This week, paper [2] by Arora. See the slides for figures.
See also `http://www.cs.princeton.edu/~arora/pubs/arorageo.ps`

Algorithms for Euclidean TSP

Introduction

This lecture is about the polynomial time approximation scheme (PTAS) for the traveling salesman problem (TSP) in the Euclidean plane developed by Arora [1]. The survey paper by the same author [2] is a good reference. Also, the book by Vazirani [5] and the book by Shmoys and Williamson [4] contain a chapter on this algorithm.
This result contains many technical details. When you browse through the versions listed above you will see many differences. In the earliest version (1996) the running time was $n^{O(1/\epsilon)}$. This was improved to $n(\log n)^{O(1/\epsilon)}$ in [1]. We will only discuss the first, less efficient result. (If you come across the terms 'patching lemma' and '(m, r)-light tours' in the literature then these refer to the improved algorithm that is not discussed here.)
All references listed here are excellent but I think these notes can be helpful since in [5] many details are missing (and are left as exercise) while the other references are long and cover the more advanced algorithm completely. These notes covers the easiest form completely.

Karp's algorithm (See slides) What is the most obvious approach for designing a polynomial time approximation scheme for Euclidean TSP? Cut the problem in smaller pieces, solve the small problems, and then combine everything to get one solution. The nice thing about working in the Euclidean plane is that cutting the metric space in small pieces is simple: Draw a square B around all input points and then cut the square in smaller squares. This was exactly what Karp did in his TSP algorithm for the Euclidean plane [3]. In general, the TSP on n points can be solved exactly in time $O(n^3 2^n)$ by a simple dynamic program. (DP: For every subset S of the points and points $u, v \in S$ store the length of the smallest path that visits all points in S and has u and v as endpoints. The value can be computed from the values for $S \setminus v$ with endpoints u and v' where v' takes all values in $S \setminus \{u, v\}$.) So if we split up the square B in small rectangles Q_i, each containing $s = \log_2 n$ input points, then we can find an optimal TSP tour T_i for each of these subproblems in time $O(s^3 2^s)$ time. There are n/s rectangles so the total time to solve all these subproblems is $O(ns^2 2^s) = O(n^2 s^2)$, which is polynomial in n. The next step of Karp's algorithm is to pick one input point v_i in each rectangle Q_i and then make a tour T on $\{v_1, v_2, \ldots, v_{n/s}\}$. For this we can use for example Christofides' algorithm (or even Few's algorithm). Now a TSP tour follows from combining the big tour T with the small tours T_i.
From a worst case point of view this algorithm performs poorly. However, if we assume that the input is formed by taking n points uniformly at random in a square, then the expected ratio is $1 + o(1)$. In other words, it goes to 1 for $n \to \infty$.

For more details see the slides of this week.

Arora's PTAS for Euclidean TSP

The algorithm of Karps described above is deterministic and performs well on a random input. Arora's algorithm is randomized and performs well an *any* input. Moreover, de-randomization is easy, although at the cost of an increased running time.
The randomization is a simple step in the algorithm but it is an important part of the analysis. The random step makes it possible to do the dynamic programming in polynomial time.

The main ingredients

The main ingredients of the algorithm are (in arbitrary order):

(A) Rounding the instance.

(B) Restricting the set of feasible solutions.

(C) Dynamic programming.

(D) Randomization.

Below, we will sketch the basics of these ideas. The algorithm is given in detail later. The randomization is actually the first step of the algorithm but it is easier to explain the idea of this step here at the end.

(A) Rounding the instance This is done by defining a grid and move each input point to the middle of the grid cell in which it is contained. Since we only want a $1+\epsilon$ approximation and not the true optimum we can afford to change the instance a little bit. The solution that we find is used for the original instance as well: just maintain the same ordering of points. The advantage of shifting points is that it simplifies the dynamic programming: For the DP we divide the plane into small squares and solve subproblems inside the squares. The effect of rounding (i.e. shifting points) is twofold: (i) there are no input points on the boundary of squares and (ii) the smallest squares (the grid cells) contain at most 1 input point.
The density of the grid is important. A denser grid gives a smaller error but leads to a higher running time.

(B) Restricting the set of feasible solutions. This is another way to speed up the dynamic programming. Instead of finding the optimum over all feasible solutions we find the optimum over a smaller set of feasible solutions which satisfy some condition. This restriction enables us to do the dynamic programming in polynomial time. Of course one should prove that the optimum over the restricted set of solutions is not far from the true optimum. In fact, it can be far off but the randomization (D) ensures that the difference is small in expectation.

