Airports and Railways: Facility Location Meets Network Design[*]

Anna Adamaszek[1], Antonios Antoniadis[2], and Tobias Mömke[3]

1 **University of Copenhagen, Denmark**
`anad@di.ku.dk`

2 **Max-Planck-Institut für Informatik, Saarbrücken, Germany**
`aantonia@mpi-inf.mpg.de`

3 **Saarland University, Germany**
`moemke@cs.uni-saarland.de`

Abstract

We introduce a new framework of Airport and Railway Problems, which combines capacitated facility location with network design. In this framework we are given a graph with weights on the vertices and on the edges, together with a parameter k. The vertices of the graph represent cities, and weights denote respectively the costs of opening airports in the cities and building railways that connect pairs of cities. The parameter k can be thought of as the capacity of an airport. The goal is to construct a minimum cost network of airports and railways connecting the cities, where each connected component in the network spans at most k vertices, contains an open airport, and the network satisfies some additional requirements specific to the problem in the framework.

We consider two problems in this framework. In the $\mathrm{AR_F}$ problem there are no additional requirements for the network. This problem is related to capacitated facility location. In the $\mathrm{AR_P}$ problem, we require each component to be a path with airports at both endpoints. $\mathrm{AR_P}$ is a relaxation of the capacitated vehicle routing problem (CVRP).

We consider the problems in the two-dimensional Euclidean setting. We show that both $\mathrm{AR_F}$ and $\mathrm{AR_P}$ are NP-hard, even for uniform vertex weights (i. e., when the cost of building an airport is the same for all cities). On the positive side, we provide polynomial time approximation schemes for $\mathrm{AR_F}$ and $\mathrm{AR_P}$ when vertex weights are uniform. We also investigate $\mathrm{AR_F}$ and $\mathrm{AR_P}$ for $k = \infty$. In this setting we present an exact polynomial time algorithm for $\mathrm{AR_F}$ with general vertex costs, which also works for general edge costs. In contrast to $\mathrm{AR_F}$, $\mathrm{AR_P}$ remains NP-hard when $k = \infty$, and we present a polynomial time approximation scheme for general vertex weights.

We believe that our PTAS for $\mathrm{AR_P}$ with uniform vertex weights and arbitrary k brings us closer towards a PTAS for Euclidean CVRP, for which the main difficulty is to deal with paths of length at most k.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, geometric approximation, facility location, network design, PTAS

Digital Object Identifier 10.4230/LIPIcs.STACS.2016.6

[*] Partially supported by the Danish Council for Independent Research DFF-MOBILEX mobility grant, and by Deutsche Forschungsgemeinschaft grant BL511/10-1 and MO 2889/1-1.

1 Introduction

We introduce a new framework of Airport and Railway Problems, which combines capacitated facility location with network design. In this framework, an input instance (G, a, r, k) consists of a complete n-vertex graph $G = (V, E)$ with vertex costs $a\colon V(G) \to \mathbb{R}$ and edge costs $r\colon E(G) \to \mathbb{R}_{\geq 0}$, and a parameter k. The goal is to compute a minimum cost network of airports $A \subseteq V(G)$ and railways $R \subseteq E(G)$ connecting all the cities (i. e., each vertex in V is connected with some vertex from A via a path of edges from R), where each connected component of the network contains at most k vertices. Problems from this framework can have additional constraints on the network, e. g., requiring a fixed number κ of connected components, or enforcing special structure of the connected components. Unless stated differently, we assume that r specifies distances in the two-dimensional Euclidean space.

Paths. We define AR_P as an airport and railway problem with an additional property that each connected component is a path with airports at the endpoints. Formally, we want to find subsets $A \subseteq V(G)$ and $R \subseteq E(G)$ of minimum total cost $a(A) + r(R)$[1] such that (i) each connected component of the graph (V, R) is a path of length at most k, and (ii) for each $v \in V$ we have $v \in A$ if and only if v is an endpoint of a path in R. For consistency we assume that each path contains two airports, i. e., when accounting for the cost of A, we consider each vertex that forms a singleton path twice.[2] For a solution H to AR_P, we define $\tilde{a}(H) = \sum_{v\in A} a(v)$, where A is the multiset of all endpoints of the paths of H.

