Nearly Linear Time Approximation Schemes for Euclidean TSP and other Geometric Problems

Sanjeev Arora*
Princeton University

Abstract

We present a randomized polynomial time approximation scheme for Euclidean TSP in $\Re^2$ that is substantially more efficient than our earlier scheme in [2] (and the scheme of Mitchell [21]). For any fixed $c > 1$ and any set of n nodes in the plane, the new scheme finds a $(1+\frac{1}{c})$-approximation to the optimum traveling salesman tour in $O(n(\log n)^{O(c)})$ time. (Our earlier scheme ran in $n^{O(c)}$ time.) For points in $\Re^d$ the algorithm runs in $O(n(\log n)^{(O(\sqrt{d}c))^{d-1}})$ time. This time is polynomial (actually nearly linear) for every fixed c, d. Designing such a polynomial-time algorithm was an open problem (our earlier algorithm in [2] ran in superpolynomial time for $d \geq 3$).

The algorithm generalizes to the same set of Euclidean problems handled by the previous algorithm, including Steiner Tree, k-TSP, k-MST, etc, although for k-TSP and k-MST the running time gets multiplied by k.

We also use our ideas to design nearly-linear time approximation schemes for Euclidean versions of problems that are known to be in P, such as Minimum Spanning Tree and Min Cost Perfect Matching.

All our algorithms can be derandomized, though the running time then increases by $O(n^d)$ in $\Re^d$. They also have simple parallel implementations (say, in NC^2).

1 Introduction

The quest to understand the *Traveling Salesman Problem* has motivated research in many areas — such as linear programming, polyhedral theory, NP-completeness, probabilistic analysis of algorithms, approximation algorithms, and MAX SNP-completeness (see [18]).

However, one paradox about the TSP remains unexplained: even though most theoretical results point to the problem's computational nastiness, in practice heuristics solve it quite well. Versions of the Lin-Kernighan heuristic [19] run —after careful tuning— in nearly *linear* time and often produce tours whose cost is within a few percent of the optimum (Johnson and McGeoch [13]).

Potentially, this paradox could have a simple explanation since many "real-life" instances are *geometric* (see for example the TSPLIB library [24], a testbed for much experimental work on the TSP), and complexity-theoretic results may not apply to them. For example, *Euclidean TSP* — the version in which the nodes lie in $\Re^2$ and distance is measured using the Euclidean or ℓ_2 norm— is not known to be *MAX-SNP*-hard, in contrast to *Metric TSP* (Papadimitriou and Yannakakis[23]). Thus one could hope to resolve the above-mentioned paradox by designing an excellent *approximation algorithm* for geometric TSP. However, until recently, we knew of no polynomial-time algorithm better than the Christofides algorithm from 1976 [7], which provably computes a tour of cost at most 1.5 times the optimum.

Recently, Euclidean TSP was shown to have a *polynomial time approximation scheme* or PTAS (Arora [2] and some time later, Mitchell [21]). For any fixed $c > 1$ and any n points in $\Re^2$, this algorithm in $n^{O(c)}$ time finds a tour of cost at most $(1 + 1/c)$ times the optimum. Arora's algorithm also generalizes to points in $\Re^d$, but its running time rises to $n^{(O(c\log n))^{d-2}}$, which is *quasipolynomial* in n for every fixed c, d. (Mitchell's algorithm seems to work only in $\Re^2$.) With the discovery of this algorithm, Euclidean TSP joined a select group of NP-hard problems that have PTAS's, a group that includes Subset-Sum (Ibarra and Kim [12]) and Bin-Packing (de la Vega and Lueker [8]; see also Karmarkar and Karp [14])

Though theoretically significant, the recent PTAS for Euclidean TSP is impractical. The constant inside the big-O is about 15, implying a running time n^{30} for 1.5-approximation (that is, when $c = 2$). By contrast, the classical double-MST and Christofides heuristics run in $O(n^{4/3})$ and $O(n^{2.5})$ time respectively[1] and compute 2-approximations and 1.5-approximations re-

*Supported by NSF CAREER award NSF CCR-9502747 and an Alfred Sloan Fellowship. Email: `arora@cs.princeton.edu`

[1] These running times for the classical heuristics, though often quoted, are overstated. If one is willing to settle for solutions

spectively. An even tougher competitor in practice is the Lin-Kernighan heuristic, which often produces good quality solutions in near-linear time.

In this paper we greatly improve our earlier PTAS. We describe a randomized PTAS that, for any fixed $c > 1$ and any set of n nodes in the plane, finds a $(1+\frac{1}{c})$-approximation to the optimum traveling salesman tour in time $O(n(\log n)^{O(c)})$. For points in $\Re^d$ the running time increases to $O(n(\log n)^{(O(\sqrt{d}c))^{d-1}})$. For any fixed c, d this running time is $O(n\mathrm{poly}(\log n))$, which is nearly linear in n. We regret, however, that the "poly$(\log n)$" term grows as something like $(\log n)^{40c}$ even for points in $\Re^2$, so finding a practical version of this algorithm remains an open problem.

Our new algorithm is similar to our earlier algorithm in [2]. That algorithm relied on the fact that the plane can be recursively partitioned such that some $(1+1/c)$-approximate salesman tour crosses each line of the partition at most $r = O(c \log n)$ times. Such a partition and salesman tour can be computed by a simple dynamic programming. For each line in the partition the algorithm first "guesses" where the tour crosses this line and the order in which those crossings occur. Then the algorithm recurses independently on the two sides of the line. There are only poly(n) distinct regions in the partition, and owing to the planarity constraint, the number of ways in which the tour could cross a line r times (i.e., the number of "guesses" the algorithm must try for the line) is only $2^{O(r)}$. Thus the running time is $\mathrm{poly}(n) \cdot n^{O(c)}$.

