, 1 \leq i \leq n$ it holds:

- *1. $X_{1_i} = j$, if $(i, j, \alpha) \in \mathcal{A}$, with $\alpha \in \{c_1, \dots, c_o\} \cup \{-\}$*
  2. $X_{1_i} - 1$ else
- *1. $X_{2_i} = k$. if $(i, \beta, k) \in \mathcal{A}$, with $\beta \in \{b_1, \dots, b_m\} \cup \{-\}$*
  2. $X_{2_i} - 1$ else

Unfortunately, a valuation according to definition 6.1.1 is ambiguous as we can see in example 6.1.1. For finding an optimal alignment we can trace back these ambiguities to an alignment of two sequences.

Example 6.1.1 *Ambiguities in the valuation*
Let an alignment $\mathcal{A}_3$ over the sequences $a = a_1 \dots a_4$, $b = b_1 \dots b_6$ and $a = c_1 \dots c_7$ be given as following:

$$\begin{array}{c} - - a_1\ a_2\ a_3 - - a_4 - - \\ b_1\ b_2\ b_3\ b_4 - b_5 - - - b_6 \\ c_1 - - c_2\ c_3\ c_4\ c_5 - c_6\ c_7 \end{array}$$

The valuation for $\mathcal{A}$ is given by $\mathbf{X} = (X_0, \dots, X_5)$:

$$\mathbf{X} = \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 3 \\ 0 \end{pmatrix}, \begin{pmatrix} 4 \\ 2 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \end{pmatrix}, \begin{pmatrix} 4 \\ 3 \end{pmatrix}, \begin{pmatrix} 7 \\ 8 \end{pmatrix} \right)$$

When we keep the matchings of the elements of a, every alignment of the subsequences $b_1 b_2$ with c_1 would lead to the same valuation. Even every other possible alignment $\mathcal{A}_2(5, 6, 4, 7)$ extended by $(a_4, -, -)$ at any position doesn't change our valuation.[1]

[1] Each pair of these alignments over two sequences will be aligned with $'-'$ to fit to the alignment of three sequences.

We still have to encode the hard constraints $X_{1_{i-1}} \leq X_{1_i}$ and $X_{2_{i-1}} \leq X_{2_i}$, for $1 \leq i \leq n+1$. Analogous to 4.1 this is done by the following function:

$$leq_i : dom(X_{1_{i-1}}) \times dom(X_{1_i}) \times dom(X_{2_{i-1}}) \times dom(X_{2_i}) \to \{-\infty, 0\} \quad (6.1)$$

Again $-\infty$ will be assigned if these constraints are broken, 0 else.

Given the following part of an alignment $\mathcal{A}_\ni$

$$\begin{array}{ccccccc} & a_i & & a_x & & a_j & \\ \cdots & b_k & \cdots & - & \cdots & b_l & \cdots \\ & c_m & & - & & c_n & \end{array}$$

with

- $a_i, a_x, a_j \in \{a_1, \ldots, a_n\},\ i < x < j$
- $b_k, b_l \in \{b_1, \ldots, b_m\} \cup \{-\},\ k < l$ if $b_k \neq - \wedge b_l \neq -$
- $c_m, c_n \in \{c_1, \ldots, c_o\} \cup \{-\},\ m < n$ if $c_m \neq - \wedge c_n \neq -$
- Only one element of each pair $(b_k, c_m), (b_l, c_n)$ can represent a gap.
- For each (a_r, b_s, c_t) between (a_i, b_k, c_m) and (a_j, b_l, c_n) it holds that (a_r, b_s, c_t) is of the following form
  - $(-, b_s, c_t)$ or
  - $(a_i, -, -)$.

Assuming constant gap costs γ we can place $(a_x, -, -)$ anywhere between (a_i, b_k, c_m) and (a_j, b_l, c_n), especially direct after (a_i, b_k, c_m), without changing the score of $\mathcal{A}_3$ and the valuation.

