

4.2 - Algorithms

Sébastien Lemieux

Elitra Canada Ltd.

lemieuxs@iro.umontreal.ca

slemieux@elitra.com



Objectives

- to understand a key class of algorithm in bioinformatics: the dynamic programming.
- to apply this class of algorithm to a variety of biologically relevant problems.
- to understand the relation between the theoretical formulation of the model and its implementation.

Historical application

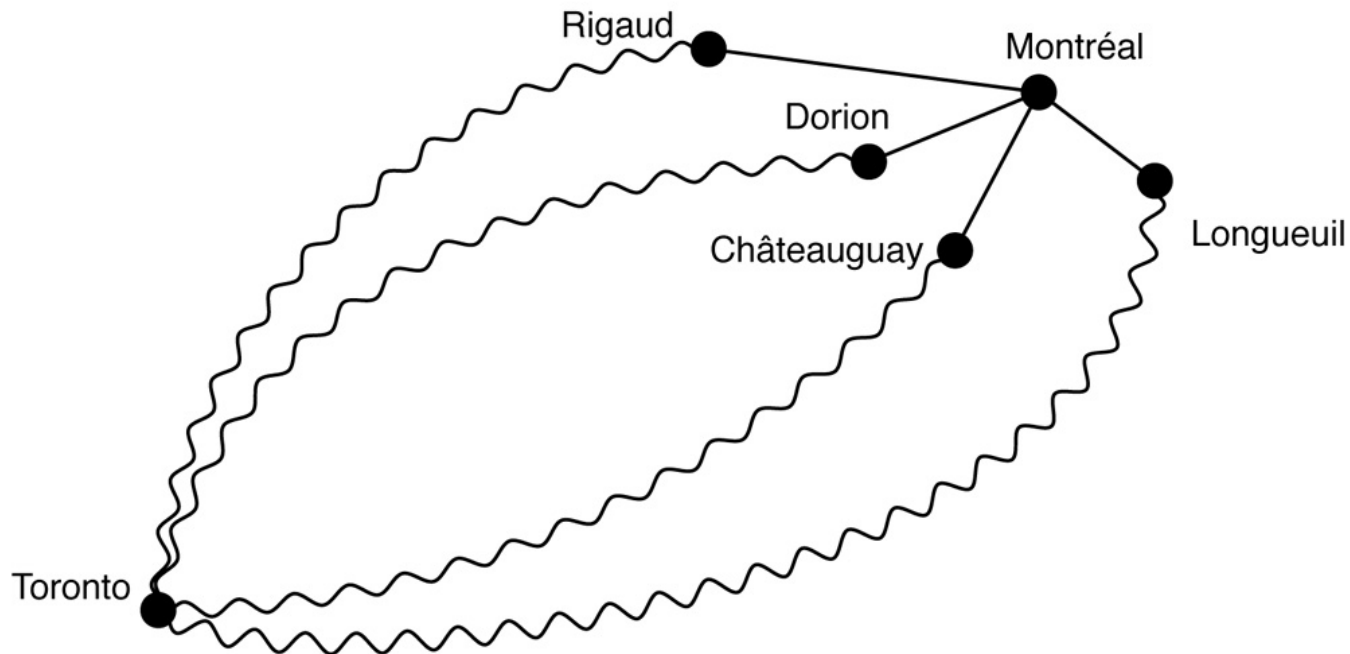
- In bioinformatics, dynamic programming is most known for its application to sequence alignment:
 - Smith-Waterman / Needleman-Wunsch.
- It was extended to do database searches and gene prediction.
- It has been extensively used for secondary structure prediction in RNA:
 - mfold (Zucker);
 - pknot (Rivas and Eddy).
- Hidden markov model (HMM) are closely related to dynamic programming:
 - Viterbi algorithm (see Durbin *et al.*)

Algorithmic description

- The principle of optimality: “in a sequence of choices, each subsequence must also be optimal.”
 - An example where it applies: sequence alignment.
 - With protein folding, it doesn't apply!
- If it applies, dynamic programming will provide the fastest exact algorithm!

Algorithmic description *(cont.)*

- An example: What is the shortest path from Toronto to Montréal?



Algorithmic description (cont.)