The approximation scheme of this paper relies on a similar result about $(1+1/c)$-approximate salesman tours, except we show that the parameter "r" needs to be only $O(1/c)$. [2] This result—shown using a "cut-and-patch" on the optimum tour similar to that in [2]— relies on a new "charging" argument that looks at all levels in the partition simultaneously. The partition used in this paper is a randomized variant of the familiar *quadtree*, and the use of randomness seems essential (our algorithm can be trivially derandomized, but this seems to mutiply the running time in $\Re^d$ by $O(n^d)$). We suspect that our charging argument will find other uses in computational geometry.

As shown in [2], our "cut-and patch" approach generalizes to many other NP-hard problems such as Minimum Steiner Tree, k-MST and k-TSP. Our new algorithm also generalizes to those problems. (We don't know yet whether it generalizes to degree restricted spanning tree, though in a preliminary draft we mistakenly claimed that it does.) The algorithm also generalizes to *Euclidean Min Cost Perfect Matching*, a problem that can be solved exactly in $n^{2.5}$ time.

1.1 Definitions and Results

The input to our algorithm is a set of n points in $\Re^d$, given by their coordinates. We use the standard Real RAM model of computation, which assumes a unit cost for arithmetic on real numbers. (Strictly speaking, we will need computations on real numbers only to "preprocess" the input in $n \cdot \mathrm{poly}(\log n)$ time, after which all internode distances will be truncated to $2\log n$ bits. This affects the cost of the final tour by a factor $\leq 1 + n/n^2 = 1 + 1/n$.) We will consider the following geometric problems.

Minimum Steiner Tree: Given n nodes in $\Re^d$, find the minimum cost tree connecting them.

k-TSP: Given n nodes in $\Re^d$ and an integer $k > 1$, find the smallest tour that visits at least k nodes.

k-MST: Given n nodes in $\Re^d$ and an integer $k \geq 2$, find k nodes with the shortest MST.

Euclidean Mincost Perfect Matching (EMCPM) : Given n points in $\Re^2$ (or $\Re^d$ in general), find the minimum cost set of nonadjacent edges that cover all vertices.

A PTAS was described for the first three problems in [2]; prior to that they had constant factor approximation algorithms (see the survey by Bern and Eppstein [5]).

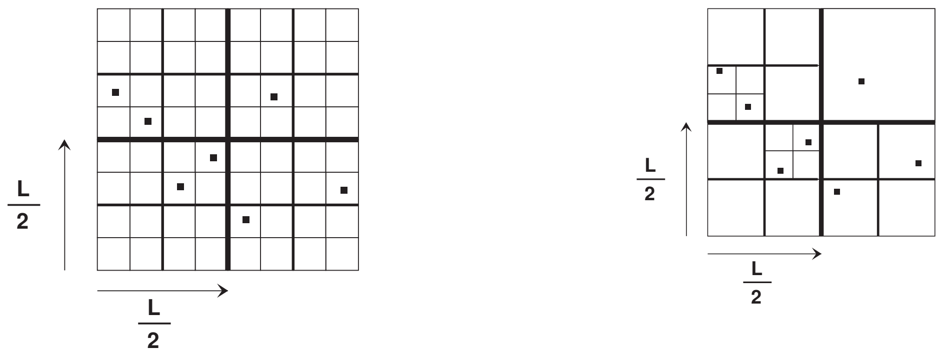
Theorem 1 (Main) *There is a randomized PTAS for all the above problems that computes a $(1+1/c)$-approximation on planar instances in $O(n(\log n)^{O(c)})$ time for TSP, Steiner Tree and EMCPM, and in $O(nk(\log n)^{O(c)})$ time for k-TSP and k-MST. On instances in $\Re^d$, all running times get multiplied by a factor $(\log n)^{(O(\sqrt{d}c))^{d-1}}$. The running time is unchanged (except for the constant in the exponent) if distance is measured using any other Minkowski norm.*

The algorithm can also be implemented in RNC^2. It can be made deterministic by increasing its running time by a factor $O(n^d)$.

---

of cost $(2+1/c)OPT$ in case of double-MST, then the running time reduces to $O(\mathrm{poly}(c) n \log^3 n)$ and if one is willing to settle for solutions of cost $(1.5+1/c)OPT$ in case of Christofides then its running time reduces to $O(\mathrm{poly}(c)n^{1.5} \log^3 n)$ (Vaidya [27]).

[2] I originally conjectured that the "r" parameter needs to be $\Omega(\log n)$. I thank Karen Wang, whose failure to construct any instance demonstrating this [28] motivated me to abandon that conjecture.



(a) (b)

Figure 1: (a) The dissection. (b) The corresponding quad tree

Prior work on polynomial-time problems. Geometric versions of problems such as Euclidean MST and Perfect Matching have been studied before, with an aim to exploit geometric structure and derive algorithms that are faster than the corresponding algorithms for general graphs. For any fixed dimension d, Agarwal et al. [1] have designed a $\tilde{O}(n^{4/3})$-algorithm for Euclidean MST and Vaidya [26] has designed a $\tilde{O}(n^{2.5})$-algorithm for matching. Vaidya [27] has also designed approximation schemes that compute $(1 + 1/c)$-approximations for MST and Euclidean Matching in $\tilde{O}(cn2^{O(d)})$ and $\tilde{O}(n^{1.5}(2c)^{O(d)})$ time respectively. Our result significantly improve upon Vaidya's matching algorithm when d is fixed: our approximation scheme runs in $O(n(\log n)^{(O(\sqrt{d}c))^{d-1}})$ time. We suspect that our ideas may apply to many other geometric problems (certainly they apply to MST).

2 The TSP Algorithm in $\Re^2$

The algorithm performs a (recursive) geometric partitioning of the instance and then performs dynamic programming over the set of instances thus produced. Thus it is reminiscent of the algorithm we described in [2]. However, the geometric partitioning in this paper is very simple: it is a variant of the familiar quadtree.