Since we can move elements $(a_x, -, -)$ of an alignment $\mathcal{A}_3$ up to a certain extend, we can define the scoring scheme of $\mathcal{A}_3$ as $f_i(X_{1_{i-1}}, X_{1_i}, X_{2_{i-1}}, X_{2_i})$, $1 \leq i \leq n+1$ as following:

$$f_i(j', j, k', k) = \begin{cases} \sigma(i,j,k) + \\ \ score(\mathcal{A}_2^*(j', j, k', k)) + & \text{if } j' < j \wedge k' < k \\ \ |\mathcal{A}_2^*(j', j, k', k)|\gamma & \\ 2\gamma & \text{else,} \end{cases} \quad (6.2)$$

with $|\mathcal{A}_2^*(j', j, k', k)|$ denoting the number of elements in the alignment of the subsequences $b'_j \ldots b_j, c'_k \ldots c_k$ given by $\mathcal{A}_2^*(j', j, k', k)$.

It holds that

$$\sum_{1\leq i\leq n+1} f_i(X_{1_{i-1}}, X_{1_i}, X_{2_{i-1}}, X_{2_i}) = score(\mathcal{A}). \tag{6.3}$$

6.1.2 Cluster tree decomposition

The cluster tree of an alignment of three sequences forms a list and consists of $n+1$ clusters. Cluster i is connected with cluster $i-1$ and cluster $i+1$, $1 < i < n+1$. Cluster 1 is only connected with cluster 2 and cluster $n+1$ with cluster n. The elements contained by cluster i are the functions $f_i(X_{1_{i-1}}, X_{1_i}, X_{2_{i-1}}, X_{2_i})$, $leq_i(X_{i-1}, X_i)$ and the vectors X_{i-1}, X_i. This forms the very same structure as visualised in figure 4.1, page 30[2].

6.1.3 Cluster tree elimination

According to the CTE algorithm, we will define the messages $g_i(j,k)$ exchanged by the clusters $i, i+1$ as following

$$g_i(j,k) = \max_{\substack{0\leq j'\leq m\\ 0\leq k'\leq o}} (g_{i-1}(j',k') + f_i(j',j,k',k) + leq_i(j',j,k',k)). \tag{6.4}$$

The initial values are $g_0(j,k) = 0$ and $g_i(0,0) = 0$. For computing an optimal alignment $\mathcal{A}_3^*$ we need $O(nm^3o^3)$ space and time. One factor mo is caused by the computation of the alignments of the subsequences of the sequences b and c.

Equation 6.4 can be improved by inserting equation 6.2 and adjusting the maximisation using equation 6.1.

$$\begin{aligned} g_i(j,k) &= \max_{\substack{0\leq j'\leq j\\ 0\leq k\leq k}} g_{i-1}(j',k') + \\ &\quad \begin{cases} \sigma(i,j,k) + score(\mathcal{A}_2^*(j',j,k',k)) + |\mathcal{A}_2^*(j',j,k',k)|\gamma \\ \text{ if } j' < j \wedge k' < k \\ 2\gamma \\ \text{ else} \end{cases} \\ &= \max \begin{cases} \sigma(i,j,k) + \max_{\substack{0\leq j'<j\\ 0\leq k'<k}} (g_{i-1}(j',k') + \\ score(\mathcal{A}_2^*(j',j,k',k)) + |\mathcal{A}_2^*(j',j,k',k)|\gamma) \\ g_{i-1}(j,k) + 2\gamma \end{cases} \end{aligned} \tag{6.5}$$

[2]Note that the functions leq_i are omitted in that representation

These slight improvements don't change the order of the time and space complexity. Unfortunately, introducing functions $g^m(j,k)$ in a similar way than we did in chapter 4.4 doesn't induce any further improvements . This is due to the fact, that we can't eliminate the variables j', k'. These variables are needed if we try to reduce $g^m(j,k)$ to its recursive ancestors, since we have to compute the alignments $\mathcal{A}_2^*$ of the subsequences in the sequences b, c. These alignments need j', k' as boundary for the subsequences.

