Notes for Lecture 15

We discuss dynamic programming approaches to two problems: time-bounded shortest paths and TSP.

1 Shortest Paths with Time Bounds

In the time-bounded shortest path problem, we are given a graph $G = (V, E)$, we are given a length $\ell(u, v)$ for each edge $(u, v) \in E$, we are given a start vertex $s \in V$, and we are also given a traveling time $t(u, v)$ for each edge $(u, v) \in E$ and a time bound T. The parameters $t(u, v)$ and T are positive integers, while $\ell(u, v)$ are arbitrary numbers.

The goal is to find, for every $v \in V$, the length of the shortest path from s to v that takes total time at most T. (The total time taken by a path is the sum of the traveling times of the edges of the path.)

The first step in devising a dynamic programming algorithm is to find a way to reduce the problem on the given data to a problem on data that is "smaller" in some sense. When the problem asks to find a path, it is often useful to reason about one of the edges of the path, for example the last edge. An approach that often works it to reason that the optimal path, once we remove the last edge, is still the optimal path for a different initial problem, so that, reasoning backwards, we can find our optimal path by first finding the optimum of this different problem, and then adding an edge to it.

In our case, we can reason that if $s, v_1, \ldots, v_k, v$ is the shortest path from s to v that takes time at most T, then $s, v_1, \ldots, v_k$ must be the shortest path from s to v_k that takes time at most $T - t(v_k, v)$.

Once we have this observation, we want to enumerate all the new problems that we can generate by iterating this reasoning, and we see that those are (contained in) the set of all problems where, given our data, a vertex v, and a time $0 \leq t \leq T$, we want to find the shortest path from s to v that takes time at most t. If we denote this quantity by $dist[v, t]$, we have the formulas

$$dist[s, 0] = 0$$

$$dist[v, 0] = \infty \quad \forall v \neq s$$

$$dist[v, t] = \min \left\{ dist[v, t-1], \min_{u:(u,v)\in E \text{ and } t \geq t(u,v)} dist[u, t - t(u, v)] + \ell(u, v) \right\}$$

which we can immediate turn into pseudocode.

initialize $|V| \times T$ array $dist[\cdot, \cdot]$
$dist[s, 0] = 0$
for all $v \in V - \{s\}$: $dist[v, 0] = \infty$
for $t = 1$ to T:
    for all v in V:
        $dist[v, t] = dist[v, t-1]$
        for all edges (u, v) into v such that $t(u, v) \leq t$:
            if $dist[v, t] > dist[v, t - t(u, v)] + \ell(u, v)$:
                $dist[v, t] = dist[v, t - t(u, v)] + \ell(u, v)$
return $dist[\cdot, T]$

To analyze the running time, we see that the outer loop is executed T times, the second loop V times, the third loop a many times as the degree of v, and the update done for a particular t and edge (u, v) takes constant time, so that the overall time is

$$O\left(T \cdot \sum_{v \in V} degree(v)\right) = O(|T| \cdot |E|)$$

2 TSP

In the *Traveling Salesman Problem* we are given a complete undirected graph on n vertices (which we will also call "cities"), and, for every pair of cities u, v, we are given a positive distance $\ell(u, v)$. (Note that we will assume that $\ell(u, v) = \ell(v, u)$.) We want to find a cycle $v_1, v_2, \ldots, v_n, v_1$ that goes through each city once and that minimizes the total distance

$$\ell(v_1, v_2) + \ell(v_2, v_3) + \cdots \ell(v_{n-1}, v_n) + \ell(v_n, v_1)$$

There are two interesting variants of the problem: we may ask for the cycle that goes through each city *exactly once*, or for the cycle that goes through each city *at least once*. The algorithm that we describe below is for the version in which we want to go through each city *exactly* once. We discuss later how to adjust it to the case in which we want to go through each city at least once.

A naive algorithm for this problem is to try all the possible orderings of the n cities, and then see which one is best, yielding a running of the order of $n!$. We will describe a considerably faster, but still exponential-time, dynamic programming algorithm that runs in time $O(n^2 2^n)$. In the following we will identify the set of cities with the set $\{1, \ldots, n\}$.

To start thinking about a dynamic programming algorithm, we should think of a way in which, for example, taking an edge out of our optimal cycle creates an optimal solution to a smaller problem. Unfortunately, taking an edge out of a cycle defines a path, rather than a shorter cycle, so it does not seem an idea that leads to reduce our problem to a similar problem on smaller data.

The next idea is to first reduce the problem of finding an optimal cycle to a problem involving finding a path, and then solve the latter problem using dynamic programming. Indeed, if we have an optimal cycle, and we take out the edge that goes from vertex 1 to the next vertex in the cycle, call it v, then what we are left with is the shortest path that starts at 1, ends at v, and visits all vertices exactly once.

So let us call $TSPP[v]$, for TSP Path, the length of the shortest path from 1 to v that goes through each vertex once. The solution to the TSP problem is

$$\min_{v \in \{2,\ldots,n\}} \quad TSPP[v] + \ell(1, v) \tag{1}$$

Now the TSPP problem is amenable to a dynamic programming approach, because if take the optimal solution to $TSPP[v]$, and we remove the last edge, call it the edge (u, v), then what we are left is the shortest path among paths that start at 1, end at u, and go through each vertex of $V - \{v\}$ exactly once. This is a "smaller" subproblem, because we have reduced to a TSPP problem involving $n - 1$ cities instead of n.

The subproblems that we generate by the reasoning above are problems in which we want to find the shortest path among paths that go from 1 to v and go through every city in a set S exactly once, where S contains both 1 and v. Let us call this problem $TSPP[S, v]$. Then the base case is when S has cardinality 2, that is

$$TSPP[\{1, v\}, v] = \ell(1, v)$$

and then, for sets S having at least three elements, and for each $v \in S - \{1\}$, we have the equation

$$TSPP[S, v] = \min_{u \in S - \{1,v\}} \quad TSPP[S - \{v\}, u] + \ell(u, v)$$

It takes time $O(n)$ to fill each entry of the array $TSPP[\cdot, \cdot]$, there are $O(n \cdot 2^n)$ entries, so in time $O(n^2 2^n)$ we can compute all the entries, and then, using Equation (1), in $O(n)$ time we find the optimum of the TSP problem.