Chapter 3

Reconstructing DNA

3.1 Introduction

In this chapter we consider several approaches to reconstruct strings based on information about their substrings.

Fragment assembly, Section 3.2, deals with the reconstruction of a string given a subset of its substrings (usually varying in length). The presentation is based on Setubal&Meidanis [12], Chapter 4: Fragment assembly of DNA.

Sequencing by hybridization, Section 3.4, uses a technique that determines strings that are characterized by a set of (all) substrings of fixed length.

Physical mapping, Section 3.5, computes the relative positions of markers on a string given certain (unordered) sets of markers that occur together on segments of the string. We present a solution using PQ-trees, following Shamir [11], Algorithms in Molecular Biology, Lecture 9, Physical Mapping.

3.2 Fragment Assembly

The problem of *fragment assembly* originates in the so-called shotgun method, where a given piece of DNA (or rather many identical copies of it) is broken into several smaller segments, that are all sequenced. The goal is to reconstruct the original DNA string based on the segments.

Hence we basically have the following problem: Given a set of strings $\mathcal{F}$ ('fragments'), construct the 'best' string that contains (as a consecutive substring) each of the strings in $\mathcal{F}$. A string with that property is called a (common) superstring of $\mathcal{F}$. Note that concatenation of the strings in $\mathcal{F}$ does the job. It defines a superstring, but one which usually is not considered to be the best possibility. Here we want to find a superstring of minimal length, called the *shortest common superstring* (SCS) of $\mathcal{F}$. The (mathematical) statement of the

problem is not precisely the (biological) problem we have to solve in practice, as we shall see in the next section.

For technical reasons it is assumed that $\mathcal{F}$ is substring free, i.e., no string in $\mathcal{F}$ occurs as substring in another element of $\mathcal{F}$.

A mathematical tool to model the problem is a graph that represents the overlap between the strings in the set $\mathcal{F}$. The *overlap graph* of $\mathcal{F}$ contains a node for each string x in $\mathcal{F}$. For each pair of strings (nodes) x, y we draw an edge from x to y labelled by the length of the maximal overlap of x and y. Here an overlap is a string w such that $x = x_1 w$ and $y = w y_1$. Usually edges with length 0 are omitted.

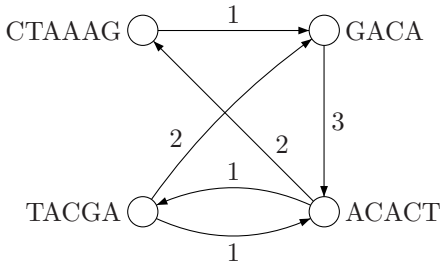
Sometimes all possible overlaps are considered, and we introduce edges for each of them. We then obtain the overlap *multi*-graph, which has parallel edges.

As we want to obtain a string x that contains each of the original strings from $\mathcal{F}$ we have to include each node of the overlap graph. As $\mathcal{F}$ is substring free its elements have a well defined order in x, i.e., if one substring starts before another it must also end first. This ordering corresponds to an ordering of the nodes of the overlap graph. Additionally overlap between the consecutive substrings as they occur in x define edges in the overlap multi-graph, and hence string x defines a Hamiltonian path (which visits all nodes). Vice versa, every Hamiltonian path determines a superstring.

The length of the common superstring defined by the Hamiltonian path equals the total length of the strings (which is a fixed by the problem, hence constant for all Hamiltonian paths) minus the overlap between the strings, which is the weight of the edges. Hence, a solution to the Shortest Common Superstring Problem translates into finding a Hamiltonian path of *maximum* weight.

Note this is an instance of the Traveling Salesman Problem, which in its general form is NP complete. Even the existence of a Hamiltonian path in a graph is an NP complete problem (for general graphs). Although the SCSP defines only specific forms of the TSP (defined by overlap) this is a strong indication that there are no efficient algorithms that solve the problem.

3.1 Example. Given the set of four segments $\mathcal{F} = \{$ TACGA, ACACT, CTAAAG, GACA $\}$ we build the following overlap graph (edges of zero weight omitted). E.g., the overlap of ACACT and CTAAAG is obviously CT with length 2.



The Hamilton path TACG(A)CA(CT)AAA(G)ACA leads to a superstring of length 16 (total length 20, minus 4 overlap).

TACGA- - - - - - - - - - -
- - - - ACACT- - - - - - -
- - - - - - - CTAAAG- - -
- - - - - - - - - - - GACA

It turns out that the set of largest edges can be arranged into a Hamiltonian path, and hence is a Hamiltonian path of maximal weigth. It leads to the length 13 superstring TACGACACTAAAG, which is the shortest 'reconstruction' of the original sequence. It corresponds to the following scheme.

