On the Approximability of the Traveling Salesman Problem with Line Neighborhoods

Antonios Antoniadis
University of Twente, Enschede, The Netherlands

Sándor Kisfaludi-Bak
Aalto University, Finland

Bundit Laekhanukit
Shanghai University of Finance and Economics, China

Daniel Vaz
Operations Research Group, Technische Universität München, Germany

Abstract

We study the variant of the Euclidean Traveling Salesman problem where instead of a set of points, we are given a set of lines as input, and the goal is to find the shortest tour that visits each line. The best known upper and lower bounds for the problem in $\mathbb{R}^d$, with $d \geq 3$, are NP-hardness and an $O(\log^3 n)$-approximation algorithm which is based on a reduction to the group Steiner tree problem.

We show that TSP with lines in $\mathbb{R}^d$ is APX-hard for any $d \geq 3$. More generally, this implies that TSP with k-dimensional flats does not admit a PTAS for any $1 \leq k \leq d-2$ unless P = NP, which gives a complete classification regarding the existence of polynomial time approximation schemes for these problems, as there are known PTASes for $k = 0$ (i.e., points) and $k = d-1$ (hyperplanes). We are able to give a stronger inapproximability factor for $d = O(\log n)$ by showing that TSP with lines does not admit a $(2-\varepsilon)$-approximation in d dimensions under the Unique Games Conjecture. On the positive side, we leverage recent results on restricted variants of the group Steiner tree problem in order to give an $O(\log^2 n)$-approximation algorithm for the problem, albeit with a running time of $n^{O(\log \log n)}$.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Approximation algorithms analysis

Keywords and phrases Traveling Salesman with neighborhoods, Group Steiner Tree, Geometric approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2022.10

Related Version The article has an earlier version available on ArXiv.
Full Version: `https://arxiv.org/abs/2008.12075`

Funding *Antonios Antoniadis*: Work done in part while the author was at Saarland University and Max Planck Institute for Informatics and supported by DFG grant AN 1262/1-1.
Daniel Vaz: This work has been supported by the Alexander von Humboldt Foundation with funds from the German Federal Ministry of Education and Research (BMBF). Work done in part while the author was at Saarland University and Max Planck Institute for Informatics.

1 Introduction

In the Euclidean Traveling Salesman problem, one is given n points in d-dimensional Euclidean space (denoted by $\mathbb{R}^d$), and the goal is to find the shortest tour visiting all the points. The problem is NP-hard for $d \geq 2$ [41], but it has a celebrated polynomial time approximation scheme (PTAS), i.e., a polynomial-time algorithm that produces a tour of length at most $1+\varepsilon$ times the optimum for any fixed $\varepsilon > 0$, due to Arora [3] and (independently) by Mitchell [38].

In the past decades, a considerable amount of work has concentrated on finding approximations for variants and generalizations of the Euclidean Traveling Salesman Problem, e.g., by changing the underlying space [4, 33, 16, 6], or the objects being visited [15, 7, 11, 20, 39, 40, 28]. In the latter case which is known as the *Traveling Salesman Problem with Neighborhoods (TSPN)*, the input consists of n *neighborhoods*, and the goal is to find the shortest tour that visits each neighborhood. More formally, we are given the sets $S_1, \ldots, S_n \subset \mathbb{R}^d$, and we wish to compute the shortest closed curve τ such that for each $i \in \{1, \ldots, n\}$ we have $S_i \cap \tau \neq \emptyset$. (Observe that the optimum curve τ consists of at most n segments.)

In contrast to regular TSP, TSPN is already APX-hard in the Euclidean plane [15], i.e., it has no PTAS unless P = NP. Worse still, even the basic case in which each neighborhood is an arbitrary finite set of points in the Euclidean plane (the so called *Group TSP*) admits no polynomial-time $O(1)$-approximation (unless P = NP) [43]. Even in the case in which each neighborhood consists of exactly two points [18] the problem remains APX-hard.

This inherent hardness of TSPN gives rise to studying variants of the problem in which the neighborhoods are restricted in some ways. In a seminal paper, Arkin and Hassin [2] looked into the problem for various cases of *bounded neighborhoods*, including translates of convex regions and parallel unit segments, and gave constant-factor approximation algorithms for them. The best known approximation algorithm for a more general case of bounded neighborhoods in the plane is due to Mata and Mitchell [35] and attains an $O(\log n)$ approximation factor. However, there exist special cases of such bounded neighborhoods in the plane that do allow for $O(1)$-approximation algorithms. These include neighborhoods which are disjoint, fat, or have comparable sizes [15, 7, 11, 20, 39, 40].

The complementary case of TSPN where neighborhoods are *unbounded* regions (the focus of this paper) is, in general, less well understood. Consider neighborhoods that are affine subspaces (*flats*) of dimension $k < d$ in $\mathbb{R}^d$. On the positive side, and despite the APX-hardness of the general TSPN problem already in $\mathbb{R}^2$, the version with flats (in this case lines) as neighborhoods can be solved exactly in $O(n^4 \log n)$-time via a reduction to the watchman route problem [29, 17]. Furthermore, Dumitrescu [19] provides a 1.28-approximation algorithm that runs in linear time. In $\mathbb{R}^3$, the problem of line and plane neighborhoods was first raised by Dumitrescu and Mitchell [20]. For the line case, they already point out that the problem is NP-hard as a direct consequence of the NP-hardness of Euclidean TSP in the plane [41]. Although this leaves the possibility for a PTAS open, the best known approximation algorithm to date for TSPN with lines in $\mathbb{R}^3$ was given by Dumitrescu and Tóth [21] and achieves an $O(\log^3 n)$-approximation. For the case of $(d-1)$-dimensional flats in $\mathbb{R}^d$ (which also includes planes in $\mathbb{R}^3$), they give a linear-time (for any constant dimension d and any constant $\varepsilon > 0$) $(1+\varepsilon)2^{d-1}/\sqrt{d}$-approximation. This result was subsequently improved by Antoniadis et al. [1] to an EPTAS that also runs in linear time for fixed d and ε. Whether this variant is NP-hard or not remains an interesting open problem. As for the case of line neighborhoods in $\mathbb{R}^d$ for $d \geq 3$, a PTAS for k-dimensional flats for $1 \leq k \leq d-2$ also remained out of reach.

We show that unless P = NP, there is no PTAS for lines in $\mathbb{R}^3$. As a direct consequence, we can rule out the existence of a PTAS in all remaining open cases of TSPN with flats: there is no PTAS for k-dimensional flat neighborhoods for any $1 \leq k \leq d-2$, unless P = NP.

Let us call the Euclidean TSP problem in $\mathbb{R}^d$ with k-dimensional flat neighborhoods (k,d)-TSPN. Although ruling out a PTAS for $(1,3)$-TSPN is an important step towards settling the approximability of the problem, the inapproximability factor obtained is very close to 1. It would be desirable to obtain a stronger inapproximability factor, especially given how far we are from any constant-approximation algorithm for the problem. A natural way to obtain such a stronger inapproximability result is to consider the problem in higher dimensional spaces. For example, regarding the classic Euclidean TSP, it is known that the problem becomes APX-hard for $d = \log n$ [44]. This result directly implies that TSPN with line neighborhoods in $\mathbb{R}^{1+\log n}$ is APX-hard, but this is barely satisfactory, since it again only gives a small inapproximability factor. However, by using a different reduction from the vertex cover problem, we are able to show that the problem has no polynomial $(2-\varepsilon)$-approximation in $\mathbb{R}^{O(\log n)}$ for any fixed $\varepsilon > 0$ under the Unique Games Conjecture [30].

On the algorithmic side, very little is known about (k, d)-TSPN. For $d = 3$, the best known polynomial time approximation for $(1, 3)$-TSPN is the aforementioned $O(\log^3 n)$-approximation algorithm due to Dumitrescu and Tóth [21]. Their approach is to discretize the problem by selecting a polynomial number of "relevant" points on each line. It is shown that restricting the solution to visiting lines at these points only increases the tour length by a constant factor. The resulting instance can now be seen as an instance of group-TSP, where the relevant points of each line form a group. By feeding this into the $O(\log^3 n)$-approximation algorithm for general group Steiner tree [25, 24] (it is easy to go from the tree solution to a tour by doubling each edge), they obtain the same asymptotic approximation factor for TSPN with line neighborhoods. This is somewhat unsatisfactory, since it ignores that the group Steiner tree instances constructed by the reduction are (i) Euclidean and (ii) all the points of a group are collinear. In other words, although the constructed group Steiner tree instances are highly restricted, there is no known technique to exploit this restriction.

However, the reduction from TSPN with line neighborhoods to the group Steiner tree problem implies that, if we allow quasi-polynomial running time, then TSPN with line neighborhoods admits an approximation ratio of $O(\log^2 n/\log\log n)$ in $O(n^{\log^2 n})$ time [13]. This approximation ratio of group Steiner tree is tight for the class of quasi-polynomial time algorithms due to the recent work of Grandoni et al. [26], which holds under the *Projection Game Conjecture* and $\mathrm{NP} \not\subseteq \bigcap_{0<\epsilon<1} \mathrm{ZPTIME}\big(2^{n^\epsilon}\big)$. Their hardness result is built on the seminal work of Halperin and Krauthgamer [27], who prove that group Steiner tree admits no $\log^{2-\epsilon} n$-approximation for any fixed $\epsilon > 0$, unless $\mathrm{NP} \subseteq \mathrm{ZTIME}\big(n^{\mathrm{polylog}(n)}\big)$.