The basic idea of this restriction is as follows. On each grid line we place a set of points that we call *portals* and add the restriction that the tour can only cross a grid line at a portal.

(C) Dynamic programming. By dynamic programming we find the true optimum over all restricted solutions (B) for the rounded instance (A). First, draw one square that contains all input points. Call this the *enclosing box* B. Divide the square into four equal sized squares. Then, divide each of the four squares into four equal sized squares. Keep dividing squares into four equal sized squares until each of the smallest squares contains at most one input point. This division defines a tree: The root is the largest square B and its four children are the four squares in which it is divided. In general, each inner vertex of the tree has degree four and that is why it was called the *quad tree* in [1]. The squares that correspond with the vertices of the tree are called *dissection* squares.
The DP starts at the leaves of the tree. Computations for the leaves of the tree (the smallest squares) become easy since they contain at most input point. In general, each dissection square defines a polynomial number of subproblems in the DP. The optimal values of any subproblem is found by looking up a polynomial number of values for subproblems of its four children in the tree. Starting from the leaves we fill the DP table until we reach the root.

(D) Randomization It is a simple but powerful step of the algorithm. Instead of taking an arbitrary enclosing box (which forms the root of the tree) we take a box uniformly at random from a set of boxes. In this way, the restriction made in (B) becomes randomized. The effect is that the restriction only gives a small change in the optimal value in *expectation*.

The algorithm

In this section we describe the algorithm in detail.

The steps of the algorithm:

1. Take a random square B which covers all input points.
2. Rounding: move each point to the middle of the grid cell it is in.
3. Build the quad tree.
4. Define portals.
5. Build the dynamic program table.
6. Fill the table.

Step 1: First we take a smallest square that covers all input points. The square may not be unique but any choice is fine. Call this the *bounding box*. Redefine the distances such that the side length of the box becomes 2^{k-1} with $k = \lceil \log_2(n/\epsilon) \rceil$. Let $(0, 0)$ be its lower left corner. Now take $a, b \in \{0, \ldots, 2^{k-1} - 1\}$ independently and uniformly at random. Let $L = 2^k$ and let $(-a, -b)$ be the lower left corner of the $L \times L$ enclosing box B. Note that the bounding box has side length $L/2$ and that it is covered by the enclosing box B for any outcome of a and b. Also note that $L = 2^k \geqslant n/\epsilon$

Step 2: Place an $L \times L$ grid over B and move each input point to the middle of the grid cell that it is in. If a point is on a grid line then choose one adjacent cell. From now on we only consider this rounded instance.

Step 3: The box B is the root of the tree and we say it is of *level* 0. It is divided into four equal sized squares of level 1. These are its children in the quad tree. Keep dividing until the smallest squares have size 1×1. These are the leaves of the tree and they are of level k (by definition of k). We say that the root has highest level and the leaves have the lowest level. The squares of the tree are called *dissection squares*. We also define levels for all inner grid lines. Denote the horizontal middle grid line and vertical middle grid line as *level* 1 lines. Hence, these are the two lines that divide the box B into its four children. In general, the level i grid lines are those lines that divide the level $i - 1$ squares of the tree each into four level i squares. So horizontally we have 2^{i-1} level i lines and the same number vertically, $i = 1, \ldots, k$.

Step 4: Each inner grid line gets a number of equidistant portals. The number depends on the level of the line. A level i lines gets $m2^i - 1$ portals such that these points divide the line into $m2^i$ segments of equal length. The portals on the boundary of a dissection square are then given by the portals on the grid lines that bound the square. For a level i square, two of its sides are level i grid lines and the other two sides are of higher level. That means that two sides have exactly $m + 1$ portals and the other sides both have at most $m/2 + 1$ portals. Then, the number of portals per square is at most $4m$ if $m \geqslant 4$. Let $m = \lceil k/\epsilon \rceil$. (Arora reduced this to $m = O(1/\epsilon)$ in [1].)
We say that a solution is *portal respecting* if it crosses grid lines only at portals.

Step 5: By dynamic programming we find the smallest portal respecting tour. We shall prove below that the optimal portal respecting tour crosses each portal at most twice. For the DP it is convenient to think of a portal as two portals: each is crossed at most ones. There is an entry in the DP table for every dissection square and for every possible way that an optimal solution may cross the boundary of this square. For each entry of the table we will store one value, which is the length of the fraction inside the square of the optimal solution satisfying the boundary conditions. A property of any optimal TSP tour in the plane is that it does not cross itself. That means that the part inside a square is a set of paths that do not cross and visit all input points inside.
Formally, an entry in the dynamic program table is given by a triple (X, Y, Z). Here, X ranges over all dissection squares, Y is any even subset P of the portals of X, and Z is a valid *pairing* for this subset P. A valid pairing is a partitioning of P in pairs such that we can connect each pair of points by a path inside the squares without crossings.

