Jade Yu Cheng
ICS 311
Homework 8
Sep 29, 2008

Question for lecture 11

Problem 15-1 on p. 364

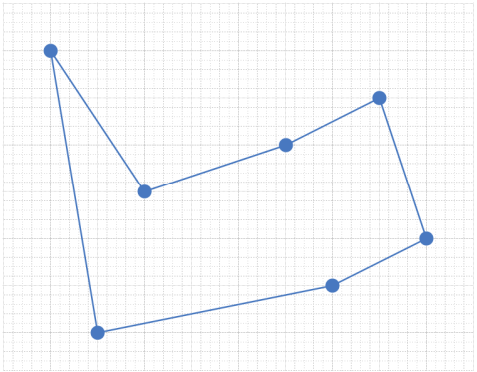
Bitonic Euclidean traveling-salesman problem

The Euclidean traveling-salesman problem is the problem of determining the shortest closed tour that connects a given set of n points in the plane. Figure b shows the solutions to a 7-point problem. The general problem is NP-complete, and its solution is therefore believed to require more than polynomial time.

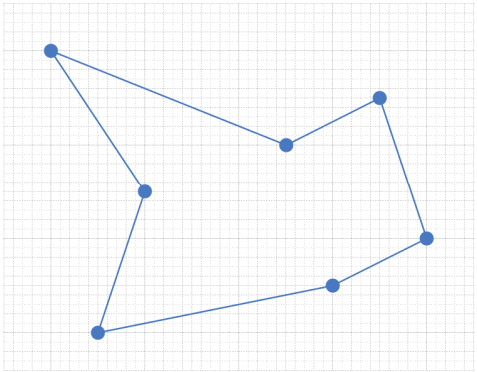
J. L. Bentley has suggested that we simplify the problem by restricting out attention to bitonic tours, that is, tours that starts at the leftmost point, go strictly left to right to the rightmost pint, and then go strictly right to left back to the starting pint. Figure a shows the shortest bitonic tour of the same 7 pints. 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.

a



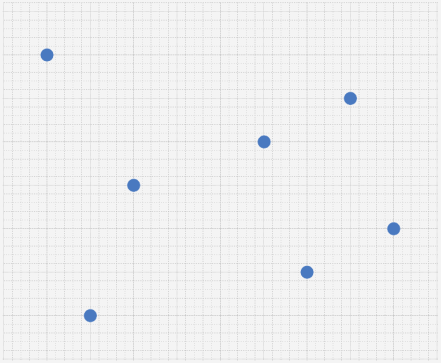
b



Answer: Consider the problem as looking for the shortest bitonic tour $L(a, b)$ among a set of pints $\{1,2,3, \dots, a, \dots b, \dots n\}$. When we assign $a = n$ and $b = n$, $L(a, b)$ represents the shorted bitonic tour from the leftmost point to the rightmost point and back to the leftmost point. During the trip, all points in between of the leftmost point and the rightmost points including the two end pints are all covered.

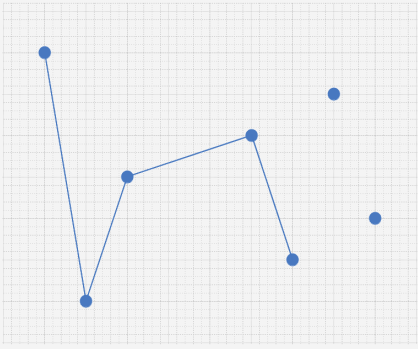
In computational geometry, a bitonic tour of a set of point sites in the Euclidean plane is a closed polygonal chain that has each site as one of its vertices, such that any vertical line crosses the chain at most twice. (From Wikipedia)

Fist I need pick a coordination system, and give each point their x, y coordinates. Then, I can sort the points based on the x coordinates. This is the same procedure as I place the points on a Euclidean plane. I would get a map as below:



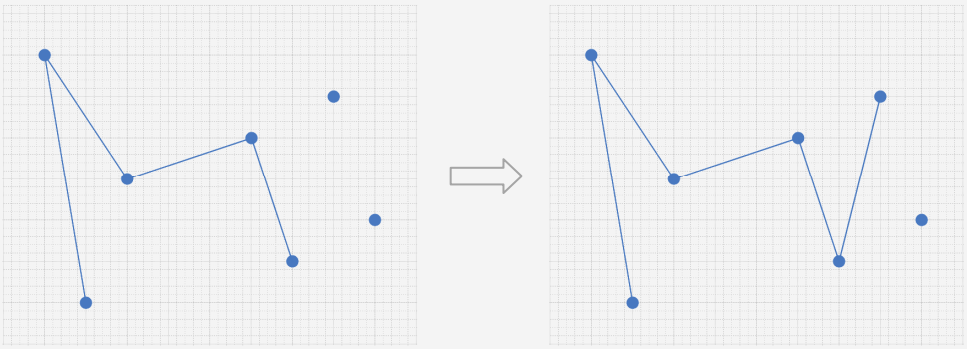
Consider any given a and b. There are three cases that are fairly independent form each other.