The AR_P problem relaxes the capacitated vehicle routing problem (CVRP), which is a fundamental vehicle routing problem (cf. Dantzig and Ramser [8]). The input to CVRP is an edge-weighted graph with a special vertex called the *depot*, and a parameter k, the capacity. The goal is to find a minimum cost tour that (i) starts and ends at the depot, (ii) visits all vertices, and (iii) visits at most k vertices between any two consecutive visits to the depot. CVRP is a special case of AR_P, where for each vertex $v \in V$ the cost $a(v)$ equals the distance between v and the depot. The existence of a PTAS for Euclidean CVRP remains an important open problem.

Notice that in some applications where goods need to be distributed or a service has to be provided to the customers, unlike in CVRP, the cost of moving the salesmen to and from the depot is not proportional to the distance from the depot. For example, a salesman can move to and from the depot using public transport, and then $a(v)$ will denote the cost of the public transport ticket. These applications motivate a model where the airport costs $a(v)$ are uniform (which we later denote by $1\mathrm{AR}_\mathrm{P}$), as well as the general AR_P model.

Trees. We define AR_F as a basic airport and railway problem, with no additional properties of the network. Formally, we want to find subsets $A \subseteq V(G)$ and $R \subseteq E(G)$ of minimum total cost $a(A)+r(R)$ such that each connected component of the graph (V, R) has at most k vertices, and contains one vertex from A (i. e., one open airport).[3] We remark that since r is nonnegative, any optimal solution to AR_F is always a forest, and that it opens the cheapest airport in each component. As for paths, for a solution H we define $\tilde{a}(H) = \sum_{v\in A} a(v)$, but here A is the set that contains the vertex with the cheapest airport of each component.

[1] We write $a(A)$ and $r(R)$ as shorthands for $\sum_{v\in A} a(v)$ and $\sum_{e\in R} r(e)$, respectively.
[2] We remark that our results can be easily adapted for the case where we count the cost of an airport at a singleton vertex once instead of twice.
[3] We restrict the number of airports per component for technical reasons. It is straightforward to adapt our algorithms to allow multiple airports per component.

$\mathrm{AR_F}$ is related to the well known capacitated facility location problem, where instead of connecting each client directly to an open facility we construct a network which jointly connects all clients to a facility, which seems more appropriate for some applications. Another interesting interpretation of $\mathrm{AR_F}$ is that each subset U of at most k vertices forms a hyperedge. The cost of each such hyperedge is $r(H) + \tilde{a}(H)$, where H is a minimum spanning tree on U, and the goal is to find a perfect hyper-matching of minimum cost.

1.1 Our Results

We consider the $\mathrm{AR_P}$ and $\mathrm{AR_F}$ problems in the two-dimensional Euclidean setting. For both $\mathrm{AR_P}$ and $\mathrm{AR_F}$ we consider the uniform airport cost case, i. e., the case where $a(v) = a(w)$ for all $v, w \in V$, and the unrestricted airport capacity case, when $k = \infty$. We call the resulting problems $1\mathrm{AR_P}$, $1\mathrm{AR_F}$ and $\mathrm{AR_P}^\infty$, $\mathrm{AR_F}^\infty$.

Using a reduction from the Euclidean TSP path problem, i.e., the problem of finding a minimum cost Hamiltonian path in a given graph, we can prove the following result.

▶ **Theorem 1.** *The $\mathrm{AR_P}$ problem is NP-hard, even in Euclidean graphs with unit airport costs and $k = \infty$.*

In a similar manner, the APX-hardness of metric-TSP [14] implies that $\mathrm{AR_P}$ is APX-hard in metric graphs. The hardness result holds even when we have unit airport costs and $k = \infty$.

