CHAPTER 12

Approximating the Euclidean Traveling Salesman Problem (TSP)

> 'Would you know how to calculate the diameter of the globe?'
>
> 'No, I'm afraid I wouldn't,' answered Svejk, 'but I'd like to ask you a riddle myself, gentlemen. Take a three-storied house, with eight windows on each floor. On the roof there are two dormer windows and two chimneys. On every floor there are two tenants. And now, tell me, gentlemen, in which year the house-porter's grandmother died?'
>
> – The good soldier Svejk, Jaroslav Hasek.

In this chapter, we introduce a general technique for approximating the shortest traveling salesperson tour in the plane (i.e., TSP). This technique found wide usage in developing approximation algorithms for various problems. We will present two different variants of the technique, the first is quadtree based, and is faster and easier to generalize to higher dimension. The other variant will be presented in the next chapter, and is slower but seems to be somewhat stronger.

12.1. The TSP problem – Introduction

Let P be a set of n points in the plane. We would like to compute the shortest tour visiting all the points of P. That is, it is the shortest closed polygonal chain where its vertices are all the points of P. We will refer to such a chain as a ***tour***. The problem of computing the shortest TSP tour is NP-HARD, even in the plane with Euclidean distances. In the general graph setting (where the points are vertices of a complete graph, and every edge has an associated weight with it) no approximation is possible. For the metric case, where the weights comply with the triangle inequality, a $3/2$ approximation is known. However, in the low dimensional Euclidean case, where the distances between points are just the Euclidean distance, a PTAS is known.

The problem had attracted a vast amount of research in computer science, due to it simplicity and the ability to easily draw and inspect a solution. In fact, for small instances people can do a decent job in solving such problems manually.

A nice historical example of TSP is Lincoln's tour of Illinois in 1850. Lincoln worked as a lawyer traveling with a circuit court. At the time, a circuit court would travel through several cities, where in each city they would stop and handle local cases, spending a few days in each city. The tour that the circuit court (and Lincoln) took in 1850 is depicted in Figure 1 (a) (they started from Springfield and returned to it in the end of the tour). Clearly, it is very close to being the shortest TSP of these points. In this case, there is a shorter solution, but it is not clear that it was shorter as far as traveling time in 1850, see Figure 1 (b).

More modern applications of TSP vary from control systems for telescopes (i.e., the telescope uses the minimum amount of motion while taking picture of all required spots in the sky during one night), to scheduling of rock or political tours, and many other applications.

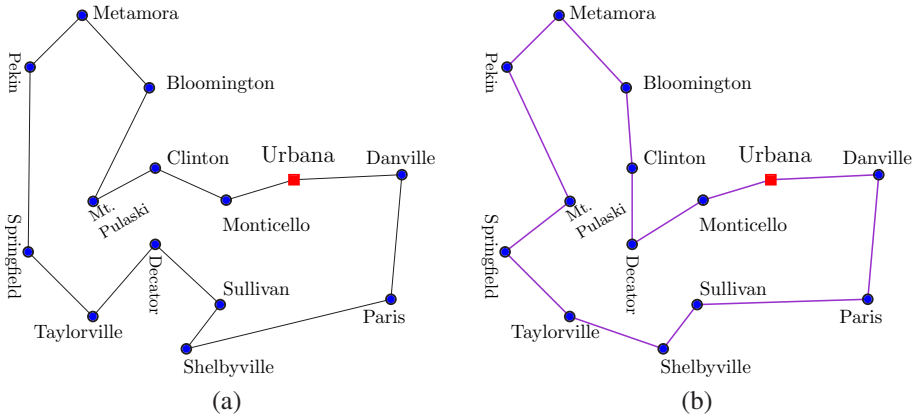


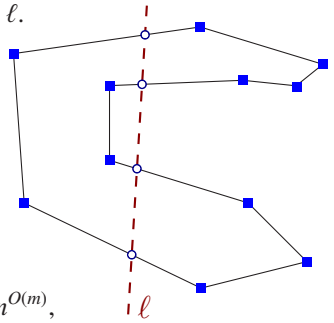
FIGURE 1. (a) Lincoln tour in 1850 in Illinois, and (b) the optimal TSP.

12.2. When the optimal solution is friendly

Here we are interested in the Euclidean variant. We are given a set P of n points in the plane, with Euclidean distance, and we are looking for the shortest TSP tour.

First try – divide and conquer. A natural approach to this problem, would be to try and divide the plane into two parts, say by a line, solve (a variant of) the problem recursively on each side of the line, and then stitch the solutions together. So let ℓ be such a separating vertical line, having at least one point of P on each size of it, and we divide the problem into two subproblems along the line ℓ.

To this end, we need to guess where the optimal TSP crosses the separating line. Let us assume, for the time being, that the TSP crosses this separating line at most m times, where m is some small constant. To solve the left (resp. right) subproblem, we need to specify how the TSP crosses the "iron curtain" formed by ℓ.



The optimal TSP path is made of segments connecting two points of P. As such, there are $N = \binom{n}{2}$ potential segments that the TSP might use. Choosing the m intersection points of the tour with ℓ, is equivalent to choosing the (at most) m segments of the TSP tour crossing the splitting line ℓ. As such, the number of possibilities is

$$\sum_{i=0}^{m} \binom{N}{i} \leq 2\left(\frac{Ne}{m}\right)^m = N^{O(m)} = n^{O(m)},$$

by Lemma 12.15$_{\text{p183}}$.

Note, however, that it is not sufficient to solve the problem on the left and on the right by just knowing the segments crossing the middle line. Indeed, whether or not a solution on the left subproblem is valid depends on the solution on the right subproblem, see Figure 2. As such, for every subproblem, we do not only have to specify the points where the tour enters the subproblem (we will refer to these points as ***checkpoints***), but also the connectivity constraints for the checkpoints of each subproblem.