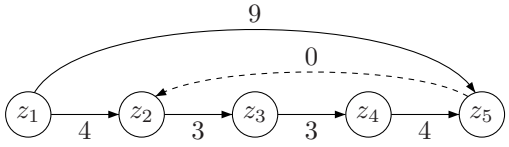
TACGA- - - - - - - -
- - - GACA- - - - - -
- - - - ACACT- - - -
- - - - - - - CTAAAG

Linkage and Coverage. The *shortest* common superstring is not always the best choice from a biological point of view. It may happen that the SCS consists of several concatenated 'contigs', pieces that have no overlap. Given an Hamiltonian path, its minimum weight edge determines the smallest overlap between consecutive substrings, called the *linkage* of the solution. In practice we want a superstring with good linkage, which means we want to avoid edges of low weight (and in particular those of weight 0).

Another measure for realistic solutions as opposed to mathematical optimal ones is *coverage*. For each position in the reconstructed string the number of associated segments should be of roughly equal size. When a segment is covered by a relatively large number of segments it may possibly be a repeat. We refer to the next section for this and additional biological considerations.

3.2 Example. Consider the strings
z_1 = AGTATTGGCAATC,
z_2 = AATCGATG,
z_3 = ATGCAAACCT,
z_4 = CCTTTTGG, and
z_5 = TTGGCAATCACT, of total length 51.

The relation between these strings is represented by the following overlap graph. Recall that all edges omitted here have weight 0, representing a pair of strings without overlap. One edge of weight 0 is given (dashed). It is used as part of a Hamiltonian path below.



Assuming edges of length 0 are present in the graph, the Hamiltonian path of maximal weight (length) 15 equals $z_1 z_5 z_2 z_3 z_4$. This leads to a superstring of length $51 - 15 = 36$, given in the final row of the following diagram. Note how the SCS consists of two contigs, parts without overlap, corresponding to the edge of weight 0 used on the path.

```
AGTATTGGCAATC---AATCGATG------------
----------------------ATGCAAACCT-----
----TTGGCAATCACT------------CCTTTTGG
AGTATTGGCAATCACTAATCGATGCAAACCTTTTGG
```

Another Hamiltonian path equals $z_1 z_2 z_3 z_4 z_5$ and has weight 14. It defines a superstring of length 37 where the consecutive fragments have overlap that is at least 3.

```
AGTATTGGCAATC--------CCTTTTGG--------
---------AATCGATG--------TTGGCAATCACT
---------------ATGCAAACCT------------
AGTATTGGCAATCGATGCAAACCTTTTGGCAATCACT
```

Greedy Algorithm. A greedy algorithm for the Shortest Common Superstring problem is easily conceived: repeatedly find the two strings with the largest overlap, and replace them with their shortest superstring. This corresponds to repeatedly choosing an edge with maximal weight in the overlap graph (and 'joining' the two adjacent nodes). This is an approximating algorithm. Its solution can be proved to be of length at most 4 times the optimal length, using rather technical arguments regarding the combinatorial structure of strings. The factor can be improved to 2.75 if some heuristics are added to the greedy approach. A counter example (see below, after the example) shows that the greedy algorithm may miss the optimal solution by a factor of 2. In fact it is conjectured that 2 is indeed the true bound, but this fact has not yet been proven.

3.3 Example. An example of the greedy algorithm: we start with the four strings TCAGT, CATCAG, GTG and GCA. The two most overlapping ones are CATCAG and TCAGT; they are replaced with CATCAGT, leaving us with CATCAGT, GTG and GCA.

Both GTG and GCA have a 2 base overlap with CATCAGT. Choose GTG (say), giving CATCAGTG and GCA. The final solution is GCATCAGTG, which happens to be optimal.

Another example of the greedy algorithm: we start with the three strings GCC, ATGC and TGCAT. The two most overlapping ones are ATGC and TGCAT; they are replaced with ATGCAT, leaving us with ATGCAT and GCC. These two strings have no overlap, so the final solution is their concatenation: either ATGCATGCC or GCCATGCAT, both of length 9. The optimal solution, TGCATGCC, has length 8.

Note how Example 3.2 shows that the greedy approach may give the optimal solution, where in fact that is not the most relevant one in biological terms.

An example that displays a factor 2 between the optimal solution and the one found by a greedy approach is the following. Starting from $\{C(AT)^k$, $(TA)^k$, $(AT)^kG\}$ for fixed $k \geq 1$, the algorithm outputs $C(AT)^kG(TA)^k$ of length $4k+2$, whereas the optimal string $C(AT)^{k+1}G$ has length $2k+4$. These solutions differ by a factor $\frac{4k+2}{2k+4}$ which is close to 2 for large k.

