Multiple alignment using hidden Markov models

Seminar Hot Topics in Bioinformatics

Jonas Böer

Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany,
jonas.boeer@student.kit.edu

Abstract. This seminar report covers the paper "Multiple alignment using hidden Markov models" by Sean R. Eddy. In the introduction, I describe why it may be desireable to use hidden Markov models (HMMs) for sequence alignment and put this method into context with other sequence alignment methods. I give an introduction on the theory of HMMs and explain the basic algorithms for solving problems with HMMs. Then I present the implementation by Eddy for sequence alignment. He reports results that are not yet comparable to state of the art methods, but still look promising. Finally, I present some further applications for HMMs in sequence alignment.

1 Introduction

With the progress of sequencing methods, more and more genome and protein sequence data becomes available. To work with the data and to extract meaningful information, it is not only important to examine the sequences individually, but to compare sequences and learn about their relationships. With this knowledge, one can for example make assumptions about the shared evolutionary history of multiple species or predict the structure and the function of not yet described proteins [4,14].

Biological sequences change over time as evolution progresses. Not only may characters be replaced, they may also be deleted or new characters may be added to the sequence. Insertions and deletions lead to sequences with the same ancestral history having different lengths. Hence, before two sequences can be compared, an *alignment* between them must be found. Finding an alignment means that two sequences are brought to the same length by inserting gap characters at appropriate places so that characters with the same evolutionary history are found in the same column [4].

Finding good alignments between multiple sequences is a complex task and has proven to be NP-complete [17]. Therefore, faster heuristic approaches have been proposed to tackle this problem. These methods often have already been explored and implemented in other applications. One promising method originates from signal processing: Hidden Markov models (HMMs). This statistical method rose to great success in speech processing due to its robustness concerning pronounciation variations and noise [13].

This robustness can also be exploited for biological sequence analysis. Hidden Markov models have successfully been used for problems such as modeling DNA sequencing errors, protein secondary structure prediction as well as multiple sequence alignment [18].

This seminar report is about this application of hidden Markov models in multiple sequence alignment, especially based on one of the first papers that introduced this method, "Multiple alignment using hidden Markov models" by Sean R. Eddy, published in 1995 [7]. I will first give an introduction to HMM theory, giving an abstract view of the problems that can be solved with HMMs and present the basic algorithms to do so. Furthermore, I will describe how they can be applied to the problem of multiple sequence alignment and present the results that Eddy obtained with this method. Finally, I will present a number of methods where the HMM idea has been refined and used in other sequence alignment methods.

1.1 Related Methods

There have been many attempts to develop heuristic strategies for calculating a multiple sequence alignment. A few of these shall be presented here: ClustalW [15] is a software package that uses a guide tree and certain heuristics for weighing sequences, but is now considered deprecated. MUSCLE [8] and MAFFT [9] are two of today's most widely used sequence alignment methods and illustrate the variety of methods that can be employed for multiple sequence alignment. FSA [3] and PRANK [12], two more recent methods, are included here because they are widely used today and prominently include HMMs in their algorithms, showing that HMMs can be considered a successful method for sequence alignment.

ClustalW [15] was considered the state of the art at the time of Eddy's paper [7]. It first calculates pairwise alignments and uses these to construct a guide tree. The alignment is calculated from this guide tree, supported by certain heuristics: Sequences are weighted so that closely related sequences, representing redundant information, receive a lower weight and are therefore added earlier in the construction of the multiple alignment. Also, gap penalties are set depending on selected differences between two sequences. ClustalW has since been superseded by other methods.

MUSCLE [15] performs a progressive alignment in two stages. At first, a "draft" tree is calculated for the sequences using the Kimura distances between the sequences. The Kimura distance describes the evolutionary distance according to a substitution model [10]. Then, this tree is progressively refined by splitting the tree at one edge, calculating the two sub-trees' profiles and calculating their alignment. If this alignment is better than the previous one, it is kept and further improved, otherwise it is discarded. If the alignment has not yet converged, the tree is split at the next edge and the refinement procedure repeats.

MAFFT [9] uses the Fast Fourier Transform, a mathematical method also originating from signal analysis. Amino acid sequences are represented by numerical values describing their chemical and physical properties. The correlation between two sequences is then calculated, an operation which can be significantly sped up by using the aforementioned Fourier transform of the sequences' numerical representation. Homologous segments are then indicated by peaks in the correlation function graph. Groups of aligned sequences are described by a weighted average of their sequences, using a weighing function adapted from *ClustalW*.