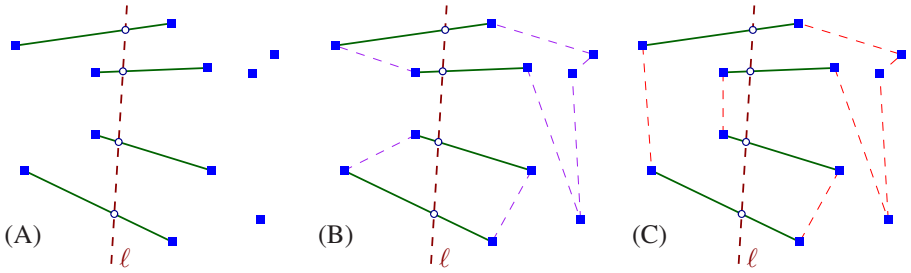
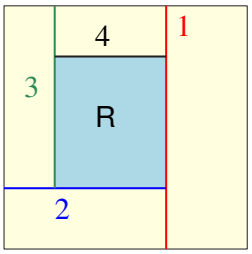
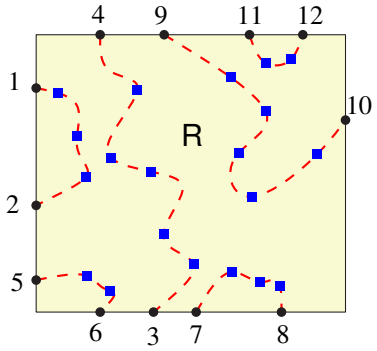


FIGURE 2. (A) The guessed middle solution fed into the two subproblems. (B) A solution to the left subproblem resulting in a disconnected tour. (C) A valid solution.



Before going into the exact details of how to do this, imagine continuing recursively in this fashion. Namely, we try to break the problem recursively into smaller and smaller regions. To this end, we will cut each subproblem by (say) a vertical line, alternating between horizontal and vertical line's. A rectangle R generated by a sequence of such cuts is depicted on the right. A subproblem is thus a rectangle R having checkpoints on its boundary, with ordering constraints (i.e., in which order does the tour visit these checkpoints), where we need to find a solution of total minimal length connecting the right checkpoints to each other and spanning all the points inside R.

Our intention is to solve this problem using dynamic programming. To this end, we would like to minimize the number of subproblems one needs to consider. Note that the cutting lines we use can be ones that pass through the given points. As such, there are $2n$ possible cutting lines, since only horizontal or vertical lines are considered. Every subrectangle is defined by the 4 cutting lines that form its boundary, and as such there $(2n)^4 = O(n^4)$ possible rectangles that might appear in a subproblem.



So, consider such a subproblem with a rectangle R with t checkpoints on its boundary. Specifying the order in which the checkpoints are being visited by the tour, can be done by numbering these endpoints. Namely, for a specific rectangle with t checkpoints on its boundary, there are $t!$ possible subproblems that one needs to compute a solution for. An example of such a numbering and one possible solution compatible with this numbering is depicted on the right. Here, the point numbered 1 is the first visited by the tour when it enters R, then the tour leaves the rectangle through the endpoint 2, reenters it using 3, and so on. In the figure, the squares are the points of P inside this rectangle.

One can perform some additional checks on a given instance of a subproblems to verify that it is indeed realizable and should be considered. For example, the total number of times the tour enter/leave a subrectangle is the same.

In fact, the number of possible subproblems one has to consider, for such a rectangle R, is somewhat smaller than $t!$, as the following exercise testifies. However, since it is still exponential it will be easier for us to use the naive bound $t!$.

Problem 12.1. Given a rectangle R and t checkpoints specified on its boundary. Show that there are at least $2^{t/4}$ different patterns for a path to enter and leave R using these endpoints. This holds even if we require the path not to self intersect. Here, a pattern is a matching of the t checkpoints, such that if p_i is matched to p_j, where the path enters the rectangle R at p_i and leaves through p_j (or vice versa).

This implies that if t is large (say $\Omega(n)$) then the running time of the algorithm is inherently exponential. We first define formally the partition scheme the above approach implies, and then we show an algorithm for the case where this partition is "good".

In the following, we will treat rectangles as being half-opened; that is, an axis-parallel rectangle will be a region of the form $[x_1, x_2)\times[y_1, y_2)$. We next define a bounding rectangle for the entire point set, such that there no points are on its boundary.

Definition 12.2. For a point set P in the plane, its ***frame*** is the rectangle $[x_1 - 1, x_2 + 1) \times [y_1 - 1, y_2 + 1)$, where $[x_1, x_2] \times [y_1, y_2]$ is the smallest axis parallel closed rectangle containing P.

Definition 12.3. Let P be a set of n points in the plane, and let R be its frame. An ***RBSP*** (rectilinear binary space partition) is a recursive partition of R into rectangles, such that each resulting rectangle contains a single point of P. Formally, an axis parallel rectangle containing a single point is a valid RBSP. As such, an RBSP of R and the point set P, is specified by a splitting line ℓ (which is either horizontal or vertical) that passes through some point of P, and RBSPs of both $\mathsf{R}^+ = \ell^+ \cap \mathsf{R}$ and $\mathsf{R}^- = \ell^- \cap \mathsf{R}$, where ℓ^- and ℓ^+ are the two halfplanes defined by ℓ.[①] Here we required that $\mathsf{R}^+ \cap \mathsf{P} \neq \emptyset$ and $\mathsf{R}^- \cap \mathsf{P} \neq \emptyset$.

A rectangle considered during this recursive construction for a given RBSP is a ***subrectangle*** of this RBSP.

Since we always take cutting lines that pass through the given points, there are $2n + 4$ lines that can be used by a subrectangle (the vertical and horizontal lines through the points, and the 4 lines defining the frame of P). Which implies that there are at most $O(n^4)$ subrectangles that can participate in an RBSP of the given point set.

Definition 12.4. Let P be a set of n points in the plane, and let π be a tour of P. The tour π is ***t-friendly***, for a parameter $t > 0$, if there exists an RBSP $\mathcal{R}$ of the bounding rectangle of P (for the point set P), such that π crosses the boundary of every subrectangle of $\mathcal{R}$ at most t times.

Theorem 12.5. *Given a point set* P *and a parameter* $t > 0$*, then one can compute the shortest t-friendly tour of* P *in* $n^{O(t)}$ *time.*

Proof. The algorithm is recursive, and we will use memoization to turn it into a dynamic programming algorithm. Let L be the set of $2n + 4$ vertical and horizontal lines passing through the points of P and the four lines used by the frame of P. Let $\mathcal{F}$ be the set of all $O(n^4)$ rectangles defined by the grid induced by the lines of L.

An instance of the recursive call is defined by (R, U, M), where R is a rectangle of $\mathcal{F}$, U is a set of at most t checkpoints placed on the boundary of R, and M is a permutation of U (this is the order in which the tour visits the checkpoints). As such, there are at most

$$O\Big(n^4 \cdot n^{2t} \cdot t!\Big) = n^{O(t)}$$

[①] Formally, one of them is open, the other is closed, such that R^+ and R^- are rectangles, and they are defined in a consistent way with the frames above.

