# Chapter 6

# DOUBLEBUILD

## 6.1 Introduction and motivation

The design of a pair HMM, i.e. its states and transitions, can be done rather quickly using pencil and paper. However, the implementation of the pair HMM into programming code and especially the implementation of alignment algorithms with which the pair HMM can be used to analyse sequences, can be a time consuming task.

[BD97] present a compiler called **DYNAMITE** with which a variety of pair HMMs can be defined and used with alignment algorithms to produce a prediction. The desired pair HMM is defined in a text file using the **DYNAMITE** language. This **DYNAMITE** file is then translated into C programming code using the **DYNAMITE** compiler and this C code must then be compiled with a standard C compiler before it *can* be executed. The **DYNAMITE** compiler shields the user from the underlying implementation into C programming code. This has the advantage of making the implementation of a pair HMM easy as the user only has to provide a short definition file using the **DYNAMITE** language. However, the introduction of an intermediate compiler makes it difficult for a user to understand and modify the underlying C source code.

My aim in programming **DOUBLESCAN** and **PROJECTOR** was to create a set of C++ classes, called **DOUBLEBUILD**, which can be used to define a variety of pair HMMs in a short time and which also provide sophisticated alignment algorithms so that the pair HMMs can be directly used for the analysis of data. In that respect, **DOUBLESCAN** and **PROJECTOR** can be seen as two sophisticated examples of what can be done with **DOUBLEBUILD.**

My main motivation for choosing C++ as a programming language was to use an object oriented language which provides all the features that I needed to realise projects such as

DOUBLESCAN and PROJECTOR. Using an object oriented language, there is no need to write an extra compiler to generate the final source code because the building blocks of the pair HMM correspond to the classes defined in DOUBLEBUILD with which the programmer can directly define and operate pair HMMs. This made things easier for me and hopefully also for the user who may not only wish to use the source code, but who may also wish to modify or extend it. C++ has the additional benefit that a freely available compiler (GNU compiler) exists and that an **ANSI** standard has already been defined which guarantees a high level of portability of the source code. All relevant data structures are provided by DOUBLEBUILD itself. In particular, the standard template library is not used.
This chapter first introduces the novel concept of special transitions and the concept of special emissions within pair HMMs and describe how they are implement within DOUBLEBUILD. It then presents the three main classes that form the foundation of DOUBLEBUILD, namely the **Sequence** class, the **PairhmmState** class and the **Pairhmm** class. Their description should also make clear how these classes interact. Finally, functions of special interest such **as** a variety of alignment algorithms are described.

## 6.2 Special transitions within DOUBLEBUILD

Special transitions are a new concept introduced in this dissertation. Special transitions within DOUBLEBUILD can be used to make any transition within a pair HMM dependent on position specific scores. These transitions are implemented in a way which conserves the probabilistic interpretation of the transition probabilities. The pair HMM underlying DOUBLESCAN and PROJECTOR is one example for a pair HMM which uses special transitions, ***see*** arrows with dots in Figure **2.4.** It uses special transitions to model the sequence signals around translation start sites and splice sites. These sequence signals are contained in a sequence interval which is too large to be easily incorporated into one state of the pair HMM, and the signal itself is too complex to be adequately modelled by the emission probabilities of a state. Before starting the gene prediction with algorithms such **as** the Hirschberg algorithm or the Stepping Stone algorithm, the two input sequences X and ***Y*** are first separately searched for potential translation start sites and splice sites by dedicated programs. **Each** potential translation start and splice site is assigned a score which is a measure of the likelihood for this site to be a true translation start or splice site. These sequence signal scores are stored for each sequence separately in its corresponding **Sequence** object. Once the two input sequences have been scanned for sequence signals, one of the algorithms is used to predict genes and align the two sequences. The sequence signal scores are used within the algorithm to modify the nominal values of the transition probabilities so that the transition has a high probability if it is supported by strong sequence signal scores at the given sequence positions.