FSA [3] first estimates the probabilities of the characters already being aligned, unaligned or gapped using a HMM. These probabilities are then sorted and used in a "sequence annealing" algorithm. Sequence annealing aligns characters one by one, starting with those probably improving the alignment the most.

PRANK [12] also performs a progressive alignment using a guide tree. Notably, this guide tree is treated as a phylogenetic tree and used to infer evolutionary information on the alignments. The alignment is then performed using a HMM that treats insertions and deletions differently under the assumption that they have different effects in evolutionary lineages.

2 Hidden Markov models

Hidden Markov models (HMMs) are a tool for the statistical analysis of sequences, especially for signal models. Their first widespread use was in speech recognition, although they have since been used in other fields as well [13].

HMMs offer a mathematical description of a system whose internal state is not known, only its output. A description of the current state of such a system would require describing all predecessor states. To calculate the properties of a state, all possible previous states would have to be also calculated and the computational complexity would increase exponentially with the number of predecessor states. To avoid this, only a *first-order Markov chain* is considered: The state of the system only depends on the previous state and the output only depends on the current states. This reduction of complexity allows the employment of efficient algorithms [13].

A common example describes an observer who is told the results of coin throws. The observer knows that there are two biased coins and there is a certain probability of the coin being switched after a throw, but doesn't know the biases or the probability of selecting the other coin. From the observations, the observer can still train a HMM that describes the results. This model might then be used to decide if a different observation was produced by the same set of coins. One possibility for such a model can be seen in figure 1.

When designing a HMM, the most important criteria are the number of states and the transitions between them. Some HMM architectures allow for transitions between all states, some only permit transitions in one direction (a "left-to-right model"), but any combination of states and transitions is possible. The HMM must be designed appropriately for the application. In the case of speech recognition, a left-to-right-model might be considered appropriate to describe a signal whose properties change over time [13].

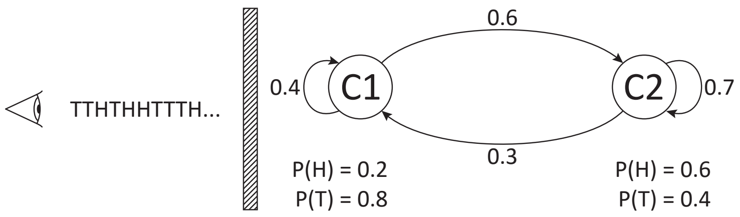

Fig. 1. Simple HMM to describe the observed results of two biased coins without knowing the coins' properties or when the other coin is selected.

Formally, a HMM can be described as a tuple $M = \{S, |a_{ij}|, X, b_i(x), \pi_i\}$:

- S are the model's states.
- $|a_{ij}|$ is a matrix describing the transition probabilities.
- X is the output alphabet.
- $b_i(x)$ describes the output probability for a letter x in state i.
- π_i is the probability to start in state i.

The output alphabet may also be continuous, requiring probability density functions for the output probabilities. In the context of this paper, the only used output alphabets are discrete, such as DNA letters or amino acids.

2.1 HMM applications

To analyse and describe data with HMMs, three major problems must be solved. First of all, what is the *probability of an observed sequence according to a given model*? This can be calculated using the *forward algorithm* and the result can be used, among others, to decide which model in a collection of HMMs best describes an observation.

The next problem is to find the *most probable state sequence* to have emitted an observation. This state sequence can be used to determine the process which has caused the observation and is calculated with the *Viterbi algorithm*. In speech recognition, this process would be the actual process of saying a word, so this algorithm gives the user the knowledge what was said according to the model.

Finally, the third problem is to *optimize the model*, to maximize the probabilities of given observations. The best-known method for HMM training is the Baum-Welch training algorithm, but Eddy has used it for comparison only. Instead, he has trained the HMM with the *Viterbi training algorithm*. It has worse convergence properties but iterates much faster [2] and I will only describe this method.

2.2 The forward algorithm

The forward algorithm [13], as well as the Viterbi algorithm, make use of the fact that a HMM's state only depends on the previous state. They both use the *dynamic programming* method to efficiently calculate the result.

