Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems

Sanjeev Arora*
Princeton University

Abstract

We present a polynomial time approximation scheme for Euclidean TSP in $\Re^2$*. Given any* n *nodes in the plane and* $\epsilon > 0$*, the scheme finds a* $(1+\epsilon)$*-approximation to the optimum traveling salesman tour in time* $n^{O(1/\epsilon)}$*. When the nodes are in* $\Re^d$*, the running time increases to* $n^{\tilde{O}(\log^{d-2} n)/\epsilon^{d-1}}$*. The previous best approximation algorithm for the problem (due to Christofides) achieves a* $3/2$*-approximation in polynomial time.*

We also give similar approximation schemes for a host of other Euclidean problems, including Steiner Tree, k-TSP, Minimum degree-k *spanning tree,* k*-MST, etc. (This list may get longer; our techniques are fairly general.) The previous best approximation algorithms for all these problems achieved a constant-factor approximation.*

All our algorithms also work, with almost no modification, when distance is measured using any geometric norm (such as ℓ_p *for* $p \geq 1$ *or other Minkowski norms).*

1 Introduction

In the *Traveling Salesman Problem* ("TSP"), we are given n nodes and for each pair $\{i, j\}$ of distinct nodes a distance $d_{i,j}$. We desire a closed path that visits each node exactly once (i.e., is a *salesman tour*) and incurs the least *cost* (which is the sum of the distances along the path). This classic problem has proved a rich testing ground for most important algorithmic ideas during the past few decades, and influenced the emergence of fields such as operations research, polyhedral theory and complexity theory. For a fascinating history, see Lawler et al. [29].

Since the 1970s, mounting evidence from complexity theory suggests that the problem is computationally difficult. Exact optimization is NP-hard (Karp [23]). So is approximating the optimum within *any* constant factor (Sahni and Gonzalez [40]). There are also other reasons to believe in the TSP's nastiness (cf. D^P completeness [37] and PLS-completeness [20]).

But TSP instances arising in practice are usually quite special, so the hardness results may not necessarily apply to them. In *metric TSP* the nodes lie in a metric space (i.e., the distances satisfy the triangle inequality). In *Euclidean TSP* the nodes lie in $\Re^2$ (or more generally, in $\Re^d$ for some d) and distance is defined using the ℓ_2 norm. Note that Euclidean TSP is a subcase of metric TSP.

Unfortunately, even Euclidean TSP is NP-hard (Papadimitriou [35], Garey, Graham, and Johnson [12]). Therefore algorithm designers were left with no choice but to consider more modest notions of a "good" solution. Karp [24], in a seminal work on *probabilistic analysis of algorithms*, showed that when the n nodes are picked uniformly and independently from unit square, then the *fixed dissection heuristic* with high probability finds tours whose cost is within $1+\epsilon$ of optimal (where $\epsilon > 0$ is arbitrarily small). Christofides [9] designed an *approximation algorithm* that on every instance of metric TSP computes a tour of cost at most 1.5 times the optimum.

Two decades of research failed to improve upon Christofides' algorithm for metric TSP. But some researchers continued to hope that even a PTAS might exist. A PTAS or *Polynomial-Time Approximation Scheme* is a polynomial-time algorithm — or a family of such algorithms— that, for each fixed $\epsilon > 0$, can approximate the problem within a factor $1+\epsilon$. (The running time could depend upon ϵ, but for each fixed ϵ has to be polynomial in the input size.) PTAS's are known for very few problems; two important ones are in [17, 22]. Recently Arora, Lund, Motwani, Sudan, and Szegedy [3] showed that if P $\neq$ NP, then metric TSP and many other problems do not have a PTAS. Their work relied upon the theory of *MAX-SNP-completeness* (Papadimitriou and Yannakakis [38]), the notion of *probabilistically checkable proofs* or *PCP*s (Feige, Goldwasser, Lovász, Safra and Szegedy [11], Arora and Safra [4]), and the connection between PCPs and hardness of approximation [11].

The status of Euclidean TSP remained open, however. In this paper, we show that Euclidean TSP has a PTAS. For

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

every $\epsilon > 0$, the PTAS computes a $(1+\epsilon)$-approximation to the optimal tour in $n^{O(1/\epsilon)}$ time. When the nodes are in $\Re^d$, the running time rises to $n^{O(\log^{d-2} n \log\log n)/\epsilon^{d-1}}$. Our techniques apply to many other geometric problems, which are described in Section 1.1.

We design the PTAS by showing that every TSP instance in $\Re^2$ (also in $\Re^d$ for every fixed d) has a $(1+\epsilon)$-approximate tour with the following very simple structure: there is a way to recursively partition the plane so that "very few" edges of the tour cross each line of the partition (see Theorem 4). A tour with such simple structure can be found using dynamic programming. We remark that the idea of partitioning a TSP instance into smaller instances has been used before, most famously in [24]. Dynamic programming has also been used before, most recently in an approximation scheme for planar graph TSP [28].

