CS5800: Algorithms — Virgil Pavlu

Bitonic Travelling Salesman Problem

Name: Jonathan Chen

1. Exercise 15.3: ***Bitonic euclidean traveling-salesman problem.***

In the ***euclidean traveling-salesman problem***, we are given a set of n points in the plane, and we wish to find the shortest closed tour that connects all n points. Figure 15.11(a) shows the solution to a 7-point problem. The general problem is NP-hard, and its solution is therefore believed to require more than polynomial time (see Chapter 34).

J. L. Bentley has suggested that we simplify the problem by restricting our attention to ***bitonic tours***, that is, tours that start at the leftmost point, go strictly rightward to the rightmost point, and then go strictly leftward back to the starting point. Figure 15.11(b) shows the shortest bitonic tour of the same 7 points. In this case, a polynomial-time algorithm is possible.

Describe an $O(n^2)$-time algorithm for determining an optimal bitonic tour. You may assume that no two points have the same x-coordinate and that all operations on real numbers take unit time. (*Hint:* Scan left to right, maintaining optimal possibilities for the two parts of the tour.)

Solution:

Let P be the array of all n points, sorted in ascending order by x-coordinate.

Let C be the two-dimensional array of all sub-problems. $C[s,e] = (s, \ldots, n, \ldots, e)$ represents the optimal solution of indices that form a bitonic path $P[s] \rightsquigarrow P[n] \rightsquigarrow P[e]$ through all points $\{P[s] \cup P[e..n]\}$. Notice that $C[s,e]$ consists of the same points as $C[e,s]$ but in reverse order.

Let L be the two-dimensional array of all sub-problem lengths. $L[s,e]$ is the length of the optimal solution to the path $C[s,e]$.

Optimal Substructure: Pick any point $P[q]$ in the path $P[s] \rightsquigarrow P[n] \rightsquigarrow P[e]$. Then it must be that $P[n]$ is in $P[s] \rightsquigarrow P[q]$ or $P[q] \rightsquigarrow P[e]$. Whichever one of the paths $P[n]$ is in, that path must be an optimal sub-problem. If it were not, we would use the exchange argument to establish a better solution for $C[s,e]$.

Recursive Definition: We can use the optimal substructure to define $C[s,e]$ recursively:

$$L[s,e] = \max \begin{cases} \text{dist}(P[s], P[\max(s,e)+1]) + L[e, \max(s,e)+1] \\ \text{dist}(P[e], P[\max(s,e)+1]) + L[s, \max(s,e)+1] \end{cases}$$

$$C[s,e] = \max_{\text{chosen by } L} \begin{cases} (s) + C[\max(s,e)+1, e] \\ C[s, \max(s,e)+1] + (e) \end{cases}$$

Graphically, this represents the decision to connect point $P[\max(s,e)+1]$ (the next leftmost point) to point $P[s]$ or $P[e]$ respectively. Now with smaller limits, we can solve a sub-problem recursively.

Building the table bottom-up: We need a 2-D table like this:

$C[n,n]$	$C[n,n-1]$	$\cdots$	$C[n,1]$
$C[n-1,n]$	$C[n-1,n-1]$	$\cdots$	$C[n-1,1]$
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$C[1,n]$	$C[1,n-1]$	$\cdots$	$C[1,1]$

The base cases are $C[n,n] = (n)$, $C[s,n] = (s, s+1, \ldots, n)$, and $C[n,e] = (n, n-1, \ldots, e)$. We can build the table left to right and top to bottom, which ensures that all sub-problems $C[a,b]$ where $a \geq s$ and $b \geq e$ have already been solved.

Expressing the solution: The solution of the problem with points P is $C[1,1]$. In the first step, L will decide at random whether to connect the second point to the start or the end. In following steps, we move s and e closer and closer to n. (Notice that only at the start can $s = e$, so really the rest of the middle column is not needed.)

Proof of runtime: Assuming constant-time lookup in the table, each cell takes $\Theta(1)$ time to identify the optimal solution for the given s and e. There are n^2 cells in total, all of which must be filled. So we have that $T(n) = \Theta(n^2)$, as desired.