In formal terms, the forward algorithm calculates the probability of being in a state i at time t and having emitted the output $o_1 \dots o_t$. These probabilities are named the *forward variables.* By calculating the sum over the last set of forward variables, one obtains the probability of the model having emitted the given sequence.

$$\begin{aligned}
\text{Initialization:} & \quad \text{for } 1 \leq i \leq N \\
F_1(i) &= \pi_i b_i(o_1) \\
\text{Induction:} & \quad \text{for } 2 \leq t \leq T, 1 \leq i \leq N \\
F_t(i) &= \left[\sum_{j=1}^{N} F_{t-1}(j) a_{ji} \right] b_i(o_t) \\
\text{Termination:} & \\
P(O) &= \sum_{i=1}^{N} F_T(i)
\end{aligned}$$

2.3 The Viterbi algorithm

The Viterbi algorithm [13] works similarly to the forward algorithm. As the goal is to get the most likely path through the HMM for a given observation, only the most likely transition from a previous state to the current one is important. Therefore, the summation from the forward algorithm is replaced by a search for the maximum value and a backwards pointer to the most likely predecessor state is saved.

$$\begin{aligned}
\text{Initialization:} & \quad \text{for } 1 \leq i \leq N \\
V_1(i) &= \pi_i b_i(o_1) \\
\text{Induction:} & \quad \text{for } 2 \leq t \leq T, 1 \leq i \leq N \\
V_t(i) &= \max_{1 \leq j \leq N} \{ V_{t-1}(j) a_{ij} b_i(o_t) \} \\
W_t(i) &= \arg\max_{1 \leq j \leq N} \{ V_{t-1}(j) a_{ij} \}
\end{aligned}$$

From the most probable finish state $\arg\max\{V_T(i)\}$, the most likely path through the model and therefore the most likely state sequence can then be constructed by following the backwards pointers $W_T \dots W_2$.

2.4 Viterbi training

As mentioned before, the Baum-Welch training is relatively slow, as it attempts to optimize all possible paths through an HMM for a given observation. In contrast, the Viterbi training algorithm [2] only attempts to optimize the most likely path associated with a sequence, sacrificing accuracy for speed.

The Viterbi training algorithm requires a representation of the HMM's parameters as normalized exponentials. This representation leads to an automatical normalization of the parameters and ensures that transmission and emission probabilites never become zero. The parameters to be modified are now v_{ix} for the emission probabilities and w_{ij} for the transmission probabilities.

$$\begin{aligned} b_i(x) &= \frac{e^{v_{ix}}}{\sum\limits_{x^\dagger \in X} e^{v_{ix^\dagger}}} \\ a_{ij} &= \frac{e^{w_{ij}}}{\sum\limits_{j^\dagger \in S} e^{w_{ij^\dagger}}} \end{aligned}$$

With this representation and a given path through the model, the actual training can be performed along the following equations:

$$\begin{aligned} \mathit{\Delta} v_{ix} &= \eta(E_{ix} - b_i(x)) \\ \text{with} \quad E_{ix} &= \begin{cases} 1 & \text{if } x \text{ has been emitted in state } i \\ 0 & \text{otherwise} \end{cases} \\ \text{and} & \\ \mathit{\Delta} w_{ij} &= \eta(A_{ij} - a_{ij}) \\ \text{with} \quad A_{ix} &= \begin{cases} 1 & \text{if the transition from } i \text{ to } j \text{ was used} \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

While Baum-Welch is an Expectation-Maximization algorithm, the Viterbi training is more similar to a gradient descent algorithm. Its learning rate is controlled with the parameter η, although Eddy makes no mention of the parameter values he used for training [7].

3 Multiple alignment with HMMs

3.1 Adapting HMMs for Bioinformatics

To use HMMs in biological sequence analysis, Eddy uses a HMM architecture that employs designated states for matches, insertions and deletions. While *match states* and *insert states* emit symbols as expected, *delete states* have the special characteristic that they don't emit a symbol at all and can be used to skip consensus positions in the profile. He also adds designated states signifying the beginning and the end of the sequence, also without any output. The aforementioned algorithms can be used for this architecture with only minor modifications:

The HMM is forced to terminate in a designated end state. This can be done in the Viterbi algorithm by not selecting the most likely state at the end but the designated end state and backtracking along the backwards pointers as usual. Delete states can be treated like other states with an emission probability $b_i(\epsilon) = 1$, ϵ signifying "no output" [11]. There are also implementations of the algorithms where the delete states' probabilities are calculated from the current state so that the algorithms have to perform multiple iterations until the probabilities have converged [2].

