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 [3] (and the scheme of Mitchell [40]). 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 [3] ran in quasipolynomial 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, Minimum degree-restricted spanning tree, 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 also have simple parallel implementations (say, in NC^2).

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

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 [35]).

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 [37] run —after careful tuning— in nearly *linear* time and often produces tours whose cost is within a few percent of the optimum (Johnson and McGeoch [24]).

Potentially, this paradox could have a simple explanation since many "real-life" instances (see for example the TSPLIB library [45], a test-bed for much experimental work on the TSP) are *geometric*, and many 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[44]). 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 [11], 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 [3] and soon after, Mitchell [40]). 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 works 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 [22]) and Bin-Packing (de la Vega and Lueker [FL81]; see also Karmarkar and Karp [27])

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 respectively. More sobering is the observed (but unproven) performance and near-linear runtimes of heuristics such as Lin-Kernighan.

In this paper we describe a randomized approximation scheme that provably runs in nearly linear time. 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 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 $O(n\text{poly}(\log n))$, 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 [3]. 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 tour can be found by 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

[1]These running times for the classical heuristics, though often quoted, are overstated. If one is willing to settle for solutions of cost $(2 + 1/c)OPT$ in case of double-MST, then the running time reduces to $O(\text{poly}(c)n\log^3 n)$ 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 $O(\text{poly}(c)n^{1.5}\log^3 n)$ (Vaidya [50]).

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 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 [3]— relies on a new charging argument that looks at all $\log n$ 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 [3], our "cut-and patch" approach generalizes to many other NP-hard problems such as Minimum Steiner Tree, k-MST and k-TSP. Thus it is straightforward to generalize our new algorithm to those problems. The algorithm also generalizes to *Euclidean Min Cost Perfect Matching*, a problem that can be solved exactly in $n^{2.5}$ time.

1.1 Preliminaries 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 $n \cdot \text{poly}(\log n)$ time, after which all coordinates will become integers.)

Now we define the problems to which our algorithm applies. All these problems had constant factor approximation algorithms prior to [3], see the survey by Bern and Eppstein [7].

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

Degree-restricted-MST. Given n nodes in $\Re^d$ and an integer $k \geq 2$, find the minimum cost spanning tree in which every node has degree at most k.

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 Min Cost 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.

Theorem 1 *There is a PTAS for all the above problems. The running time for $(1+1/c)$-approximation on planar instances is $n(\log n)^{O(c^2)}$ for degree-restricted MST, $n(\log n)^{O(c)}$ for Steiner Tree and Min-Cost-Euclidean Matching, and $nk(\log n)^{O(c)}$ for k-TSP and k-MST. All running times on instances in $\Re^d$ get multiplied by a factor $\log n)^{(O(\sqrt{d}c))^{d-1}}$.*

Prior work on polynomial-time problems. Problems such as Euclidean MST and Perfect Matching have been studied before. Exploiting geometric structure for these problems leads to algorithms that are faster than the corresponding algorithms for general graphs [31, 16]. Agarwal et al. [1] have designed a $\tilde{O}(n^{4/3})$-algorithm for Euclidean MST in when the dimension d is fixed,

[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 [51] motivated me to abandon that conjecture.

and Vaidya [49] has designed a $\tilde{O}(n^{2.5})$-algorithm for matching. Vaidya [50] 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 [3]. 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 $c > 1$ represents an arbitrary 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.

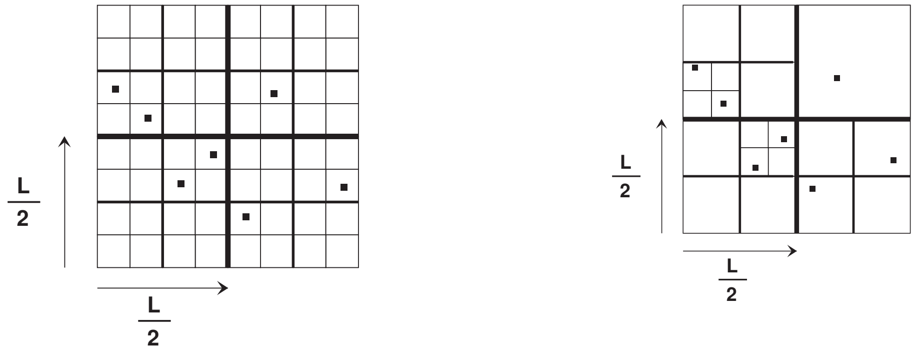
Since the bounding box has size $O(n)$, the depth of both partitions is $O(\log n)$. The dissection has $O(n^2)$ squares and the quad-tree has

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

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

Let L be the length of the bounding box. If a, b are integers, 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," 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. It is easy to see that in general the shifted



(a) (b)

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

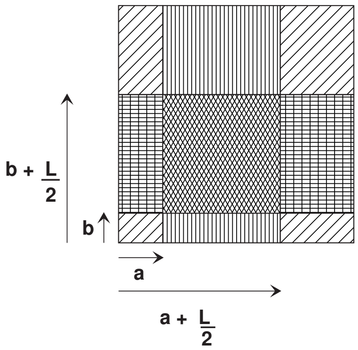


Figure 2: Dissection with shift (a, b). Only the four children of the root are shown; each is identified by a distinctive shading. Note that three of the children are "wrapped-around" squares.