For the class of polynomial-time approximation algorithms, the group Steiner tree problem admits an approximation ratio of $O(\log^2 n)$ on some special cases, e.g., trees [25] and bounded treewidth graphs [10, 9]. It is still open whether the group Steiner tree problem in general graphs admits a polynomial-time $O(\log^2 n)$-approximation algorithm; the best running time to obtain an $O(\log^2 n)$-approximation is $n^{O(\log n)}$ [13].

The connection between TSPN and group Steiner tree also holds in the reverse direction: Given an instance of group Steiner tree, one may embed the input metric into a Euclidean space with distortion $O(\log n)$ [8] and cast it as TSPN with "set neighborhoods".

While we cannot improve the approximation factor in polynomial time, we can do so in quasi-polynomial time: we give an $O(\log^2 n)$-approximation in $n^{O(\log\log n)}$ time. We obtain this result by using Arora's PTAS for TSP [3], together with the framework of Chalermsook et al. [10, 9], to transform TSPN into a variant of group Steiner tree when the input graph is a tree, and then employing an $O(\log^2 n)$-approximation algorithm.

Our Contribution. Our first contribution is to show that unlike the problem with hyperplane neighborhoods, the problem with line neighborhoods is APX-hard.

▶ **Theorem 1.** *The TSPN problem for lines in $\mathbb{R}^3$ is APX-hard. More specifically, it has no polynomial time $(1 + \frac{1}{230000})$-approximation unless* $\mathrm{P} = \mathrm{NP}$.

The reduction is from the vertex cover problem on tripartite graphs. The idea is to represent the graph edges with lines, where two lines intersect if and only if they correspond to incident edges. The main challenge is to keep the pairwise distance between non-intersecting lines large enough. We solve this by carefully placing the intersection points on non-adjacent edges of a cube. For technical reasons, we do not work directly with this placement, but rather on a "flattened" version of this point set. Additionally, we want to restrict the optimal tour so that it visits each line near one of its intersection points with other lines. This is achieved by forcing the optimal tour to follow a certain closed curve using special point gadgets (each consists of polynomially many lines), and to visit the lines representing the edges only at (or close to) intersection points. Visiting an intersection point corresponds to including the corresponding vertex in the vertex cover of the graph. As a direct consequence of Theorem 1, we obtain the following.

▶ **Corollary 2.** *The Euclidean TSP problem with k-dimensional flat neighborhoods in $\mathbb{R}^d$ is APX-hard for all $1 \leq k \leq d-2$.*

To prove Corollary 2, suppose we are given a set $\mathcal{L}$ of lines in $\mathbb{R}^3$. We can first change each line $\ell \in \mathcal{L}$ into the flat $\ell \times \mathbb{R}^{k-1}$, resulting in k-dimensional flats in $\mathbb{R}^{k+2}$. Since $k \leq d-2$, we have that $\mathbb{R}^{k+2}$ is a subspace of $\mathbb{R}^d$, so this is a valid construction for (k,d)-TSPN. Moreover, any tour in $\mathbb{R}^3$ visiting the lines is also a valid tour of the k-flats, and a valid tour of the k-flats can be projected into a valid tour of $\mathcal{L}$ in $\mathbb{R}^3$ of less or equal length.

Our second contribution is to show a larger inapproximability factor in higher dimensions under the Unique Games Conjecture:

▶ **Theorem 3.** *For any $\varepsilon > 0$, there exists a constant c such that there is no $(2-\varepsilon)$-approximation algorithm for TSPN with line neighborhoods in $\mathbb{R}^{c\cdot\log n}$, unless the Unique Games Conjecture is false. Moreover, for any $\varepsilon > 0$, there is a constant c such that it is* NP*-hard to give a $(\sqrt{2}-\varepsilon)$-approximation for TSPN with line neighborhoods in $\mathbb{R}^{c\cdot\log n}$.*

This reduction is from the general vertex cover problem. Again we represent the edges of the graph with lines and the vertices correspond to intersection points. This time however the intersection points are almost equidistant: they are obtained via the Johnson-Lindenstrauss lemma applied on an n-simplex. This allows the tour to visit the intersection points in any order. To obtain a direct correspondence with vertex cover, we need to ensure that lines are visited near intersection points. To this end, we blow up the underlying graph by replacing each edge by a complete bipartite graph. Thus, we get the following corollary of Theorem 3.

▶ **Corollary 4.** *For any $\varepsilon > 0$ there is a number $c = c(\varepsilon)$ such that the Euclidean TSP problem with k-dimensional flat neighborhoods in $\mathbb{R}^d$ has no polynomial $(2-\varepsilon)$-approximation for any $k \in \{1, \ldots, d - c\log n\}$, unless the Unique Games Conjecture is false.*

On the positive side, our third contribution is to develop an $O(\log^2 n)$-approximation algorithm with slightly superpolynomial running time.

▶ **Theorem 5.** *There is a deterministic $O(\log^2 n)$-approximation algorithm for TSPN with line neighborhoods in $\mathbb{R}^d$ that runs in time $n^{O(\log\log n)}$ for any fixed dimension d.*

The algorithm is based on adapting the dynamic program by Arora [3], and reformulating TSPN into the problem of finding a solution in the dynamic programming space that visits all the line neighborhoods. We then build upon the techniques of Chalermsook et al. [10, 9], and show that this task can be reduced to a variant of the group Steiner tree problem that admits an $O(\log^2 n)$-approximation in slightly superpolynomial running time. The $O(\log\log n)$-factor in the exponent of the running time is a consequence of the running time of Arora's algorithm, and it is possible that we can improve it to polynomial time if an appropriate EPTAS for TSP with running time $O(f(\varepsilon,d)n\log n)$ is discovered.

Due to space constraints, missing proofs are deferred to the full version of the paper.

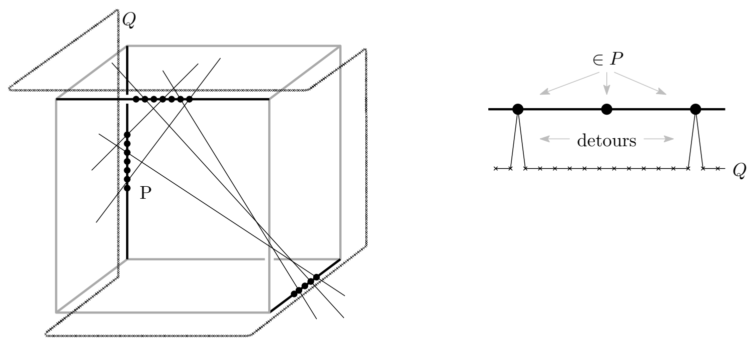

Figure 1 Left: Overview of a basic construction with a cube. Right: The optimal tour must visit all points of Q, and it makes detours to some points p_v.

2 Inapproximability in 3 dimensions

The goal of this section is to prove Theorem 1. The overall setup of our construction is lightly inspired by a reduction in Elbassioni et al. [22]. The reduction is from vertex cover on 3-partite graphs (i.e., on graphs G where the vertices can be partitioned into three independent sets V_1, V_2 and V_3). It is NP-hard to decide whether a given instance has a vertex cover of size $n/2$ or if all vertex covers have size at least $\frac{34}{33}\frac{n}{2}$ [14]. In our construction, each vertex v of G is assigned to a point p_v on an edge of a unit cube; the classes V_1, V_2, V_3 are mapped to pairwise non-adjacent and non-parallel (i.e., skew) cube edges. For each edge $uv \in E(G)$, we add the line $p_u p_v$; see Fig. 1.

Consider now a closed curve γ of length 10 which is disjoint from the cube, but follows some edges of the cube at a distance c/n for some constant c. Let Q be a set of points along γ such that any two consecutive points have distance $c/(10n)$.

We define a special *point gadget* – which consists of a large collection of lines – at each point $q \in Q$. This ensures that any TSP tour that has length at most 20 will touch an infinitesimally small ball around each vertex of Q. Consequently, any not too long TSP tour will have to "trace" γ. The points in $P = \cup p_v$ which are placed near the cube edges are arranged so that one can visit each point p_v with a short detour from γ of length c/n. Given a vertex cover of size k in G one can create a TSP tour of length at most $10 + kc/n$, namely by folowing γ and making the short detour at p_v if and only if v is in the vertex cover. Conversely, by a careful arrangement of the lines and point gadgets, we can ensure that a tour of length $10 + kc/n$ implies the existence of a vertex cover of size at most $1.011k$.

For technical reasons, we need to transform the constructed cube to a very flat parallelepiped; it is convenient to define the point set Q and the point gadgets only after this flattening transformation takes place. We are now ready to define our construction.

2.1 The construction

Let $G = (V, E)$ be a tripartite graph on n vertices with partition classes V_1, V_2, V_3. We add dummy vertices (without any incident edges) to G so that each class has n vertices; the vertices of V_a ($a = 1, 2, 3$) are denoted by $v_1^a, \ldots, v_n^a$. Notice that the addition of dummy vertices does not change the set of vertex covers of G. Let $\mathcal{C}$ denote the unit cube $[0,1]^3$, and let e^1, e^2, e^3 be the unit segments $(0,0,1)^T(1,0,1)^T$, $(1,0,0)^T(1,1,0)^T$ and $(0,1,0)^T(0,1,1)^T$ respectively. We assign each vertex v_i^a to a point on the middle third of e^i. The assignment is denoted by p, and defined as:

$$p(v_i^a) = \begin{cases} (\frac{n+i}{3n}, 0, 1)^T & \text{if } a = 1 \\ (1, \frac{n+i}{3n}, 0)^T & \text{if } a = 2 \\ (0, 1, \frac{n+i}{3n})^T & \text{if } a = 3. \end{cases}$$

We denote by $P = p(V(G))$ the set of points created this way. For each edge $uv \in E(G)$, let $\ell(uv)$ be the line through $p(u)$ and $p(v)$, and let $\mathcal{L}$ be the set of lines created this way: $\mathcal{L} = \{\ell(uv) \mid uv \in E(G)\}$. The following technical lemma plays a key role in the contruction.