3.3 Biological Complications

As we have seen, the abstraction of Fragment Assembly to Shortest Common Superstring leads to a problem (travelling salesman) that is NP complete, hence has no efficient exact solution. Unfortunately the situation is even more complicated by the fact that in the practical the problem of fragment assembly has to deal with further details that are not captured by the SCS reformulation.

Base errors. We start with the strings ACCGT, CGTGC, TTAC and TACCG over the usual four-base alphabet {A, C, G, T}. The usual solution is to assemble them according the following layout (left). The sequence below the line perfectly matches the segments.

Usually in experiments there will be errors in reading the segments. The simplest ones are base call errors: base substitutions, insertions and deletions. As first example replace fragment TACCG by T**G**CCG, where the A has erronously been read as G. If our matching process allows for such errors, we can use majority voting to produce a *consensus* string (middle).

Also when CGTGC has been incorrectly reported as C**A**GTGC (inserting an extra A) we should be able to obtain the right consensus string (discarding the gap, right).

TTAC- - - - -	TTAC- - - - -	TTAC- - - - - -
-TACCG- - -	-TGCCG- - -	-TGCC-G- - -
- -ACCGT- -	- -ACCGT- -	- -ACC-GT- -
- - - -CGTGC	- - - -CGTGC	- - - -CAGTGC
TTACCGTGC	TTACCGTGC	TTACC-GTGC

This means that we should not try to find exact overlaps, but rather the kind of overlaps that are reported by alignment, allowing for a small number of errors.

Orientation. Experiments for reconstructing DNA rarely consider single stranded DNA. This means that our fragments can come from either strand, and can have two orientations, forward or backward (in reverse complement, also swapping A-T and C-G) In that case CGTAGT should alternatively be considered as synonym for ACTACG.

Thus for the set { CACGT, ACTACG, GTACT, ACTGA} a possible reconstruction is as follows.

```
CACGT    ⟶   CACGT- - - - - - - -
ACTACG   ⟵   - - CGTAGT- - - - -
GTACT    ⟵   - - - - - AGTAC- - -
ACTGA    ⟶   - - - - - - - - ACTGA
              CACGTAGTACTGA
```

Note that we suddenly have around 2^n possible combinations to consider for a set of n fragments.

Repeats. Repeated regions (or repeats for short) are sequences that occur two or more times in the original molecule. This causes many algorithmic complications, in particular if the repeats are long. We mention several issues.

• When fragments are totally contained in a repeat they can be placed in either one of them., see Example 3.4 below.

• If a relatively long repeat X occurs, i.e., when X is longer than the given fragments, the target $\alpha X\beta X\gamma X\delta$ can be assembled as $\alpha X\gamma X\beta X\delta$, swapping intermediate β and γ, which has the same substrings up to the length we are interested in. Similarly, when trying to reconstruct $\alpha X\beta Y\gamma X\delta Y\varepsilon$ one might end up with $\alpha X\delta Y\gamma X\beta Y\varepsilon$.

• An inverted repeat is a segment that occurs at least once on each strand of DNA. Thus the single stranded representation is of the form $\alpha X\beta X'\gamma$ where X' is the inversion of X. When X is large enough this can be reconstructed as $\alpha X\beta' X'\gamma$, where α' is the inversion of α.

An example of an inverted repeat is AACTG / CAGTT in the following sequence:

```
- - - TGCCTA- - - - - - - -        - - - - - - - - TAGGCA- - -
- - - - - - - TAGCTCA- - -        - - - TGAGCTA- - - - - - -
AACTGCCTAGCTCAGTT                 AACTGAGCTAGGCAGTT
```

Indeed, as we have seen already, repeats can give unwanted results especially when we stick to the formulation as shortest common superstring. Suppose the target looks like $\alpha X\beta X\gamma$ with some large repeat X. The SCS model could give a shortest solution $\alpha X\beta X'\gamma$ with X' shorter than X, where all fragments totally contained in the rightmost X are moved to the leftmost X, while X' consists of unlinked parts.

3.4 Example. Consider a string as below (left), that is the source of substrings TGAGT, GTACCG, ACGAT, GATAG, AGTACC, CCTT. Note how AGTACC occurs twice. Moving segment AGTACC to the left we obtain a (slightly) shorter superstring (and zero linkage right). We miss the symbol A (at the position indicated by the dot).