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 may exist).

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. The algorithm then uses dynamic programming to find such a tour.

To simplify exposition we will allow the algorithm to introduce certain "Steiner" points, called *portals*, on the edges of the square of the dissection. Whenever the salesman tour crosses the boundary of any region in the dissection, it has to go through a portal. In terms of the original set of nodes, such a tour can be viewed as having "bent" edges. We call such tours *salesman paths.* Note that straightening the bent edges at the end of the algorithm changes a salesman path into a salesman tour and can only decrease the cost. In the rest of this section we will let OPT denote the cost of the optimal salesman tour (with no bent edges).

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.)

Note: We will assume w.l.o.g. that nodes do not lie 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* $0 \leq a, b \leq L$ *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.

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 explorted by the algorithm is smaller), since the salesman path cannot go between the two (or four) 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 [3], but is a folk-theorem) will be important.

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: *See the appendix.* $\square$

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

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

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 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. (m,r)-lightness means, firstly, that for vertical line l, if i is the maximal level of this line, then for $p = 0, 1, 2, \ldots, 2^j - 1$, the segment of this 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. Secondly, 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

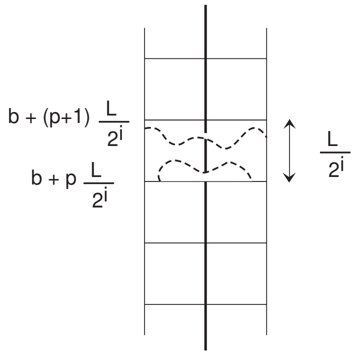


Figure 3: If a vertical line l is at level i, then for $p = 0, 1, 2, \ldots, 2^i - 1$, the segment of the line between the y-coordinates $b + p \cdot \frac{L}{2^i} \bmod L$ and $b + (p+1) \cdot \frac{L}{2^i} \bmod L$ is the edge of a level i square. An (m, r)-light tour must cross this segment at most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

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 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

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

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 $0 \leq b \leq L-1$,

$$\begin{aligned}
E_a[\text{charge to } l] &= \sum_{i\geq 1} \frac{2^i}{L} \cdot \text{cost increase due to MODIFY}(l,i,b) \\
&\leq \sum_{i\geq 1} \frac{2^i}{L} \cdot \sum_{j\geq i} c_{l,j}(b) \cdot g \cdot \frac{L}{2^j} \\
&= g \cdot \sum_{j\geq 1} \frac{c_{l,j}(b)}{2^j} \cdot \sum_{i\leq j} 2^i \\
&\leq g \cdot \sum_{j\geq 1} 2 \cdot c_{l,j}(b) \\
&\leq \frac{2gt(\pi,l)}{r-3}
\end{aligned}$$

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 $m \geq 2r\log L$.

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

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

(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 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.

□

3 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 Appendix).

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

4 Other NP-hard problems

The design of efficient PTAS's for other geometric problems uses similar ideas to the ones used for the TSP. Specifically, the Patching Lemma holds for all of them (see [3] for some of the details). The preprocessing step (which makes the instance well-rounded) has to be slightly modified in the case of k-TSP and k-MST. We only mention the features of the TSP algorithm that need to be changed. (See appendix for details.)

Steiner Tree: The one change 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.

It is easy to see that if the Steiner tree is (m,r) light, then a square in the shifted tree that has no input nodes can only contain $O(r)$ Steiner nodes. So the portals chosen on such a square define a problem instance with at most $4r$ nodes, which can be solved in constant time (since r is constant).

k-TSP and k-MST: The "perturbation" at the start of the algorithm requires some care, but otherwise the algorithm and its analysis is similar to the TSP case. The running time is higher than in the TSP case by $O(k)$. The reason is that we have 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.

Degree-restricted spanning tree: The patching lemma doesn't apply directly because of the degree constraint, but can be modified for this case. However, our current analysis needs to allow each square in the partition to be crossed $k = O(c^2)$ times instead of $k = O(c)$ as in the other problems.

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.

5 Conclusions

Can the running time of our algorithm be improved?

Since Euclidean TSP is strongly NP-hard (this follows from the reductions in [41], 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 [48] 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 under randomized reductions.

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 $\Re^2$.

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] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. Finding cuts in the TSP (A preliminary report). Report 95-05, DIMACS, Rutgers University, NJ.

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

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

[5] S. Arora and S. Safra. Probabilistic Checking of Proofs: A new characterization of NP. In *Proc. 33rd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1992.

