

# SPANNOID ALIGNMENT

Stefan N. Hansen, Rita Pancsa & Marcus D. Webb  
University of Oxford

August 13, 2010

## Abstract

A new approach to multiple sequence alignment is proposed. One takes a set of sequences to align, modify a guide phylogenetic tree to obtain a Steiner tree (which we refer to in this context as a Spannoid), align the full components of the tree and combine them into one alignment using the internal sequence nodes as a scaffold. The method is flexible, with the optimality and speed of alignment dependent on the size of the components and the component alignment method.

This provides a fast method of aligning thousands of sequences. We give data on the detrimental effect on the final alignment that dividing the problem up in this manner does to assess the reliability alignments of this size.

## Contents

<b>1</b>	<b>Background And Introduction</b>	<b>4</b>
1.1	The Alignment Problem . . . . .	4
1.2	Pairwise Alignment . . . . .	4
1.2.1	Needleman-Wunsch Dynamic Programming Algorithm . . . . .	4
1.3	Multiple Alignment . . . . .	5
1.3.1	Score Based Alignment . . . . .	5
1.3.2	Progressive Alignment . . . . .	6
1.3.3	Iterative Refinement Methods . . . . .	6
1.3.4	Statistical Alignment . . . . .	7
1.4	Phylogenetic Trees . . . . .	7
1.4.1	Trees . . . . .	7
1.4.2	Phylogenetic Trees . . . . .	8
1.4.3	Making A Tree From Pairwise Distances . . . . .	8
<b>2</b>	<b>Project Aims</b>	<b>10</b>
<b>3</b>	<b><math>k</math>-Restricted Steiner Trees And Spannoids</b>	<b>11</b>
3.1	Steiner trees . . . . .	11
3.2	Spannoids . . . . .	11
<b>4</b>	<b>Spannoid Alignment</b>	<b>13</b>
4.1	Overview . . . . .	13
4.2	Creating The Guide Tree . . . . .	13
4.3	Creating The Spannoid . . . . .	13
4.3.1	Edge Contractions . . . . .	14
4.4	Aligning The Components . . . . .	16
4.5	Merging The Spannoid . . . . .	17
4.6	SpanAlign . . . . .	20
<b>5</b>	<b>Testing</b>	<b>22</b>
5.1	BAlIbASE Scoring . . . . .	22
5.2	Bonphy Methods . . . . .	23
5.3	Optimisation Of StatAlign . . . . .	26
5.3.1	Data Collection . . . . .	26
5.3.2	Loglikelihood Method . . . . .	26
5.3.3	Alignment Based Method . . . . .	27
5.4	StatAlign Performance . . . . .	29
5.5	SpanAlign . . . . .	32
<b>6</b>	<b>Conclusions</b>	<b>35</b>
<b>7</b>	<b>Future Work</b>	<b>36</b>
7.1	Bonphy Optimisation . . . . .	36
7.2	Alternatives To Bonphy . . . . .	36
7.2.1	Inferring Internal Nodes . . . . .	36
7.2.2	Making A Spannoid By Clustering From Pairwise Distances . . . . .	36
7.2.3	Taking Alignment Samples For Random Spannoids . . . . .	37
7.3	Edge Lengths . . . . .	37
7.4	Parameter Estimation . . . . .	39

---

7.5	Annotation . . . . .	39
<b>A</b>	<b>Appendix</b>	<b>40</b>
A.1	SpanAlign Source Code . . . . .	40
A.2	Harvester Source Code . . . . .	43
A.2.1	Harvester Package . . . . .	44
A.2.2	Spannoid Structure Package . . . . .	50
A.2.3	Mergers Package . . . . .	58
A.2.4	Clique Mergers Package . . . . .	59
A.2.5	Pair Mergers Package . . . . .	60
A.2.6	Likelihood Strategies Package . . . . .	68
A.2.7	Edge Length Strategies Package . . . . .	68
	<b>References</b>	<b>69</b>

# 1 Background And Introduction

## 1.1 The Alignment Problem

Sequential alignment is a classic theme in Bioinformatics. It involves finding the insertion, deletion (abrev. indel) and substitution points along the elements of a set of protein, DNA or RNA sequences with a common ancestor (homologous), based on explicit probabilistic models or scoring systems.

Models for DNA sequence alignment do not take into account large structural rearrangements in chromosome structure or gene duplication, conversion or recombination. For protein or RNA sequence alignment we do not take into account the chemical, physical or spatial structure of the molecules, just the ordering of the amino acids or bases.

```

1HSTA  ---SHPTYSEMIAAAITRAEKSRGGSSRQSIQKYIKSHYKVGHNADLQIKLSIRRLAAGV
1IDY   MEVKKTSWTEEDRILYQAHKRLGNRWAEIAKLLPG--RTDNAIKNHWNSTMRRKV----
```

Figure 1: A pairwise alignment between proteins 1HSTA and 1IDY.

## 1.2 Pairwise Alignment

Pairwise sequence alignment methods are used to find the best alignment of two sequences. There are 2 main questions to answer: What do we mean by ‘best’? How do we find this ‘best’ alignment?

Best can be defined by a scoring system or probabilistic model based on empirical data or theoretical grounds. Finding better and better models is still of great importance as it underlies the whole procedure.

Finding this optimal alignment is difficult. There are

$$\binom{2n}{n} \approx \frac{4^n}{\sqrt{\pi n}}$$

possible alignments (by Stirling’s approximation) on 2 sequences of length  $n$ , so searching this space by brute force is not an option for even rather small  $n$ . However, if the scoring system is additive, dynamic programming algorithms can be used to find the optimal alignment (based on this scoring system) in  $\mathcal{O}(nm)$  time, where  $n$  and  $m$  are the lengths of the sequences. Here is an example:

### 1.2.1 Needleman-Wunsch Dynamic Programming Algorithm

The technique of dynamic programming can be applied to produce score based global alignments via the Needleman-Wunsch algorithm. In typical usage, score based protein alignments use a substitution matrix (e.g. Figure 2) to assign scores to amino-acid matches or mismatches, and a gap penalty for matching an amino acid in one sequence to a gap in the other. DNA and RNA alignments may use a scoring matrix, but in practice often simply assign a positive match score, a negative mismatch score, and a negative gap penalty. In standard dynamic programming, the score of each amino acid position is independent of the identity of its neighbors. The usage of two different gap penalties for opening a gap and for extending a gap makes alignments biologically more relevant. Typically the former is much larger than the latter, e.g.  $-10$  for gap

open and  $-2$  for gap extension. Thus, the number of gaps in an alignment is usually reduced and residues and gaps are kept together, which typically makes more biological sense. The dynamic programming method is guaranteed to find an optimal alignment given a particular scoring function; however, identifying a good scoring function is often an empirical rather than a theoretical matter. Although dynamic programming is extensible to more than two sequences, it is prohibitively slow for large numbers of or extremely long sequences.

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	5	-2	-1	-2	-1	-1	-1	0	-2	-1	-2	-1	-1	-3	-1	1	0	-3	-2	0
R	-2	7	-1	-2	-4	1	0	-3	0	-4	-3	3	-2	-3	-3	-1	-1	-3	-1	-3
N	-1	-1	7	2	-2	0	0	0	1	-3	-4	0	-2	-4	-2	1	0	-4	-2	-3
D	-2	-2	2	8	-4	0	2	-1	-1	-4	-4	-1	-4	-5	-1	0	-1	-5	-3	-4
C	-1	-4	-2	-4	13	-3	-3	-3	-3	-2	-2	-3	-2	-2	-4	-1	-1	-5	-3	-1
Q	-1	1	0	0	-3	7	2	-2	1	-3	-2	2	0	-4	-1	0	-1	-1	-1	-3
E	-1	0	0	2	-3	2	6	-3	0	-4	-3	1	-2	-3	-1	-1	-1	-3	-2	-3
G	0	-3	0	-1	-3	-2	-3	8	-2	-4	-4	-2	-3	-4	-2	0	-2	-3	-3	-4
H	-2	0	1	-1	-3	1	0	-2	10	-4	-3	0	-1	-1	-2	-1	-2	-3	2	-4
I	-1	-4	-3	-4	-2	-3	-4	-4	-4	5	2	-3	2	0	-3	-3	-1	-3	-1	4
L	-2	-3	-4	-4	-2	-2	-3	-4	-3	2	5	-3	3	1	-4	-3	-1	-2	-1	1
K	-1	3	0	-1	-3	2	1	-2	0	-3	-3	6	-2	-4	-1	0	-1	-3	-2	-3
M	-1	-2	-2	-4	-2	0	-2	-3	-1	2	3	-2	7	0	-3	-2	-1	-1	0	1
F	-3	-3	-4	-5	-2	-4	-3	-4	-1	0	1	-4	0	8	-4	-3	-2	1	4	-1
P	-1	-3	-2	-1	-4	-1	-1	-2	-2	-3	-4	-1	-3	-4	10	-1	-1	-4	-3	-3
S	1	-1	1	0	-1	0	-1	0	-1	-3	-3	0	-2	-3	-1	5	2	-4	-2	-2
T	0	-1	0	-1	-1	-1	-1	-2	-2	-1	-1	-1	-1	-2	-1	2	5	-3	-2	0
W	-3	-3	-4	-5	-5	-1	-3	-3	-3	-3	-2	-3	-1	1	-4	-4	-4	15	2	-3
Y	-2	-1	-2	-3	-3	-1	-2	-3	2	-1	-1	-2	0	4	-3	-2	-2	2	8	-1
V	0	-3	-3	-4	-1	-3	-3	-4	-4	4	1	-3	1	-1	-3	-2	0	-3	-1	5

Figure 2: BLOSUM50 score matrix for proteins.

### 1.3 Multiple Alignment

Multiple sequence alignment is an extension of pairwise alignment to incorporate more than two sequences at a time. They are often used in identifying conserved sequence regions across a group of sequences hypothesized to be evolutionarily related. Such conserved sequence motifs can be used in conjunction with structural and functional information to locate the catalytic active sites of enzymes.

Biologists can produce high quality multiple sequence alignments by hand using expert knowledge of protein sequence evolution [2]. However, this is tedious, time consuming and sometimes impossible for many sequences, so automatic multiple sequence alignment has been a topic of extensive research over the past four decades.

#### 1.3.1 Score Based Alignment

When doing score based multiple sequence alignment one assumes that the individual columns of the alignment are statistically independent. Using this assumption, a scoring function for a multiple sequence alignment  $m$  would be of the form

$$S(m) = G + \sum_i S(m_i),$$

where  $m_i$  is the  $i$ 'th column of the alignment and hence  $S(m_i)$  is the score for the  $i$ 'th column. The function  $G$  is a gap penalty function which for now is unspecified. Most multiple alignment methods use a gap penalty function that pays a higher cost

for opening a gap than extending it. This means that successive gaps are not treated as being independent. One method for calculating the score of the  $i$ 'th column is called *SP* (Sum of Pairs) scores. The *SP* score uses a score matrix such as the BLOSUM50 matrix seen in Figure 2. The score for the  $i$ 'th column is defined as

$$S(m_i) = \sum_{k < l} s(m_i^k, m_i^l),$$

where  $m_i^k$  is the  $i$ 'th residue of the  $k$ 'th sequence and  $s(a, b)$  is the entry in the scoring matrix corresponding to the residues  $a$  and  $b$ . One could now use dynamic programming as suggested in Section 1.2.1 to obtain an optimal multiple alignment, but as mentioned this is very slow for large numbers of or extremely long sequences. In case of using the *SP* scores the computational time is  $\mathcal{O}(k^2 2^k n^k)$ , where  $k$  is the number of sequences and  $n$  is the length. Another method for calculating the score is to minimise the entropy. This will result in computational time of  $\mathcal{O}(k 2^k n^k)$  if the column scores are computed in  $\mathcal{O}(k)$  as with the minimum entropy method. Both methods are extremely slow, and therefore a widely used method is the progressive alignment method.

### 1.3.2 Progressive Alignment

In progressive alignment one starts off by choosing two sequences and align those by the pairwise alignment method and this alignment is then held fixed. A third sequence is now chosen and aligned to the first alignment and so on. When all sequences have been aligned we obtain a final alignment. There are three things to keep in mind when doing progressive alignment. How do we decide optimally which sequence is next to be aligned? Should we allow the fixed alignment to be aligned with other alignments rather than just sequences? And lastly, which method (i.e. *SP* scores or minimum entropy) should be used when computing the successive alignments. A guide tree is often used to get an overview of how the sequences are related and then one would choose closely related sequences to align first and move upwards in the tree.

This method is very fast and often the final alignment is a fairly good approximation. The downside however is that the method does not do any overall optimisation of a scoring function. It does only optimise scoring functions in the small steps when aligning sequences and alignments. Examples of progressive alignment algorithms are the Feng-Doolittle algorithm and the ClustalW algorithm (which incorporates the notion of profile-based alignments into the progressive alignment method).

### 1.3.3 Iterative Refinement Methods

A main issue with the progressive alignment method is that once a number of sequences has been aligned their mutual relationship are 'frozen'. This means that their alignment cannot change as you add more sequences to the alignment. Iterative refinement methods was invented to overcome this. It works by generating an initial alignment such as the one obtained using the progressive alignment method. Then a single or more sequences are taken out of the alignment according to a set of specified rules and these are realigned to a profile of the remaining alignment. This either results in an increase of the overall score or the overall score stays the same. After this another sequence is chosen and the steps are repeated. This is done until the alignment does not change and this will be the final alignment. A well known iterative refinement method is the Barton-Sternberg algorithm.

### 1.3.4 Statistical Alignment

Score based alignments were developed in the 70s, 80s and 90s, but are still the basis of many alignment programs still used today. Statistical Alignment, which rose to prominence in the last decade, uses explicit stochastic models of indel and substitution such as Hidden Markov Models (HMM's), a much more sophisticated approach in terms of testing evolutionary hypotheses and estimating parameters [5]. One of the most known statistical models is the TKF91 founded by Thorne, Kishino and Felsenstein in 1991 [6]. In the TKF91 model you allow substitutions, insertions and deletions of residues. The substitutions are described as a prescribed reversible continuous-time Markov process with state space being the possible characters and the insertion and deletions are modelled as a birth-death process with insertion rates  $\lambda$  and deletion rates  $\mu$ . Given two sequences  $A$  and  $B$  of length  $s_A$  and  $s_B$  respectively, the likelihood of  $A$  and  $B$  can be written as

$$L_\theta(A, B) = \pi_A^{r_A} \pi_G^{r_G} \pi_C^{r_C} \pi_T^{r_T} \gamma_{s_A} P_t(B | A, \theta), \quad (1)$$

where  $\pi_i$  and  $r_i$  are the probability and occurrence (in sequence A) of the nucleotide  $i$  for  $i = A, G, C, T$ . The parameter  $\theta$  is the vector of all unknown parameters in the model, that is the probability of the four residues and insertion, deletions and substitution rates. Furthermore  $\gamma_{s_A}$  is the equilibrium probability of sequences of length  $s_A$  and  $P_t(B | A, \theta)$  is the transition probability from sequences A to B in time  $t$  given the parameter  $\theta$ .

Along with the TKF91 model came two algorithms, namely the alignment algorithm and the parameter estimation algorithm. The first is an algorithm that finds an optimal alignment in the same manner as other known score based algorithm using trace back through a matrix. But unlike the standard dynamic programming algorithm the matrix consists not of scores but of likelihoods and thus eliminating the subjective factor in the alignment problem. The parameter estimation algorithm uses the likelihood equation in (1) and uses numerical methods to find the maximum likelihood solution of  $\theta$ .

An extension to the TKF91 model, that introduced an 'affine gap cost' in the statistical model, was later made. This new model called TKF92 allows insertions and deletions of whole segments whose length are taken from an arbitrary distribution due to the fact that insertions and deletions often occurs in more than one single nucleotide.

## 1.4 Phylogenetic Trees

The genetic similarity between species of all organisms studies suggests that all living things on Earth have a common ancestor. Therefore any set of species is related somehow, and we call this relationship a *phylogeny*. The relationship can usually be represented by a tree, so we call it a *phylogenetic tree*.

### 1.4.1 Trees

A graph  $G$  is a tuple  $(V, E)$ , where  $V$  is the set of vertices or nodes and  $E \subseteq V \times V$  is the set of edges, where  $(v_1, v_2) \in E$  if and only if there is an edge between  $v_1$  and  $v_2$ . A graph  $G = (V, E)$  is said to be *undirected* if  $(v_1, v_2) \in E$  if and only if  $(v_2, v_1) \in E$ , and an undirected graph is then said to be *connected* if for all  $v_1 \in V$  exists  $v_2 \in V$  such that  $(v_1, v_2) \in E$ . A tree is a connected, acyclic graph and a leaf

node is a node with only one neighbouring node. All other nodes are called internal nodes.

In this project, we will only consider binary trees, where all internal nodes have 3 neighbours. This is not a serious limitation, because any other branching pattern can be approximated by a binary tree in which some of the branches are very short.

### 1.4.2 Phylogenetic Trees

In a phylogenetic tree, nodes correspond to sequences. The leaf nodes are sequences of observed current species and internal nodes represent evolutionary ancestors we have not observed.

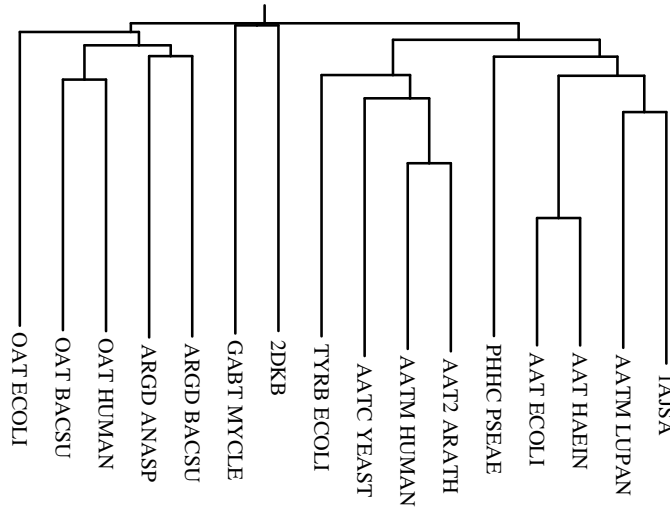


Figure 3: A binary tree for the phylogeny of 16 protein sequences.

### 1.4.3 Making A Tree From Pairwise Distances

Building a phylogenetic tree from a family of sequences can be done intuitively by first defining a set of distances  $d_{ij}$  between each pair  $i, j$  of sequences. The simplest example for  $d_{ij}$  is the fraction  $f$  of the sites where the two sequences differ (after aligning the two sequences by some method). This works for small fractions  $f$ , but when the two sequences are unrelated we see that random substitutions will cause  $f$  to approach the fraction of the differences caused by chance. We would, however, like the distance to tend to infinity. The Jukes-Cantor distance is  $d_{ij} = -\frac{3}{4} \log(1 - 4f/3)$ , which tends to infinity as the equilibrium value of  $f$  is approached (0.75 of residues different). There are a good few methods for tree-making based on pairwise distances, but the one we shall consider is Neighbour-Joining.

**Neighbour-Joining** The edge lengths of a tree are said to be additive if the distance between any pair of leaves is the sum of the length of the edges on the path connecting them. So given a tree with additive lengths, we can try to reconstruct it from the pairwise distances of its leaves. We construct a tree  $T$ , keeping a list of active leaf nodes in the tree,  $L$ .  $L$  corresponds to the current remaining set of leaf nodes to put into  $T$  and  $T$  corresponds to the tree being built from the leaf nodes  $L$  in the iterations. Define:

$$D_{ij} = d_{ij} - (r_i + r_j)$$

where,

$$r_i = \frac{1}{|L| - 2} \sum_{k \in L} d_{ik}$$

It is claimed that a pair of leaves  $i, j$  for which  $D_{ij}$  is minimal will be neighbouring leaves; this is proved in [2] Chapter 7.

---

**Algorithm 1** Neighbour-Joining

---

**Initialisation:** Define  $T$  to be the set of leaf nodes, one for each given sequence, and put  $L = T$ .

**while**  $|L| > 2$  **do**

    Pick a pair  $i, j$  in  $L$  for which  $D_{ij}$ , defined above, is minimal.

    Define a new node  $k$  and set  $d_{km} = \frac{1}{2}(d_{im} + d_{jm} - d_{ij})$ , for all  $m$  in  $L$ .

    Add  $k$  to  $T$  with edges of lengths  $d_{ik} = \frac{1}{2}(d_{ij} + r_i - r_j)$ ,  $d_{jk} = d_{ij} - d_{ik}$ , joining  $k$  to  $i$  and  $j$ , respectively.

    Remove  $i$  and  $j$  from  $L$  and add  $k$ .

**Termination:** Add the remaining edge between  $i$  and  $j$ , with length  $d_{ij}$ .

---

## 2 Project Aims

Initially we had several project aims. The main idea is to use the notion of  $k$ -spannoids as a tool to increase the number of sequences alignable and hopefully still end up with a fairly good alignment. Another key tool is the statistical alignment program StatAlign. According to [5] the MCMC (Markov Chain Monte Carlo) technique used in StatAlign allows only analysis of sequences of order 10-15 because of the hard time complexity. Since we believe that statistical alignment is the best method for aligning sequences, we would want to be able to align more than 15 sequences using statistical alignment. The project aims can be outlined as follows.

- Describe a framework for the spannoid notion, so it is easy to implement in a Java program.
- Analyse the performance of StatAlign to conclude whether to use this or another alignment program on the full components.
- Make a versatile program that automatically aligns many sequences and can handle several alignment programs on the components.
- Be able to align thousands of sequences in fairly short time.
- Test the performance of our alignment program to other existing programs.

Depending on the time left we would want to find a method for doing parameter estimation (insertion, deletions and substitution rates) in the spannoid framework. Since aligning thousands of sequences can be very useful in terms of annotation, it is worth looking into the problem of combining annotations on the full components, but this will most likely be out the scope of this project

### 3 $k$ -Restricted Steiner Trees And Spannoids

#### 3.1 Steiner trees

When constructing a multiple alignment of a set of sequences  $S$  we will be using Steiner trees in order to decompose the alignment problem into smaller alignment problems. Given a set of sequences  $S$ , a spanning tree is a tree where every edge connects exactly two sequences. We will refer to sequences as terminal nodes and denote them by solid circles. If we include common ancestors to our tree we obtain a Steiner tree, where the common ancestors will correspond to internal Steiner nodes. Here internal means that a node is not a leaf. The Steiner nodes will be denoted by hollow circles. Examples of a spanning tree and a Steiner tree are shown in Figure 4.

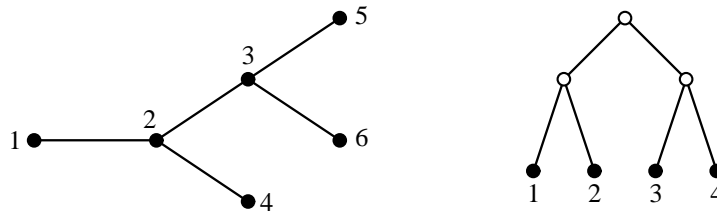


Figure 4: At left a spanning tree on the set of sequences  $S = \{1, 2, 3, 4, 5, 6\}$ . At right a Steiner tree on the sequences  $S = \{1, 2, 3, 4\}$  with three Steiner nodes.

We say that a Steiner tree on a subset  $G \subseteq S$  of the terminal nodes is a *full component* if all the internal nodes are Steiner nodes. If the full components of a Steiner tree has at most  $k$  terminal nodes we call it a  $k$ -restricted Steiner tree, which is illustrated in Figure 5. Note that two full components are connected through a single common sequence. Full components cannot be connected through two or more common sequences, since this will introduce a cycle in our tree. The multiple alignment problem becomes much easier if we are looking at a  $k$ -restricted Steiner tree, since aligning the full components will involve at most  $k$  sequences. Of course  $k$  affects both computational time and how good the alignment approximation is. If  $k = 2$  we will align full components with at most 2 terminal nodes separately. That is we pairwise align the sequences. Since we are not expecting this to be a very good approximation, we would want  $k \geq 3$ .

#### 3.2 Spannoids

Given a Steiner tree on a set  $S$  of terminal nodes we can compute the full components and introduce a hypergraph called the *Spannoid* of the Steiner tree. The Spannoid will be a hypergraph on the full components where an edge between two full components means that they have a common sequence. Similarly, a  $k$ -Spannoid is the hypergraph induced by a  $k$ -restricted Steiner tree.

The 3-Spannoid of the Steiner tree given in Figure 6 is seen in Figure 7, and the common sequences are denoted by stars. The Spannoid will contain as many nodes as there are full components in the original tree and as many stars as there are common sequences.

