CHAPTER 2

Chaining of Matches

2.1 Anchor based sequence comparison

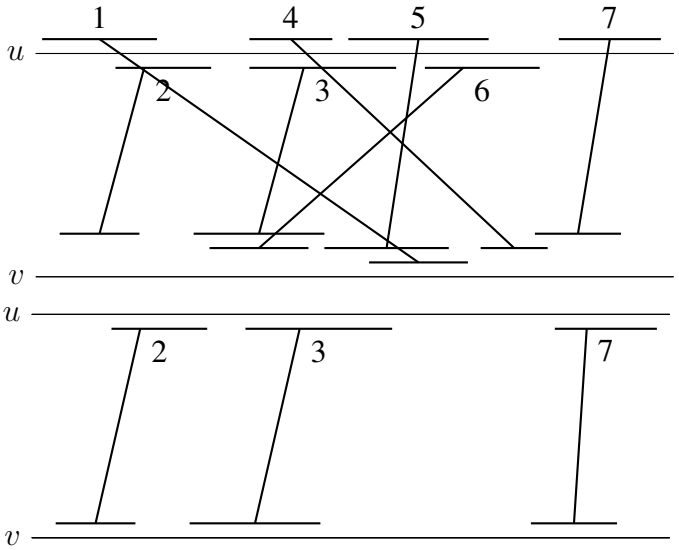
Comparative genomics is concerned with comparing genomic sequences to each other. If the organisms under consideration are closely related (that is, if no or only a few genome rearrangements have occurred) or one compares regions of conserved synteny (regions in which orthologous genes occur in the same order), then global alignments can be used for the prediction of genes and regulatory elements. This is because coding regions are relatively well preserved, while non-coding regions tend to show varying degree of conservation. Non-coding regions that do show conservation are thought important for regulating gene expression, maintaining the structural organization of the genome and possibly have other, yet unknown functions. Several comparative sequence approaches using alignments have recently been used to analyze corresponding coding and non-coding regions from different species. These approaches are based on software-tools for aligning two or multiple genomic DNA-sequences. Here we will focus on comparing two genomes, but most methods described in this chapter can be extended to the comparison of multiple genomes. To cope with the shear volume of data, most of the software-tools use an anchor-based method. Anchor-based alignment methods are usually composed of three phases:

1. computation of matches (substrings in the genomes that are similar),

2. computation of a highest-scoring chain of colinear non-overlapping matches: the anchors that form the basis of the alignment,

3. alignment of the regions between the anchors.

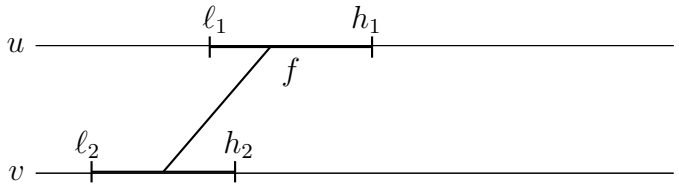
Figure 2.1: A set of seven matches between u and v (top) and a colinear chain of non-overlapping matches (bottom figure). The matches 2 and 3 are colinear, while 1 and 6 are not. Matches 1 and 2 overlap, while 2 and 3 do not.



2.2 The match chaining problem

This chapter deals with algorithms for solving the combinatorial chaining problem of the second phase: finding a highest-scoring chain of colinear non-overlapping matches. Roughly speaking, two matches are *colinear* if the order of their respective matches is the same in both genomes. Two matches *overlap* if the sequence involved in the matches overlap in at least one of the genomes. In the pictorial representation of Figure 2.1 (left), two matches are colinear if the lines connecting their matches are non-crossing.