Figure **6.1** shows a generic example with which the general concept of special transitions is elucidated in the following. Any alignment algorithm such as the Viterbi algorithm, Hirschberg algorithm or Stepping Stone algorithm derives the optimal state path according to the transition and emission probabilities encountered on the state paths through the pair HMM. The alignment algorithms work internally with scores which are derived from probabilities by $\texttt{score} = \log_2(\texttt{probability})$. The transition score for **a** transition from state **from** to state **to** at position **xpos** in sequence X and position **y-pos** in sequence Y, see Figure 6.1, is calculated in the following way (description given in pseudo-code):

```
special_transition_score(from, to, X, x_pos, Y, y_pos) {
  if from → to special {
    return-score = score(special_transition_prob(from,to,X,x_pos,Y ,y_pos))
  }
  else {
    if exists to' with from → to' special {
      returnscore = score(special_transition_prob(from,to,X,x_pos,Y,y_pos))
                    + score(scale_factor)
      where
      $\texttt{scale\_factor} = \left( \frac{1 - \sum_{\substack{to' \\ from \to to' \text{ special}}} \texttt{special\_transition\_prob(from,to',X,x\_pos,Y,y\_pos)}}{\sum_{\substack{to' \\ from \to to' \text{ not special}}} \texttt{transition\_prob(from,to')}} \right)$
    }
    else {
      returnscore = transition_score(from, to)
    }
  }
  return(returnscore)
}
```

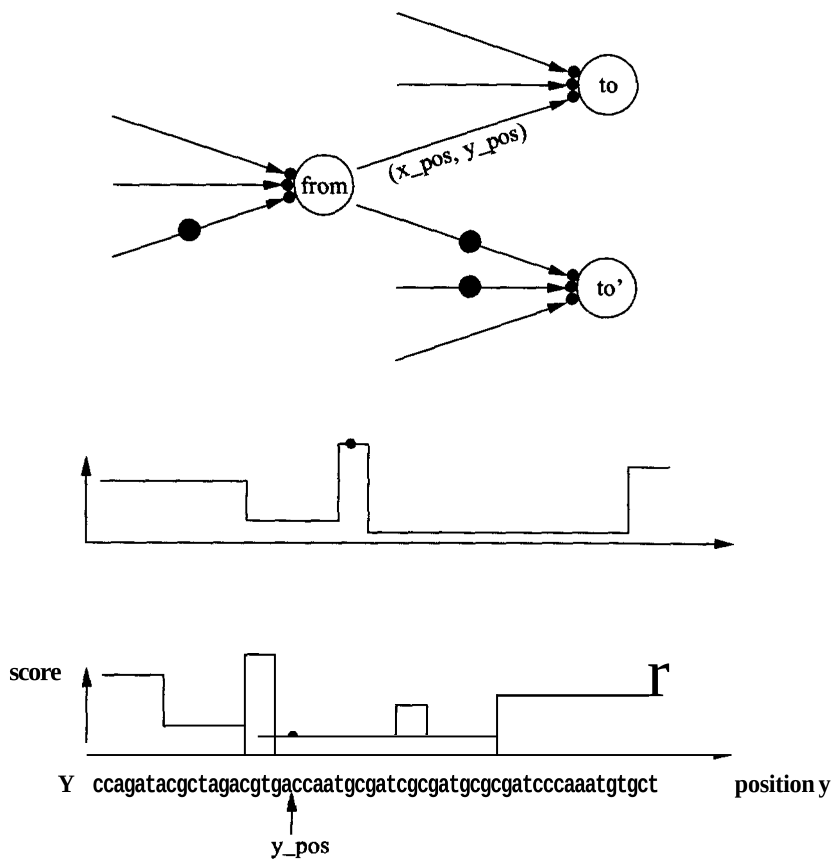

Figure **6.1:** Part of a pair HMM with special transitions. Special transitions correspond to the arrows marked with a big dot. Transitions belong to the state to which they are leading as indicated by a small dot between the tip of an arrow and its state. When calculating the probability for the transition from state `from` to state `to` which is not special, we have to take into account the position dependent values of all special transitions emerging from state `from` in order to ensure that the probabilities of all transitions emerging from state `from` always sum up to one. See the text for a detailed description.