Our Structure Theorem about near-optimal tours also seems to shed some light — at least at an intuitive level— on one mysterious aspect of the TSP: the remarkable performance of simple heuristics. The most well-known of them, such as K-OPT or Lin-Kernighan [31], date to the 1960s and 70s. Using simple local-exchange rules, they quickly come up with very good salesman tours on "real-life" TSP instances [19, 5]. But many of these "real-life" instances are either Euclidean or derived from Euclidean instances! (See for example the TSPLIB library [39].) Since our structure theorem shows that such instances have near-optimal salesman tours with a very simple structure, the fact that simple heuristics can find such tours should be no mystery.

We cannot show, however, that any known heuristic is a PTAS[1]. But maybe our techniques will motivate further research on this topic. For example, even our current dynamic programming algorithm can be viewed (after some twists in the definition of "local search") as a local search algorithm that performs upto $O(\log n/\epsilon)$ edge exchanges per step (see Section 2.4).

Finally, the inevitable question: Is our PTAS practical? A straightforward implementation (for even moderate values of ϵ) is very slow, but we see no reason why a speedier, more subtle, implementation may not exist (see Section 4.1 in the appendix). At the very least, the Theorem gives a way of decomposing TSP instances into a large number of "independent" and smaller instances, and this may prove helpful in parallelizing existing TSP routines.

[1] In fact, thus far there is no evidence that any of the known heuristics is a PTAS for Euclidean TSP. The few published results in fact suggest quite the opposite. With an adversarially-chosen starting tour, K-OPT (for any constant K) may produce a tour whose cost is $\Omega(\log n/\log\log n)$ times the cost of the optimum tour, even when the n nodes lie in $\Re^2$ [8]. In case of metric TSP, finding a locally-optimum tour for K-OPT (for $K \geq 8$) is PLS-complete [26]. (This strongly suggests that no polynomial-time algorithm can find such a local optimum; see [20].) Many variants of Lin-Kernighan are also PLS-complete [36].

1.1 Steiner Tree and other geometric problems.

Many network problems identified in the past few decades are very similar to the TSP. Below, we define some of them. We will restrict attention to the Euclidean (or geometric) versions of these problems. The best approximation algorithms for all of them achieve a constant factor approximation in polynomial time (see the survey by Bern and Eppstein [6]). We are able to design PTAS's for the planar versions, and $n^{O(\log^{d-2} n \log\log n)/\epsilon^{d-1}}$ time approximation schemes for the $\Re^d$ versions.

Minimum Steiner Tree: Given n nodes in $\Re^d$, find the minimum cost tree connecting them[2]. In general, the minimum spanning tree is not an optimal solution (as observed many decades ago). In $\Re^2$ the cost of the MST can be as far as a factor $2/\sqrt{3}$ from the optimum. (Furthermore, the famous Gilbert-Pollak [13] conjecture said it can't be any further from the optimum; the conjecture was proved by Du and Hwang [10]). A spate of research activity in recent years (starting with the work of Zelikovsky[42]) has provided better algorithms; with an approximation ratio approaching 1.10. The metric case is MAX-SNP-hard.

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. When $k = 2$, the problem is polynomial-time equivalent to the TSP and hence NP-hard. The case $k = 3$ is NP-hard; $k = 4$ is open, and when $k \geq 5$ the problem can be solved optimally in polynomial time. For the cases $k = 3, 4$ in $\Re^2$, a constant-factor approximation algorithm is given by Khuller, Raghavachari, and Young [27].

k-TSP: Given n nodes in $\Re^d$ and an integer $k > 1$, find the smallest tour that visits at least k nodes. An approximation algorithm due to Mata and Mitchell [32] achieves a constant factor approximation in $\Re^2$.

k-MST: Given n nodes in $\Re^d$ and an integer $k \geq 2$, find k nodes with the shortest MST. Blum, Chalasani, and Vempala [7] gave the first $O(1)$-factor approximation algorithm for points in $\Re^2$; there has been much other work before and since.

We remark that until recently the approximation algorithms for the last three problems heavily used the geometry of the plane and broke down even in $\Re^3$. But recent algorithms — discovered independently of our paper — work for any metric space.

[2] It appears that this problem was first posed by Gauss in a letter to Schumacher [15].

2 The TSP Algorithm

As mentioned in the introduction, we design our PTAS for Euclidean TSP in $\Re^2$ by showing that there is a way to recursively partition the plane so that there exists a $(1+\epsilon)$-approximate tour that crosses each line of this partition "very few" times. We state this formally in Theorem 4 and then describe the PTAS. Theorem 4 is proved in Section 2.1, and the algorithm for $\Re^d$ is described in Section 2.2.

In this paper, whenever we say "rectangle", we mean an axis-aligned rectangle. The *size* of the rectangle is the length of its longer edge. The *bounding box* of a set of nodes is the smallest rectangle enclosing them.

To simplify the exposition, we first modify the TSP instances a little so that internode distances are not too different from one another.