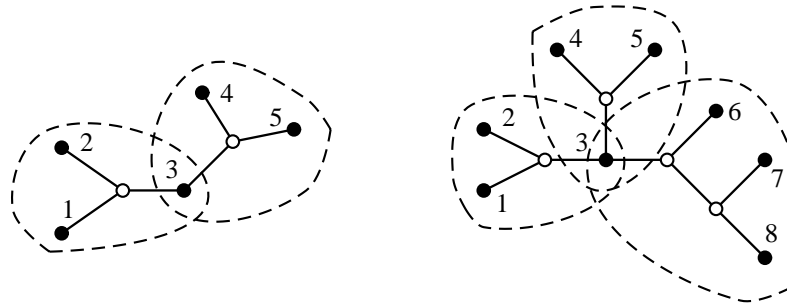


Figure 5: To the left a 3-restricted Steiner tree with  $G_1 = \{1, 2, 3\}$  og  $G_2 = \{3, 4, 5\}$ . The dashed lines encircles the full components. Had the node 3 been a Steiner node it would have been a 5-restricted Steiner tree. To the right a 4-restricted Steiner tree with  $G_1 = \{1, 2, 3\}$ ,  $G_2 = \{3, 4, 5\}$  and  $G_3 = \{3, 6, 7, 8\}$ .

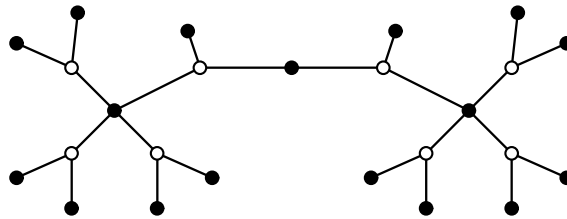


Figure 6: Another example of a bigger 3-restricted Steiner tree with eight full components. Here three of the terminal nodes corresponds to common sequences.

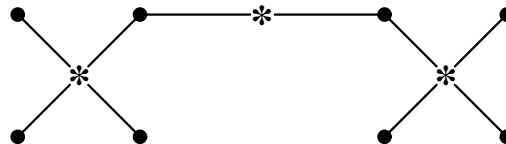


Figure 7: The Spannoid of the Steiner tree given in Figure 6. Every node in this tree corresponds to a full component and the common sequences are denoted by stars.

## 4 Spannoid Alignment

### 4.1 Overview

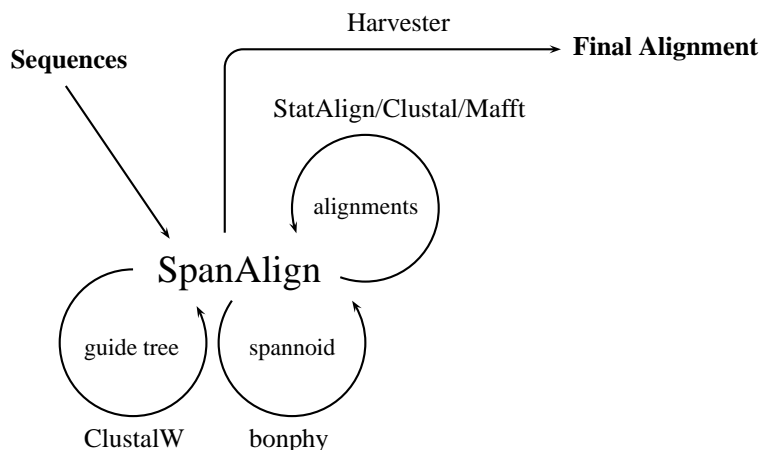


Figure 8: An overview of the method we are using to obtain a final alignment.

Figure 8 illustrates how we from a set of sequences obtain a final alignment. The whole procedure consists of four steps. The first three steps involves already existing software such as Clustal, bonphy and StatAlign. The last step makes use of the java program Harvester which is the main product of this project. In short the procedure takes in a number of sequences, computes a guide phylogeny and decomposes this tree into smaller subtrees according to some rules. From here we just run already existing alignment programs on the smaller subtrees and at last Harvester combines all pieces into one final alignment. How Harvester functions is explained in Section 4.5. The whole procedure relies heavily on the notion of spannoids as introduced in Section 3. A more thorough explanation of the procedure will be given with an example. So let us consider the 14 sequences in Figure 9 and assume we want to align them.

### 4.2 Creating The Guide Tree

These 14 sequences will be read by Clustal (you can input sequences in .FASTA format), and it will construct a guide tree using the Neighbour-Joining algorithm. This guide tree is shown in Figure 10. Obtaining a guide tree is very fast even for many sequences, so it will not slow down the overall procedure. The Neighbour-Joining algorithm is used, because it provides an unrooted tree which is exactly what bonphy requires.

### 4.3 Creating The Spannoid

With this guide tree, which can be arbitrarily big depending on the number of sequences, we want to obtain a  $k$ -restricted Steiner tree. In order to do so, we need to contract some edges. We will now discuss how to do optimal edge contractions.

```

>1IDY
MEVKKTSWTEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHWNSTMRRKV
>MYB3_MAIZE
DLKRGNFDAEDDLIVKLHSLGKNSLIARLPGRTDNEIKNYWNTHVRRK
>GL1_ARATH
NVNKGNFTEQEEDLIIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTHLSKK
>MYB3_HORVU
DLKRGCFSSQEEEDHIVALHQILGNRWSQIASHLPGRTDNEIKNFWSNCIKKK
>MYB2_PHYPA
DLKRGIFSEAEENLIDLHATLGNRWSRIAAQLPGRTDNEIKNYWNTRLKKR
>1HSTA
SHPTYSEMIAAAIRAESRGGSSRQSIQYIKYKSHYKVGHNADLQIKLSIRLLAAGV
>H1L_MYTTR
KPSTLSMIVAAITAMKNRKGSSVQAIRKYILANNKGINTSHLGSAMKLAFAKGLKSGV
>H1_1_ARATH
SHPTYEEMIKDAIVTLKERTGSSQYAIQKFIEEKRKELPPTFRKLLLLNLKRLVASGK
>H1_DICDI
NHPTYQVMISTAIAHYKDRGTGSSQPAIIKYEANYNVAPDTFKTQLKLALKRLVAKGT
>1AOY
MRSQAEELVKAFAKALLKEEFSSQGEIVAALQEQGFDNINQSKVSRMLTKFGAVRT
>ARGR_STRCL
MARHRRIVDILNRQPVRSSQSLAKLLADNGLSVTQATLSRDLDELGAVKI
>G3273713
ENLNPVTRTRARQALILQILDKQKVTQVQLSELLLDEGIDITQATLSRDLDELGARKV
>AHRC_BACSU
MNGQQRHIKIREIITSNEIETQDELVDMLKQDGYKVTQATVSRDIKELHLVKV
>ARGR_BACST
MNGQQRHIKIREIIMSNDIETQDELVDRLREAGFNVTQATVSRDIKEMQLVKV
    
```

Figure 9: An example of 14 sequences with an average length of 53 residues.

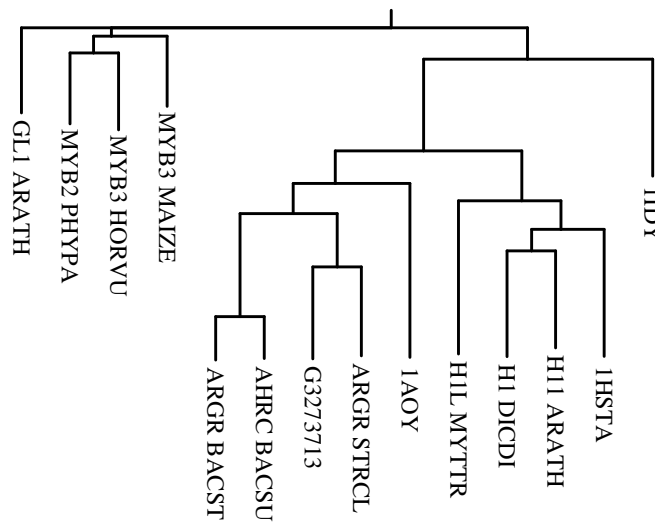


Figure 10: The guide tree obtained by Clustal.

### 4.3.1 Edge Contractions

Let  $k \geq 2$  be a fixed number. Given a set  $S$  of sequences one might compute a guide tree using e.g. the Neighbour-Joining or the UPGMA algorithm. This tree is not neces-

sarily a  $k$ -restricted Steiner tree. One method to obtain a  $k$ -restricted Steiner tree would be contracting edges, that is to bring terminal nodes in as internal nodes and thereby getting rid of a Steiner node. This idea is illustrated in Figure 11.

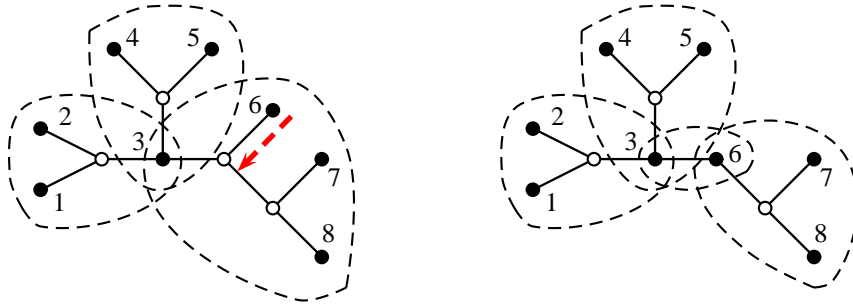


Figure 11: To the left is seen the 4-restricted Steiner tree from Figure 5. Bringing the terminal node 6 in as an internal node we obtain the 3-restricted Steiner tree to the right.

So by doing edge contractions we decrease the size of the full components, but on the other hand we increase the number of full components. The question is now how to contract edges optimally and what is meant by optimal. Since we want our new tree to be as close to the original tree optimal could, in the light of Figure 11, mean that the total weight of the contracted edges should be minimal. But instead of seeing edge contractions as pulling in terminal nodes, we could see it as elongating the two edges incident to the Steiner node we want to eliminate. This idea is illustrated in Figure 12.

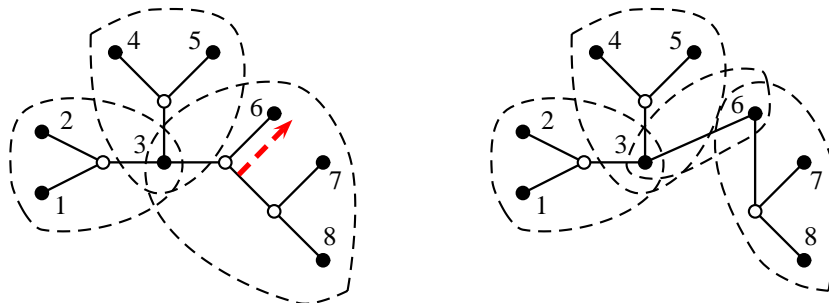


Figure 12: The same Steiner tree as in Figure 11, but here we elongate edges incident to the Steiner node we want to remove and thereby not moving the position of terminal node 6.

Seeing edge contractions this way, optimal could mean that we minimise the total edge weight in the tree obtained by elongating the relevant edges.

The program bonphy offers 12 different methods for contracting edges. Eight of these are discussed in Section 5.2. In the case of our example, setting  $k = 3$ , we might end up with the 3-restricted Steiner tree seen in Figure 13. The full components are {MYB2 PHYPA, MYB3 HORVU, **GL1 ARATH**}, {**GL1 ARATH**, MYB3 MAIZE}, {**GL1 ARATH**, **1IDY**}, {**1IDY**, **H1L MYTTR**}, {**H1L MYTTR**, 1HSTA}, {**H1L MYTTR**, H1 DICDI, H11 ARATH}, {**1IDY**, **1AOY**}, {**1AOY**, ARGR BACST, AHRC BACSU} and {**1AOY**, G3273713, ARGR STRCL}. Here boldfaced sequences

are the common sequences which tie the subtrees together. So what we got was a 3-Spannoid with 9 full components of size 2 and 3. Setting  $k$  bigger we might have obtained fewer but bigger full components.

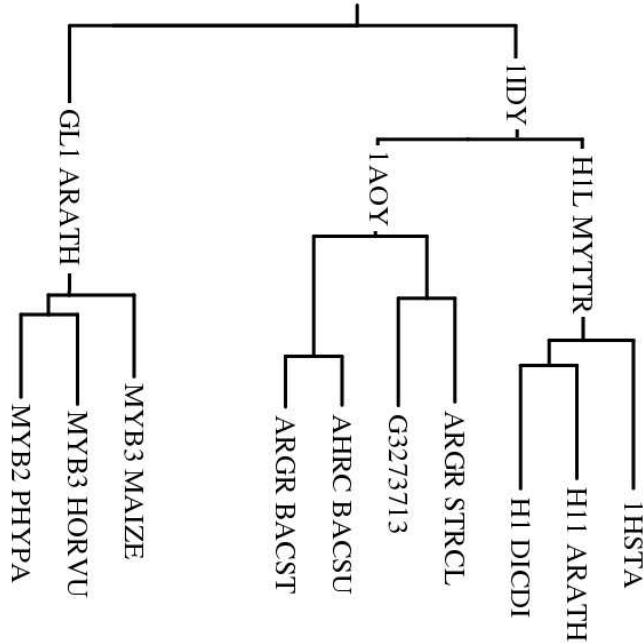


Figure 13: The 3-restricted Steiner tree obtained by bonphy. This is also referred to as the 3-Spannoid. The common sequences in this case are GL1 ARATH, 1IDY, 1AOY and H1L MYTTR.

#### 4.4 Aligning The Components

Now we are facing 9 small alignment problems. In this step one could use any multiple sequence alignment software available to align the components. We have implemented StatAlign, Clustal and Mafft into our SpanAlign program and the user can choose whichever he or she prefers. Some of the aligned components is shown in Figure 14.

If computational time is not a problem one should choose  $k$  as big as possible. As shown in Figure 20 and Figure 21, the higher  $k/n$  the better result we get. This also seems intuitive because when  $k$  increases we use more of the potential of the alignment programs and fewer components have to be merged in the end. In the case of using StatAlign on the component computational time is a dominant factor since aligning more than 7-10 sequences can take days with the optimal MCMC parameters. So here we would suggest putting  $k \leq 6$  or maybe even  $k \leq 5$ .

```

>ARGR_STRCL
-----MARHRRIVDILNRQPVRSQSOLAKLLADNGLS-VTQATLSRDLDDELGAVKI
>G3273713
ENLNPVTRTRARQALILQILDKQKVTSQVQLSELLLDEGID-ITQATLSRDLDDELGARKV
>1AOY
-MRSSAKQEELVKAFKALLKEEFSSQGEIVAALQEQQFDNINQSKVSRMLTKFGAVRT

>AHRG_BACSU
MNGQQRH---IK-IREIITSNEIETQDELVDMLKQDGYK-VTQATVSRDIKELHLVKV
>ARGR_BACST
MNGQQRH---IK-IREIIMSNDIETQDELVDRLREAGFN-VTQATVSRDIKEMQLVKV
>1AOY
MRSSAKQEELVKAFKALLKEEFSSQGEIVAALQEQQFDNINQSKVSRMLTKFGAVRT

>1IDY
MEVVKTSWTEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHNWSTMRKRV
>GLI_ARATH
-NVNGNFTEQEEDLIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTHLSKK-

```

Figure 14: Three of the full components aligned using Clustal. This can be done using any multiple sequence alignment program.

#### 4.5 Merging The Spannoid

We implemented a java prototype program called Harvester, to merge a given set of alignments (the spannoid) into one final alignment.

It basically stores the spannoid as a hypergraph of alignments and merges the components one hyperedge at a time. There are three major questions: In what order should we merge the hyperedges? How shall we merge the components of a single hyperedge? Should we realign parts of merged alignments?

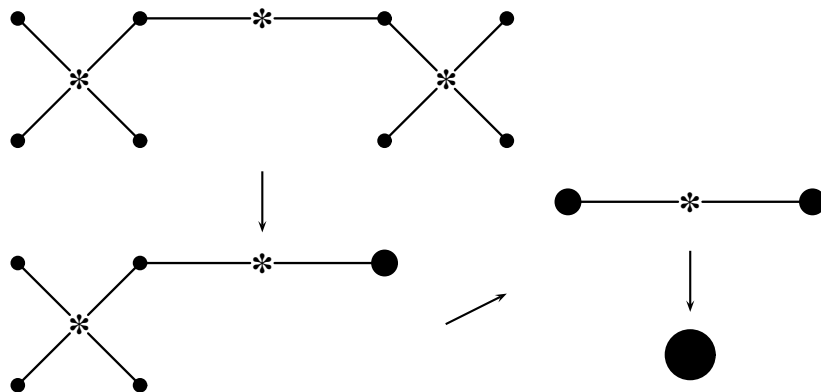


Figure 15: Harvester merges the components one hyperedge at a time until there is only one component left.

How the order in which we merge the hyperedges will affect the final alignment is not clear, so we implemented 2 methods. We implemented a "most hyper first" approach in which the hyperedges with most components are merged first, and a "leaves first" approach in which we merge the hyperedges of leaves (nodes with only one neighbouring node) of the spannoid first.

The base step in merging the spannoid is merging a pair of alignments with a common sequence. This can be done repeatedly to merge a set of alignments with a common sequence (we refer to these as *cliques*). To merge two alignments with a common sequence, we have a basic strategy called SumGap.

```

>1IDY
MEVKKTSW----TEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHWNSTMRRKV
>GL1 ARATH
-NVKNKNFFTFTEQEEDLIIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTHLSKK-
>MYB2 PHYPA
-DLKRGI FA-KRSEAEENLILDLHATLGNRWSRIA AOLPGRTDNEIKNYWNTLKKR-
>MYB3 HORVU
-DLKRGCFSFK-SQQEEDHIVALHQLLGNRWSQIASHLPGRTDNEIKNFWNSCIKKK-
>MYB3 MAIZE
-DLKRGNFT-FSFAEDDLIVKLHSLGKNSLIAARLPGR TDNEIKNYWNTHVRRK-
>MYBP MAIZE
ADVYKRGNIS-FFAKEEDI I I I KLHATLGNRWSLIAASHLPGRTDNEIKNYWNSHLSRQ-

>1AOY
-MRSSAKQE-EELVKAFKALLKEEFSSQGEIVAALQE QGFDNINQSKVS-RMLTKFGAVRT
>1HSTA
SHPTYSEMIAAAAIRA EKSRGGS-SRQSIQKYIKSHYKVGHNADLQ--IK-LSIRLLAAGV
>1IDY
-MEVKKT SW--TEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHWNSTMRRKV----
>H1L MYTTR
K-PSLMSMIVAVAITAMKNR KGS-SVQAIRKYILANNKGINTSHLGSAMK-LAF AKGLKSGV
>H1 ECHCR
AHP PV IDMITAAAIAAQKERRGS-SVAKIQSYIAAKYRCDINA-LNPHIR-RALKNQVKS GA

```

Figure 16: Two alignments with common sequence 1IDY

We note the gaps created in the common sequence and for every gap in one of the common sequences we create another gap of the same size in the same position in every sequence of the other alignment. Note that if there is a gap in the same place in the common sequences in each alignment, a gap the size of the sum of the sizes of these gaps is created (hence the name SumGap). The gaps created in one alignment should be made to the left of the residues and in the other to the right, as shown in Figure 17.

After this, the common sequences in each alignment have exactly the same gap structure, so we can just put the sequences above each other in a new, merged alignment. This may seem a little sloppy, but the assumptions are justified by the way we construct the spannoid from a guide phylogeny. The situation of a gap in the same position in the common sequence in both alignments corresponds to an assumption that the residues that are aligned to the gap in the common sequence in each alignment are independent insertions in the common sequence, justified by seeing the components as independent subphylogenies with the common sequence being a common root. Simply putting the sequences above each other in a new alignment corresponds to an assumption of a sort of transitivity on pairwise alignment, justified by the fact that the sequences in neighbouring components are well related as they are close together in the guide tree.

A second approach is to realign the gaps where we had a gap in the same place on both common sequences. This is a minor improvement as this situation does not occur very often. This approach called StatAlignGapsPairMerger has been implemented in Harvester and works, but as mentioned it rarely changes the alignment. Therefore a third approach was done.

The third approach is to realign specific regions of the merged alignment. Deciding which regions to realign can be done when using statistical alignment software such as StatAlign to align the components, because a likelihood is assigned to every column.

```

>GL1 ARATH
- -NVNKGNF -- FTFF TEQEEELIIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTLHLSKK- -----
>MYB2 PHYPA
- -DLKRGIF -- A-KR SEAEENLILDLHATLGNRWSRIAAQLPGRDTDNEIKNYWNTLHLSKK- -----
>MYB3 HORVU
- -DLKRGCF -- SFK- SQQEEDHIVALHQILGNRWSQIASHLPGRDTDNEIKNFWNSCIKKK- -----
>MYB3 MAIZE
- -DLKRGNF -- T-FS FADEDDLIVKLHSLGKWSLIAARLPGRDTDNEIKNYWNTLVRRK- -----
>MYBP MAIZE
- ADVKRGNI -- S-FF AKEEEDI I I K LHATLGNRWSLIAASHLPGRDTDNEIKNYWNSHLSRQ- -----

>1IDY
- MEVKKTSW -- ---- TEEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHWNSTMRRKV -----

>1AOY
- MRSSAKQE -E ---- ELVKAFKALLKEEFSSQGEIVAALQEQQGFDNINQSKVS-RMLTKF GAVRT
>1HSTA
S HPTYSEMI AA ---- AAIRAEKSRGGS-SRQSIQKYIKSHYKVGHNADLQ--IK-LSIRRL LAAGV
>H1L MYTTR
K -PSTLSMI VA ---- VAITAMKNRKG-SVQAIRKYILANNKGINTSHLGSAMK-LAFAGK LKSGV
>H1 ECHCR
A HPPVIDMI TA ---- AAIAAQKERRGS-SVAKIQSYIAAKYRCDINA-LNPHIR-RALKNQ VKSGA

```

Figure 17: The two alignments merged using SumGap method.

```

>GL1 ARATH
- -NVNKGNF FTFF TEQEEELIIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTLHLSKK- -----
>MYB2 PHYPA
- -DLKRGIF -AKR SEAEENLILDLHATLGNRWSRIAAQLPGRDTDNEIKNYWNTLHLSKK- -----
>MYB3 HORVU
- -DLKRGCF -SFK SQQEEDHIVALHQILGNRWSQIASHLPGRDTDNEIKNFWNSCIKKK- -----
>MYB3 MAIZE
- -DLKRGNF -TFS FADEDDLIVKLHSLGKWSLIAARLPGRDTDNEIKNYWNTLVRRK- -----
>MYBP MAIZE
- ADVKRGNI -SFF AKEEEDI I I K LHATLGNRWSLIAASHLPGRDTDNEIKNYWNSHLSRQ- -----

>1IDY
- MEVKKTSW ---- TEEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHWNSTMRRKV -----

>1AOY
- MRSSAKQE --E- ELVKAFKALLKEEFSSQGEIVAALQEQQGFDNINQSKVS-RMLTKF GAVRT
>1HSTA
S HPTYSEMI -AA- AAIRAEKSRGGS-SRQSIQKYIKSHYKVGHNADLQ--IK-LSIRRL LAAGV
>H1L MYTTR
K -PSTLSMI -VA- VAITAMKNRKG-SVQAIRKYILANNKGINTSHLGSAMK-LAFAGK LKSGV
>H1 ECHCR
A HPPVIDMI -TA- AAIAAQKERRGS-SVAKIQSYIAAKYRCDINA-LNPHIR-RALKNQ VKSGA

```

Figure 18: The two alignments merged using SumGap method after realigning the common gap.

Then by setting a likelihood threshold one can pick out regions (selection of successive columns) where the likelihood is below the threshold. These regions can then be realigned using StatAlign once again. This either results in a region with a higher overall likelihood and this will replace the old region or the realigning will not improve the region and the old region is kept. This approach is called StatAlignLowLikelihood-PairMerger and it is implemented in Harvester.