1. Case 1 is when $a = 0$. The bitonic tour $L(a, b)$ need to be a single line from the leftmost point to b. For example the solution of $L(0,4)$ looks like:



There's only one path satisfying the definition of a bitonic tour. That is the single line connecting all of the points that are smaller than or equals to b. Therefore in order to solve $L(0,b)$, I can solve $L(0,b-1)$ and let $L(0,b) = L(0,b-1) + edge(b-1,b)$. For each step, if I start from $L((0,b-1)$, which we assume is already known. I need to compute only one step to get the solution of $L((0,b)$.

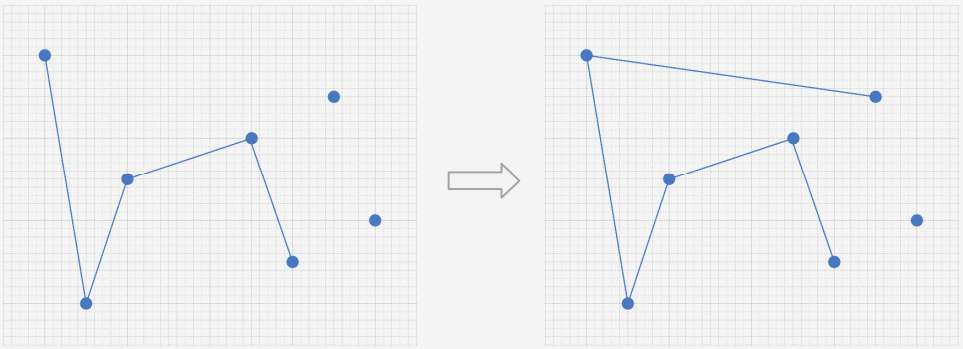
2. Case 2 is when $a < b-1$, the bitonic tour $L((a,b)$ has still only one choice in order to satisfy the definition of a bitonic tour. $L(a,b) = L(a,b-1) + edge(b-1,b)$. For example as show below, $L(1,5) = L(1,4) + edge(4,5)$.



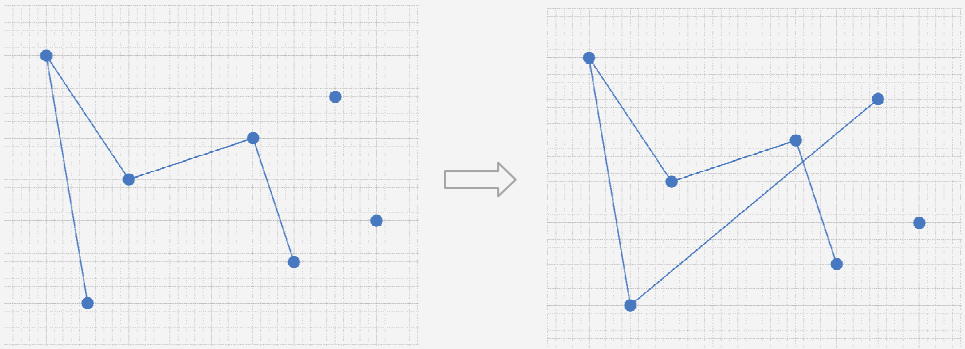
In order to solve $L(a,b)$, while $a < b-1$, I can solve $L(a,b-1)$, and let $L(a,b) = L(a,b-1) + edge(b-1,b)$. For each step, if I start from $L(a,b-1)$, which we assume is an already known value, I need to compute only one step to get the solution of $L(a,b)$.

3. Case 3 is when $a = b-1$, in this case, the bitonic tour $L(a,b)$ has more than one option to choose from, if the only constrain is to satisfy the definition of a bitonic tour. For example as shown below, $L(4,5)$ needs to be the minimal of a collection of possible solutions. They are $\{L(0,4) + edge(0,5)\}$, $\{L(1,4) + edge(1,5)\}$, $\{L(2,4) + edge(2,5)\}$, $\{L(3,4) + edge(3,5)\}$.

