Illinois Institute of Technology  
Department of Computer Science

**Greedy Matching and Suboptimality**  
CS 430 Design and Analysis of Algorithms  
Spring Semester, 2018

Suppose you are given a set of points on the plane. Consider the problem of connecting each point to exactly one other point in such a way that the cost, the sum of the lengths of the edges, is minimized.

A brute force approach to this problem is to generate each possible matching of points and to find the corresponding costs. The matching with the minimum cost is thus the solution to the problem. While this approach will inarguably produce correct results, it does require exponential time. This encourages us to search for a more efficient algorithm.

One simple algorithm that comes to mind is a greedy algorithm. Find the two nearest points that are not yet connected (to other points) and connect them. Repeat this process exhaustively. In the case of a tie, select a pair of points arbitrarily. How efficient is this algorithm? If there are \( n \) points, it will take time \( \binom{n}{2} + \binom{n-2}{2} + \binom{n-4}{2} + \cdots = \Theta(n^3) \) to iteratively look for two nearest points.

Before declaring success, we must ask if our algorithm is correct: does it, in fact, give optimal results? Unfortunately it does not. Consider four points equally spaced along a line:

\[
\begin{array}{cccc}
  \cdot & \cdot & \cdot & \cdot \\
\end{array}
\]

In the first iteration, there are three optimal pairs of points, so the algorithm chooses a pair arbitrarily. Say it chooses the two points in the middle and is then forced to connect the two remaining points:

\[
\begin{array}{cccc}
  \cdot & \cdot & \cdot & \cdot \\
\end{array}
\]

This is clearly not the optimal solution, however. The optimal solution connects the two leftmost points and the two rightmost points:

\[
\begin{array}{cc}
  \cdot & \cdot \\
  \cdot & \cdot \\
\end{array}
\]

In fact, we can exploit the above problem with the greedy algorithm by putting two instances of the above problem side by side:

\[
\begin{array}{cccccc}
  \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
  \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\end{array}
\]

Again, the optimal solution does much better:

\[\text{This choice of the middle points first can forced by separating the points by distances 1, } 1 - \epsilon, \text{ and 1, where } \epsilon \text{ is some small positive number.}\]
We can repeat this process to get examples where the greedy performance gets worse and worse. Specifically, consider repeating this process until we have \( N = 2^n \) points. The minimum cost of the optimal algorithm is:

\[
OPT_n = 2^{n-1}, \quad \text{where} \quad OPT_1 = 1
\]

Compare this to the solution given by the greedy algorithm. Let \( L_n \) be the length from the leftmost point to the rightmost point if there are \( 2^n \) points. This leads to the recurrence:

\[
GREEDY_n = 2GREEDY_{n-1} + L_n - 2L_{n-1} + L_{n-1}
\]

\[
= 2GREEDY_{n-1} + L_n - L_{n-1}
\]

\[
= 2GREEDY_{n-1} + 3^n - 3^{n-2}
\]

The recurrence is annihilated by \((E - 2)(E - 3)\), leading to the solution:

\[
GREEDY_n = 2 \cdot 3^{n-1} - 2^{n-1}
\]

The error in the greedy algorithm relative to the optimal solution is therefore:

\[
\frac{GREEDY_n}{OPT_n} = \frac{2 \cdot 3^{n-1} - 2^{n-1}}{2^{n-1}}
\]

\[
= 2 \cdot \left( \frac{3}{2} \right)^{n-1} - 1
\]

\[
= 4 \cdot \left( \frac{3}{2} \right)^n - 1
\]

\[
= 4 \cdot \left( \frac{3}{2} \right)^{\log_3 N} - 1
\]

\[
= 4 \cdot N^{\log_3 2} - 1
\]

\[
\approx O(N^{0.5849625...})
\]

Not only is the solution given by the greedy algorithm not optimal, its relative error grows with the size of the problem.