▶ **Lemma 6.** *If $\ell, \ell' \in \mathcal{L}$ correspond to non-incident edges, then they are disjoint and the distance between them is at least $\frac{1}{20n}$.*

Flattening. We must ensure that the points of Q near a cube edge are similarly distant from the lines in $\mathcal{L}$ incident to the cube edge (we will do this in Claim 9). We achieve this by transforming the above construction so that the angle of each line ℓ with the plane $x + y + z = 0$ is at most some small constant. Practically, we transform the point set P and the line set $\mathcal{L}$ with the linear transformation $x \mapsto Ax$, where $A = I - 0.3J$ and J is the all-ones matrix.

Essentially, the transformation pushes everything closer to the plane $H : x + y + z = 0$: for a given point p and its perpendicular projection q on H, the point Ap is the point on the segment pq for which $\text{dist}(q, Ap) = \frac{1}{10}\text{dist}(q, p)$. Note that if pq is any segment of length λ, then its length after the transformation is at least $\lambda/10$ and at most λ. When the transformation is applied to an edge e^a of the cube $\mathcal{C}$, then the resulting segment has length $\sigma = \sqrt{0.7^2 + 0.3^2 + 0.3^2} \simeq 0.8185$. Consequently, $Ap(v_i^a)$ and $Ap(v_{i+1}^a)$ has distance $\sigma/(3n)$.

Let $\bar{P}$ and $\bar{\mathcal{L}}$ be the resulting point set and line set. Using Lemma 6 and the above arguments we get the following corollary.

▶ **Corollary 7.** *The minimum distance between points of $\bar{P}$ is $\frac{\sigma}{3n}$, and the minimum distance between lines of $\bar{\mathcal{L}}$ corresponding to non-incident edges of G is at least $\frac{1}{200n}$.*

Defining the point gadgets, and wrapping up the construction. For a point set X, let $\bar{X}$ denote its image under the flattening transformation A. Let F_1^a and F_2^a be the planes of the faces of $[0,1]^3$ incident to e^a. The following claim shows that $\bar{F}_1^a$ and $\bar{F}_2^a$ are two planes through $\bar{e}^a$ whose angle is small.

▷ Claim 8. For $a = 1, 2, 3$ we have $\sphericalangle(\bar{F}_1^a, \bar{F}_2^a) < \frac{1}{4}$.

Let H^a be the angle bisector plane of $\bar{F}_1^a$ and $\bar{F}_2^a$ which does not intersect the image of $\mathcal{C}$, see Figure 2(i). Within H^a, we place a set of points Q^a, which we define next.

Let $\delta = \frac{1}{4000n}$, and let δ^* be the height of the isoceles triangle T_δ with base δ and two sides of length 10δ, that is $\delta^* = \sqrt{99.75}\delta$. Consider a half-plane in H_a whose boundary is parallel to $\bar{e}^a$ and is at distance δ^* from it. Within this half-plane, let Q_a be a set of at most $4000n$ points with the following properties: (i) for each $p(v_i^a)$ there are two points $q, q' \in Q^a$ such that $p(v_i^a)$, q and q' form an isoceles triangle of side lengths $10\delta, 10\delta, \delta$ and (ii) there is a unique shortest TSP path of Q_a, whose edges are of length exactly δ; see Figure 2(ii).

Let Q be a point set with the following properties:

- $Q^1 \cup Q^2 \cup Q^3 \subseteq Q$
- For any pair of distinct points $q, q' \in Q$, $\text{dist}(q, q') \geq \delta$.
- Each segment of the minimum TSP tour $T(Q)$ of Q has length δ, and $cost(T(Q)) = 10$.
- The minimum distance of points of Q from $\bar{\mathcal{L}}$ is attained only in $Q^1 \cup Q^2 \cup Q^3$
- Q is disjoint from the cylinder $\mathcal{Y}$ of axis $(0,0,0)^T(1,1,1)^T$ and radius $\sigma/2$.

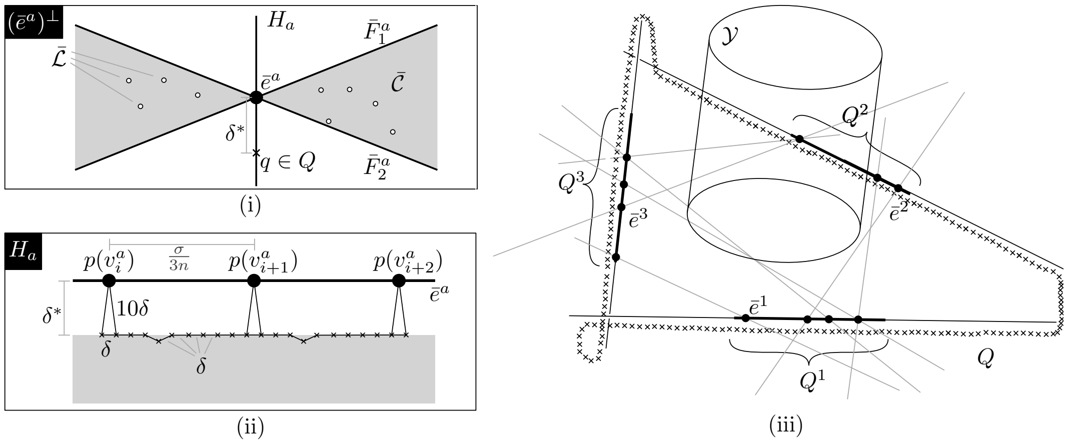

Figure 2 (i) Cross-section given by a plane perpendicular to $\bar{e}^a$. (The segment $\bar{e}^a$ appears as a point, and the plane H_a as a line in this picture.) All lines of $\bar{\mathcal{L}}$ intersect such a plane in the gray area. (ii) Defining Q^a within the plane H_a, so that all points have distance at least δ^* from $\bar{e}^a$. (iii) Defining Q so that it has all the required properties. The cylinder $\mathcal{Y}$ is perpendicular to the plane $x + y + z = 0$, a plane to which all points of the construction are close to. The "triangle" defined by the skew lines $\bar{e}^a$ wraps around the cylinder $\mathcal{Y}$.

Such a set Q is easy to find, for example by following the lines $\bar{e}^a$ and connecting them far from the origin. See Fig. 2(iii) for an illustration.

We need the following claim on the distance of Q from the lines in $\bar{\mathcal{L}}$. Intuitively, it shows that the points in Q are far from the lines in $\bar{\mathcal{L}}$, and thus a certain detour is necessary to visit a line in $\bar{\mathcal{L}}$. Note that the bound would not be strong enough without the flattening.

▷ Claim 9. For any $q \in Q$ and $\ell \in \bar{\mathcal{L}}$ we have $\operatorname{dist}(q, \ell) > 9.9\delta$.

▶ **Lemma 10** (Point gadget). *Given a positive integer n and a point $q \in \mathbb{R}^3$, there is a set $\mathcal{L}$ of $O(n^6)$ lines through q such that any TSPN tour of $\mathcal{L}$ which is disjoint from the ball $B(q, \frac{1}{n^3})$ has length at least 20.*

Our construction is the union of the line set $\bar{\mathcal{L}}$ together with a point gadget placed at each point $q \in Q$; let $\mathcal{L}^*$ denote the resulting line set.

2.2 Putting things together

▶ **Lemma 11.** *If G has a vertex cover of size k, then there is a tour in $\mathcal{L}^*$ of length $10 + 19\delta k$. If $\mathcal{L}^*$ has a tour of length $10 + 19\delta k$, then G has a vertex cover of size $1.011k$.*

The proof of the first part of the lemma is straightforward. To prove the second claim, we use the fact that the tour must touch the small balls $B(q, \frac{1}{n^3})$ for each point $q \in Q$ by Lemma 10. We can then consider a portion of the tour between two consecutive ball visits, i.e., a polygonal curve g that starts near some point $q \in Q$ and ends near some other point $q' \in Q$, and visits some of the lines in $\bar{\mathcal{L}}$ along the way. In the full version we show that g cannot touch lines from all three classes, in other words there is a segment $\bar{e}^i$ such that all lines visited by g have an endpoint on $\bar{e}^i$. The proof relies on the property that Q avoids the cylinder $\mathcal{Y}$ with axis $(0,0,0)^T, (1,1,1)^T$ and radius $\sigma/2$. Intuitively, if g would touch lines from all three classes, then it would have to go around the cylinder partially, which would be too costly. We can then define a vertex cover based on the tour portions g: for each line ℓ visited by g, the line ℓ has a point on $\bar{e}^i$ that corresponds to some vertex v of the graph. These vertices v form a set W which is clearly a vertex cover; the goal is then to prove that $|W| \leq 1.011k$. The proof hinges on the fact that if a tour portion g contributes s unique vertices to W, then it must jump between non-incident lines of $\bar{\mathcal{L}}$ at least $(s-1)$ times, which incurs a cost of at least $20(s-1)\delta$ by Corollary 7. In case of $s = 1$, the tour still needs to visit *some* line in $\bar{\mathcal{L}}$, which incurs a cost of at least 19.8δ by Claim 9. Putting these observations together (and that the minimum cost tour of the balls $B(q, 1/n^3)$ has length very close to 10) yields the desired bound on $|W|$.