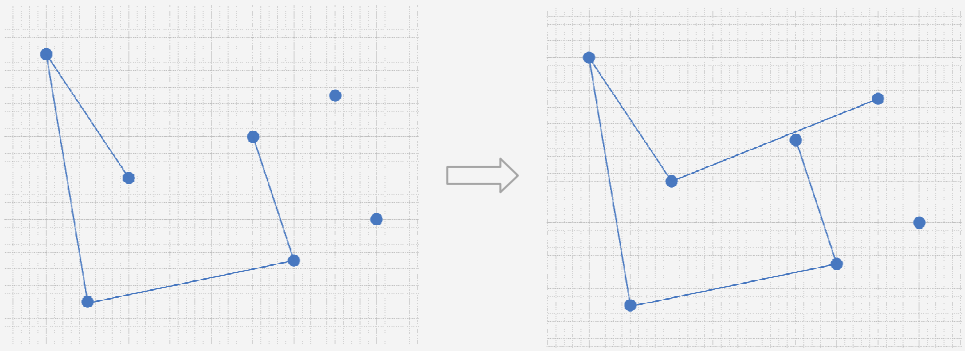
a. Plan1: $L(0,4) + edge(0,5)$:



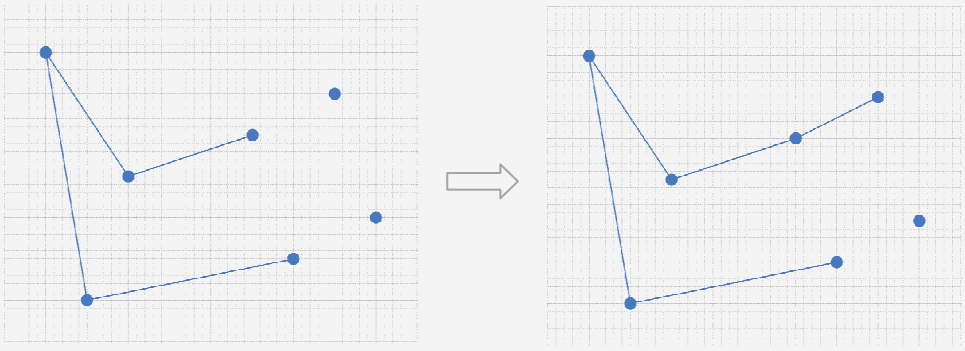
b. Plan2: $L(1,4) + edge(1,5)$:



c. Plan3 $L(2,4) + edge(2,5)$:



d. Plan4 $L(3,4) + edge(3,5)$:



Of course, for each step $L(0,4), L(1,4), L(2,4), L(3,4)$ are assumed to be the best solution for their picked a and b. For example, the $L(3,4)$ I used to build $L(4,5)$ is assumed to be the best solution, the shorted bitonic tour of point 3 and point 4.

Apparently, the overall best solution contains the best solutions of sub-problems. In order to compare all the possible combinations of $L(4,5)$, I need to know the best solution of $L(3,4)$. The first Hallmark of Dynamic-programming is the optimal substructure. An optimal solution to a problem (instance) contains optimal solutions to sub-problems.

Also, to compute all the combinations for different sets of a and b values, I need to repeat some of the sub-problem works. For example, in order to find the best solution of $L(3,5)$, I need to know $L(3,4)$. In order to find the best solution of $L(4,5)$, I need to know $L(0,4), L(1,4), L(2,4)$, and also $L(3,4)$. I don't need to compute $L(3,4)$ more than once, instead I would store the result of $L(3,4)$ somewhere. When the solution for $L(3,4)$ is needed later on, I can just use it. This feature matches with the second Hallmark of Dynamic-programming—Overlapping sub-problems. A recursive solution contains a "small" number of distinct sub-problems repeated many times.

Therefore, the bitonic Euclidean traveling-salesman problem is a good candidate of using dynamic programming. Based on this thought, I can implement the Memoization algorithm. I can try keeping a table to store the sub-problems that will be gradually computed and will be used more than once while computing the larger sets of inputs. The table looks like:

$L(0,1)$							
$L(0,2)$	$L(1,2)$						
$L(0,3)$	$L(1,3)$	$L(2,3)$					
$L(0,4)$	$L(1,4)$	$L(2,4)$	$L(3,4)$				
$L(0,5)$	$L(1,5)$	$L(2,5)$	$L(3,5)$	$L(4,5)$			
:	:	:	:	:			
$L(0,n-1)$	$L(0,n-1)$	$L(2,n-1)$	$L(3,n-1)$	$L(4,n-1)$	:	$L(n-2,n-1)$	
$L(0,n)$	$L(0,n)$	$L(2,n)$	$L(3,n)$	$L(4,n)$	:	$L(n-2,n)$	$L(n-1,n)$

The number of operations associated with each cell could be expressed as table:

1							
1	1						
1	1	2					
1	1	1	3				
1	1	1	1	4			
:	:	:	:	:			
1	1	1	1	1	:	$n-2$	
1	1	1	1	1	:	1	$n-1$

Therefore the total number of operations processed to compute the best bitonic tour of an input size of n is: $1 + 2 + 4 + 6 + 8 + \cdots + 2(n-1) = \Theta(n^2)$.