Figure 19 shows how we end up with a final alignment from the example earlier. The alignment is obtained by using the SumGapPairMerger.

```

>1HSTA
S-HPTYSE-MIAAAIRA EKSRGGSSRQSIQKYIKSHYKVGHNADL--QIKLSIRRLAAGV-----
>H1L_MYTTR
--KPSTLS-MIVAAITAMKNRKGSSVQAIRKYILANNKGINTSHLGSAMKLAFAKGLKSGV-----
>H11_ARATH
-SHPTYEE-MIKDAIVTLKERTGSSQYAIQKFIEEKRK-ELPPTFRKLLLLNLKRLVASGK-----
>H1_DICDI
-NHPTYQV-MISTAIAHYKDRGTGSSQPAIIKYIEANYN-VAPDTFKTQLKLALKRLVAKGT-----
>1IDY
-----ME-VKKTSWTEEDRILYQAHKRLGNRWAEIAKLLPGRTDNAIKNHNWSTMRRKV-----
>1AOY
-----MRSSAKQEELVKAFKALLK--EEKFSSQGEI VAALQE QGFDNINQSKVSRMLTKFGAVRT
>MYB3_MAIZE
-----D-LKRGNFTEADDDLI VKLHSL LGNKWSLIAARLPGRTDNEIKNYWNTHVRRK-----
>GL1_ARATH
-----N-VNKGNFTEQEE DL IIRLHKLLGNRWSLIAKRVPGRTDNQVKNYWNTHLSKK-----
>MYB3_HORVU
-----D-LKRGCFSSQEE DHIVALHQILGNRWSQIASHLPGRTDNEIKNFWNSCIKKK-----
>MYB2_PHYPA
-----D-LKRGIFSEAEENLILDHATLGNRWSRIAAQLPGRTDNEIKNYWNTRLKRR-----
>AHRC_BACSU
-----MNLGQRH---IK-IREIIT--SNEIETQDELVDMLKQDGYK-VTQATVSRDIKELHLVKV
>ARGR_BACST
-----MNLGQRH---IK-IREIIM--SNDIETQDELVDRLREAGFN-VTQATVSRDIKEMQLVKV
>ARGR_STRCL
-----MARHRRIVDILN--RQPVRSQLAKLLADNGLS-VTQATLSRDLDELGAVKI
>G3273713
-----ENLNPVTRTRARQALILQILD--KQKVT SQVQLSELLLEDEGID-ITQATLSRDLDELGARKV

```

Figure 19: The final alignment after merging the full components from Figure 14 using Harvester with the SumGapPairMerger.

## 4.6 SpanAlign

SpanAlign is a program written in the perl programming language which integrates the inputs, outputs and parameters of all the programs used for the Spannoid alignment.

SpanAlign inputs a fasta file which contains all the sequences we want to align. It calls ClustalW or ClustalW2 to create the guide tree for the input sequences with the Neighbour-Joining method. Then it calls the python program bonphy to create the Spannoid from the guide tree. When bonphy produces the Spannoid SpanAlign creates small fasta sequence files for all the full components and gives them one by one to that aligning program that was chosen in the command line to do the alignments. This procedure is parallelized, which is advantageous in case of machines having more than one CPUs. When all the small alignments for the full components are ready, SpanAlign collects them into one file with the extension *fastas*. This file is given to Harvester which merges these small alignments into one final alignment. If the aligning program chosen is StatAlign, then SpanAlign will calculate the appropriate MCMC parameters for all the full components.

When using SpanAlign one has to have ClustalW or ClustalW2 on his machine and bonphy.py in the same directory where SpanAlign is and the same goes for the aligning program chosen (Mafft/ClustalW2/StatAlign). When we want to make SpanAlign run we have to give it the following parameters in this order:

- Input fasta file
- Bonphy method for contracting edges
- A  $p$  value for bonphy, which has to be 0 for all methods except method 5
- A  $k$  value, which defines the maximum size of a full component

- A number which represents the aligning program that we want to use to align the full components (0 = ClustalW2, 1 = StatAlign, 2 = Mafft)
- Method for merging the alignments with Harvester ( $-p=1$  in case of SumGap and  $-p=6$  in case of low likelihood based merging)

## 5 Testing

### 5.1 BALiBASE Scoring

When testing the accuracy of a multiple sequence alignment program one often uses the BALiBASE benchmark. BALiBASE is a database consisting of many multiple sequence alignments done manually (these will be referred to as reference alignments). It also provides a scoring program called *bali\_score* which can compare two alignments by assigning two scores. One of them is usually the reference alignment. The two scoring methods are the *SP* (Sum of Pairs) score and the *TC* (Total Column) score. They are defined in the following way.

**SP score** Assume that our test alignment consists of  $N^t$  sequences all of length  $M^t$ . Furthermore let the corresponding reference alignment consist of  $N^r$  sequences of length  $M^r$ . Then the two alignments corresponds to a  $N^t \times M^t$  matrix and  $N^r \times M^r$  matrix respectively. Let  $A_{i1}, \dots, A_{iN^t}$  be the residues in the  $i$ 'th column of the test alignment. Then for each pair of residues  $A_{ij}$  and  $A_{ik}$  in the  $i$ 'th column we define  $p_{ijk}$  as follows

$$p_{ijk} = \begin{cases} 1 & \text{if } A_{ij} \text{ and } A_{ik} \text{ are aligned in the reference alignment} \\ 0 & \text{otherwise} \end{cases}$$

The score  $S_i$  for the  $i$ 'th column is now defined as

$$S_i^t = \sum_{j=1}^{N^t} \sum_{\substack{k=1 \\ k \neq j}}^{N^t} p_{ijk}.$$

Letting  $S_i^r$  be the similar score for the reference alignment, the *SP* score is

$$SP = \frac{\sum_{i=1}^{M^t} S_i^t}{\sum_{i=1}^{M^r} S_i^r},$$

and hence  $0 \leq SP \leq 1$  and it is the percentage of well aligned residue pairs in the test alignment when compared to the reference alignment.

**TC score** The *TC* scores compares columns instead of residue pairs. Let  $C_i = 1$  if all the residues in the  $i$ 'th column of the test alignment are aligned in the reference alignment and  $C_i = 0$  otherwise. Then the *TC* score is defined as

$$TC = \sum_{i=1}^{M^t} \frac{C_i}{M^t}$$

We see that  $0 \leq TC \leq 1$  and it is the percentage of well aligned columns in the test alignment when compared to the reference alignment.

As mentioned in [1] these are not exactly the scores used in the *bali\_score* program. The program only takes columns in the reference alignment into account if they have fewer gaps than the gap threshold *gt*, which is defined as

$$gt = \left\lfloor \frac{20N}{100} \right\rfloor,$$

where  $\lfloor \cdot \rfloor$  is the floor function. So for the *TC* score *bali\_score* only counts the columns with fewer gaps than *gt* and for *SP* only residue pairs in the columns with fewer gaps than *gt* are taken into account.

## 5.2 Bonphy Methods

We will be using the *bali\_score* program to find the optimal bonphy method(s). The eighth methods listed beneath (enumerated according to the input option of bonphy) are the bonphy methods taken into consideration.

1. Minimise total length of remaining edges, when they are seen as being elongated to the terminal node of incident contracted edges.
2. Minimise total length of contracted edges.
3. Minimise total amount internal nodes are moved, when they are seen as being moved to the leaf an incident contracted edge can trace a path of contracted edges to.
4. Minimise product of relative length increases for remaining edges, when they are seen as being elongated to the terminal node of incident contracted edges.
5. Minimise sum of change in length to power 2 of remaining edges, when they are seen as being elongated to the terminal node of incident contracted edges.
8. Minimise maximum increase of lengths of remaining edges, when they are seen as being elongated to the terminal node of incident contracted edges; ties are broken by considering second largest increase and so on.
11. Minimise sum of pairwise distances in restricted tree, when edges are seen as being elongated to the terminal node of incident contracted edges.
12. Minimise sum of pairwise distances in the restricted tree, when uncontracted edges are seen as being elongated to the terminal node of incident contracted edges.

The course of optimising the bonphy method is the following: Starting with a set of  $n$  sequences a guide tree is obtained by Clustal using the NJ-algorithm, edge contractions and decomposition into a  $k$ -Spannoid is done using the eighth different bonphy methods and for  $k = 3, \dots, n$ , the components are aligned using Clustal and finally the components are joined together using the SumGapPairMerger of Harvester. These  $8 \times (n+1-3)$  final alignments were all scored against the reference alignment using the *bali\_score* program. This was done using two families, the first being *big\_fam* which has 74 sets of sequences where each set consists of at least 11 sequences. The second is called *midsize\_fam* which has 27 sets of sequences with less than 11 sequences in every set. Since the number of sequences vary for every set the ratio  $k/n$  is computed and a plot of the score against  $k/n$  is done in Figure 20 and Figure 21.

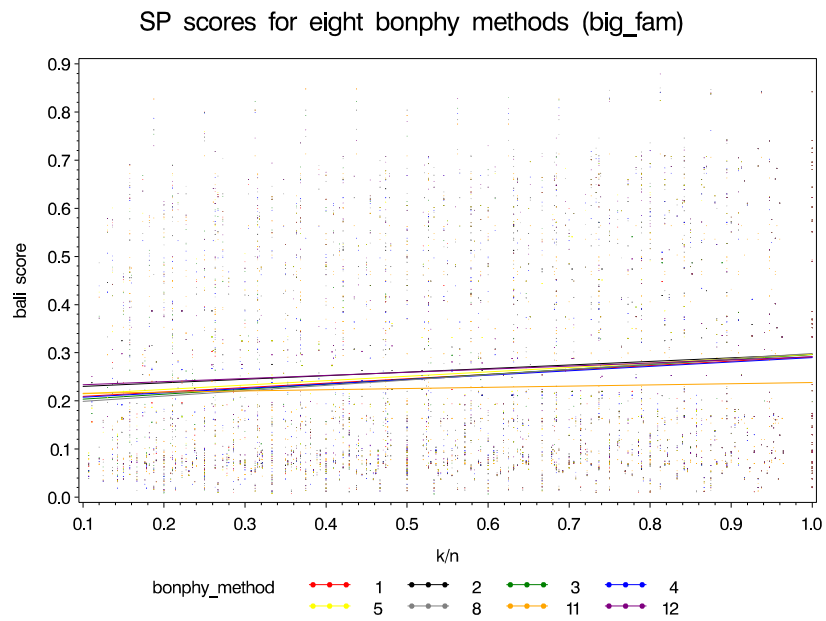


Figure 20: Showing the *SP* score against BALiBASE for the *big\_fam* family for various bonphy methods with regression lines.

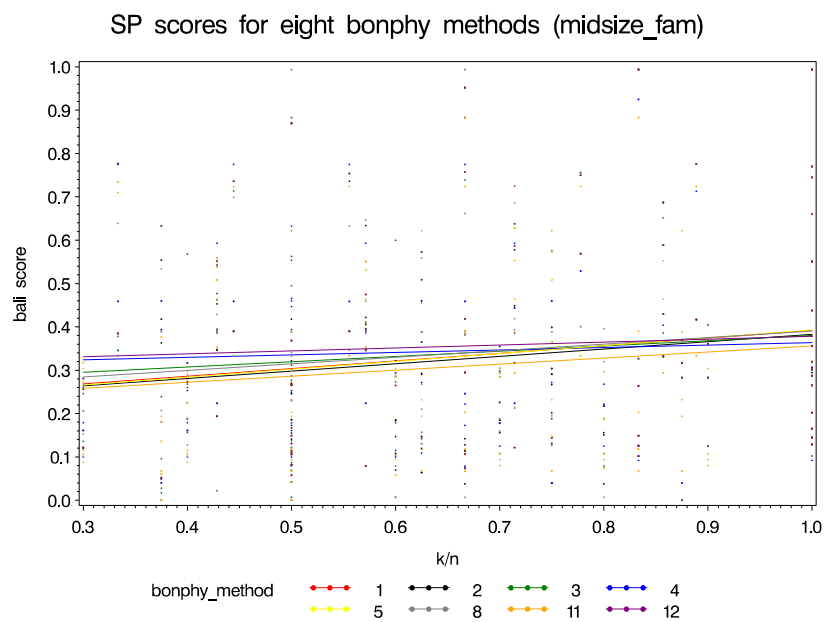


Figure 21: Showing the *SP* score again BALiBASE for the *midsize\_fam* family for various bonphy methods with regression lines.

In case of the *big\_fam* family we see that the best methods are 2 (black) and 12 (purple). For  $k/n$  values between 0.1 and 0.8 they are doing far better than any other

method and when approaching 1 all lines except for method 11 (orange) meet at approximately the same value. Moving on to the *midsize\_fam* family again we see that method 12 (purple) is on top with method 4 (blue) just below it. Method 2 (black) is doing very poorly for this family, and should not be used for sets of sequences with less than 11 sequences. Since method 12 is being superior both in the case of less than 11 sequences and more than 11 sequences this will be a method to consider the most optimal.

Another way to choose the optimal method is to compare the final alignments with the alignment got from using Clustal on the whole set of sequences. We will again use the *bali\_score* program to score our alignments with the one from Clustal. The result for the *big\_fam* family is shown in Figure 22 and the result for the *midsize\_fam* family in Figure 23.

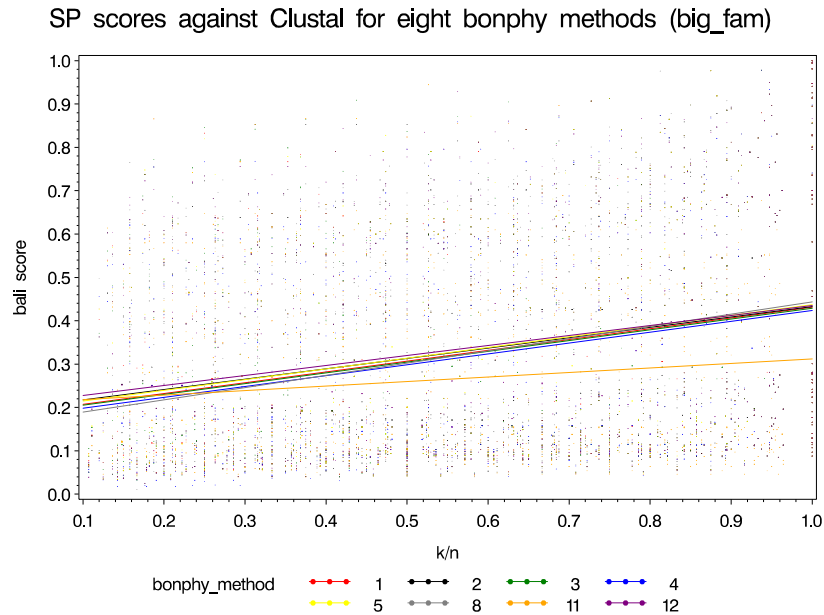


Figure 22: Showing the *SP* score against Clustal for the *big\_fam* family for various bonphy methods with regression lines.

From these two figures we get two conclusions. If  $k/n = 1$  then  $k = n$  and this means that our component sizes are allowed to be as big as the number of sequences, hence no matter which bonphy method is used, we should get one component of size  $n$  containing all sequences. This means that the alignment we get using SpanAlign should be exactly the same as the alignment got using Clustal. But this is just not the case, since the score is not 1. So for  $k = n$  bonphy clearly has some kind of error, since it does not return the right phylogeny. But we are still able to compare the different methods and see which of the methods is closest in average to the original Clustal method. Again we see that method 12 is superior in both cases and we must conclude that this is the best method out of all eight.

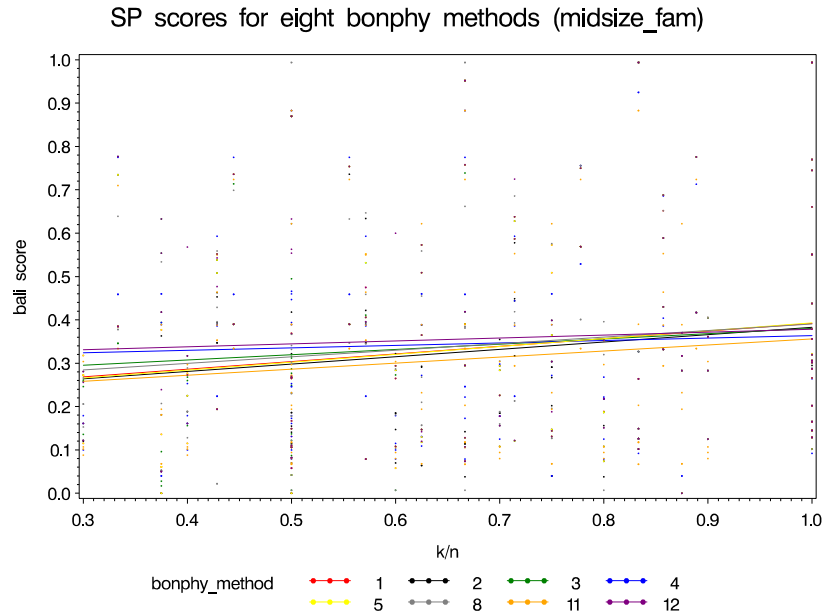


Figure 23: Showing the  $SP$  score against Clustal for the midsize\_fam family for various bonphy methods with regression lines.

## 5.3 Optimisation Of StatAlign

### 5.3.1 Data Collection

Using the likelihoodMerger of SpanAlign results in many runs with StatAlign. First we use StatAlign to align the sequences in the full components, but depending on the  $k$  value this will often only be a couple of sequences with normal protein length. After Harvester collects and joins the component alignments into one alignment it will call StatAlign to realign some regions with low likelihood. In this case what needs to be aligned are shorter regions but it involves all of the sequences. Being a statistic alignment method StatAlign is quite slow. If we want to have an efficient method for the spannoid alignments using StatAlign, we have to know the appropriate burn in and cycle number values in case of a given size of alignment.

We decided to optimise the running parameters on alignments of 3, 6, 9, 12 and 15 sequences with lengths 3, 10, 29, 66, 127, 218, 345. In case of the longer alignments we did not run StatAlign on more than 6 sequences, because it would have taken too much time. We got the smaller alignments by cutting out the desired size of alignment from the BALiBASE alignments. When using StatAlign on these sequences we did not use any burn in value and used a very high cycle number in order to make sure that the loglikelihood data would have converged. Table 1 contains the used running parameters for StatAlign.

### 5.3.2 Loglikelihood Method

At first we tried to use the loglikelihood values to get the appropriate cycle numbers for burn in and for the amount of cycles after burn in for the different kinds of alignments. For every alignment we searched for the highest loglikelihood value (maximum) among

ASL	Number of sequences				
	3	6	9	12	15
3	0,10,10	0,15,10	0,20,10	0,25,10	0,30,10
10	0,20,20	0,30,20	0,40,20	0,50,20	0,60,20
29	0,40,40	0,60,40	0,80,40	0,100,40	0,120,40
66	0,100,50	0,200,50	0,500,50	-	-
127	0,500,50	0,1000,200	-	-	-
218	0,1000,500	0,2000,1000	-	-	-
345	0,2000,1000	0,4000,2000	-	-	-

Table 1: The running parameters for StatAlign for different number of sequences and average sequence length (ASL). The first parameter is the burn in, the second is the cycle number (shown in thousands) and the last is the cycle rate.

all the samples. We calculated the average and the standard deviation of the values of the samples after the maximum (we supposed that here the loglikelihood values are converged already). After this we searched for the index of the first sample that had a loglikelihood value above the calculated average minus the deviation. Finally we multiplied this index with the sampling rate used in case of the given alignment. For every different alignment with the same size we got a cycle number value for the burn in by this method, and we calculated the burn in cycles needed for the given size of alignment by calculating the average of these data and adding up their deviation two times. With this method we couldn't get reliable data, the deviation of the burn in values got for the same size of alignment was extremely large and sometimes we got huge numbers for very small alignment sizes and smaller values for bigger sizes. The results of this method were ignored.

### 5.3.3 Alignment Based Method

We then decided to use the alignments themselves to get the burn in and cycle number. We used the final MPD (Maximum Posterior Decoding) of the given run and all the MPD alignments of the samples (we had these in the .log files produced by StatAlign). The *bali\_score* program then compared the MPD alignments of the samples to the final MPD alignment. We agreed on using an *SP* score of 0.80 as the lower bound for the burn in, meaning that once a given sample MPD has reached an *SP* score of 0.80 we should stop the burn in. When 0.80 was reached we should begin sampling and we decided that we would stop sampling when an *SP* score of 0.95 was reached and hence we got the cycle number. The 0.80 *SP* was obtained by examining several loglikelihood files by eye and find the sample which seems to be at the end of the burn in on the loglikelihood graph. This samples MPD alignment had an average *SP* score of 0.80 compared to the final MPD alignment of the run.

After we figured out the two borderline values, we wrote a program that goes through the .log file of a given run and scores the MPD alignment of the samples against the final MPD with the *bali\_score* program. We found that by plotting the *SP* score against the sample number did not give a strictly increasing graph. When trying to find the burn in value, we searched for the index of the first sample whose alignment

scored at least 0.80 against the final MPD. In case of the total cycle numbers we went through the .log file from the end and searched for the index of that sample whose alignment scored less than 0.95 for the first time. At first we wanted this second value to be 1.00, but in this case we often got very huge cycle numbers because in most cases there are several almost equally scoring and equally likely alignments competing for being the MPD, and hence the MPD is changing even after a very big amount of cycles.

Once we retrieved these indexes for all the test alignments, we multiplied these indexes with the sampling rate used for the given size of alignment and calculated the average for the burn in value and the cycle number for every size. Important to know is that these values are still overestimating both the burn in and the cycle number after burn in, because we did not use any burn in when testing, and because of this a lot of bad samples are taken into consideration which cause the correct final MPD alignment to appear later. We do not know anything about the sequence identity of our data, so we can not estimate the level of sequence identity for which the calculated cycle numbers are enough in case of a given size of alignment. The number of cycles needed to reach the upper bound of 0.95 is shown in Table 2. Here we discarded the results from the small alignments, since the score of 0.80 and 0.95 was reached in the same cycle which would mean that the total cycle number should be 0.

ASL	Number of sequences				
	3	6	9	12	15
3	-	-	-	-	-
10	-	10800	13200	19200	31500
29	13200	25600	30100	38000	45000
66	41400	82500	194300	-	-
127	107800	436800	-	-	-
218	249800	448600	-	-	-
345	440000	1905000	-	-	-

Table 2: The total cycle number needed to reach an *SP* score of 0.95 of the MPD alignments (rounded to hundred). Here ASL means Average Sequence Length.

From the data in Table 2 we found a connection between the cycle number after burn in, the number of sequences in a given alignment and the average length of the sequences. We got the following formula

$$X = 1.1^N \cdot L^{1.3} \cdot 200, \quad (2)$$

where  $X$  is the cycle number after burn in,  $N$  the number of sequences and  $L$  the average sequence length. The number of burn in cycles should be half of the cycle number after burn in, and the sampling rate should be given such that we gain at least 100 but not more than 200 samples. We agreed on having a sampling rate of  $X/200$ .

Our results for the burn in plus the samples after burnin using the formula in (2) (though multiplied by 300 instead 200, because  $X$  is only the number of cycles *after* the burn in) is seen in Table 3.

ASL	Number of sequences				
	3	6	9	12	15
3	1666	2217	2951	3927	5227
10	7967	10604	14114	18786	25004
29	31799	42325	56334	74981	99800
66	92620	123278	164083	218394	290682
127	216892	288684	384238	511421	689701
218	437817	582735	775620	1032350	1374058
345	795178	1058381	1408706	1874987	2495608

Table 3: The total cycle number needed to reach an  $SP$  score of 0.95 of the MPD alignments (rounded to hundred). Here ASL means Average Sequence Length.

## 5.4 StatAlign Performance

The performance of StatAlign is of course of highly importance to the performance of SpanAlign when using StatAlign on the full components. Since no previous test of the performance of StatAlign was done, we decided to compare the StatAlign alignments to that of Clustal and Mafft. The reasons for choosing these two alignment programs is that Clustal is the most commonly used and Mafft, according to the analysis in [7], is one of most accurate programs available. We ran StatAlign, Clustal and Mafft on 80 families from BALiBASE, where 75 of them were from the *Reference 1*. Reference 1 contains alignments of (less than six) equi-distant sequences. That is the percent identity between two sequences is within a specified range. These ranges are grouped by  $< 25\%$ ,  $20\% - 40\%$  and  $> 35\%$ .  $SP$  scores for StatAlign, Clustal and Mafft for these families are shown in Figure 24.

We clearly see that StatAlign is doing much better than Clustal scoring on average 0.2 points higher in the  $SP$  score according to the smoothing spline. The plot also shows that StatAlign is doing better than Mafft on the first 45 sequences and for the rest Mafft scores higher. Looking at the difference in  $SP$  scores, namely  $SP_{\text{StatAlign}} - SP_{\text{Mafft}}$  and  $SP_{\text{StatAlign}} - SP_{\text{Clustal}}$ , we get the plot seen in Figure 25.