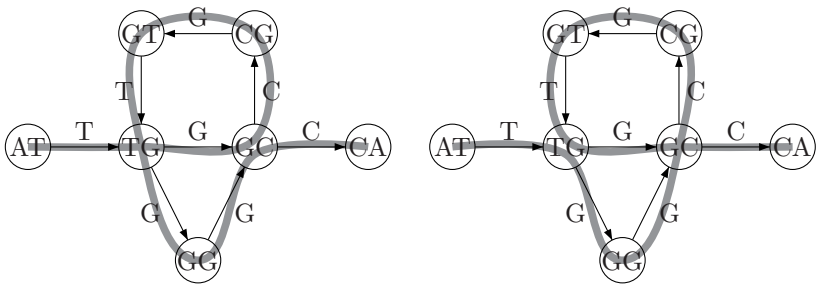
```
TGAGT- - - GATAGT- - - - -        TGAGT- - - GATAGT - - - -
- - - GTACCG- - AGTACC- -         - - - GTACCG- - - - - - - - -
- - - - - - CCGAT- - - - CCTT     - - - - - - CCGAT- - - CCTT
AGAGTACCGATAGTACCTT               - - AGTACC- - - - - - - - - -
                                  AGAGTACCGATAGT·CCTT
```

3.4 Sequencing by Hybridization.

Another technique to reconstruct DNA from knowledge about fragments is *sequencing by hybridization* (SBH). This approach uses a DNA-array, containing many small pieces of DNA of fixed length (say, all $4^6 = 212 = 4096$ possible strings of length 6, or some clever selection of these strings). Using the array one determines whether or not each of these strings occurs as a substring of our original piece of DNA, by observing the hybridizations that occur. This information is used for reconstruction. We try to find a (shortest) string that (exactly) has these substrings.

Instead of looking for Hamiltonian paths in the overlap graph, we now focus on Eulerian paths (that traverse all edges) in another graph which is in a sense dual to the overlap graph as here substrings are represented by edges rather than by vertices. Finding Eulerian paths is a relatively simple task compared to the complexity of finding Hamiltonian paths . Suppose we consider substrings of length ℓ, and get a set F of fragments, consisting of strings of length ℓ. We build a graph for F as follows. Vertices correspond to all $(\ell-1)$-tuples, edges correspond to the fragments in F. Fragment $a\alpha b$ (with a, b letters) forms an edge from prefix node $a\alpha$ to suffix node αb.

3.5 Example. Let $\ell = 3$. With fragments ATG, TGG, TGC, GTG, GGC, GCA, GCG and CGT, we build the following graph that contains each fragment as an edge. Thus, ATG forms an edge between nodes AT and TG (here labelled by overlap T).



We find two solutions, each corresponding to an Euler path in the graph. One for path AT·TG·GC·CG·GT·TG·GG·GC·CA with string ATGCGTGGCA, and the other for path AT·TG·GG·GC·CG·GT·TG·GC·CA with string ATGGCGTGCA.

3.5 Physical Mapping

A *physical map* contains the positions of a set of markers along a strand of DNA. Usually these markers are specific short and well-defined pieces of DNA. These

	E	B	F	C	A	G	D
1			1	1	1	1	
2				1	1		
3		1	1				
4						1	1
5	1	1					
6			1	1	1		

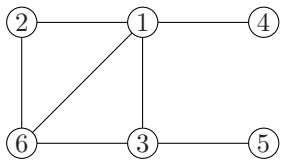


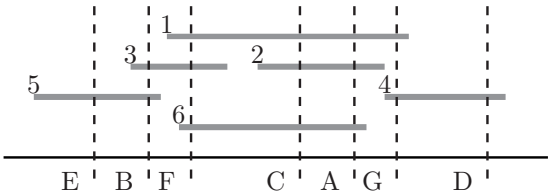
Figure 3.1: Adjacent 1's for Example 3.6, and corresponding interval graph.

small segments are for example the binding sites of restriction enzymes, or small segments known to occur only once in the DNA (sequence tagged sites).

Here we consider an algorithm for *hybridization mapping*, where the presence of markers has been obtained by hybridization of short pieces of DNA ('probes') on overlapping segments of DNA ('clones').

3.6 Example. Cosider seven probes A,B, ... G on a DNA molecule, and six clones 1,2,... 6 distributed as in the picture below.

The matrix represents the distribution of the probes over the clones. The ordering of the probes is not directly obvious from the matrix. For the clones we can only observe where they overlap, as indicated by a common probe.



	A	B	C	D	E	F	G
1	1		1			1	1
2	1		1				
3		1				1	
4				1			1
5		1			1		
6	1		1			1	

The information that is obtained by hybridization covers only the set of probes for each clone, i.e., the rows in the matrix. In this example $C_1 = \{A,C,F,G\}$, $C_2 = \{A,C\}$, $C_3 = \{B,F\}$, $C_4 = \{D,G\}$, $C_5 = \{B,E\}$, $C_6 = \{A,C,F\}$.