The common way to visualize these HMMs is by representing match states with squares, insert states with diamonds, and delete states (including the beginning and end states as "mute" states) with circles [6]. An example for such a HMM structure can be seen in figure 2.

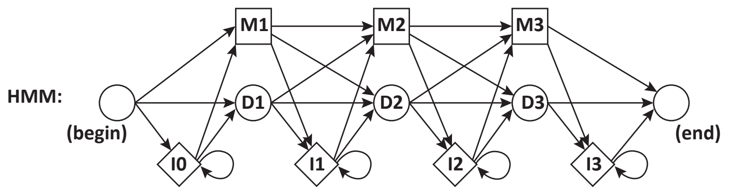

Fig. 2. A typical HMM structure for multiple alignment (adapted from [7])

3.2 Training HMMs with simulated annealing

To avoid running into local optima, Eddy extends the HMM methods with the simulated annealing technique. Simulated annealing uses a simulated temperature factor kT, also named the Boltzmann temperature. The higher kT is, the more randomized the alignments are: $kT = \infty$ results in all alignments being sampled with the same probability, $kT = 0$ means that the most likely alignment is always selected.

With simulated annealing, the aim is to probabilistically sample an alignment Q of a sequence O, given a model M and the Boltzmann temperature kT:

$$P(Q) = \frac{P(Q|O,M)^{\frac{1}{kT}}}{\sum_{all_{Q^\dagger}} P(Q^\dagger, O|M)^{\frac{1}{kT}}}$$

To implement simulated annealing in the forward algorithm, the model's parameters are replaced with exponentiated parameters. The parameters used in this variation are $\hat{\pi}_i = \pi_i^{\frac{1}{kT}}$, $\hat{a}_{ij} = a_{ij}^{\frac{1}{kT}}$ and $\hat{b}_j(x) = b_j(x)^{\frac{1}{kT}}$ so that they include the current temperature in the algorithm.

$$\begin{aligned}
\text{Initialization:} & \quad \text{for } 1 \le i \le N \\
F_1(i) &= \hat{\pi}_i \hat{b}_i(o_1) \\
\text{Induction:} & \quad \text{for } 2 \le t \le T, 1 \le i \le N \\
F_t(i) &= \left[\sum_{j=1}^{N} F_{t-1}(j) \hat{a}_{ji} \right] \hat{b}_i(o_t) \\
\text{Termination:} & \\
Z &= \sum_{i=1}^{N} F_T(i)
\end{aligned}$$

Z now describes the probability of the observation according to the model and the current temperature. As Eddy wants to use the Viterbi training method, he needs a path along which he can optimize the model. With the exponentiated forward variables, a probabilistic traceback can be performed: In contrast to the Viterbi algorithm's backwards pointers, he selects one of the possible predecessor states according to their probability of being the actual predecessor state:

$$\begin{aligned}
\text{Initialization:} & \\
P(q_T) &= \frac{F_T(q_T)}{Z} \\
\text{Induction:} & \quad \text{for } T \ge t \ge 2 \\
P(q_{t-1} | q_t) &= \frac{F_{t-1}(q_{t-1}) \hat{a}_{q_{t-1}, q_t}}{\sum_{i=1}^{N} F_{t-1}(i) \hat{a}_{i, q_t}}
\end{aligned}$$

The path which describes the suboptimal alignment is then reused to further improve the model with the Viterbi training method to increase the likelihood of the model for this path. As the training starts with a high temperature kT, the alignments are very randomized at first, with well-determined motifs freezing first. The author expresses his hope that the simulated annealing alleviate the problem of running into local optima.

For simulated annealing, Eddy starts with a kT value of 5 and reduced it by 5% at each iteration. If the model didn't change for more than an arbitrary threshold after an iteration, the algorithm is assumed to have converged and the calculation is stopped. On the other hand, if reducing kT brought its value below 0.1, they switched to the standard Viterbi algorithm and Viterbi training, effectively setting $kT = 0$ to avoid problems with floating point precision for very small numbers.

The resulting HMM is called consensus HMM [7] or, especially when it is used to describe the profile of a number of sequences, a profile HMM [6]. An example for a HMM trained with a simple dataset is given in figure 3.