Proof of Theorem 1. Suppose that there is a polynomial time algorithm that approximates TSPN with lines in $\mathbb{R}^3$ within a factor of $1 + \frac{1}{230000}$. Let G be a given 3-partite graph. If G has a vertex cover of size $n/2$, then the above construction would have a tour of length $10 + 19\delta\frac{n}{2} = 10.002375$. On the other hand, if all vertex covers of G have size at least $\frac{34}{33}\frac{n}{2}$, then all tours of the construction have length at least $10 + 19\delta\frac{34}{33}\frac{n}{2 \cdot 1.011} > 10.00242$. As $10.00242/10.002375 > 1 + \frac{1}{230000}$, we could use the hypothetical approximation algorithm to distinguish between these two cases in polynomial time, which would imply P = NP. ◀

3 No $(2 - \epsilon)$-approximation Algorithm

In this section we prove Theorem 3. In particular, we will show that when the objects are lines, TSPN is at least as hard to approximate as the Vertex Cover problem which is known to be hard to approximate to within a factor of $2 - \varepsilon$, for any constant $\varepsilon > 0$, under the Unique Games Conjecture (and inapproximable within a factor of 1.42 unless P = NP [31]).

▶ **Theorem 12** ([32]). *Unless the Unique Games Conjecture is false, for any constant $\varepsilon > 0$, there is no polynomial-time algorithm that, given a graph $G = (V, E)$ and an integer k, distinguishes between the cases (i) G has a vertex cover of size at most k or (ii) G has no vertex cover of size less than $(2 - \varepsilon)k$.*

The main idea behind the reduction is to represent a graph G in Euclidean space such that:
- Each vertex $v \in V(G)$ corresponds to a point $p_v \in \mathbb{R}^d$,
- Each edge $e = uv \in E(G)$ corresponds to a line going through the points p_u and p_v,
- An optimal tour visits each line sufficiently close to the points p_v, and therefore the vertex set corresponding to the points in the vicinity of the tour is a vertex cover.

However, in order to enforce that an optimal tour passes through (or not too far from) the points p_v, we will have to further build upon this idea. In particular, for each vertex v, instead of constructing only one point p_v, we will construct a set P_v of polynomially many points corresponding to v. If there is an edge $e = uw \in E(G)$, then we connect each point corresponding to u with each point corresponding to w. More precisely, for each edge uv and for every pair of points (p_u, p_w) with $p_u \in P_u$ and $p_w \in P_w$, we add a line going through p_u and p_w. Notice that the number of edges increases quadratically in the number of vertex copies. Therefore, tours that visit lines away from the vertices are disproportionally affected, which forces an optimal tour to visit lines at (or close to) the points in P_v.

Another key aspect of our construction is that we position the points of $\mathcal{P} := \bigcup_{v \in V(G)} P_v$ in $\mathbb{R}^d$ so that the distance between any pair of distinct points is (roughly) the same. This helps us to have a more direct correspondence between the cost of the optimal tour and the size of an optimal vertex cover. The reduction is desribed formally in the next subsection.

3.1 Reduction: Vertex Cover to TSP with Line Neighborhoods

Take an instance of the Vertex Cover problem on a graph $G = (V, E)$ with n vertices and m edges. We first take a *lexicographic product* of the graph G with an independent set of size $\alpha = n^2$. Informally speaking, we construct a graph G' by making α copies of each vertex $v \in V(G)$, and denote the corresponding vertex set by Q_v. Then, for each edge $vw \in E(G)$, we add edges between every pair of vertices $v^i \in Q_v$ and $w^j \in Q_w$, thus forming a complete bipartite graph on Q_v and Q_w. More formally, the graph G' is defined as:

$$V(G') = \{v^i : v \in V(G) \wedge i \in [\alpha]\} \quad \text{and} \quad E(G') = \{v^i w^j : vw \in E(G) \wedge v \neq w \wedge i, j \in [\alpha]\}.$$

Next, we use the graph G' to construct an instance $I(G')$ of the TSPN with line neighborhoods problem in $d = O(\delta^{-2} \ln n')$ dimensions for any small enough $\delta > 0$ and with $n' = |V(G')| = \alpha \cdot n$. We map each vertex v of G' to a point p_v in $\mathbb{R}^d$ such that for any two points p_v and p_u with $v \neq u, v, u \in V(G')$ the distance $\text{dist}(p_v, p_u)$ between them satisfies the following property: $1 \leq \text{dist}(p_v, p_u) \leq 1 + \delta$.

The fact that this is possible and can be done in polynomial time follows by Theorem 3.1 by Engebretsen, Indyk and O'Donnell [23], which we restate for completeness in the Appendix as Theorem 19. In particular, we can employ the theorem in order to deterministically map a unit side length simplex from $\mathbb{R}^{n'-1}$ to $\mathbb{R}^d$ such that the desired property holds for all pairs of points.

We denote the resulting point set by P. Next, we create a collection of lines $\mathcal{L}$ in an instance of TSPN, by adding to $\mathcal{L}$ a line ℓ_{vw} passing through points p_v and p_w if $vw \in E(G')$.

We devote the rest of this section to prove completeness and soundness of our reduction.

Completeness. Suppose the graph G has a vertex cover of size $\leq k$. Then we claim that there is a tour T of cost at most $\alpha k(1+\delta)$ that touches each line at least once. To see this, let $S = \{v_1, \dots, v_k\}$ denote the vertex cover of G. By construction, $S' = \{v_j^i : i \in [\alpha] \wedge j \in [k]\}$ is a vertex cover of G'. By the construction of $\mathcal{L}$ and by the fact that S' is a vertex cover of G', it follows that any tour that visits points $p_{v_1^1}, p_{v_1^2}, \dots p_{v_k^\alpha}$ (in any order) is a feasible tour, i.e., it touches all lines in $\mathcal{L}$. So, in total such a tour visits a total of at most αk points, and the distance between any pair of these points is by construction at most $1 + \delta$. Thus, there is a solution to TSPN with cost at most $\alpha k(1 + \delta)$.

Soundness. We show that if there is a tour of cost x (where $x \leq \alpha n(1 + \delta)$), then there is a vertex cover in G of size at most $\frac{x}{\alpha(1-2\Delta)\lambda}$, where Δ is a small positive number and $\lambda \in [0, 1]$ is very close to 1.

The intuition behind Δ is that it describes the maximum distance that the tour is allowed to have to a given point, assuming that the vertex corresponding to that point contributes to the vertex cover. For each point $p_{v^i} \in P$ (note that $v^i \in V(G')$), let $B(v^i)$ be a d-dimensional ball of radius Δ centered at p_{v^i}. Note that Δ is small enough so the only lines from $\mathcal{L}$ intersecting a ball $B(v^i)$ are the ones that go through p_{v^i}. Given a tour T, we say that a ball $B(v^i)$ is *non-empty* if $T \cap B(v^i) \neq \emptyset$; otherwise, we say that $B(v^i)$ is *empty*. We say that a line ℓ_{uw} is *covered by a ball* if at least one of the balls $B(u)$ and $B(w)$ is non-empty. Otherwise ℓ_{uw} is *not covered by a ball*. We first show that any point $p \in \ell_{uw}$ that is outside the two balls corresponding to u and w will not be "too close" to any other line:

▶ **Lemma 13.** *For any point $p \in \ell_{uw}$ such that $p \notin B(u)$ and $p \notin B(w)$ and for any $\ell \in \mathcal{L} \setminus \{\ell_{uw}\}$ we have $dist(p, \ell) \geq \Delta/2$.*

We are now ready to prove that any optimal tour T must cover almost all lines by balls:

▶ **Lemma 14.** *Let T be a tour of cost at most x with $x \leq \alpha n(1+\delta)$ for the instance $I(G')$. Then the number of lines of $I(G')$ that are not covered by balls is at most $\frac{2x}{\Delta}$.*

Proof. Note it is without loss of generality to assume that T consists of line segments with endpoints on lines of $I(G')$. By Lemma 13 any line $\ell_{u^i w^j} \in \mathcal{L}_{uw}$ that is visited at a point p with $p \notin B(u^i)$ and $p \notin B(w^j)$, must have two adjacent segments on T of length at least $\Delta/2$ each. Since the total tour cost of T is at most x there can be at most $\frac{x}{\Delta/2} = \frac{2x}{\Delta}$ lines that are visited by T outside a ball. ◀

Let $\lambda = 1 - \varepsilon^2$ and set $\alpha = n^2$. We can construct a vertex cover of G based on a tour T the following way: if a set Q_v has at least $\lambda\alpha$ non-empty balls, then we add v to the vertex cover.