[6] J. Bentley. Fast algorithms for geometric traveling salesman problem. *ORSA J. on Computing,* **4**:387–411, 1992.

[7] M. Bern and D. Eppstein. Approximation Algorithms for Geometric Problems. In [21].

[8] 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.

[9] A. Blum, P. Chalasani, and S. Vempala. A constant-factor approximation for the k-MST problem in the plane. In Proc. 27th ACM Symposium on Theorem of Computing, 1995.

[10] B. Chandra, H. Karloff, and C. Tovey. New results for the old K-OPT algorithm for the TSP. In *Proc. 5th ACM-SIAM Symposium on Discrete Algorithms,* pp 150–159, 1994.

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

[FL81] 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.

[12] D. Z. Du and F. K. Hwang. A proof of Gilbert-Pollak's conjecture on the Steiner ratio. *Algorithmica*, **7:** 121–135, 1992.

[13] D. Eppstein. Faster geometric k-point MST approximation. *To appear in CGTA.*

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

[15] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 2–12, 1991.

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

[17] M. R. Garey, R. L. Graham, and D. S. Johnson. Some NP-complete geometric problems. In *Proc. ACM Symposium on Theory of Computing*, pp 10-22, 1976.

[18] E. N. Gilbert and R. O. Pollak. Steiner minimal trees. *SIAM J. Appl. Math.* **16:**1–29, 1968.

[19] R. L. Graham. Bounds for certain multiprocessing anomalies. *Bell System Tech. J.,* 45: 1563–1581.1966.

[20] R. L. Graham. Personal communication, 1996.

[21] D. Hochbaum, ed. Approximation Algorithms for NP-hard problems. PWS Publishing, Boston, 1996.

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

[23] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Computer and Systems Sci.* **9** (1974), 256–278.

[24] 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.)

[25] D. S. Johnson, C. Papadimitriou and M. Yannakakis. How easy is local search? *J. Computer and Systems Sci.* **37** (1988), 79–100.

[26] W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into Hilbert space. *Contemporary Mathematics* **26** (1984) 189–206.

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

[28] 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.

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

[30] P. Klein and D. Karger. Personal communication, March 1996.

[31] D. Karger, P. Klein, and R. Tarjan. A randomized linear time algorithm for finding minimum spanning trees. *JACM* 1996.

[32] M. W. Krentel. Structure in locally optimal solutions. *Proc. 30th IEEE Symposium on Foundations of Computer Science*, pp 216–221, 1989.

[33] S. Khuller, B. Raghavachari, and N. Young. Low degree spanning tree of small weight. In *Proc. 26th ACM Symposium on Theory of Computing*, 1994.

[34] E. Koutsoupias, M. Grigni, and C. H. Papadimitriou. An Approximation Scheme for Planar Graph TSP. In *Proc. IEEE Symposium on Foundations of Computer Science,* pp 640–645, 1995.

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

[36] S. Lin. Computer solutions for the traveling salesman problem. *Bell System Tech. J.,* 44, 2245–2269.

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

[38] C.S. Mata and J. Mitchell. Approximation Algorithms for Geometric tour and network problems. In *Proc. 11th ACM Symp. Comp. Geom.*, pp 360-369, 1995.

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

[40] 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.

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

[42] C. H. Papadimitriou. The complexity of the Lin-Kernighan heuristic for the traveling salesman problem. Manuscript, 1990.

[43] C. Papadimitriou and M. Yannakakis. The complexity of facets (and some facets of complexity). In *J. of Computer and System Sciences* **28**, pp 244-259, 1984 (prelim. version in Proc. 24th ACM STOC, 1982).

[44] C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *J. of Computer and System Sciences* **43**, pp. 425–440, 1991.

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

[46] S. Sahni and T. Gonzales. P-complete approximation problems. *JACM*, 23:555–565, 1976.

[47] W. D. Smith Finding the optimum N-city traveling salesman tour in the Euclidean plan in subexponential time and polynomial space. *Manuscript*, 1988.

[48] L. Trevisan. The n-dimensional Euclidean TSP is Max SNP-hard, In *ACM STOC*, 1997.

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

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

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

[52] A.Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner Problem. *Algorithmica*, **9:** 463-470, 1993.

Appendix

Proof of Patching Lemma Suppose π crosses S a total of t times, and let $2k$ be the largest even number less than t. Identify $2k$ points $M_1, \ldots, M_{2k}$ where π crosses S. Break π at those points, thus causing it to fall apart into $2k$ paths $P_1, P_2, \ldots, P_{2k}$. 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 J be the multiset of line segments consisting of the following: (i) A minimum cost salesman tour through $M_1, \ldots, M_{2k}$, (ii) A minimum cost near-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. Together with the paths $P_1, \ldots, P_{2k}$, these added segments define a connected 4-regular graph on

$$\{M_1', \ldots, M_{2k}'\} \cup \{M_1", \ldots, M_{2k}"\}\,.$$

An Eulerian traversal of this graph is a closed path that contains $P_1, \ldots, P_{2k}$ and crosses S at most twice. Hence we have proved the theorem for $g = 6$.