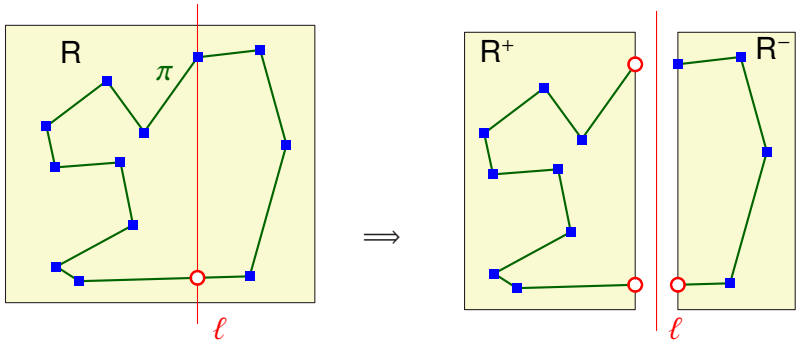


FIGURE 3. The top level split of the RBSP of the optimal solution.

different recursive calls. Indeed, there are $O(n^4)$ possible subrectangles. For a given subrectangle, a checkpoint is the intersection of a segment defined by two input points and the boundary of the rectangle. Since the boundary of the rectangle might intersect such a segment twice, the number of possible checkpoints for a specified rectangle is $2\binom{n}{2}$. Now, if a given subrectangle has i checkpoints, then there are $\left(2\binom{n}{2}\right)^i$ possible sets of checkpoints that it could have, and $i!$ orderings for each set of checkpoints. Since a tour visiting the rectangle can use up to t checkpoints, a subrectangle has at most $\sum_{i=2}^{t}\left(2\binom{n}{2}\right)^i i! \leq 2n^{2t} \cdot t!$ different instances of the subproblems defined for it.

The recursive call (R, U, M) computes the best possible solution to this instance by trying all possibilities to split it. As a base case, if $|\mathsf{P} \cap \mathsf{R}| \leq t$, then we solve this instance using brute force. This takes $t^{O(t)}$ time, by just trying all possible tours restricted to the points inside R and the checkpoints. For each such solution, we can easily verify that it complies with the given constraints.

Otherwise, we try all ways to split it by a line $\ell \in \mathsf{L}$. We require that $\ell^+ \cap \mathsf{P} \neq \emptyset$ and $\ell^- \cap \mathsf{P} \neq \emptyset$, where ℓ^+ and ℓ^- are the two halfplanes induced by ℓ. For such a line, we compute the two subrectangles $\mathsf{R}^+ = \mathsf{R} \cap \ell^+$ and $\mathsf{R}^- = \mathsf{R} \cap \ell^-$, and the splitting segment $s = \mathsf{R} \cap \ell$. Next, we enumerate all possibilities of the at most t checkpoints on s. For each such set U' of checkpoints, we enumerate all possible subproblems defined for R^+ and R^- using the checkpoints of $U \cup U'$.

For any two such subproblems for R^+ and R^- we verify that they are consistent with the given larger instance for R. In particular, the given instance R has a given permutation on it checkpoints, and the two permutations for the subinstances R^+ and R^- need to be consistent with this permutation, and need to be to be consistent on the shared checkpoints along the splitting line. Note, that each one of these permutations has at most t checkpoints in it, and as such the check can be easily done in time $t^{O(t)}$. If so, we recursively compute the shortest solution for these two subproblems. We combine the two recursive solutions into a solution for the given problem for R; that is, we add up the price of these two solutions.

We try all such possible solutions, and we store the cheapest solution found as the solution to this subproblem.

The work associated with each such subproblem is $n^{O(t)}$ (ignoring the work in the recursive calls). Indeed, we generate all possible cuts (i.e., n), and enumerate all possible subsets, up to size t, of checkpoints on the cutting line (i.e., $n^{O(t)}$). Then, we try all possible permutations on the two subproblems, and verify that they are consistent which the parent

permutation and between the two subproblems (this takes $t^{O(t)}$ time). Finally, if found such a consistent instance for the two subproblems, we recursively compute their optimal solution.

This implies that the total running time is $n^{O(t)}$, since the number of different subproblems is $n^{O(t)}$.

The initial call, uses the frame of P, and has no checkpoints (note that this is the only feasible instance in all the recursive calls without checkpoints).

The consistency check throughout the recursive execution of the algorithm guarantee that the solution returned by the algorithm is indeed a tour of the given input points. Now, let π denote the shortest t-friendly tour of P. Consider an RBSP that π is t-friendly for. Clearly, the algorithm would consider during the recursive call this RBSP and would consider π as one possible solution to the given instance. Indeed, this specific RBSP defines a recursion tree over subproblems. For each such subproblem, we know exactly how the optimal solution behaves in these subproblems and uses their checkpoints, see Figure 3 for an example. As such, the algorithm would return the portions of π for each subproblems of this RBSP, which glued together yields π.

To this end, one can use simple induction to argue that inside each such subproblem the algorithm computed the optimal solution (complying with the exact constraints on the checkpoints, and how they should be visited as induced by the optimal solution). As such, the solution returned by the algorithm is at least as good as the optimal solution (there might be several equivalent optimal solutions). Namely, it would return π is the computed solution (assuming, of course, that the optimal solution in this case is unique). □

It is not too hard to verify that any point set has a tour that is 2-friendly (if we do not require that vertical and horizontal cuts alternate). Intuitively, as t increases the length of the shortest t-friendly tour decreases. Assume, that we could show that the shortest t-friendly tour is of length $\leq (1 + 1/t) \left\|\pi_{\text{opt}}\right\|$, where π_{opt} is the shortest TSP tour of the given point set, and $\left\|\pi_{\text{opt}}\right\|$ is its length. Then, the above algorithm would provide us with a $(1+\varepsilon)$ approximation algorithm, with running time $n^{O(1/\varepsilon)}$, by setting $t = 1/\varepsilon$.

Surprisingly, showing that such a t-friendly tour exists is not trivial, and will be the undertaking of the next sections. In fact, we will use similar concepts that use the underlying idea of limited interaction between subproblems to achieve a PTAS for the approximation problem.