We consider two sequences u and v of length m and n. A *match* f is given by two pairs $s(f) = (\ell_1, \ell_2)$ and $e(f) = (h_1, h_2)$ such that the strings $u[\ell_1 \dots h_1]$ and $v[\ell_2 \dots h_2]$ are "similar". $s(f)$ is the start point and $e(f)$ is the end point of f. See the following for a graphical representation:



Matches are often exact matches, *i.e.*, $u[\ell_1 \dots h_1] = v[\ell_2 \dots h_2]$. Examples of exact matches are maximal unique matches, maximal exact matches, and exact q-grams. In general, however, one may also allow substitutions or even insertions and deletions. Each match f has a positive

weight *weight*(f) that can, for example, be the length of the match (in case of exact matches) or its similarity score.

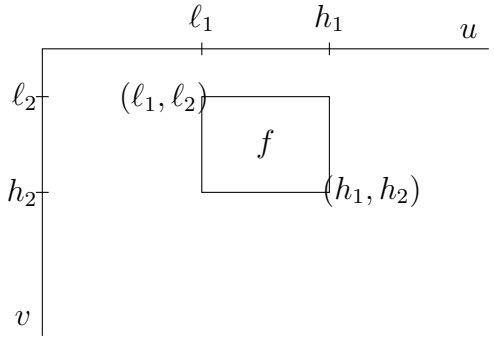
Each match f can be represented by a rectangle in the $(m+1) \times (n+1)$-matrix E_δ, such that the point

$$s(f) = (\ell_1, \ell_2)$$

is the upper left corner and the point

$$e(f) = (h_1, h_2)$$

is the lower right corner of the rectangle $[\ell_1 \ldots h_1] \times [\ell_2 \ldots h_2]$, see the following for a graphical representation:

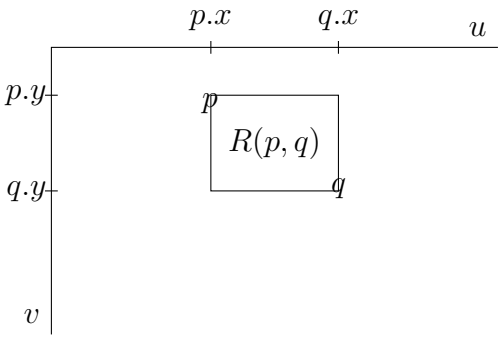


In general, a rectangle is denoted by $R(p, q)$ where p and q are the upper left and the lower right corner, respectively.

For any point $p = (p_1, p_2) \in [0, m] \times [0, n]$, let

$$p.x := p_1 \text{ and } p.y := p_2.$$

Here is a graphical representation:

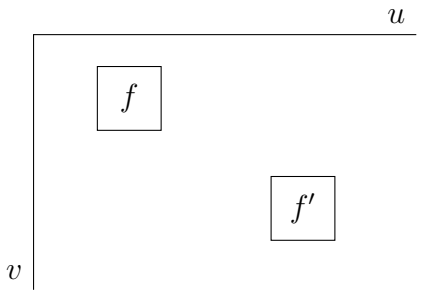


Definition 5 Let $\mathcal{M}$ be the given set of matches. We define a binary relation $\ll$ on $\mathcal{M}$ by $f \ll f'$ if and only if

$$e(f).x < s(f').x \text{ and } e(f).y < s(f').y.$$

If $f \ll f'$, then we say that f *precedes* f'.

Here is a graphical representation of f preceding f'.



Definition 6 A *chain* of colinear non-overlapping matches (or chain for short) is a sequence of matches $f_1, f_2, \ldots, f_\ell$ such that

- $f_i \in \mathcal{M}$ for all i, $1 \leq i \leq \ell$ and
- $f_i \ll f_{i+1}$ for all i, $1 \leq i < \ell$.

The *score* of C is $score(C) = \sum_{i=1}^{\ell} \mathit{weight}(f_i)$.

Definition 7 Given a set $\mathcal{M}$ of b weighted matches, the *match-chaining problem* is to determine a chain of maximal score. Such a chain is called *optimal* chain.