Proposition 1 *Let $n, \epsilon > 0$ be such that $n > 10/\epsilon$. Then the problem of computing a $(1+\epsilon)$-approximation to the optimum tour length in an n-node instance can be reduced in poly(n) time to the problem of computing a $(1 + 9\epsilon/10)$-approximation in an instance in which the the smallest internode distance is 1 unit and the size of the bounding box is at most $1.5n^2$.*

Proof: The reduction involves perturbing the n-node instance a little. Let T be the cost of the minimum spanning tree. Note that the optimum salesman tour has cost at least T and at most $1.5T$ [9]. So the size of the bounding box is at most $0.75T$. Construct a new instance by placing a grid of granularity $T/2n^2$ in the plane and moving each node to its nearest gridpoint (this may cause some nodes to merge). Because each node moved by at most $T/2n^2$ and a salesman tour has n edges, the tour cost changed by at most $2n \cdot T/2n^2 = T/n$. Now since $\epsilon/10 > 1/n$, it suffices to compute a $(1 + 9\epsilon/10)$-approximation in the new instance.

Finally, divide all distances in the new instance by $T/2n^2$, so that the smallest internode distance is at least 1 and the bounding box has size at most $0.75T/(T/2n^2) < 1.5n^2$.
$\Box$

Now we define a recursive partition of a rectangle. A *line separator* of a rectangle is a straight line segment parallel to its shorter edge that partitions it into two rectangles of at least $1/3$rd the area. For example, if the rectangle's width W is greater than its height, then a line separator is any vertical line in the middle $W/3$ of the rectangle.

Definition 1 (1/3:2/3 tiling) *A* $1/3 : 2/3$-tiling *of a rectangle R is a binary tree (i.e., a hierarchy) of sub-rectangles of R. The rectangle R is at the* root. *If the size of R is ≤ 1, then the hierarchy contains nothing else. Otherwise the root contains a line separator for R, and has two subtrees that are $1/3 : 2/3$-tilings of the two rectangles into which the line separator divides R.*

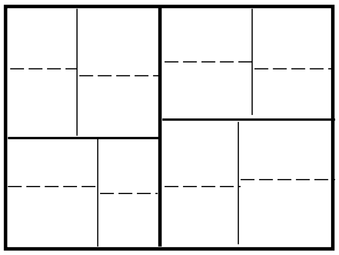

Figure 1. A $1/3 : 2/3$ tiling

The depth *of the tiling is the maximum depth of this tree.*

Note that rectangles at depth d in the tiling form a partition of the root rectangle. The set of all rectangles at depth $d+1$ is a refinement of this partition obtained by putting a line separator through each depth d rectangle of size > 1. The area of any depth d rectangle is at most $(2/3)^d$ times the total area. The following proposition is therefore immediate.

Proposition 2 *If a rectangle has width W and height H, then its every $1/3 : 2/3$ tiling has depth at most $\log_{1.5} W + \log_{1.5} H + 2$.*

We need one more definition about tilings.

Definition 2 (portals) *A* portal *in a $1/3 : 2/3$-tiling is any point that lies on the edge of some rectangle in the tiling. If m is any positive integer then a set of portals P is called* m-regular *for the tiling if there are exactly m equidistant portals on the line separator of each rectangle of the tiling. (We assume that the the end-points of the line separator are also portals. In other words the line separator is partitioned into exactly $m-1$ equal parts by the portals on it.)*

Now we indicate how the above ideas are used. We first note that an optimum salesman tour is always a simple polygon. To simplify the exposition, we will allow tours with "bent" edges. These bent edges arise as follows. We introduce additional ("Steiner") nodes in the plane (these will be portals in some $1/3 : 2/3$ tiling) and ask that the tour visit these nodes in addition to the input nodes. We call such a tour a *salesman path*. Of course, at the end of the algorithm we can change a salesman path into a polygonal tour by straightening the bent edges (i.e., removing the additional points).

To avoid disturbing the tour too much, we wish to limit the number of additional nodes. This motivates the following definition.

Definition 3 (m-light) *Let $m \in \mathcal{Z}^+$ and π be a salesman path on some set of nodes. Let $\mathcal{S}$ be a $1/3 : 2/3$ tiling of the bounding box and P be an m-regular set of portals on this tiling. Then π is* m-light with respect to $\mathcal{S}$ *if the following* *are true: (i) in each rectangle of tiling $\mathcal{S}$, the path crosses the line separator of that rectangle at most m times (ii) the path crosses the line separator only at portals in P (iii) the path is nonself-intersecting (except possibly at a portal).*

Remark: We allowed the path to intersect itself by passing through a portal k times, where $k > 1$. Alternatively, we could insist on non-self-intersecting tours as follows. We allow the portal to be replaced by a tiny array of k points around it and then the path can use a different point for each crossing. This affects the cost negligibly, so the two viewpoints are equivalent.

Proposition 3 *If a salesman path π is m-light with respect to a $1/3 : 2/3$-tiling of the bounding box, then the perimeter of every rectangle in the tiling is crossed by the path at most $4m$ times.*

Proof: Each side of the rectangle is a line separator in some rectangle that is its ancestor in the tiling tree. Therefore the salesman path crosses that side at most m times. □

The following theorem (which is proved in Section 2.1) shows that every TSP instance has a near-optimal salesman path (and hence a near-optimal salesman tour) with a very simple structure.