Our algorithm starts with an $O(n \log n)$-time "perturbation" of the instance that makes it *well-rounded.* This means:(i) all nodes have integral coordinates (ii) each (nonzero) internode distance is at least 8 units, and (iii) the maximum internode distance is $O(n)$. The instance is made well-rounded as follows. Let the *bounding box* of the instance be the smallest axis-aligned square that contains it, and L be the length of its side. Let OPT be the optimum tour cost, so OPT $\geq L$. We place a grid of granularity $L/8nc$ in the plane and move each node to its nearest gridpoint (thus making all coordinates integral). Note that the cost of the optimum tour changed by at most $2n \cdot L/8nc < OPT/4c$. Then we scale distances by $L/64nc$, so that the minimum (nonzero) internode distance is at least 8, and the size of the bounding box is $O(nc)$, which is $O(n)$ since c is a constant. Thus the instance has become well-rounded. Note that we now need to compute a $(1 + 3/4c)$-approximation in this new instance instead of a $(1+1/c)$-approximation, but that doesn't matter since c represents an arbitrary positive constant.

A *dissection* of the bounding box is a recursive partitioning into smaller squares. We view it as a 4-ary tree whose root is the bounding box. Each square in the tree is partitioned into four equal squares, which are its children. We stop partitioning a square if it has size ≤ 1 (and therefore at most one node). A *quadtree* is defined similarly, except we stop the recursive partitioning as soon as the square has at most one node. The quadtree may in general have fewer squares than the dissection. See Figure 1.

Since the bounding box has size $O(n)$, the depth of both partitions is $O(\log n)$. The dissection has $O(n^2)$ squares and the quad-tree has

$$O(\text{number of leaves with a node } \times \text{ depth})$$

squares, which is $O(n \log n)$.

Let L be the length of the bounding box. If $a, b \leq L$, then the (a, b)*-shift* of the dissection is defined by shifting the x- and y- coordinates of all lines by a and b respectively, and then reducing modulo L. In other words, the middle vertical line of the dissection is moved from the x-coordinate $L/2$ to the x-coordinate $a + \frac{L}{2} \bmod L$, and the middle horizontal line from the y-coordinate $L/2$ to the y-coordinate $b + \frac{L}{2} \bmod L$. The rest of the dissection is then "wrapped-around,"

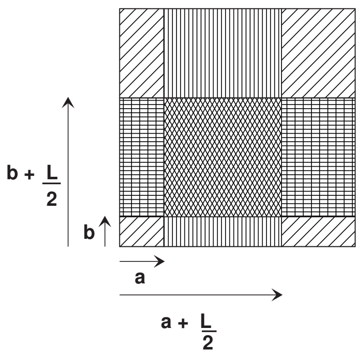


Figure 2: Dissection with shift (a, b). Only the four children of the root are shown; each is identified by a distinctive shading. Note that three of the children are "wrapped-around" squares.

so that the left edge of the dissection comes to rest at the x-coordinate a, and the lower edge of the dissection comes to rest at the y-coordinate b (see Figure 2). Note that for ease of notation we treat a "wrapped-around" square in the shifted dissection as a single region. The reader can also think of it as a disjoint union of 2 (or 4) rectangles.

The *quadtree with shift* (a, b) is obtained from the corresponding shifted dissection by cutting off the partitioning at squares that contain only 1 node. In general the shifted quadtree has a very different structure than the original quadtree. However, since both have depth $O(\log n)$, they can be constructed very efficiently using a sorting-based algorithm in $O(n \log^2 n)$ time (even faster algorithms exist [6]).

Our algorithm picks a random shift and constructs the corresponding shifted quadtree. Our Structure Theorem below guarantees that with high probability (over the choice of the random shifts), there exists a $(1 + 1/c)$-approximate tour that crosses each square in the shifted quadtree $O(c)$ times. In fact the Structure Theorem says something stronger: these crossings happen at one of a small set of prespecified points (called *portals*). The previous sentence may mystify the reader: how can the crossing points be prespecified?

The reason is that we will allow the salesman to deviate from the straight-line path while travelling between nodes. Thus it can make digressions, and pass through any desired prespecified point. In other words, an edge in such a tour might be "bent" in several places. Such a salesman tour with "bent" edges will be called a *salesman path*.

In the rest of this section we will let OPT denote the cost of the optimal salesman tour (with no bent edges). Our algorithm will return a salesman path of cost at most $(1 + \epsilon)OPT$. We can straighten the bent edges at the end to get a salesman tour, without raising the cost (this follows from triangle inequality).

Definition 1 Let m, r be positive integers. An m-*regular set of portals* for a shifted dissection is a set of points on the edges of the squares in it. Each square has a portal at each of its 4 corners and m other equally-spaced portals on each edge.

A salesman path is (m, r)-*light* with respect to the shifted dissection if it crosses each edge of each square in the dissection at most r times and always through one of the portals. (It could go through a portal multiple times.)

Notes: (i) By forcing the salesman path to cross region boundaries only at a portal, we force it to have "bent" edges in general. The "bent" edges will also play an important role in the proof of Theorem 2. (ii) We will assume w.l.o.g. that no node lies on any boundary of the dissection. We can ensure this (among other ways) by scaling distances by a factor 2 and then ensuring that nodes have odd coordinates and the lines of the dissection have even coordinates.