5.1 TSP Algorithm for $\Re^d$

Suppose d is a constant independent of n. Our methodology for $\Re^d$ mimics that for $\Re^2$. As before, we start with a rescaling and perturbation to make the instance well-rounded. If L is the length of the smallest axis-aligned cube enclosing the instance (implying $OPT \geq L$), we place a grid of granularity $L/8cn\sqrt{d}$ in $\Re^d$ and move each node to its nearest gridpoint. This changes the cost of the optimal tour by at most $2n \cdot \sqrt{d}L/8cn\sqrt{d}$, which is at most $L/4c$. Then we rescale distances by $L/64cn\sqrt{d}$, so that the minimum nonzero internode distance is 8 units and the size of the bounding cube is $O(c\sqrt{d}n)$. Since c, d are constant, we think of the size of the cube as $O(n)$. Thus the instance is now well-rounded. (Note that this procedure just requires d rounds of sorting, and thus takes $O(nd \log n)$ time.)

The "quad" trees in $\Re^d$ are 2^d-ary trees. The shifted 2^d-ary tree is defined in the obvious way. Note that in d dimensions the boundaries of a region of this tree are not lines but cubes of dimension $d-1$. An m-regular set of *portals* on a dimension $d-1$ cube is an orthogonal lattice of m points in the cube. Thus if the cube has length W, then the spacing between the portals is $W/m^{\frac{1}{d-1}}$

Theorem 5 (Structure Theorem for $\Re^d$) *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 dynamic programming algorithm for $\Re^d$ is basically the same as for $\Re^2$. The 2^d-ary tree has $O(2^d n \log n)$ regions, and each region has 2^d facets. Hence the amount of time needed to "guess" the portals at which the tour enters and leaves a region is $m^{O(2^d r)}$. Hence the total running time is $O(2^d m^{O(2^d r)} n \log n)$, which is $O(n (\log n)^{(O(\sqrt{d}c))^{d-1}})$.

The proof of Theorem 5 is analogous to that of Theorem 2, and we only list the calculations that need to be modified.

Patching Lemma: If a salesman tour crosses a dimension $d-1$ cube k times (where we assume that the cube contains no nodes itself), then the number of crossings can be reduced to at most 2, by augmenting the tour with segments in the cube whose total length is $O(k^{1-\frac{1}{d-1}} W)$, where W is the length of the cube's side.

The proof of the lemma uses remark (ii) following Lemma 3.

Modification to Lemma 4: The grid "lines" are $d-1$ dimensional hyperplanes in $\Re^d$. The Lemma now says: If the minimum internode distance is at least 4, then the sum of the number of times a salesman tour crosses the hyperplanes of the unit grid is at most $5\sqrt{d}/4$ times the cost of the tour.

The proof of the modified lemma is the same as in the planar case, except for the fact that $\sqrt{d}$ is the only available upperbound on the ratio of the ℓ_1 and ℓ_2 costs of the tour.

Proof of Structure Theorem. As before, we start with an optimal salesman tour and use the Patching Lemma repeatedly to make it (m,r)-light. We charge resulting cost increases to grid hyperplanes and show that the expected charge to hyperplane h is at most $t(\pi,h)/2c\sqrt{d}$, where $t(\pi,h)$ denotes the number of times the optimum tour π crosses hyperplane h. Then the theorem follows by linearity of expectations and the modified Lemma 4.

The *maximal level* of a grid hyperplane h is the level of the largest subcube of the shifted 2^d-ary tree that has its boundary on h. Let $a_1, a_2, \ldots, a_d \in [0, L-1]$ be the random shifts used to construct the shifted 2^d-ary tree. In the discussion below let a grid hyperplane h be perpendicular to the jth coordinate axis. Then the maximal level of h only depends upon a_j, and

$$\Pr_{a_j}[\text{maximal level of } h \text{ is } i] = \frac{2^i}{L}. \tag{5}$$

Note that if the maximal level of h is i, then 2^{id} level-i cubes (each of side $L/2^i$) have their boundaries on h. An (m,r)-light salesman path can cross each side of these cubes at most r times, and always through a portal. To ensure that this is the case, we use the Patching Lemma. Furthermore we apply it in a bottom-up fashion: we invoke a MODIFY procedure that ensures for $j = \log L, \log L - 1, \ldots, i$, that the tour crosses the boundaries of each level j cube adjacent to h at most r times.