▶ **Lemma 15.** *The set $S = \{v : |\cup_{i\in[a]}\{v^i : B(v^i) \text{ is non-empty}\}| \geq \lambda\alpha\}$ is a vertex cover of G of size $|S| < \frac{x}{\alpha(1-2\Delta)\lambda}$.*

Proof. We first argue that S is a vertex cover of G. Assume for the sake of contradiction that some edge $uv \in E(G)$ is not covered by S. Then it must be the case that there are at least $\alpha(1-\lambda)$ empty balls among the balls corresponding to both u and v. But any line defined by two such empty balls corresponding to u and v is not covered by a ball. In total there are more than $(1-\lambda)^2\alpha^2 = \Omega(n^4)$ many such lines. This is a contradiction, since by Lemma 14 there can be at most $\frac{2n\alpha k(1+\delta)}{\Delta} = O(n^3)$ such lines in total over the whole instance.

Let S be the vertex cover of G we have obtained. Since the dsitance of any two balls is at least $1-2\Delta$, and we have visited at least $\alpha\lambda$ balls among Q_v for each $v \in S$, the total cost of the tour is at least

$$x > |S|\alpha\lambda(1-2\Delta),$$

therefore we have that $|S| < \frac{x}{\alpha\lambda(1-2\Delta)}$. ◀

Proof of Theorem 3. Suppose that there is an algorithm that can distinguish in polynomial time, for any $0 < \varepsilon'$ and any $x \in \mathbb{R}_+$, whether there is a tour of length at most x or all tours have length at least $(2-\varepsilon')x$. Take some instance of vertex cover, where the goal is to decide if there is a vertex cover of size at most k or all vertex covers of the graph have size at least $(2-\varepsilon)k$, where $\varepsilon \in (0, 0.1]$. By the above polynomial construction, it would be sufficient to distinguish the cases where $\mathcal{I}(G')$ has a tour of size at most $k\alpha(1-2\Delta)\lambda$ (implying a vertex cover of size at most k), or all tours have length at least $(2-\varepsilon)k\alpha(1+\delta)$ (implying that all vertex covers have size at least $(2-\varepsilon)k$). If we set $\delta = \Delta = \varepsilon^2$, then we get that the ratio of these tours is:

$$\frac{(2-\varepsilon)k\alpha(1+\delta)}{k\alpha(1-2\Delta)\lambda} = \frac{(2-\varepsilon)(1+\varepsilon^2)}{(1-2\varepsilon^2)(1-\varepsilon^2)} < 2,$$

so the hypothetical algorithm on $I(G')$ distinguishes these cases, which is a contradiction. ◀

We note that our reduction implies that TSPN with Line Neighborhoods is Vertex Cover hard, and therefore also inapproximable within a factor of $\sqrt{2} - \varepsilon$ unless $P = NP$ [31].

4 A Quasipolynomial-Time Approximation Algorithm

In this section, we will show a quasi-polynomial time algorithm to approximate TSPN for lines to a factor of $O(\log^2 n)$. In fact, our approach is more general: we show how to $O(\log N \log n)$-approximate TSPN for discrete neighborhoods of total size N, in running time

$N^{O(\log\log N)}$ for any fixed d. In this problem, we are given n neighborhoods $P_i \subset \mathbb{R}^d$, which are discrete sets of points. Let $P = \bigcup_{i\in[n]} P_i$, $N = |P|$. Using the approach of Dumitrescu and Tóth [21], we can convert any instance of TSPN with line neighborhoods into an instance of discrete TSPN on a set of $N = O(n^4)$ points and n neighborhoods. This transformation has a running time of $O(N)$, and incurs the loss of a constant factor in the approximation. From now on, we focus on TSPN for discrete neighborhoods.

Our main result is an $O(\log N \log n)$-approximation algorithm that runs in time $N^{O(\log\log N)}$ for constant d. Our algorithm combines the dynamic program by Arora [3] with the framework of Chalermsook et al. [10, 9]. As Dumitrescu and Tóth show [21], TSPN is related to the group Steiner tree problem, and can be reduced to this problem to obtain an $O(\log^3 n)$-approximation. We show that, using the structure of the Euclidean space, which is exploited in the algorithm presented by Arora for TSP, we can use the techniques of Chalermsook et al. to approximate discrete instances of TSPN and group Steiner tree in $\mathbb{R}^d$.

We note that, even on tree metrics, the group Steiner tree problem is $\Omega(\log^2 n/\log\log n)$-hard to approximate under the projection games conjecture [27, 26]. As every tree metric can be embedded into some Euclidean space with distortion $O(\sqrt{\log\log n})$ [34, 36, 37], the group Steiner tree problem in Euclidean space is also hard to approximate to within $\Omega(\log^2 n/(\log\log n)^{3/2})$ under the same assumption.

▶ **Theorem 16.** *There is a randomized $O(\log N \log n)$-approximation algorithm for TSPN with discrete neighborhoods in $\mathbb{R}^d$ that runs in time $N^{O(\log\log N)}$ for constant d.*

The theorem above, together with the result of Dumitrescu and Tóth [21] imply Theorem 5, along with the derandomization techniques of Arora [3] and Charikar et al. [12].

We start by recalling the main steps of the PTAS for TSP by Arora, as our result builds upon the dynamic program used there. While describing the algorithm, we state some modifications that are necessary for our purpose. Then, we show how to use the framework of Chalermsook et al. [10, 9] to find a feasible solution using the dynamic program. We note that, among different results on Euclidean TSP, the work of Arora yields the best running time for our algorithm as the use of other techniques, e.g., spanners [42, 5], to improve the running time is not compatible with our approach. Nevertheless, an existence of a "pure" dynamic program for Euclidean TSP that has constant number of branches in each recursive call will immediately lead to a polynomial-time algorithm without further modification.

4.1 Chalermsook et al.'s Framework

Firstly, we discuss the technique of Chalermsook et al. [10, 9] that reduces an instance of the group Steiner tree problem on bounded treewidth graphs into a tree-instance. In fact, the algorithm of Chalermsook et al. works on arbitrary graphs, but the size of the tree produced depends on the running time of a dynamic program or any recursive algorithm that solves the corresponding point-to-point problem, i.e., the classical Steiner tree problem. Specifically, the resulting tree will have a polynomial-size only if the recursive algorithm terminates in polynomial-time. Indeed, each node in the tree corresponds to a state in the search space of the dynamic program. In our case, while a polynomial-time algorithm is not available for Euclidean TSP (due to its NP-hardness), there exists a pure dynamic-programming based PTAS by Arora [3], which suffices for reducing the Euclidean TSP instance into a tree-instance of the group Steiner tree problem, albeit losing a constant factor in the approximation ratio.

Now we wish to extract a near-optimal solution of the original problem from the tree-instance, which appears as a subtree. This is relatively straightforward for the point-to-point network design problems like the classical Steiner tree problem and Euclidean TSP. The neighborhood problem (i.e., TSPN), on the other hand, has multiple points where we can enter a neighborhood, and each choice affects the solution globally. Chalermsook et al. solves this issue by exploiting linear programs to find an (fractional) optimal solution that satisfies the global constraints. Henceforth, we are left with rounding a fractional solution obtained from the LP, and this problem reduces to solving a group Steiner tree problem on the tree-instance, which admits an $O(\log N \log n)$-approximation algorithm.

Lastly, we remark that, while Chalermsook's algorithm runs in polynomial-time, our algorithm for TSPN runs in slightly super polynomial-time even in constant dimension. This is because each recursive call in Arora's algorithm may branch up to $O(\log n)^{2^{O(d \log d)}}$ sub-processes, thus causing the transformation to produce a tree of size $O(n^{2^{O(d \log d)}} \log \log n)$.

The remaining part of this section is devoted to give more details on our algorithm. The full discussion and proofs are deferred to an appendix due to page limit.

4.2 Arora's Algorithm

In this section, we briefly summarize the algorithm of Arora [3] (a more detailed description can be found in Appendix B.1). This algorithm approximates TSP to a factor of $1 + 1/c$; for our purpose, it is sufficient to consider $c = 1$. Arora's algorithm has three main steps:

1. Perturbation, which ensures that all coordinates are integral and bounded by $O(n)$;
2. Construction of a shifted quadtree;
3. Dynamic program, which finds the approximate solution for TSP.

The dynamic program is based on the (m, r)-*multipath problem* (see Definition 21), which given a cell of the quadtree and a set of pairs of *portals* on the boundary of the cell, has as its objective to find a minimum-cost set of paths, each connecting a pair of portals, and such that all of the points in the cell are visited. We refer to the multiset of portals and their pairing as the *state* of an (m, r)-multipath problem.

Two main changes are required to use the algorithm by Arora to approximate TSPN. First, we must guess a point v_0 in an optimum solution, as well as a value $R = O(\mathrm{OPT})$ (see Appendix B.1.1). Second, we must allow the solutions to the (m, r)-multipath problem to not visit every point in the cell. We achieve this by adding a *visit bit* to the state of the (m, r)-multipath problem on leaves, which indicates if the (unique) point in the cell must be visited (it is True), or it is sufficient to connect the portals (see Appendix B.1.3).

4.3 Approximating TSPN using the framework by Chalermsook et al.

After perturbation and construction of the shifted quadtree, we use the dynamic program above to define a *dynamic programming graph*. The intuition is that a solution to the problem can be represented as a tree in this graph, where the vertices in the tree correspond to all of the (m, r)-multipath problems that assemble into the solution.

We now describe the nodes and edges of this graph, denoted by H.