Theorem 1 implies in particular that both $1\mathrm{AR_P}$ and $\mathrm{AR_P}^\infty$ are NP-hard. On the positive side we develop polynomial time approximation schemes for these two problems. In fact, for $\mathrm{AR_P}^\infty$ we present a stronger result: we may specify the exact number of paths κ required by the solution. We denote the corresponding instance by (G, a, r, k, κ).

▶ **Theorem 2.** *For any $\kappa \in \mathbb{N}$ there is a PTAS for the two-dimensional Euclidean $\mathrm{AR_P}^\infty$ problem, when we require both the optimal and the algorithmic solution to have exactly κ connected components.*

Our technique for proving Theorem 2 is an intricate extension of the well known PTAS of Arora [4] for the Euclidean TSP. One obstacle is that unlike in Arora's result, in $\mathrm{AR_P}^\infty$ one cannot in general restrict the bounding box to be of linear size. We cope with this by scaling the instance with a carefully chosen scaling parameter and afterwards dissecting it into a collection of separate instances. A further difficulty is caused by crossings in the solution. We prove that paths can be uncrossed without an increase in the cost of the solution. After this preparation, we are able to extend Arora's dynamic program to be applicable for our setting. Theorem 2 is discussed in Section 3.2.

We can prove that the $1\mathrm{AR_F}$ problem is NP-hard, which in particular implies NP-hardness of the general $\mathrm{AR_F}$.

▶ **Theorem 3.** *The $\mathrm{AR_F}$ problem is NP-hard, even in Euclidean graphs with unit airport costs.*

The proof of the theorem consists of a reduction from the NP-hard planar monotone cubic One-in-Three Satisfiability problem (PMC-1-in-3-SAT). Notice that, in contrast to Theorem 1, this hardness result requires a bound on the airport capacity k.

We show that the $\mathrm{AR_F}^\infty$ problem, unlike $\mathrm{AR_P}^\infty$, can be solved exactly in polynomial time, even when we require that the solution has exactly κ connected components.

▶ **Theorem 4.** *For an arbitrary $\kappa \in \mathbb{N}$ and edge-cost function $r\colon E \to \mathbb{R}_{\geq 0}$, there is an exact polynomial-time algorithm for the $\mathrm{AR_F}^\infty$ problem when we require both the optimal and the algorithmic solution to have exactly κ connected components.*

To prove Theorem 4, we augment the input graph by a special vertex corresponding to the airports, and then we reduce the problem to the matroid intersection problem. Notice that here r is not required to be a metric. We discuss Theorem 4 in detail in Section 3.1.

We will use our algorithms for $\mathrm{AR_P}^\infty$ and $\mathrm{AR_F}^\infty$ with a requirement of κ on the number of connected components to obtain a PTAS for $1\mathrm{AR_F}$ and $1\mathrm{AR_P}$, which is the main result of this paper.

▶ **Theorem 5.** *There is a PTAS for the two-dimensional Euclidean* $1\mathrm{AR_P}$ *and* $1\mathrm{AR_F}$ *problem.*

We note that a generalization of the above result for $\mathrm{AR_P}$ from uniform airport costs to a constant range of airport costs would yield a PTAS for Euclidean CVRP, using a result of Adamaszek et al. [1].

In order to prove Theorem 5, we first partition the problem instance into a collection of independent subinstances. The partitioning borrows ideas from the random shift method used by Arora in his PTAS for Euclidean TSP. However, we obtain a collection of independent subinstances where each subinstance has a bounding box of constant size, instead of linear as Arora's random shift would give. We distinguish two types of subinstances: sparse and dense ones, depending on the number of connected components in an optimal solution for the instance. We obtain a PTAS for the sparse instances by again slightly adapting the PTAS by Arora. In contrast, developing an algorithm for dense instances is much more involved. We first obtain an initial solution by running our algorithms for $\mathrm{AR_P}^\infty$ ($\mathrm{AR_F}^\infty$, respectively). This solution can be infeasible for $1\mathrm{AR_P}$ ($1\mathrm{AR_F}$, respectively), and we proceed by dissecting each connected component into smaller pieces of similar size. Finally, we show how these smaller pieces can be reassembled into a feasible solution which has cost not much greater than the cost of an optimal solution. Theorem 5 is discussed in Section 4.