Theorem 4 (Structure Theorem) *There exists a $c > 0$ such that the following is true for each $\epsilon > 0$. Every set of nodes in $\Re^2$ has a $(1+\epsilon)$-approximate salesman path π and an associated $1/3 : 2/3$ tiling of the bounding box such that the tour is m-light for this tiling, where $m = c \log L/\epsilon$ and L is the size of the bounding box.*

Assuming the truth of the Structure Theorem, we describe the PTAS. Since $\epsilon > 0$ is arbitrary, we use Proposition 1 first and assume w.l.o.g. that the minimum internode distance is 1 and the bounding box has size at most $O(n^2)$. The Structure Theorem guarantees the existence of a $(1+\epsilon)$-optimal salesman path π and a tiling $\mathcal{S}$ with respect to which the path is m-light, where $m = O(\log n/\epsilon)$. By Propostition 2, the depth of any such tiling is at most $O(\log n)$. We describe a simple dynamic programming that finds both $\mathcal{S}$ and π in poly$(n)2^{O(m)} = n^{O(1/\epsilon)}$ time.

The intuition behind the dynamic programming is simple. The bounding box of the instance is guaranteed to have a line separator which π crosses only $2k$ times, where $2k \leq m$. Suppose a "little birdie" announces this separator, the portals where the crossings occur, and the order in which these crossings occur in the final tour. Then on each side of the separator we are left with an instance of what we call the *subtour problem.* In this problem we are given a rectangle with some nodes in it, and a set of k pairs of portals on its boundary (a portal could appear in more than one pair). We wish to find vertex-disjoint paths that connect all portal pairs, in such a way that every node inside the rectangle is visited by one of the paths, and the total length of the paths is minimized.

The subtour problem is no easier than the TSP in general. But note that our two instances are guaranteed to have a nice structure: each has a line separator that is crossed by the subtour at most m times. Assume the "little birdie" continues to help us and announces those line separators and the portals on them where the crossings occur. Then we can break each instance into another 2 instances of the subtour problem. Continuing this way we obtain smaller and smaller instances of the subtour problem, and we stop when the instance is small enough to allow a brute-force solution.

All this assumed the existence of a helpful "little birdie." But notice that m is so small that we don't really need the birdie. Instead, at each step we try all combinatorially distinct line separators[3], of which there are n, and all possible orders in which the tour could cross this separator $\leq m$ times. (The number of such ways turns out to be $2^{O(m)}$ in $\Re^2$.) This is the idea behind the dynamic programming.

The algorithm's lookup table will have poly$(n) \cdot 2^{O(m)}$ entries, and the algorithm will spend poly$(n) \cdot 2^{O(m)}$ time to build each entry. Since $m = O(\log n/\epsilon)$, the total running time will be $n^{O(1/\epsilon)}$. Now we describe the table. An entry is indexed by a triple of the following type: (a) A rectangle inside the bounding box. Only combinatorially distinct rectangles are considered. (b) A multiset consisting of $2k$ portals on the perimeter of the rectangle, where $2k \leq 4m$. (c) A partition of the $2k$ portals into k pairs. Each choice in (a), (b) and (c) gives rise to an instance of the subtour problem. The entry in the lookup table stores the lowest cost subtour found by the algorithm for this instance.

The size of the lookup table is at most

$$(\#\ choices\ in\ (a)) \times (\#\ choices\ in\ (b)) \times (\#\ choices\ in\ (c)).$$

To upperbound this, we assume —by maximally "shrinking" the rectangles in the $1/3 : 2/3$-tiling — that every line separator passes through a node. Hence the number of combinatorially distinct rectangles in (a) is at most $\binom{n}{4}$, the number of 4-tuples of nodes.

The number of choices in (b) is $O(n^8) \times 2^{4m+2k}$. The reason for the $O(n^8)$ is that each rectangle has 4 sides, and each is part of the line separator of some ancestor of the rectangle. The m portals on that line separator are evenly spaced, so they are completely determined once we know the line separator. But the number of choices for a line separator is at most the number of pairs of nodes, which is $\binom{n}{2}$. This observation accounts for the factor $O((n^2)^4) = O(n^8)$. Furthermore, once we have identified the set of

[3] Two line separators are *combinatorially distinct* if at least one node lies between them.

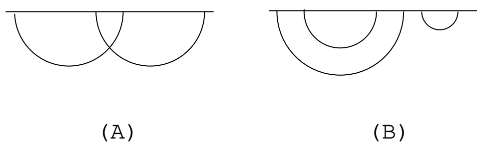

Figure 2. For nonself-intersecting tours, valid pairings of $2k$ portals correspond to balanced arrangements of k pairs of parenthesis. (A) shows an invalid pairing and (B) shows a valid pairing.

$\leq 4m$ portals on the four sides, the number of ways of choosing a multiset of $2k$ out of them is at most 2^{4m+2k}.

Finally, the number of possible pairings in (c) is at most 2^{4m}. The reason is that in a nonself-intersecting salesman path, an acceptable pairing of the portals corresponds to a balanced arrangement of $2k$ parentheses (see Figure 2). The number of such arrangements is the k'th *Catalan number*, which is $\leq 2^{2k} \leq 2^{4m} = n^{O(1/\epsilon)}$.