Theorem 2 (Structure Theorem) *Let the minimum nonzero internode distance in a TSP instance be 8 and L be the size of the bounding box. Let shifts $0 \le a, b \le L$ be picked randomly. Then with probability at least $1/2$, the dissection with shift (a, b) has an associated (m, r)-light salesman path of cost at most $(1 + 1/c)$-OPT, where $m = O(c \log L)$ and $r = O(c)$.*

(Currently the best constant we know how to achieve in the $O(c)$ is around 10, but that can probably be improved.) Note that even though we allow a salesman path to have an infinite number of "bends", the optimum (m, r)-light salesman path has only a finite number. This is because the number of portals and nodes visited by the salesman path is finite, and optimality implies that it must follow a straight-line path between those visits. Our algorithm will use this fact.

Assuming the truth of the Structure Theorem, we outline an $O(T \cdot (m)^{O(c)})$-time (randomized) algorithm to find a $(1 + 1/c)$-approximate salesman path where $T = O(n \log n)$ is the number of squares in the shifted quad-tree. The algorithm first picks a shift (a, b) randomly and computes a quadtree with these shifts. Then it uses a (deterministic) dynamic programming to find the optimal (m, r)-light salesman path with respect to this shifted quadtree, where $m = O(c \log n)$ and $r = O(c)$. With probability at least $1/2$ (over the choice of the shift), this salesman path has cost at most $(1 + 1/c)OPT$. (Note that though we have described the algorithm as randomized, it can be derandomized by going through all choices of the shifts a, b. This mutiplies the running time by $O(n^2)$.)

The dynamic programming uses the fact that in an optimal (m, r)-light salesman path, the portions of the path inside each of the four subsquares must also be optimal. Furthermore, these four portions are only weakly interrelated, since the tour crosses the boundary of each subsquare at most $4r$ times. The dynamic programming tries all possible arrangements of these $4r$ crossings, thus ensuring that it will hit upon the optimal one. Then it recurses in the four subsquares.

Specifically, the dynamic programming maintains a lookup table that contains optimal solutions to all subproblems specified by the following inputs: (a) a square in the shifted quadtree (b) a multiset of $\leq r$ portals on each of the four edges of this square (c) a pairing between the $\leq 4r$ portals specified in (b). This pairing represents the order in which the final path will traverse these portals.

For each square in the quadtree, each choice in (b), (c) determines an instance of a *multiple traveling salesmen* problem inside the square, in which a team of at most $2r$ salesmen with designated starting and stopping points on the boundary are required to each visit a subset of "clients" inside the square, such that every client is visited by exactly one salesman. Of course, the algorithm is only interested in (m, r)-light solutions to this multiple salesman problem.

Note that the number of subproblems corresponding to distinct triples in (a), (b), (c) (and hence the size of the lookup table) is at most $O(T \cdot (m+3)^{4r} \cdot (4r)!)$. The table can be built up in a bottom-up fashion as follows. Subproblems at the leaves of the quadtree contain at most 1 node and $O(r)$ portals, so they are solved optimally in $2^{O(r)}$ time. For every other square and for every choice in (b), (c) the algorithm enumerates all possible ways in which the multiple-salesman tour could traverse the edges of the four subsquares of the square. Specifically, it enumerates all possible choices of (a') a multiset of $\leq r$ portals on each of the four sides of the subsquares (b') an order of traversing these $\leq 4r$ portals. The number of choices in (a') and (b') is $((m+3)^r)^4$ and $(4r)^{4r}(4r)!$ respectively. Each choice in (a') and (b') leads to a multiple-salesman problem in each of the four subsquares, whose optimal solutions already exist in the lookup table. Thus the algorithm can determine the optimal choice in (a'), (b').

Thus the running time is $O(T \cdot (m+3)^{8r}(4r)^{4r}(4r!)^2$.

We didn't mention thus far how the subproblems corresponding to "wrapped-around" squares are solved. Note however that dynamic programming over such squares is if anything easier than in "normal" squares (i.e., the number of choices explored by the algorithm is smaller), since the salesman path cannot go between the two (or four) disjoint portions of the wrapped-around square.

Remark: It should be clear now why we insisted that the salesman path enter and leave the regions of the quad tree only at portals. This greatly aids efficiency, since our dynamic programming has to enumerate all possible ways in which the path could enter and leave the region. If we had allowed the salesman tour to use normal edges while entering and exiting, then this enumeration could have required as much as $\binom{n^2/2}{O(r)}$ time (since the number of edges among n nodes is $\binom{n}{2} \approx n^2/2$), instead of $m^{O(r)}$.

2.1 Proof of Structure Theorem

The following simple lemma (which also appeared in [2], but is a folk-theorem) will be important. When we later use this lemma, the "closed path" π of the hypothesis will be a salesman path on some set of nodes. The closed path π' of the conclusion of the lemma will be a new salesman path. (This is where we need the fact that a salesman path is allowed to have "bent" edges, since otherwise it might be impossible to add line segments from S to π.)

Lemma 3 (Patching Lemma) *There is a constant $g > 0$ such that the following is true. Let S be any line segment of length l and π be a closed path that crosses S at least thrice. Then there exist line segments on S whose total length is $g \cdot l$ and whose addition to π changes it into a closed path that crosses S at most twice.*

Proof: Suppose π crosses S a total of t times. Let $M_1, \ldots, M_t$ be the points on which π crosses S. Break π at those points, thus causing it to fall apart into t paths $P_1, P_2, \ldots, P_t$. In what follows, we will need two copies of each M_i, one for each side of S. Let M_i' and M_i" denote these copies.

Let $2k$ be the largest even number less than t. Let J be the multiset of line segments consisting of the following: (i) A minimum cost salesman tour through $M_1, \ldots, M_t$. (ii) A minimum cost perfect matching among $M_1, \ldots, M_{2k}$. Note that the line segments of J lie on S and their total length is at most $3l$. We take two copies J' and J" of J and add them to π. We think of J' as lying on the left of S and J" as lying on the right of S.

