Informatics 2 – Introduction to Algorithms and Data Structures

Solutions for tutorial 7

1. *Execute our dynamic programming algorithm for Edit Distance from lecture 17 on the following two DNA sequences:*

$$ACTGGT$$
$$ATGGCT$$

   (a) Note that in filling the entries of the "a" matrix, I sometimes enter more than one flag (eg 1/2) when the optimum can be obtained with more than one option for the final column of the alignment. It is true the algorithm/pseudocode would not add more than one flag (priority is 1 (substitution), then 2 (insertion)) if there is a tie - however; I am indicating the "spirit" of the a-matrix rather than the literal definition.

   The d matrix will have dimensions 7×7, and as specified in the slides we can fill the left-column and top-row straight away:

$$d = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & & & & & & \\ 2 & & & & & & \\ 3 & & & & & & \\ 4 & & & & & & \\ 5 & & & & & & \\ 6 & & & & & & \end{bmatrix}, a = \begin{bmatrix} - & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \end{bmatrix}$$

   Next we consider row 1, where we compare "A" against all non-empty prefixes of "ATGGCT". We can do this quickly without using the recurrence (cost is always length of the "ATGGCT" prefix, less 1, as the first character is 'A' is always available to match).

   Row 2, with the first prefix fixed as "AC" is not *quite* as simple as row 1, but it's not hard to notice cost of 1 for "AC" against "A" or "AT", then the value increasing with length of prefix 2 except once we have 5 chars in prefix 2 we can also match the 'C' of "AC": So we have:

$$d = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 1 & 2 & 3 & 3 & 4 \\ 3 & & & & & & \\ 4 & & & & & & \\ 5 & & & & & & \\ 6 & & & & & & \end{bmatrix}, a = \begin{bmatrix} - & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 0 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 1 & 1/2 & 1/2 & 0 & 2 \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \end{bmatrix}$$

Next consider row 3, corresponding to prefix “ACT” against the various prefixes of the second string. Against “A”, the final characters do not match, so using the recurrence from the slides, the cost evaluates to 1 greater than $\min\{d[2,1], d[3,0], d[2,0]\}$, this min being $d[2,1] = 1$. So $d[3,1] \leftarrow 2$. Against “AT”, the final characters match.so $d[3,2] \leftarrow d[2,1]$, which is 1. Against “ATG”, “ATGG”, “ATGGC”, the final characters clash (‘T’ does not match the final char of any of these), so we end up with adding 1 to the “min” of three terms each time. This 1+“min” works out at 2, 3 and 4 respectively. Next we must compare “ACT” against “ATGGCT” and the final characters now *do* match so we just set $d[3,6] \leftarrow d[2,5] = 3$.

$$d = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 1 & 2 & 3 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 & 4 & 3 \\ 4 & & & & & & \\ 5 & & & & & & \\ 6 & & & & & & \end{bmatrix}, a = \begin{bmatrix} - & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 0 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 1 & 1/2 & 1/2 & 0 & 2 \\ 3 & 3 & 0 & 1/2 & 1/2 & 1/2/3 & 0 \\ 3 & & & & & & \\ 3 & & & & & & \\ 3 & & & & & & \end{bmatrix}$$

Am just doing rows 4 (“ACTG”), 5 (“ACTGG”), 6 (“ACTGGT”) manually because I can’t type all reasons/consideration-of-recurrences.

$$d = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ 1 & 0 & 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 1 & 2 & 3 & 3 & 4 \\ 3 & 2 & 1 & 2 & 3 & 4 & 3 \\ 4 & 3 & 2 & 1 & 2 & 3 & 4 \\ 5 & 4 & 3 & 2 & 1 & 2 & 3 \\ 6 & 5 & 4 & 3 & 2 & 2 & 2 \end{bmatrix}, a = \begin{bmatrix} - & 2 & 2 & 2 & 2 & 2 & 2 \\ 3 & 0 & 2 & 2 & 2 & 2 & 2 \\ 3 & 3 & 1 & 1/2 & 1/2 & 0 & 2 \\ 3 & 3 & 0 & 1/2 & 1/2 & 1/2/3 & 0 \\ 3 & 3 & 3 & 0 & 0 & 2 & 2/3 \\ 3 & 3 & 3 & 0 & 0 & 2 & 2 \\ 3 & 3 & 0 & 3 & 3 & 1 & 0 \end{bmatrix}$$

(b) *Use the details in table a to reconstruct an optimum alignment (which meets the edit distance) for our sequences.*

answer: We look at cell $[6,6]$ in the matrices, which scores 2 in d. The flag $a[6,6]$ is 0, indicating a *match* of final positions. So we have an alignment which ends with matched ‘T’ characters, hence we next look back at cell $[5,5]$. The value $d[2,2]$ is 2 (of course), the flag $a[5,5]$ is 2, indicating a *insertion.* So the final two columns of our alignment look like this:

-	‘T’
’C’	‘T’

Our next recursive call is to consider $s[1\ldots5]$ against $t[1\ldots4]$, so we examine cell $[5,4]$ ($d[5,4] = 1$, as we expect) - $a[5,4]$ is signal 4, indicating a match. Then we need to look next at cell $[4,3]$, again another match. The final four columns of the alignment are:

‘G’	‘G’	-	‘T’
‘G’	‘G’	’C’	‘T’

We are now next examining cell $[3,2]$, we have $d[3,2] = 1$ as we should (just checking consistency) and $a[3,2] = 0$, indicating another match.