Hence we can upperbound the size of the lookup table by

$$\sum_{k=1}^{2m} n^4 \times n^8 \times 2^{4m+2k} \times 2^{4m},$$

which is $O(n^{12}2^{12m}) = n^{O(1/\epsilon)}$.

Now we describe how to build the entries of the lookup table by working up from the leaves of the tiling. Entries corresponding to rectangles that contain at most $4m$ nodes are computed using a brute-force algorithm in $2^{O(m)}$ time. To compute a table-entry corresponding to any other rectangle, we enumerate all triples of the following kind: (i) all possible $\leq n$ combinatorially distinct line separators for the rectangle (ii) $j \leq m$ portals in which the tour crosses this line separator and (iii) all possible orders in which these portals could be placed in the subtour while respecting the existing pairing imposed by the $\leq 4m$ portals on the perimeter of the rectangle. Then we look up the appropriate entries for the rectangles on either side of the separator to determine the lowest cost subtour that uses those separator edges in that order. At the end we retain only the lowest cost subtour over all enumerated triples.

The running time of the dynamic programming is clearly upperbounded by

$$2^{O(m)} \times \text{poly}(n) \times \text{Size of the table},$$

which is $n^{O(1/\epsilon)}$.

2.1 Proof of the Structure Theorem

A *random line separator* of a rectangle is a line parallel to its shorter side that had has been placed randomly in the middle $1/3$rd of the box. The following simple theorem about random separators will play a key role.

Theorem 5 *Let C be any collection of straight line segments whose total length is T, and which lie entirely inside a rectangle of size W. Then the expected number of segments of C that cross a random line separator of the rectangle is at most $3T/W$.*

Proof: Let e be a line segment of length l_e. The probability it crosses the separator is $\leq l_e/(W/3)$. (Note: if $l_e > W/3$, the probability could be 1, which however is smaller than $l_e/(W/3)$.) Hence the expected number of edges crossing the separator is at most

$$\sum_{e \in C} \frac{l_e}{W/3} = \frac{3T}{W}.$$

$\square$

Remarks: (i) Note, in particular, that the theorem proves the *existence* of a line separator that is crossed by at most $3T/W$ edges of C. (ii) The theorem also holds for rectangles in $\Re^d$; the separators then are hyperplanes. (iii) Whenever we use Theorem 5 in connection with the TSP, the collection of line segments C mentioned in the hypothesis is part of an optimal or near-optimal salesman tour. In later sections, C will be part of some other geometric object: Spanning Tree, Steiner Tree, etc.

Now we present another simple lemma. In proving the Structure Theorem we will want to say the following: If a salesman tour has a "large" set of edges crossing through a "narrow" strip, then there is tour of "not much greater" cost in which very few edges cross that strip. Also, this new tour should be obtained from the original tour using some local edge exchanges that do not affect edges far away from the strip. The lemma lets us prove exactly this; it has probably been known for a long time (for example, it is implicit in the calculations of [24]).

Lemma 6 (Patching Lemma) *There is a constant $b > 0$ 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 $b \cdot l$ 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, 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 $2k - 1$ paths

$P_1, P_2, \ldots, P_{2k-1}$. 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 $P_1, \ldots, P_{2k-1}$, 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 $P_1, \ldots, P_{2k-1}$ and crosses S at most twice. Hence we have proved the theorem for $b = 6$.
□

Remarks: (i) 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 (see Proposition 7 in the appendix), the patching cost is $O(k^{1-1/d}l)$. (ii) There is an algorithmic version of the Patching Lemma if we don't care about the precise value of b. We will not need the algorithmic version.

Now we prove the Structure Theorem (Theorem 4).

Proof: (Of Theorem 4) The main idea is to start with an optimal tour on the original set of nodes and an empty tiling, and to refine the tiling (i.e., increase its depth) one level at a time, ensuring always that the current tour is m-light with respect to rectangles that are already in the tiling. The refinement at each level puts a line separator through all rectangles of the previous level that have size > 1, identifies some portals on this separator to add to the current set of nodes, and modifies the current tour to cross the separator using only these portals. We will show that such modifications to the current tour increase its cost by a multiplicative factor $(1 + 3b/m)$, where b is the constant appearing in the statement of the Patching Lemma. Since the refinement of the tiling stops after $2\log_{1.5} L$ levels, the cost of the final tour is within a factor $(1 + 3b/m)^{2\log_{1.5} L}$ of the optimal. This factor is less than $(1 + \epsilon/2)$ when $m = 12b\log_{1.5} L/\epsilon$.

Now we describe how to refine a depth d tiling into a depth $d+1$ tiling. Let R be a rectangle at depth d of size W and let T be the length of the edges of the current salesman path that lie inside R. (By our invariant, we know that the tour enters R only through portals.) Now we show how to partition R into two rectangles by picking an appropriate line separator.