The algorithmic task we try to solve is to propose a possible ordering of the probes on the original sequence of DNA based on the information of which probes occur together on a clone. Note that the relative ordering of probes on each clone is not known.

Consecutive ones property. Clones contain consecutive probes. This means that if we order the columns (probes) according to their order on the DNA the probes for each clone must be adjacent. Thus the 1's in each row fill consecutive columns.

Technically a matrix like this can be used to define an *interval graph*: nodes correspond to intervals (here the clones) and edges between them correspond to the intersection of two intervals (here the presence of a common clone). Interval

graphs have been extensively studied, and several algorithms have been proposed to recognize them.

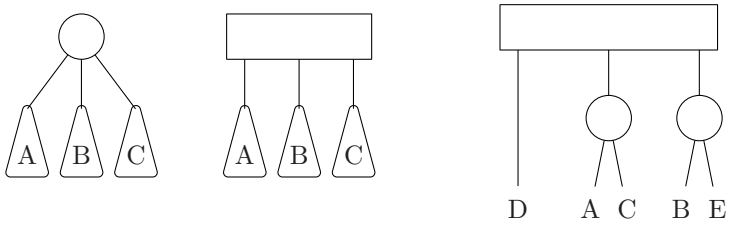
Some approaches based on interval graphs deal with the columns in the matrix. These correspond to a probe in the application, while in the graph they define a *clique*, a set of nodes that are fully connected in the graph.

3.6 PQ-trees

Classically in computer science a tree defines a linear order on its leaves: the linear order on the children of each node is inherited downwards in the tree. The tree data structure used to solve the consecutive ones problem defines not just a single order but rather a set of permutations on the leaves. When the leaves are labelled by probes, the permutations represented correspond to those permutations that are consistent with a set of clones, i.e., those orderings where each of the clones induces consecutive probes.

The trees used are called *PQ-trees* [2], having two types of nodes. *P-nodes* (depicted as circles) define permutations on their children, whereas *Q-nodes* (depicted as rectangles) define a linear order on their children (either from left to right or reversely).

3.7 Example. The P-node (left) defines the set of permutations { ABC, ACB, BAC, BCA, CAB, CBA }, the Q-node (middle) defines the two linear orders {ABC, CBA }.



Given the two clones {A,C,D} and {A,B,C,E} we observe that the intersection {A,C} must be consecutive, while flanked on one side by D and on the other side by B,E. All permutations of A,B,C,D,E that satisfy these restrictions are represented by the PQ tree given above (right): DACBE, DACEB, DCABE, DCAEB, BEACD, EBACD, BECAD, EBCAD.

Rearranging the children of a P-node will lead to a tree that defines the same set of permutations on the leaves. Similarly reversing the children of a Q-node will not change the permutations. Hence trees obtained by such operations are considered equivalent (or even equal, depending on your point of view).

We assume that every P-node has at least two cildren, whereas every Q-node usually has at least three children (on two elements the permutations are just the two linear orders).

Marking the tree. Starting with a tree that consists of a single P-node with all probes as leaves, thus representing all permutations, the clones are considered one by one. For each clone the structure of the tree is restricted so that its permutations also reflect the new clone. If this is possible the clone is called *consistent* with the tree. If the clones completely fix the relative order of the probes we obtain a tree consisting of a single Q-node.

Consider a clone C. Start by marking all leaves that belong to C. We then work upwards in the tree. First, when a node is handled its children are rearranged in such a way that the marked leaves are all consecutive. If this is not possible then the clone is not consistent with the tree. Then we may change the node to locally fix this new requirement.

In order to describe the rules to do this we need some terminology. A node is called *full* (*empty*) if all (no) leaves under it are marked. If a node is neither full nor empty it is *mixed*, in the algorithm below it is assumed that mixed nodes are Q-nodes (and even may have only two children). Note that it is impossible that a node has three mixed children (assuming the clone is consistent with the tree).

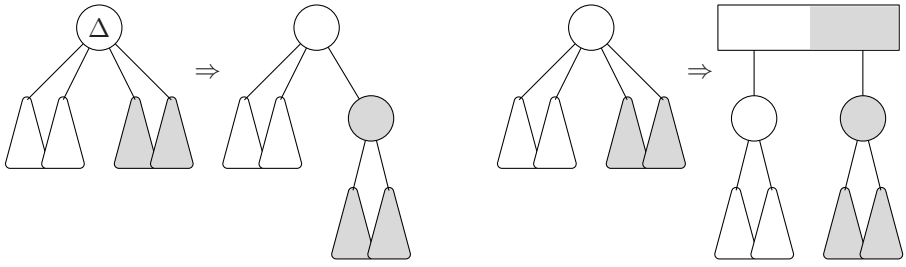
The rules to transform nodes are presented below. There are separate rules for P-nodes and Q-nodes, while they also depend on the children of the node: their type (P or Q), and whether they are empty, full or mixed. A special position in the tree is the lowest node that has marked nodes in its subtree, which is called the *top*. In fact the top is the only node that can have two mixed children. Only nodes below the top need to be investigated, bottom-up. Some rules have a special form for the top, to stress this the node is marked by symbol Δ.