Again we see that StatAlign performs much better than Clustal, but the fitted spline for difference between StatAlign and Mafft is slightly above zero for the first 30 – 35 families and after that slightly below zero. This is probably because the first 26 sequences are of short length, the sequences 27–47 are medium length and the last 47–80 are quite long. So it seems like StatAlign is doing best for small length sequences and for longer sequences Mafft is the best. A possible explanation for why this is the case could be the parameter choices for the MCMC (Markov Chain Monte Carlo) method used in StatAlign.

As seen in Table 4 the MCMC parameters for the StatAlign runs were based on heuristics and not the analysis given in Section 5.3. Therefor it seems likely that if the parameters were chosen according to methods described in Section 5.3 we could have gotten a better result. For sake of completeness the  $TC$  scores for StatAlign, Mafft and Clustal are shown in Figure 26 and their respective difference in Figure 27. Both plots show exactly the same trend as discussed before, although StatAlign seems to be doing a little better for the long sequences.

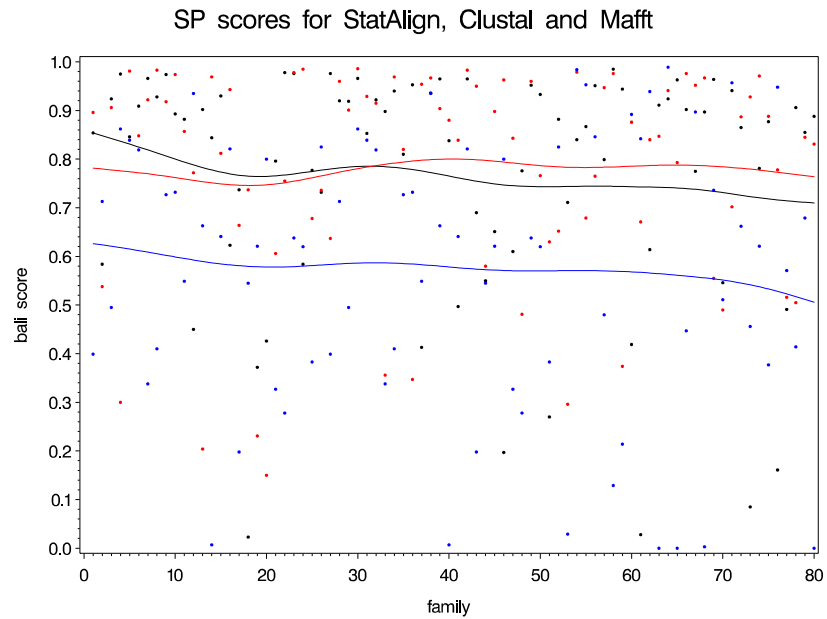


Figure 24: Showing the  $SP$  scores for StatAlign (black), Mafft (red) and Clustal (blue) for 80 protein families with number of sequences less than or equal to 6. Smoothing spline fitted using  $i=sm60$  in SAS.

Difference in  $SP$  scores for StatAlign and Clustal, StatAlign and Mafft

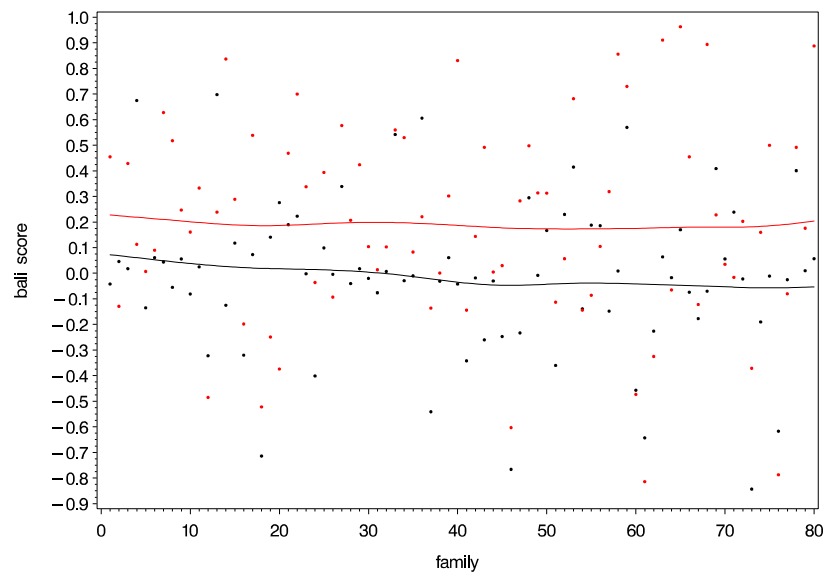


Figure 25: Showing the difference in  $SP$  scores for StatAlign and Mafft (black) and for StatAlign and Clustal (blue) for 80 protein families with number of sequences less than or equal to 6. Smoothing spline fitted using  $i=sm60$  in SAS.

	$n \leq 5$	$n > 5$
$ml \leq 150$	(50.000,1.000.000,500)	(200.000,2.000.000,500)
$150 < ml \leq 300$	(100.000,1.500.000,500)	(200.000,2.000.000,500)
$300 < ml$	(100.000,2.000.000,500)	(200.000,2.000.000,500)

Table 4: MCMC parameters for the StatAlign runs. Here  $n$  is the number of sequences and  $ml$  is the maximum length of the sequences in the given family. The first parameter is the burn-in, the second is the cycle number and the third is the sample rate.

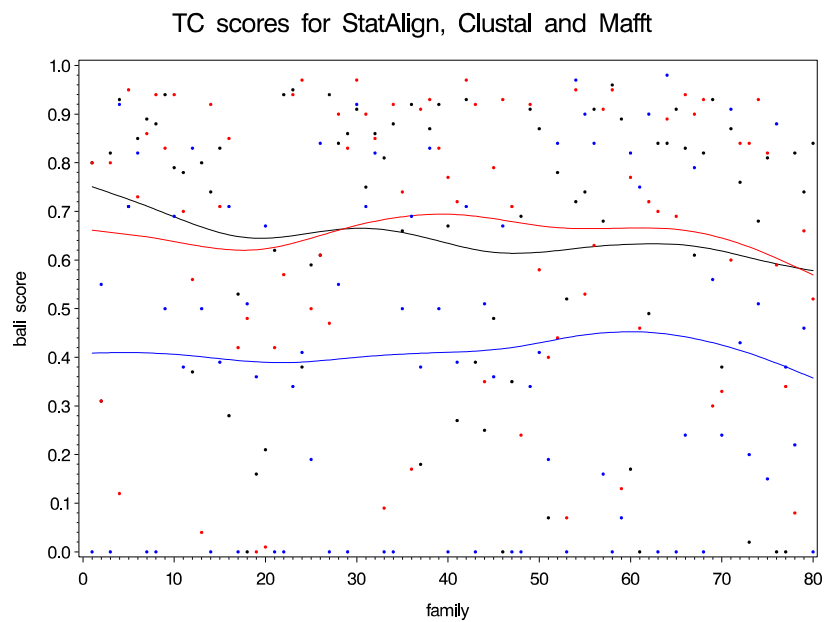


Figure 26: Showing the  $TC$  scores for StatAlign (black), Mafft (red) and Clustal (blue) for 80 protein families with number of sequences less than or equal to 6. Smoothing spline fitted using `i=spl60` in SAS.

All in all we are quite confident that by choosing optimal parameters for StatAlign you will get the best result among any multiple sequence alignment programs available.

Difference in TC scores for StatAlign and Clustal, StatAlign and Mafft

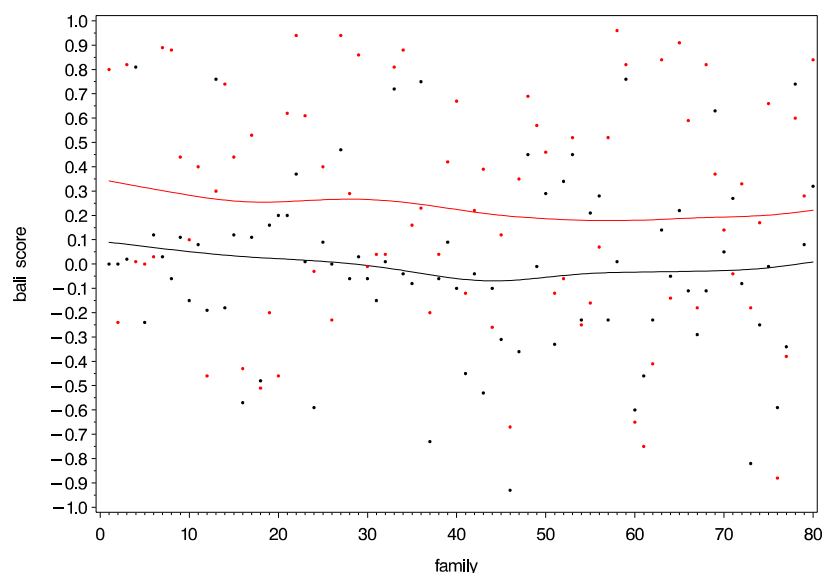


Figure 27: Showing the difference in  $SP$  scores for StatAlign and Mafft (black) and for StatAlign and Clustal (blue) for 80 protein families with number of sequences less than or equal to 6. Smoothing spline fitted using `i=sm60` in SAS.

## 5.5 SpanAlign

We will now test our alignments using various methods and compare them to that of Clustal and Mafft. We will be running the methods on 29 families from BALiBASE and scored the alignment using the *bali\_score* program, and the methods used are

1. Clustal on the full components and the SumGapPairMerger with method 12 of bonphy and  $k = 4, 5$ .
2. StatAlign on the full components and the SumGapPairMerger with method 12 of bonphy and  $k = 4, 5$ .
3. StatAlign on the full components and the SumGapPairMerger with method 12 of bonphy and  $k = 4, 5$ . Furthermore using the realignment of low likelihood regions.

In Figure 28 are shown the  $SP$  scores for the methods just mentioned along with the scores using only Mafft and Clustal on the families. In Figure 29 are shown only the fitted smoothing splines for our methods.

We see that the results does not vary much between our suggested methods. We do not seem to get a much better result using StatAlign on the full components rather than Clustal which is a surprise. Although we are doing much worse than both Mafft and Clustal we see that in a few cases we are able to score better than both. One possible explanation for this is that the full components are not big enough. Even though setting  $k = 10$  we are not ensured of getting any components of size 10. We could just as easily get components of size less than 5. In Figure 30 we see the scores using Clustal on the components with  $k = 4, 5, 7, 10$ . Even though  $k = 10$  (green) seems to get a

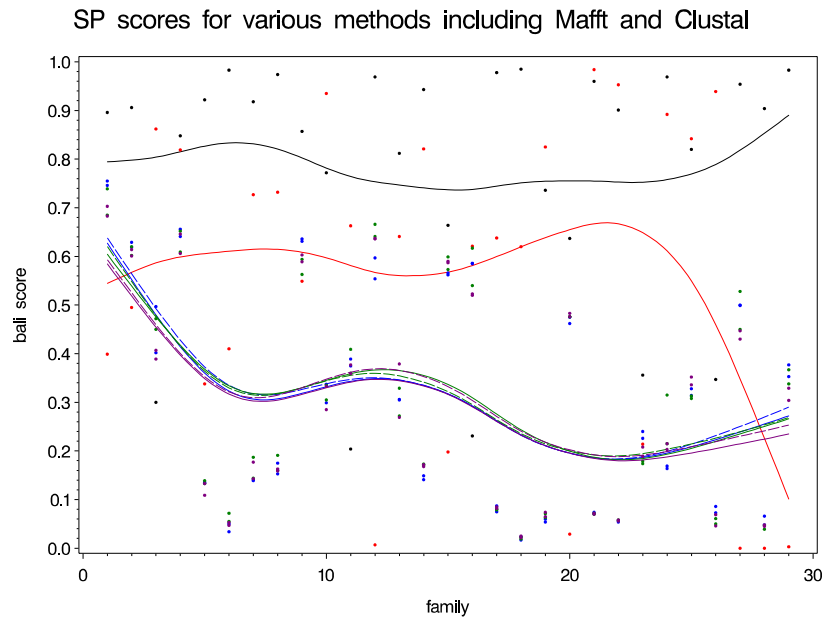


Figure 28: Showing the  $SP$  scores for Mafft (black), Clustal (red), Clustal on components (blue,  $k = 5$  dashed), StatAlign on components with likelihood (green,  $k = 5$  dashed), StatAlign on components without likelihood (purple,  $k = 5$  dashed).

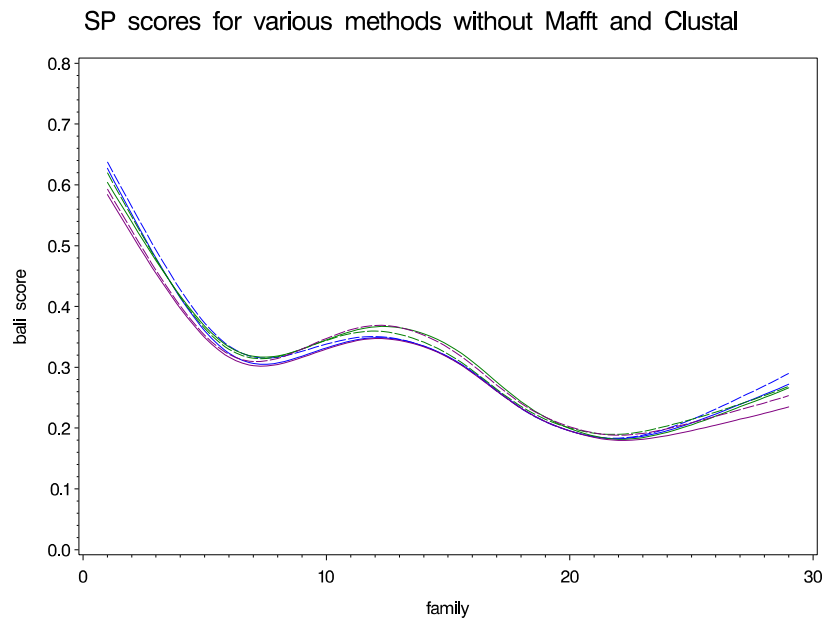


Figure 29: Showing only the fitted smoothing spline for Clustal on components (blue,  $k = 5$  dashed), StatAlign on components with likelihood (green,  $k = 5$  dashed), StatAlign on components without likelihood (purple,  $k = 5$  dashed).

better results overall, the difference is not very significant. We would have expected, as  $k$  increases, the scores would tend to the scores of running only Clustal on the whole family. This does not seem to be the case, and maybe another way of obtaining a spannoid should be considered.

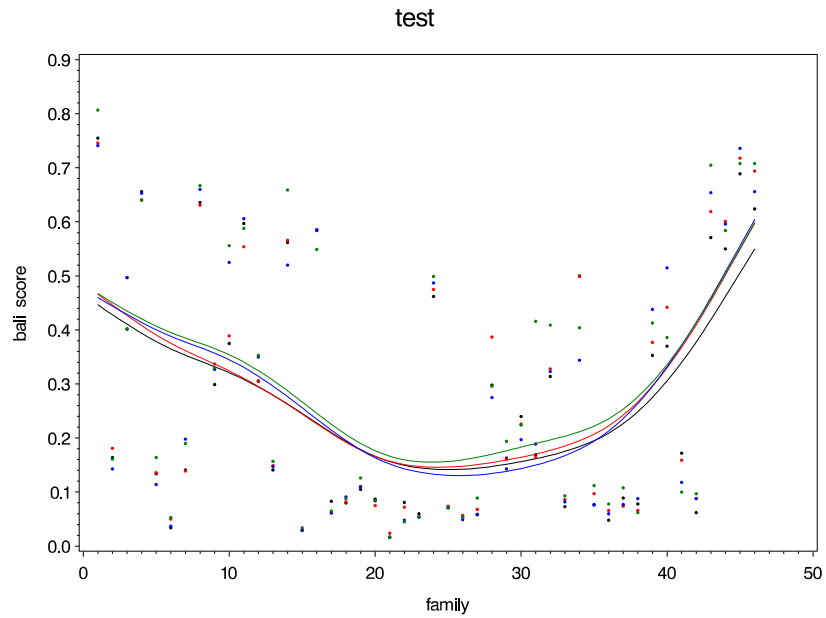


Figure 30: Showing the  $SP$  scores for 46 families using Clustal on the components for  $k = 4$  (black),  $k = 5$  (red),  $k = 7$  (blue) and  $k = 10$  (green).

## 6 Conclusions

Looking back we are pleased with the outcome of the project, although we had hoped for a better performance by our SpanAlign program. As seen in Section 5.5 we are far behind both Mafft and Clustal scoring wise and as Figure 30 shows there is not clear pattern of the scores when  $k$  increases. If the idea of doing edge contractions to obtain a  $k$ -restricted Steiner tree was guaranteed to provide us with as many components of size  $k$  as possible, then maybe the performance would have been much different. The problem is that if we, for instance, put  $k = 10$  we do not have any clue of the sizes of the components. We could have gotten 15 components of size 2 which would mean that we are doing 15 pairwise alignments and of course the final result will be bad. If we would instead have gotten 2 components of size 10 and 5 respectively we would have used the alignment programs to their fullest potential. Because the engine behind the idea is still the alignment programs, when they are doing their job on the full components and we want to utilize this to its fullest. This is discussed in Section 7.

A minor disappointment in our results was that, as seen in Figure 29, there was not much of a difference made between any of our different methods of merging the spannoid. Even when we realigned low likelihood regions of the alignments as we went along, it did not make much of a difference to the final alignment (although it did make a major difference to the time taken to run). We suspect that the spannoid we use greatly affects the final outcome, and since the edge contraction method does seem to output random component sizes, we should attempt different method of spannoid production for further work, as is described in Section 7.

We also wanted to align a thousand sequences before the end of this project, but we had some problems with bonphy. An error occured randomly when trying to align more than 50-80 sequences, and thus stopping us from reaching that goal. If a correction of that error is made to bonphy it should be able to align a thousand sequences in a fairly short amount of time.

Implementing Harvester in an object oriented manner has left it as a flexible prototype making modifications possible, so we think that any further work on Harvester would be very straightforward, and although no formal documentation was left, this report explains some of the methods used and there are comments throughout the code.

The analysis of the MCMC parameters was a success. A good estimate of the required parameters for an alignment from a convergent maximum posterior decoding was made in the form of a function exponential in the number of sequences  $N$  and almost linear in the length of the sequences  $L$ . Hopefully these results will be used in any future research using the StatAlign software.

The spannoid idea is a good one, but to utilise its potential fully further work is required. We would like to thank *Ádám Novák*, *Rune Lyngsø*, and *Jotun Hein*, for helping us to complete this project.

## 7 Future Work

### 7.1 Bonphy Optimisation

As mentioned a problem with bonphy is that for a given  $k$ -value it does not guarantee the sizes of the full components. Ideally we would want as many components with the size  $k$  (or as close to  $k$ ) as possible. One way of doing that would be to let bonphy compute the full components and then merging the full components along the phylogenetic guide tree until the desired size is reached. So given a set of full components one should start from the leaves and move up according to the guide tree. If we had put  $k = 6$  and gotten the spannoid shown in Figure 13 we could merge the components to increase their size. Starting from the leaves and merging upwards we would get the components {1HSTA, H11 ARATH, H1 DICDI, H1L MYTTR, **1IDY**}, {ARGR STRCL, G3273713, AHRC BACSU, ARGR BACST, 1AOY, **1IDY**}, {MYB3 MAIZE, MYB3 HORVU, MYB2 PHYPA, **GL1 ARATH**} and {**GL1 ARATH, 1IDY**}. So instead of nine components of size 2 and 3 we get four components of size 5, 6, 4 and 2 respectively. By optimising the component sizes to get as close to  $k$  as possible, first of all the user has more power of the method and we utilize the alignment programs on the components to their fullest.

### 7.2 Alternatives To Bonphy

#### 7.2.1 Inferring Internal Nodes

Bonphy works by minimising the change to a guide phylogeny to get a spannoid. A method suggested that wouldn't modify the guide tree would be to infer the internal sequence nodes rather than contract them. An advantage of statistical alignment is that the most probable ancestral sequences can be inferred from the alignment process. Using this, we would align  $k-1$  neighbouring leaves of the phylogeny and get an alignment of these  $k$  sequences plus an inferred sequence for the root of that subphylogeny. This could be done until there are no leaves that are not in an aligned component. Then the procedure could be repeated for the inferred internal nodes until a  $k$  restricted steiner tree i.e. a spannoid is created. Then the current Harvester procedure can be done and the inferred sequences can be removed from the final alignment.

A problem with this may be that we have sequences that are inferred from an alignment of inferred sequences which are in turn inferred from an alignment of inferred sequences and so on. There is some error involved in the sequences inferred, and this can only be propagated by repeatedly inferring more sequences from them. However, we do not disturb the phylogeny at all, which is a major advantage.

#### 7.2.2 Making A Spannoid By Clustering From Pairwise Distances

One could totally avoid the idea of a guide phylogeny altogether and create the spannoid in the following way. Plot the sequences on the plane using the pairwise distances as in UPGMA clustering. Then using some optimisation technique, separate the sequences into clusters with a common sequence between neighbouring clusters. Then we have a spannoid. If one requires a phylogeny from this, you could create subphylogenies from the clusters and then combine them into a full phylogeny using the common sequences.

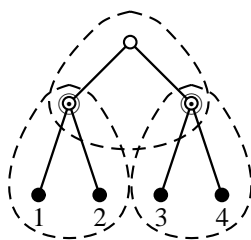


Figure 31: Sequences 1 and 2 are aligned and the root sequence is inferred ( $\odot$ ). The same is done for sequences 3 and 4 and then the two inferred sequences are aligned as the third component, giving a 3-spannoid with components of sizes 3, 2 and 3.

### 7.2.3 Taking Alignment Samples For Random Spannoids

Another idea is to generate random spannoids with the given  $k$  value, it would mean to pick  $k$  amount of sequences randomly, choose one of them randomly to be the common sequence with the next full component, and then choose  $k - 1$  sequences randomly to fill the the second full component, and choose one of the sequences already used in the first two full components to be the common sequence with the third full component.

If we compute the final alignment for these full components with Harvester and do the whole procedure many times, than we could gain a lot of alignments for the same set of sequences and then find a kind of consensus alignment at the end with the same method that StatAlign finds the actual MPD alignment from the samples. We did the first part of this method, we have a program, that generates random spannoids for a list of sequences with a given  $k$  value, aligns them (we used Mafft when aligning the full components and Harvester to merge the small alignments with the SumGap merging method), and gives a list of alignments as an output. We do not have the part that could combine these alignments into a consensus alignment, but in our opinion it would be great to see if this idea would work.

What was surprising is that we scored these alignments against the BALiBASE alignment of the same sequences and sometimes we got quite good results with a randomly generated Spannoid. The average quality ( $SP$  score) of these alignments was lower than the quality of the alignment gained with the Spannoid produced by bonphy. It would be great to see what quality we can reach with a consensus alignment from these samples.

In Figure 32 we see a plot of the  $SP$  score for three families using a completely random spannoid (black dots) and the eight bonphy methods (red dots). A total of 500 random spannoids were created and scored using *bali\_score* with Mafft running on the full components and the SumGap merging method. We clearly see a correlation between score and alignment length, so if we were to sample randomly among the spannoids we should try to minimise the total length since this will most likely give a better consensus alignment. We suspect that the average score of alignments would not increase much, but the deviation of scores for the alignments would be less because bad alignments caused by a randomly bad spannoid would be eliminated.