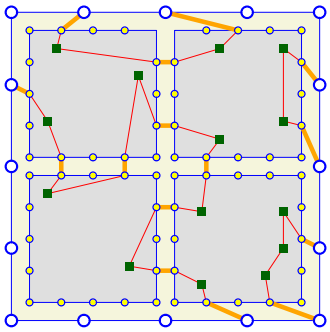
12.3. TSP Approximation via portals and sliding quadtrees

12.3.1. Portals and sliding – idea and intuition. The basic idea is to do a direct divide and conquer by imposing a grid on the point set. We will compute the optimal solution inside each grid cell, and then stitch the solutions together into a global solution. As such, the basic subproblem we will look at is a square. To reduce the interaction between subproblems, we will place m equally spaced points on each boundary of a square, and require the solution under consideration to use these ***portals***. Naturally, we will have to snap the solution to these portals.

For efficient implementation, the stitching has to be done in a controlled fashion that involves only a constant number of subsquares being "stitched" together. The most natural way to achieve this is by using a hierarchical grid; that is, a quadtree.

It might be that the boundary of a large square of the grid intersects the optimal solution many times. As such, the snapping to portals might introduce a huge error. To avoid

this, we will randomly translate the quadtree/grid. This will distribute the snapping error in a more uniform way in the quadtree. The exact details of this are described below.



The picture on the right depicts a square and a solution for it using its portals, and how this solution propagates to the four subsquares. (The yellow region is of zero width and is enlarged for the sake of clarity.)

12.3.2. The algorithm.

12.3.2.1. *Normalizing the point set.* The first stage is to snap the points to a grid, such that the resulting point set has a bounded polynomial spread. This would guarantee that the quadtree for the point set would have logarithmic depth. Naturally, this would introduce some error and has to be done carefully so that the error is sufficiently small.

So, let P be a set of n points in the plane for which we would like to compute the TSP tour, and $1 > \varepsilon > 1/n$ a prespecified constant. Assume, that P is contained in the square $\left[1/2, 1\right]^2$ and $\operatorname{diam}(\mathsf{P}) \geq 1/4$. This can be guaranteed by scaling and translation, and clearly given a solution to this instance we can map it to the original point set.

Note, that the optimal solution π_{opt} has to be of length at least $\operatorname{diam}(\mathsf{P}) \geq 1/4$, and at most $n\sqrt{2}/2$, as the TSP is made out of n segments, none of which can be longer than the diameter of the square $\left[1/2, 1\right]^2$. Let $\mathcal{E} = \lceil 32/\varepsilon \rceil$. Consider the grid

$$\mathsf{G} = \left\{ \frac{1}{n\mathcal{E}} (i, j) \;\middle|\; i, j \text{ are integers} \right\}$$

and snap each point of P to its closest point on this grid (if several points gets snapped to the same grid point, we treat the snapped points as a single point). Clearly, every point of P is moved by at most a distance of $z = \sqrt{2}/n\mathcal{E}$. As such, solving the problem for the snapped point set can be translated back to the original point set. Indeed, by taking this tour and for each snapped point walking out to the its true location and back, we get a tour of the original points which is not much longer. Specifically, this introduces an error of length at most $2z$ per point (as each point needs to move from its true location to its snapped location and back), and overall an error of

$$n \cdot 2z = 2n \frac{\sqrt{2}}{n\mathcal{E}} \leq \frac{4\varepsilon}{32} \leq \frac{\varepsilon}{2} \left\| \pi_{\text{opt}} \right\|, \tag{12.1}$$

since $\left\| \pi_{\text{opt}} \right\| \geq 1/4$. Let Q denote the resulting point set.

12.3.2.2. *The Dynamic Programming over the quadtree.* We randomly select a point $(x, y) \in [0, 1/2]^2$. Consider the translated canonical grid

$$\mathsf{G}^i = (x, y) + \mathsf{G}_{2^{-i}}, \tag{12.2}$$

for $i = 0, 1, 2, \ldots$; that is, G^i is a grid with side length 2^{-i} with its "origin" at (x, y).

Construct (a regular) quadtree $\mathcal{T}$ over Q using the square $\square = [x, 1+x] \times [y, 1+y] \in \mathsf{G}^0$ as the root (note, that Q is contained inside $\square$). Since the diameter of Q is larger than $1/4$, and the minimal distance between a pair of points of Q is at least $1/n\mathcal{E}$, the height of the quadtree is going to be at most

$$H \leq 1 + \lceil \log_2 n\mathcal{E} \rceil \leq 6 + \lceil \log_2 (n/\varepsilon) \rceil = O(\log n).$$

Along each edge of a quadtree node, we will place

$$m = O(H/\varepsilon) = O\left(\frac{\log n}{\varepsilon}\right)$$

equally spaced points, such that the path entering/exiting the square must use one of these ***portals*** (we will also add in the four corners of the square). Every portal can be used at most twice. Indeed, if a portal is being used more than twice by a tour, it can be modified to use it at most twice (see Remark 12.7 below for a proof of this). Furthermore, the path can use at most

$$k = O(1/\varepsilon)$$

such portals on each side of the square.

This (uncompressed) quadtree has $O(nH) = O(n \log n)$ nodes. For each such square a subproblem is specified by how the tour interact with the portals. We consider its four children, generate all the possible ways the tour might interact with the four children, verify that information is consistent across the subproblems and with the parent constraints, and we recursively solve these subproblems. For each such consistent collection of four subproblems, we add up the costs of the solutions of the subproblems and set it as a candidate price to the parent instance. We try all possible such solutions, and return the cheapest one.

One minor technicality is that the portals of the children are not necessarily compatible with the parent portals, even if they lie on the same edge. As such, the recursive program adds to the cost of the subproblem the price of the tour moving from the parent portals to the children portals.

Naturally, if a square in the quadtree has only $O(1/\varepsilon)$ points of Q points stored in it, then given a subproblem for this square, the algorithm would solve the problem directly by brute force enumeration. Of course, only solutions that visit all the points in the square would be considered.

Thus, by using memoization, this recursive algorithm becomes a dynamic programming algorithm. The resulting dynamic programming algorithm is similar to the one we saw for the t-friendly case, with the difference here that the portals are fixed, and the partition scheme we use is determined by the quadtree and is as such simpler.

We claim, that the resulting tour of the points of Q is a $(1+\varepsilon)$-approximation to the shortest TSP tour of Q. (To get a $(1+\varepsilon)$-approximation to TSP for the original point set, we need to use a slightly smaller approximation parameter, say $\varepsilon/10$. For the sake of simplicity we will ignore this minor technicality.)