Let $c_j(h)$ (or just c_j since h is clear from the context) be the number of level j cubes for which the MODIFY procedure invokes the Patching Lemma. For $k \leq c_j$, suppose the kth invocation of the Lemma involved replacing $t_{jk} \geq r+1$ by at most $2 \cdot 2^d$ crossings (where the reason for the $2 \cdot 2^d$ is that the cube may be "wrapped around" and thus actually be a union of up to 2^d smaller regions, for all of which the Patching Lemma may have to be invoked separately). Thus we have

$$\sum_{j=1}^{\log L} \sum_{k=1}^{c_j} (t_{jk} - 2^{d+1}) \leq t(\pi, h), \tag{6}$$

Furthermore, the cost of the MODIFY procedure if the maximal level of h is i is

$$\sum_{j=i}^{\log L} \frac{L}{2^j} \sum_{k=1}^{c_j} g \cdot t_{jk}^{1-\frac{1}{d-1}},$$

where g is the constant appearing in the Patching Lemma for $\Re^{d-1}$. Thus by (5), the expected cost increase at hyperplane h is at most

$$\sum_{i=1}^{\log L} \frac{2^i}{L} \sum_{j=i}^{\log L} \frac{L}{2^j} \sum_{k=1}^{c_j} g \cdot t_{jk}^{1-\frac{1}{d-1}}.$$

By rearranging the order of summation, this becomes

$$\sum_{j=1}^{\log L} \frac{1}{2^j} \sum_{k=1}^{c_j} g \cdot t_{jk}^{1-\frac{1}{d-1}} \sum_{i=1}^{j} 2^i,$$

which simplifies to

$$2g \cdot \sum_{j=1}^{\log L} \sum_{k=1}^{c_j} t_{jk}^{1-\frac{1}{d-1}}.$$

Since the t_{jk}'s satisfy (6) and are $\geq r+1$, this cost is maximized when each $t_{jk} = r+1$. In this case (6) simplifies to

$$\sum_{j=1}^{\log L} c_j \leq \frac{t(\pi, h)}{r + 1 - 2^{d+1}}.$$

Similarly, the upperbound on the expected cost simplifies to

$$2g \cdot (r+1)^{1-\frac{1}{d-1}} \sum_{j=1}^{\log L} c_j,$$

which is at most

$$2g(r+1)^{1-\frac{1}{d-1}} \cdot \frac{t(\pi, h)}{r + 1 - 2^{d+1}}.$$

When $r = (O(\sqrt{d}c))^{d-1}$, this upperbound is at most $t(\pi, h)/4c\sqrt{d}$.

The analysis of the cost of moving all crossings to their nearest portal is essentially unchanged from the planar case. Namely, on a grid hyperplane h, at most $t(\pi, h)$ crossings will ever need to be moved to their nearest portal. If the maximal level of the hyperplane is i, then each crossing needs to move at most a distance $\sqrt{d}L/2^i 2^d m^{1-\frac{1}{d-1}}$ (since the portals on the boundary of a level i cube form a grid of granularity $L/2^i m^{1-\frac{1}{d-1}}$). The probability that i is the maximal level of h is at most $2^i/L$, so an upperbound on the cost of moving all crossings on h to their nearest portals is

$$\sum_{i=1}^{\log L} \frac{2^i}{L} \cdot t(\pi, h) \cdot \frac{\sqrt{d}L}{2^{i+d} m^{\frac{1}{d-1}}}.$$

When $m = (O(\sqrt{d}c \log L))^{d-1}$, this cost is at most $t(\pi, h)/4c\sqrt{d}$.

5.2 Other Geometric Norms

When distance is measured using any geometric norm other than ℓ_2, Lemma 4 and the Patching Lemma continue to hold. So the Structure Theorem still holds, and the algorithm works unchanged.

6 Approximation Schemes for other Geometric Problems

To obtain approximation schemes for the geometric problems mentioned earlier, we imitate our TSP methodology. First we rescale and perturb the instance to make it "well-rounded." A well-rounded instance is one in which all coordinates are integral, the minimum internode distance is 8, and the size of the bounding square is $O(n^3)$. The perturbation procedure for the different problems are similar the one for the TSP, although some care is occasionally needed (see below). Next, for all the problems we prove an analogue of the Structure Theorem, whose statement is essentially the same as that of Theorem 2 except the words "salesman path" have to be replaced by problem-specific words such as "Steiner Tree." The theorem is proved just as in the TSP case using a charging argument. The dynamic programming algorithm to find the optimum (m, r)-light solution is straightforward.

We emphasize that the main reason that the TSP methodology can be applied to these other problems is that the Patching Lemma and Lemma 4 generalize to these problems.

Lemma 6 (General Patching Lemma) *For each of the problems Min-Steiner Tree, k-TSP, k-MST and Min-Cost-Perfect-Matching, there is a constant $g > 0$ such that the following is true. Suppose π is any solution and S is any line segment of length l that π crosses least thrice. Then all but at most 2 of these crossings can be broken and instead some line segments on S can be added to the solution so that the new solution has cost at most $g \cdot l$ more than the old solution.*

Notes: (i) As in the TSP case, we are allowing solutions to contain "bent" edges. When we augment the original solution π with line segments on S, some of the new edges are "bent" edges. Solutions with bent edges pose no problems for our dynamic programming, and at the end we can straighten bent edges without increasing the cost. (ii) The Patching Lemma also generalizes to Minimum Degree-restricted Spanning Tree; see Section 6.3 below.