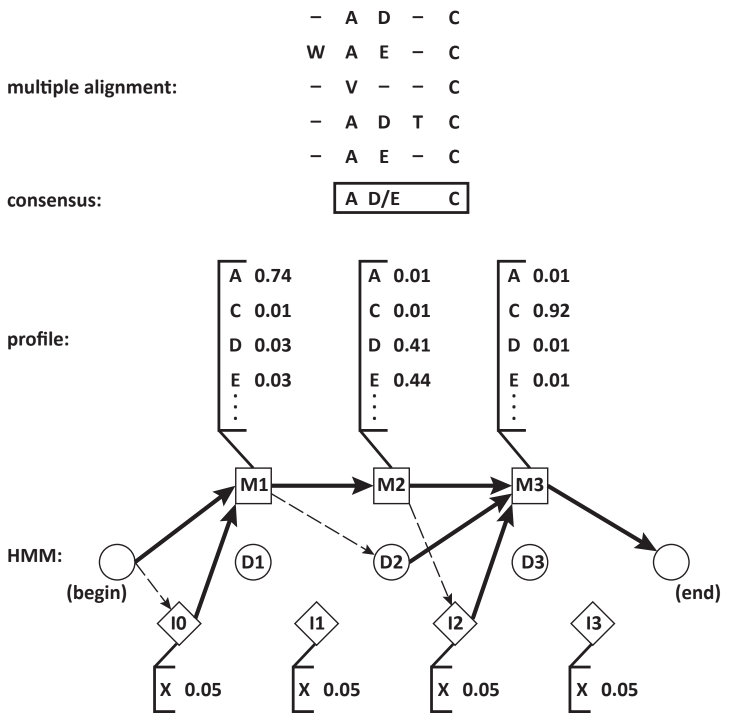

Fig. 3. The HMM from figure 2 trained with some sequences. Higher transition probabilities are indicated by bolder arrows. (adapted from [7])

4 HMM evaluation

Eddy primarily aims to address the following questions:

– Do the standard HMM training methods run into local optima?
– Does simulated annealing alleviate this problem?
– Is there a correlation between mathematically good optima (higher HMM scores) and good alignments?
– How well do the HMM methods work compared to the state of the art?

He developed a software named HMMER employing the methods described previously. To test its performance, he used real-world data from ten protein families: Alpha amylase, (multi) EF hand, cytochrome C, EGF domain, globin, homeodomain, Ig light chain V, insulin, protease inhibitors and kringle modules. As ground truth data, trusted alignments by Šali and Overington [16] were used, although the author advises that these do not neccessarily reflect the absolute truth and may not reach the quality of an expert's manual alignment.

The quality of a HMM's parameter estimation depends on the amount of training data. As the protein families from the test data only consist of 2 to

23 sequences, additional homologues from the SWISS-PROT database [1] were added to the test data. Every protein familiy was thusly extended to 100 sequences.

The ten test sets were aligned with different methods to be able to evaluate HMMER's performance. The alignment methods were:

– HMM training with the Viterbi training algorithm ("Vit" in table 1 and the following figures)
– HMM training with Baum-Welch expectation maximization ("B-W")
– HMM training with simulated annealing as described in section 3 ("SA")
– "Best of 10" simulated annealing runs according to the HMM score ("SA10")
– ClustalW, the state of the art software at the time ("CW")

Two measures were calculated from the resulting alignment: The HMM log-odds score as the HMM score and the alignment accuracy. For HMM scoring, a HMM is constructed from the resulting alignment and the combined log-odds score for the sequences in its family was calculated. For a better comparison, its representation in bits (log base 2) is used by Eddy. To make the results from ClustalW comparable, such an HMM is also constructed from the ClustalW alignment and its alignment is scored with this model.

The alignment accuracy compared to the trusted alignment is calculated by counting the number of correctly aligned symbol pairs divided by the total number of aligned symbol pairs in the trusted alignment. Eddy also tried to compare only a number of key columns, but dismissed the approach because the results correlated with the overall results and because he considered the key column selection process to be arbitrary.

Family	Vit	B-W	SA	SA10	CW
EGF domain	53.07	54.96	55.46	57.17	47.36
	83%	74%	80%	88%	75%
globin	203.98	237.33	250.82	255.93	253.31
	38%	69%	81%	75%	93%
homeodomain	132.74	131.71	133.58	134.31	133.91
	68%	91%	50%	61%	96%

Table 1. Selected alignment HMM scores (first line) and alignment accuracies (second line) (from [7])

The results from three of the protein families shall be discussed in more detail: The EGF domain, the globins and the homeodomain. There appear to be local optimum problems, especially with the naïve Viterbi training algorithm. It often leads to low alignment and likelihood scores. Baum-Welch training has significantly less problems with local optima, but still doesn't reach the alignment quality ClustalW offers. One family where this can be seen well are the globins, visualized in figure 4.