12.3.3. Analysis.

12.3.3.1. *Running time analysis.*

Number of subproblems. Given a square and its portals, a tour needs to list the portals it uses in the order it uses them. The square has $4m+4$ portals on its boundary, and each portal can be used at most twice. To count this in naive fashion, we duplicate every portal twice, and select $i \leq 8k$ of them to be used. We then select one of the $i!$ different orderings these portals might be visited by the tour. Overall, the total number of different ways such a tour might interact with a square is bounded by

$$T = \sum_{i=0}^{8k} \binom{2(4m+4)}{i} i! \leq 8k \binom{2(4m+4)}{8k} (8k)! \leq \left(\frac{(4m+4)e}{8k}\right)^{8k} = (\log n)^{O(1/\varepsilon)},$$

since $\binom{N}{i} \leq (Ne/i)^i$ (see Lemma 12.16$_{\text{p183}}$).

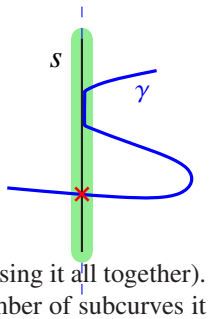
Overall running time. It easy to verify that each subproblem can be solved recursively in $T^{O(1)}$ time, ignoring the time it takes the solve the recursive subproblems. Since the quadtree has $O(n \log n)$ nodes, we conclude that the algorithm has overall running time

$$n \log^{O(1/\varepsilon)} n.$$

12.3.3.2. *Quality of approximation.* We need to argue that the optimal solution for Q can be made to comply with the constraints above. Namely, subproblems can connect to one another only by using the portals, such that no more than k portals are used on each side of a subsquare, out of the $4m + 4$ portals allocated to each subsquare.

We will first argue that restricting the optimal TSP to cross each side of a square at most k times introduces a small error. This will be done by arguing that one can patch a path so that it crosses every such square edge only a few times. We will argue that this increases the cost of the tour only mildly. Next, we will argue that moving the paths to use the portals incurs only a low cost.

Fit the first – the patching lemma. In the following, we consider a segment as having two sides. Consider a curve π traveling along a segment s, such that it enters s on one side and leaves s on the same side. Conceptually, the reader might want to think of s as being a very thin rectangle[②]. We will not consider this to be a crossing of s by π, see figure on the right for an example of a curve that crosses s once. Formally, a subcurve γ ***crosses*** a segment s, if $\gamma \setminus s$ is made out of two non-empty connected components, and these two connected components lie on different sides of the line supporting s (of course π might go from one side of s to the other side of s by bypassing it all together). The number of times a curve crosses a segment s is the maximum number of subcurves it can be broken into, such that each one of these subcurves crosses s.



Lemma 12.6. *Let π be a closed curve crossing a segment s at least three times. One can replace it by a new curve π' that is identical to π outside s, that crosses s at most twice, and its total length is at most $\|\pi\| + 4\|s\|$.*

Proof. Conceptually, think about s as being a vertical thin rectangle. Let $\mathsf{p}_1, \ldots, \mathsf{p}_k$ (resp. $\mathsf{q}_1, \ldots, \mathsf{q}_k$) be the k intersection points of π with the left (resp. right) side of s. We build an Eulerian graph having $V = \{\mathsf{p}_1, \ldots, \mathsf{p}_k, \mathsf{q}_1, \ldots, \mathsf{q}_k\}$ as the vertices. Let $s_i = \mathsf{p}_i\mathsf{p}_{i+1}$ and $s'_i = \mathsf{q}_i\mathsf{q}_{i+1}$, for $i = 1, \ldots, k-1$.

Let E_π be the edges formed by taking each connected component of $\pi \setminus s$ as an edge.

Consider the multi-set of edges

$$E = \left\{s_1, \ldots s_{k-1}, s'_1, \ldots s'_{k-1}\right\} \bigcup \{s_1, s_3, \ldots\} \bigcup \left\{s'_1, s'_3, \ldots\right\} \bigcup \{\mathsf{p}_1\mathsf{q}_1\} \cup E_\pi,$$

and the connected graph they form $\mathsf{G} = (V, E)$. If k is odd then all the vertices in the graph have even degree. As such, it has an Eulerian tour, that crosses s exactly once using the zero length bridge $\mathsf{p}_1\mathsf{q}_1$, and is identical to π outside s, and as such it's the required path. (Naturally, this implies that the modified tour crosses s and goes around s to form a closed tour.)

Otherwise, if k is even, then p_k and q_k are the only odd degree vertices in G. We modify G by adding $\mathsf{p}_k\mathsf{q}_k$ as an edge (which has zero length). Clearly, the resulting graph

[②]Naturally, the reader might not want to think about this segment at all. Be as it may, this myParagraph deals with this segment and not with the reader wishes. Hopefully the reader would not become bitter over this injustice.

is now Eulerian, and the same argument goes through. (In this case, however, there are exactly two crossings of s.)

The total length of the edges of this graph (and as such of the resulting curve) is at most

$$\|\pi\| + 2\Big(\text{total length of odd segments along } s\Big) + 2\|s\| \le \|\pi\| + 4\|s\|,$$

as required. □

By being more careful in the patching, the resulting path can be made to be of length at most $\|\pi\| + 3\|s\|$, see Exercise 12.1.

Remark 12.7. Applying Lemma 12.6 to a segment of length zero, we conclude that an optimal solution needs to use a portal at most twice.

Second fiddle – properties of sliding grids. Consider a segment s of length r, and a grid G^i. We claim that the expected number of intersections of s with G^i, for $i \ge 1$, is roughly $\|s\|/2^{-i}$, where 2^{-i} is the sidelength of a cell of the grid G^i. Intuitively, this implies that the number of times a grid intersects a segment is a good estimator of its length (up to the right scaling). More importantly, it implies that if a segment is very short, and the grid is large (i.e., $\|s\| \ll 2^{-i}$), then the probability of a segment to intersect the grid is small, and is proportional to its length. We first prove this in one dimension.

Lemma 12.8. *Let s be an interval on the x-axis, and let x be a random number picked in range $[0, 1/2]$. For $i \ge 1$, consider the periodic point set $T[x] = \Big\{ x + j/2^i \;\Big|\; j \text{ is an integer} \Big\}$. (The set $T[z]$ can be interpreted as the set of intersections of G^i, centered at z in the x coordinate, with the x-axis.)*

If $\|s\| \le 2^{-i}$ then the probability of s to contain a point of $T[x]$ is $\dfrac{\|s\|}{\text{sidelength}(\mathsf{G}^i)} = 2^i\|s\|$.