‘T’	‘G’	‘G’	-	‘T’
‘T’	‘G’	‘G’	’C’	‘T’

We do the rest of the traceback from cell $[2, 1]$ (indicating a ‘deletion”, this time) and the final alignment is

’A’	’C’	‘T’	‘G’	‘G’	-	‘T’
‘A’	-	‘T’	‘G’	‘G’	’C’	‘T’

2. The Viterbi matrix is as follows:

	the	old	man	the	lifeboats
DT	.4x.5 = .2	0	0	.00096x.4x.5 = .000192	0
N	0	.2x.6x.2 = .024	.032x.5x.3 = .0048	0	etc.
V	0	0	.024x.4x.1 = .00096	0	0
Adj	0	.2x.4x.4 = .032	0	0	0

Thus the most probable tagging is:

The/DT old/N man/V the/DT lifeboats/N

The `prev`/backtrace details can be read off from the above matrix in an ad-hoc fashion: e.g. in the cell for (man,N), the first factor is .032 which comes from the cell for (the,Adj).

Here is the `prev` array for students to verify their own.

	DT	N	V	Adj
the	-	-	-	-
old	-	DT	-	DT
man	-	Adj	N	-
the	V	-	-	-
lifeboats	-	DT	-	-

3. *This question is concerned with how we might decide whether or not DP is an appropriate technique for a given problem.*

The Travelling Salesman Path problem asks us to construct an ordering π of the vertices of V such that $\pi_1 = s, \pi_n = t$ and $(\pi_i, \pi_{i+1}) \in E$ for all $i = 1, \ldots, n-1$, such that the cost of visiting the vertices in this order is least possible for such orderings. So we are looking for a path *that visits* all *vertices. The cost for such an ordering is $\sum_{i=1}^{n-1} w(\pi_i, \pi_{i+1})$ … more informally, the sum of the weights of the edges in the path.*

Let $TSP((V, E), s, t)$ represent the cost of the optimum s, t TSP. Then we have the following recurrence:

$$TSP((V,E),s,t) = \begin{cases} w(s,t) & \text{if } V = \{s,t\} \\ \min_{\substack{u \in V \setminus \{s,t\}, \\ (u,t)\in E}} \{w(u,t) + TSP((V \setminus \{t\}, E_{\setminus\{t\}}), s, u)\} & \text{if } |V| \geq 3 \end{cases}$$

So we have a recurrence which solves the original problem in terms of $n-2$ slightly smaller subproblems (graphs with one fewer vertex).

Given the collection of smaller subproblems, and the recurrence above, it would seem that TSP is a possible candidate for a dynamic programming solution. Discuss whether *a dynamic programming algorithm is feasible, referring to the features/conditions (dp1)-(dp4) discussed at the end of Lecture 18.*

answer: In Lecture 18 we mentioned four properties we need to have to achieve a dynamic programming algorithm. The first property, (dp1), requires us to understand/find some way of solving an initial problem instance in terms of some calls to (smaller) instances of the same problem. This is closely linked with property (dp2), which asks that we are able to write down a recurrence which expresses the answer for the initial instance (left-hand side) as a computation on results returned for the smaller subproblems. Both (dp1) and (dp2) are given to us in the statement of this question.

However, we also need to be able to achieve (dp3) and (dp4). (dp3) requires the collection of smaller subproblems to be of "polynomially- bounded" size (something like $O(n), O(m \cdot n), O(n^3), O(n^2 \cdot \log(m)))$ and to be describable in terms of a few parameters. Let's consider this in the light of the TSP problem, and let's imagine our input graph is *complete* (there is an edge between every pair of edges) - we make no assumption about the edge weights though.

In this scenario, considering initial source/target vertices s, t, we generate $(n-2)$ sub-problems (and subgraphs) wrt s, u (for $n-2$ different u). Then at the second level of the recursion tree, each of the specific s, u subproblems will generate $(n-3)$ subproblems (and subgraphs) for $(n-3)$ options for u'. And so on ...

We don't have as high a recurrence of the same little subproblems - we do see the subproblems repeat, but we also see different (exponentially many) different subproblems.

To estimate the *number* of subproblems more accurately, we note that while s remains the same, we will see all $u \in V \setminus \{s\}$ generated as the 4th argument of TSP in the recursion tree. Also, we will see almost every possible $U \subseteq V$ be generated as the consequence of the contraction of a number of intermediate path vertices in argument 1 (with induced edge set in argument 2) of TSP.

Hence the *number of different* TSP calls in the recursion tree will be proportional to $\Theta(n \cdot 2^n)$

This is not polynomially-bounded, and we are unable to exploit dynamic programming to get an efficient algorithm.

Mary Cryan