Recall that match-weights are positive. Hence an optimal chain is maximal on both ends. More formally, an optimal chain ends with a match f such that there is no match f' satisfying $f \ll f'$. Similarly, an optimal chain begins with a match f' such that there is no match f satisfying $f \ll f'$.

2.3 A simple chaining algorithm

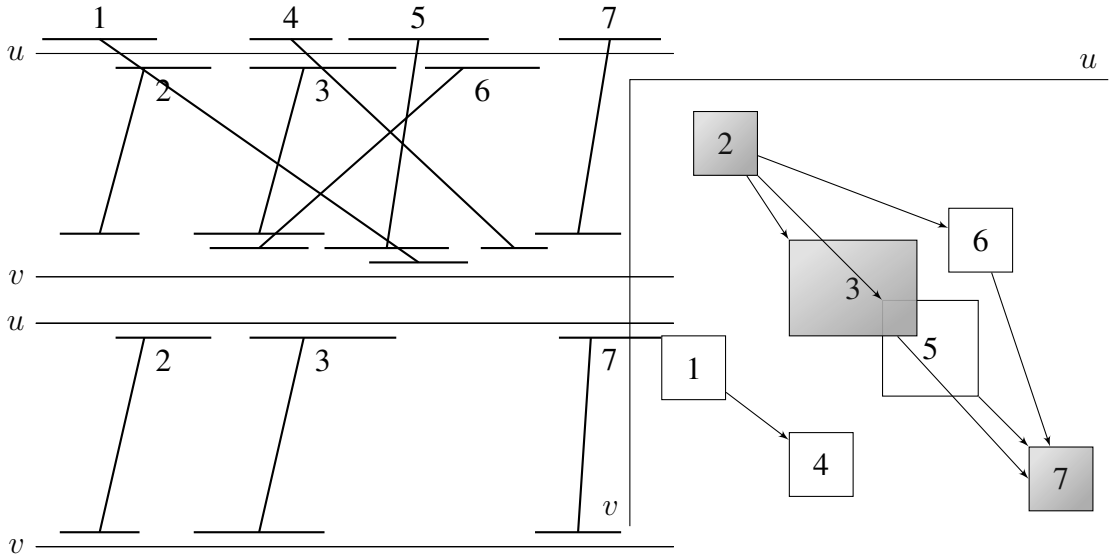
A direct solution to the match-chaining problem is to construct a weighted directed acyclic graph $G = (V, E)$ such that

- the set V of vertices is $\mathcal{M}$,
- the set of edges E is characterized as follows: There is an edge $f \rightarrow f'$ with weight $\mathit{weight}(f')$ if and only if $f \ll f'$; see Figure 2.2 for an example.

The first match f of a chain gets weight $\mathit{weight}(f)$. An optimal chain of matches corresponds to a path of maximum score in the graph. Because the graph is acyclic, such a path can be computed as follows. The idea is to divide the set of all chains according to the last match they consist of. Let $\mathit{score}(f')$ be defined as the maximum score of all chains ending with f'. To compute $\mathit{score}(f')$, we consider all chains ending with some match f which may precede f' and then add $\mathit{weight}(f')$ to that chain. Obviously, $\mathit{score}(f')$ satisfies the following recurrence:

$$\begin{aligned} \mathit{score}(f') &= \max(\{0\} \cup \{\mathit{score}(f) \mid f \in \mathcal{M}, f \ll f'\}) + \\ &\quad \mathit{weight}(f') \end{aligned}$$

Figure 2.2: Consider the matches and a corresponding chain of colinear matches on the left (see also Figure 2.1). This chain corresponds to an optimal path in the graph (in which some edges are omitted).