Finally if $t = 2k+1$ (i.e., t is odd), then we add an edge between M'_{2k+1} and $M_{2k+1}"$. If $t = 2k+2$, then we add an edge between M'_{2k+1} and $M_{2k+1}"$ and an edge between M'_{2k+2} and $M_{2k+2}"$. (Note that these edges have length 0.)

Together with the paths $P_1, \ldots, P_{2k}$, these added segments and edges define a connected 4-regular graph on

$$\{M'_1, \ldots, M'_t\} \cup \{M_1", \ldots, M_t"\}.$$

An Eulerian traversal of this graph is a closed path that contains $P_1, \ldots, P_t$ and crosses S at most twice. Hence we have proved the theorem for $g = 6$. $\square$

Remarks: (i) A more careful argument shows $g = 3$ suffices. (ii) A similar patching can be done for paths in $\Re^{d+1}$ whose endpoints lie inside a d-dimensional cube of side l. By a well-known upperbound on the length of a tour on k nodes in a cube the patching cost is $O(k^{1-1/d}l)$.

The next lemma will be useful in the algorithm's analysis. Let us grid the bounding box by putting a sequence of vertical and horizontal lines at unit distance from one another. The lemma relates the cost of a tour to the total number of times it crosses the grid lines. If l is one of these lines and π is a salesman tour then let $t(\pi, l)$ denote the number of times that π crosses l.

Lemma 4 *If the minimum internode distance is at least* 4*, and T is the length of π, then*

$$\sum_{l:vertical} t(\pi, l) + \sum_{l:horizontal} t(\pi, l) \quad \leq \quad 2T$$

Proof: The main observation is that the left hand side roughly measures the ℓ_1 length of the tour, which is at most $\sqrt{2}$ times T.

Specifically, we show that an edge of π that has length s contributes at most $2s$ to the left hand side. Suppose u and v are the lengths of the horizontal and vertical projections of the edge; thus $u^2 + v^2 = s^2$. Then it contributes at most $(u+1) + (v+1)$ to the left hand side, and

$$u + v + 2 \quad \leq \quad \sqrt{2(u^2+v^2)} + 2 \quad \leq \quad \sqrt{2s^2} + 2.$$

Finally, since $s \geq 4$, we have $\sqrt{2s^2} + 2 \leq 2s$. $\square$

Now we are ready to prove the Structure Theorem.

Proof:(*Structure Theorem*) Let $r \geq 12gc$, where g is the constant appearing in the Patching Lemma. We prove the Structure Theorem by starting with an optimum tour π and modifying it (using a deterministic procedure and over many steps) into a salesman path that is (m, r)-light with respect to the randomly-shifted dissection. This may increase the tour cost slightly, which we upperbound (in the expectation) as follows. For accounting purposes, we place a grid of unit granularity in the bounding box. We "charge" any cost increase to some (horizontal or vertical) line of the grid. We will show that for each line l of the grid,

$$E_{a,b}[\text{charge to line } l \text{ when the shift is } (a,b)] \leq \frac{3g\, t(\pi, l)}{r}, \tag{1}$$

where g is the constant appearing in the Patching Lemma. By linearity of expectations it then follows that the expected increase in the cost of the tour is

$$\leq \sum_{l:\text{vertical}} \frac{3g\, t(\pi, l)}{r} + \sum_{l:\text{horizontal}} \frac{3g\, t(\pi, l)}{r},$$

which is $\leq \frac{6g\, OPT}{r}$ by Lemma 4.

Thus when $r \geq 12gc$, the expected increase in the tour cost is at most $OPT/2c$. Markov's inequality implies that with probability at least 1/2 this increase is no more than OPT/c. We conclude that with probability at least 1/2 the cost of the best (m, r)-light salesman path for the shifted dissection is at most $(1 + 1/c)OPT$.

Thus to prove the theorem it suffices to describe how we modify the optimum tour π and charge resulting cost increases. Assume w.l.o.g. that the size of the bounding box L is a power of 2. Thus all lines used in the dissection are grid lines. Recall how the dissection with shift (a, b) is obtained from the dissection. The middle vertical line of the dissection is moved from the x-coordinate $L/2$ to the x-coordinate $a + \frac{L}{2} \bmod L$, and the middle horizontal line from the y-coordinate $L/2$ to the y-coordinate $b + \frac{L}{2} \bmod L$. Then the rest of the dissection is "wrapped-around," so that the left edge of the dissection comes to rest at the x-coordinate a, and the lower edge of the dissection comes to rest at the y-coordinate b (see Figure 2). Note that since a, b are integers, the lines of the shifted dissection still lie on grid lines.

Recall that squares in a dissection form a hierarchy, and have a natural notion of "level" (the bounding box is at level 0, its four children are the squares at level 1, and so on). We say that a grid line l has *level* i in the shifted dissection if it contains the edge of some level i square. Note that the edge of a level i square gets subdivided to yield the edges of two level $i+1$ squares, so a line that is at level i is also at level j for all $j > i$. For each $i \geq 1$ there are 2^i horizontal lines and 2^i vertical lines at level i. The vertical lines have x-coordinates $a + p \cdot \frac{L}{2^i} \bmod L$, where $p = 0, \ldots, 2^i - 1$ and

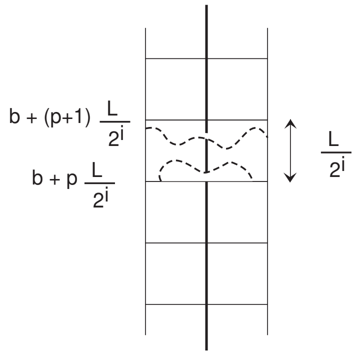


Figure 3: If a vertical line l is at level i, then for $p = 0, 1, 2, \ldots, 2^i - 1$, the segment of the line between the y-coordinates $b+p\cdot\frac{L}{2^i} \bmod L$ and $b+(p+1)\cdot\frac{L}{2^i} \bmod L$ is the edge of a level i square. An (m,r)-light tour must cross this segment at most r times, and only through one of the m portals on it.

the horizontal lines have y-coordinates $b+p\cdot\frac{L}{2^i} \bmod L$, where $p = 0, \ldots, 2^i - 1$. The *maximal level* of a line is the highest level it is at.