If only non-special transitions are emerging from state from, the transition score **for** the transition from state from to state to at position x-pos and y-pos is equal to the nominal value of the transition score **for** going from state from to state to (transition_score(from, to)) which is independent of the positions in the input sequences. If the transition from state from to state to is special, the transition score depends on the positions in the input sequences and is equal to **score(special_transition_prob(from,** to, X, **x_pos,** Y, y_pos)), where the probability returned by **special_transition_prob(from,** to, X, **x_pos,** Y, y_pos) is calculated by:

```
special_transition_prob(from, to, X, x_pos, Y, y-pos) {
  return_prob = transition_prob(from, to)
  if from → to special {
    return_prob *= posterior_prob(prior, score)
    where
    prior = $\sqrt{\text{prior\_x} \cdot \text{prior\_y}}$ if to state of type EmitXY
            prior_x                                   if to state of type EmitX
            prior_y                                   if to state of type EmitY
    score = score-x + score-y if to state of type EmitXY
            score_x           if to state of type EmitX
            Score-y           if to state of type EmitY
    prior_x = X.prior(from, to, x-pos)
    prior-y = Y.prior(from, to, y-pos)
    score_x = X.score(from, to, x-pos)
    score-y = Y.score(from, to, y-pos)
    and
    posteriorqrob(prior, score) = $\left(\frac{\text{prior}\cdot 2^{\text{score}}}{\text{prior}\cdot 2^{\text{score}}+1-\text{prior}}\right)$
  }
}
```

If the transition from state from to state to is not special, but if there **are** special transitions emerging from state from **(as** is shown in the example in Figure 6.1), the nominal value of the non special transition from state from to state to is adjusted so that the sum of all transition probabilities emerging from state **from** at any pair of sequence positions **(xpos , y-pos)** remains one. This is done by calculating a scaling factor **(scalefactor)** by which the value of the nominal transition probability is multiplied.

Generally, the priors for the special transitions may depend on the position within the sequence. To name an example, the value of the prior for the special transition between the match exon and the emit *x 5'* splice site phase 0 state in the pair HMM underlying DOUBLESCAN and PROJECTOR depends on whether this is a consensus GT or a non-consensus GC splice site (see Table D.1 and Figure 2.4).

The details of how special transitions are implemented into the C++ classes of DOUBLEBUILD are described in Section 6.4.

## 6.3 Special emissions within DOUBLEBUILD

Not only transition probabilities, but also emission probabilities can be made dependent on the sequence positions. States whose emission probabilities depend on the positions in the input sequences are called states with special emissions. In PROJECTOR (see Chapter 3), special emissions are used to implement constraints into the calculation of the optimal state path. Only those state paths are considered in the calculation of the optimal state path which reproduce the known annotation of one of the two input sequences. This way we can project the known genes of one input sequence onto the other input DNA sequence of yet unknown annotation. PROJECTOR is just one of many possible applications of special emissions within pair HMMs. The following paragraph illustrates the generality of the concept of special emissions.

Again, the algorithms internally employ scores, the logarithm of the probabilities, in order to avoid the numerical difficulties which arise when dealing with small probabilities. However, as scores and probabilities have a one-to-one correspondence, they can be easily converted into each other. For any given state **this** in a pair HMM, see Figure 6.2, the emission score is calculated in the following way:

```
special-emissionscore(this, X, x-pos, Y, y_pos) {
  return-score = emission-score(this, X, x-pos, y, y_pos)
  if this state has special emissions {
    returnscore += score
```