If there is no $f \in \mathcal{M}$ satisfying $f \ll f'$ then

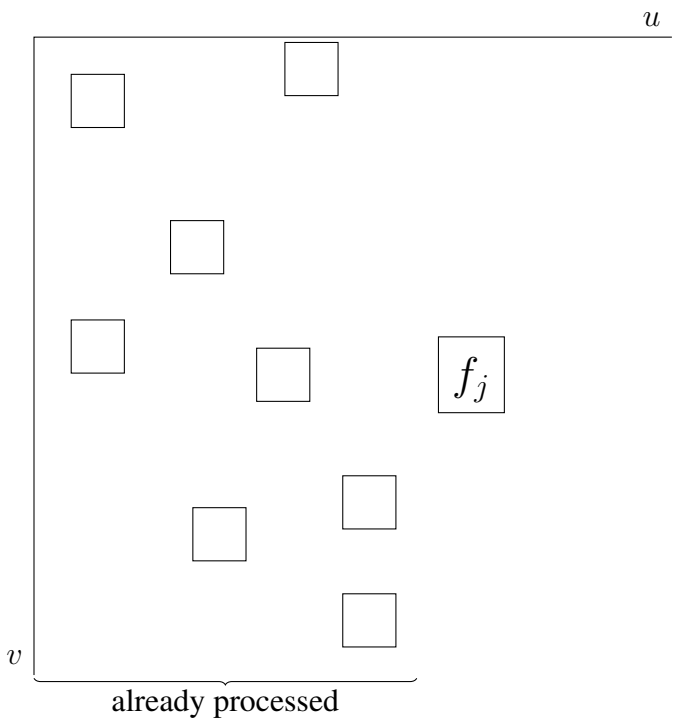
$$\begin{aligned} \mathit{score}(f') &= \max(\{0\} \cup \{\mathit{score}(f) \mid f \in \mathcal{M}, f \ll f'\}) + \mathit{weight}(f') \\ &= \max(\{0\} \cup \emptyset) + \mathit{weight}(f') \\ &= \max\{0\} + \mathit{weight}(f') \\ &= 0 + \mathit{weight}(f') \\ &= \mathit{weight}(f') \end{aligned}$$

The DP algorithm based on this recurrence, see Algorithm 2, first sorts the matches according to the x-coordinate of the end-points. In an outer loop it iterates over all matches f_j in this order. To find the matches f satisfying $f \ll f_j$ it only has to consider the matches which have previously been considered and for these $\mathit{score}(f)$ has been determined, see Figure 2.3.

From all matches previously considered, a match f_i maximizing $\mathit{score}(f_i) + \mathit{weight}(f_j)$ is determined. The fact that f_i is a predecessor in an optimal chain ending in f_j is maintained in table $prec$ by setting $prec(f_j) := f_i$. Note that $prec(f_j) = \perp$ if f_j is the first element in the chain. There is an optimal chain ending with match *bestmatch*. To obtain this, one starts at *bestmatch* and traces back the $prec$-values until a match f satisfying $prec(f) = \perp$ is found.

In each of the b iterations of the outer loop, on average $b/2$ previous matches have to be considered. Checking the relation $\ll$ and comparing the score can be done in constant time. Hence the algorithm runs in $O(b^2)$ time. The space requirement is $O(b)$, as the matches are processed twice which requires to store them. This algorithm can be generalized to any

Figure 2.3: The matches considered when processing match f_j in the simple chaining algorithm (Algorithm 2).



number of genomes. However, for two sequences, the $O(b^2)$ time bound can be improved by considering the geometric nature of the problem. This is explained in the following section. The following example shows that such an improvement is really necessary.

Example 2 Consider the two bacterial strains K12 and O157:H7 of the bacterium *Escherichia coli*. The length of the corresponding genomes is 4,639,221 bp and 5,471,376 bp. There are 815,030 matches of minimum length 13 between these two genomes on the forward strand and 1,587,527 matches of minimum length 13 when considering both strands. So the number of iterations of the inner for loop of Algorithm 2 is roughly

$$\begin{aligned} \frac{815030 \cdot 815030}{2} &\approx 3.32 \cdot 10^{11} \text{ and} \\ \frac{1587527 \cdot 1587527}{2} &\approx 1.26 \cdot 10^{12} \end{aligned}$$

2.4 An efficient match chaining algorithm

To compute $score(f')$, we have to maximize the score over all matches f satisfying $f \ll f'$. By Definition 5, $e(f).x \leq s(f').x - 1$ and $e(f).y \leq s(f').y - 1$. Thus, in geometric terms,

Algorithm 2 A simple chaining algorithm running in $O(b^2)$ time.