Since a is chosen randomly, we have for each vertical line l in the grid, and each $i \leq \log L$,

$$\Pr_a[l \text{ is at level } i] = \frac{2^i}{L}. \tag{2}$$

Of course, a similar statement is true for horizontal lines.

What does it mean for the tour to be (m,r)-light? For simplicity we describe this only for vertical lines; a similar statement holds for horizontal lines. For vertical line l, (m,r)-lightness means, first, that if i is the maximal level of this line, then for $p = 0, 1, 2, \ldots, 2^i - 1$, the segment of the line lying between the y-coordinates $b + p\cdot\frac{L}{2^i} \bmod L$ and $b + (p+1)\cdot\frac{L}{2^i} \bmod L$ is crossed by the salesman path at most r times. Second, all these crossings happen at portals. (See Figure 3.)

How can we modify the tour to satisfy the first condition? An obvious idea suggests itself. Go through all lines l in the grid, and if its maximal level is i, then go through each of its 2^i segments of length $L/2^i$ and apply the Patching Lemma whenever one of them is crossed by the tour more than k times. This certainly works, but unfortunately the resulting cost increase may be too high (the reader may wish to check this). A better idea is to call the procedure MODIFY(l,i,b), which does the patching "bottom up" for all levels $j \geq i$ and therefore is more miserly with cost increases. (We describe the procedure only for vertical lines; the description for horizontal lines is identical.)

MODIFY(l,i,b)

(l is a vertical grid line, b is the vertical shift of the dissection, and i is the maximal level of line l)

For $j = \log L$ down to i do:

For $p = 0, 1, \ldots, 2^j - 1$, if the segment of l between the y-coordinates $(b + p\cdot\frac{L}{2^j} \bmod L)$ and $(b + (p+1)\cdot\frac{L}{2^j} \bmod L)$ is crossed by the current tour more than r times, then use the patching lemma to reduce the number of crossings to 4.

Remarks on MODIFY*:* (i) The careful reader might note that we assume that the number of crossings after patching is 4, whereas the statement of the Patching Lemma seems to ensure this number is 2. The reason for the discrepancy is that the segment could be "wrapped-around", and the patching has to be done separately for its two disjoint parts. (ii) The tour is updated dynamically, so the modifications made for the $j = k$ iteration of the loop are influenced by the modifications for $j = k+1, \ldots, \log L$. (iii) The patching on a vertical line adds to the cost of the tour and could increase the number of times the tour crosses a horizontal line. We ignore this effect, for reasons explained at the end of the proof. (iv) The structure of the loop (i.e., the fact that j decreases from $\log L$ to i) is important and will be used in the argument that follows.

If $j \geq i$, let $c_{l,j}(b)$ denote the number of segments to which we apply the patching lemma in the jth iteration of the "for" loop in MODIFY(l,i,b). (Note that $c_{l,j}(b)$ is independent of i since the loop computation for j doesn't depend on i.) Since each application of the Patching Lemma replaces at least $r+1$ crossings by at most 4 and tour π crossed line l only $t(\pi,l)$ times, we have for each vertical shift b

$$\sum_{j\geq 1} c_{l,j}(b) \leq \frac{t(\pi,l)}{r-3}. \tag{3}$$

Furthermore, the Patching Lemma implies that

$$\text{Cost increase due to MODIFY}(l,i,b) \leq \sum_{j\geq i} c_{l,j}(b)\cdot g\cdot\frac{L}{2^j}, \tag{4}$$

where g is the constant appearing in the Patching Lemma. We charge this cost to l. Of course, whether or not this charge occurs depends on whether or not i is the maximal level of line l, which by (2) happens with probability at most $2^i/L$ (over the choice of the horizontal shift a). Thus for every vertical line l and every $0 \leq b \leq L-1$,

$$E_a[\text{charge to } l]$$

$$\begin{aligned}
&\le \sum_{i\ge 1} \frac{2^i}{L} \cdot \text{cost increase due to } \textsc{modify}(l,i,b) \\
&\le \sum_{i\ge 1} \frac{2^i}{L} \cdot \sum_{j\ge i} c_{l,j}(b) \cdot g \cdot \frac{L}{2^j} \\
&\le g \cdot \sum_{j\ge 1} \frac{c_{l,j}(b)}{2^j} \cdot \sum_{i\le j} 2^i \\
&\le g \cdot \sum_{j\ge 1} 2 \cdot c_{l,j}(b) \\
&\le \frac{2gt(\pi,l)}{r-3}
\end{aligned}$$

The next modification to the tour consists in moving each crossing to the nearest portal. If a line l has maximal level i, then each of the $t(\pi,l)$ crossings might have to be displaced by $\frac{L}{2^i m}$ to reach the nearest portal. Thus the expected cost of moving every crossing in l to its nearest portal is at most

$$\sum_{i=1}^{\log L} \frac{2^i}{L} \cdot t(\pi,l) \cdot \frac{L}{2^i m} = \frac{t(\pi,l)\log L}{m},$$

which is at most $t(\pi,l)/2r$ when $m \ge 2r\log L$.

Thus the expected cost (over the random choice of a) of making the final salesman path (m,r)-light at line l is

$$\frac{2gt(\pi,l)}{r-3} + \frac{t(\pi,l)}{2r} \le \frac{3gt(\pi,l)}{r}.$$

(Where the last calculation assumes $r > 15$.) This is what we set out to prove.