## 7.3 Edge Lengths

An improvement to Harvester would be to incorporate edge lengths in the spannoid data structure. The edge length between two components could be derived from the

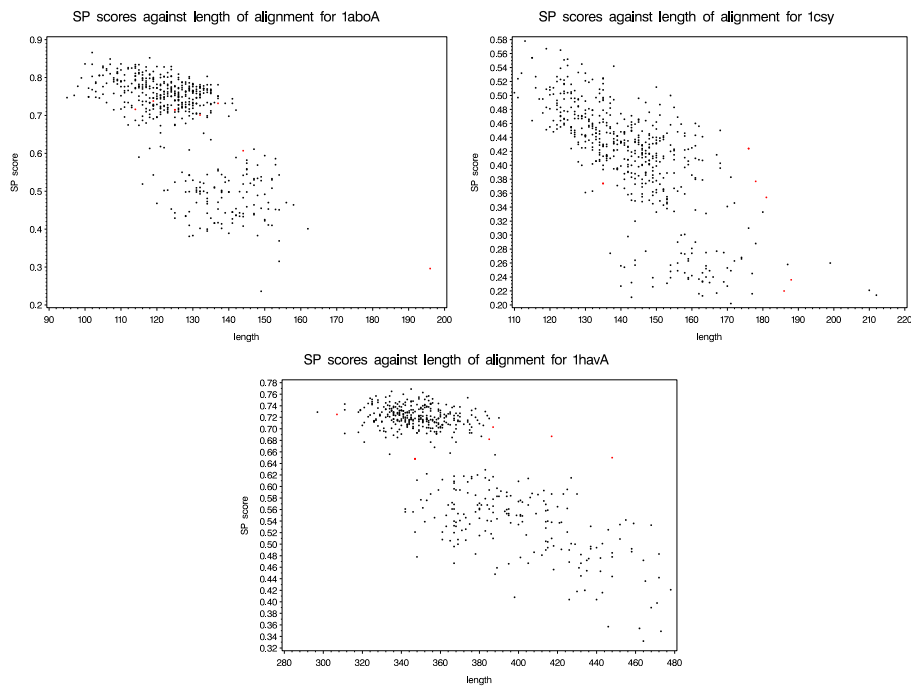


Figure 32: A plot of  $SP$  score against the length of the alignment for the three families `laboA`, `1csy` and `1havA` using the random spannoid approach. The black dots represent the score using a random spannoid and the red corresponds to the score using the eight bonphy methods.

distances in the  $k$ -restricted steiner tree output from bonphy. This would give one advantage to the approach taken by bonphy and it would be simple to implement. The edge length could be the average or the sum of the distances from the common sequence to its neighbours as in Figure 33.

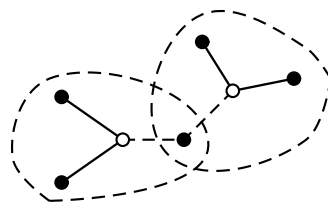


Figure 33: The edge length between these two components could be defined as the mean or sum of the dotted edge lengths in the steiner tree.

Harvester could use the edge lengths to make decisions for scoring the final alignment or for the order in which to align sequences, if further work gave evidence for a heuristic on that. So parameter estimation and annotation could use these edge lengths, as explained below.

## 7.4 Parameter Estimation

One thing that most certainly needs to be implemented in SpanAlign is the ability to output parameter estimates for the substitution, insertion and deletion rates. Since StatAlign also does samples of the parameters one could do the following. After running StatAlign on the components we get a .log file for each component which contains estimates of the insertion and deletion rates for every sample. Assume we have  $n$  components  $C_1, \dots, C_n$  of size  $c_1, \dots, c_n$  respectively and that we do  $s_i$  samples on component  $C_i$ . Then a fair estimate of the insertion rate  $\lambda_i$  and deletion rate  $\mu_i$  for the  $i$ 'th component would be the arithmetic mean, that is

$$\hat{\lambda}_i = \frac{1}{s_i} \sum_{j=1}^{s_i} \hat{\lambda}_{ij}, \quad \text{for } i = 1, \dots, n,$$

where  $\hat{\lambda}_{ij}$  is the parameter estimate for the insertion rate in the  $i$ 'th component for the  $j$ 'th sample. Similarly for the deletion rate we get

$$\hat{\mu}_i = \frac{1}{s_i} \sum_{j=1}^{s_i} \hat{\mu}_{ij}, \quad \text{for } i = 1, \dots, n,$$

where  $\hat{\mu}_{ij}$  is the parameter estimate for the deletion rate in the  $i$ 'th component for the  $j$ 'th sample. We then need to combine the parameter estimates on the components to an estimate on the whole phylogeny. The subphylogeny corresponding to the component  $C_i$  would have  $c_i - 1$  pairwise alignments (here we mean that we have a pairwise alignment if and only if there is an edge between the two sequences in the subphylogeny). Using this as a weight for the components we get parameter estimates by taking a weighted average over the components

$$\hat{\lambda} = \frac{\sum_{i=1}^n (c_i - 1) \hat{\lambda}_i}{\sum_{i=1}^n (c_i - 1)},$$

and

$$\hat{\mu} = \frac{\sum_{i=1}^n (c_i - 1) \hat{\mu}_i}{\sum_{i=1}^n (c_i - 1)}.$$

This approach would be fairly easy to implement in Harvester.

## 7.5 Annotation

It would of course be great to see if annotation could be applied to the spannoid framework. Since aligning lots of sequences can be very helpful in connection to annotation the spannoid technique would come in handy. This idea was to be worked on alot and is not something we had to time to cover.

One idea would be to use a statistical program called BigFoot to annotate the components and this will give a posterior distribution of the annotation on each components. So doing this with one component at first one can use the posterior distribution as a prior distribution on the next component and thus obtaining a posterior distribution on this component and so on. What it requires is BigFoot to take in a prior distribution and since it does not currently accept this a modification would be needed.

## A Appendix

This is the documentation for the two programs we wrote for this project. The first being SpanAlign and the second being Harvester.

### A.1 SpanAlign Source Code

```

1  #!/usr/bin/perl
2  use File::Copy;
3  use Thread qw(async);
4
5  use warnings;
6  use strict;
7
8  # Here we make sure, that all the parameters needed are given.
9
10 if ($#ARGV < 4 ) {
11     print "usage: SpanAlign.pl fastafile(all seq) bonphy_method bonphy_p_value bonphy_k_value alignmentmethod
12     \nalignmentmethods: \nClustal = 0\nStatAlign = 1 burn_in cycles sampling_rate harvester_opts";
13     exit;
14 }
15
16 my $fasta_file = $ARGV[0];
17 chomp $fasta_file;
18
19 my $bonphy_method = $ARGV[1];
20 chomp $bonphy_method;
21 my $bonphy_p_value = $ARGV[2];
22 chomp $bonphy_p_value;
23 my $sk_value = $ARGV[3];
24 chomp $sk_value;
25
26 # Here we define variables for calculate the mcmc cycle numbers, this is only needed
27 # in case o using StatAlign to align the fullcomponents.
28
29 my $faktor = 200;
30 my $sample_num = 200;
31 my $fA = 1.1;
32 my $fB = 1.3;
33
34 my $harvesteropts = "";
35 my $alignment_command = "";
36
37 # In case of the different aligning programs we have to use different command when aligning the fullcomponents.
38 # Here we chose which command should be used later.
39
40 if ($ARGV[4] == 0) {
41     $alignment_command = "ClustalW2 --INFILE=infile --OUTFILE=outfile --OUTPUT=FASTA --ALIGN";
42 } elsif ($ARGV[4] == 1) {
43     $alignment_command = "java -Xmx512m --jar statalign.jar infile -ot=Fasta --mcmc=";
44     $harvesteropts = $ARGV[5];
45     chomp $ARGV[5];
46     $harvesteropts =~s/_/ /g;
47 } elsif ($ARGV[4] == 2) {
48     $alignment_command = "mafft --auto infile > outfile";
49 }
50
51 # Here we eliminate vertical lines from the names of sequences,
52 # because they would cause serious problems later. This step is important because
53 # in uniprot database in the headline of every entry there are vertical lines.
54
55 chomp $alignment_command;
56
57 open (FST,"$fasta_file");
58 my @fst = <FST>;
59 close (FST);
60
61 open (FSTA,">$fasta_file");
62
63 for (my $i=0; $i<@fst; $i++) {
64     $fst[$i]=s/_/ /g;
65     print FSTA $fst[$i];
66 }
67
68 close (FSTA);
69
70 #Here we call Clustalw or ClustalW2 to create the guide tree with the neighbour joining method.
71
72 `clustalw --INFILE=$fasta_file --TREE`;
73
74 print "Guide tree created.\n";
75
76 my $tree_file = $fasta_file;
77 $tree_file =~s/_/\.fasta/\.ph/g;
78
79 my $bonoutfile = $tree_file;
80 $bonoutfile =~s/_/\.ph/g;
81
82 $bonoutfile = $bonoutfile."_s".$bonphy_method."_k".$sk_value.".bontree";
83
84 my $testfile = $bonoutfile;
85 $testfile =~s/_/\.bontree/\.test/g;

```

```

86
87 my $scorefile = $bonoutfile;
88 $scorefile =~ s/\./bontree/\./msf/g;
89
90 # Here we call bonphy to make the edge contractions in the guide tree, creating a Spannoid.
91 # The method chosen and the k value have to be given to the program before.
92
93 if ($bonphy_p_value == 0) {
94
95     'python ./bonphy.py -s $bonphy_method -k $k_value -t $tree_file > ./$bonoutfile';
96
97 }
98
99 else {
100
101     'python ./bonphy.py -s $bonphy_method $bonphy_p_value -k $k_value -t $tree_file > ./$bonoutfile';
102
103 }
104
105 print "Bonphy created spannoid.\n";
106
107 open (SPAN, "$bonoutfile");
108 my @spannoid = <SPAN>;
109 close (SPAN);
110
111 open (FASTA, "$fasta_file");
112 my @fastak = <FASTA>;
113 close (FASTA);
114
115 my @span;
116
117 $bonoutfile =~ s/\./bontree//g;
118
119 my $fastas_out_file = "./.$bonoutfile._fastas";
120
121 open (ALNFASTAS, ">$fastas_out_file");
122
123 my $fasta_dir = $bonoutfile."_fastas";
124
125 unlink_glob "$fasta_dir/*";
126 rmdir $fasta_dir;
127
128 mkdir $fasta_dir;
129
130
131 # In the following two for loops we go through the bonphy output file, the Spannoid,
132 # and create small fasta files containing the sequences of one fullcomponent.
133 # We create a separate fasta file for every full component.
134
135 for (my $h=0; $h<@spannoid; $h++) {
136     chomp $spannoid[$h];
137     $span[$h] = $spannoid[$h];
138 }
139
140 for (my $i=0; $i<@span; $i++) {
141     my @names = ();
142     chomp $span[$i];
143
144     $span[$i] =~ s//g;
145
146     $span[$i] =~ s/\^{//g;
147     $span[$i] =~ s/\)//g;
148
149     my @telj_sor = split(' ', $span[$i]);
150
151     my $num_brac = 0;
152
153     for (my $j=0; $j<@telj_sor; $j++) {
154         if (($telj_sor[$j] eq "(") || ($telj_sor[$j] eq "(")) { $num_brac++ }
155     }
156
157     if ($num_brac != 1) {
158
159         $span[$i] =~ s/\^{//g;
160         $span[$i] =~ s/\)//g;
161
162         my @egys = split(' ', $span[$i]);
163
164         for (my $j=0; $j<@egys; $j++) {
165             my @egynev = split(':', $egys[$j]);
166             push (@names, $egynev[0]);
167         }
168
169     }
170
171     else {
172
173         #print $span[$i], "\n\n";
174
175         my @egys = split('\ ', $span[$i]);
176
177         push (@names, $egys[1]);
178
179         #print $egys[0], "\n\n", $egys[1], "\n\n";
180
181         my @elso_nev = split(':', $egys[0]);
182         push (@names, $elso_nev[0]);
183
184

```

```

185     }
186
187
188
189     my $filenev = "temp".$i.".seq";
190
191     open (F,">$fasta_dir/$filenev");
192
193     for (my $j=0; $j<@names; $j++) {
194         my $name = $names[$j];
195         #print $name, "\n";
196
197         my $print = 0;
198
199         for (my $k=0; $k<@fastak; $k++) {
200             chomp $fastak[$k];
201
202             if ($fastak[$k]==m/^>/) {
203                 if ($fastak[$k]==m/$name/) {
204                     $print = 1;
205                 }
206                 else { $print = 0 }
207             }
208
209             if ($print == 1) {print F $fastak[$k], "\n"}
210         }
211     }
212
213     close (F);
214
215 }
216
217
218 # In the next for loop we go through on all the small fasta files , open them ,
219 # count the sequences and calculate the average sequence length in them .
220 # From these data we can calculate the mcmc parameters for every full component
221 # and the give them to StatAlign , or any of the methods chosen to align them .
222 # These alignments can be created paralelly on machines , that have more CPUs .
223
224 my @intostat = <$fasta_dir/*.seq>;
225 my @threads = ();
226 my $alignment_command_with_file = $alignment_command;
227
228 for (my $i=0; $i<@intostat; $i++) {
229     $alignment_command_with_file = $alignment_command;
230
231     chomp $intostat[$i];
232
233     my $input_file = $intostat[$i];
234
235     my $length_sum = 0;
236
237     #count the number of sequences in the file
238
239     open (TOCOUNT, $intostat[$i]);
240     my @to_count = <TOCOUNT>;
241
242     my $count = 0;
243
244     for (my $k = 0; $k < @to_count; $k++ ) {
245         chomp $to_count[$k];
246         if ($to_count[$k]==m/^>/) {
247             $count++;
248         }
249         if ($to_count[$k]==m/^A-Z/) {
250             $length_sum += length($to_count[$k]);
251         }
252     }
253
254
255     close (TOCOUNT);
256
257     my $avg_length = int($length_sum/$count);
258
259     my $scyclenum = int($faktor*(($fa**$count)*($avg_length**$fb)));
260     my $burnin = int($scyclenum/2);
261     my $samp_rate = int($scyclenum/$sample_num);
262
263     $alignment_command_with_file =~s/infile/$input_file/g;
264
265     my $output_file1 = $input_file;
266     my $output_file2 = $input_file;
267     $output_file1 =~s/$fasta_dir/\\/g;
268     $output_file2 =~s/$output_file1/\\/g;
269     my $output_file = $output_file2."alignments/".$output_file1;
270
271     mkdir $output_file2."alignments";
272
273     $alignment_command_with_file =~s/outfile/$output_file/g;
274     $alignment_command_with_file = $alignment_command_with_file.$burnin.".".$scyclenum.".".$samp_rate;
275
276     my $tmptthread = async {
277         '$alignment_command_with_file';
278         print $input_file, "\n";
279     };
280
281     push(@threads, \ $tmptthread);
282
283 }

```

```

284
285 #wait for components to be aligned
286
287 for (my $i = 0; $i < @threads; $i++) {
288     ${threads[$i]}->join;
289 }
290
291
292 if ($ARGV[4] == 1) {
293     `cp $fasta_dir/*.mpd $fasta_dir/alignments/`;
294 }
295
296 # In the next for loop we collect all the small alignments into one big file with the extension .fastas.
297 # This will be given to Harvester to merge these small alignments.
298
299 my @intoharvester = <$fasta_dir/alignments/*>;
300
301
302 for (my $i=0; $i<@intoharvester; $i++) {
303     chomp $intoharvester[$i];
304
305     my $akt_file = $intoharvester[$i];
306
307     open (ALN," $akt_file ");
308     my @akt_file = <ALN>;
309     close (ALN);
310
311     print ALNFASTAS " ", "\n";
312
313     print ALNFASTAS @akt_file;
314 }
315
316 # We give the .fastas file to Harvester and define the method of merging for it,
317 # which was chosen by the user and read from the command line.
318
319 `java -jar harvester2.jar $fastas_out_file $harvesteropts `;

```

## A.2 Harvester Source Code

Harvester is an object oriented Java program. Here is a class diagram for clarity:

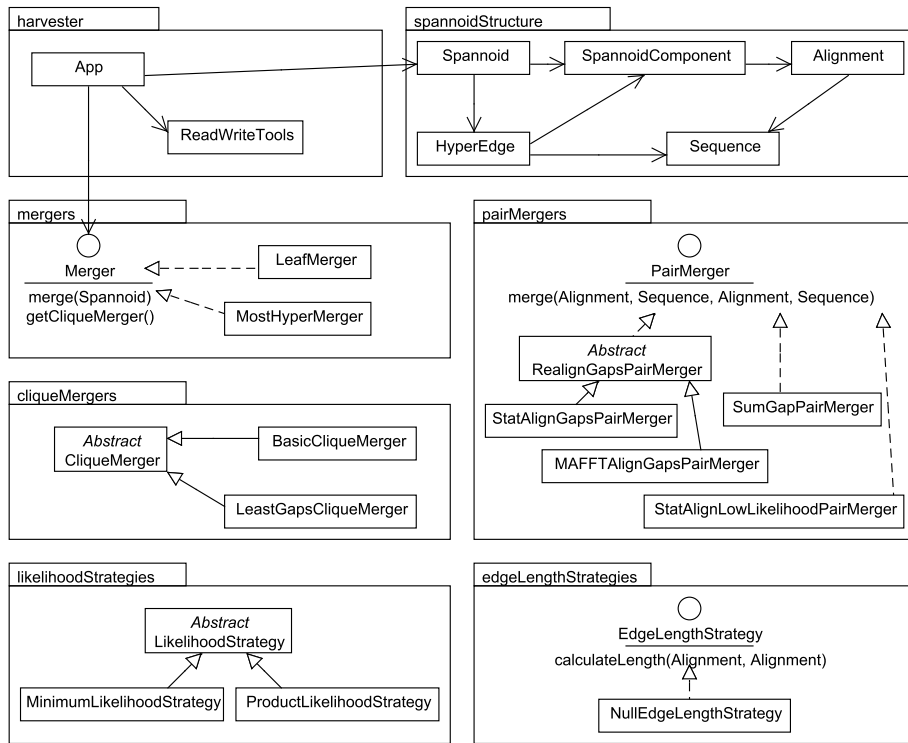


Figure 34: Class diagram for Harvester.

This is the help file:

```
Usage: java -jar harvester2.jar [-i=inputfile] [-o=outfile]
[-m=merger] [-c=clique merger] [-p=pair merger] [-e=edge length strategy]
[-l=likelihood strategy] [-t=likelihood threshold]
```

```
pair merger:
1 SumGapPairMergerBasic (default)
2 MaxGapPairMergerBasic (removed)
3 WindowingPairMerger (not finished)
4 MaxGapPairMergerScored (not finished)
5 StatAlignGapsPairMerger
6 StatAlignLowLikelihoodPairMerger
```

```
clique merger:
1 LeastGapsCliqueMerger (default)
2 BasicCliqueMerger
```

```
merger:
1 LeafMerger
2 MostHyperMerger (default)
```

```
edge length strategy:
1 NullEdgeLengthStrategy (default)
2 NumberOfSequencesEdgeLengthStrategy
```

```
likelihood strategy:
1 ProductLikelihoodStrategy (default)
2 MinimumLikelihoodStrategy
```

```
likelihood threshold:
Type Float between 0 and 1, default 0.5F
```

### A.2.1 Harvester Package

This package contains 2 miscellaneous classes.

**App Class** This is where the program is entered from the command line.

```
1 public class App {
2     protected Merger merger;
3     public static Point reAlignmentsAccepted = new Point(0, 0);
4
5     public static void main(String[] args) throws IOException {
6
7         //read options input from command line
8         Options opt = new Options(args, Multiplicity.ZERO_OR_ONE, 0, Integer.MAX_VALUE);
9         opt.setSet()
10            .addOption("i", Separator.EQUALS)
11            .addOption("o", Separator.EQUALS)
12            .addOption("m", Separator.EQUALS)
13            .addOption("c", Separator.EQUALS)
14            .addOption("p", Separator.EQUALS)
15            .addOption("e", Separator.EQUALS)
16            .addOption("l", Separator.EQUALS)
17            .addOption("t", Separator.EQUALS);
18
19         OptionData opdat;
20         String infile = "";
21         String outfile = "";
22         String mergerno = "0";
23         String cliqueMergerno = "0";
24         String pairMergerno = "0";
25         String edgeLengthStrategyno = "0";
26         String likelihoodStrategyno = "0";
27         Float likelihoodThreshold = 0.5F;
28         opt.getMatchingSet(true, false);
29
30         if (opt.getSet().getData().size() > 0) {
31             if (opt.getSet().getData().contains("help")) {
32                 help();
33             }
34         }
35
36         if ((opdat = opt.getSet().getOption("i")).getResultCount() > 0)
37             infile = opdat.getResultValue(0);
38         else if ((opt.getSet().getData()).size() > 0)
39             infile = opt.getSet().getData().get(0);
40         else {
41             System.out.println("You must specify a file for input");
42             System.exit(0);
43         }
44         if ((opdat = opt.getSet().getOption("o")).getResultCount() > 0)
```

```

45         outfile = opdat.getResultValue(0);
46     else
47         outfile = infile.replaceAll(".fastas", "");
48     if ((opdat = opt.getSet().getOption("m")).getResultCount() > 0)
49         mergerno = opdat.getResultValue(0);
50     if ((opdat = opt.getSet().getOption("c")).getResultCount() > 0)
51         cliqueMergerno = opdat.getResultValue(0);
52     if ((opdat = opt.getSet().getOption("p")).getResultCount() > 0)
53         pairMergerno = opdat.getResultValue(0);
54     if ((opdat = opt.getSet().getOption("e")).getResultCount() > 0)
55         edgeLengthStrategyno = opdat.getResultValue(0);
56     if ((opdat = opt.getSet().getOption("l")).getResultCount() > 0)
57         likelihoodStrategyno = opdat.getResultValue(0);
58     if ((opdat = opt.getSet().getOption("t")).getResultCount() > 0)
59         likelihoodThreshold = Float.valueOf(opdat.getResultValue(0));
60
61     Merger merger;
62     CliqueMerger cliqueMerger;
63     PairMerger pairMerger;
64     EdgeLengthStrategy edgeLengthStrategy;
65     LikelihoodStrategy likelihoodStrategy;
66
67     switch (Integer.valueOf(pairMergerno)) {
68         case 1: pairMerger = new SumGapPairMerger(); break;
69         case 2: pairMerger = new MaxGapPairMerger(); break;
70         //case 3: pairMerger = new WindowingPairMerger( args needed! ); break;
71         //case 4: pairMerger = new MaxGapPairMergerScored( args needed! ); break;
72         case 5: pairMerger = new StatAlignGapsPairMerger(); break;
73         case 6: pairMerger = new StatAlignLowLikelihoodPairMerger(); break;
74         case 7: pairMerger = new MAFFTAlignGapsPairMerger(); break;
75     default: pairMerger = new SumGapPairMerger(); break;
76     }
77
78     switch (Integer.valueOf(cliqueMergerno)) {
79         case 1: cliqueMerger = new LeastGapsCliqueMerger(pairMerger); break;
80         case 2: cliqueMerger = new BasicCliqueMerger(pairMerger); break;
81     default: cliqueMerger = new LeastGapsCliqueMerger(pairMerger); break;
82     }
83
84     switch (Integer.valueOf(mergerno)) {
85         case 1: merger = new LeafMerger(cliqueMerger); break;
86         case 2: merger = new MostHyperMerger(cliqueMerger); break;
87     default: merger = new MostHyperMerger(cliqueMerger); break;
88     }
89
90     switch (Integer.valueOf(edgeLengthStrategyno)) {
91         case 1: edgeLengthStrategy = new NullEdgeLengthStrategy(); break;
92         case 2: edgeLengthStrategy = new NumberOfSequencesEdgeLengthStrategy(); break;
93     default: edgeLengthStrategy = new NullEdgeLengthStrategy(); break;
94     }
95
96     switch (Integer.valueOf(likelihoodStrategyno)) {
97         case 1: likelihoodStrategy = new ProductLikelihoodStrategy(likelihoodThreshold); break;
98         case 2: likelihoodStrategy = new MinimumLikelihoodStrategy(likelihoodThreshold); break;
99     default: likelihoodStrategy = new ProductLikelihoodStrategy(likelihoodThreshold); break;
100    }
101
102    //read in the spannoid
103    Spannoid spannoid = ReadWriteTools.readFASTASFile(infile, edgeLengthStrategy, likelihoodStrategy);
104
105    //work out the sizes of all the components in the spannoid to confirm input to user (for testing)
106    ReadWriteTools.printSpannoidStructure(infile, spannoid);
107
108    //merges the spannoid
109    merger.merge(spannoid);
110
111    //outputs 3 files
112    ReadWriteTools.writeToALNFileNoScores(spannoid, outfile);
113    ReadWriteTools.writeToALNFiles(spannoid, outfile);
114    ReadWriteTools.writeToMSFFile(spannoid, outfile);
115
116    //only used when likelihoods are used during pairmerging
117    if (App.reAlignmentsAccepted.y > 0)
118        System.out.println("Accepted " + App.reAlignmentsAccepted.x + " of "
119            + App.reAlignmentsAccepted.y + " possible partial realignments");
120
121    }
122
123    //displays help
124    private static void help() {
125        String errorMessage = "\nUsage: java -jar harvester2.jar [-i=inputfile [-o=outfile]" + "\n" +
126            "[-m=merger] [-c=clique merger] [-p=pair merger] [-e=edge length strategy]" +
127            "[-l=likelihood strategy] [-t=likelihood threshold]" +
128            "\n\n";
129        System.out.println(errorMessage);
130        System.out.println("pair merger: \n" +
131            "1 SumGapPairMergerBasic (default) \n" +
132            "2 MaxGapPairMergerBasic \n" +
133            "3 WindowingPairMerger (not finished) \n" +
134            "4 MaxGapPairMergerScored (not finished) \n" +
135            "5 StatAlignGapsPairMerger \n" +
136            "6 StatAlignLowLikelihoodPairMerger \n" +
137            "7 MAFFTAlignGapsPairMerger \n\n" +
138            "clique merger: \n" +
139            "1 LeastGapsCliqueMerger (default) \n" +
140            "2 BasicCliqueMerger \n\n" +
141            "merger: \n" +
142            "1 LeafMerger \n" +