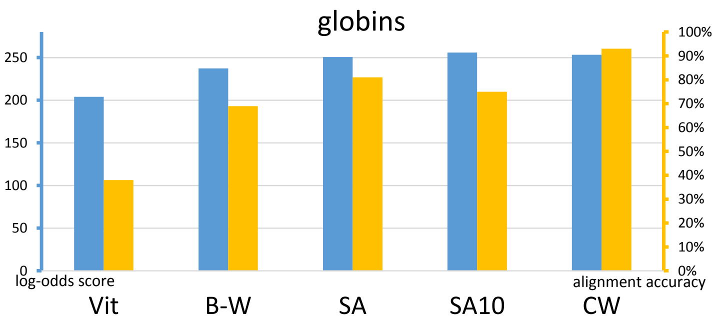

Fig. 4. The scores of the globin family

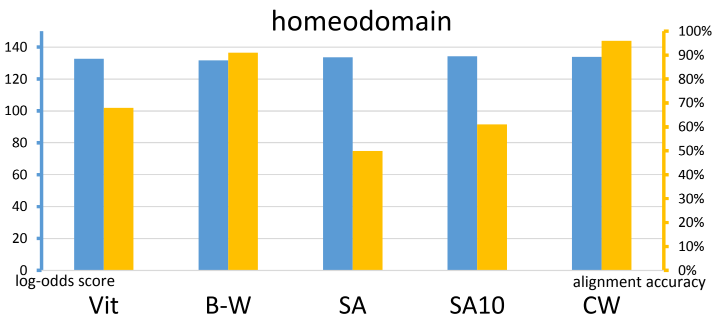

Fig. 5. The scores of the homeodomain family

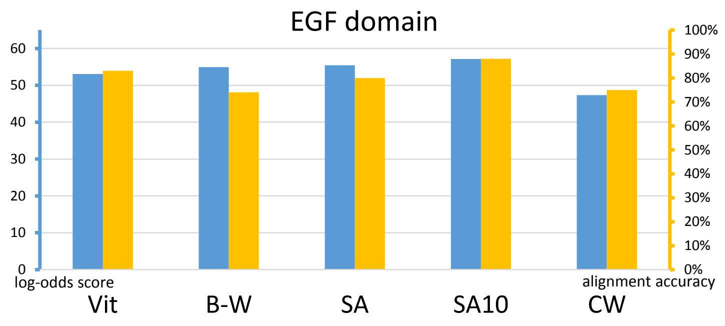

Fig. 6. The scores of the EGF family

Better HMM scores do correspond with a better alignment quality, although this is not always the case, see the results from the homeodomain proteins (figure 5). Eddy proposes that a better correspondence between the two measures might be reached by including prior biological information in the HMMs, such as amino acid substitution probabilities.

The author emphasizes that the results from the EGF domain and the EF hand are significantly better with HMMs than the ClustalW results (figure 6) These families have the largest amount of insertions and deletions in the set. The authors argue that HMMs have a consistent theory for insertions and deletions and therefore are well suited to deal with this family.

Still, in many cases ClustalW reaches the best results. Eddy assumes that this is because ClustalW was designed with careful heuristics, e.g. weighing certain sequences differently to avoid a bias towards dominant subfamilies.

5 Applications and extensions

5.1 Searching with HMMs

One of the most common applications for HMMs in bioinformatics is searching for homologous sequences. The *profile HMM*, describing the profile of a group of sequences, can be used to score the similarity of other sequences. This can also be done with the implementations of the Viterbi and the forward algorithm given in this paper.

Durbin et al. advise that the usual log-likelihood score should not be used in this case, as it depends on the sequence length. The log-odds score does not present this problem and returns a lower variance of the non-matches' score. The algorithms can be modified to directly output the log-odds score [4].