Input: Set of matches $\mathcal{M} = \{f_1, f_2, \ldots, f_b\}$ ordered by x-coordinate of end-point
Output: optimal chainscore, last element of chain with such score, $prec$ function.

```
1:  overallmaxscore := 0
2:  bestmatch := ⊥
3:  for j := 1 upto b do
4:      maxscore := weight(f_j)                          // chain with f_j only
5:      prec(f_j) := ⊥
6:      for i := 1 upto j − 1 do
7:          if f_i ≪ f_j and maxscore < score(f_i) + weight(f_j) then
8:              maxscore := score(f_i) + weight(f_j)     // extend by f_i
9:              prec(f_j) := f_i                         // set predecessor of f_j
10:         end if
11:     end for
12:     score(f_j) := maxscore
13:     if overallmaxscore < maxscore then
14:         overallmaxscore := maxscore
15:         bestmatch := f_j
16:     end if
17: end for
18: return (overallmaxscore, bestmatch, prec)
```

we have to maximize over all matches in the rectangle $R((0,0),(s(f').x - 1, s(f').y - 1))$. This is a two dimensional search problem. It can be reduced to a one dimensional search problem by processing the matches in ascending order of their x-coordinate. In this way, only a subset of the matches needs to be considered, namely those whose end point is strictly to the left of the start point of the current fragment, both with respect to the x-coordinate. This was already done for the quadratic chaining algorithm described above. However, this processes the matches in a linear scan to obtain the best predecessor. We show how to maintain a relevant subset of the previous matches in a dictionary structure, ordered by the y-coordinate. This allows to efficiently determine the predecessor of the current match. See Figure 2.4 for a graphical explanation.

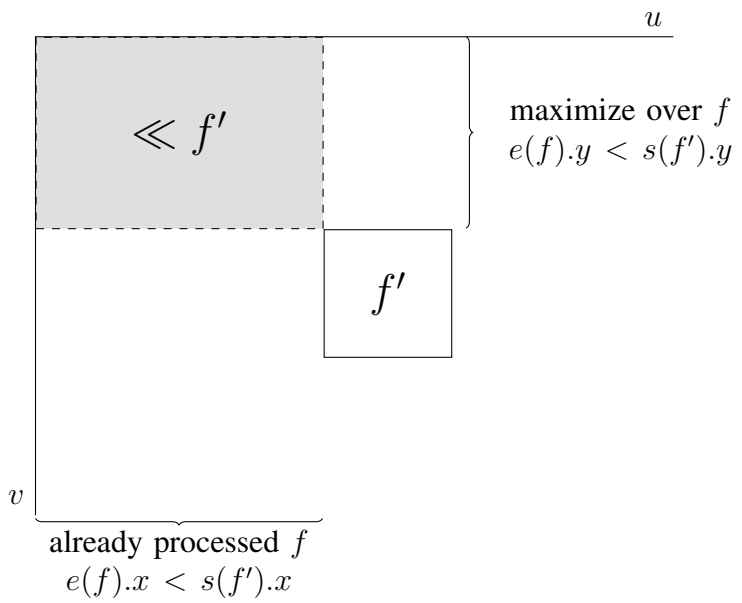
The efficient algorithm considers each match twice, once with respect to its start point and once with respect to its end points. Therefore, we duplicate $\mathcal{M}$ to obtain a set $\mathcal{M}_{S,E}$ as follows:

$$\mathcal{M}_{S,E} = \{(f,0) \mid f \in \mathcal{M}\} \cup \{(f,1) \mid f \in \mathcal{M}\}.$$