Proof: For k-TSP the proof is the same as for the TSP. For k-MST and Min-Steiner-Tree, the proof is also trivial: we just break the tree at all but one of the points where it crosses S, and add straight line segments along S between all connected components of the tree. This makes the portion of the tree on each side of S connected, and the length of the added segments is clearly $\leq 2l$. The same method works for Min-Cost-Perfect-Matching. $\square$

Lemma 7 *Let S be a collection of line segments in $\Re^2$ each of length ≥ 4. For a line l in the unit grid let $t(S,l)$ be the number of times these segments cross l. Then*

$$\sum_{l:vertical} t(S,l) + \sum_{l:horizontal} t(S,l) \quad \leq \; 2 \cdot cost(S).$$

Proof: Just uses the observation that the left hand side roughly measures the ℓ_1 cost of segments in S. $\Box$

How to make the instance well-rounded. The procedure to make TSP instances well-rounded does not generalize to k-TSP, k-MST and Euclidean matching. Instead we outline a more general procedure. The procedures for the individual problems are described in more detail later.

First we note why the TSP procedure does not work: it relies on the fact that $OPT \geq L$, where L is the size of the bounding box. Let us recall why. 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 our procedure coalesces nodes that are too close together, thus ensuring $d_{min} \geq L/8cn$. Hence the quadtree has depth $O(\log n)$, which is crucial for obtaining an efficient algorithm. 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 $OPT \geq L$.

The above argument breaks down in case of k-TSP, k-MST and Euclidean Matching because there $OPT \ll L$ is possible.

Instead, the rounding procedure first 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.) Let $A \leq O(OPT \cdot n)$ be this crude approximation. The procedure lays a grid of granularity $O(A/8cn^2)$ in the plane and moves every node to its nearest gridpoint. This ensures $d_{min} \geq A/8cn^2$ in the new instance, and triangle inequality implies that the change in the cost of the optimum solution is at most $2n \cdot A/8cn^2 \leq OPT/4c$.

We haven't yet ensured that $O(\log(\frac{L}{d_{min}})$, the depth of the quadtree, is $O(\log n)$. Now suppose $A \ll L$. We pick random shifts as before and construct a shifted quadtree (this can be done in $O(n \log n)$ time [8]). 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 $O(\log 8cn^2) = O(\log n)$.) To see that this is justified, recall that Lemma 7 assures us that at most $2A$ lines of the 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!

This finishes our outline of the perturbation procedure. We note that the use of randomness in the previous step can be removed by trying all possible shifts between 1 and $A \log n/d_{min}$, which multiplies the running time by a factor $O(n^6)$ (by using a better quality approximation at the start, this factor can be reduced to $O(n^2 \log^2 n)$ in case of Min Cost Matching and to $O(k^3)$ in case of k-TSP and k-MST).

6.1 Steiner Tree

A *well-rounded* instance of the Steiner Tree problem is one in which all nodes and Steiner nodes have integer coordinates, the minimum (nonzero) internode distance is at least 8 and the smallest square bounding the instance has size at most $O(n)$.

In order to make the instance well-rounded, we use the fact that a Steiner node has degree ≥ 3 (this is also true in $\Re^d$ and also if distances are measured in any fixed geometric norm), so a Steiner tree problem on n nodes has at most n Steiner nodes.

Let L be the size of the bounding square, so $OPT \geq L$. We place a grid of granularity $L/16nc$ and move each node to its nearest grid point. We thereafter require all Steiner nodes to also lie on the grid. Since the optimum solutions contains only $2n$ nodes (including Steiner nodes) and each moved by a distance at most $L/16nc$, the cost of the optimal tree changed by at most $L/4c$, which is at most $OPT/4c$. Rescaling distances by $L/128nc$, the instance becomes well-rounded.

Note that by requiring the Steiner nodes to lie on the grid, we have forestalled any precision issues which would otherwise have arisen when the algorithm "guesses" the location of a Steiner node.

Recall that in the TSP case we had introduced portals in the quadtree and (somewhat unnaturally) forced the salesman path to cross region boundaries only at portals. In the Steiner Tree problem portals can be viewed more naturally: they are "Steiner" nodes. We define an (m, r)-light Steiner tree by analogy to an (m, r)-light salesman path.