Transformation rules. Most rules actually have several special cases, not depicted here. For instance rules may assume that there is more than one subtree of the same colour which are then joined under a P-node. When the rule is applied for only a single node of the proper colour then the P-node above the single child can be omitted.

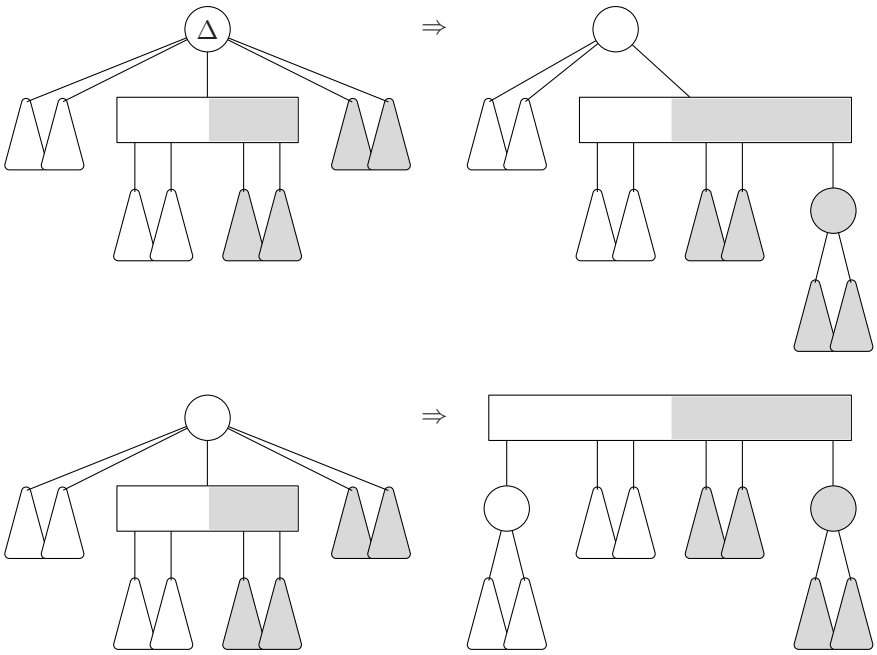
In the diagrams full nodes are shaded gray.

P1. If the children of a P-node are either all empty or all full, then the node itself is empty (full, respectively) and is not changed.

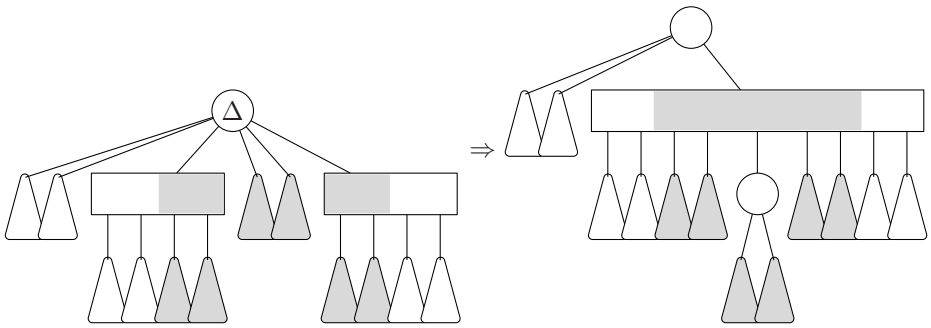
P2&3. If a P-node has both empty and full children, the full children must be forced next to one another. This rule has a special form for the top node 'Δ'.



P4&5. If a P-node has a mixed child, the full children of the node must be forced adjacent to the full children of that mixed child. Again we have a special form for the top node.

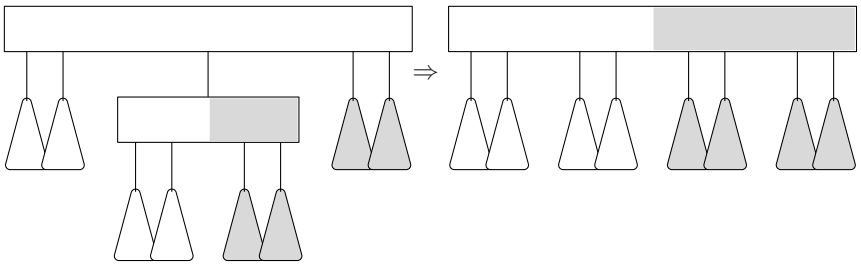


P6. P-node with two mixed children, only for the top node. Generally holds for *empty-full-empty* patterns.