That is, $\mathcal{M}_{S,E}$ contains each match in $\mathcal{M}$ twice. Once we consider the start point of the match (signified by 0) and once the end point of the match (signified by 1). This is expressed by introducing the projection function φ defined by

$$\varphi(f,0) \quad = \quad s(f).x \text{ and}$$

Figure 2.4: In each step, when considering a match f', the efficient chaining algorithm only searches for matches in the shaded rectangle. With regard to the x-coordinate, the selection is achieved via processing matches in sorted order (according to $\prec_x$). With regard to the y-coordinate, the selection is achieved by searching a dictionary storing matches in sorted order according to $\prec_y$.



$$\varphi(f,1) \;=\; e(f).x$$

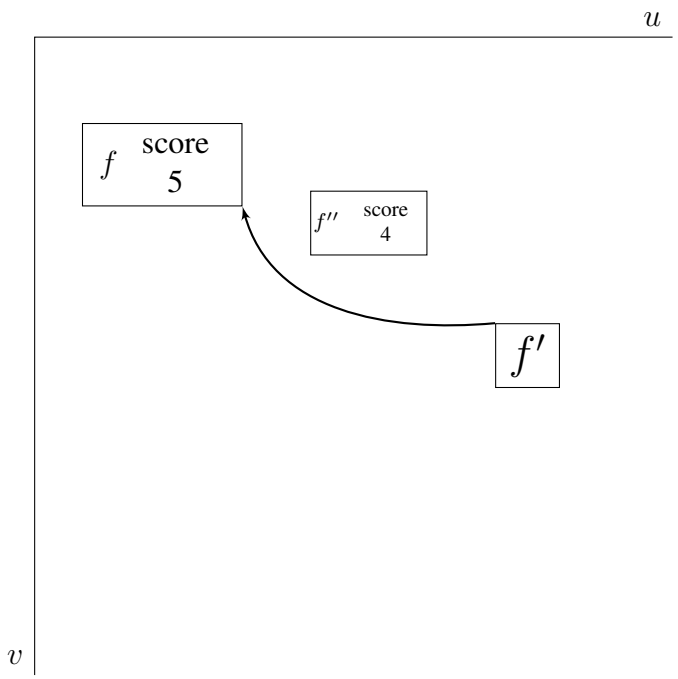
To sort $\mathcal{M}_{S,E}$, we introduce an order $\prec_x$ on $\mathcal{M}_{S,E}$ as follows:

$$(f,a) \prec_x (f',a') \;\Leftrightarrow\; \begin{array}{l}\varphi(f,a) < \varphi(f',a') \text{ or} \\ (\varphi(f,a) = \varphi(f',a') \text{ and } a < a')\end{array}$$

In other words, the matches are ordered in ascending order with respect to their x-coordinate. If a start point and an end point have the same x-coordinate, then the start point is smaller than the end point. The efficient algorithm processes the matches in $\mathcal{M}_{S,E}$ ordered according to $\prec_x$. Each match is considered twice, once with the y-coordinate of its start point and once with the y-coordinate of its end point. Thus consider two cases:

Case 1: Let $(f',0)$ be the element in $\mathcal{M}_{S,E}$ that is currently processed, where 0 means that we consider the start point $s(f')$. We have already processed all matches f ending in the rectangle $R((0,0),(s(f').x-1, s(f').y-1))$. By stating that f is processed, we mean that $score(f)$ and the previous match (if any) in an optimal chain ending with f has been determined. Among the processed matches we only have to consider those matches f satisfying $e(f).y < s(f').y$. If such a match exists, we take the one with maximal score, say f. Then we know that the optimal chain ending with f' has the previous match f and the score is

Figure 2.5: The match f'' with score 4 can be eliminated because each chain that could use f'' could also use f with score 5 and increase the score of the chain.



score$(f') =$ *score*$(f) +$ *weight*(f'). If there is no such match, then the optimal chain ending with f' only consists of f' and *score*$(f') =$ *weight*(f').