- **Nodes:** There are two types of nodes, which we refer to as *subproblem nodes* and *combination nodes*. The graph contains one subproblem node for every entry of the modified dynamic programming table in Section 4.2, i.e. one node for each instance of the (m, r)-multipath problem for every cell, pairing of portals and visit bit (for leaves). Combination nodes correspond to the possibilities of recursion for a given subproblem: for a given (m, r)-multipath problem (for a non-leaf cell), there is a combination node for every possible way for the p paths to cross the boundary between children cells.
- **Root:** The root of H corresponds to (m, r)-multipath on the root cell with no portals.

- **Edges:** There are (directed) edges connecting the node for each (m, r)-multipath problem to the corresponding combination nodes, and then the combination nodes to the corresponding nodes for the subproblems in the children cells.
- **Costs:** Edges incident to leaf nodes have cost equal to the corresponding entry in the dynamic programming table; all other edges have cost 0.

Using this definition, we can represent any (m, r)-light salesman path, that is a path which crosses each facet of each cell at most r times and always at a portal, as a tree T in H. For each cell, the solution restricted to that cell consists of a union of disjoint paths, which induce a set of portals and their pairing, and hence an instance of the (m, r)-multipath problem. We include the corresponding subproblem node in T. For each non-leaf cell, there is a combination node which represents the way in which the paths cross boundaries between children cells. We add that combination node to T, as well as all of the edges containing it.

The trees obtained by this process have a specific structure, which was implicitly formulated in the work of Chalermsook et al. [10, 9], and which we formalize below.

▶ **Definition 17** (Solution tree). *Let H be a DAG with root r, and its nodes be partitioned into* combination nodes *H_c, and* subproblem nodes *H_p. We say an out-arborescence $T \subseteq H$ rooted at r is a* solution tree *if:*

1. *Every combination node $t^c \in T \cap H_c$ has full out-degree (i.e., all children are also in T),*
2. *Every non-leaf subproblem node $t \in T \cap H_p$ (including the root r) has out-degree 1 in T.*

As we mentioned above, we can construct a solution tree for any (m, r)-light salesman path. The converse is also true: for each solution tree, there is a corresponding (m, r)-light salesman path. The final requirement for a solution to be feasible is that each neighborhood must be covered, meaning that the tour must intersect every neighborhood.

Consider a tour corresponding to a solution tree in T. The set of points visited by this tour is exactly the set of points contained in the leaf cells for subproblem nodes whose visit bit is set to True. In other words, the points of the leaf subproblems with visit bit True are visited by the tour, and so the corresponding neighborhoods are covered. Therefore, we can solve TSPN by formulating it as finding a solution tree that covers every neighborhood.

Let S_i be the set of all subproblem nodes whose visit bit is True, and whose cell contains a point in P_i. We can formulate our goal as finding a minimum-cost solution tree that contains at least one node of each set S_i. This problem resembles GST, and is defined in the work of Chalermsook et al. [10, 9]. We redefine the problem using our own notation.

▶ **Definition 18** (Solution Tree Group Steiner Tree (STGST)). *Let H be a DAG with edge-costs $\mathrm{cost} : E(H) \to \mathbb{R}$ and root r, as well as* groups *$S_i \subseteq V(H)$, for $i \in [h]$, and a partition of the nodes into H^c and H^p. The objective of this problem is to find a minimum-cost solution tree T that contains at least one vertex of every group S_i.*

Their work shows that we can obtain an $O(\log^2 n)$-approximation to STGST (see [10, Sec. 4]). By reducing TSPN to STGST, we can apply the same approximation result, thus proving Theorem 16. We show the details of these steps in Appendix B.2.

5 Conclusion

We have shown that TSPN with line neighborhoods is APX-hard, so a PTAS for this problem is unlikely. This implies the same hardness for k-dimensional flats in $\mathbb{R}^d$ for $1 \le k \le d - 2$, which together with the known PTAS results for $k = 0$ and $k = d - 1$ gives a complete classification of these problems. We have also proved a stronger inapproximability factor for $d = O(\log n)$: there is no $(\sqrt{2} - \varepsilon)$-approximation assuming P $\neq$ NP and no $(2 - \epsilon)$-approximation assuming the UGC. On the positive side, we gave an $O(\log^2 n)$-approximation algorithm in slightly superpolynomial time.

There is still a large gap between the lower bounds and the algorithms for TSPN with line neighborhoods. Perhaps the most important question related to TSPN is to find a constant-approximation for line neighborhoods in $\mathbb{R}^3$, or to prove that it does not exist. Furthermore, for general point sets in higher dimensions there is an inapproximability of $\Omega(\log^2 n/(\log\log n)^{3/2})$ under the Projection Games Conjecture. Whether that holds for flats or lines is an open problem.

References

1 Antonios Antoniadis, Krzysztof Fleszar, Ruben Hoeksma, and Kevin Schewior. A PTAS for Euclidean TSP with hyperplane neighborhoods. In *SODA'19*, pages 1089–1105. SIAM, 2019. doi:10.1137/1.9781611975482.67.

2 Esther M. Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discret. Appl. Math.*, 55(3):197–218, 1994.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998. doi:10.1145/290179.290180.

4 Sanjeev Arora, Michelangelo Grigni, David R. Karger, Philip N. Klein, and Andrzej Woloszyn. A polynomial-time approximation scheme for weighted planar graph TSP. In *SODA'98*, pages 33–41. ACM/SIAM, 1998. URL: http://dl.acm.org/citation.cfm?id=314613.314632.

5 Yair Bartal and Lee-Ad Gottlieb. A linear time approximation scheme for Euclidean TSP. In *FOCS'13*, pages 698–706. IEEE Computer Society, 2013. doi:10.1109/FOCS.2013.80.

6 Yair Bartal, Lee-Ad Gottlieb, and Robert Krauthgamer. The traveling salesman problem: Low-dimensionality implies a polynomial time approximation scheme. *SIAM J. Comput.*, 45(4):1563–1581, 2016. doi:10.1137/130913328.

7 Hans L. Bodlaender, Corinne Feremans, Alexander Grigoriev, Eelko Penninkx, René Sitters, and Thomas Wolle. On the minimum corridor connection problem and other generalized geometric problems. *Comput. Geom.*, 42(9):939–951, 2009. doi:10.1016/j.comgeo.2009.05.001.

8 Jean Bourgain. On Lipschitz embedding of finite metric spaces in Hilbert space. *Israel Journal of Mathematics*, 52(1-2):46–52, 1985.

9 Parinya Chalermsook, Syamantak Das, Guy Even, Bundit Laekhanukit, and Daniel Vaz. Survivable network design for group connectivity in low-treewidth graphs. In *APPROX-RANDOM*, volume 116 of *LIPIcs*, pages 8:1–8:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

10 Parinya Chalermsook, Syamantak Das, Bundit Laekhanukit, and Daniel Vaz. Beyond metric embedding: Approximating group Steiner trees on bounded treewidth graphs. In *SODA'17*, pages 737–751. SIAM, 2017. doi:10.1137/1.9781611974782.47.

11 T.-H. Hubert Chan and Shaofeng H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. *ACM Trans. Algorithms*, 14(1):9:1–9:18, 2018. doi:10.1145/3158232.

12 Moses Charikar, Chandra Chekuri, Ashish Goel, and Sudipto Guha. Rounding via trees: Deterministic approximation algorithms for group Steiner trees and k-median. In Jeffrey Scott Vitter, editor, *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 114–123. ACM, 1998. doi:10.1145/276698.276719.

13 Chandra Chekuri and Martin Pál. A recursive greedy algorithm for walks in directed graphs. In *FOCS'05*, pages 245–253. IEEE Computer Society, 2005. doi:10.1109/SFCS.2005.9.

14 Andrea E. F. Clementi, Pierluigi Crescenzi, and Gianluca Rossi. On the complexity of approximating colored-graph problems. In *COCOON'99*, volume 1627 of *Lecture Notes in Computer Science*, pages 281–290. Springer, 1999. doi:10.1007/3-540-48686-0_28.

15 Mark de Berg, Joachim Gudmundsson, Matthew J. Katz, Christos Levcopoulos, Mark H. Overmars, and A. Frank van der Stappen. TSP with neighborhoods of varying size. *J. Algorithms*, 57(1):22–36, 2005. doi:10.1016/j.jalgor.2005.01.010.

16 Erik D. Demaine, MohammadTaghi Hajiaghayi, and Ken-ichi Kawarabayashi. Contraction decomposition in h-minor-free graphs and algorithmic applications. In *STOC'11*, pages 441–450, 2011. doi:10.1145/1993636.1993696.

17 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *STOC'03*, pages 473–482. ACM, 2003. doi:10.1145/780542.780612.

18 Moshe Dror and James B. Orlin. Combinatorial optimization with explicit delineation of the ground set by a collection of subsets. *SIAM J. Discret. Math.*, 21(4):1019–1034, 2008. doi:10.1137/050636589.

19 Adrian Dumitrescu. The traveling salesman problem for lines and rays in the plane. *Discrete Math., Alg. and Appl.*, 4(4):44:1–44:12, 2012.

20 Adrian Dumitrescu and Joseph S. B. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *J. Algorithms*, 48(1):135–159, 2003. doi:10.1016/S0196-6774(03)00047-6.

21 Adrian Dumitrescu and Csaba D. Tóth. The traveling salesman problem for lines, balls, and planes. *ACM Trans. Algorithms*, 12(3):43:1–43:29, 2016. doi:10.1145/2850418.

22 Khaled M. Elbassioni, Aleksei V. Fishkin, and René Sitters. Approximation algorithms for the Euclidean traveling salesman problem with discrete and continuous neighborhoods. *Int. J. Comput. Geometry Appl.*, 19(2):173–193, 2009. doi:10.1142/S0218195909002897.