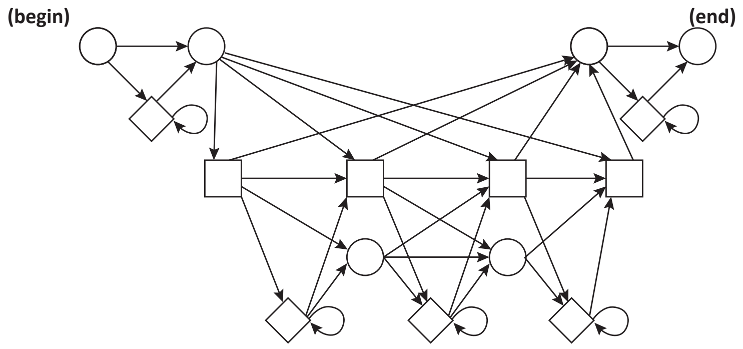

Fig. 7. HMM structure for a local alignment search (adapted from [4])

A local alignment, which allows only a substring of the query to match, can also be implemented with the given HMM structures. The HMM simply has to be extended with two additional transition states that allow transitions from or to every match state (figure 7) [4].

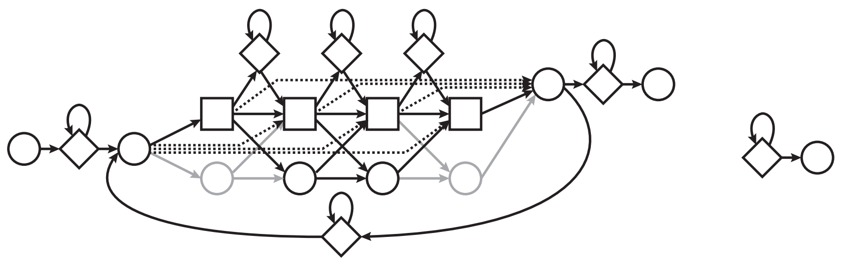

Fig. 8. Left: "Plan 7" HMM structure for HMMER2, Right: Null model for log-odds scoring (adapted from [5])

Searching with HMMs may also require more sophisticated HMM architectures. Eddy constructed the "Plan 7" HMM architecture for version 2 of the HMMER software (figure 8, left). Plan 7 does not allow for transitions from insert to delete states and vice versa. To ensure that at least one match state is visited at every cycle through the model during the search, the first and last delete states (displayed in grey) are removed after training. HMMer2 also uses a "null model" (figure 8, right) as a reference to calculate log-odds scores, with the single insert state's emission probabilities being the average amino acid frequencies according to SWISSPROT34 or a uniform distribution for nucleotides [5].

5.2 Pair HMMs

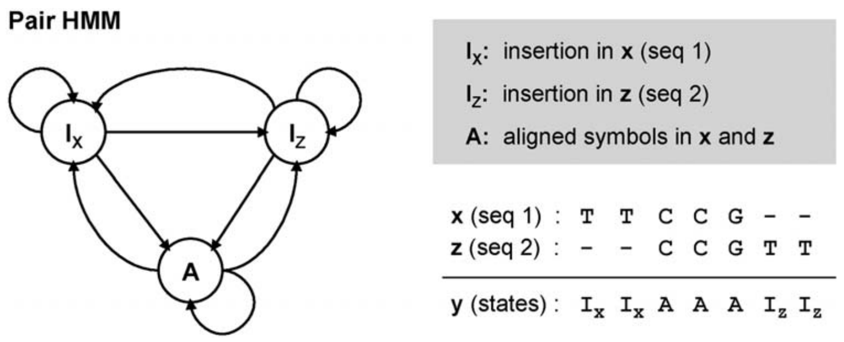
Pair HMM
I_X
I_Z
A
I_X: insertion in **x** (seq 1)
I_Z: insertion in **z** (seq 2)
A: aligned symbols in **x** and **z**
x (seq 1) : T T C C G – –
z (seq 2) : – – C C G T T
y (states) : I_x I_x A A A I_z I_z

Fig. 9. Pair HMM (from [18]

Pair HMMs are an extension of HMMs that emit a pairwise alignment instead of a single sequence. They can be used to efficiently align two sequences including insertions and deletions [4]. This is especially useful when trying to align non-coding DNA sequences. Pair HMMs are especially used in algorithms that perform a progressive alignment, such as *PRANK* [12]. They also give an estimation of the posterior probability of each aligned colum which can be used as an indicator for the alignment quality at this position [18].