We believe that our technique of dissecting a cheap but infeasible solution and reassembling the pieces to form a cheap feasible solution can be applied to other geometric graph problems, including Euclidean CVRP.

1.2 Related Work

Our new framework of Airport and Railway Problems combines capacitated facility location with network design. In the capacitated facility location (CFL) problem we are given a complete graph $G = (V, E)$ on n vertices $v_1, \ldots, v_n$, with edge costs $d\colon E(G) \to \mathbb{R}_{\geq 0}$ and vertex costs $c\colon V(G) \to \mathbb{R}_{\geq 0}$, together with a capacity parameter k. Here $d(v_i, v_j)$ denotes the distance between vertices v_i and v_j, and $c(v_i)$ denotes the cost of opening a facility at v_i. A solution to the problem consists of a set of facilities $F \subseteq V$ to be opened, and an assignment of each vertex v_i to some open facility $f(v_i)$ such that each facility has at most k vertices assigned to it. The goal is to find a solution minimizing the total cost of opening the facilities and connecting all vertices to the assigned facilities, i.e., $\sum_{v \in V : v \in F} c(v) + \sum_{v \in V} d(v, f(v))$. Here all vertices are connected to the open facilities by *direct links.*

In the network design problems we are given a graph with weights on the edges and possibly also on the vertices, and the goal is to find a minimum cost set of edges satisfying some given constraints, e.g., connectivity constraints. That might be for example constructing a network that allows routing flow from all the vertices to a *specified* set of sinks. Our framework combines the two settings above.

The $\mathrm{AR_F}$ problem is related to the problems of *capacitated facility location* and *capacitated minimum spanning tree* (CMST). In CMST the goal is to construct a minimum cost collection of trees, each spanning at most k vertices and connected to a single pre-specified root, covering all the input vertices. Jothi and Raghavachari [13] give a 3.15-approximation algorithm for

CMST instances in the Euclidean plane, allowing demands on the vertices. Other problems (more remotely) related to AR_F can be found in [15, 10] and [17].

The AR_P problem relaxes the capacitated vehicle routing problem (CVRP), which has been introduced by Dantzig and Ramser [8] in 1959, and has been studied extensively by the Computer Science and Operations Research communities. When the underlying graph G resembles a metric, CVRP is known to be APX-hard [6]. However, Arora [4] conjectured that when G resembles the Euclidean metric, then CVRP admits a PTAS. Das and Mathieu [9] developed a quasi-polynomial time approximation scheme (QPTAS) for Euclidean CVRP, which gives raise to expect the existence of a PTAS for this setting. There have been several results in this direction. There is a PTAS for the cases of very large capacity $k = \Omega(n)$ [6], and small capacity $k \leq 2^{\log^{o(1)} n}$ [1]. Despite these efforts, the exact approximability of CVRP in the Euclidean metric remains an open question. As Arora [4] and Das and Mathieu [9] point out, the main difficultly in adapting Arora's algorithm to CVRP with arbitrary parameter k lies in controlling the interface of the dynamic program. In particular, one cannot recombine different tours arbitrarily since an "uncrossing" may transform two tours of length at most k into one longer tour and one shorter tour. A second difficulty is due to the cost associated with returning to the depot. In the problems considered in this paper we were able to circumvent these obstacles, and we believe that our results could provide a step towards obtaining a PTAS for CVRP with an arbitrary parameter k.

