Lecture 4 — Divide and Conquer Continued

Parallel and Sequential Data Structures and Algorithms, 15-210 (Fall 2012)

Lectured by Guy Blelloch — September 6

Material in this lecture:

- Maximum Contiguous Subsequence Sum (MCSS) problem (continued from last lecture) with two different divide-and-conquer solutions.
- Solving recurrences using substitution (in the last lecture notes)
- Euclidean Traveling Salesperson Problem using Divide and conquer.

Divide and Conquer Example II: The Euclidean Traveling Salesperson Problem

We'll now turn to another example of divide and conquer. In this example, we will apply it to devise a heuristic method for an **NP**-hard problem. The problem we're concerned with is a variant of the traveling salesperson problem (TSP) from Lecture 1. This variant is known as the Euclidean traveling salesperson (eTSP) problem because in this problem, the points (aka. cities, nodes, and vertices) lie in a Euclidean space and the distance measure is the Euclidean measure. More specifically, we're interested in the planar version of the eTSP problem, defined as follows:

Definition 0.1 (The Planar Euclidean Traveling Salesperson Problem)**.** Given a set of points P in the 2-d plane, the *planar Euclidean traveling salesperson* (eTSP) problem is to find a tour of minimum total distance that visits all points in P exactly once, where the distance between points is the Euclidean (i.e. ℓ_2) distance.

Not counting bridges, this is the problem we would want to solve to find a minimum length route visiting your favorite places in Pittsburgh. As with the TSP, it is **NP**-hard, but this problem is easier[1] to approximate.

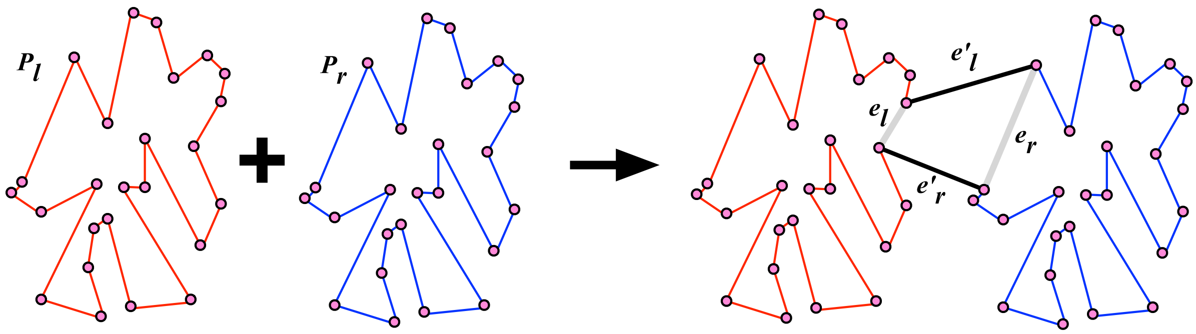
Here is a heuristic divide-and-conquer algorithms that does quite well in practice. In a few weeks, we will see another algorithm based on Minimum Spanning Trees (MST) that gives a constant-approximation guarantee. This divide-and-conquer is more interesting than the ones we have done so far because it does work both before and after the recursive calls. Also, as we will see, the recurrence it generates is root dominated.

The basic idea is to split the points by a cut in the plane, solve the TSP on the two halves, and then somehow merge the solutions. For the cut, we can pick a cut that is orthogonal to the coordinate lines. In particular, we can find in which of the two dimensions the points have a larger spread, and then find the median point along that dimension. We'll split just below that point.

†Lecture notes by Guy E Blelloch, Margaret Reid-Miller, and Kanat Tangwongsan.

[1]Unlike the TSP problem, which only has constant approximations, it is known how to approximate this problem to an arbitrary but fixed constant accuracy ε in polynomial time (the exponent of n has $1/\varepsilon$ dependency). That is, such an algorithm is capable of producing a solution that has length at most $(1+\varepsilon)$ times the length of the best tour.

To merge the solutions we join the two cycles by making an edge swap



To choose which edge swap to make, we consider all pairs of edges of the recursive solutions consisting of one edge $e_\ell = (u_\ell, v_\ell)$ from the left and one edge $e_r = (u_r, v_r)$ from the right and determine which pair minimizes the increase in the following cost:

$$\texttt{swapCost}((u_\ell, v_\ell), (u_r, v_r)) = \left\|u_\ell - v_r\right\| + \left\|u_r - v_\ell\right\| - \left\|u_\ell - v_\ell\right\| - \left\|u_r - v_r\right\|$$

where $\|u - v\|$ is the Euclidean distance between points u and v.

