

CS 124, 2002 Lecture 5, How to efficiently compute string similarity

Before we directly consider that issue, we talk about a visual representation of alignment. It is often useful to represent alignments of two strings in terms of a *weighted alignment graph*:

Definition Given two strings S_1 and S_2 of lengths n and m respectively, a weighted alignment graph has $(n + 1) \times (m + 1)$ nodes, each labeled with a distinct pair (i, j) ($0 \leq i \leq n, 0 \leq j \leq m$). The edge weights depend on the specific string problem.

The alignment graph contains a directed edge from each node (i, j) to each of the three nodes $(i, j + 1)$, $(i + 1, j)$ and $(i + 1, j + 1)$, provided those nodes exist. The weight on the first two of these edges is negative one; the weight on the third (diagonal) edge is $t(i + 1, j + 1)$, i.e. 1 if character $i + 1$ of string S_1 is the same as character $j + 1$ of string S_2 . Figure 1 shows the alignment graph for strings CAN and ANN .

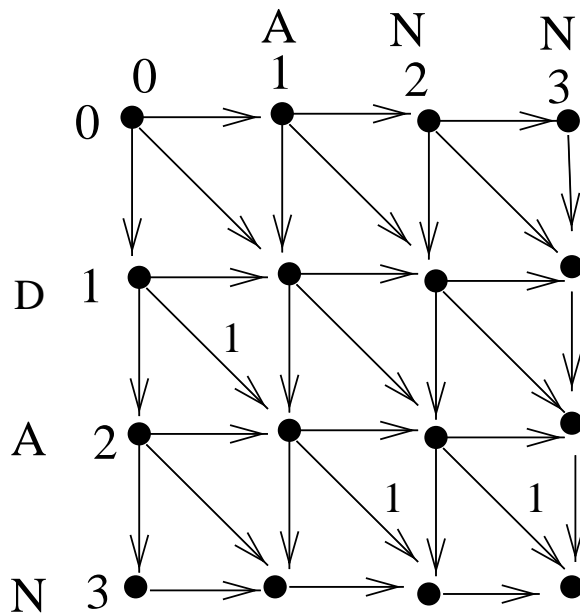


Figure 1: Alignment graph for the strings CAN and ANN . The weight on each edge is minus one, except for the three edges marked in the figure, which each have weight one.

The central property of an alignment graph is that any path from the $(0, 0)$ node to the (n, m) node defines a unique alignment of S_1 and S_2 , and conversely any alignment of S_1 and S_2 defines a unique path from $(0, 0)$ to (n, m) in the alignment graph. To see the correspondence, use the following rules:

1. A diagonal edge going into node (i, j) means we should align character i of S_1 opposite character j of S_2 .
2. A horizontal edge going into column j means we should put character j of S_2 opposite a space.
3. A vertical edge going into row i means we should put character i of S_1 opposite a space.

So as we traverse a path from $(0, 0)$ to (n, m) , we build up the alignment left to right using these three rules. Conversely, if we have an alignment of S_1 and S_2 , we can scan it from left to right and making reference to the above rules, build up the associated path.

Hence, the paths from $(0, 0)$ to (n, m) in the alignment graph are in one-one correspondence with the alignments of S_1 and S_2 .

Question: Assume $n = m$. How many paths from $(0, 0)$ to (n, n) are there? Haven't we already answered that question?

Now we add in a weight or a distance to each edge. Put a weight of minus one on every horizontal or vertical edge. Put a weight of one on the diagonal edge into node (i, j) if character i of S_1 and character j of S_2 are identical (i.e., match), otherwise put a weight of minus one on that edge.

Claim: The path whose total weight (or distance) is maximum from start node $(0, 0)$ to destination node (n, m) specifies an alignment with the maximum value, i.e. one that maximizes the alignment objective function of $(\# \text{matches} - \# \text{mismatches} - \# \text{spaces})$ and hence that value is the similarity value for S_1 and S_2 . In summary, we compute the similarity of the two strings by finding the path with the maximum value: the distance of the path defines the similarity value of the strings, and the path itself defines the associated alignment.

Theorem 0.1 *An alignment of S_1, S_2 has the largest value over all possible alignments if and only if it corresponds to a longest (total distance) path from $(0, 0)$ to (n, m) in the alignment graph.*

Corollary 0.1 *The set of all longest paths from $(0, 0)$ to (n, m) in the alignment graph exactly specifies the set of all optimal alignments of S_1 to S_2 .*

Now we consider how we will efficiently find the longest path.

References