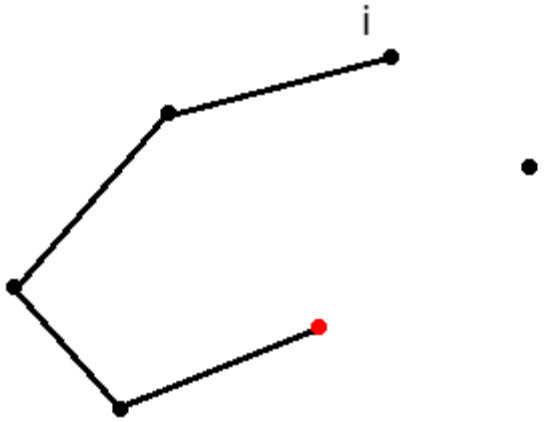
Toni Annala
toni.annala@helsinki.fi

# A solution to Bitonic euclidean traveling-salesman problem

We are given an array of n points $p_1, \ldots, p_n$. We can assume that this array is sorted by the x-coordinate in increasing order, otherwise we could just sort it O(n*log(n)) time and the time complexity of this algorithm wouldn't change. For each index i=1..n-1 we will calculate what is the minimum cost when we assign all the points $p_1, \ldots, p_i$ to exactly one of the two paths (except the leftmost point $p_0$, which is in both of the paths). The cases would look like this:




From the picture we can see that there are two paths that both start from the leftmost point and each of the points on the left side of $p_i$ are in at least one of the two paths. The minimum weight of such a graph will be denoted by c(i) and it is the sum of all used edges. We will also keep track for each index i what is the rightmost point before $p_i$ that is in the different path (this might be $p_1$ even though p1 is always on the same path with $p_i$). We will denote this point by f(i). The point f(i) is colored red in the picture.

The base case is quite easy. c(1)=0 as there can be no edges if we use only one point. $f(1)=p_1$ as $p_1$ belongs to both of the paths.

Now let 1<i<n. There must always be a point that precedes $p_i$ in the same path with it. There are two different cases: either $p_{i-1}$ precedes $p_i$ or the point that precedes $p_i$ is on the left side of $p_{i-1}$. This will make a difference as then there will be at least one point between $p_{i-1}$ and $p_i$.

Lets denote the index of the point that precedes $p_i$ by e. The optimal cost for the situation that $p_e$ precedes $p_i$ is denoted by $c_e(i)$, and the rightmost point that is in the different path from $p_i$ in that case is denoted by $f_e(i)$. They can be calculated as follows:

Case 1, e=i-1:

$$c_{i-1}(i)=c(i-1)+d(p_{i-1},p_i),\ f_{i-1}(i)=f(i-1)$$

Case 2, e<i-1:

$$c_e(i) = c(e) + d(p_e, p_i) + d(f(e), p_{e+1}) + \sum_{a=e+1}^{i-2} d(p_a, p_{a+1})$$

$$f_e(i) = p_{i-1}$$

In case 2 we have to take the points between $p_e$ and $p_i$ into account and so we have two new terms in our formula. As all points between $p_e$ and $p_i$ are on the same path with each other we must add the weight by edges between them. This is the fourth term.

If we have calculated a prefix sum array for weights of edges of form ($p_a$, $p_{a+1}$) we can calculate the 4th term in constant time, so for each pair (i,e) the calculation of $c_e(i)$ and $f_e(i)$ takes only constant time. Let m be the index that minimizes $c_e(i)$. Then $c(i)=c_m(i)$ and $f(i)=f_m(i)$.

After we have c(i) and f(i) we can discard all $c_e(i)$ and $f_e(i)$ as we wont need them anymore.

When we have calculated the c(i) for all i<n (which clearly takes $O(n^2)$ time) we can calculate the minimum weight of a Bitonic Hamiltonian cycle, this is denoted by M. It is quite similar to the previous calculations but there is one extra term in both of the cases. The extra term follows from the fact that both of the paths must connect with the point $p_{n,}$. In every other way calculating M is identical to calculating c(i) for some i<n.

Case 1, e=n-1:

$$M_{n-1} = c(n-1) + d(p_{n-1}, p_n) + d(f(n-1), p_n)$$

Case 2, e<n-1

$$M_e = c(e) + d(p_e, p_n) + d(p_{n-1}, p_n) + d(f(e), p_{e+1}) + \sum_{a=e+1}^{n-2} d(p_a, p_{a+1})$$

M is just the minimum of $M_i$ for i<n. The actual path could be computed quite easily if we just kept track of the point that precedes $p_i$ in the same path with it, for every i.

The time complexity of this algorithm is $O(n^2)$ and the space complexity is O(n).