CS 684: Approximation and Network Algorithms	**October 31, 2001**
Euclidean Traveling Salesman Problem	**scribe: Davina Kunvipusilkul**

1 Introduction

The Traveling Salesman Problem: Given a graph with weights on edges. We want to visit all the vertices exactly once at minimum cost.

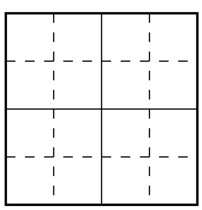
This problem is NP-hard, and for a small enough constant there is also no constant-factor approximation, unless P=NP. Today we will restrict ourselves to Euclidean graphs, and get a approximation scheme, that is an algorithm that for any fixed $\epsilon > 0$, gets a $1+\epsilon$-approximation that is polynomial as long as ϵ is fixed (but may depend exponentially on ϵ). That is, we assume we have as input a set $S \subset \Re^d$ of vertices and we will output $s_1, s_2, \ldots, s_n$, a permutation of S such that $\sum_i ||s_i - s_{i-1}||_2$ is minimal.

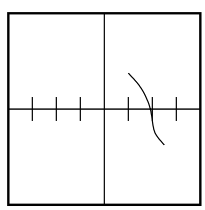
Theorem 1.0.1 (Arora & Mitchell). *There exists a polynomial-time approximation scheme for Euclidean graphs.*

For this lecture, we will focus on the case where $d = 2$. Without loss of generality, we assume that we have n points which lie within a box of size $4n^2$ and that this is the smallest box possible so that the tour will have length at least $4n$; otherwise, we can use scaling. Assume further that every input point is a grid point since moving each point to a grid point will cost at most 1. Thus, moving n points to a grid point will cost at most n, which is negligable (comparing to the tour length).

An Overview of the Algorithm

- Cut each problem into 4 smaller subproblems and recurse.





- Designate certain points in the square at which a tour can enter/leave the square. The algorithm will cross the squares only through these predetermined points.
- Use Dynamic Programming to move from small squares to larger ones. Essentially, we will create a hierarchical structure of $k = O(\log n)$ levels.

2 The Algorithm

Algorithm:

For $1 \leq i \leq \log(4n^2)$

- Define boxes of level i as follows:

Level 0: Big square containing S.

Level $i+1$: Cut each square of level i to 4 equal quarters.

At level i, we have 4^i boxes (2^i horizontal lines and 2^i vertical lines).

With each such line, we associate a number i, the first level at which the line is introduced.

- Define portals for the square as follow:

  For each new line added, put in $m = O(\epsilon^{-1} \log n)$ portals in each line segment so that a line of level i has $m \cdot 2^i$ equally spaced portals. Without loss of generality, we assume all portals sit on coordinates which are powers of 2.

We now have three tasks to do:

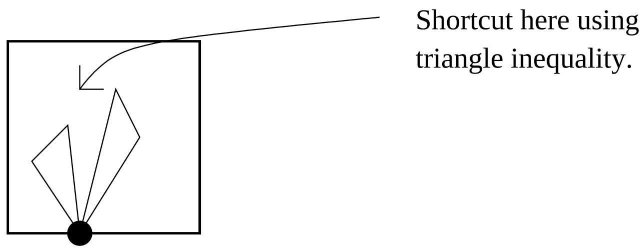
1. Define a notion of legal tours (with respect to the partitions and portals).
2. Show that an optimal legal tour can be found efficiently.
3. Show that there exists a legal tour that approximates the optimal tour.

Definition 2.0.2 (A legal/well-behaved tour). *A "well-behaved tour" is a path in $\Re^2$ that crosses every point of S, returns to the starting point, crosses itself only on portals, and crosses between the squares we defined only on portals.*

A well-behaved tour is "minimal crossing" if it crosses each portal at most twice.

Claim 2.0.3. *For every well-behaved tour, there exists a well-behaved, minimal crossing tour that does not cost more.*

Proof. We will not provide a complete proof here. The idea of the proof is that if the tour crossses a portal more than twice, than it has two subtours on the same side of the portal, and then we can shortcut and use triangle inequality to show that the new tour does not cost more.