Here is the pseudocode for the algorithm

```
fun eTSP(P) =
  case (|P|)
    of 0,1 ⇒ raise TooSmall
     | 2 ⇒ {(P[0],P[1]),(P[1],P[0])}
     | n ⇒ let
          val (Pℓ,Pr) = splitLongestDim(P)
          val (L,R) = (eTSP(Pℓ) || eTSP(Pr))
          val (c,(e,e')) = minVal_first {(swapCost(e,e'),(e,e')) : e ∈ L, e' ∈ R}
        in
          swapEdges(append(L,R),e,e')
        end
```

The function $\texttt{swapEdges}(E, e, e')$ finds the edges e and e' in E and swaps the endpoints (there are two ways to swap, so the cheaper is picked).

Now let's analyze the cost of this algorithm in terms of work and span. We have

$$\begin{aligned} W(n) &= 2W(n/2) + O(n^2) \\ S(n) &= S(n/2) + O(\log n) \end{aligned}$$

We have already seen the recurrence $S(n) = S(n/2) + O(\log n)$, which solves to $O(\log^2 n)$. Here we'll focus on solving the work recurrence.

In anticipation of recurrences that you'll encounter later in class, we'll attempt to solve a more general form of recurrences. Let $\varepsilon > 0$ be a constant. We'll solve the recurrence

$$W(n) = 2W(n/2) + k \cdot n^{1+\varepsilon}$$

by the substitution method.

Theorem 0.2. *Let $\varepsilon > 0$. If $W(n) \leq 2W(n/2) + k \cdot n^{1+\varepsilon}$ for $n > 1$ and $W(1) \leq k$ for $n \leq 1$, then for some constant κ,*

$$W(n) \leq \kappa \cdot n^{1+\varepsilon}.$$

Proof. Let $\kappa = \frac{1}{1-1/2^\varepsilon} \cdot k$. The base case is easy: $W(1) = k \leq \kappa_1$ as $\frac{1}{1-1/2^\varepsilon} \geq 1$. For the inductive step, we substitute the inductive hypothesis into the recurrence and obtain

$$\begin{aligned}
W(n) &\leq 2W(n/2) + k \cdot n^{1+\varepsilon} \\
&\leq 2\kappa \left(\frac{n}{2}\right)^{1+\varepsilon} + k \cdot n^{1+\varepsilon} \\
&= \kappa \cdot n^{1+\varepsilon} + \left(2\kappa \left(\frac{n}{2}\right)^{1+\varepsilon} + k \cdot n^{1+\varepsilon} - \kappa \cdot n^{1+\varepsilon}\right) \\
&\leq \kappa \cdot n^{1+\varepsilon},
\end{aligned}$$

where in the final step, we argued that

$$\begin{aligned}
2\kappa \left(\frac{n}{2}\right)^{1+\varepsilon} + k \cdot n^{1+\varepsilon} - \kappa \cdot n^{1+\varepsilon} &= \kappa \cdot 2^{-\varepsilon} \cdot n^{1+\varepsilon} + k \cdot n^{1+\varepsilon} - \kappa \cdot n^{1+\varepsilon} \\
&= \kappa \cdot 2^{-\varepsilon} \cdot n^{1+\varepsilon} + (1 - 2^{-\varepsilon})\kappa \cdot n^{1+\varepsilon} - \kappa \cdot n^{1+\varepsilon} \\
&\leq 0.
\end{aligned}$$

□

Solving the recurrence directly. Alternatively, we could use the tree method and evaluate the sum directly. As argued before, the recursion tree here has depth $\log n$ and at level i (again, the root is at level 0), we have 2^i nodes, each costing $k \cdot (n/2^i)^{1+\varepsilon}$. Thus, the total cost is

$$\begin{aligned}
\sum_{i=0}^{\log n} k \cdot 2^i \cdot \left(\frac{n}{2^i}\right)^{1+\varepsilon} &= k \cdot n^{1+\varepsilon} \cdot \sum_{i=0}^{\log n} 2^{-i \cdot \varepsilon} \\
&\leq k \cdot n^{1+\varepsilon} \cdot \sum_{i=0}^{\infty} 2^{-i \cdot \varepsilon}.
\end{aligned}$$

But the infinite sum $\sum_{i=0}^{\infty} 2^{-i \cdot \varepsilon}$ is at most $\frac{1}{1-1/2^\varepsilon}$. Hence, we conclude $W(n) \in O(n^{1+\varepsilon})$.