Step 6: For each entry (X, Y, Z) we compute the length of the shortest set of paths visiting all input points inside X and that satisfies the boundary conditions Y and Z. Denote this value by $F(X, Y, Z)$. If X is a leaf of the tree then the value can easily be computed: If there is no input point inside X then the optimal solution is to connect each pair in Z by a straight line and if there is one input in X then one path will visit the point and the other paths are straight lines.
If X is not a leaf, then the value $F(X, Y, Z)$ can be computed by looking at all values for the children of X in the tree, say X_1, X_2, X_3, X_4. Consider any quadruple (X_1, Y_1, Z_1), (X_1, Y_2, Z_2), (X_3, Y_3, Z_3), (X_4, Y_4, Z_4). We say that it is consistent with (X, Y, Z) if (i) portals are consistent and (ii) the pairing is consistent. With (i) we mean that if two squares (for example X and X_1 or X_1 and X_2) have a boundary in common then on this part they use the same set of portals. With (ii) we mean that the pairing Z should be consistent with the pairing that is defined by the pairings Z_1, Z_2, Z_3, Z_4. For example, assume that p_1 and p_2 are portals in Y and assume that in Z_1 portal p_1 is paired with some portal q and in Z_2 portal q is paired with p_2. Then p_1 and p_2 should be paired in Z. Further, we should check that the pairings do not define a subtour, unless Y is the emptyset in which case the four pairings should add up to a single tour, which is the case when X is the root.

Lemma 1 *Let T be an optimal portal respecting tour for the rounded instance. Show that T crosses each portal at most twice.*

Proof. Exercise.

The analysis

We need to check that the algorithm can be implemented in polynomial time and that the approximation guarantee is $(1 + \epsilon)\mathsf{OPT}$. We start with the latter.

Analysis of the approximation ratio

To prove that the algorithm is a PTAS it is enough to prove that the ratio is $1 + O(\epsilon)$. We need to check that rounding the instance and restricting to portal respecting tours does not change the optimal value to much.
Let OPT be the optimal value of the original instance and let OPT' be the optimal value of the rounded instance, say I'.

Lemma 2 $|\mathsf{OPT} - \mathsf{OPT}'| \leqslant \epsilon\sqrt{2}\mathsf{OPT}$.

Proof. The bounding box has side length $L/2$ and was taken as small as possible. Hence, $\mathsf{OPT} \geqslant L$. Further, $L \geqslant n/\epsilon$ which implies $\mathsf{OPT} \geqslant n/\epsilon$. The maximum distance by which a point is moved by the rounding step is $\sqrt{2}/2$. Therefore,

$$|\mathsf{OPT} - \mathsf{OPT}'| \leqslant n\sqrt{2} \leqslant \epsilon\sqrt{2}\mathsf{OPT}.$$

□

Let Π be an optimal tour for I'.

Lemma 3 *The tour* Π *has at most* $\sqrt{2}\mathsf{OPT}'$ *crossings with grid lines.*

Proof. Tour Π is formed by n straight line segments. Consider any such segment of length s. Assume it goes from (x_1, y_1) to (x_2, y_2). The number of crossings with vertical grid lines is $|x_1 - x_2|$ and it has at most $|y_1 - y_2|$ crossings with horizontal grid lines. The sum is

$$|x_1 - x_2| + |y_1 - y_2| \leqslant \sqrt{2}s.$$

Hence the total number of crossings is at most $\sqrt{2}\mathsf{OPT}'$. □

Let $\mathsf{OPT}''_{a,b}$ be the optimal value of the portal respecting tour for the rounded instance I' given the random choices a and b and let $E[\mathsf{OPT}'']$ be the expected value over random choices a and b of the optimal solution for instance I'.

