Informatics 2 – Introduction to Algorithms and Data Structures

Tutorial 7: Dynamic Programming (cont'd)

Mary Cryan

week 4: Mon 3rd - Fri 7th February, 2025

1. Recall the dynamic programming algorithm for Edit Distance that we discussed in Lecture 20. Consider the following two DNA sequences:

   ACTGGT
   ATGGCT

   (a) Execute the Edit-Distance algorithm on this pair of sequences, filling in the entries of $d[i,j], a[i,j]$ as we progress (working through $i \leftarrow 1 \dots m$ and $j \leftarrow 1 \dots n$) through the rows.

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

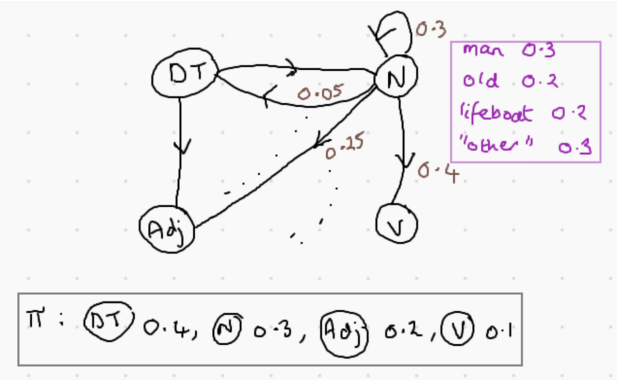
2. We have mentioned that HMMs are often used to model sentences in natural language. For this application, the "most likely path" will correspond to a most probable part-of-speech tagging for the given sentence.

   In this question we will consider a simple HMM to model simple sentences in English, with respect to a very small vocabulary. We are interested in the following sentence:

   The old man the lifeboats.

   Our dictionary of words (and their probabilities) only directly mentions the words *man*, *old*, *lifeboats* and *the* ... with all other words being grouped under "other".

   Non-native speakers should be aware (for context) that *old* is usually an adjective (Adj), but *the old* can be used to refer to a group of older people (a noun phrase). Also, the word *man* usually just means one male individual (N), but it is also a verb (V) sometimes, meaning to work. However, the words *the* and *lifeboats* can only be tagged as DT and N respectively.



We consider a HMM with 4 states, DT, N, V and Adj, with the suggested interpretations with respect to the English language. The transitions of the HMM, and their probabilities, are given in the matrix below left (row N gives the probabilities of the transitions leaving N, for example). The picture above only shows the details of the distribution for state N, and its outgoing transitions, but details for other states are given in the matrices.

	DT	N	V	Adj
DT	0	0.6	0	0.4
N	0.05	0.3	0.4	0.25
V	0.4	0.3	0.1	0.2
Adj	0.1	0.5	0.2	0.2

Transitions

	lifeboats	man	old	the	"other"
DT	0	0	0	0.5	0.5
N	0.2	0.3	0.2	0	0.3
V	0	0.1	0	0	0.9
Adj	0	0	0.4	0	0.6

Emissions

The "start state" distribution assigns probabilities as follows:
$\pi(\text{DT}) = 0.4, \pi(\text{N}) = 0.3, \pi(\text{Adj}) = 0.2, \pi(\text{V}) = 0.1$.

(note the notation π here is being used for a different concept than it was for the shortest paths algorithms)

Now use the *Viterbi algorithm* to compute the/a "most likely path", and thereby tag the sentence

The old man the lifeboats

using the following transition and emission probabilities.

Note that in executing the Viterbi algorithm, you should make sure to build the `prev` array with the backtracking pointers, as shown in the pseudocode. The middle section of our *pre-recorded lecture* discusses the building of this `prev` values.

You may want to use a calculator to help with the arithmetic.

Note that in the transition matrix, rows represent the 'previous state', and columns represent the 'next state'. The diagram on the previous page shows part of this HMM, focusing on the output transitions and "emissions" distribution for state N.

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* is a classic problem studied in Computational Complexity[1]. The input for this problem is an undirected graph $G = (V, E)$, together with a distinguished start node s, a distinguished end node t ($t \neq s$), and a weight function $w : E \to \mathbb{Q}^+$ (we are guaranteed that all edge-weights are strictly positive).

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.

If we were to think about trying a recursive/dynamic-programming approach for this problem, we could notice the following:

- Every candidate solution to this problem *must end with some edge of the graph* $G = (V, E)$, ie in some edge (u, t) for some $u \in V \setminus \{s, t\}$.
- The best TSP solution that ends in the specific edge $e = (u, t)$ is equivalent to the value $w(u, t)$ added to the best s, u TSP solution on $G' = (V \setminus \{t\}, E_{\setminus\{t\}})$, where $E_{\setminus\{t\}}$ is the set of original edges E with all edges adjacent to t deleted.

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

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

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.

[1] What we describe here is a slight variant of the classic TSP problem, which asks for a minimum cost *cycle* (wrapping round to the start) rather than a path from s to t.