```

```

144         "2 MostHyperMerger (default) \n\n" +
145         "edge length strategy: \n" +
146         "1 NullEdgeLengthStrategy (default) \n" +
147         "2 NumberOfSequencesEdgeLengthStrategy \n\n" +
148         "likelihood strategy: \n" +
149         "1 ProductLikelihoodStrategy (default) \n" +
150         "2 MinimumLikelihoodStrategy \n\n" +
151         "likelihood threshold: \n" +
152         "Type Float, default 0.5F \n");
153     System.exit(0);
154 }
155 }
156 }
157 }
158 }
159 }

```

**ReadWriteTools Class** This is where static methods for reading FASTA files or FASTAS files with or without likelihood scores from stalign and for writing to FASTA with or without likelihood scores and MSF format. There is also a method for outputting the spannoid components of a spannoid to a text file for testing purposes.

```

1 public class ReadWriteTools {
2
3     private ReadWriteTools() {}
4
5     //read fasta files
6
7     public static Alignment readFASTAFile(String file, LikelihoodStrategy combiner)
8         throws IOException {
9         BufferedReader reader = new BufferedReader(new FileReader(file));
10
11         ArrayList<Sequence> alignmentList = new ArrayList<Sequence>();
12         ArrayList<Float> likelihoods = null;
13         String line = null;
14         String name = "";
15         String seq = "";
16         boolean readSeq = false;
17         boolean readingScores = false;
18
19         while ((line = reader.readLine()) != null) {
20             if (!line.trim().isEmpty()) {
21                 if (line.charAt(0) == '>') {
22                     if (readSeq) {
23                         alignmentList.add(new Sequence(seq, name));
24                     }
25                     readSeq = true;
26                     name = line.substring(1, line.length());
27                     seq = reader.readLine();
28
29                 } else if (line.trim().equals("#scores")) {
30                     if (readSeq) {
31                         readingScores = true;
32                         likelihoods = new ArrayList<Float>();
33                     }
34                 } else {
35                     if (readingScores)
36                         likelihoods.add(Float.valueOf(line));
37                     else
38                         seq += line;
39                 }
40             }
41         }
42
43         if (readSeq) {
44             alignmentList.add(new Sequence(seq, name));
45         }
46
47         return new Alignment(alignmentList, likelihoods, combiner);
48     }
49
50
51     public static Alignment readSingleFASTAFileNoScores(String file) throws IOException {
52         BufferedReader reader = new BufferedReader(new FileReader(file));
53
54         ArrayList<Sequence> alignmentList = new ArrayList<Sequence>();
55
56         String line = null;
57         String name = "";
58         String seq = "";
59
60         while ((line = reader.readLine()) != null) {
61             if (!line.trim().isEmpty()) {
62                 if (line.charAt(0) == '>') {
63                     if (name != "") {
64                         alignmentList.add(new Sequence(seq, name));
65                     }
66                     name = line.substring(1, line.length());
67                     seq = reader.readLine();
68                 } else {
69                     seq = seq + line;
70                 }
71             }
72         }
73
74         return new Alignment(alignmentList, null, null);
75     }
76 }

```

```

71         }
72     }
73 }
74
75     alignmentList.add(new Sequence(seq, name));
76
77     return new Alignment(alignmentList);
78 }
79
80 public static Spannoid readFASTAFiles(String[] files) throws IOException {
81     Spannoid spannoid = new Spannoid();
82     for (int i=0; i< files.length; i++) {
83         spannoid.add(new SpannoidComponent(readSingleFASTAFileNoScores(files[i]));)
84     }
85     return spannoid;
86 }
87
88 //read fastas files
89
90 public static Spannoid readFASTASFile(String file) throws IOException {
91     return readFASTASFile(file, new NullEdgeLengthStrategy(),
92         new ProductLikelihoodStrategy(0.5F));
93 }
94
95 public static Spannoid readFASTASFile(String file, EdgeLengthStrategy edgeLengthStrategy,
96     LikelihoodStrategy combiner) throws IOException {
97     BufferedReader reader = new BufferedReader(new FileReader(file));
98
99     Spannoid spannoid = new Spannoid(edgeLengthStrategy);
100
101     ArrayList<Sequence> alignmentList = null;
102     ArrayList<Float> likelihoods = null;
103     String line = null;
104     String name = "";
105     String seq = "";
106     boolean readSeq = false;
107     boolean readingScores = false;
108
109     while ((line = reader.readLine()) != null) {
110         if (!line.trim().isEmpty()) {
111             if (line.charAt(0) == '*') {
112                 if (readSeq) {
113                     alignmentList.add(new Sequence(seq, name));
114                     if (!readingScores) {
115                         spannoid.add(new SpannoidComponent(new Alignment(alignmentList)));
116                     } else {
117                         spannoid.add(new SpannoidComponent(new Alignment(alignmentList,
118                             likelihoods)));
119                     }
120                 }
121                 readSeq = false;
122                 readingScores = false;
123                 alignmentList = new ArrayList<Sequence>();
124             } else if (line.charAt(0) == '>') {
125                 if (readSeq) {
126                     alignmentList.add(new Sequence(seq, name));
127                 }
128                 readingScores = false;
129                 readSeq = true;
130                 name = line.substring(1, line.length());
131                 seq = reader.readLine();
132             } else if (line.trim().equals("#scores")) {
133                 if (readSeq) {
134                     readingScores = true;
135                     likelihoods = new ArrayList<Float>();
136                 }
137             } else {
138                 if (readingScores)
139                     likelihoods.add(Float.valueOf(line));
140                 else
141                     seq += line;
142             }
143         }
144     }
145
146     if (readSeq) {
147         alignmentList.add(new Sequence(seq, name));
148         if (!readingScores) {
149             spannoid.add(new SpannoidComponent(new Alignment(alignmentList)));
150         } else {
151             spannoid.add(new SpannoidComponent(new Alignment(alignmentList, likelihoods)));
152         }
153     }
154
155     return spannoid;
156 }
157
158 //write fasta files
159
160 public static void writeToALNFile(Alignment alignment, String fileName) throws IOException {
161     BufferedWriter writer = new BufferedWriter(new FileWriter(new File(fileName + ".aln")));
162     for (Sequence seq : alignment) {
163         writer.write(">");
164         writer.write(seq.getName());
165         writer.newLine();
166         writer.write(seq.getSequence());
167         writer.newLine();
168     }
169 }

```

```

170     }
171     writer.write("#scores");
172     for (Float likelihood : alignment.getLikelihoods()) {
173         writer.newLine();
174         writer.write(String.valueOf(likelihood));
175     }
176     writer.close();
177 }
178
179 public static void writeToALNFiles(Spannoid spannoid, String fileName) throws IOException {
180     int i = 1;
181     if (spannoid.size() > 1) {
182         for (SpannoidComponent component : spannoid) {
183             writeToALNFile(component.getAlignment(), fileName + i);
184             i++;
185         }
186     } else if (spannoid.size() == 1) {
187         writeToALNFile(spannoid.get(0).getAlignment(), fileName);
188     } else {
189         System.out.println("Spannoid has no components");
190     }
191 }
192
193 public static void writeToFASTAFileNoScores(ArrayList<Sequence> sequences, String fileName)
194 {
195     BufferedWriter writer = new BufferedWriter(new FileWriter(new File(fileName + ".fasta")));
196     for (Sequence seq : sequences) {
197         writer.write(">");
198         writer.write(seq.getName());
199         writer.newLine();
200         writer.write(seq.getSequence());
201         writer.newLine();
202     }
203     writer.close();
204 }
205
206 //write msf files
207
208 public static void writeToMSFFile(Alignment alignment, String fileName) throws IOException {
209     BufferedWriter writer = new BufferedWriter(new FileWriter(new File(fileName + ".msf")));
210
211     writer.write("MSF: " + alignment.getSequenceLength() + " ");
212     writer.write("Type: P ");
213     writer.write("Check: 0 ");
214     writer.write("...");
215     writer.newLine();
216     writer.newLine();
217
218     for (Sequence seq : alignment) {
219         writer.write("Name: ");
220         writer.write(seq.getName());
221         for (int n = 0; n < 20 - seq.getName().length(); n++) {
222             writer.write(" ");
223         }
224         writer.write("Len: ");
225         writer.write(String.valueOf(seq.getLength()));
226         writer.write(" Check: 0");
227         writer.write(" Weight: 1.0");
228         writer.newLine();
229     }
230
231     writer.newLine();
232     writer.write("//");
233     writer.newLine();
234     writer.newLine();
235
236     int remainder50 = alignment.getSequenceLength() % 50;
237     int remainder10 = alignment.getSequenceLength() % 10;
238
239     int maxNameLength = 20;
240     for (Sequence seq : alignment) {
241         if (seq.getName().length() > maxNameLength)
242             maxNameLength = seq.getName().length();
243     }
244     maxNameLength+=2;
245
246     for (int i = 0; i < alignment.getSequenceLength() - remainder50; i += 50) {
247         for (Sequence seq : alignment) {
248             writer.write(seq.getName());
249             for (int n = 0; n < maxNameLength - seq.getName().length(); n++) {
250                 writer.write(" ");
251             }
252             for (int j = 0; j < 50; j += 10) {
253                 writer.write(seq.getSequence().substring(i+j, i + j + 10));
254                 writer.write(" ");
255             }
256             writer.newLine();
257         }
258     }
259     writer.newLine();
260
261     int i = alignment.getSequenceLength() - remainder50;
262
263     if (remainder50 != 0) {
264         for (Sequence seq : alignment) {
265             writer.write(seq.getName());
266         }
267     }
268 }

```

```

269         for (int n = 0; n < maxNameLength - seq.getName().length(); n++) {
270             writer.write(" ");
271         }
272         for (int j = 0; j < remainder50 - remainder10; j += 10) {
273             writer.write(seq.getSequence().substring(i+j, i + j + 10));
274             writer.write(" ");
275         }
276         writer.write(seq.getSequence().substring(alignment.getSequenceLength() - remainder10,
277         alignment.getSequenceLength()));
278         writer.newLine();
279     }
280 }
281
282 writer.close();
283 }
284
285 public static void writeToMSFFile(Spannoid spannoid, String fileName) throws IOException {
286     int i = 1;
287     if (spannoid.size() > 1) {
288         for (SpannoidComponent component : spannoid) {
289             writeToMSFFile(component.getAlignment(), fileName + i);
290             i++;
291         }
292     } else if (spannoid.size() == 1) {
293         writeToMSFFile(spannoid.get(0).getAlignment(), fileName);
294     } else {
295         System.out.println("Spanoid has no components");
296     }
297 }
298
299 public static void writeToALNFileNoScores(Alignment alignment, String fileName) throws IOException {
300     BufferedWriter writer = new BufferedWriter(new FileWriter(new File(fileName + "_no_scores.aln")));
301     for (Sequence seq : alignment) {
302         writer.write(">");
303         writer.write(seq.getName());
304         writer.newLine();
305         writer.write(seq.getSequence());
306         writer.newLine();
307     }
308     writer.close();
309 }
310
311 public static void writeToALNFileNoScores(Spannoid spannoid, String fileName) throws IOException {
312     int i = 1;
313     if (spannoid.size() > 1) {
314         for (SpannoidComponent component : spannoid) {
315             writeToALNFileNoScores(component.getAlignment(), fileName + i);
316             i++;
317         }
318     } else if (spannoid.size() == 1) {
319         writeToALNFileNoScores(spannoid.get(0).getAlignment(), fileName);
320     } else {
321         System.out.println("Spanoid has no components");
322     }
323 }
324
325 public static void printSpannoidStructure(String infile, Spannoid spannoid) {
326     ArrayList<Integer> compSizes = new ArrayList<Integer>();
327
328     for (SpannoidComponent comp : spannoid) {
329         int size = comp.getAlignment().getNumberOfSequences();
330         if (size >= compSizes.size())
331             for (int i = compSizes.size(); i <= size; i++){
332                 compSizes.add(0);
333             }
334         compSizes.set(size, compSizes.get(size) + 1);
335     }
336     int numberOfComps = 0;
337     int index = 0;
338     String compStructure = "";
339     for (Integer n : compSizes) {
340         if (n > 0) {
341             numberOfComps += n;
342             compStructure += n + " x " + index + "\n";
343         }
344         index++;
345     }
346     System.out.println("Read spannoid components:");
347     System.out.println("Read file " + infile + " with " + numberOfComps + " components:");
348     System.out.println(compStructure);
349
350     BufferedWriter writer;
351     try {
352         writer = new BufferedWriter(new FileWriter(new File(infile + ".comps")));
353         writer.write(compStructure);
354         writer.close();
355     } catch (IOException e) {
356         e.printStackTrace();
357     }
358 }
359
360 }

```

## A.2.2 Spannoid Structure Package

This package contains the classes that represent the underlying data structure of the spannoid and of sequences and alignments.

**Spannoid Class** This is the data structure for storing the hypergraph of alignments, AKA the spannoid.

```

1
2 public class Spannoid extends ArrayList<SpannoidComponent> {
3     private ArrayList<HyperEdge> edges;
4     private EdgeLengthStrategy edgeLengthStrategy;
5
6     //create a spannoid with all edgelengths 1
7     public Spannoid() {
8         super();
9         this.edges = new ArrayList<HyperEdge>();
10        this.edgeLengthStrategy = new NullEdgeLengthStrategy();
11    }
12
13    //create a spannoid with a particular strategy for edgelengths
14    public Spannoid(EdgeLengthStrategy edgeLengthStrategy) {
15        super();
16        this.edges = new ArrayList<HyperEdge>();
17        this.edgeLengthStrategy = edgeLengthStrategy;
18    }
19
20    @Override
21    public boolean add(SpannoidComponent newComponent) {
22        //initialise the variables
23        int edgeLength = 0;
24        HyperEdge connector = null;
25        //checks for neighbours and adds connectors if common sequences occur
26        for (SpannoidComponent component : this) {
27            for (Sequence seq1 : component.getAlignment()) {
28                for (Sequence seq2 : newComponent.getAlignment()) {
29                    if (seq1.getName().equals(seq2.getName())) {
30                        connector = component.getConnector(seq1.getName());
31                        if (connector == null) {
32                            edgeLength = edgeLengthStrategy.calculateLength(component.getAlignment(),
33                                seq2);
34                            connector = new HyperEdge(component, seq1, newComponent, seq2, edgeLength);
35                            edges.add(connector);
36                            component.addConnector(connector);
37                            newComponent.addConnector(connector);
38                        } else if (!connector.contains(newComponent)){
39                            connector.add(newComponent, seq2);
40                            newComponent.addConnector(connector);
41                        }
42                        break;
43                    }
44                }
45            }
46        }
47
48        super.add(newComponent);
49        return true;
50    }
51
52    //returns the maximum size of the components in the spannoid
53    public int getK() {
54        int k = 0;
55        for (SpannoidComponent component : this) {
56            k = Math.max(k, component.getAlignment().getNumberOfSequences());
57        }
58        return k;
59    }
60
61    public ArrayList<HyperEdge> getEdges() {
62        return edges;
63    }
64
65    public void remove(HyperEdge edgeToMerge) {
66        edges.remove(edgeToMerge);
67    }
68
69
70 }

```

**SpannoidComponent Class** This is basically a node in the spannoid hypergraph. It contains a reference to an alignment and connectors via which one can find its neighbouring components in the spannoid.

```

1 public class SpannoidComponent {
2     private Alignment alignment;
3     private List<HyperEdge> connectors;
4     //^^^a list of the hyperedges this component is connected to other comps by

```

```

5
6     public SpannoidComponent(Alignment alignment) {
7         setAlignment(alignment);
8         connectors = new ArrayList<HyperEdge>();
9     }
10
11     public void setAlignment(Alignment alignment) {
12         this.alignment = alignment;
13     }
14
15     public Alignment getAlignment() {
16         return alignment;
17     }
18
19     public void addConnector(HyperEdge connector) {
20         connectors.add(connector);
21     }
22
23     public HyperEdge getConnector(int connectorIndex) {
24         return connectors.get(connectorIndex);
25     }
26
27     public HyperEdge getConnector(SpannoidComponent component) {
28         for (HyperEdge connector : connectors) {
29             if (connector.contains(component))
30                 return connector;
31         }
32         return null;
33     }
34
35     public HyperEdge getConnector(String commonSequenceName) {
36         for (HyperEdge connector : connectors) {
37             if (connector.getCommonSequenceName().equals(commonSequenceName))
38                 return connector;
39         }
40         return null;
41     }
42
43     public List<HyperEdge> getConnectors() {
44         return connectors;
45     }
46
47     public int getDegree() {
48         return connectors.size();
49     }
50
51     public int getNumberOfConnectors() {
52         return connectors.size();
53     }
54
55     public void removeConnector(HyperEdge hyperEdge) {
56         connectors.remove(hyperEdge);
57         if (hyperEdge.contains(this))
58             hyperEdge.remove(this);
59     }
60
61     @Override
62     public String toString() {
63         String out = "[";
64         for (Sequence seq : getAlignment())
65             out += seq.getName() + ", ";
66         if (out.contains(", "))
67             out = out.substring(0, out.length() - 2);
68         out += "]";
69         return out;
70     }
71 }

```

**HyperEdge Class** This class represents the hyperedge, which contains references to the components it is connecting together and a list of the common sequence in each alignment. Note they are the same common sequence by name, but actually different sequence objects because they have different gaps.

```

1 public class HyperEdge {
2     protected ArrayList<SpannoidComponent> comps;
3     protected ArrayList<Sequence> commonSequences;
4     protected int edgeLength; //this is used if there are only two components,
5                               //but in this prototype it is unused
6
7     public HyperEdge(SpannoidComponent comp1, Sequence commonSequence1,
8                     SpannoidComponent comp2, Sequence commonSequence2, int edgeLength) {
9         comps = new ArrayList<SpannoidComponent>();
10        comps.add(comp1);
11        comps.add(comp2);
12        commonSequences = new ArrayList<Sequence>();
13        commonSequences.add(commonSequence1);
14        commonSequences.add(commonSequence2);
15        this.edgeLength = edgeLength;
16    }
17
18    public HyperEdge(ArrayList<SpannoidComponent> comps, ArrayList<Sequence> commonSequences){
19        for (SpannoidComponent comp : comps)

```

```

20         this.comps.add(comp);
21         this.commonSequences = commonSequences;
22         this.edgeLength = 0;
23     }
24
25     public boolean contains(SpannoidComponent component) {
26         return comps.contains(component);
27     }
28
29     public void add(SpannoidComponent comp, Sequence commonSequence) {
30         comps.add(comp);
31         commonSequences.add(commonSequence);
32     }
33
34     public SpannoidComponent getComponent(int i) {
35         return comps.get(i);
36     }
37
38     public Sequence getCommonSequence(SpannoidComponent component) {
39         int index = comps.indexOf(component);
40         if (index != -1)
41             return commonSequences.get(index);
42         else
43             return null;
44     }
45
46     public void setEdgeLength(int edgeLength) {
47         this.edgeLength = edgeLength;
48     }
49
50     public int getEdgeLength() {
51         return edgeLength;
52     }
53
54     public int getNumberOfComponents() {
55         return comps.size();
56     }
57
58     public void remove(SpannoidComponent component) {
59         for (int i = 0; i < comps.size(); i++) {
60             if (comps.get(i) == component) {
61                 comps.remove(i);
62                 commonSequences.remove(i);
63                 break;
64             }
65         }
66     }
67
68     public ArrayList<Sequence> getCommonSequences() {
69         return commonSequences;
70     }
71
72     public String getCommonSequenceName() {
73         return commonSequences.get(0).getName();
74     }
75
76     public String toString() {
77         return "Common Sequence: " + getCommonSequenceName() +
78             " Components: " + comps.toString();
79     }
80
81 }

```

**Alignment Class** This is where all the sequences of an alignment are stored. An invariant is that all the sequences have the same length.

```

1 public class Alignment implements Iterable<Sequence> {
2     protected ArrayList<Sequence> sequences;
3     protected ArrayList<Float> likelihoods;
4     ///^^^a list of the likelihood values for each column in the alignment
5     protected LikelihoodStrategy likelihoodStrategy;
6     protected int mergeCount = 0;
7
8
9     public Alignment(ArrayList<Sequence> alignment) {
10        int length = 0;
11        // ensure all sequences are of the same length
12        if (alignment.size() > 0) {
13            length = alignment.get(0).getLength();
14            for (Sequence s : alignment) {
15                if (s.getLength() != length)
16                    System.out.println("Not valid alignment");
17            }
18        }
19        this.sequences = alignment;
20        likelihoods = new ArrayList<Float>();
21        for (int i = 0; i < length; i++) {
22            likelihoods.add(0.0F);
23        }
24        likelihoodStrategy = new ProductLikelihoodStrategy(0.5F);
25    }
26
27     public Alignment(ArrayList<Sequence> alignment, ArrayList<Float> likelihoods,
28         LikelihoodStrategy combiner) {

```

```

29         int length = 0;
30         // ensure all sequences are of the same length
31         if (alignment.size() > 0) {
32             length = alignment.get(0).getLength();
33             for (Sequence s : alignment) {
34                 if (s.getLength() != length)
35                     System.out.println("Not valid alignment");
36             }
37
38             if (likelihoods.size() != length)
39                 System.out.println("Not valid likelihood list for this alignment");
40         }
41
42         this.sequences = alignment;
43         this.likelihoods = likelihoods;
44         this.likelihoodStrategy = combiner;
45     }
46
47     public void addSequence(Sequence seq) {
48         if (sequences.size() > 0)
49             if (seq.getLength() != sequences.get(0).getLength()) {
50                 System.out.println("Aligned sequence " + seq.getName()
51                                     + " not of same length as others in alignment.");
52             }
53         sequences.add(seq);
54     }
55
56     public void addSequence(int index, Sequence seq) {
57         if (sequences.size() > 0)
58             if (seq.getLength() != sequences.get(0).getLength()) {
59                 System.out.println("Aligned sequence " + seq.getName()
60                                     + " not of same length as others in alignment.");
61             }
62         sequences.add(index, seq);
63     }
64
65     public int getNumberOfSequences() {
66         return sequences.size();
67     }
68
69     public Sequence getSequence(int i) {
70         return sequences.get(i);
71     }
72
73     // insert a gap in all sequences
74     public void insertGap(int index) {
75         for (Sequence s : sequences)
76             s.insertGap(index);
77         likelihoods.add(index, 0.0F);
78     }
79
80     // so that we can use for(Sequence seq : alignment) ...
81     @Override
82     public Iterator<Sequence> iterator() {
83         return sequences.iterator();
84     }
85
86     // add the sequences from another alignment onto this one
87     public void append(Alignment alignment) {
88         for (Sequence seq : alignment)
89             this.addSequence(seq);
90         likelihoodStrategy.combine(getLikelihoods(), alignment.getLikelihoods());
91     }
92
93     public ArrayList<Float> getLikelihoods() {
94         return likelihoods;
95     }
96
97     public LikelihoodStrategy getLikelihoodStrategy() {
98         return likelihoodStrategy;
99     }
100
101     // remove a particular sequence from the alignment
102     public void remove(int i) {
103         sequences.remove(i);
104     }
105
106     // returns a list of new corresponding sequences without gaps
107     public ArrayList<Sequence> rawAlignment() {
108         ArrayList<Sequence> sequences = new ArrayList<Sequence>();
109         for (Sequence s : this) {
110             String seq = "";
111             for (int i = 0; i < s.getLength(); i++) {
112                 char c = s.getChar(i);
113                 if (c != '-') {
114                     seq = seq + c;
115                 }
116             }
117             sequences.add(new Sequence(seq, s.getName()));
118         }
119         return sequences;
120     }
121
122     // returns a new alignment corresponding to alignment between column $a and $b
123     public Alignment subAlignment(int a, int b) {
124         ArrayList<Float> likelihoods = new ArrayList<Float>();
125         for (int i = a; i < b; i++) {
126             likelihoods.add(this.likelihoods.get(i));
127         }