Furthermore, the expected number of intersections of s with $T[x]$ (i.e., $|s \cap T[x]|$) is exactly $2^i\|s\|$, even if $\|s\| \ge 2^{-i}$.

Proof. Assume that $\|s\| \le 2^{-i}$ and consider the set

$$U = \Big\{ x \;\Big|\; T[x] \cap s \ne \emptyset \Big\}.$$

Observe that U is a periodic set. Indeed, if $y = x + j/2^i \in U = T[x] \cap s$ then $y = (x+1/2^i)+(j-1)/2^i \in T[x+1/2^i] \cap s$. Thus $x+1/2^i$ is also in U. As such, $U = \bigcup_{j \in \mathbf{Z}} (s + j/2^i)$. Since U is periodic (with period $1/2^i$), the required probability is

$$\mathbf{Pr}\Big[T[x] \cap s \ne \emptyset\Big] = \frac{\|U \cap [0,1/2]\|}{\|[0,1/2]\|} = \frac{\Big\|U \cap [0,1/2^i]\Big\|}{\Big\|[0,1/2^i]\Big\|} = \frac{\|s\|}{1/2^i} = 2^i\|s\|.$$

As for the second claim, break s into m sub-segments $s_1, \ldots, s_m$, such that each one of them is of length at most 2^{-i-1}. Note, that each s_j can contain at most one point of $T[x]$, and let X_j be an indicator variable that is one if this happens, for $j = 1, \ldots, m$. Clearly, the number of points of $T[x]$ in s is exactly $X_1 + X_2 + \cdots + X_m$. By linearity of expectations, we have that

$$\mathbf{E}\Big[|T[x] \cap s|\Big] = \mathbf{E}\left[\sum_{j=1}^{m} X_j\right] = \sum_{j=1}^{m} \mathbf{E}\Big[X_j\Big] = \sum_{j=1}^{m} \mathbf{Pr}\Big[T[x] \cap s_j \ne \emptyset\Big] = \sum_{j=1}^{m} 2^i \Big\|s_j\Big\| = 2^i\|s\|.$$

□

Lemma 12.9. *Let s be a segment in the plane. For $i \geq 1$, the probability that s intersects the edges of the grid G^i is bounded by $\sqrt{2}\,\|s\|$. Furthermore, the expected total number of intersections of s with the vertical and horizontal lines of G^i is in the range $\left[2^i\,\|s\|,\ \sqrt{2}\cdot 2^i\,\|s\|\right]$.*

Proof. Let $\mathcal{I}_x$ and $\mathcal{I}_y$ be the projections of s into the x and y axises, respectively. By Lemma 12.8, the probability that a vertical (resp. horizontal) line of G^i intersects $\mathcal{I}_x$ (resp. $\mathcal{I}_y$) is bounded by $2^i\,\|\mathcal{I}_x\|$. By Lemma 12.17$_{\text{p183}}$, we have that $\|\mathcal{I}_x\| + \left\|\mathcal{I}_y\right\| \leq \sqrt{2}\,\|s\|$ and

$$\mathbf{Pr}\Big[s \text{ intersects an edge of } \mathsf{G}^i\Big] \leq 2^i\,\|\mathcal{I}_x\| + 2^i\left\|\mathcal{I}_y\right\| \leq \sqrt{2}\cdot 2^i\,\|s\|\,.$$

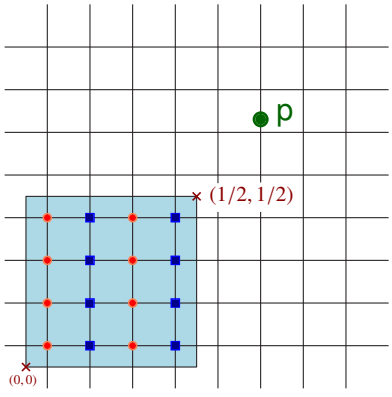
As for the second claim, the expected number of intersections of s with the horizontal (resp. vertical) lines of G^i is, by Lemma 12.8, exactly $2^i\left\|\mathcal{I}_y\right\|$ (resp. $2^i\,\|\mathcal{I}_x\|$). As such, the expected number of intersections of s with the grid lines is exactly $2^i\left\|\mathcal{I}_y\right\| + 2^i\,\|\mathcal{I}_x\|$ (note, that s passes through a vertex of the shifted grid is zero). Now, $\|s\| \leq \left\|\mathcal{I}_y\right\| + \|\mathcal{I}_x\| \leq \sqrt{2}\,\|s\|$, which implies the claim. □

Remark 12.10. Note that the bounds of Lemma 12.9 hold even if s is a polygonal curve by breaking it into segments and using linearity of expectation.

We also need the following easy observation.

Claim 12.11. *Let E_i be the union of the open edges of G^i (i.e., it does not include the vertices of G^i). Then, for a point p in the plane, we have that $\mathbf{Pr}\Big[\mathsf{p} \in E_{i-1} \;\Big|\; \mathsf{p} \in E_i\Big] = 1/2$.*

Intuitively, every line of E_i has probability half to survive and be a line of E_{i-1}. The formal argument requires a bit more care.



Proof. Assume that p lies on a vertical line of G^i, as the other possibility follows by similar argumentation. Consider the points of G^i in the square $[0, 1/2)^2$. There are $\Big((1/2)/2^{-i}\Big)^2 = 2^{2(i-1)} > 2$ such points. Each of these points has the same probability to be the shift (x, y) used in generating G^i, see Eq. (12.2). For exactly half of these points, if they had been chosen to be the shift then p would be on an edge of G^{i-1}, see figure on the right. □

The third wheel – the cost of reducing the number of intersections. We patch the path[③] starting in the lowest canonical grid G^H (i.e., the bottom of the quadtree), then fix the resulting path in G^{H-1} (i.e., the next level of the quadtree) and so on, till we reach the canonical grid G^0. Let π_i denote the resulting path after handling the grid G^i. As such, π_{H+1} is just the original optimal TSP π_{opt}, and π_1 is the resulting patched tour (note that G^0 does not intersect the generated path).

The patching is done as follows. In the ith step, for $i = 1, \ldots, H$, we start with π_{H+2-i} and consider the grid G^{H+1-i}. If an edge of the grid intersects π_{H+2-i} more than k times, then we patch it, using Lemma 12.6. We repeat this process over all the edges of the grid, and the resulting tour is G^{H-i}. Here k is a parameter of this patching process.

[③] Say it quickly a 100 times.