6.2 Aligning m sequences

Given m sequences $s_1^1 s_2^1 \dots s_{n_1}^1,\ s_1^2 \dots s_{n_2}^2,\ \dots,\ s_1^m \dots s_{n_m}^m$. The valuation according to an alignment $\mathcal{A}_m$ of these sequences will be defined equivalently to definition 6.1.1, with some slight differences:

$$X_i = \begin{pmatrix} X_{2_i} \\ X_{3_i} \\ \vdots \\ X_{m_i} \end{pmatrix},\ 1 \le i \le n_1,\ 1 \le X_{2_i} \le n_2,\ \dots,\ 1 \le X_{m_i} \le n_m$$

$$X_0 = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ o \end{pmatrix},\ X_{n_1+1} = \begin{pmatrix} n_2+1 \\ n_3+1 \\ \vdots \\ n_m+1 \end{pmatrix}$$

The value assigned to the variables X_{y_i}, $2 \le y \le m$ is equivalent to the values in definition 6.1.1.

The hard constraints $X_{y_{i-1}} \le X_{y_i}$, $2 \le y \le m$ are encoded by the functions:

$$leq_i : dom(X_{2_{i-1}}) \times dom(X_{2_i}) \times \dots \times dom(X_{m_{i-1}}) \times dom(X_{m_i}) \to (-\infty, 0)$$

Let $\mathcal{A}^*_{m-1}(j'_2, j_2, \dots, j'_m, j_m)$ denote an optimal alignment of the subsequences $s^2_{j'_2}, s^2_{j_2},\ \dots,\ s^m_{j'_m}, s^m_{j_m}$. Let σ be a cost function on an alignment over m sequences.

The very same considerations as in chapter 6.1 lead to the scoring scheme:

$$f_i(j'_2, j_2, \dots, j'_m, j_m) = \begin{cases} \sigma(j_1, j_2 \dots, j_m) + \\ \ score(\mathcal{A}^*_{m-1}(j'_2, j_2, \dots, j'_m, j_m)) + \\ \ |\mathcal{A}^*_{m-1}(j'_2, j_2, \dots, j'_m, j_m)|\gamma \\ \ \text{if } j'_2 < j_2 \wedge \dots \wedge j'_m < j_m \\ (m-1)\gamma \\ \ \text{else} \end{cases} \tag{6.6}$$

The cluster tree decomposition is equivalent to chapter 6.1.2.

Finally we obtain the following recursive function for the messages exchanged by the clusters

$$\begin{aligned} g_i(j_2,\ldots,j_m) \quad = \quad & \max_{0\le j'_2\le n_2} (g_{i-1}(j'_2,\ldots,j'_m) \\ & \vdots \\ & {\scriptstyle 0\le j'_m\le n_m} \\ & +f_i(j'_2,j_2,\ldots,j'_m,j_m) + leq_i(j'_2,j_2,j'_m,j_m)). \end{aligned} \tag{6.7}$$

Using this approach for computing an optimal alignment of m sequences we achieve a time and space complexity of $O(n_1 n_2^3 n_3^5 \ldots n_{m-1}^{2m-3} m_m^{2m-3})$.

6.3 Adding constraints

In this chapter, we will add constraints on an alignment over m sequences $s^1,\ldots,s^m$. We will focus on anchor and precedence constraints. We will only allow constraints between s^1 and s^x, $2 \le x \le m$. This is very similar to chapter 4.5.

6.3.1 Anchor constraints

When we add an anchor constraint between the elements s_i^1 and s_j^x we have to add the following constraints to our valuation:

$$X_{1_i-1} < j, X_{1_{i+1}} > j, \text{ and } X_{1_i} = j \vee X_{1_i} = X_{1_{i-1}}$$

6.3.2 Precedence constraints

When we want to force s_i^1 to align left (resp. right) to s_j^x we have to adjust the domains of the valuation as following:

$$X_{1_i} < j, \text{ (resp. } X_{1_i} > j)$$

Chapter 7

Conclusion

After a short intoduction on computing alignments using the method of dynamic programming, we discussed how to solve anchor, precedence and aligned segments constraints. Even though it is possible to compute alignments under anchor, precedence and aligned segments constraints using dynamic programming, it is difficult to adapt them to the standard approach especially the anchor constraints. Therefore it is hardly possible to combine them when computing the alignment with this approach.