```

```

128     }
129     Alignment subAlignment = new Alignment(new ArrayList<Sequence>(), likelihoods ,
130
131     for (Sequence seq : sequences) {
132         subAlignment.addSequence(new Sequence(seq.getSequence().substring(a,b) , seq.getName()));
133     }
134     return subAlignment;
135 }
136
137
138 // returns true if column a in alignment contains only gaps
139 public Boolean isGapColumn(int a) {
140     Boolean contains = true;
141     for (int i = 0; i < sequences.size(); i++) {
142         if (this.getSequence(i).getChar(a) != '-')
143             contains = false;
144     }
145     return contains;
146 }
147
148 public void remove(Sequence commonSequence) {
149     sequences.remove(commonSequence);
150 }
151
152 @Override
153 public String toString() {
154     String out = new String();
155     for (Sequence seq : sequences) {
156         out += seq.toString() + "\n";
157     }
158     return out;
159 }
160
161 //increment the number of times this alignment has been merged
162 public void increaseMergeCount() {
163     mergeCount++;
164 }
165
166 //return the number of times this alignment has been merged
167 public int getMergeCount() {
168     return mergeCount;
169 }
170
171 // returns the number of columns between a and b containing only gaps
172 public int getNoOfPureGapColumnsBetweenPoints(int a, int b) {
173     int noOfColumns = 0;
174     if (a < b) {
175         for (int i = a; i < b; i++) {
176             if (this.isGapColumn(i) == true)
177                 noOfColumns++;
178         }
179     }
180     return noOfColumns;
181 }
182
183 // delete the a'th column
184 public void deleteColumn(int a) {
185     for (Sequence seq : this)
186         seq.deleteChar(a);
187     likelihoods.remove(a);
188 }
189
190 public int getSequenceLength() {
191     return getSequence(0).getLength(); // remember we ensure lengths are all the same
192 }
193
194 // insert $number of gaps at position $start
195 public void insertGaps(int start, int number) {
196     if (start >= 0 && start <= getSequenceLength()) {
197         for (int j = 0; j < number; j++) {
198             insertGap(start);
199         }
200     } else {
201         throw new IndexOutOfBoundsException("Index out of bounds for insert gap index: "
202             + start + " size: " + getSequenceLength());
203     }
204 }
205
206 public boolean contains(Sequence seq) {
207     return sequences.contains(seq);
208 }
209
210
211 //replaces the characters between indices a and b in all sequences
212 //with a new alignment
213 public void replace(Alignment newSubAlignment, int a, int b) {
214     if (newSubAlignment.getNumberOfSequences() == getNumberOfSequences()) {
215         for (int i = 0; i < getNumberOfSequences(); i++) {
216             getSequence(i).replace(newSubAlignment.getSequence(i), a, b);
217         }
218         for (int i = a; i < b; i++) {
219             likelihoods.remove(a);
220         }
221         for (int i = 0; i < newSubAlignment.getSequenceLength(); i++)
222             likelihoods.add(a + i, newSubAlignment.getLikelihoods().get(i));
223     } else {
224         System.out.println("Incorrect number of sequences reinput into alignment");
225     }
226 }

```

```

227
228
229 //returns a list of the indices of sequences that contain actual characters and not just gaps
230 public ArrayList<Integer> getNoneEmptySequences() {
231     ArrayList<Integer> noneEmptySeqs = new ArrayList<Integer>();
232     for (int i = 0; i < getNumberOfSequences(); i++) {
233         if (!getSequence(i).getSequence().replaceAll("-", "").isEmpty())
234             noneEmptySeqs.add(i);
235     }
236     return noneEmptySeqs;
237 }
238
239 //returns a list of the sequences' names
240 public ArrayList<String> getNames() {
241     ArrayList<String> names = new ArrayList<String>();
242     for (Sequence seq : sequences) {
243         names.add(seq.getName());
244     }
245     return names;
246 }
247
248 //returns a sequence from input name, returns null if alignment does not
249 //contain sequence with that name
250 public Sequence getSequence(String name) {
251     for (Sequence seq : sequences) {
252         if (seq.getName().equals(name))
253             return seq;
254     }
255     return null;
256 }
257
258
259 //returns a list of doubles (region start, region length) for regions where the
260 //likelihood is lower than the threshold specified in the command line or the default 0.5F
261 public ArrayList<Point> lowLikelihoodRegions() {
262     ArrayList<Point> regionList = new ArrayList<Point>();
263     ArrayList<Integer> regionStart = new ArrayList<Integer>();
264     ArrayList<Integer> regionEnd = new ArrayList<Integer>();
265     Float threshold = likelihoodStrategy.threshold;
266
267     if (getLikelihoods().get(0) < threshold)
268         regionStart.add(0);
269
270     for (int i=1; i < getSequenceLength(); i++) {
271         if (getLikelihoods().get(i-1) >= threshold && getLikelihoods().get(i) < threshold)
272             regionStart.add(i);
273
274         else if (getLikelihoods().get(i-1) < threshold && getLikelihoods().get(i) >= threshold)
275             regionEnd.add(i-1);
276     }
277
278     if (regionEnd.size() < regionStart.size())
279         regionEnd.add(getSequenceLength() - 1);
280
281     for (int i=0; i < regionStart.size(); i++) {
282         Point region = new Point();
283         region.x = regionStart.get(i);
284         region.y = regionEnd.get(i) - regionStart.get(i) + 1;
285         regionList.add(region);
286     }
287     return regionList;
288 }
289
290
291 //returns a list of doubles (region start, region length) where regions of
292 //likelihood are treated separate from the common gaps, which are input as a parameter
293 public ArrayList<Point> lowLikelihoodRegionsPlusGaps(ArrayList<Point> gapList) {
294     ArrayList<Point> regionList = new ArrayList<Point>();
295     ArrayList<Integer> regionStart = new ArrayList<Integer>();
296     ArrayList<Integer> regionEnd = new ArrayList<Integer>();
297     Float threshold = likelihoodStrategy.threshold - mergeCount/10;
298
299     int j = 0;
300     boolean inGap = false;
301     boolean inRegion = false;
302
303     if (j < gapList.size()) {
304         if (gapList.get(j).x == 0) {
305             regionStart.add(0);
306             inGap = true;
307         }
308     }
309
310     if (!inGap && getLikelihoods().get(0) < threshold) {
311         regionStart.add(0);
312         inRegion = true;
313     }
314
315     for (int i=1; i < getSequenceLength(); i++) {
316         if (inGap) { //if we are in a gap region, check if we are at the end of it
317             if (gapList.get(j).x + gapList.get(j).y == i) {
318                 regionEnd.add(i-1);
319                 inGap = false;
320                 j++;
321                 if (getLikelihoods().get(i) < threshold) {
322                     regionStart.add(i);
323                     inRegion = true;
324                 }
325             }
326         }
327     }

```

```

326     } else if (inRegion) { //if we are in a nongap region, check
327         //if we are at the end of it
328         if (getLikelihoods().get(i) >= threshold) {
329             regionEnd.add(i-1);
330             inRegion = false;
331             if (j < gapList.size()) {
332                 if (gapList.get(j).x == i) {
333                     regionStart.add(i);
334                     inGap = true;
335                 }
336             }
337         }
338     } else { //if we are not in any gap, check for the start of one
339         if (j < gapList.size()) {
340             if (gapList.get(j).x == i) {
341                 regionStart.add(i);
342                 inGap = true;
343             }
344         }
345         if (!inGap && getLikelihoods().get(i) < threshold) {
346             regionStart.add(i);
347             inRegion = true;
348         }
349     }
350 }
351
352 if (regionEnd.size() < regionStart.size())
353     regionEnd.add(getSequenceLength() - 1);
354
355 for (int i=0; i < regionStart.size(); i++) {
356     Point region = new Point();
357     region.x = regionStart.get(i);
358     region.y = regionEnd.get(i) - regionStart.get(i) + 1;
359     regionList.add(region);
360 }
361 return regionList;
362 }
363 }

```

**Sequence Class** This class basically has a reference to a character array for the actual sequence, because it is mutable, and a string for the name of the sequence.

```

1 public class Sequence {
2     protected char[] seqArray; //An array to store the sequence and null spaces for insertions
3     protected int length; //number of non-null chars in seqArray
4     protected String name;
5
6     public Sequence(String seq, String name) {
7         this.seqArray = seq.toCharArray();
8         length = this.seqArray.length;
9         this.name = name;
10    }
11
12    //returns number of non-null chars in seqArray
13    public int getLength() {
14        return length;
15    }
16
17    public String getName() {
18        return name;
19    }
20
21    //returns character at index index in seqArray
22    public char getChar(int index) {
23        if ((index < 0))
24            throw new IndexOutOfBoundsException();
25        else if (index >= length)
26            return '*';
27        else {
28            return seqArray[index];
29        }
30    }
31
32    }
33
34    //if we get too many chars in seqArray (>length), extend it to minimumCapacity
35    public void ensureCapacity(int minimumCapacity) {
36        if (seqArray.length <= minimumCapacity) {
37            char[] newSeq = new char[minimumCapacity];
38            System.arraycopy(seqArray, 0, newSeq, 0, seqArray.length);
39            seqArray = newSeq;
40        }
41    }
42
43    //insert a gap in the sequence at index i
44    public void insertGap(int i) {
45        if (length + 1 >= seqArray.length)
46            ensureCapacity(seqArray.length * 3/2);
47        if ((i < 0 || i > length))
48            throw new IndexOutOfBoundsException();
49        else if (i == length) {
50            seqArray[i] = '-';
51            length++;
52        }
53        else {

```

```

53         System.arraycopy(seqArray, i, seqArray, i+1, length - i);
54         seqArray[i] = '-';
55         length++;
56     }
57 }
58
59 //returns the non-null characters in the sequence
60 public String getSequence() {
61     return new String(seqArray).trim();
62 }
63
64 //output the sequence as a string
65 public String toString() {
66     return ">" + getName() + "\n" + getSequence();
67 }
68
69 //returns indices of start of gaps and lengths of the gaps
70 public ArrayList<Point> getGaps() {
71     ArrayList<Point> gapList = new ArrayList<Point>();
72     ArrayList<Integer> gapStart = new ArrayList<Integer>();
73     ArrayList<Integer> gapEnd = new ArrayList<Integer>();
74
75     if (this.getChar(0) == '-')
76         gapStart.add(0);
77
78     for (int i=0; i < getLength(); i++) {
79         if (getChar(i) != '-' && getChar(i+1) == '-')
80             gapStart.add(i+1);
81
82         else if (getChar(i) == '-' && getChar(i+1) != '-')
83             gapEnd.add(i);
84     }
85
86     for (int i=0; i < gapStart.size(); i++) {
87         Point gap = new Point();
88         gap.x = gapStart.get(i);
89         gap.y = gapEnd.get(i)-gapStart.get(i) + 1;
90         gapList.add(gap);
91     }
92     return gapList;
93 }
94
95 public void deleteChar(int i) {
96     System.arraycopy(seqArray, i, seqArray, i-1, length - i);
97     length--;
98 }
99
100 public void setChar(int i, char c) {
101     if (i < getLength())
102         seqArray[i] = c;
103 }
104
105 public void replace(Sequence newSubsequence, int a, int b) {
106     removeChars(a, b);
107     int num = newSubsequence.getLength();
108     insertGaps(a, num);
109     for (int i = 0; i < num; i++) {
110         setChar(a + i, newSubsequence.getChar(i));
111     }
112 }
113
114 private void removeChars(int a, int b) {
115     if (b > a) {
116         System.arraycopy(seqArray, b, seqArray, a, length - b);
117         length -= (b-a);
118     }
119 }
120
121 private void insertGaps(int start, int num) {
122     if (length + num >= seqArray.length)
123         ensureCapacity((getLength() + num) * 3/2);
124     if ((start < 0 || start > this.length))
125         throw new IndexOutOfBoundsException("Insert Gaps Error Index: " + start + " Length: " + length);
126     else {
127         System.arraycopy(seqArray, start, seqArray, start + num, length - start);
128         for (int i = start; i < start + num; i++)
129             setChar(i, '-');
130         this.length += num;
131     }
132 }
133
134 public int numberOfNoneGapChars() {
135     int num = 0;
136     for (int i = 0; i < getLength(); i++) {
137         if (getChar(i) != '-')
138             num++;
139     }
140     return num;
141 }
142 }
143 }
144 }

```

### A.2.3 Mergers Package

This package contains the simple interface for a merger, and 2 classes that implement it for merging the spannoid on the larger scale, deferring small scale merging to the clique merger.

```

1  public interface Merger {
2      //merges the spannoid into one alignment
3      public void merge(Spannoid spannoid);
4      //returns the clique merger
5      public CliqueMerger getCliqueMerger();
6  }
7
8
9
10 public class MostHyperMerger implements Merger {
11     CliqueMerger cliqueMerger;
12
13     public MostHyperMerger(CliqueMerger cliqueMerger) {
14         this.cliqueMerger = cliqueMerger;
15     }
16
17     @Override
18     public CliqueMerger getCliqueMerger() {
19         return cliqueMerger;
20     }
21
22     @Override
23     public void merge(Spannoid spannoid) {
24         ArrayList<HyperEdge> edges = spannoid.getEdges();
25         HyperEdge edgeToMerge = null;
26         int currentMax = 0;
27
28         while (edges.size() > 0) {
29             currentMax = (edgeToMerge = edges.get(0)).getNumberOfComponents();
30             for (int i = 1; i < edges.size(); i++) {
31                 if (edges.get(i).getNumberOfComponents() > currentMax)
32                     currentMax = (edgeToMerge = edges.get(i)).getNumberOfComponents();
33             }
34             cliqueMerger.merge(spannoid, edgeToMerge);
35         }
36     }
37 }
38
39
40 public class LeafMerger implements Merger {
41     protected CliqueMerger cliqueMerger;
42
43     public LeafMerger(CliqueMerger cliqueMerger) {
44         this.cliqueMerger = cliqueMerger;
45     }
46
47
48     @Override
49     public void merge(Spannoid spannoid) {
50         LinkedList<HyperEdge> deg1Leaves;
51         LinkedList<SpannoidComponent> theirNeighbours;
52
53         SpannoidComponent comp1;
54         SpannoidComponent comp2;
55
56         while (spannoid.size() >= 2) {
57             if (spannoid.size() > 2) {
58
59                 deg1Leaves = new LinkedList<HyperEdge>();
60                 theirNeighbours = new LinkedList<SpannoidComponent>();
61
62                 for (HyperEdge edge : spannoid.getEdges()) {
63                     if (edge.getNumberOfComponents() == 2) {
64                         comp1 = edge.getComponent(0);
65                         comp2 = edge.getComponent(1);
66                         if (comp1.getDegree() == 1) {
67                             if (theirNeighbours.contains(comp2)) {
68                                 HyperEdge oldEdge = deg1Leaves.get(theirNeighbours.indexOf(comp2));
69                                 if (edge.getEdgeLength() < oldEdge.getEdgeLength()) {
70                                     deg1Leaves.remove(oldEdge);
71                                     theirNeighbours.remove(comp2);
72                                     deg1Leaves.add(edge);
73                                     theirNeighbours.add(comp2);
74                                 }
75                             }
76                         } else {
77                             deg1Leaves.add(edge);
78                             theirNeighbours.add(comp2);
79                         }
80                     } else if (comp2.getDegree() == 1) {
81                         if (theirNeighbours.contains(comp1)) {
82                             HyperEdge oldEdge = deg1Leaves.get(theirNeighbours.indexOf(comp1));
83                             if (edge.getEdgeLength() < oldEdge.getEdgeLength()) {
84                                 deg1Leaves.remove(oldEdge);
85                                 theirNeighbours.remove(comp1);
86                                 deg1Leaves.add(edge);
87                                 theirNeighbours.add(comp1);
88                             }
89                         } else {

```

```

90         deg1Leaves.add(edge);
91         theirNeighbours.add(compl);
92     }
93     }
94 }
95 }
96 }
97 if (deg1Leaves.size() > 0) {
98     //merge the leaves
99     while (deg1Leaves.size() > 0) {
100         HyperEdge edgeToMerge = deg1Leaves.get(0);
101         cliqueMerger.merge(spannoid, edgeToMerge);
102         deg1Leaves.remove();
103         theirNeighbours.remove(0);
104         System.gc();
105     }
106 }
107 //merge the cliques
108 ArrayList<HyperEdge> edges = spannoid.getEdges();
109 int i = 0;
110 while (i < edges.size()) {
111     HyperEdge edge = edges.get(i);
112     for (int j = 0; j < edge.getNumberOfComponents(); j++) {
113         if (edge.getComponent(j).getDegree() == 1) {
114             cliqueMerger.merge(spannoid, edge);
115             break;
116         }
117         i++;
118     }
119 }
120 }
121 }
122 } else if (spannoid.size() == 2) {
123     cliqueMerger.merge(spannoid, spannoid.getEdges().get(0));
124 }
125 }
126 }
127 }
128 public CliqueMerger getCliqueMerger() {
129     return cliqueMerger;
130 }
131 }
132 }

```

#### A.2.4 Clique Mergers Package

Here the skeleton for a cliquemerger is implemented as an abstract class with 2 subclasses, the least gaps clique merger and the basic clique merger. In the former we pair-merge the two alignments where the common sequences have the least gaps first repeatedly until we have one alignment remaining. In the latter we pair-merge the alignments in any order.

```

1 public abstract class CliqueMerger {
2     protected PairMerger pairMerger;
3
4     //cleanup of the redundant hyperedge is done here after merging
5     public abstract void merge(Spannoid spannoid, HyperEdge hyperedge);
6
7     public abstract PairMerger getPairMerger();
8
9     //cleanup of the redundant component is done here after merging
10    protected void pairMerge(Spannoid spannoid, HyperEdge hyperEdge, int index1, int index2) {
11        SpannoidComponent xComp = hyperEdge.getComponent(index1);
12        SpannoidComponent yComp = hyperEdge.getComponent(index2);
13
14        Sequence xSeq = hyperEdge.getCommonSequence(xComp);
15        Sequence ySeq = hyperEdge.getCommonSequence(yComp);
16
17        pairMerger.merge(xComp.getAlignment(), xSeq, yComp.getAlignment(), ySeq);
18
19        for (HyperEdge yCompsEdge : yComp.getConnectors()){
20            Sequence commonSequence = yCompsEdge.getCommonSequence(yComp);
21            //remove yComp from connector
22            yCompsEdge.remove(yComp);
23            //add xComp to connector
24            if (yCompsEdge != hyperEdge) {
25                yCompsEdge.add(xComp, commonSequence);
26                xComp.addConnector(yCompsEdge);
27            }
28        }
29        spannoid.remove(yComp);
30    }
31 }
32
33
34
35 public class LeastGapsCliqueMerger extends CliqueMerger {
36     public LeastGapsCliqueMerger(PairMerger pairMerger) {
37         this.pairMerger = pairMerger;
38     }

```

```

39     }
40
41     public PairMerger getPairMerger() {
42         return pairMerger;
43     }
44
45     public void merge(Spannoid spannoid, HyperEdge hyperEdge) {
46         int numberOfAlignments = hyperEdge.getCommonSequences().size();
47
48         while (numberOfAlignments > 2) {
49
50             ArrayList<Integer> numberOfGaps = new ArrayList<Integer>();
51
52             for (int i = 0; i < numberOfAlignments; i++) {
53                 numberOfGaps.add(hyperEdge.getCommonSequences().get(i).getGaps().size());
54             }
55
56             //find 2 components where common sequence have least number of gaps
57             int index1;
58             int index2;
59             if (numberOfGaps.get(0) < numberOfGaps.get(1)) {
60                 index1 = 0;
61                 index2 = 1;
62             } else {
63                 index1 = 1;
64                 index2 = 0;
65             }
66             for (int i = 2; i < numberOfAlignments; i++) {
67                 if (numberOfGaps.get(i) < numberOfGaps.get(index2)){
68                     if (numberOfGaps.get(i) <= numberOfGaps.get(index1)) {
69                         index2 = index1;
70                         index1 = i;
71                     } else {
72                         index2 = i;
73                     }
74                 }
75             }
76
77             //merge the two alignments
78             pairMerge(spannoid, hyperEdge, index1, index2);
79
80             numberOfGaps.remove(index2);
81             numberOfAlignments--;
82
83         }
84
85         pairMerge(spannoid, hyperEdge, 0, 1);
86
87         hyperEdge.getComponent(0).removeConnector(hyperEdge);
88         spannoid.remove(hyperEdge);
89     }
90 }
91
92
93 public class BasicCliqueMerger extends CliqueMerger {
94
95     public BasicCliqueMerger(PairMerger pairMerger) {
96         this.pairMerger = pairMerger;
97     }
98
99     public PairMerger getPairMerger() {
100        return pairMerger;
101    }
102
103    @Override
104    public void merge(Spannoid spannoid, HyperEdge hyperEdge) {
105        while (hyperEdge.getNumberOfComponents() > 1) {
106            //merge the first two alignments
107            pairMerge(spannoid, hyperEdge, 0, 1);
108        }
109
110        hyperEdge.getComponent(0).removeConnector(hyperEdge);
111        spannoid.remove(hyperEdge);
112    }
113 }

```

### A.2.5 Pair Mergers Package

This is a very important part of the program. The interface for a pair merger class simply has a merge method.

```

1 public interface PairMerger {
2     public void merge(Alignment xComp, Sequence xSeq, Alignment yComp, Sequence ySeq);
3 }

```

**SumGap Pair Merger Class** This is the implementation of the basic approach to merging two alignments, as described in Section 4.5.

```

1 public class SumGapPairMerger implements PairMerger {
2
3     public void merge(Alignment xComp, Sequence xSeq, Alignment yComp, Sequence ySeq) {
4         sumGap(xComp, xSeq, yComp, ySeq);
5     }
6
7     public void sumGap(Alignment align1, Sequence commonSequence1, Alignment align2, Sequence commonSequence2) {
8         ArrayList<Point> gaps1 = commonSequence1.getGaps();
9         ArrayList<Point> gaps2 = commonSequence2.getGaps();
10        int j = 0; int k = 0; int gap1Pos, gap2Pos, gap1Length, gap2Length;
11        boolean jdone = (j >= gaps1.size());
12        boolean kdone = (k >= gaps2.size());
13        while (!(kdone || jdone)) {
14            gap1Pos = gaps1.get(j).x;
15            gap2Pos = gaps2.get(k).x;
16            gap1Length = gaps1.get(j).y;
17            gap2Length = gaps2.get(k).y;
18            if (gap1Pos < gap2Pos) {
19                align2.insertGaps(gap1Pos, gap1Length);
20
21                for (int i = k; i < gaps2.size(); i++)
22                    gaps2.get(i).x += gap1Length;
23
24                j++;
25                if (j == gaps1.size())
26                    jdone = true;
27            } else if (gap1Pos > gap2Pos) {
28                align1.insertGaps(gap2Pos, gap2Length);
29
30                for (int i = j; i < gaps1.size(); i++)
31                    gaps1.get(i).x += gap2Length;
32                k++;
33                if (k == gaps2.size())
34                    kdone = true;
35            } else {
36                align1.insertGaps(gap2Pos + gap1Length, gap2Length); //insert new gaps at end of gap
37                align2.insertGaps(gap1Pos, gap1Length); //insert new gaps at start of gap
38
39                for (int i = j + 1; i < gaps1.size(); i++)
40                    gaps1.get(i).x += gap2Length;
41
42                for (int i = k + 1; i < gaps2.size(); i++)
43                    gaps2.get(i).x += gap1Length;
44
45                k++;
46                if (k == gaps2.size())
47                    kdone = true;
48
49                j++;
50                if (j == gaps1.size())
51                    jdone = true;
52            }
53        }
54
55        while (!jdone) {
56            gap1Pos = gaps1.get(j).x;
57            gap1Length = gaps1.get(j).y;
58            align2.insertGaps(gap1Pos, gap1Length);
59            for (int i = k; i < gaps2.size(); i++)
60                gaps2.get(i).x += gap1Length;
61            j++;
62            if (j == gaps1.size())
63                jdone = true;
64        }
65
66        while (!kdone) {
67            gap2Pos = gaps2.get(k).x;
68            gap2Length = gaps2.get(k).y;
69            align1.insertGaps(gap2Pos, gap2Length);
70            for (int i = j; i < gaps1.size(); i++)
71                gaps1.get(i).x += gap2Length;
72            k++;
73            if (k == gaps2.size())
74                kdone = true;
75        }
76
77        align1.append(align2);
78        align1.remove(commonSequence2);
79    }
80 }