Since we are doing the patching process bottom up, every edge of the grid G^{H+1-i} corresponds to two edges of G^{H+2-i}. Each of these two edges intersects the tour at most k times, but together they might intersect this merged edge more than k times, requiring a patch.

A minor technicality is that we assume that π_{opt} does not pass through any vertex of the grid G^H. Note, that since we randomly shifted the quadtree, the probability that any grid vertex of G^H would lie on π_{opt} is zero.

Intuitively, this patching process introduces only low error because when we fix and edge of a grid so that the tour does not intersect it too many times, the number of times the patched tour crosses boundaries of higher level nodes of the quadtree also goes down. Thus, fix-ups in low levels of the quadtree help us also in higher levels. Similarly, the total number of crossings (of the tour with the grids) drop exponentially as we use larger and larger grids, thus requiring less fix-ups. Thus, intuitively, one can think about all the patching happening in the bottom level of the quadtree. The formal argument, below, is slightly more involved and one has to be careful, because we are dealing with expectations.

Claim 12.12. *Let π_{opt} be the optimal TSP of Q. There exists a tour π_1 that is a TSP of Q that crosses every side of a square of the quadtree $\mathcal{T}$ at most k times, and the total length of π_1, in expectation, is $\leq (1 + 8/(k-2))\left\|\pi_{\text{opt}}\right\|$.*

Proof. The algorithm to compute this modified path is outlined above. We need to analyze how much error this patching process introduced.

Let Y_i denote the number of times the path π_{i+1} crosses the edges of the canonical grid G^i. When the path π_{i+1} crosses an edge of this grid more than k times, we apply the patching operation (i.e., Lemma 12.6), so that it crosses this edge at most twice. Let F_i be the number of patching operations performed at this level, in generating the path π_i from π_{i+1}.

Note, that after all the fix-ups are applied to π_{i+1} in the grid G^i, it has at most

$$n_i \leq Y_i - (k-2)F_i$$

crossings with the edges of the grid G^i, for $i \geq 1$. Indeed, every patching of an edge removes at least k crossings and replaces it with at most two crossings. As such, since the probability of a grid line of G^i to be a grid line of G^{i-1} is *exactly* $1/2$, by Claim 12.11, we have that

$$\mathbf{E}[Y_{i-1}] = \mathbf{E}\left[\mathbf{E}\left[Y_{i-1} \;\middle|\; n_i\right]\right] = \mathbf{E}\left[\frac{n_i}{2}\right] \leq \mathbf{E}\left[\frac{Y_i - (k-2)F_i}{2}\right] = \frac{1}{2}\mathbf{E}[Y_i] - \frac{k-2}{2}\mathbf{E}[F_i]$$

$$\implies \quad \mathbf{E}[F_i] \leq \frac{1}{k-2}(\mathbf{E}[Y_i] - 2\,\mathbf{E}[Y_{i-1}])\,.$$

The price of each fix-up in the grid G^i is $4/2^i$, by Lemma 12.6. As such, the expected total cost of this patching operation is

$$\begin{aligned}
\mathbf{E}[\text{error}] = \mathbf{E}\left[\sum_{i=1}^{H} \frac{4F_i}{2^i}\right] &\leq \mathbf{E}\left[2F_1 + \sum_{i=2}^{H} \frac{4F_i}{2^i}\right] = \mathbf{E}[2F_1] + \frac{4}{k-2}\sum_{i=2}^{H} \frac{(\mathbf{E}[Y_i] - 2\,\mathbf{E}[Y_{i-1}])}{2^i} \\
&\leq \mathbf{E}\left[2\frac{Y_1}{k-2}\right] + \frac{4}{k-2}\left(\frac{\mathbf{E}[Y_H]}{2^H} - \frac{\mathbf{E}[Y_1]}{2}\right) \\
&= \frac{4}{k-2}\left(\frac{\mathbf{E}[Y_1]}{2} + \frac{\mathbf{E}[Y_H]}{2^H} - \frac{\mathbf{E}[Y_1]}{2}\right) = \frac{4}{k-2}\cdot\frac{\mathbf{E}[Y_H]}{2^H}.
\end{aligned}$$

since $F_1 \leq Y_1/k \leq Y_1/(k-2)$. The number Y_H is the number of times π_{opt} crosses the edges of the bottom grid G^H. By Remark 12.10 and Lemma 12.9 we have that $\mathbf{E}[Y_H] \leq 2 \cdot 2^H \left\|\pi_{\text{opt}}\right\|$. As such, we have that

$$\mathbf{E}[\text{error}] \leq \frac{4}{k-2} \cdot \frac{\mathbf{E}[Y_H]}{2^H} = \frac{4}{k-2} \frac{2 \cdot 2^H \left\|\pi_{\text{opt}}\right\|}{2^H} = \frac{8}{k-2} \left\|\pi_{\text{opt}}\right\|. \qquad \square$$

The fourth wall – bounding the price of snapping the tour to the portals. We finally need to bound the price of snapping the tour to the portals. We remind the reader that every edge e of a square of the quadtree has $m+2$ (two of them are on the endpoints of the edge) equally spaced portals on it. As such, the distance between an intersection of the tour (with the edge) and its closest portal (on the edge) is at most $\|e\|/2(m+1)$. As such, a single snapping operation on e introduces an error of twice the snapping distance by the triangle inequality; that is $2(\|e\|/2(m+1)) = \|e\|/(m+1)$.

Note, that if an edge of a grid has portals from several levels, the snapped tour uses only the portals on this edge that belong to the highest level.

Lemma 12.13. *The error introduced by snapping π_1 so that it uses only portals in all levels of the quadtree is bounded, in expectation, by $\frac{2H}{m+1}\left\|\pi_{\text{opt}}\right\|$.*