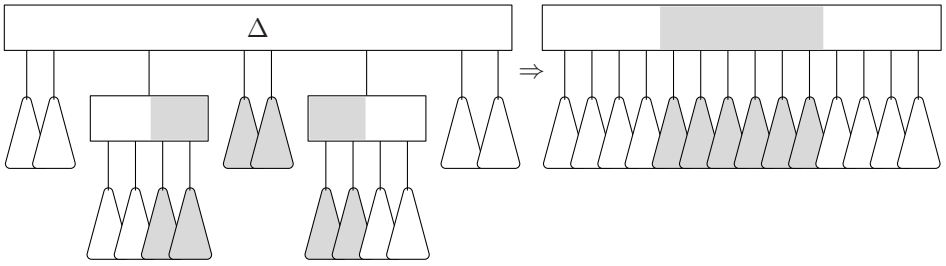
Q1. If the children of a Q-node are either all empty or all full, then the node itself is empty (full, respectively) and is not changed.

Q2. Rule for Q-nodes with a single mixed child, *empty-full* pattern. Applicable when empty and full children at different sides of the mixed child.



Q2^a If a Q-node has both full and empty children the node itself is mixed, and is not changed. The transformation is aborted if one or more of the empty children is between full ones, in that case the clone is not consistent. (this can be seen as a special case of Q2, i.e., without mixed child.)

Q3. P-node with two mixed children, *empty-full-empty* pattern. Is only applicable for the top node. The rule is rather general, in particular one or both of the mixed nodes may be missing.

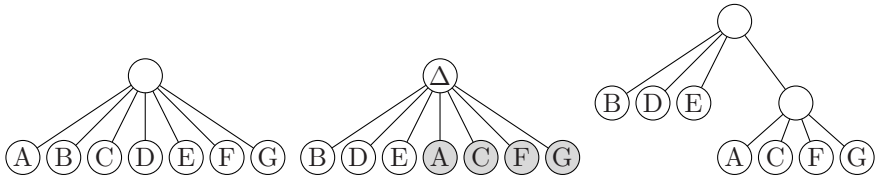


We end with an extensive example that illustrates the concrete application of the above rules.

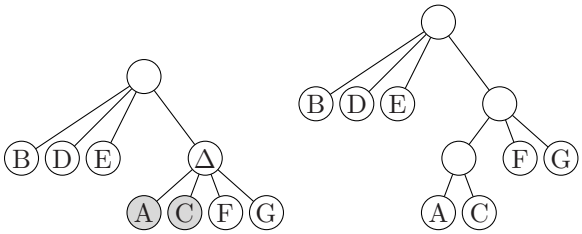
3.8 Example. We reconstruct the order of seven probes given clones {A,C,F,G}, {A,C}, {B,F}, and {D,G}, as in the previous Example 3.6.

We start with an arbitrary permutation of A,B,C,D,E,F,G.

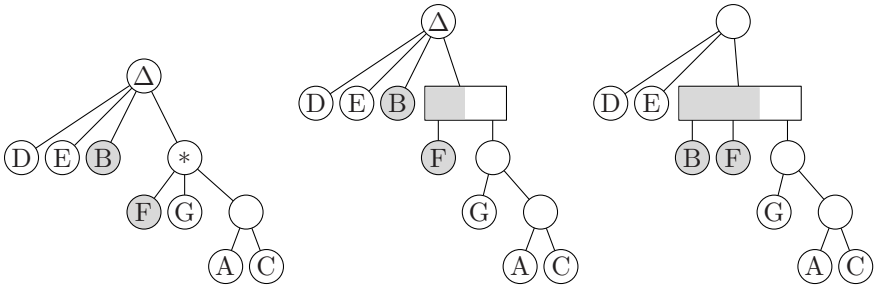
Given clone {A,C,F,G} we colour the leaves with corresponding probes, and reorder such that coloured leaves form a contiguous segment. We apply rule P2 to the root, which also is the top node Δ.



