Computational Molecular Biology

Last updated: September 10, 2024

Hello! These lecture notes were originally created during the 2018 iteration of CS 181 (Computational Molecular Biology) taught at Brown University by Sorin Istrail. The original notes were provided by Shivam Nadimpalli, and future updates have been made by course TAs including Elliot Youth, Daniel Ben-Isvy, Sam Maffa, Iris Huang, and Hannah Beakley. Please email any comments or typos to cs1810tas@lists.brown.edu.

Contents

1 Sequence Alignment

(Sept. 10)

In this chapter of the course, we will take a look at those algorithms that revolutionized the field of computational biology, and laid down robust foundations for its future.

We will first look at the problem of global alignment, after which we will consider the statistical foundations of this topic, and then finally we'll study local and gap alignment. The algorithmic technique of dynamic programming will play a key role throughout. Let's get started.

1.1 Global Alignment

Consider two DNA sequences $X =$ `ATTACG` and $Y =$ `ATATCG`. What's the optimal way to *align* these two sequences?

We haven't formally defined what any of this means, but going off our intuition, we would guess that the fewer mismatches an alignment has, the better it is. It's easy to see that the best alignment for these two sequences, then, is:

```
ATTA-CG
|||||||
A-TATCG
```

The problem of *global alignment* of two sequences is that of finding an alignment for every residue in each sequence.

Some foreshadowing: we will soon be looking at *local* alignment, in which we look for the best alignment of any subsequences of the given sequences. The motivation behind this, as well as how it differs from global alignment, will be made clear then.

1.1.1 Formal setup

However, what if a `(T, -)` alignment is really bad for us for some reason? Or if we want to minimize the number of `(A, C)` alignments? We can do so by penalizing such undesirable alignments, and this can be done by choosing an appropriate scoring scheme.

Given two strings over a finite alphabet Σ, a *scoring scheme* is a function $\delta : \Sigma \times \Sigma \to \mathbb{R}$. In the case of DNA sequences, $\Sigma = \{\texttt{A}, \texttt{T}, \texttt{G}, \texttt{C}\}$, and in the case of protein sequences Σ would be the set of 20 amino acids.

It's usually convenient to represent a scoring scheme in the form of a table. For example:

	A	T	C	G
A	1	0	0	0
T	0	1	0	0
C	0	0	1	0
G	0	0	0	1

Figure 1. The Unitary or Diagonal Scoring Scheme.

We also introduce a related concept: the *gap penalty* is a function $\delta : \Sigma \to \mathbb{R}$ which tells you the cost for aligning a character with a gap (-). Gaps are also called *indels* by biologists, which stands for "insertion/deletion mutations".

Abuse of Notation: We will use δ to refer to the scoring scheme and the gap penalty.

Finally, suppose we have an alignment $x_1 \dots x_n$ and $y_1 \dots y_n$ where $x_i, y_i \in \Sigma \cup \{\texttt{-}\}$, then the *score* of the alignment is given by $\Sigma_{i=1}^{m} \delta(x_i, y_i)$.

For example, under the unitary scoring scheme (Figure 1) with 0 gap penalty, the alignment of X and Y given in §1.1:

```
A   T   T   A   -   C   G
|   |   |   |   |   |   |
A   -   T   A   T   C   G
-----------------------------
1 + 0 + 1 + 1 + 0 + 1 + 1  =  5
```

has a score of 5.

If we change the gap penalty to -1, then the score of the same alignment changes to 3. From this, it's clear that you can get different scores for the same alignment under different scoring schemes, and that the optimal alignment of two sequences depends on the scoring scheme being used.

Finally, we can formally state the problem of global alignment:

> The optimal global alignment of two sequences under a chosen scoring scheme and gap penalty is the alignment of the two sequences with maximum score.

Now, how would one find this optimal global alignment? One way is to enumerate all possible alignments of the given sequences, and pick the one with the best score. But we'll see in the homework that this process is computationally inefficient. The principle of *dynamic programming*, however, comes to our rescue, as we will see in §1.1.3.

1.1.2 The edit graph

Let Σ be a finite alphabet, and let $X = x_1 \dots x_m$ and $Y = y_1 \dots y_n$ be strings over Σ. To X and Y, we associate a directed graph termed the *edit graph* of X and Y.

The vertices of the graph are the vertices of the rectangular grid of size $(m+1) \times (n+1)$. By $v(i, j)$ we will mean the vertex in the grid corresponding to the $(i+1)^{\text{th}}$ row and the $(j+1)^{\text{th}}$ column.

Between the vertices, we have three kinds of edges:

1. A *gap-in-Y* edge: for $1 \le i \le m, 1 \le j \le n$, we have an edge $v(i-1, j) \to v(i, j)$. This edge corresponds to $(x_i, \texttt{-})$ in the alignment.
2. A *gap-in-X* edge: for $1 \le i \le m, 1 \le j \le n$, we have an edge $v(i, j-1) \to v(i, j)$. This edge corresponds to $(\texttt{-}, y_i)$ in the alignment.
3. An *alignment* edge: for $1 \le i \le m, 1 \le j \le n$, we have an edge $v(i-1, j-1) \to v(i, j)$. This edge corresponds to (x_i, y_j) in the alignment.

What exactly does this graph tell us? This will become clear through the following example:

Example 1.1.1. Suppose we want to construct an edit graph for two sequences `AC` and `AGC`. We set up a grid of dimensions 3×4 as shown in the picture below:

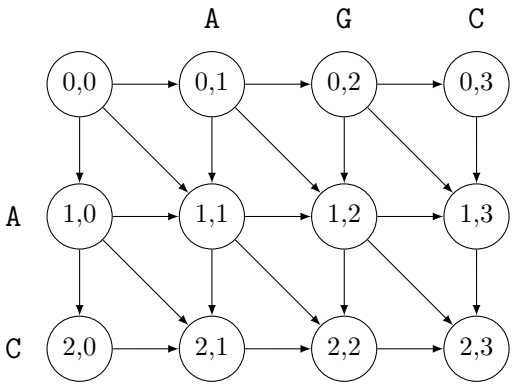

Figure 2. Edit Graph for `AC` and `AGC`.

Let's try looking at a path in this graph. Say we have the following sequence of vertices:

$$v(0,0) \to v(1,1) \to v(1,2) \to v(2,3)$$

Note that it can be interpreted as the following alignment:

```
A-C
AGC
```

Similarly, suppose we're given an alignment:

```
AC--
A-GC
```

We can construct a directed path in the graph:

$$v(0,0) \to v(1,1) \to v(2,1) \to v(2,2) \to v(2,3)$$

We come to the crucial insight which will allow us to develop an algorithm for global alignment:

Given two sequences X and Y, there is a 1-to-1 correspondence between directed paths from $v(0,0)$ to $v(m,n)$ in their edit graph and their alignments.

We will explore this idea further in the next section.