Proof. Let Z_i be the number of crossings of π_{opt} with G^i, for $i \geq 1$. Clearly, the tour π_1 has less crossings (because of the patching operations) with G^i than π_{opt}. Thus, we have that the expected price of snapping π_1 to the portals of the quadtree (summed over all the levels) is bounded by

$$\sum_{i=1}^{H} Z_i \cdot \frac{1/2^i}{m+1}.$$

As such, in expectation this error is bounded by

$$\begin{aligned}\mathbf{E}[\text{error}_2] &\leq \mathbf{E}\left[\sum_{i=1}^{H} \frac{Z_i}{2^i(m+1)}\right]. \leq \sum_{i=1}^{H} \mathbf{E}\left[\frac{Z_i}{2^i(m+1)}\right] = \sum_{i=1}^{H} \frac{\mathbf{E}[Z_i]}{2^i(m+1)} \\ &\leq \sum_{i=1}^{H} \frac{2 \cdot 2^i \left\|\pi_{\text{opt}}\right\|}{2^i(m+1)} = \frac{2H}{m+1} \left\|\pi_{\text{opt}}\right\|\end{aligned}$$

by Remark 12.10 and Lemma 12.9. $\square$

The fifth elephant – putting things together. Normalizing the point set might cause the optimal solution to deteriorate by a factor of $1 + \varepsilon/2$, see Eq. (12.1). Forcing the tour to cross every edge of the quadtree at most k times, increases its length by a factor of $1 + 8/(k-2)$, see Claim 12.12. Finally, forcing this tour to use the appropriate portals for every level, increases its length by a factor of $1 + 2H/(m+1)$, see Lemma 12.13. Putting everything together, the expected length of the optimal solution when forced to be in the form considered by the dynamic programming is at most

$$\left(1+\frac{\varepsilon}{2}\right)\left(1+\frac{8}{k-2}\right)\left(1+\frac{2H}{m+1}\right)\left\|\pi_{\text{opt}}\right\| \leq \left(1+\frac{\varepsilon}{2}\right)\left(1+\frac{\varepsilon}{10}\right)^2 \left\|\pi_{\text{opt}}\right\| \leq (1+\varepsilon)\left\|\pi_{\text{opt}}\right\|,$$

by selecting $k = 90/\varepsilon = O(1/\varepsilon)$ and $m \geq 20H/\varepsilon = \Theta((\log n)/\varepsilon)$. As such, we proved the following theorem.

Theorem 12.14. *Let P be a set of n points in the plane. One can compute, in $n \log^{O(1/\varepsilon)} n$ time, a path π that visits all the points of P, and whose total length (in expectation) is $(1+\varepsilon)\left\|\pi_{\text{opt}}\right\|$, where π_{opt} is the shortest TSP visiting all the points of P.*

12.4. Bibliographical notes

A beautiful historical survey of the TSP problem is provided in the first chapter of the book by Applegate *et al.* [**ABCC07**]. This chapter is available online for free, and I highly recommend to the interested reader to bless it with her attention (i.e., read it).

The approximation algorithm of Section 12.3 is due to Arora [**Aro98**]. An alternative technique by Mitchell [**Mit99**] is described in the next chapter. Arora's technique had a big impact and a lot of follow-up work came out of it, from faster algorithms [**RS98**], to k-median clustering algorithms [**ARR98, KR99**].

A nice presentation of the Arora technique and some of the algorithms using it is provided by a survey of Arora [**Aro03**]. Our presentation follows to some extent his presentation. However, we argued directly on the shifted quadtree, while Arora uses instead a clever argument about lines in the grid the expect error that each one of them introduces. In particular, our direct proof of Claim 12.12 seems to be new, and we believe it might be marginally simpler and more intuitive than the analysis of Arora.

On the number of tours in a subproblem. Concerning the number of different subproblems one has to consider for a given rectangle and t checkpoints (i.e., Proposition 12.1). This is equal to the number of matchings of t points in strictly convex position, which is related to the Catalan number. It is known that the number of non-crossing matchings of n points (not necessarily in convex position) in the plane is $2^{\Theta(n)}$, see [**SW06**].

But is it practical? The clear answer seems to be a resounding no. Finding a PTAS to a problem will usually leaves us with a impractical algorithm. It does however implies that a PTAS exists. At this point, one should start hunting for a more practical approach that might work better in practice.

Much research went into solving TSP in practice. Algorithms that work well in practice seems to be based on integer programming coupled with various heuristics. See [**ABCC07**] for more details.

12.5. Exercises

Exercise 12.1. (Better shortcutting.)

Prove a slightly stronger version of Lemma 12.6. Specifically, given a closed curve π crossing a segment s at least three times, prove that one can replace it by a new curve π' that is identical to π outside s that crosses s at most twice, and its total length is at most $\|\pi\| + 3\|s\|$.

Exercise 12.2. (TSP with neighborhoods.)

Given a set of n unit disks $\mathcal{F}$, we are interested in the problem of computing the minimum length TSP that visits all the disks. Here, the tour has to intersect each disk somewhere.

(A) Let P be the set of centers of the disks of $\mathcal{F}$. Show how to get a constant factor approximation for the case that $\operatorname{diam}(\mathsf{P}) \leq 1$.

(B) Extend the above algorithm to the general case.

12.6. From previous lectures

Lemma 12.15. *For $n \geq 2\delta$ and $\delta \geq 1$, we have $\left(\frac{n}{\delta}\right)^{\delta} \leq \mathcal{G}_\delta(n) \leq 2\left(\frac{ne}{\delta}\right)^{\delta}$, where $\mathcal{G}_\delta(n) = \sum_{i=0}^{\delta} \binom{n}{i}$.*

Lemma 12.16. *For any positive integer n, the following holds.*

(i) $(1 + 1/n)^n \leq e$. *(ii)* $(1 - 1/n)^{n-1} \geq e^{-1}$.

(iii) $n! \geq (n/e)^n$. *(iv)* *For any* $k \leq n$, *we have* $\left(\frac{n}{k}\right)^k \leq \binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.

Lemma 12.17. *For any* $\mathsf{p} \in \mathbb{R}^{\mathsf{d}}$, *we have that* $\|\mathsf{p}\|_1 / \sqrt{\mathsf{d}} \leq \|\mathsf{p}\|_2 \leq \|\mathsf{p}\|_1$.

Bibliography

[ABCC07] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2007.

[Aro98] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. *J. Assoc. Comput. Mach.*, 45(5):753–782, Sep 1998.

[Aro03] S. Arora. Approximation schemes for np-hard geometric optimization problems: a survey. *Math. Prog.*, 97:43–69, 2003.

[ARR98] S. Arora, P. Raghavan, and S. Rao. Approximation schemes for Euclidean k-median and related problems. In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 106–113, 1998.

[KR99] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the Euclidean κ-median problem. In *Proc. 7th Annu. European Sympos. Algorithms*, pages 378–389, 1999.

[Mit99] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28:1298–1309, 1999.

[RS98] S. Rao and W. D. Smith. Improved approximation schemes for geometric graphs via "spanners" and "banyans". In *Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 540–550, 1998.

[SW06] Micha Sharir and Emo Welzl. On the number of crossing-free matchings, cycles, and partitions. *SIAM J. Comput.*, 36(3):695–720, 2006.