23 Lars Engebretsen, Piotr Indyk, and Ryan O'Donnell. Derandomized dimensionality reduction with applications. In *SODA'02*, pages 705–712. SIAM, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545476.

24 Jittat Fakcharoenphol, Satish Rao, and Kunal Talwar. Approximating metrics by tree metrics. *SIGACT News*, 35(2):60–70, 2004. doi:10.1145/992287.992300.

25 Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *J. Algorithms*, 37(1):66–84, 2000. doi:10.1006/jagm.2000.1096.

26 Fabrizio Grandoni, Bundit Laekhanukit, and Shi Li. $O(\log^2 k\ /\ \log\log k)$-approximation algorithm for directed Steiner tree: a tight quasi-polynomial-time algorithm. In *STOC'19*, pages 253–264. ACM, 2019. doi:10.1145/3313276.3316349.

27 Eran Halperin and Robert Krauthgamer. Polylogarithmic inapproximability. In *STOC'03*, pages 585–594. ACM, 2003. doi:10.1145/780542.780628.

28 Su Jia and Joseph S. B. Mitchell. Geometric tours to visit and view polygons subject to time lower bounds, 2019. URL: https://www.andrew.cmu.edu/user/sjia1/wrp_with_tlb.pdf.

29 Håkan Jonsson. The traveling salesman problem for lines in the plane. *Inf. Process. Lett.*, 82(3):137–142, 2002. doi:10.1016/S0020-0190(01)00259-9.

30 Subhash Khot. On the power of unique 2-prover 1-round games. In *STOC'02*, pages 767–775. ACM, 2002. doi:10.1145/509907.510017.

31 Subhash Khot, Dor Minzer, and Muli Safra. Pseudorandom sets in Grassmann graph have near-perfect expansion. In *FOCS'18*, pages 592–601. IEEE Computer Society, 2018. doi:10.1109/FOCS.2018.00062.

32 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008. doi:10.1016/j.jcss.2007.06.019.

33 Robert Krauthgamer and James R. Lee. Algorithms on negatively curved spaces. In *FOCS'06*, pages 119–132. IEEE Computer Society, 2006. doi:10.1109/FOCS.2006.9.

34 Nathan Linial, Avner Magen, and Michael E Saks. Low distortion Euclidean embeddings of trees. *Israel Journal of Mathematics*, 106(1):339–348, 1998.

35 Cristian S. Mata and Joseph S. B. Mitchell. A new algorithm for computing shortest paths in weighted planar subdivisions (extended abstract). In *SoCG'97*, pages 264–273. ACM, 1997. doi:10.1145/262839.262983.

36 Jiří Matoušek. On embedding trees into uniformly convex Banach spaces. *Israel Journal of Mathematics*, 114(1):221–237, 1999.

37 Jiri Matoušsek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

38 Joseph S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999. doi:10.1137/S0097539796309764.

39 Joseph S. B. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *SODA'07*, pages 11–18. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283385.

40 Joseph S. B. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *SoCG'10*, pages 183–191. ACM, 2010. doi:10.1145/1810959.1810992.

41 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.

42 Satish Rao and Warren D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *STOC'98*, pages 540–550, 1998. doi:10.1145/276698.276868.

43 Shmuel Safra and Oded Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. *Comput. Complex.*, 14(4):281–307, 2006. doi:10.1007/s00037-005-0200-3.

44 Luca Trevisan. When Hamming meets Euclid: The approximability of geometric TSP and Steiner tree. *SIAM J. Comput.*, 30(2):475–485, 2000. doi:10.1137/S0097539799352735.

A Theorem 3.1 from [23]

▶ **Theorem 19** (3.1 from [23]). *Let $v_1, v_2, \dots v_m$ be a sequence of vectors in $\mathbb{R}^d$ and let $\varepsilon, F \in (0,1]$. Then we can compute, in deterministic time $O(dm\left(\log n + \frac{1}{\varepsilon}\right)^{O(1)})$, a linear mapping $A : \mathbb{R}^d \to \mathbb{R}^k$ where $k = O(\log(1/F)/\varepsilon^2)$ such that:*

$$k|v_i|^2 \leq |Av_i|^2 \leq k(1+\varepsilon)|v_i|^2$$

for at least a fraction $1 - F$ of i's.

B Details of Section 4

Among all the PTASes for Euclidean TSP, we choose to base our algorithm on the work of Arora, as it results in the lowest running time for our algorithm. Unfortunately, the results of Rao and Smith [42], and Bartal and Gottlieb [5] cannot be adapted for our purposes, since their algorithms use spanners to reduce the total weight of the graph to be a constant factor away from the optimum. It is unclear if this technique can be used for discrete TSPN, as the spanner contains the entire set P of points, and a minimum-cost tree spanning P may be much larger than the optimum solution. In other words, the total weight of the graph can be more than a constant factor away from the weight of an optimal solution.

B.1 Arora's Algorithm

Arora's algorithm consists of three main steps:

1. Perturbation, which changes the instance so that all coordinates are integral and bounded by $O(n)$;
2. Construction of a shifted quadtree;
3. Dynamic program, which finds the approximate solution for TSP.

We describe all of these steps, including any minor alterations needed for them to work in our setting.

B.1.1 Perturbation

Arora shows how to perturb the solution such that:

1. All nodes have integer coordinates;
2. Every (non-zero) distance between two points is at least 8 units;
3. The maximum distance between two points is $O(n)$.

Given a bounding box on the instance of size L_0, Arora achieves this perturbation by snapping points to an appropriately fine grid. To use this step for our problem, we need to specify a value of L_0 such that $\mathrm{OPT} \le L_0 \le O(\mathrm{OPT})$. To this effect, we guess the value of OPT rounded up to a power of 2, as well as a vertex v_0 that is included in an optimum solution. We implement this guessing step by iterating over all of the possible values, and computing a feasible solution for each possibility. The best feasible solution we obtain will be at least as good as the solution for the correct guess (in expectation).

The guessing step is done as follows. We start by guessing a vertex v_0 that is contained in an optimum solution. Then, we compute the minimum radius R_0 such that at least one point from each neighborhood is contained in the ball B of radius R_0 centered at v_0. Such a ball can be computed simply by iterating over all neighborhoods and finding the neighborhood's nearest point to v_0. If the optimum solution contains v_0, then its cost is at least R_0, as it must visit the farthest neighborhood, at distance R_0. On the other hand, $\mathrm{OPT} \le 2R_0 n$, since the ball B contains at least one point from each neighborhood, and the distance between any two points in B is at most $2R_0$. Hence, there is a tour of cost at most $2R_0 n$. Knowing that $R_0 \le \mathrm{OPT} \le 2R_0 n$ (assuming v_0 is in an optimum solution), we can simply run the algorithm for every v_0 and for any $R \in [R_0, 4R_0 n]$ that is a power of 2.

Given a vertex v_0 and a guess R for the value of the optimum solution, we set $L_0 = R/2$ (so that if $R/2 \le \mathrm{OPT} \le R$, $L_0 \le \mathrm{OPT}$). Finally, we remove all of the vertices $u \in P$ that are at a distance more than R from v_0, that is, $\mathrm{dist}(v_0, u) > R$. A solution containing both v_0 and u would cost more than $R \ge \mathrm{OPT}$, implying that for correct choices of R and v_0, such vertices can be safely removed. We now have a bounding box of side length $4L_0$ containing all the points in the instance, and hence the perturbation step in Arora's algorithm ensures the stated properties.

B.1.2 Construction of a shifted quadtree

Let $L = O(n)$ be the size of the bounding box. The algorithm computes a random shift $a' = (a'_1, a'_2, \dots, a'_d)$, with $a'_i \in \{0, \dots, L-1\}$, $i \in [d]$. Then, it constructs a quadtree where the dissection points are shifted according to a'. The resulting quadtree has height $O(\log n)$, and $O(n \log n)$ cells. For our purpose, no changes are needed to this process.

B.1.3 Dynamic Program

Arora's algorithm uses dynamic programming to find a *salesman path*, which may visit additional points along the boundary of the cells of the quadtree. The following definition formalizes this concept.

▶ **Definition 20.** *Let m, r be positive integers. An m-*regular set of portals *for a shifted dissection is a set of points on the facets of the cells in it. Each cell has a portal at each of its vertices and m other portals on each facet, placed in a $d-1$-dimensional square grid whose vertices are identical to the vertices of the facet.*

A salesman path *is a path in $\mathbb{R}^d$ that visits all the input points, and some subset of portals. It may visit a portal more than once.*

*The salesman path is (m, r)-*light *with respect to the shifted dissection if it crosses each facet of each cell in the dissection at most r times and always at a portal.*

The goal of the dynamic program is to find a minimum cost (m, r)-light salesman path, for the instance. For our purpose, a 2-approximation of TSP is sufficient, and hence we set $m = O(\sqrt{d}\log n)^{d-1}$ and $r = O(\sqrt{d})^{d-1}$. By restricting the solution to cross the cell boundaries only through portals, we can see that any solution to the problem, restricted to a single cell, consists of a set of paths that together cover all of the points inside the cell. Since we want to find an (m, r)-light solution, this further implies that at most r portals per facet of the cell are used. This motivates the definition of the *(m, r)-multipath problem*, which is the problem solved by the dynamic program for each cell:

▶ **Definition 21** ((m, r)-multipath problem [3])**.** *An instance of this problem is specified by the following inputs:*

1. *A nonempty cell in the quadtree.*
2. *A multiset of r portals on each of the $2d$ facets of this cell such that the sum of the sizes of these multisets is an even number $2p \leq 2dr$.*
3. *A pairing $(a_1, a_2), (a_3, a_4), \dots (a_{2p-1}, a_{2p})$ between the $2p$ portals specified in Item 2.*

The goal in the (m, r)-multipath problem is to find a minimum cost collection of p paths in the cell that is (m, r)-light. The i-th path connects a_{2i-1} to a_{2i}, and the p paths together visit all the points in the cell.