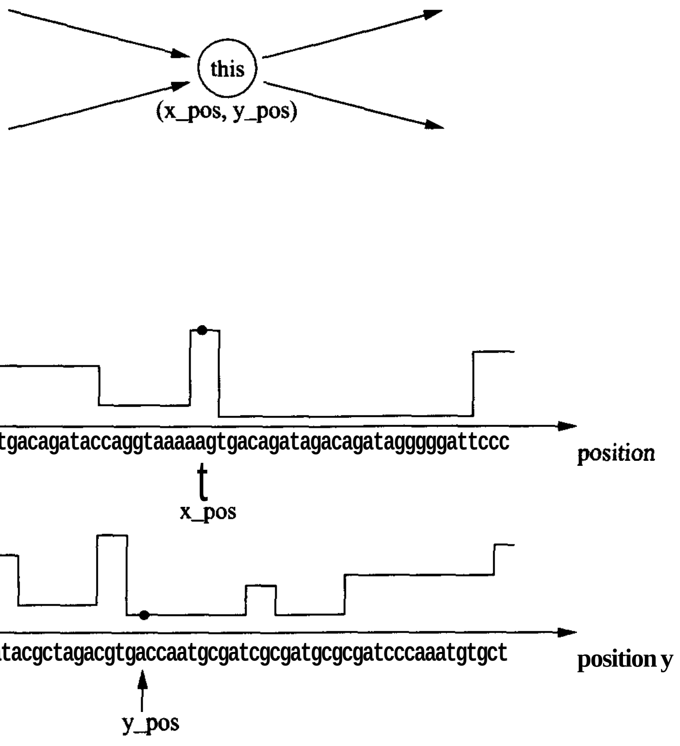
this
(x_pos, y_pos)
score
X acgatgacagataccaggtaaaaagtgacagatagacagatagggggattccc
position
x_pos
score
Y ccagatacgctagacgtgaccaatgcgatcgcgatgcgcgatcccaaatgtgct
position y
y_pos

Figure 6.2: State of a pair HMM with special emissions. The emission probability at positions (xpos, y_pos) not only depends on the letters read at these sequence positions, but also on the score at position x-pos in input sequence X and on the score at position ypos in input sequence *Y*. See the text for a detailed description.

```
        where
        score = score_x + score-y if this state of type EmitXY
                score_x           if this state of type EmitX
                score_y           if this state of type EmitY
        and
        scores = X.score(this, xpos)
        score-y = Y.score(this, y_pos)
    }
    return(return_score)
}
```

If **this** state does not have special emissions, the emission score (**emission_score(this, X, x-pos, Y, y-pos))** only depends on the ***letters*** read from the input sequences at the given sequence position **(x-pos, y_pos)**, but ***not the sequence positions.*** If **this** state has special emissions, the nominal value of the emission score **(emissionscore(this, X, x-pos, Y, ypos))** which only depends on the letters read is modified by a score which depends on the scores at the given sequence positions **(score-x** and **score-y).**

## 6.4 The main classes

The three main classes of **DOUBLEBUILD** are the **Sequence,** the **PairhmmState** and the **Pairhmm** class. Each of the three classes has a set of private variables whose values characterise every instance of each class.
We first introduce the private variables of each class in order to illustrate how the different classes interact within a pair HMM. The private variables are more important for the understanding of the concept of **DOUBLEBUILD** than the set of public functions by which the values of the private variables are accessed.

### 6.4.1 The Pairhmm class

**A Pairhmm** object knows the number of states it consists of **(int number-ofstates)** and has an array with pointers to each of its states (**Pairhmm_State* model).** It has private variables for storing a state path and provides private functions which are used as the building blocks of public functions such as the Stepping Stone algorithm and the Hirschberg algorithm.

### 6.4.2 The Pairhmm-State class

**PairhmmState** objects constitute the building blocks of a pair HMM and interact with **Sequence** objects. The definition of the **PairhmmState** class was motivated by the idea that each state should know about itself and its direct neighbours within the pair HMM (a direct neighbour being a state that can be reached within a single transition).
In its simplest variant, a **PairhmmState** object knows:

- **int number-ofstate** its number within the **Pairhmm**
- **int alphabet** the alphabet of letters it reads from an input sequence

- **int number_of_letters_to_read** the number of letters it **reads** from an input sequence
- **array<Labelseq> labels_of_letter_to_read** the labels it assigns to the letters **read**
- **array<Phase> phases_of_letters_to_read** the phases it assigns to the letters
- **State_type state_type** its own state type, e.g. if it is an **EmitXY**, **EmitX** or **EmitY** state or some other type of state
- **array<Prob> emissionprobs** the array of its emission probabilities
- **array<Prob> transitionprobs** the array of transition probabilities to states which **can** be reached from this state
- **int number-ofstates** the number of states in the **pair HMM** to which this state belongs
- **int number-ofnext-states** number of states which can be reached from this state
- **array<int> numbers-ofnext-states** array of the numbers of the states which *can* be reached from this state
- **int number-of-previous-states** number of states which have a transition to this state
- **array<int> numbers-of-previousstates** array of the numbers of the states which have a transition to this state