Lemma 4

$$\mathsf{OPT}' \leqslant E[\mathsf{OPT}''] | \leqslant (1 + \epsilon\sqrt{2})\mathsf{OPT}'.$$

Proof. Let δ_i be the distance between two neighboring portals on a level i grid line, call this the *inter portal distance*. Then

$$\delta_i = \frac{L}{m2^i} = \frac{2^k}{m2^i}.$$

Given the optimal tour Π for I' we make it portal respecting by moving each crossing with a grid line to the nearest portal. The length of such detour is at most twice the distance to the nearest portal.
Consider an arbitrary crossing of Π with some grid line l. The inter portal distance depends on the level of the line, which is determined by the randomization step of the algorithm. The probability that this grid line l becomes a level 1 line is exactly $1/(L/2) = 1/2^{k-1}$. Also, the probability that this grid line l becomes a level 2 line is exactly $1/2^{k-1}$. For any $i \geqslant 2$ this probability is $1/2^{k-i+1}$. Hence, for any $i \geqslant 1$, the probability that a grid line becomes a level i line is at most

$$\frac{1}{2^{k-i}}.$$

The expected length of the detour for moving the crossing with l to the nearest portal is at most

$$\sum_{i=1}^{k} \delta_i \cdot \frac{1}{2^{k-i}} = \sum_{i=1}^{k} \frac{2^k}{m2^i} \cdot \frac{1}{2^{k-i}} = \frac{k}{m} = \frac{k}{\lceil k/\epsilon \rceil} \leqslant \epsilon.$$

By Lemma 3, there are at most $\sqrt{2}\mathsf{OPT}'$ crossings with grid lines. So the total expected detour for making Π portal respecting is at most $\epsilon\sqrt{2}\mathsf{OPT}'$. □
By combining Lemma 2 and Lemma 4 we get the desired result.

Corollary 1 $|\mathsf{OPT} - E[\mathsf{OPT}'']| = O(\epsilon)\mathsf{OPT}$.

Analysis of the running time

The running time of the algorithm is dominated by the dynamic programming. We need to show that the size of the DP table is polynomially bounded and that each value of each entry can be computed in polynomial time.
The number of dissection squares is $O(4^k) = O(4^{\log_2(n/\epsilon)}) = O(n^2/\epsilon^2)$. Each square has no more than $4m$ portals. Remember that we copied each portal to enhance the DP. This gives at most $8m$ portals per square. The number of possible subsets is at most $2^{8m} = (n/\epsilon)^{O(1/\epsilon)} = n^{O(1/\epsilon)}$. Given an even subset Y of the portals, the number of valid pairings is at most $2^{|Y|}$. This upper bound is obtained as follows. Assume we are given a valid pairing. Now choose one of the corners of X and walk one round around the boundary of X. Every time that a portal of Y is encountered label it 0 if it comes before its paired portal in this walk and label it 1 otherwise. This gives a $0,1$-vector of length $|Y| \leqslant 8m$. If we fix the starting point, then every pairing gives a *unique* $0,1$-vector. This follows from the fact that valid pairings have no crossings. Hence, given Y the number of valid pairings is at most $2^{8m} = n^{O(1/\epsilon)}$. Over all, the number of entries in the DP table is upper bounded by $O(n^2/\epsilon^2) \times n^{O(1/\epsilon)} \times n^{O(1/\epsilon)} = n^{O(1/\epsilon)}$.
Now we compute an upper bound on the computation time of $F(X, Y, Z)$. If X is a leaf then we only need to check $|Y|/2$ solutions: exactly one of the $|Y|/2$ pairs is connected with the input point (if any) and the other pairs are connected by straight lines. If X is not a leaf then the value $F(X, Y, Z)$ can be computed by considering all combinations of entries of its four children. The number of possible combinations is $(n^{O(1/\epsilon)})^4 = n^{O(1/\epsilon)}$.
We have shown that the size of the DP table is $n^{O(1/\epsilon)}$ and that it takes $n^{O(1/\epsilon)}$ time to compute one value. Therefore the total running time of the algorithm is

$$n^{O(1/\epsilon)} \cdot n^{O(1/\epsilon)} = n^{O(1/\epsilon)}.$$

Questions for this week:

1. Prove Lemma 1.
2. Describe how to modify Arora's PTAS to get a PTAS for the following problem: Let K be some integer *constant*. Given a set of points in the plane, find K tours which together visit all points. Goal is to minimize the sum of the lengths. (Only describe the important changes you make to Arora's algorithm and proof.)

References

[1] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45:753–782, 1998.

[2] S. Arora. Approximation schemes for NP-hard geometric optimization problems: A survey. *Mathematical Programming*, 97:43–69, 2003.

[3] R. Karp. Probalistic analysis for partitioning algorithms for the traveling salesman problem in the Euclidean plane. *Mathematics of Operations Research*, 2, 1977.

[4] D. Shmoys and D. Williamson. *The design of approximation algorithms*. Cambridge University Press, 2011.

[5] V. Vazirani. *Approximation Algorithms*. Springer, Berlin, 2001.