Case 1: $T \leq mW/3$**.** Let $\mu = 3T/W \leq m$. Theorem 5 guarantees the existence of a line separator S that is crossed by at most μ edges of the current path. We place m portals on S at intervals of $W/(m-1)$. Next, for each point on S at which the path crosses it, we identify the nearest portal. We displace the edge in question by at most $W/2(m-1)$ so that it passes through this portal. Then we add that portal to the current set of points. At the end, we have ensured that all $\leq \mu$ crossings happen on one of the m portals. In ensuring this, we needed to raise the cost incurred inside rectangle R from T to at most $T + 2\mu \cdot W/2(m-1) = T + \mu W/(m-1) = T + 3T/(m-1)$. For $b \geq 2$, this is $< T + 3bT/m$.

Case 2: $T > mW/3$**.** In this case, take *any* line separator S of R. The current path may be crossing it many times, but we use the Patching Lemma to modify the path so that it then crosses S in at most 2 places. The cost of the patching is atmost $bW \leq 3bT/m$, where b is the constant mentioned in the Patching Lemma. Thus the cost of the current tour inside R has increased from T to $T(1+3b/m)$. (The cost of moving the two crossings to the nearest portals is $2W/m < T/m^2$, which is tiny.)

Finally, notice that in both cases the modifications to the tour involved edges that lie inside R, so the m-lightness property is not affected anywhere else in the tiling.

Ensuring that the salesman path is nonself-intersecting. After each refinement, we replace the portion of the current path inside each rectangle by the optimum — and hence nonself-intersecting — subtour that visits all nodes inside the rectangle while continuing to enter and leave the rectangle through the portals already identified on the boundary. (Note that even a 2-optimal subtour doesn't interesect itself, so we could also use that instead of an optimal subtour.)

This finishes the proof of Theorem 4. □

2.2 Algorithm for $\Re^d$

The statement of the structure theorem for $\Re^d$ is identical to that of Theorem 4, except $m = O((\log L/\epsilon)^{d-1})$ instead of $m = O(\log L/\epsilon)$, and we use "hyperplane separators" instead of "line separators." The dynamic programming algorithm is essentially unchanged, with the following minor quirk. The Catalan-number based upperbound of 2^m on the number of non-crossing pairings of m points does not hold in even $\Re^3$. So we have to use the trivial $m!$ upperbound, which is the reason for the extra $\log\log n$ in the expression for the running time, which is $n^{O(\log^{d-2} n \log\log n)/\epsilon^{d-1}}$.

While proving the structure theorem for $\Re^d$, Case 1 is unchanged from Section 2.1, (see the remark after Theorem 5). In Case 2, instead of using an arbitrary hyperplane separator, we use a hyperplane separator that is crossed by the current salesman path at most $O(T/W)$ times (the existence of such a separator is guaranteed by Theorem 5). We break the path at those crossing points and use the $\Re^d$ version of the Patching Lemma (see the remark after Lemma 6) to patch the tour on both sides of the separator. This involves patching $O(T/W)$ paths whose endpoints lie on a $(d-1)$-dimensional cube of width W. Hence the patching cost is $O((T/W)^{1-1/(d-1)}W)$, which is $\epsilon T/\log L$ since $T/W \leq 3m$. Hence the increase in cost due to each level of refinement is a multiplicative factor $(1+\epsilon/\log L)$, and therefore the cost increase can be sustained for $O(\log L)$ levels.

2.3 Other Geometric Norms

When distance is measured using any geometric norm other than ℓ_2, Theorem 5 and the Patching Lemma continue to hold. So the Structure Theorem still holds, and the algorithm works unchanged.

2.4 The Local-Exchange View

We can also state our dynamic programming algorithm as a divide-and-conquer algorithm with a running time of $n^{O(\log n/\epsilon)}$. The divide-and-conquer version somewhat resembles a local exchange procedure (once we twist the definition of the latter a bit). It starts with a trivial tour (e.g., the "strip" tour) and then tries to find a good $1/3 : 2/3$-tiling and a low-cost m-light tour. It first guesses the line separator of the bounding box, the tour edges that cross the separator, and the order in which these edges occur in the tour. Then it goes ahead to recursively find the portion of the salesman path on both sides of the separator.

Thus the algorithm can be viewed as a local-exchange algorithm that improves the current tour by changing $O(\log n/\epsilon)$ edges at a time. In general the algorithm maintains not a complete salesman tour, but an almost-tour that is a union of $O(\log n/\epsilon)$ disjoint paths. The endpoints of these paths lie inside the rectangle being worked on.

Thus the algorithm is reminiscent of K-OPT for $K = O(\log n/\epsilon)$. The difference from K-OPT is that the algorithm doesn't always maintain a complete tour, and that its exchanges are not always cost-decreasing. (Furthermore, cost-increasing changes seem necessary since the algorithm needs a mechanism to undo bad guesses.) Perhaps the algorithm is closer in spirit to more ad-hoc heuristics such as genetic algorithms, which do allow cost-increasing exchanges under some conditions.

3 PTAS for Steiner Tree and other Geometric Problems

To obtain approximation schemes for the geometric problems mentioned earlier, we imitate our TSP methodology. We only give sketchy proofs here, but the reader should be able to fill in the details.

For each problem we first prove an analogue of the Structure Theorem, whose statement is essentially the same as that of Theorem 4 except the words "salesman path" have to be replaced by the problem-specific words such as "Steiner Tree." The theorem is proved just as in the TSP case: we start with an empty tiling, and refine it one level at a time, while ensuring the m-lightness property. Occasionally the special structure of the problem requires some care, as described below.