- With recursion:

$$D(j) = \min_{i \in V_j} D(i) + d_{ij}$$

where $D(i)$ is the length of the shortest route from Montréal to the city i , d_{ij} is the distance between cities i and j , and V_j represent the neighborhood of city j .

- With “memory functions”:
 - Create a table D with an entry for each city. Before computing $D(j)$, check in the table and if it is already there, just return that value.
- Iteratively:
 - In some situation, it is possible to order the entries in the table such that each entry only depends on the previous ones. Then you can compute each entry one after the other.

The sequence alignment problem

- Given two sequences **a** and **b**, find the best alignment considering that an insertion or deletion costs I and matching a_i and b_j costs $M(a_i, b_j)$:

- Alignment of ATGAGCGGG vs. GCGCTAGCG would return:

```
ATGAGC--GGG
--GCGCTAGCG
```

- The score quantifies the similarity between the two sequences.

Needleman - Wunsch

- Finds the best global alignment of two sequences using dynamic programming.

$$S(\mathbf{a}_i, \mathbf{b}_j) = \min \begin{cases} S(\mathbf{a}_{i-1}, \mathbf{b}_{j-1}) + M(a_i, b_j) \\ S(\mathbf{a}_{i-1}, \mathbf{b}_j) + I \\ S(\mathbf{a}_i, \mathbf{b}_{j-1}) + I \end{cases}$$

	-	A	D	F	G
-	0				
A					
S					
D					
F					?

- There are two formulations depending of the semantic of M and I. Here we are minimizing the cost of the alignment, we could maximize a score instead.
- Sometime erroneously referred as Smith-Waterman, which is a variation of the Needleman - Wunsch algorithm that identifies the best **local** alignment.

... with affine gap costs

- It was proposed that the opening of a new gap should cost more than extending an existing one: $C(x) = g + hx$

where $C(x)$ is the cost of a gap of length x .

- The use of three matrices becomes necessary:

$$\begin{aligned} M(\mathbf{a}_i, \mathbf{b}_j) &= \min \begin{cases} M(\mathbf{a}_{i-1}, \mathbf{b}_{j-1}) + M(a_i, b_j) \\ I(\mathbf{a}_{i-1}, \mathbf{b}_{j-1}) + M(a_i, b_j) \\ D(\mathbf{a}_{i-1}, \mathbf{b}_{j-1}) + M(a_i, b_j) \end{cases} \\ I(\mathbf{a}_i, \mathbf{b}_j) &= \min \begin{cases} M(\mathbf{a}_{i-1}, \mathbf{b}_j) + g + h \\ I(\mathbf{a}_{i-1}, \mathbf{b}_j) + h \\ D(\mathbf{a}_{i-1}, \mathbf{b}_j) + g + h \end{cases} \\ D(\mathbf{a}_i, \mathbf{b}_j) &= \min \begin{cases} M(\mathbf{a}_i, \mathbf{b}_{j-1}) + g + h \\ I(\mathbf{a}_i, \mathbf{b}_{j-1}) + g + h \\ D(\mathbf{a}_i, \mathbf{b}_{j-1}) + h \end{cases} \end{aligned}$$

Dynamic programming version

- Each matrix stores the best cost achieve for each subsequence assuming the alignment ends with a match (M), an insertion (I) or a deletion (D).
- Is the principle of optimality preserved?

M :

	-	A	D	F	G
-	0				
A					
S					
D					
F					?

I :

	-	A	D	F	G
-	0				
A					
S					
D					
F					?

D :

	-	A	D	F	G
-	0				
A					
S					
D					
F					?

HMM - crash course! *(cont.)*

- The Viterbi algorithm can be used to find the sequence of states that is the most likely to generate the observed sequence.
- The Viterbi algorithm is defined by the recurrence:

$$v_l(i + 1) = e_l(x_{i+1}) \max_k (v_k(i) \cdot a_{kl})$$

where $v_k(i)$ is the probability of ending in state k with observation i , $e_k(x_i)$ is the probability of observing x_i in state e_k and a_{kl} is the transition probability from state k to l .

- We can get rid of the “nasty” multiplications by seeking the log-probability of the sequence:

$$\log[v_l(i + 1)] = \log[e_l(x_{i+1})] + \max_k (\log[v_k(i)] + \log[a_{kl}])$$

- That's dynamic programming!