5.3 Context-sensitive HMMs

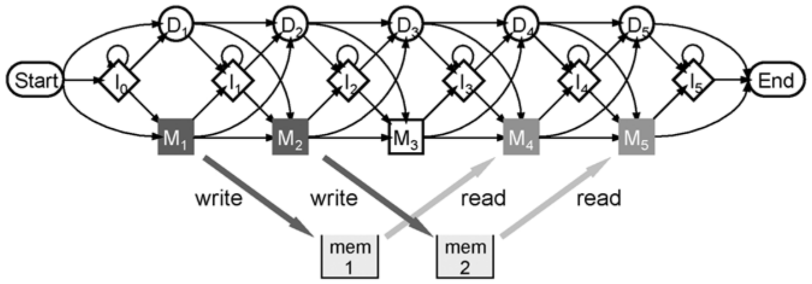

Fig. 10. Context-sensitive HMM (from [18]

The fact that HMMs don't have a "memory" is also a weakness. One application where access to information on previous states can be useful is RNA sequence alignment: The secondary structure of RNA molecules is determined by matching parts of the RNA sequence. Yoon et al. have developed the concept of *context-sensitive HMMs* (csHMMs) [18] These models are extended with a functionality similar to a pushdown automaton, giving certain match states the ability to either *push* to a stack or *pop* the topmost element from the stack. This simple memory enables the csHMMs to incorporate structural information in the matching. Figure 10 shows an example that might be used for RNA matching.

6 Conclusion

In 1995, HMMs were a promising method for sequence alignment, although they could not always compete with the state of the art. Still, this relatively unrefined approach worked well enough to inspire more research.

Today, methods that exclusively use HMMs are relatively rare, but HMMs are used as a part of many methods such as *FSA* or *PRANK*. Additionally, the theory of HMMs is still under development to extend their capabilities, making HMMs a versatile tool for multiple or pairwise sequence alignment.

One can expect that HMMs will continue to be an integral part of multiple sequence alignment as well as other fields of bioinformatics.

References

1. Bairoch, A., Boeckmann, B.: The swiss-prot protein sequence data bank, recent developments. Nucleic acids research 21(13), 3093 (1993)
2. Baldi, P., Brunak, S.: Bioinformatics: the machine learning approach. MIT press (2001)
3. Bradley, R.K., Roberts, A., Smoot, M., Juvekar, S., Do, J., Dewey, C., Holmes, I., Pachter, L.: Fast statistical alignment. PLoS Comput Biol 5(5), e1000392 (2009)
4. Durbin, R., Eddy, S.R., Krogh, A., Mitchison, G.: Biological sequence analysis: probabilistic models of proteins and nucleic acids. Cambridge university press (1998)
5. Eddy, S.R.: Hmmer user's guide (1998), `http://www.csb.yale.edu/userguides/seq/hmmer/docs/node11.html`
6. Eddy, S.R.: Profile hidden markov models. Bioinformatics 14(9), 755–763 (1998)
7. Eddy, S.R., et al.: Multiple alignment using hidden markov models. In: Ismb. vol. 3, pp. 114–120 (1995)
8. Edgar, R.C.: Muscle: multiple sequence alignment with high accuracy and high throughput. Nucleic acids research 32(5), 1792–1797 (2004)
9. Katoh, K., Misawa, K., Kuma, K.i., Miyata, T.: Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform. Nucleic acids research 30(14), 3059–3066 (2002)
10. Kimura, M.: A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. Journal of molecular evolution 16(2), 111–120 (1980)
11. Krogh, A., Brown, M., Mian, I.S., Sjölander, K., Haussler, D.: Hidden markov models in computational biology: Applications to protein modeling. Journal of molecular biology 235(5), 1501–1531 (1994)
12. Löytynoja, A., Goldman, N.: An algorithm for progressive multiple alignment of sequences with insertions. Proceedings of the National academy of sciences of the United States of America 102(30), 10557–10562 (2005)
13. Rabiner, L.R.: A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE 77(2), 257–286 (1989)
14. Russell, D.J.: Multiple sequence alignment methods. Springer (2014)
15. Thompson, J.D., Higgins, D.G., Gibson, T.J.: Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic acids research 22(22), 4673–4680 (1994)
16. Šali, A., Overington, J.P.: Derivation of rules for comparative protein modeling from a database of protein structure alignments. Protein Science 3(9), 1582–1596 (1994)
17. Wang, L., Jiang, T.: On the complexity of multiple sequence alignment. Journal of computational biology 1(4), 337–348 (1994)
18. Yoon, B.J.: Hidden markov models and their applications in biological sequence analysis. Current genomics 10(6), 402–415 (2009)