The dynamic programming table consists of all of these instances of (m, r)-multipath problem, for each cell and pairing of portals (considered here to include the multiset of portals in Item 2. We refer to the multiset of portals and their pairing as the *state* of an (m, r)-multipath problem.

The values of the table can be computed recursively. The entries corresponding to leaves of the quadtree can be easily determined: given the portal set of size $2p$ and the pairing, we simply need to find the shortest paths between the paired portals, and add the (single) point in the cell to one of these paths. For all other entries, the algorithm enumerates all possible ways that the p paths can cross the boundary between children cells. For each of these arrangements, the cost of the solution can be obtained by summing the costs of the respective instances for the children cells. Once all of the entries have been computed, the minimum cost (m, r)-light salesman path can be found by looking at the (m, r)-multipath problem for the root cell of the quadtree with no portals used.

The dynamic programming table contains a total of $O\big(n(\log n)^{O(d)^{(d-1)/2}}\big)$ entries, and the value at each cell can be computed in time $(\log n)^{O(d)^{(d-1)/2}}$. Therefore, the running time of this algorithm is $O\big(n(\log n)^{O(d)^{(d-1)/2}}\big)$.

Our algorithm uses a very similar dynamic program, with only a small change needed at the leaves. In the TSP problem, all of the points must be visited, which implies that any feasible solution to the (m, r)-multipath problem must visit all the points contained in that cell. However, the same is not true of the TSPN problem: as long as one point from each neighborhood is visited in the whole path, the solution is feasible, which means that not all neighborhoods are visited in every cell that intersects them. To that effect, we add an extra input to the (m, r)-multipath problem for leaf cells, which we call *visit bit*. If the visit bit is set to True, then the (single) point in the cell must be visited; if it is set to False, then the solution only needs to connect the portals as specified in the input (meaning that the optimum solution will be a union of shortest paths between paired portals).

We remark that finding a solution in this new dynamic program can be thought of as two different tasks: choosing which points in the leaf cells should be visited (by choosing the corresponding subproblems with visit bit True or False) and choosing the tour visiting these points (using the portals as in the original dynamic program).

B.2 Approximating TSPN using the framework by Chalermsook et al.

To prove Theorem 16, we need to show how to formulate TSPN as an instance of STGST, and then show how to obtain an $O(\log^2 n)$-approximation to this problem. In this section, we provide details to both of these steps.

B.2.1 Formulating discrete TSPN as an instance of STGST

We will formally describe the construction of a DAG H based on the dynamic program for TSP. We assume that the perturbation and random shift steps implemented by Arora have been performed, with the alterations described in Section 4.2.

We now consider the dynamic program as presented by Arora, and construct our DAG H as follows. The vertex set is partitioned into *subproblem nodes* H_p and *combination nodes* H_c.

- For every (m, r)-multipath subproblem considered by Arora, we create a *subproblem node*. Formally, for every cell C in the quadtree, and every state A or (A, b) (where b represents the visit bit if C is a leaf cell), we create a node $t[C, A]$ (resp. $t[C, A, b]$) in H_p.
- For every non-leaf cell C with children $C_1, \dots, C_k$ and states X for C and X_i for C_i, we add a *combination node* $t^c[C, X, \{X_i\}_{i\in[k]}]$ if the states are consistent, that is, if the combination of the portal pairings for each of the cells C_i forms the portal pairing represented by X in C.
- For each combination node $t' = t^c[C, X, \{X_i\}_{i\in[k]}]$, we add edges from $t[C, X]$ to t' and from t' to $t[C_i, X_i]$ for each $i \in [k]$.
- The edges entering leaf nodes $t[C, A, b]$ have cost equal to the minimum cost of a solution to the (m, r)-multipath problem in C with portal pairings specified by A, and which visits the point in C if $b =$ True.
- All other edges have cost 0.

The root of H is the node $t[C, X]$, where C is the root cell of the quadtree (the bounding box of the instance), and X represents an empty set of portals. The height of the resulting DAG H is $O(\log n)$, since its structure is similar to the dynamic program of Appendix B.1.

▶ **Lemma 22.** *Let $v_0 \in P$ be a point and R_0 be a radius guessed in Appendix B.1.1.*

For every (m, r)-light tour F in the resulting quadtree there is a solution tree X in H such that $cost(F) = cost(X)$ and they visit the same set of points in P.

Similarly, for any solution tree $X \subset H$, there is an (m, r)-light tour F of the same cost, which visits the same points in P.

B.2.2 Obtaining an $O(\log^2 n)$-approximation

We will use the following theorem, which is implicit in the work of Chalermsook et al. [10, Section 4]. We remark that, even though the work of Chalermsook et al. is formulated in terms of the tree decomposition of a graph, it also applies when obtaining the DAG from a quad-tree, as the necessary properties still hold.

▶ **Theorem 23** ([10, 9]). *Let H be a DAG with edge-costs $cost : E(H) \to \mathbb{R}$ and root r, as well as groups $S_i \subseteq V(H)$, for $i \in [h]$, and a partition of the nodes into H_c and H_p. There is an algorithm that outputs a solution tree $X \subseteq H$ sampled from a distribution $\mathcal{D}$ such that:*

1. $\mathrm{E}_{X\sim\mathcal{D}}[cost(X)] \leq cost(\mathrm{OPT})$*, where* $cost(\mathrm{OPT})$ *denotes the cost of the optimal solution*
2. *For any group S_i, the probability that the group is covered (for some constant $\alpha > 1$) is* $\Pr_{X\sim\mathcal{D}}\left[|S_i \cap X| > 0\right] \geq \frac{1}{\alpha\,\mathrm{height}(H)}$

The algorithm runs in time $\Delta(H)^{O(\mathrm{height}(H))}$, where $\Delta(H)$ is the max-degree of H.

We will now show how to use Theorem 23 and Lemma 22 to obtain an $O(\log N \log n)$-approximation for the TSPN problem on discrete neighborhoods, and hence prove Theorem 16.

We start by guessing a vertex v_0 to be the starting point of our solution. For every vertex $v_0 \in P$, we compute the minimum radius R_0 such that every neighborhood contains a point at distance at most R_0 from v_0. Next, we guess R, an approximation for OPT, in the range $[R_0, 4nR_0]$. For the powers $R = 2^i$, $i \in \mathbb{Z}, R_0 \leq R \leq 4nR_0$, we can now preprocess the instance according to the perturbation step of Arora's algorithm. (Appendix B.1.1). Next, we enumerate the shift $a = (a_1, \ldots, a_d) \in \{0, \ldots, L-1\}^d$, and construct the shifted tree as in Arora's algorithm (Appendix B.1.2). Finally, we construct the DAG H based on the dynamic programming table, as specified in Appendix B.2.1. We recall that the height of the tree, as well as of DAG H is $O(\log N)$.

We now use Theorem 23 repeatedly to obtain solution trees $X_1, \ldots, X_\ell$, where $\ell = c \log n \log N$, and c is a large constant. Then, we use Lemma 22 to convert each solution tree X_i into a tour F_i, and finally take the union of all these tours to obtain a solution F. While F is not necessarily a tour, it is simple enough to remove crossings. For every neighborhood P_i that is not visited by F, we add a detour visiting the closest point in P_i. We denote by F^* the minimum-cost solution among all solutions F for all the enumerated values of v_0, R_0, and a.

By construction, F^* is a feasible solution, as it is a tour that visits every group. To prove that it is $O(\log N \log n)$-approximate, consider the solution F' that we obtained for the correct values of v_0, R_0, and a, that is, for a vertex v_0 in an optimum solution, R_0 such that $R_0/2 \leq \mathrm{OPT} \leq R_0$, and a shift a for which an (m, r)-light tour exists. By Theorem 23, each of the solution trees X_i' obtained has expected cost at most OPT, and by Lemma 22, the corresponding tour F_i' also has expected cost at most OPT. Therefore, the union of all tours F_i' costs at most $O(\log N \log n\,\mathrm{OPT})$ in expectation. The probability that a neighborhood is not visited, and hence that we must add a detour, is (for sufficiently large c)

$$\begin{aligned}\Pr\left[\bigcap_j |S_i \cap X_j| = 0\right] &\leq \left(1 - \frac{1}{\alpha\,\mathrm{height}(H)}\right)^{\ell} \\ &\leq e^{-O(\log n)} \\ &\leq \frac{1}{n^3}\end{aligned}$$

We conclude that the expected cost of F' is at most $O(\log N \log n \,\mathrm{OPT})$, and since, by Lemma 22, $cost(F^*) \leq cost(F')$, F^* is $O(\log N \log n)$-approximate in expectation. By Theorem 23, the running time of our algorithm is

$$N^{O(d)} \, (\log N)^{O(d)^{(d-1)/2} \, O(\log N)} = N^{O(d)^{(d-1)/2} \log\log N}.$$

This completes the proof of Theorem 16.