### PairhmmState objects with special emissions

If a state has special emissions, its emission probabilities depend both on the letters it **reads** and position specific sequence scores. This concept is .employed in **PROJECTOR** to predict an annotation for one of the two input sequences while keeping that of the other sequence fixed. Suppose we are dealing with the match exon state of **PROJECTOR** and are keeping the annotation of sequence $X$ fixed, see Figure **2.4** and Figure **B.3** in Appendix B. We want the match exon state to have non-zero emission probabilities only **for** letters whose annotation matches the labels and phases of sequence $X$. For a given position in sequence $X$, the emission probability within the match exon state is calculated by requesting the corresponding score for that position from object **Sequence X.** Instead of storing the information for every state with special emission probabilities and all sequence positions in **Sequence** X, the information is only stored for a few states from which the information of the remaining states with special emissions can be derived. The match exon state derives its information on the position specific emission probabilities of sequence X by requesting that of state emit x exon state as it suffices to know where this state is allowed in order to know where the match exon state is allowed. The private variables used to implement special emissions are:

- int special-emission indicates if this state has special emissions or not. Its value is one if this state has special emissions and zero otherwise.
- int number_of_child_state_x is the number of the state under which the position specific sequence scores for this state are stored in Sequence X, in the above example this would be the number of the state emit x exon
- int number_of_child_state_y is the number of the state under which the position specific sequence scores of this state are stored in Sequence Y (In the above example, special emission probabilities within the match exon state are only used for the position specific scores of sequence X as only the annotation of sequence X is kept fixed. In this case, int number-of_child_state_y would be set to zero.)

### PairhmmState objects with special transitions

If a state has special transitions, the probability of one or several transitions leading into the state depends on position specific scores within the input sequences. Both DOUBLESCAN and PROJECTOR use special transitions to improve the detection of splice sites and translation start sites. Dedicated programs score potential translation start sites and splice sites within the two input sequences separately. These position specific scores are then used within the pair HMM to modify the nominal values of the transition probabilities. If both sequences have strong signals for being 5' splice sites at the given sequence positions, the probability of transferring from the match exon to the match 5' splice site state is high and small otherwise, see Figure 2.4.

As for special emissions, the information about the position specific scores as well as the priors is stored within the Sequence objects of the two input sequences, Sequence X and Sequence Y. And as for special emissions, also special transitions are implemented in a memory efficient way by storing information only for a minimum of transitions from which the information of the remaining special transitions can be easily derived. The probability of the special transition from the match exon to the match 5' splice site phase 1 state, see Figure 2.4 and Figure B.3 in Appendix B, is derived from that of the special transition from the match exon

to the emit x 5' splice site phase 0 state for input sequence X and from that of the special transition from the match exon to the emit y 5' splice site phase 0 state for input sequence Y by considering the offset of one base pair. For long sequences, these tricks save a considerable amount of memory.
The private variables dealing with special transitions are:

- **int special** indicates if this state has special transitions or not. Its value is one if this state has special transitions and zero otherwise.
- **int number_of_special_transitions_to_previous_states** is the number of special transition leading into this state
- **array<int> special_flags_of_transitions_to_previous_states** one-dimensional array indicating for each transition leading into this state if it is special (array element has value one) or not (array element has value zero)
- **array<int> numbers_of_from_child_states_x_previous** one-dimensional array indicating for each transition leading into this state the number of the 'from' state to be used for deriving the special transition score for sequence X, if the transition is special. If an alternative transition is to be used, this number is the number of the 'from' state of the alternative transition. In the above example in which we are dealing with the match 5' splice site phase 1 state, the array element for the transition from the match exon state to the match 5' splice site phase 1 state is the number of the match exon state as this is the 'from' state of the alternative transition from the match exon state to the emit x 5' splice site phase 0 state which is used for deriving the position specific score of sequence X.
- **array<int> numbers_of_to_child_states_x_previous** onedimensional array indicating for each transition leading into this state the number of the 'to' state to be used for deriving the special transition score for sequence $X$, if the transition is special. If an alternative transition is to be used, this number is the number of the 'to' state of the alternative transition. In the above example in which we are dealing with the match 5' splice site phase 1 state, the array element for the transition from the match exon state to the match 5' splice site phase 1 state is the number of the emit x 5' splice site phase 0 state as this is the 'to' state of the alternative transition from the match exon state to the emit x 5' splice site phase 0 state which is used for deriving the position specific