The algorithm for all the problems is similar to the TSP algorithm. First we use an analogue of Proposition 1 to perturb the instance. We compute, using existing algorithms, a $O(1)$-factor approximation A to the cost of the optimum. (Of course, even if those algorithms didn't exist, we could just "guess" the optimum in polynomial time using binary search.) Then we place a grid of granularity A/n^2 in the plane and assume that all nodes (including Steiner nodes) lie on the grid.

Now we discuss the problem-specific aspects.

Steiner Tree. The perturbation mentioned above ensures that each node and Steiner node is a grid point. This forestalls any precision issues, which would have otherwise arisen when the algorithm "guesses" the location of a Steiner node.

Now we sketch the proof of the Structure Theorem for Steiner Tree. As in the proof of Theorem 4, we start with an empty tiling and refine it one level at a time. Let W denote the size of the rectangle being refined currently and T denote the cost of the Steiner tree edges inside the rectangle. Again, there are two cases. In Case 1, $T \leq mW/3$. Then we use Theorem 5 to infer the existence of a line separator that is crossed by at most $3T/W \leq m$ segments of the Steiner tree. In Case 2, $T > mW/3$. Now we use a modified version of the Patching Lemma to modify the optimum subtree so that it crosses the separator at most twice. When we remove the crossing edges, the Steiner tree breaks into subtrees $P_1, P_2, \ldots$, (these were paths in the TSP case). The patching then has to connect them using line segments along the line separator (in the TSP case it had to not just connect them but also turn them into a single closed path). Therefore the patching argument is trivial.

The dynamic programming algorithm is more or less immediate from the Structure Theorem. The only twist is that when enumerating all choices for $O(m)$ external edges for a rectangle and their pairings, we have to allow all pairings that lead to a tree (in the TSP algorithm, these pairings had to lead to a closed path). The number of such pairings is $2^{O(m)}$ when we do not allow edge-crossings.

Finally, we observe something that may be of independent interest as a statement about the structure of a near-optimal Steiner tree. The tree returned by our algorithm uses only two types of Steiner nodes: portals, and the Steiner nodes added in the brute-force algorithm inside rectangles that contain at most $O(m)$ nodes (i.e., the "base-case" of the dynamic programming).

k-TSP: The only modification to the TSP methodology is in the dynamic programming. When we try all combinatorially distinct line separators of a rectangle, we also "guess" the number of nodes on the two sides that are visited by the tour. The number of such choices is at most k, so the running time gets multiplied by k.

k-MST: Similar to the k-TSP.

Degree-restricted-MST: The only subtlety is in maintaining the degree restriction in the argument that proves the Structure Theorem. The reason this is possible is that our actions in both Case 1 and Case 2 do not increase the degree of any node. (For Case 1, this fact is trivial. For Case 2, this has to be ensured using a careful patching argument.) Thus the degree of each node at the end of the refinement/tiling process is no larger than the degree at the start.

Furthermore, 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 $O(m)$ nodes. Thus the running time is $2^{O(m)}$poly(n) as before.

4 Conclusions

Our original goal was to extend the inapproximability results in [3] to Euclidean TSP. Doing this seemed to call for a reduction that produces instances in which every near-optimal tour encodes the complicated code-like objects of [3]. Then we realized (upon discovering an early version of Theorem 5) that such an encoding may be impossible. This realization led to an approximation algorithm in a natural way.

Our techniques apply to all Euclidean optimization problems in which the objective function is a sum of edge lengths (so that Theorem 5 can be applied) and which satisfy some version of the Patching Lemma 6. We don't have any formal characterization of this class of problems.

Two major open problems now remain: (i) Design an approximation algorithm better than Christofides' for general metric spaces. Some progress has been made by Karger and Klein [25], who extended our ideas to design a quasi-polynomial time approximation scheme for any metric that is the shortest-path metric of a weighted planar graph. (ii) Design a more efficient version of our algorithm. Now we discuss this in more detail.

Our algorithm's running time depends exponentially on $1/\epsilon$ even in $\Re^2$. Some kind of exponential dependence on $1/\epsilon$ seems unavoidable, since almost all the problems we consider are "strongly-NP-hard". But a running time of $2^{O(1/\epsilon)}$poly(n) may be achievable.

Can the running time in $\Re^d$ be improved to $n^{O(1/\epsilon)}$ or even $n^{O(d/\epsilon)}$? This question was posed in an earlier draft of this paper. Recently, Trevisan [41] has shown that the answer to these questions is probably "No." He shows that Euclidean TSP in $\Re^{O(\log n)}$ (n = number of nodes) is MAXSNP-hard under randomized reductions. Thus if the running time of our approximation scheme in $\Re^d$ could be improved to $n^{d/\epsilon}$ for each $d > 0$, then the running time for $d = O(\log n)$ would be $n^{O(\log n/\epsilon)}$. Using a well-known reduction of [3], such a scheme could be used to solve every NP problem probabilistically in $n^{O(\log n)}$ time, which contradicts a well-known conjecture about NP. Note, however, that Trevisan's result does not rule out a running time like poly$(n)2^{2^d/\epsilon}$. Interestingly, his reduction (owing to its dependence on [21]) also seems to break down for norms other than ℓ_2.