Finally, the *capacitated location routing problem* (CLR, see [12]) and the *k-location capacitated vehicle routing* (k-LocVRP, see [11]) can be seen as problems in our framework, although they have been studied for general graphs and allow demands on the vertices. CLR is a generalization of multiple depot CVRP, with costs of opening the depots. k-LocVRP is also related to multiple depot CVRP, where we can choose k depots to be opened, one per vehicle. Harks et al. [12] and Gørtz and Nagarajan [11] provide constant-factor approximation algorithms for CLR and k-LocVRP, respectively. Another related problem is *capacitated geometric network design* (CGND) where the goal is to create a network of capacitated links which allows sending flow from all the vertices to the sink. CGND resembles AR_F, when instead of a bound on the airport capacity we have a bound on the railroad capacity. In CGND the sink is pre-specified, and the optimal network can be more complicated than a tree. Adamaszek et al. [2] provide a PTAS for the two-dimensional Euclidean CGND for link capacities $k \leq 2^{O(\sqrt{\log n})}$, where the network can use Steiner vertices anywhere in the plane.

2 Preliminaries: Arora's Scheme

Some of our results build on Arora's PTAS for Euclidean TSP [3]. Therefore we provide a short reminder of the main steps of the algorithm.

Step 1 (Perturbation): Perturb the instance such that it becomes "well rounded", i. e., (i) all nodes are at integer coordinates, and (ii) the maximum inter-node distance is $O_\epsilon(n)$. Such an operation can be performed because the cost of an optimal TSP solution is at least as large as the maximum inter-node distance.

Step 2 (Shifted Quadtree): Starting from a square that contains all nodes of the instance, we recurse over a logarithmic number of levels, each time subdividing the square into four smaller subsquares. That gives us a quadtree of logarithmic depth, where the root corresponds to the square containing all nodes of the instance, and all other nodes of the tree correspond to the subsquares of the dissection. The leaves correspond to unit squares.

We introduce randomness and shift the quadtree in the following way. Choose two integers $a, b \in [0, L)$ uniformly at random, where L is the length of the bounding box side. Shift the basic quadtree by a units in the horizontal and b units in the vertical direction and wrap around the boundaries. Note that this does not affect the position of the nodes.

Step 3 (Dynamic Programming): Fix $m = O_\epsilon(\log n)$ and $r = O_\epsilon(1)$. We introduce portals on the plane in the following way. For the quadtree constructed in the previous step, for each side of each dissection square, we create m equidistant portals along the side. We call a TSP solution (m, r)-light if (i) it is portal-respecting (i.e., it enters and exits dissection squares only at the portals), and (ii) any edge of a dissection square is crossed at most r times by the solution. Arora showed that with respect to the random shift of Step 2, an optimal (m, r)-light TSP tour has an expected cost not much greater than the optimal TSP tour.
We can find an optimal (m, r)-light TSP tour using dynamic programming (DP). There is a DP cell for each combination of (i) a dissection square of the quadtree, and (ii) the description of pairs of portals used by the tour to enter and exit the square.

In the remaining document, when referring to Steps 1–3 of Arora's algorithm, we refer to the above steps.

3 Unrestricted Airport Capacity

In this section we will consider the problems $\mathrm{AR_F}^\infty$ and $\mathrm{AR_P}^\infty$.

In general, given some algorithm Alg, we denote by alg the cost of the solution computed by Alg. Similarly, an optimal solution Opt has a cost of opt.

3.1 An Exact Algorithm for $\mathrm{AR_F}^\infty$ and Theorem 4

We first introduce a simple algorithm MST-$\mathrm{AR_F}$ for the $\mathrm{AR_F}^\infty$ problem, for any edge-cost function $r \colon E \to \mathbb{R}_{\geq 0}$, and without the restriction on the number of connected components κ. The algorithm is a slight extension of finding a minimum spanning tree, and for any instance $(G, a, r, k = \infty)$ of $\mathrm{AR_F}^\infty$ it performs the following operations.

1. Construct a minimum spanning tree instance (G', r') by augmenting (G, r) by an auxiliary vertex v', and an edge (v', v) with weight $a(v)$ for each $v \in V(G)$.
2. Compute a minimum spanning tree MST' of G'.
3. Output $A = \{v \in V : \{v, v'\} \in E(MST')\}$, $R = \{\{v_1, v_2\} \in E(MST') : v_1, v_2 \neq v'\}$.