score of sequence $X$.

- **array<int> offset_for_previous_states_x** onedimensional array indicating for each transition leading into this state the offset in base pairs to be used if the transition is special. In the above example in which we are dealing with the ***match 5' splice site phase 1*** state, the array element for the transition from the ***match exon*** state to the ***match 5' splice site phase 1*** state is one as this is the offset between the alternative transition from the ***match exon*** state to the ***emit x 5' splice site phase 0*** state which is used when dealing with sequence $X$ and the transition from the ***match exon*** state to the ***match 5' splice site phase 1*** state.
- **array<int> numbers-of_from_child_states_y_previous**

  same as **array<int> numbers-of_from_child_states_x_previous**, but for dealing with input sequence $Y$
- **array<int> numbers-of_to_child_states_y_previous**

  same as **array<int> numbers-of_to_child_states_x_previous**, but for dealing with input sequence $Y$
- **array<int> offset_for_previous_states_y**

  same as **array<int> offsetforprevious-states3** , but for dealing with input sequence $Y$

### 6.4.3 The Sequence class

In its most fundamental form, a **Sequence** object consists **of:**

- **Letter* sequence** the sequence of symbolic letters
- **int length_ofsequence** the length of the sequence
- **Sequence-Type sequence-type** the type of the sequence
- **int orientation** the orientation of the sequence
- **int start_position** and **end-position,** the start and end positions of the sequence
- **char* i d** the name of the sequence

### `Sequence` objects for use with special emissions

If the input sequence is to be used with a pair HMM whose states have special emissions, the private variables of the **`Sequence`** object have to be set up accordingly, refer to Section **6.4.2** to see how special emissions are dealt with in the **`Pairhmm-State`** class.
The private variables which store information on special emissions are:

- **`int number-ofspecial-emissions`** the number of states for which special emissions have to be implemented. In general, this is not the number of states in the pair HMM that have special emissions, but a smaller number of states from which the special emissions of all states with special emissions are derived. In the case of PROJECTOR, **`int number-ofspecial-emissions`** is **22,** but the number of states with special emissions is **52** (all non silent states of the pair HMM).
- **`array<int> indices-ofspecial-emissions`** one-dimensional array which assigns an index to every implemented state with special emissions. This index is used as the first of two indices (the second index indicating the position within the sequence) for the arrays **`array<Prob> posterior_probs_for_special_emissions`** and **`array<Score> scores--for_special_emissions`** to look up the value of the corresponding posterior probability or score.
- **`array<Prob> posterior_probs_for_special_emissions`** two-dimensional array with the posterior probabilities for all implemented states with special emissions (first index) and all sequence positions (second index), if the posterior probabilities are to be used instead of scores
- **`array<Score> scores_for-special_emissions`** two-dimensional array with the scores for all implemented states with special emissions (first index) and all sequence positions (second index), if scores are to be used instead of posterior probabilities

### `Sequence` objects for use with special transitions

If the input sequence is to be used with a pair HMM which has special transitions, the private variables of the **`Sequence`** object have to provide the information needed by the states which have special transitions, see Section **6.4.2.**
The private variables which store information on special transitions are:

- int number-of special-transitions the number of special transitions which are implemented. As for special emissions, this is generally not the number of special transitions in the pair HMM, but a smaller number of special transition from which the information of all remaining special transitions can be derived, see Section **6.4.2** for more information. For DOUBLESCAN, the number of special transitions is **25,** but they derive their information from only six special transitions (three for ***each*** sequence) and the information on only these three transitions has to be provided by the Sequence object of each input sequence.
- array<int> indices-of -special-transitions two-dimensional array which assigns an index to every implemented special transition. This index is used to refer to the transition within other arrays (arrays array<Prob> posterior_probs_for_special_-transitions, array<Prob> priors-of special_transitions and array<Score>scores--for-special-transitions). The fist index of this array is the state number of the 'from' state of the special transition and the second index the state number of the 'to' state of the special transition and the return value is the index which is to be used to refer to that transition within the previously mentioned arrays.
- array<Prob> posterior_probs_for_special_transitions two-dimensional array with the posterior probabilities for all implemented special transitions (first index) and all sequence positions (second index), if the special transitions are to be used with posterior probabilities rather than with priors and scores.
- array<Prob> priorsnf -special-transitions two-dimensional array with the priors for **all** implemented special transitions (first index) and all sequence positions (second index), if the special transitions are to be used with priors and scores rather than with posterior probabilities. Generally, the priors for special transitions can be dependent on the position within the sequence. To name an example, the value of the prior for the special transition between the match **exon** and the emit **x 5'** splice site phase *0* state in the pair HMM underlying DOUBLESCAN and PROJECTOR depends on whether this is a consensus GT or a non-consensus GC splice site (see Table D.1 and Figure **2.4).**
- array<Score> scores_for_special_transitions two-dimensional array with the scores for **all** implemented special transitions (first index) and all sequence positions (second index), if the special transitions are to be used with priors and scores rather than with posterior probabilities.

## 6.5 Alignment algorithms

The public functions of the **Pairhmm** class provide several algorithms by which the pair HMM can be used to analyse data.

### 6.5.1 The Viterbi algorithm

The Viterbi algorithm in its original form [Vit67] was mainly implemented to verify the other algorithms. It calculates the optimally scoring state path for the given pair HMM and two input sequences.

### 6.5.2 The Hirschberg algorithm

The Hirschberg algorithm, see Section **1.5.2** and Figure **1.8,** takes two **Sequence** objects and an integer value **(int max-area) as** the input and calculates the optimally scoring state path for the given pair HMM and the two input sequences in linear memory and quadratic time dependence. The value of **max-area** indicates the maximum size (in the two sequence dimensions) of sub-matrices which are directly calculated by the Viterbi algorithm. The smaller this value, the more iterations have to be performed within the Hirschberg algorithm before ***each*** sub-matrix is small enough to be calculated using the Viterbi algorithm.
As the Hirschberg algorithm works internally with two pair HMMs, the original pair HMM and the mirrored model of the original pair HMM, the mirrored version has to be provided **as an**-other input parameter. It is created by the public **Pairhmm** function **`int get_mirrored(void)`** before the Hirschberg algorithm is called.

### 6.5.3 The Stepping Stone algorithm

The Stepping Stone algorithm, see Section **2.4** and Figure **2.5,** takes two **Sequence** objects and a list of $(x, y)$ coordinates **as** input which are simultaneously ordered in their $x$ and y coordinates and derives the highest scoring state path in the thus restricted sub-space of the Viterbi matrix. The input values, **`int x_margin`** and **`int y_margin`**, indicate the size of the overlap in the two sequence dimensions that two adjacent sub-matrices shall have. The input parameter **int max-area *is*** the same **as for** the the Hirschberg algorithm. If a sub-matrix is smaller than its value, it is directly calculated by the Viterbi algorithm during the traceback process. Otherwise, it is calculated using a special variant of the Hirschberg algorithm.

As opposed to the previously described implementation of the Hirschberg algorithm whose calculation is initialised by the constraint that every state path has to start in the begin state $s = 0$ at sequence positions $(x, y) = (1, 1)$, the calculation within this variant of the Hirschberg algorithm is initialised by a small sub-matrix of already pre-calculated values (this sub-matrix corresponds to the volume by which the current sub-matrix overlaps its lower left neighbouring sub-matrix, see Figure 2.5). The internal use of the Hirschberg algorithm is the reason why the mirrored model of the original pair HMM has to be provided as input to the Stepping Stone algorithm.