HMM - crash course! *(cont.)*

- The Viterbi algorithm is dynamic programming:

	b	b	b	r	b	b	b	b	r	b	r	r	r	r	r
B	0,9														
R	0,1														

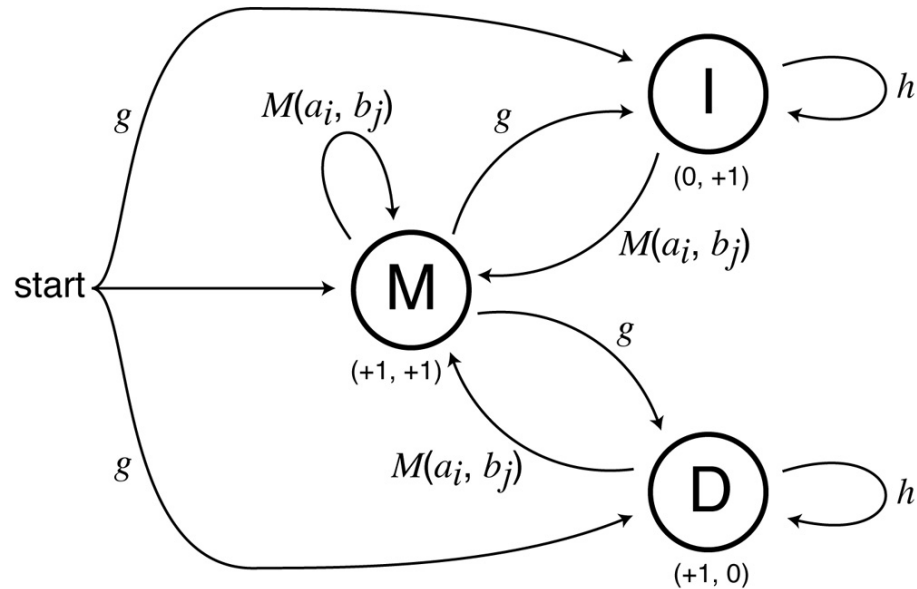
Alignment as a HMM

- Assume the following alignment is the observed sequence produced by a HMM:

```
ATGAGC --GGG  
--GCGCTAGCG
```

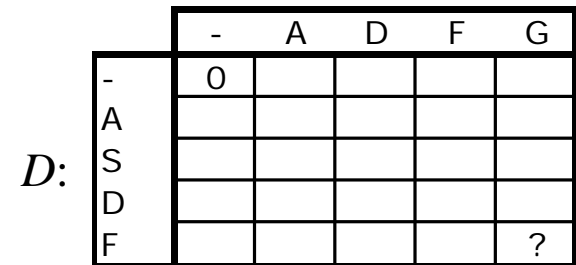
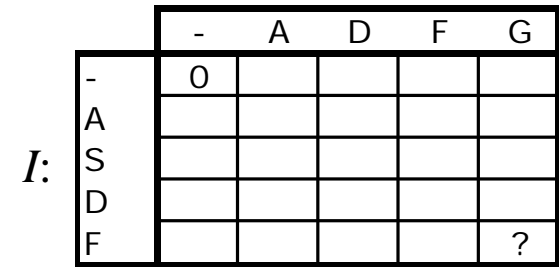
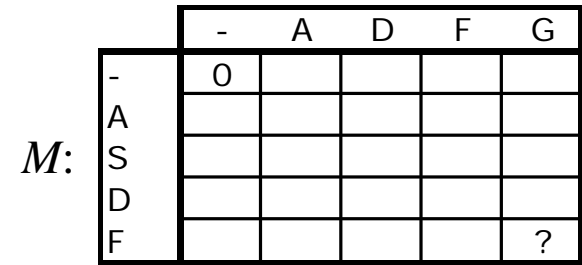
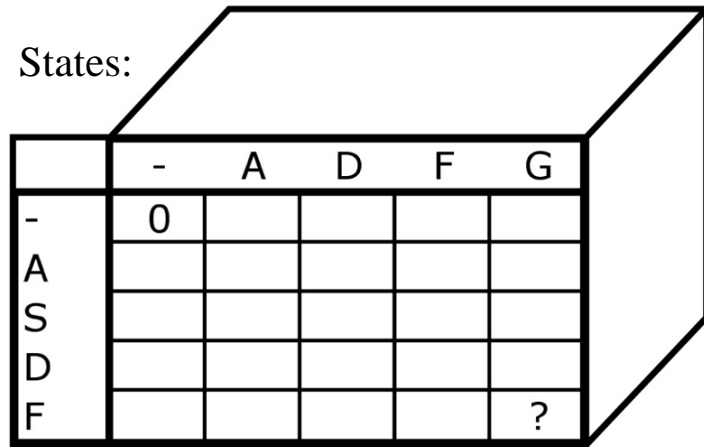
- As far as the Viterbi algorithm is concerned, scores would work the same way as log-probabilities.