It is easy to see that for each solution for the $\mathrm{AR_F}^\infty$ instance there is a corresponding spanning tree of G' with the same cost, and vice versa. That gives us the following result.

▶ **Corollary 6.** MST-$\mathrm{AR_F}$ *is an exact polynomial time algorithm for* $\mathrm{AR_F}^\infty$*, for any edge-cost function* $r \colon E \to \mathbb{R}_{\geq 0}$.

If we enforce a given number of connected components in the solution, as required by Theorem 4, the algorithm becomes more involved. It uses the augmented graph (G', r') and matroid intersection.

Proof of Theorem 4. The theorem follows from matroid intersection. Let $(E', \mathcal{I}_1)$ be the set system with $E' = E(G')$ and $\mathcal{I}_1$ the collection of the edge sets of all forests in G', where G' is the graph constructed in step one of MST-$\mathrm{AR_F}$. Then $(E', \mathcal{I}_1)$ is the cycle matroid of G' and its bases are the spanning trees of G', of size $|V(G') - 1|$. To define a second matroid, let $E'' \subset E'$ be the set of all edges incident to v'. Then we define the second set system

$(E', \mathcal{I}_2)$ where a set $I \subseteq E'$ is in $\mathcal{I}_2$ if (i) $|E'' \cap I| \leq \kappa$ and (ii) $|(E' \setminus E'') \cap I| \leq |V(G')| - 1 - \kappa$. Clearly, $(E', \mathcal{I}_2)$ is the union of two uniform matroids and therefore a matroid itself. The bases of $(E', \mathcal{I}_2)$ are edge sets of size $|V(G') - 1|$ with exactly κ edges incident to v'.

The common bases of $(E', \mathcal{I}_1)$ and $(E', \mathcal{I}_2)$ are all spanning trees in G' with exactly κ edges from E'': they are spanning trees due to $(E', \mathcal{I}_1)$, and the κ edges incident to v' are ensured by $(E', \mathcal{I}_2)$. We can find a minimum cost common base in strongly polynomial time (see, for instance, Theorem 41.8 in Schrijver's book [16]). ◀

3.2 A PTAS for $\mathrm{AR_P}^\infty$ and Theorem 2

The $\mathrm{AR_P}^\infty$ problem is a generalization of the s,t-path TSP problem. To transform an s,t-path TSP instance to an instance of $\mathrm{AR_P}^\infty$, we set the airport costs of s and t to zero, and all other airport costs to infinity. This already implies NP-hardness of $\mathrm{AR_P}^\infty$, but in Theorem 1 we show that the problem is NP-hard even with unit airport costs. We extend Arora's scheme for Euclidean TSP to obtain a PTAS for the more general setting of $\mathrm{AR_P}^\infty$.

Perturbation. First, we need to obtain a well-rounded instance. We cannot immediately use the bounding box of Step 1 in Arora's algorithm for TSP, because there may be large gaps between distinct connected components of an optimal solution, and the length of the sides of the bounding box can be arbitrarily large with respect to opt. A similar problem occurs for example when trying to apply Arora's algorithm to the Euclidean k-median problem. Arora et al. [5] obtain a PTAS for the Euclidean k-median, where for the perturbation step they apply a min-max clustering algorithm of Bern and Eppstein [7]. Unlike in the case of k-median, we do not have such an algorithm. For $\mathrm{AR_P}^\infty$, we can prove the following result.

