Dynamic Programming: Multi-Parameter Problems

Sequence Alignment

Text comparison is a common computational task that takes many forms. From autocorrect to search engine term completion to DNA analysis, one version of the problem looks to “align” two sequences of characters. Informally, this means matching as many of the characters in one sequence to those in another, allowing for gaps and character changes. For example,

```
better  roust  happy-  obit-  o-currance  o-curr-ance
butter  r-ust  happen  -bite  occurrence  occurre-nce
```

How can we formally define an alignment of two strings?

An alignment of $X = x_1 \ldots x_m$ with $Y = y_1 \ldots y_n$ is a matching $M$ of some of the characters of $X$ with some of the characters of $Y$ such that there are no crossing pairs: for $x_i - y_{i'}$ and $x_j - y_{j'}$, if $i < j$ then $i' < j'$.

In order to measure the level of similarity captured by the matching, we introduce some quantities

- We charge a gap penalty $\delta$ for each character in one string not matched with a character in the other.
- We charge a mismatch penalty $\alpha_{pq}$ for matching $p$ with $q$; usually $\alpha_{pp} = 0$.
- The cost of the matching is then defined to be the sum of all of the gap and mismatch penalties.

We’ll assume that we have been given $\delta$ and the $\alpha_{pq}$ values. The inclusion of these tunable values allow for a very flexible similarity measuring tool.

As always, we begin with an obvious observation—and we’ll assume that $\alpha_{pp} = 0$ for all $p$ in our alphabet.

**Observation 1.** In an optimal (min-cost) alignment $M$ at least one of the following is true

- $(m, n) \in M$,
- $x_m$ is not matched,
- $y_n$ is not matched.

So, let $Opt(i, j)$ denote the minimum cost of an alignment between $x_1 \ldots x_i$ and $y_1 \ldots y_j$.

Then, considering the three cases above, we can say

- If $(i, j) \in M$, then $Opt(i, j) = \alpha_{x_i y_j} + Opt(i - 1, j - 1)$.
- If $x_i$ is not matched, then $Opt(i, j) = \delta + Opt(i - 1, j)$
- If $y_j$ is not matched, then $Opt(i, j) = \delta + Opt(i, j - 1)$

Putting these all together yields

**Theorem 1.** The optimum alignment between $x_1 \ldots x_m$ and $y_1 \ldots y_n$ satisfies the recurrence

$$Opt(i, j) = \min\{\alpha_{x_i y_j} + Opt(i - 1, j - 1), \delta + Opt(i - 1, j), \delta + Opt(i, j - 1)\}$$

Once we have built the table $Opt[1 \ldots m, 1 \ldots n]$ of values, we can trace back through it to construct the actual matching in an additional $O(m + n)$ time.
Analysis

We have already verified the correctness of the recurrence, so we only need to determine the time and space requirements. But this is simple: We need only build an array $Opt[1 \ldots m, 1 \ldots n]$ and this can be done in $O(mn)$ time and space, which is polynomial in the size of the input.

In fact, though, with some reflection, we can improve on the space bounds: See Section 6.7 of K&T for details.