To finish our proof it only remains to explain our remark (iii) on the MODIFY procedure. This concerned the following situation. Whenever we apply MODIFY on a vertical line l, we use the Patching Lemma and augment the tour with some segments lying on l. These segments could cause the tour to cross some horizontal line l' much more than the $t(\pi,l')$ times it was crossing earlier. However, our analysis assumed that the number of crossings remains constant at $t(\pi,l')$ throughout the modification, and this requires an explanation. The explanation is simple: we can assume w.l.o.g. that the increase in the number of crossings at l' due to the patching on l is at most 2. The reason is that if the increase were more than 2, we could just use the Patching Lemma to reduce it to 2. Furthermore, since the Patching Lemma is being invoked for segments lying on l, these have zero horizontal separation (that is, they lie on top of each other) and therefore the tour cost does not increase! Arguing similarly about all intersecting pairs of grid lines, we can ensure that at the end of all our modification, each side of each square in the shifted dissection is crossed by the modified tour upto $r+4$ times (instead of the "r" we wanted to argue). So we just replace the "r" by "$r+4$."

A similar accounting trick explains why we assumed that the $t(\pi,l)$'s are not affected when we move each edge-crossing to its nearest portal. □

2.2 Algorithms for $\Re^d$ and non-Euclidean norms

The "quad" trees become 2^d-ary trees in $\Re^d$. In this tree the boundaries of regions are cubes of dimension $d-1$. An m-regular set of portals on a dimension $d-1$ cube is a lattice of equally spaced points in the cube. Thus if the cube has length W then the spacing between the portals is $W/m^{1/d-1}$.

The structure theorem for $\Re^d$ says that when the shifts of the dissection are picked randomly, then with probability at least $1/2$ the shifted dissection has an associated $(1+1/c)$-approximate salesman path that is (m,r)-light, where $m = O((\sqrt{d}c\log n)^{d-1})$, $r = O((\sqrt{d}c)^{d-1})$. The theorem's proof is analogous to the $\Re^2$ case but uses the $\Re^d$ version of the Patching Lemma. See [3] for details.

Other geometric norms. The Patching Lemma holds for any Minkowski norm, so our algorithms work when distance is measured using any of those norms.

3 Approximation schemes for other problems

Our algorithms for other geometric problems are similar to the TSP algorithm, namely, we construct a quadtree with random shifts, and compute the optimum (m,r)-light structure (i.e., Steiner tree, salesman path etc). The proof of this algorithm's correctness uses a charging argument and the fact that the Patching Lemma and Lemma 4 (suitably modified) holds for all of our problems (see [3] for details). Now we mention the problem-specific features of the algorithm.

Steiner Tree: In the initial preprocessing we move all nodes to a grid of small enough granularity (namely, $O(L/nc)$, where L is the size of the bounding box), and from then on also restrict the Steiner nodes to lie on the grid. Since the optimum Steiner tree has $O(n)$ edges, the cost of the optimum tree changes by at most a factor $O(nL/nc) = O(OPT/c)$. The rest of the algorithm is similar to the TSP case. Note however that *portals* in this case are not an artificial invention, since they can just be considered as Steiner nodes.

One difference from the TSP case is that even though a square in the shifted quadtree may not contain any input nodes, it may contain Steiner nodes, which have to be "guessed" by the algorithm. But since the desired Steiner tree is (m, r) light, such a square is entered and left only $4r$ times. We treat it as an instance of the Steiner Tree problem of size at most $4r$, which can be solved in constant time (since r is constant).

k-TSP and k-MST. The algorithms are similar to that for TSP and Steiner tree respectively. We mention the two major differences. First, the initial preprocessing step needs care (see below). Second, the running time of the dynamic programming is higher than in the TSP case by $O(k)$. The reason is that for each square it has to "guess" how many of the input nodes from each square are included in the final tour/spanning tree, and the number of choices for this is k. (For a detailed analsis see [3].)

Min Cost Perfect Matching The perturbation at the start of the algorithm requires some care, but otherwise the algorithm is similar to the TSP case.

How to make the instance well-rounded. The procedure to make TSP instances well-rounded does not generalize to k-TSP, k-MST and EMCPM. It relies on the fact that $OPT \geq L$ for the TSP, where L is the size of the bounding box. This may not hold for our problems (since $OPT \ll L$ is possible).

Let us recall why $OPT \geq L$ was needed in the TSP case. If d_{min} denotes the minimum internode distance, then the depth of the quadtree is $O(\log \frac{L}{d_{min}})$. *A priori* this depth could be large but we laid a grid of granularity $L/8cn$ and moved each node to the nearest gridpoint, thus ensuring $d_{min} \geq L/8cn$. Hence the quadtree has depth $O(\log n)$, which is crucial for efficiency. Furthermore, the coalescing affects the cost of the optimal salesman tour by at most $2n \cdot L/8cn$, which is at most $OPT/4c$ since $OPT \geq L$.

For the three problems at hand our rounding procedure will ensure that the size of the bounding box is $\leq \text{poly}(n) \cdot d_{min}$. First it computes in nearly-linear time a crude approximation to OPT that is correct within a factor n. (Algorithms to compute such approximations were known before our work; see [27] for matching and [9] for k-MST and k-TSP. The latter algorithm runs in $O(nk \log n)$ time.) Let $A \leq OPT \cdot n$ be this approximation. The procedure lays a grid of granularity $A/8cn^2$ in the plane and moves every node to its nearest gridpoint. Now $d_{min} \geq A/8cn^2$, and triangle inequality implies that cost of the optimum solution changed by at most $2n \cdot A/8cn^2 \leq OPT/4c$.