Theorem 8 (Structure Theorem for Steiner Tree) *Let $c \geq 0$ be any constant. Let L be the size of the bounding box of some well-rounded Steiner tree instance and let shifts $0 \leq a, b \leq L$ be picked randomly. Then with probability at least $1/2$, the dissection with shift (a, b) has an associated Steiner tree of cost at most $(1 + 1/c)$-OPT that is (m, r)-light, where $m = O(c \log L)$ and $r = O(c)$.*

Proof: The proof is essentially identical to the TSP case. □

The dynamic programming algorithm to compute the optimal (m, r)-light Steiner tree is similar to that for the TSP, except for two changes. (i) The basic subproblem in the dynamic programming involves finding the portion of the optimal tree inside a given square of the quadtree, and this portion may not be connected. Hence the input to the subproblem consists of the square, a multiset of $\leq r$ portals on each of the four sides of the square, and *a partition of this multiset.* (In case of TSP the subproblem's input contained a *pairing* of the portals in this multiset.) The partition is meant to specify subsets of portals that lie on the same subtree inside the square. (ii) A square in the shifted quadtree that does not contain any input nodes may nevertheless contain Steiner nodes, which have to be "guessed" by the algorithm. However, since the square is entered and left only $4r$ times, we can 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).

Finally, we observe that the proof of Theorem 8 actually proves something about the structure of a near-optimal Steiner tree that may be of independent interest.

Corollary 9 *In the $(1+1/c)$-approximate Steiner tree whose existence is guaranteed by Theorem 8, every Steiner node used in the tree is either a portal of the shifted quadtree or contained in a square at the leaf of the quadtree.*

The $\Re^d$ case. The description of the approximation scheme for Minimum Steiner Tree in $\Re^d$ can also be derived easily from the corresponding TSP algorithm.

6.2 k-TSP and k-MST

The description of the algorithm for these two problems is almost the same as for TSP and Steiner tree respectively. A *well-rounded* instance for these problems is one in which all coordinates are integral, the minimum nonzero internode distance is 8, and the size of the bounding square is

$O(k^2)$. A simple $O(nk \log n)$ time perturbation makes the instance well-rounded[3]. This relies on the observation that the cost of the optimum solution to k-MST (also k-TSP) is within $2\sqrt{k}$ of the size of the smallest square containing at least k points (in $\Re^d$ it is within $dk^{1-\frac{1}{d-1}}$ of the size of such a square). Eppstein and Erickson [14] show how to approximate the size of this square within a factor 2 by computing for each of the n nodes its k nearest neighbors. This takes $O(nk \log n)$ time.

Once we have an approximation A such that $OPT \le A \le 2\sqrt{k}OPT$, we place a grid of granularity $A/16ck^{3/2}$ and move each node to its nearest gridpoint. By the triangle inequality, the cost of the optimum changed by at most $2k \cdot A/16ck^{3/2} \le OPT/4c$. Then we pick random shifts and construct the shifted quadtree. We treat all squares of size $A \log k$ in the quadtree as independent instances of the problem. In each of these squares the ratio of the size of the square to the minimum internode distance is $\log k \cdot 16ck^{3/2} = O(k^2)$, so the square is well-rounded.

We define (m,r)-lightness for k-TSP and k-MST by analogy with the TSP. The statement of the Structure Theorem for them is entirely analogous. The dynamic programming for finding the optimum (m,r)-light k-TSP solution differs from the TSP algorithm in only the definition of the subproblem for the dynamic programming. In the TSP case this subproblem involves finding a set of disjoint paths that visit *all* the nodes inside the given square. In the k-TSP case the set of paths have to visit only k_1 nodes inside the given square, where $k_1 \le k$. Since k_1 is not known to the algorithm *a priori*, it has to try all possible values for it, and we then need to redo our previous analysis for the size of the lookup table and the running time.

For $i \le O(\log n)$ and $j \le n$, let t_{ij} be the number of squares at level i of the quadtree that contain exactly j input nodes. Then we have for all i,

$$\sum_j j \cdot t_{ij} = n.$$

Suppose a square has j input nodes. It contains $k_1 \le \min\{k,j\}$ nodes from the optimum (m,r)-light solution. Not knowing k_1, the algorithm tries all values for it. For each value of k_1, the lookup table contains $m^{O(r)}$ subproblems (obtained by trying all possible ways in which the tour could enter/leave the square). Thus the table contains solutions to $m^{O(r)} \cdot \min\{k,j\}$ subproblems for this square. Thus the size of the lookup table is

$$\sum_{i \le O(\log n)} \sum_{j \le n} \min\{k,j\} \cdot m^{O(r)} t_{ij},$$

which is $O(nm^{O(r)} \log n)$.

Now we analyze the running time of the dynamic programming. First we modify the dynamic programming a little. Namely, instead of storing subproblems corresponding only to squares, we also store subproblems corresponding to halves of squares. We will refer to these halves as *rectangles.* (Viewed in terms of the quadtree, our modification involves introducing additional nodes to reduce the degree of the tree from 4 to 2.) This modification only doubles the size of the lookup table.

Now suppose a square/rectangle has j input nodes. The lookup table contains $m^{O(r)} \cdot \min\{k,j\}$ subproblems for it, where each subproblem involves an input $k_1 \le \min\{k,j\}$ that is the algorithm's "guess" for the number of nodes from the optimum (m,r)-light solution that lie in the square/rectangle. When the algorithm partitions the square/rectangle into two equal regions, it is faced with a potential $k_1 + 1$ number of guesses for the number nodes from the optimum solution in the two halves.