□

Lemma 2.0.4. *There is an efficient search (with time $2^{O(m)} poly(n) = O(n^{O(\epsilon^{-1})} poly(n))$ for a minimal cost well-behaved, minimal crossing tour.*

Proof. Given a tour τ, it defines a pattern of visits on the portal of every box. Consider the order in which the portals were being crossed. This order pairs up the portals, every time the tour enters the square on a portal, the portal is matched to where the tour exits. We will define a pattern of visits to be the number of times each portal is crosses (at most twice for any portal) along with this pairing of the crossings. We claim that for every square, the number of patterns that may be generated by a well-behaved, minimal crossing τ is $2^{O(m)}$. This is as there are at most 3^m different possible number of crossings. Now we have to estimate the number of legal pairings. There are at most $2m$ crossings, and there can be more than $M!$ pairings of these crossings. However, the tour may not cross itself. This implies that the pairing forms a sort of parethesis structure: if a crossing at a portal x is matched to a crossing at a portal y, than the crossings between x and y muct be matched to each other. It is not hard to show that there are at most $2^{O(m)}$ such non-crossing pairings.

We will construct a dynamic programming table that will hold for every square and for every consistent pattern, the cost of the optimal well-behaved, minimal crossing tour of $S \cap \text{Box}$ that meets this pattern. In other word, the pattern tells us where the tour enters and exits the square; however, we need to figure out what it does inside the square. When there are no points in the square, than the tour can proceed from the entering crossing straight to the matched exit crossing (if it enters at all). When there is only one point in the square, it is obvious as to what the best path is, given the entry portal and the exit portal, one of the paths has to pick up the point, the other proceed straight to the exit.

For larger squares, we can use information on these smaller squares to build solutions for larger squares. Our table will have one entry for each square in our subdivision. Figuring out the optimal solution for each entry will require examining the patterns, but since there are only $2^{O(m)}$ of them, the running time will still be acceptable. To bound the number of subproblems recall that the depth of the recursion is $k = O(\log n)$, so there are only $poly(n)$ subsquares to consider, and there are $2^{O(m)}$ different crossing patters for each subsquare.

See Figure 1 for the tree of points. At the bottom, each box has at most 1 points and only $|S|$ leaves have points in them.

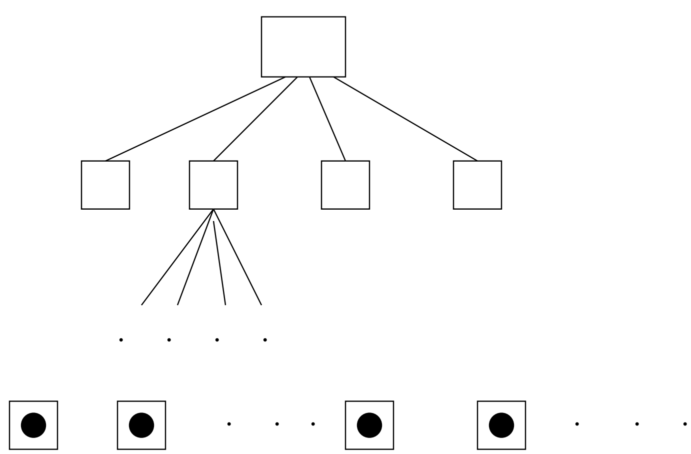
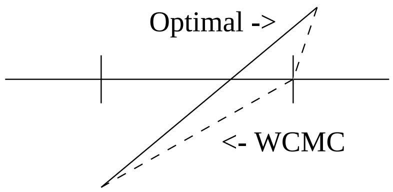


Figure 1: The tree of points.

□

3 Why can a well-behaved, minimal crossing tour approximate the optimal tour?

So far we have presented an algorithm which runs in $O(n^{O(\epsilon^{-1})} poly(n))$ and which gives us a tour. In this section, we will give a sketch on how well our tour approximate the optimal tour. Clearly, the optimal tour should not cross itself. We still need to show that the restriction of having the tour cross on portals only will not affect the lenth of the tour by too much. However, the presented algorithm is deterministic and it is possible to create an input which the algorithm will perform poorly. To remedy the situation, we will use randomization to randomly shift the lines dividing the boxes.



At each line crossing, we add at most the distance between 2 neighboring portals.

Thus, to determine the total error, we need to estimate the number of lines we cross and the distance between portals on these lines.

1. How many lines do we cross?

   When we go from one point to another, the worst case is that we cross all horizontal and vertical lines. That is, we cross at most L_1 distance $\leq 2 \cdot L_2$ distance lines. Thus, we cross at most $2 \cdot Opt$ lines.

2. How much is the distance between portals on these lines?

   We will shift the horizonal and vertical lines uniformly at random with wraparound (to take care of the case where some points may spill over). Since there are more lines of deep levels (with more portals), shifting lines uniformly at random makes it more likely for the tour to cross lines of deeper levels in the tree where there are more portals. That is, the expected distance between portals is approximately $\frac{1}{m}$. With m as $\epsilon^{-1} \cdot \log n$, we get that the expected loss is at most $2\epsilon \cdot Opt$.