Lecture 10

Euclidean TSP
Tree decompositions

November 5, 2004
Lecturer: Kamal Jain
Notes: Ioannis Giotis

1.1 Euclidean TSP continued

In the last lecture we considered a bounding box of the points $L \times L$ for $L = 4n^2$ and the resulting grid formed by unit squares. We also showed that at a small cost to the optimum we can move all points to the centers of their respective square and that the tour is not self intersecting using the triangle inequality.

W.l.o.g we assume that $n = 2^k$. We now recursively divide the bounding box into 4 squares. The recursion has $log\ L$ levels and at the last level are the unit squares. On each square we place $m = O\left(\frac{log\ n}{\epsilon}\right)$ *portals* on each edge placed equally distanced (portals on corner points need to be moved by a small amount). We're allowed to enter or exit the square only by its portals and we are going to move a tour's entries and exits to pass through the nearest portals.

Assumption. There exists a TSP tour with cost $(1 + O(\epsilon))OPT$, which enters and exits into squares only through portal points.

Lemma 1.1. *W.l.o.g. we can assume a tour passes through a given portal at most twice.*

Proof. Suppose otherwise. For any subset of 3 edges e_1, e_2, e_3 passing though the same portal, if we remove edges e_1, e_2 on the inside of the square, if the graph is still connected we can get a tour of lesser cost by adding an edge connecting e_1, e_2 endpoints. Otherwise, if the graph is not connected, we reset edge e_1 and remove edge e_3. The resulting graph is connected and we get a tour of lesser cost by adding an edge connecting e_2, e_3 endpoints. □

Since each portal has $0, 1$ or 2 edges, there are $3^{4m} \sim n^{O(1/\epsilon)}$ possible configurations for the portals. We only need the ones that the sum of entries and exits is even.

Consider the following procedure. We select a subset $2r \leq 4m$ of the portals and try to pair the portals. However, since the tour is not self-intersecting the only possibilities for say lined up portals a, b, c, d are 1) $(a - d), (b - c)$ or 2) $(a - b), (c - d)$. But these pairings correspond to balanced arrangements of $2r$ parentheses which is the rth Catalan number $2^{2r} \sim n^{O(1/\epsilon)}$, therefore we only have a polynomial number of problems.

The dynamic programming approach is, given the entry and exit points, to divide the square in 4 and solve the problem until the squares are small enough to apply brute force. At each level we are going through a polynomial number of problems.

Finally, there exist some bad cases where most of the points are very close to a separating line (defining squares) but are alternatively on different sides of the line. To deal with this situation we move the separating lines randomly by a small amount.

1.2 Tree decompositions

In the problem of maximum independent set, we are given a graph G with weights on vertices w_v. Our goal is to find an independent set S such that $\sum_{v \in S} w_v$ is maximized. If G is a tree then the problem can be solved with dynamic programming as follows.

Consider "hanging" the graph from an arbitrary vertex. For each vertex v, let T_v be the subtree rooted at v and f_v the cost of a maximum independent set of T_v which uses v, and g_v the cost of a m.i.s of T_v which doesn't use v. Clearly these values for leaves are equal to the weights of the vertices or 0 respectively. Now, for each vertex v which has children u_i these functions are computable by

$$\begin{aligned} f(v) &= w_v + \sum_i g_{u_i} \\ g(v) &= \sum_i max(g_{u_i}, f_{u_i}) \end{aligned}$$

This problem can also be solved in polynomial time in tree like graphs. These graphs have a tree subgraph as a skeleton and some "flesh" surrounding each tree edge. To define this formally, we define the tree decomposition of a graph G as

- T is a tree subgraph of G. $V(T)$ is the vertex set of T.
- $\forall v \in V(T)$, assign a subgraph $H_v \subseteq G$, such that $\bigcup H_v = G$.
- $\forall u, v, w \in V(T)$ s.t. v is on a unique path $u - w$ then $V(H_u) \cap V(H_w) \subseteq V(H_v)$.

The tree width is defined as the size of $max|H_u| - 1$.

Lemma 1.2. *G has tree width 1 iff G is a forest. Alternatively, a forest doesn't have K_3 as a minor.*

We say that G has H as a minor if we can perform the following procedure.

1. Start with G.
2. Remove an edge from G.
3. Pick two adjacent vertices and contract them.
4. Get H by repeating previous steps as necessary.

Lemma 1.3. *G has tree width 2 iff G is a series-parallel graph. Alternatively, G has tree width 2 iff G doesn't have K_4 as a minor.*

Unfortunately, G has tree width 3 iff G doesn't have K_5 as minor, is not true. To see this, consider the $n \times n$ grid. Another interesting lemma is

Lemma 1.4. *A graph is planar iff it doesn't have K_5 or $K_{3,3}$ as a minor.*

Continued on the next lecture...