In contrast to the approach mentioned above we regarded the approach using cluster tree decomposition and elimination. After a review of the article of Will, Busch and Backofen [21] about computing alignments using CTD, CTE and adding constraints, we formalised the mechanism of adding constraints. We also discussed the sequence structure constraints in detail and developed a mechanism that transforms a CTD without sequence structure constraints into a CTD that encodes these constraints. Sequence structure constraints were examined rather briefly in [21]. Later we have shown how to combine several constraints in an alignment. Though the theory behind cluster trees is a slightly more complicated, it is to a great extend easier to add constraints and mix them than with the approach of dynamic programming. Sequence structure alignments can be computed and combined with different constraints in a very elegant way - simply by changing tree structures, changing the messages exchanged by the clusters and restricting the domains of the variables in the clusters.

Furthermore we solved multiple alignments using the heuristic method of progressive alignment. Later on we added anchor and precedence constraints to our multiple alignments and we have shown how they influence each other. We discussed how to handle with constraints when we follow the guide tree

in order to solve the multiple alignment. It not surprising that it is possible to solve multiple alignments using the approach of progressive alignment with cluster trees. Due to the variability toward constraints of the cluster tree approach it is possible to add different constraints, such as anchor and precedence constraints.

We found an exponential approach to solve multiple alignments that uses cluster tree decomposition and elimination. For this approach we developed a new constraint model, related to the constraint model used in chapter 4 [21]. In contrast to the progressive alignment we were able to find an optimal alignment using this approach. Up to a certain extend, we were able to add some further constraints to this approach, namely anchor and precedence constraints.

7.1 Further perspectives

In chapter 5 we only discussed constraints for which the undirected graph that represents the relation '$\sim$' is cycle free. One could try to formulate the constraints and compute induced constraints using constraint handling rules [9]. With an inconsistency check using constraint handling rules, we would be able to enlarge our approach to constraints without any restriction. Another interesting task is to enlarge our approach in order to deal with aligned segments and segment structure constraints and to examine how they interact with other constraints.

We were able to find an exponential approach to solve an optimal multiple alignment. Sadly the order of the time and space complexity of this approach is much worse than the complexities of the that of dynamic programming[1]. We only added anchor and pecedence constraints on two sequences, where one sequence has to be the one that is represented by the structure of the cluster tree decomposition. It would be useful to find a constraint model for multiple alignments that admits to:

- solve multiple alignments more efficiently, perhaps in the same order than the method of dynamic programming
- add constraints between any sequences

Furthermore one coud try to add aligned segments and sequence structure constraints to the cluster tree decomposition of the multiple alignments as

[1] $O(n^m)$ for m sequences of the length n

well as mix several constraints and consider their influences on the cluster tree decomposition and elimination.

List of Figures

1.1 RNA compared to DNA 8

2.1 Example of a pseudoknot-free structure 14

3.1 Dynamic programming matrix of an alignment 18
3.2 Dynamic programming matrix with one anchor constraint . . 19
3.3 Dynamic programming matrix with one precedence constraint, 'left' 21
3.4 Dynamic programming matrix with one precedence constraint, 'right' 22
3.5 Dynamic programming matrix with aligned segment constraint 23
3.6 Closer look on $\mathcal{M}'$ for segment constraint 25

4.1 Cluster tree of a sequence alignment 30
4.2 CTD of an alignment with segment constraints 35
4.3 Sequence structure alignment CTD 38

5.1 Guide tree of a multiple alignment 50

Bibliography

[1] Rolf Backofen, Danny Hermelin, Gad M. Landau, and Oren Weimann. Local alignment of rna sequences with arbitrary scoring schemes. volume 4009 of *Lecture Notes in Computer Science*, pages 246–257. Springer, 2006.