Alignment as a HMM (cont.)



- The sequence produced corresponds to the alignment, it is in a two-dimensional space. The Viterbi matrix will have three dimensions: sequence **a**, sequence **b** and the three states.

Alignment as a HMM (cont.)



- Does it remind you something???

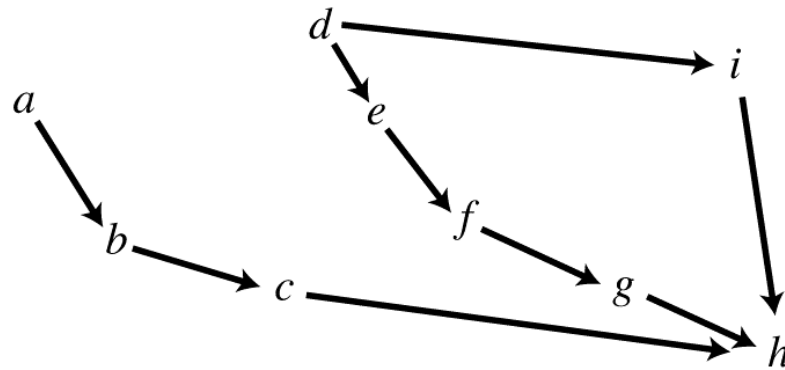
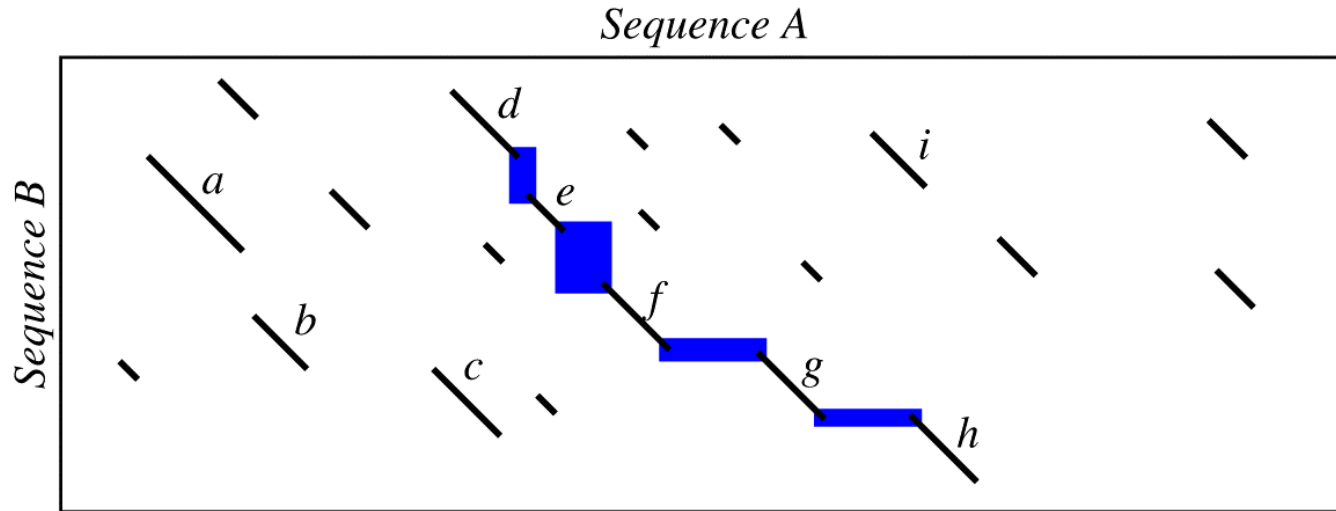
Alignment as a HMM *(cont.)*

- What is so special about HMM?
 - They are very flexible. It is easy to represent a complex sequence analysis problem using the HMM form. Examples: cDNA vs. gDNA alignment, DNA vs. protein, gene prediction, intron prediction, etc.
 - Their parameters can be learned. The most difficult task is often to parameterize the final algorithm. With HMM, parameters can be optimized from a set of observed sequences! (see Baum-Welch algorithm, Durbin *et al.*, p. 63)
- Their mathematical “flavor” scares a lot of people!