If $A \geq L/n^2$, say, then we're done because the depth of the quadtree is $O(\log(\frac{L}{d_{min}}) = O(\log n)$ in that case. Now suppose $A < L/n^2$. We pick random shifts as before and construct a shifted quadtree (this can be done in $O(n \log n)$ time [6]). Then we treat all squares of size more than $A \log n$ in the shifted quadtree as independent instances of the problem. (Note that each of these squares has a quadtree of depth $O(\log 8cn^2) = O(\log n)$, where n is the original number of nodes.) To justify this, we recall from Lemma 4 that at most $2A$ lines of the unit grid are crossed by any edges of the optimum solution. Hence the probability (over the choice of random shifts) that any of these $2A$ grid lines is on the boundary of a square of size $> A \log n$ in the shifted quadtree is at most $2/\log n$. Hence with probability $> 1 - 2/\log n$, no square of size more than $A \log n$ will be crossed by any edge of the optimum solution, in which case our decision to treat those squares as independent instances is justified!

4 Conclusions

Can the running time of our algorithm be improved? Since Euclidean TSP is strongly NP-hard (this follows from the reductions in [11, 22], for example), any algorithm that computes $(1 + c)$-approximations must have a running time with some exponential dependence on c (unless of course, NP-complete problems can be solved in subexponential time). Likewise, Trevisan [25] has presented evidence that every approximation scheme for $\Re^d$ must have a running time with a doubly exponential dependence on d. He shows for some $c > 1$ that finding a $(1+1/c)$-approximate solution in $\Re^{O(\log n)}$ is NP-hard.

Thus the running time of our algorithm is "in the right ball park," but could be improved in little ways (and most importantly, in the constants). For example, a running time of $O(n \log^2 n + 2^{O(c)})$ is conceivable in $\Re^2$.

Finally, it would be nice to reduce the runtime of the PTAS for planar graph TSP [17] to near-linear from $n^{O(c)}$.

Acknowledgements

I thank Tom Leighton for useful discussions at the start of this project, and Mihalis Yannakakis for helpful comments. I also thank Pankaj Agarwal, David Eppstein, and Jeff Erickson for their pointers to existing work in computational geometry.

References

[1] P. Agarwal, H. Edelsbrunner, O. Schwarzkopf and E. Welzl. Euclidean MST and bichromatic closest pairs. *Discrete and Comp. Geo.* **6**:407–422, 1991.

[2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Proc. 37th IEEE FOCS*[3], pages 2-12, 1996. Preliminary manuscript (with "quasipolynomial" in the title), January 20, 1996.

[3] Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal submission*, 1997. Available from `http://www.cs.princeton.edu/ arora`.

[4] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proc. 33rd IEEE FOCS*, pages 13–22, 1992.

[5] M. Bern and D. Eppstein. Approximation Algorithms for Geometric Problems. In *Approximation Algorithms for NP-hard Problems,* editor D. Hochbaum, PWS Publishing, 1996.

[6] M. Bern, D. Eppstein, and S.-H.Teng. Parallel construction of quadtree and quality triangulations. In *Proc. 3rd WADS,* pp 188-199. Springer Verlar LNCS 709, 1993.

[7] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Report 388, Grad School of Industrial Administration, CMU, 1976.

[8] W.Fernandez de la Vega and G.S.Lueker. Bin packing can be solved within $1+\epsilon$ in linear time. *Combinatorica*:1(4), 349–355, 1981.

[9] D. Eppstein and J. Erickson. Iterated Nearest Neighbors and Finding Minimal Polytopes. *Discrete Comp. Geo.*, **11:** 321-350, 1994.

[10] H. Gabow. Data Structures for weighted matching and nearest common ancestors with linking. *Proc. 1st ACM SODA*, pages 434–443, 1990.

[11] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proc. ACM STOC*, pp 10-22, 1976.

[12] O. H. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subsets problems. *JACM*, 22(4):463–468, 1975.

[13] D. S. Johnson and Lyle A. McGeoch. The Traveling Salesman Problem: A Case Study in Local Optimization. Chapter in *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds.), John Wiley and Sons, NY. (To appear.)

[14] N. Karmarkar and R.M. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. In *Proc.* 23^{rd} *IEEE FOCS*, pages 312–320, 1982.

[15] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, Advances in Computing Research, pages 85–103. Plenum Press, 1972.

[16] R. M. Karp. Probabilistic analysis of partitioning algorithms for the TSP in the plane. *Math. Oper. Res.* **2** (1977), 209-224.

[17] E. Koutsoupias, M. Grigni, and C. H. Papadimitriou. An Approximation Scheme for Planar Graph TSP. In *Proc. IEEE FOCS,* pp 640–645, 1995.

[18] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. The traveling salesman problem. John Wiley, 1985

[19] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research 21*:498–516, 1973.

[20] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple new method for the geometric k-MST problem. In *Proc. ACM-SIAM SODA*, pp. 402-208, 1996.

[21] J. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: Part II- A simple PTAS for geometric k-MST, TSP, and related problems. Preliminary manuscript, April 30, 1996.

[22] C. H. Papadimitriou. Euclidean TSP is NP-complete. *TCS*, **4**, pp 237-244, 1977.

[23] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *JCSS* **43**, pp. 425–440, 1991.

[24] G. Reinelt. TSPLIB–A travelling salesman problem library. *ORSA J. Computing*, **3**, pp 376–384, (1991).

[25] L. Trevisan. When Hamming meets Euclid: the approximability of geometric TSP and MST. In *Proc. 29th ACM STOC*, pp 21–39, 1997.

[26] P. Vaidya *Geometry helps in matching. SIAM J. Comp.*, **18**:1201-1225, 1988.

[27] P. Vaidya Approximate minimum weight matching on k-dimensional spaces. *Algorithmica*, **4**: 569–583, 1989.

[28] K. Wang. Is the m parameter in Arora's TSP algorithm the best possible? Junior project, Princeton University, Fall 1996.

[3] FOCS= Foundations of Computer Science, STOC =Symposium on Theory of Computing, SODA = Symposium on Discrete Algorithms