[2] Rolf Backofen and Sven Siebert. Fast detection of common sequence structure patterns in RNAs. *Journal of Discrete Algorithms*, 2005. accepted.

[3] Rolf Backofen and Sebastian Will. Local sequence-structure motifs in RNA. *Journal of Bioinformatics and Computational Biology (JBCB)*, 2(4):681–698, 2004.

[4] Michael Brudno, Alexander Poliakov, Asaf Salamov, Gregory M. Cooper, Arend Sidow, Edward M. Rubin, Victor Solovyev, Serafim Batzoglou, and Inna Dubchak. Automated whole-genome multiple alignment of rat, mouse, and human. *Genome Res*, 14(4):685–92, 2004.

[5] Anke Busch and Rolf Backofen. INFO-RNA–a fast approach to inverse RNA folding. *Bioinformatics*, 22(15):1823–31, 2006.

[6] Deniz Dalli, Andreas Wilm, Indra Mainz, and Gerhard Steger. STRAL: progressive alignment of non-coding RNA using base pairing probability vectors in quadratic time. *Bioinformatics*, 22(13):1593–9, 2006.

[7] Chuong B. Do, Mahathi S. P. Mahabhashyam, Michael Brudno, and Serafim Batzoglou. ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Res*, 15(2):330–40, 2005.

[8] Sean R. Eddy. What is dynamic programming? *Nat Biotechnol*, 22(7):909–10, 2004.

[9] Thom Frühwirth. Theory and practice of constraint handling rules. *The Journal Of Logic Programming*, 19, 1994.

[10] Robert Giegerich. Lecture notes on algebraic dynamic programming. Technische Fakultät, Universität Bielefeld, 2002.

[11] Robert Giegerich, Carsten Meyer, and Peter Steffen. Towards a discipline of dynamic programming. Technical report, Faculty of Technology, Bielefeld University Postfach 10 01 31, 33501 Bielefeld, Germany, 2002.

[12] J Gorodkin, LJ Heyer, and GD Stormo. Finding the most significant common sequence and structure motifs in a set of RNA sequences. *Nucleic Acids Res*, 25(18):3724–32, 1997.

[13] H. W. Hellinga and F. M. Richards. Optimal sequence selection in proteins of known structure by simulated evolution. 91(13):5803–7, 1994.

[14] Ian Holmes. Using guide trees to construct multiple-sequence evolutionary hmms. *Bioinformatics*, 19, 2003. Proceedings of the 11th International Conference on Intelligent Systems for Molecular Biology (ISMB 2003).

[15] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between RNA structures. 9(2):371–88, 2002.

[16] Kalev Kask, Rina Dechter, and Javier Larossa. Unifying cluster-tree decompositions for automated reasoning. 2003.

[17] Kalev Kask, Rina Dechter, Javier Larossa, and Avi Dechter. Unifying cluster-tree decompositions for reasoning in graphical models. 2005.

[18] Arun Siddharth Konagurthu, James Whisstock, and Peter J. Stuckey. Progressive multiple alignment using sequence triplet optimizations and three-residue exchange costs. *J Bioinform Comput Biol*, 2(4):719–45, 2004.

[19] Giuseppe Lancia, Robert Carr, Brian Walenz, and Sorin Istrail. 101 optimal PDB structure alignments: a branch-and-cut algorithm for the maximum contact map overlap problem. ACM Press, 2001.

[20] Bin Ma, Zhuozhi Wang, and Kaizhong Zhang. Alignment between two multiple alignments. In *Combinatorial Pattern Matching, 14th Annual Symposium, CPM 2003, Morelia, Michocán, Mexico, June 25-27, 2003, Proceedings*, 2003.

[21] Sebastian Will, Anke Busch, and Rolf Backofen. Efficient sequence alignment with side-constraints by cluster tree elimination. *Constraints Journal*, 2007. submitted.

[22] Sebastian Will, Kristin Reiche, Ivo L. Hofacker, Peter F. Stadler, and Rolf Backofen. Inferring non-coding rna families and classes by means of genome-scale structure-based clustering. *PLOS Computational Biology*, 2007. to appear.