Finally, we mention that a few weeks after the circulation of the complete version of our paper[2], Mitchell [34] noticed that some ideas in his earlier paper [33] can be used to derive a PTAS for most of the problems we have considered. Like our PTAS, his PTAS also relies on a geometric partitioning scheme.

Acknowledgements

I thank Mic Grigni, David Johnson, Ravi Kannan, Haim Kaplan, Sanjeev Khanna, Dick Lipton, Rajeev Motwani, and Umesh Vazirani for useful conversations. Thanks to David Eppstein and Moses Charikar for pointing out various bugs/omissions in earlier drafts of this paper. Thanks to Warren Smith and Michel Goemans for suggesting the use of "portals" to simplify the description of the PTAS.

References

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

[2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. Preliminary manuscript (with "quasipolynomial" in the title), January 20, 1996. Final version, March 30, 1996. Circulated widely on April 10, 1996.

[3] 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.

[4] 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.

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

[6] M. Bern and D. Eppstein. Approximation Algorithms for Geometric Problems. In [16].

[7] 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.

[8] 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.

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

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

[11] 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.

[12] 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.

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

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

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

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

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

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

[19] 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.)

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

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

[22] 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.

[23] 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.

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

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

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

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

[28] 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.

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

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

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

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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).

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

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

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

[41] L. Trevisan. The n-dimensional Euclidean TSP is Max SNP-hard, Manuscript, May 1996. Available from trevisan@dsi.uniroma1.it.

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

Appendix

The following proposition was used in the analysis of the Patching Lemma for $\Re^d$.

Proposition 7 *The length of the optimum salesman tour on k nodes in $\Re^d$ that lie inside a rectangle of size L is at most $O(k^{1-1/d}L)$.*

Proof: We give a proof for $d = 2$; the proof for general d uses a simple induction (see Chapter 6 of [29]).

Partition the rectangle into horizontal "strips" of width $L/\sqrt{k}$. Then find a "strip tour," which first visits all the nodes in the first strip, then all the nodes in the second strip, and so on. If k_i is the number of nodes in the ith strip, then the cost of the strip tour is

$$\sum_i k_i \times O(\frac{L}{\sqrt{k}}) \qquad = k \times O(\frac{L}{\sqrt{k}}) \qquad = O(\sqrt{k}L).$$

$\square$

4.1 How Practical is our PTAS?

Many years of research and rapid increase in computer power have led to approaches — the best of which currently is branch-and-cut [1]— that in less than an hour finds *provably* optimal tours on instances with a few hundred nodes. Finding optimal tours on a few thousand nodes take a few computer-years, however.

We see some hope that our techniques, in combination with recent branch-and-cut implementations, could be used to find *provably* near-optimal tours on $5,000$ to $50,000$ nodes in reasonable amount of time. Below, we sketch our ideas. We also indicate holes in these ideas, where further work is required.

On an instance with $n = 25,000$ nodes, we would implement our partitioning scheme (namely, a $1/3 - 2/3$-tiling) not for $O(\log n)$ levels, but only for 7 levels. This would break the instance into $2^7 = 128$ instances of size $\approx 25,000/128 \approx 200$. If published numbers are to be believed, solving 128 instances of size ≈ 200 may take less than a day. (Note that these 128 are instances of the subtour problem and not the TSP. But adjustments to the edge weights can turn them instances of —non Euclidean–TSP.)

How can we do 7 levels of partitioning? We abandon our dynamic-programming approach for a simpler quicksort-like algorithm. At each step, instead of trying all combinatorially distinct line separators of a rectangle, the algorithm (a) picks a random line separator, (b) uses a "magic" heuristic —for more on this, see below — to find a set of m portals where the tour crosses this separator as well as the order in which these crossings occur, and (c) recurses on the two sides of the separator. If N is the time taken by the magic heuristic to find the m crossing points of the tour, the running time for 7 levels should be around $128N$.

It can be shown that with some reasonable probability, the partitioning works: there is a reasonable m for which some 1.01-approximate tour is m-light. (This calculation currently gives m that is a couple of thousand, but we believe it can be tightened.) Thus the only missing piece in the rosy scenario above is the "magic" heuristic. Several candidates come to mind. One heuristic is based upon the following intuition. The m places in which the tour crosses the line separator (and the order in which these crossings occur) should almost certainly be dictated by "global" structure and not by local structure. Therefore one should try the following "clustering". If W is the size of the bounding box, then partition the box into squares of side W/M for some number $M < m$ (for concreteness, say $M = \sqrt{m}$, although we are unable to show that this is good enough). and replace each nonempty box by a "supernode." This gives a reduced TSP instance with at most M^2 (= a few hundred?) supernodes, which we solve using branch-and-cut. Then we identify the places where the tour on this reduced instance crosses the line separator, and use those points as portals.

Of course, the ideas given above are very rough, and should be taken with a grain of salt. But we see reason for hope. A more careful theoretical analysis —with attention to constants — and a careful implementation are needed.