Case 2: Let $(f', 1)$ be the element in $\mathcal{M}_{S,E}$ currently processed, where 1 means that we consider the end point $e(f')$. As explained in the previous case, we have to make sure that, among the processed matches, we can efficiently determine the match with the maximum score such that the y-coordinate of its end point is smaller than some value depending on f'. The idea is to store processed matches in dictionary D sorted in ascending order of the y-coordinate of their end points. Let $\prec_y$ denote this order, that is, for any pair of matches f and f'':

$$f \prec_y f'' \iff e(f).y < e(f'').y.$$

Suppose there are two processed matches f and f'' such that $e(f).y \leq e(f'').y$ and *score*$(f) >$ *score*(f''). Then an optimal chain will not include f''. Each chain that could use f'' could also use f and increase the score of the chain, see Figure 2.5 for a graphical explanation. Therefore, a match f' is only inserted if there is no predecessor or if the predecessor has a smaller score. Once it it inserted, one follows the chain of successor matches and deletes all matches which have a smaller score than f'. As a consequence, $f \prec_y f''$ always implies *score*$(f) \leq$ *score*(f'') for two matches f and f'' in D. So the order $\prec_y$ on the processed matches implies the order with respect to their score. To maintain D, we suppose that the following operations are supported:

Algorithm 3 Two dimensional chaining of b matches.

```
1: Compute M_{S,E} and sort it according to ≺_x
2: D := ∅
3: for all (f', a) ∈ M_{S,E} in sorted order do
4:     if a = 0 then                                          // process start point
5:         if there is an f ∈ D satisfying e(f).y < s(f').y then
6:             let f ∈ D be maximal w.r.t. ≺_y s.t. e(f).y < s(f').y
7:             score(f') := weight(f') + score(f)
8:             prec(f') := f
9:         else
10:            score(f') := weight(f')
11:            prec(f') := ⊥
12:        end if
13:    else                                                   // process end point
14:        if pred(D, f') = ⊥ or score(pred(D, f')) < score(f') then
15:            D := insert(D, f')
16:            f'' := succ(D, f')
17:            while f'' ≠ ⊥ and score(f') > score(f'') do
18:                tmp := f''
19:                f'' := succ(D, f'')
20:                D := delete(D, tmp)
21:            end while
22:        end if
23:    end if
24: end for
```

- $insert(D, f)$: if f is not in D, then insert f into D
- $delete(D, f)$: remove f from D
- $pred(D, f)$: determine the largest match $f'' \in D$ (with respect to $\prec_y$) satisfying $f'' \prec_y f$. If there is no such match, then $pred(D, f) = \perp$ which stands for undefined.
- $succ(D, f)$: determine the smallest match $f'' \in D$ (with respect to $\prec_y$) satisfying $f \prec_y f''$. If there is no such match, then $succ(D, f) = \perp$.

The complete algorithm is specified in Algorithm 3.

The dictionary D can be implemented by a balanced binary search tree, for example an AVL-tree or a red-black-tree. These support all necessary operations in $O(\log b)$ time. Since $2b$ match points are processed, the for-loop in Algorithm 3 requires $O(b \log b)$ time. Additionally we have to sort the $2b$ match points, which requires $O(b \log b)$ time. This is also the overall running time. The space requirement is clearly $O(b)$.

Example 3 Again consider the comparison of the two *Escherichia coli* strains K12 and O157:H7, see Example 2. Recall that there are 815,030 matches of minimum length 13 between these

two genomes on the forward strand. The program Vmatch `http://www.vmatch.de` takes 2.6 seconds to compute and output these matches. Most of the time is used for the output. Instead of outputting these matches one can directly compute an optimal chain, which in our case contains 40,926 matches and has score 7,286,560, when assigning a score of 1 to each position of a match. Vmatch uses the algorithm described in this section and it requires less than 2 seconds to compute the matches and perform the chaining.