[3]Eppstein and Erickson have pointed out to us that the perturbation can be implemented in $n\mathrm{poly}(\log n)$ time.

Thus the running time is

$$\sum_{i \le O(\log n)} \sum_{j \le n} n_{ij} \cdot m^{O(r)} \cdot (\min\{k, j\})^2,$$

which is upperbounded by $O(nkm^{O(r)} \log n)$.

The description of the algorithm for k-MST is entirely analogous. The description of the algorithm in $\Re^d$ is also straightforward.

6.3 Degree-restricted-MST

The statement of the Patching Lemma for degree-restricted MST is somewhat different from the other problems. Instead of concentrating on line segments, we need to concentrate on *squares* (or *cubes* in $\Re^d$).

Lemma 10 *For every degree bound k there is a constant $g > 0$ such that the following is true. Suppose π is any spanning tree of degree at most k and suppose S is any square of length l whose boundaries π crosses $t \ge 3$ times.*

Then all but at most 1 of these crossings can be broken and instead some new edges can be added to π so that the portion of π inside S becomes a connected tree of degree $\le k$ and the cost rises by at most $g\sqrt{t} \cdot l$.

Proof: If the degree bound $k = 2$, the tree is a salesman tour with a missing edge, and the lemma is true. Let $k \ge 3$.

Let $P_1, \ldots P_t$ be the points on the boundary where the t crossings occur. Break the tree at $P_1, \ldots P_{t-1}$, thus breaking the portion of the tree inside S into some $k \le t-1$ connected components. For each component, pick one or two *representatives* as follows. If the component has a leaf inside S, let it be the representative. Otherwise the component must touch the boundary at least twice, so pick two of these points (which are among $P_1, \ldots, P_{t-1}$) as its representatives.

Now take any degree-2 MST on the representatives and add it to the solution. By the well-known bound, the cost of this is $O(\sqrt{t}l)$. $\square$

The Structure Theorem for this problem shows the existence of an (m, r)-light tree, where $m = O(\log n)$ and $r = O(c^2)$. This leads to a $O(n(\log n)^{O(c^2)})$ time dynamic algorithm.

Note that the dynamic programming needs to guess the degrees only of the portals; all other degrees are decided in the "base case", when a brute force algorithm is used on an instance with $O(r)$ nodes.

6.4 Min Cost Perfect Matching

A *well-rounded* instance of the Euclidean Min Cost Perfect Matching problem is one in which all coordinates are integral, the minimum nonzero distance is 8, and the size of the bounding square is $O(n^2)$. We already outlined a procedure that makes the instance well-rounded. Now we describe it in more detail. The procedure uses a crude approximation algorithm due to Vaidya [50] that runs in $O(n \log^3 n)$ time and computes an $O(\log n)$ approximation to the cost of the optimum matching[4].

[4]Alternatively, one may use the following greedy procedure: each node v proposes a match to its nearest neighbor $N(v)$. $N(v)$ accepts iff v is its closest neighbor. If $N(v)$ decides to reject v, we try to match v with any other unsuccessful suitor of $N(v)$. However, if the number of unsuccessful suitors of $N(v)$ is odd, this leaves one suitor still unmatched. At the end of the round, all nodes who managed to match are removed and the rest are sent into another round of matching. It is easy to see that at every step a vertex can receive at most 6 proposals (and $2^{O(d)}$

Let $A \leq O(OPT \log n)$ be the cost of this solution. We place a grid of granularity $A/n \log^2 n$ and move each node to its nearest gridpoint. If a gridpoint receives an even number of nodes, we remove them; if it receives an odd number, we replace them by one node. (At the end of the algorithm we will put a trivial matching among each set of removed nodes, which will add at most $O(n \cdot A/n \log^2 n) \leq OPT/4c$ to the cost.) Using triangle inequality it is easy to see that the cost of the min-cost perfect matching in the new instance is at most $OPT + O(n \cdot A/cn \log n) \leq OPT(1 + 1/2c)$. Clearly, $d_{min} \geq A/n \log^2 n$ in the new instance.

Finally, we construct a quadtree with random shifts and treat all squares of size more than $A \log n$ as independent instances, in each of which the depth of the quadtree is $O(\log(n \log^2 n / \log n)) = O(\log n)$. This completes the description of the perturbation procedure.

The rest of the algorithm and its analysis mimics the TSP case.

proposals if the points are in $\Re^d$) so the number of unmatched nodes falls by a constant factor at each step. Using standard algorithms for nearest neighbor searching, the running time is

$$O(n \log n + \frac{n}{2} \log \frac{n}{2} + \frac{n}{4} \log \frac{n}{4} + \cdots,)$$

which is $O(n \log^2 n)$. Furthermore, a simple "charging" argument shows that the resulting matching is within a factor $O(\log n)$ of the optimal.