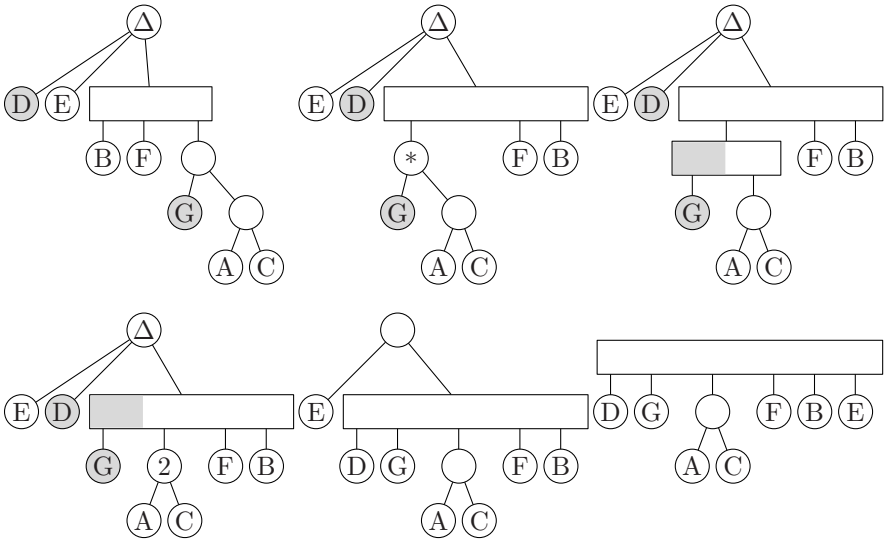
Given clone {A,C} we colour the leaves with corresponding probes, and reorder such that coloured leaves are next to one another. Again, we apply rule P2 to top node Δ.



Then we consider clone {B,F}. Colour the leaves with corresponding probes, and reorder such that coloured leaves are next to one another. At node $*$ we apply rule P3, and obtain a mixed Q-node. Then at Δ we apply rule P4 to join the coloured nodes under the Q-node.



Now we consider clone {D,G}. Colour the leaves with corresponding probes, and reorder such that coloured leaves are next to one another. As above, at node $*$ we apply rule P3, and obtain a mixed Q-node. This node is joined to its parent Q-node using rule Q2. Then at Δ we apply rule P4 to join the coloured nodes under the Q-node.



Last picture above: Adding clone {B,E} follows the same scheme. After colour-

ing and rearranging apply rule P4 at the root (which is Δ). Adding clone {A,C,F} will not change the tree. This involves an application of rule Q3 (in a variant where the empty-full-empty pattern does not include mixed nodes).

Bibliography

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D.J. Lipman (1990). Basic local alignment search tool. Journal of Molecular Biology 215 (3): 403–410. doi:10.1006/jmbi.1990.9999
▷ alignment heuristics, BLAST, Section 1.5

[2] K.S. Booth, G.S. Lueker: Testing for the consecutive ones property, interval graphs, and planarity using PQ-tree algorithms, Journal of Computational Systems Science, Vol. 13 (1976), pp. 335-379.
▷ physical mapping, Section 3.5

[3] Fitch and Margoliash, Construction of Phylogenetic Trees, Science Vol. 155, 20 Jan. 1967.

[4] A.P. Gultyaev, Computational Molecular Biology, Application-oriented view, Leiden University, 2009.

[5] D.S. Hirschberg. Algorithms for the Longest Common Subsequence Problem, Journal of the ACM, 24 (1977) 664–675.
▷ linear space alignment, Section 1.4

[6] V.I. Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals. Soviet Physics Doklady 10 (1966):707–710.
▷ global alignment, edit distance, Section 1.2

[7] D.J. Lipman, W.R. Pearson, Rapid and sensitive protein similarity searches. Science. 1985 Mar 22;227(4693):1435-41.
▷ alignment heuristics, FASTA, Section 1.5

[8] S.B. Needleman, C.D. Wunsch. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol 48 (3): 443–53. doi:10.1016/0022-2836(70)90057-4
▷ global alignment, Section 1.2

[9] N. Saitou and M. Nei, (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol. Biol. Evol. 4(4):406-425
▷ phylogeny, unrooted trees, Section 2.5

[10] D. Sankoff (1975). Minimal mutation trees of sequences. SIAM Journal of Applied Mathematics 28: 35-42.
▷ character based, small parsimony, Sankoff algorithm, Section 2.3

[11] R. Shamir, Algorithms in Molecular Biology, lecture notes, 2001-2002, Tel Aviv University School of Computer Science. www.cs.tau.ac.il/ rshamir/algmb/01/algmb01.html

[12] J. Setubal, J. Meidanis. *Introduction to Computational Molecular Biology*, PWS Publishing Company, 1997.

[13] T.F. Smith, M.S. Waterman (1981). Identification of Common Molecular Subsequences. Journal of Molecular Biology 147: 195–197. doi:10.1016/0022-2836(81)90087-5
▷ local alignment, Section 1.3