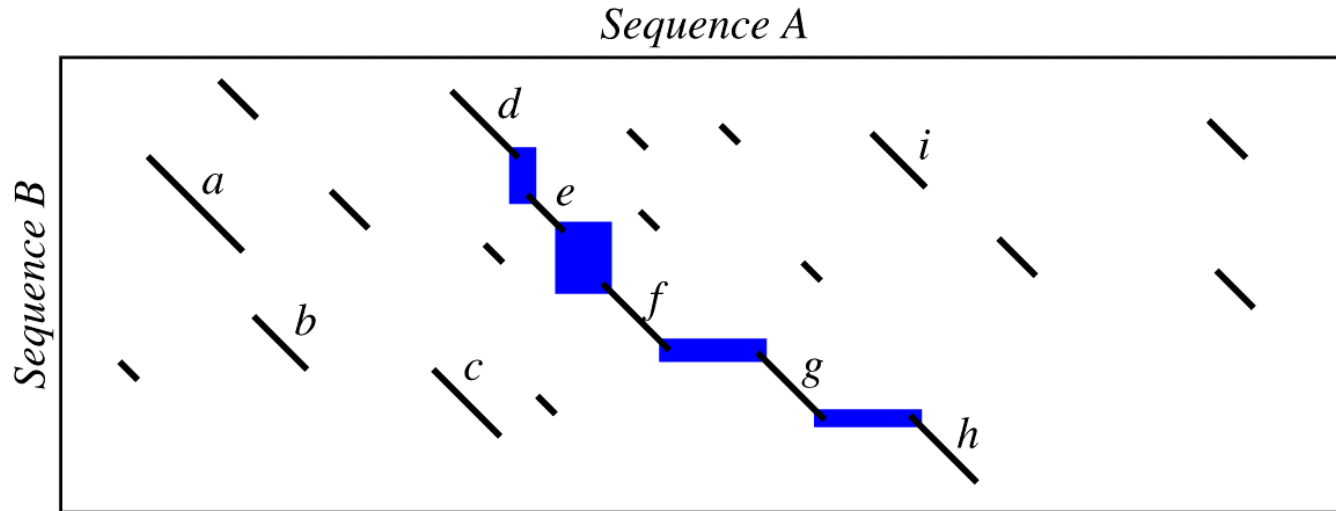
Linking matches

- Very fast algorithms are available to identify matches without insertions or deletions between two large sequences.
 - Aho-corasick engine: à la BLAST.
 - Dictionary: à la FASTA.
 - Suffix trees
- Linking these matches together to identify the most similar region is performed using dynamic programming.

Linking matches (cont.)



Linking matches (cont.)



$$S(j) = \max_i \begin{cases} S(i) + T_{ij} + M_j \\ 0 \end{cases} \leftarrow \text{Why?}$$

- $S(j)$: Optimal score up to match j .
- T_{ij} : Score of the transition between matches i and j . Should be negative, can be $-\infty$.
- M_j : Score of match j . Should be positive!

Some other classes of algorithms

- Greedy algorithms:
 - At each step the “best” choice is made and optimally is guaranteed.
- Probabilistic algorithms:
 - At each step choices are made randomly.
 - Sometime used to avoid a frequent worst case.
- Heuristic algorithms:
 - Optimality of the solution is not guaranteed.
 - Heuristics are often probabilistic: monte carlo.
 - Very useful for “difficult” problems or when the formulation is already an approximation.

References

- R. Durbin, S. Eddy, A. Krogh and G. Mitchison, *Biological Sequence Analysis : Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press, 1998.
- G. Brassard and P. Bratley, *Fundamentals of Algorithms*, Prentice Hall, 1996.