```

**RealignGapsPairMerger Abstract Class** This class is a template for pairmergers where we want to note the common gaps in the common sequences and realign that region of the merged alignment in with a multiple alignment program, which could even be SpanAlign in a further implementation. Only the realignGaps method needs to be implemented for subclasses.

```

1 public abstract class RealignGapsPairMerger implements PairMerger {
2
3     @Override
4     public void merge(Alignment xComp, Sequence xSeq, Alignment yComp, Sequence ySeq) {

```

```

5         ArrayList<Point> commonGaps = maxGapReturningCommonGaps(xComp, xSeq, yComp, ySeq);
6
7         try {
8             realignGaps(xComp, commonGaps);
9         } catch (IOException e) {
10            e.printStackTrace();
11        } catch (InterruptedException e) {
12            e.printStackTrace();
13        }
14    }
15
16    protected abstract void realignGaps(Alignment alignment, ArrayList<Point> gapList) throws IOException,
17    InterruptedException;
18
19    public ArrayList<Point> maxGapReturningCommonGaps(Alignment align1,
20    Sequence commonSequence1, Alignment align2, Sequence commonSequence2) {
21
22        ArrayList<Point> gaps1 = commonSequence1.getGaps();
23        ArrayList<Point> gaps2 = commonSequence2.getGaps();
24        ArrayList<Point> commonGaps = new ArrayList<Point>();
25        int j = 0;
26        int k = 0;
27        int gap1Pos, gap2Pos, gap1Length, gap2Length;
28        boolean jdone = (j >= gaps1.size());
29        boolean kdone = (k >= gaps2.size());
30
31        while (!(kdone || jdone)) {
32            gap1Pos = gaps1.get(j).x;
33            gap2Pos = gaps2.get(k).x;
34            gap1Length = gaps1.get(j).y;
35            gap2Length = gaps2.get(k).y;
36            if (gap1Pos < gap2Pos) {
37                align2.insertGaps(gap1Pos, gap1Length);
38
39                for (int i = k; i < gaps2.size(); i++)
40                    gaps2.get(i).x += gap1Length;
41
42                j++;
43                if (j == gaps1.size())
44                    jdone = true;
45            } else if (gap1Pos > gap2Pos) {
46                align1.insertGaps(gap2Pos, gap2Length);
47
48                for (int i = j; i < gaps1.size(); i++)
49                    gaps1.get(i).x += gap2Length;
50                k++;
51                if (k == gaps2.size())
52                    kdone = true;
53            } else {
54                int maxGap = Math.max(gap1Length, gap2Length);
55                commonGaps.add(new Point(gaps1.get(j).x, maxGap));
56
57                align1.insertGaps(gap2Pos + gap1Length, maxGap - gap1Length);
58                align2.insertGaps(gap1Pos + gap2Length, maxGap - gap2Length);
59
60                for (int i = j; i < gaps1.size(); i++)
61                    gaps1.get(i).x += maxGap - gap1Length;
62
63                for (int i = k; i < gaps2.size(); i++)
64                    gaps2.get(i).x += maxGap - gap2Length;
65
66                k++;
67                if (k == gaps2.size())
68                    kdone = true;
69
70                j++;
71                if (j == gaps1.size())
72                    jdone = true;
73            }
74        }
75
76        while (!jdone) {
77            gap1Pos = gaps1.get(j).x;
78            gap1Length = gaps1.get(j).y;
79            align2.insertGaps(gap1Pos, gap1Length);
80            for (int i = k; i < gaps2.size(); i++)
81                gaps2.get(i).x += gap1Length;
82            j++;
83            if (j == gaps1.size())
84                jdone = true;
85        }
86
87        while (!kdone) {
88            gap2Pos = gaps2.get(k).x;
89            gap2Length = gaps2.get(k).y;
90            align1.insertGaps(gap2Pos, gap2Length);
91            for (int i = j; i < gaps1.size(); i++)
92                gaps1.get(i).x += gap2Length;
93            k++;
94            if (k == gaps2.size())
95                kdone = true;
96        }
97
98        align1.append(align2);
99        align1.remove(commonSequence2);
100
101
102
103

```

```

104         return commonGaps;
105     }
106 }
107

```

**StatAlignGapsPairMerger Class** The `realignGaps` method is implemented with `StatAlign`. The MCMC parameters are automated according to the function defined in Section 5.3.

```

1 public class StatAlignGapsPairMerger extends RealignGapsPairMerger {
2
3     @Override
4     protected void realignGaps(Alignment alignment, ArrayList<Point> gapList) throws IOException,
5         InterruptedException {
6         for (int i = 0; i < gapList.size(); i++) {
7
8             int a = gapList.get(i).x;
9             int b = a + gapList.get(i).y;
10
11             Alignment subAlignment = alignment.subAlignment(a, b);
12
13             ArrayList<Integer> noneEmptySeqs = subAlignment.getNoneEmptySequences();
14
15             if (noneEmptySeqs.size() > 1) { //if an alignment is necessary
16
17                 //remove the empty sequences from the subalignment
18                 for (int j = alignment.getNumberOfSequences() - 1; j >= 0; j--) {
19                     if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
20                         subAlignment.remove(j);
21                     }
22                 }
23
24                 ArrayList<Sequence> rawAlignment = subAlignment.rawAlignment();
25
26                 ReadWriteTools.writeToFASTAFileNoScores(rawAlignment, "tmpalign");
27
28                 int F = 200;
29                 Float A = 1.1F;
30                 Float B = 1.3F;
31                 int N = subAlignment.getNumberOfSequences();
32
33                 int totalOfAllLengths = 0;
34
35                 for (Sequence seq : rawAlignment) {
36                     totalOfAllLengths += seq.getLength();
37                 }
38
39                 int L = totalOfAllLengths/N;
40
41                 Double cycles = F * Math.pow(A, N) * Math.pow(L,B);
42                 int burnin = (int) (cycles / 2);
43                 int samplingRate = (int) (cycles / 200);
44
45                 System.out.println("Waiting for StatAlign on "
46                     + subAlignment.getNumberOfSequences() + " sequences of length "
47                     + subAlignment.getSequenceLength() + " mcmc=" + burnin + "," + Math.round(cycles) +
48                     ", " + samplingRate + "...");
49
50                 Process pr = Runtime.getRuntime().exec(
51                     "java -Xmx512m -jar statalign.jar "
52                     + "tmpalign.fasta -ot=Fasta -mcmc=" + burnin + "," + Math.round(cycles) +
53                     ", " + samplingRate);
54
55                 String line = null;
56
57                 BufferedReader input = new BufferedReader(
58                     new InputStreamReader(pr.getInputStream()));
59                 while ((line = input.readLine()) != null) {
60                     if (!line.isEmpty()) {
61                         if (!(line.charAt(0) == 'S' && line.charAt(1) == 'a')) {
62                             System.out.println(line);
63                         }
64                     }
65                 }
66                 input.close();
67                 pr.waitFor();
68
69                 System.out.println("Performed StatAlign.");
70
71                 subAlignment = ReadWriteTools.readFASTAFile(
72                     "tmpalign.fasta.mpd", subAlignment.getLikelihoodStrategy());
73
74                 String blankSeq = "";
75                 int newAlignmentLength = subAlignment.getSequenceLength();
76                 for (int j = 0; j < newAlignmentLength; j++) {
77                     blankSeq += "-";
78                 }
79
80                 ArrayList<Sequence> subAlignmentToInsert = new ArrayList<Sequence>();
81                 ArrayList<Float> scoresToInsert = subAlignment.getLikelihoods();
82
83                 //add the sequences to the sub alignment to insert back into alignment one by one
84

```

```

85         //adding blank sequences where necessary
86         // (StatAlign changes the order of the sequences)
87         for (int j = 0; j < alignment.getNumberOfSequences(); j++) {
88             if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
89                 subAlignmentToInsert.add(j, new Sequence(blankSeq, alignment
90                     .getSequence(j).getName()));
91             } else {
92                 subAlignmentToInsert.add(j, subAlignment.getSequence(
93                     alignment.getSequence(j).getName()));
94             }
95         }
96
97         alignment.replace(new Alignment(subAlignmentToInsert, scoresToInsert,
98             alignment.getLikelihoodStrategy()), a, b);
99
100         new File("tmpalign.fasta").delete();
101         new File("tmpalign.fasta.mpd").delete();
102
103         int offset = newAlignmentLength - gapList.get(i).y;
104
105         for (int j = i + 1; j < gapList.size(); j++) {
106             gapList.get(j).x += offset;
107         }
108     }
109 }
110 }
111 }
112 }

```

**MAFFT Align Gaps Pair Merger Class** This `realignGaps` method is implemented with MAFFT.

```

1 public class MAFFTAlignGapsPairMerger extends RealignGapsPairMerger {
2
3     @Override
4     protected void realignGaps(Alignment alignment, ArrayList<Point> gapList) throws IOException,
5         InterruptedException {
6         for (int i = 0; i < gapList.size(); i++) {
7
8             int a = gapList.get(i).x;
9             int b = a + gapList.get(i).y;
10
11             Alignment subAlignment = alignment.subAlignment(a, b);
12
13             ArrayList<Integer> noneEmptySeqs = subAlignment.getNoneEmptySequences();
14
15             if (noneEmptySeqs.size() > 1) { //if an alignment is necessary
16
17                 //remove the empty sequences from the subalignment
18                 for (int j = alignment.getNumberOfSequences() - 1; j >= 0; j--) {
19                     if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
20                         subAlignment.remove(j);
21                     }
22                 }
23
24                 ArrayList<Sequence> rawAlignment = subAlignment.rawAlignment();
25
26                 ReadWriteTools.writeToFASTAFileNoScores(rawAlignment, "tmpalign");
27
28                 System.out.println("Waiting for MAFFT on "
29                     + subAlignment.getNumberOfSequences() + " sequences of length "
30                     + subAlignment.getSequenceLength());
31
32                 Process pr = Runtime.getRuntime().exec(
33                     "mafft --auto tmpalign.fasta > tmprealign.fasta");
34
35                 String line = null;
36
37                 BufferedReader input = new BufferedReader(
38                     new InputStreamReader(pr.getInputStream()));
39                 while ((line = input.readLine()) != null) {
40                     System.out.println(line);
41                 }
42                 input.close();
43                 pr.waitFor();
44
45                 System.out.println("Performed MAFFT.");
46
47                 System.exit(0);
48
49                 subAlignment = ReadWriteTools.readFASTAFile(
50                     "tmprealign.fasta", subAlignment.getLikelihoodStrategy());
51
52                 String blankSeq = "";
53                 int newAlignmentLength = subAlignment.getSequenceLength();
54                 for (int j = 0; j < newAlignmentLength; j++) {
55                     blankSeq += "-";
56                 }
57
58                 ArrayList<Sequence> subAlignmentToInsert = new ArrayList<Sequence>();
59                 ArrayList<Float> scoresToInsert = subAlignment.getLikelihoods();
60
61                 //add the sequences to the sub alignment to insert back into alignment one by one
62

```

```

63 //adding blank sequences where necessary
64 //(StatAlign changes the order of the sequences)
65 for (int j = 0; j < alignment.getNumberOfSequences(); j++) {
66     if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
67         subAlignmentToInsert.add(j, new Sequence(blankSeq, alignment
68             .getSequence(j).getName()));
69     } else {
70         subAlignmentToInsert.add(j, subAlignment.getSequence(alignment.getSequence(j).getName()));
71     }
72 }
73
74 alignment.replace(new Alignment(subAlignmentToInsert, scoresToInsert,
75     alignment.getLikelihoodStrategy()), a, b);
76
77 new File("tmpalign.fasta").delete();
78 new File("tmprealign.fasta").delete();
79
80 int offset = newAlignmentLength - gapList.get(i).y;
81
82 for (int j = i + 1; j < gapList.size(); j++) {
83     gapList.get(j).x += offset;
84 }
85 }
86 }
87 }
88 }

```

**StatAlign Low Likelihood Regions Pair Merger Class** This is the most advanced implementation of the pair merger interface. After doing a SumGap pair merge on the 2 alignments, we identify the common gap regions as with the above methods and attempt to realign them. If we do not get a new score higher than the likelihood threshold, we reject the new alignment and assume the residues correspond to independent insertions on the common sequence. We also identify columns with lowlikelihood below the likelihood threshold and try to realign these regions. If the new alignment does not have a better average likelihood, we reject the new alignment. This is what the public static `reAlignmentsAccepted` variable is for in the `App` class.

```

1 public class StatAlignLowLikelihoodPairMerger implements PairMerger {
2
3     public void merge(Alignment xComp, Sequence xSeq, Alignment yComp, Sequence ySeq) {
4
5         ArrayList<Point> commonGaps = sumGapReturningCommonGaps(xComp, xSeq, yComp, ySeq);
6
7         ArrayList<Point> regionList = xComp.lowLikelihoodRegionsPlusGaps(commonGaps);
8
9         try {
10             statAlignRegions(xComp, regionList);
11         } catch (IOException e) {
12             e.printStackTrace();
13         } catch (InterruptedException e) {
14             e.printStackTrace();
15         }
16     }
17
18     public ArrayList<Point> sumGapReturningCommonGaps(Alignment align1,
19         Sequence commonSequence1, Alignment align2, Sequence commonSequence2) {
20
21         ArrayList<Point> gaps1 = commonSequence1.getGaps();
22         ArrayList<Point> gaps2 = commonSequence2.getGaps();
23         ArrayList<Point> commonGaps = new ArrayList<Point>();
24         int j = 0;
25         int k = 0;
26         int gap1Pos, gap2Pos, gap1Length, gap2Length;
27         boolean jdone = (j >= gaps1.size());
28         boolean kdone = (k >= gaps2.size());
29
30         while (!(kdone || jdone)) {
31             gap1Pos = gaps1.get(j).x;
32             gap2Pos = gaps2.get(k).x;
33             gap1Length = gaps1.get(j).y;
34             gap2Length = gaps2.get(k).y;
35             if (gap1Pos < gap2Pos) {
36                 align2.insertGaps(gap1Pos, gap1Length);
37
38                 for (int i = k; i < gaps2.size(); i++)
39                     gaps2.get(i).x += gap1Length;
40
41                 j++;
42                 if (j == gaps1.size())
43                     jdone = true;
44             } else if (gap1Pos > gap2Pos) {
45                 align1.insertGaps(gap2Pos, gap2Length);
46
47                 for (int i = j; i < gaps1.size(); i++)
48                     gaps1.get(i).x += gap2Length;
49             }
50         }

```

```

51         k++;
52         if (k == gaps2.size ())
53             kdone = true;
54
55     } else {
56         int sumGap = gap1Length + gap2Length;
57
58         commonGaps.add(new Point(gaps1.get(j).x, sumGap));
59
60         align1.insertGaps(gap2Pos + gap1Length, gap2Length); //insert new gaps at end of gap
61         align2.insertGaps(gap1Pos, gap1Length); //insert new gaps at start of gap
62
63         for (int i = j + 1; i < gaps1.size (); i++)
64             gaps1.get(i).x += gap2Length;
65
66         for (int i = k + 1; i < gaps2.size (); i++)
67             gaps2.get(i).x += gap1Length;
68
69         k++;
70         if (k == gaps2.size ())
71             kdone = true;
72
73         j++;
74         if (j == gaps1.size ())
75             jdone = true;
76     }
77 }
78
79 while (!jdone) {
80     gap1Pos = gaps1.get(j).x;
81     gap1Length = gaps1.get(j).y;
82     align2.insertGaps(gap1Pos, gap1Length);
83     for (int i = k; i < gaps2.size (); i++)
84         gaps2.get(i).x += gap1Length;
85     j++;
86     if (j == gaps1.size ())
87         jdone = true;
88 }
89
90 while (!kdone) {
91     gap2Pos = gaps2.get(k).x;
92     gap2Length = gaps2.get(k).y;
93     align1.insertGaps(gap2Pos, gap2Length);
94     for (int i = j; i < gaps1.size (); i++)
95         gaps1.get(i).x += gap2Length;
96     k++;
97     if (k == gaps2.size ())
98         kdone = true;
99 }
100
101 align1.append(align2);
102 align1.remove(commonSequence2);
103 align1.increaseMergeCount ();
104
105 return commonGaps;
106 }
107
108 public void statAlignRegions(Alignment alignment, ArrayList<Point> regionList) throws IOException,
109     InterruptedException {
110     for (int i = 0; i < regionList.size (); i++) {
111         int a = regionList.get(i).x;
112         int b = a + regionList.get(i).y;
113
114         Alignment subAlignment = alignment.subAlignment(a, b);
115
116         ArrayList<Float> oldScores = subAlignment.getLikelihoods ();
117
118         ArrayList<Integer> noneEmptySeqs = subAlignment.getNoneEmptySequences ();
119
120         if (noneEmptySeqs.size () > 1) { //if realignment is necessary
121
122             //remove the empty sequences from the subalignment
123             for (int j = alignment.getNumberOfSequences () - 1; j >= 0; j--) {
124                 if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
125                     subAlignment.remove(j);
126                 }
127             }
128
129             ArrayList<Sequence> rawAlignment = subAlignment.rawAlignment ();
130
131             ReadWriteTools.writeToFASTAFileNoScores(rawAlignment, "tmpalign");
132
133             int F = 200;
134             float A = 1.1F;
135             float B = 1.3F;
136             int N = subAlignment.getNumberOfSequences ();
137
138             int totalOfAllLengths = 0;
139
140             for (Sequence seq : rawAlignment) {
141                 totalOfAllLengths += seq.getLength ();
142             }
143
144             int L = totalOfAllLengths / N;
145
146             double cycles = F * Math.pow(A, N) * Math.pow(L, B);
147             int burnin = (int) (cycles / 2);
148             int samplingRate = (int) (cycles / 200);
149

```

```

150 System.out.println("Waiting for StatAlign on "
151 + subAlignment.getNumberOfSequences() + " sequences of length "
152 + subAlignment.getSequenceLength() + " mcmc=" + burnin + "," +
153 + Math.round(cycles) + "," + samplingRate + "...");
154
155 Process pr = Runtime.getRuntime().exec(
156 "java -Xmx512m -jar statalign.jar "
157 + "tmpalign.fasta -ot=Fasta -mcmc=" + burnin + "," +
158 + Math.round(cycles) + "," + samplingRate);
159
160 String line = null;
161
162 BufferedReader input = new BufferedReader(
163     new InputStreamReader(pr.getInputStream()));
164 while ((line = input.readLine()) != null) {
165     if (!line.isEmpty()) {
166         if (!(line.charAt(0) == 'S' && line.charAt(1) == 'a')) {
167             System.out.println(line);
168         }
169     }
170 }
171 input.close();
172 pr.waitFor();
173
174 System.out.println("Performed StatAlign.");
175
176 subAlignment = ReadWriteTools.readFASTAFile(
177     "tmpalign.fasta.mpd", subAlignment.getLikelihoodStrategy());
178
179 ArrayList<Float> newScores = subAlignment.getLikelihoods();
180
181 float oldScore = 0;
182 float newScore = 0;
183
184 for (Float score : oldScores){
185     oldScore += score;
186 }
187 oldScore /= oldScores.size();
188
189 for (Float score : newScores){
190     newScore += score;
191 }
192 newScore /= newScores.size();
193
194 App.reAlignmentsAccepted.y += 1;
195 System.out.println("Old mean likelihood: " + oldScore);
196 System.out.println("New mean likelihood: " + newScore);
197 if (newScore > oldScore && !(oldScore == 0.0 && newScore <
198     alignment.getLikelihoodStrategy().threshold)) {
199     System.out.println("Realignment accepted.");
200     System.out.println();
201     App.reAlignmentsAccepted.x +=1;
202
203     String blankSeq = "";
204     int newAlignmentLength = subAlignment.getSequenceLength();
205     for (int j = 0; j < newAlignmentLength; j++) {
206         blankSeq += "-";
207     }
208
209     ArrayList<Sequence> subAlignmentToInsert = new ArrayList<Sequence>();
210
211     //add the sequences to the sub alignment to insert back into alignment one by one
212     //adding blank sequences where necessary
213     //(StatAlign changes the order of the sequences)
214     for (int j = 0; j < alignment.getNumberOfSequences(); j++) {
215         if (!noneEmptySeqs.contains(Integer.valueOf(j))) {
216             subAlignmentToInsert.add(j, new Sequence(blankSeq, alignment
217                 .getSequence(j).getName()));
218         } else {
219             subAlignmentToInsert.add(j, subAlignment.getSequence(alignment.getSequence(j).getName()));
220         }
221     }
222
223     alignment.replace(new Alignment(subAlignmentToInsert, newScores,
224         alignment.getLikelihoodStrategy()), a, b);
225
226     int offset = newAlignmentLength - regionList.get(i).y;
227
228     for (int j = i + 1; j < regionList.size(); j++) {
229         regionList.get(j).x += offset;
230     }
231
232     new File("tmpalign.fasta").delete();
233     new File("tmpalign.fasta.mpd").delete();
234     new File("tmpalign.fasta.log").delete();
235     new File("tmpalign.fasta.ll").delete();
236
237 } else {
238     System.out.println("Realignment rejected.");
239     System.out.println();
240 }
241 }
242 }
243 }
244 }
245 }
246 }
247 }

```

### A.2.6 Likelihood Strategies Package

A likelihood strategy contains the threshold with which to class a column as being of low likelihood and a method to combine the likelihood score of two columns when pairmerged.

```

public abstract class LikelihoodStrategy {
    public Float threshold = 0.5F;

    public LikelihoodStrategy(Float threshold) {
        this.threshold = threshold;
    }

    public abstract void combine(Float likelihood1, Float likelihood2);

    public void combine(ArrayList<Float> likelihoods1, ArrayList<Float> likelihoods2) {
        if (likelihoods1.size() == likelihoods2.size()) {
            for (int i = 0; i < likelihoods1.size(); i++)
                combine(likelihoods1.get(i), likelihoods2.get(i));
        } else {
            System.out.println("Illegal likelihood lists as arguments for likelihood combiner");
        }
    }
}

public class MinimumLikelihoodStrategy extends LikelihoodStrategy {
    public MinimumLikelihoodStrategy(Float threshold) {
        super(threshold);
    }
    public void combine(Float likelihood1, Float likelihood2) {
        likelihood1 = Math.min(likelihood1, likelihood2);
    }
}

public class ProductLikelihoodStrategy extends LikelihoodStrategy {
    public ProductLikelihoodStrategy(Float threshold) {
        super(threshold);
    }
    public void combine(Float likelihood1, Float likelihood2) {
        likelihood1 *= likelihood2;
    }
}

```

### A.2.7 Edge Length Strategies Package

In the current program, this is not used, but it has potential.

```

public interface EdgeLengthStrategy {
    public int calculateLength(Alignment alignment1, Alignment alignment2);
}

public class NullEdgeLengthStrategy implements EdgeLengthStrategy {
    public int calculateLength(Alignment alignment1, Alignment alignment2) {
        return 1;
    }
}

```

## References

- [1] J. BŁAZEWICZ, P. FORMANOWICZ, AND P. WOJCIECHOWSKI, *Some Remarks on Evaluating the Quality of the Multiple Sequence Alignment based on the BALiBASE Benchmark*, *Int. J. Appl. Math. Comput. Sci.*, Vol. 19, No. 4, 675-678, (2009).
- [2] R. DURBIN, S. EDDY, A. KROGH, AND G. MITCHISON, *Biological Sequence Analysis. Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, 2008.
- [3] R. LYNGSØ, *Statistical Alignment via  $k$ -Restricted Steiner Trees*. Retrieved at [http://www.stats.ox.ac.uk/\\_\\_data/assets/file/0018/3492/spannoids.pdf](http://www.stats.ox.ac.uk/__data/assets/file/0018/3492/spannoids.pdf), 2009.
- [4] I. MIKLÓS, A. NOVÁK, R. LYNGSØ, AND J. HEIN, *StatAlign: An Extendable Software Package for Joint Bayesian Estimation of Alignments and Evolutionary Trees*, Vol. 24 no. 20 2008, pages 2403-2404, (2008).
- [5] I. MIKLÓS, A. NOVÁK, R. SATIJA, R. LYNGSØ, AND J. HEIN, *Stochastic Models of Sequence Evolution including Insertion-Deletion events*. 2008.
- [6] J. L. THORNE, K. HIROSHIA, AND J. FELSTENSTEIN, *An Evolutionary Model for Maximum Likelihood Alignment of DNA Sequences*, *J Mol Evol* (1991) 33:114-124, (1991).
- [7] S. YAMADA, O. GOTOH, AND H. YAMANA, *Improvement in Accuracy of Multiple Sequence Alignment using Novel Group-to-Group Sequence Alignment Algorithm with Piecewise Linear Gap Cost*, *BMC Bioinformatics* 2006, 7:524, (2006).