▶ **Lemma 7.** *For an arbitrary $\epsilon > 0$ and an instance $(G, a, r, k = \infty, \kappa)$ of $\mathrm{AR_P}^\infty$ requiring κ connected components, one can compute in polynomial time a collection of independent subproblems $(G_i, a_i, r_i, k = \infty, \kappa_i)$ such that (i) for each subproblem all vertices have integer coordinates and the maximum inter-node distance is bounded by $O_\epsilon(n^2)$, and (ii) if we know $(1+\epsilon)$-approximate solutions for all the subproblems we can compute in polynomial time a $(1 + O(\epsilon))$-approximate solution for the original problem instance.*

The main idea of the proof of Lemma 7 is to guess the longest edge e of an optimal solution (by trying all possibilities). Then $r(e)$ is a lower bound on opt. Using this lower bound and after an appropriate scaling of the costs, we can move all vertices to the nearest integer coordinates. We can also cluster the vertices such that the maximum inter-node distance in each cluster is $O_\epsilon(n^2)$, and the distance between any two vertices from different clusters is at least $2r(e)$. This ensures that we do not split any connected components of OPT between multiple clusters. That gives us independent problem instances, and from their solutions we can restore a solution for the original instance.

Random shift. We perform a random shift of the quadtree, as in Step 2 of Arora's algorithm. Let α, β be the random variables describing the offsets of the random shift. Note that in the quadtree, there is a new vertex at each portal. We refer to these vertices as *portal vertices*. Unlike all other vertices, the portal vertices are not required to be contained in the solution.

Uncrossing and (m, r)-thin solutions. Two paths P, P' are intersecting, if $P \cap P' \neq \emptyset$. We have to consider a technical detail here: there may be several vertices located at the same position, as well as pairs of edges whose intersection is an interval. We call an intersection of two paths (i.e., a connected component of $P \cap P'$) a *crossing* if the intersection cannot be avoided by moving the vertices of the paths by infinitesimal distances. Similarly, a self-intersection of a single path that cannot be avoided in this way is also called a *crossing*.

We create m equidistant portals on each side of each dissection square, the same way as in the Arora scheme. We call a solution (m, r)*-thin* if it is portal-respecting, and each portal is crossed at most r times by the paths of the solution. Note that this is different from the notion of (m, r)-light, where each side of a square of the quadtree is entered at most r times.

The following lemma shows that there is a near-optimal solution which is (m, r)-thin with some good parameters m, r, and which has no crossings.

▶ **Lemma 8.** *Let $(G, a, r, k = \infty, \kappa)$ be a well-rounded instance of* $\mathrm{AR_P}^\infty$ *with an optimal solution* $\mathrm{O{\small PT}}$*. Then, for an arbitrary $\epsilon > 0$, there is an $m = O_\epsilon(\log n)$ such that an optimal $(m, 2)$-thin solution* $\mathrm{O{\small PT}}'$ *to this instance which has no crossings has expected cost of at most* $(1 + \epsilon)\mathrm{opt}$.

Dynamic program. With this preparation we are ready to construct a dynamic program which computes a near-optimal solution for an instance of the $\mathrm{AR_P}^\infty$ problem.

▶ **Lemma 9.** *Let $(G, a, r, k = \infty, \kappa)$ be a well-rounded instance of* $\mathrm{AR_P}^\infty$*, and* $\mathrm{O{\small PT}}'$ *an optimal $(m, 2)$-thin solution for this instance which has no crossings. There is a dynamic program which computes in polynomial time a feasible solution for the problem with cost at most* opt'.

The idea of the dynamic program is as follows. We specify the DP cells by the following parameters for each square S of the quadtree, where Π denotes the set of portals of S.

1. A number κ' of paths entirely contained in S.
2. For each portal $\pi \in \Pi$, a number η_π of paths within S which use π and have one endpoint inside S.
3. A multiset P of pairs (π, π') of portals such that there is a path within S connecting π and π', and such that the portal pairs do not generate crossings within S.

The DP extends Arora's DP by adding the parameters κ' and η_π. The parameter κ' is needed in order to control the total number of paths in the solution. The parameters η_π are essential since they enable building airports within the squares of the quadtree. For each dynamic program cell $DP(S, \kappa', (\eta_\pi)_{\pi \in \Pi}, P)$, corresponding to a dissection square S and parameters κ', $(\eta_\pi)_{\pi \in \Pi}$ and P, the DP computes the value of a minimum cost solution for S which is consistent with the above parameters.

Proof of Theorem 2. Using Lemma 7 we can reduce our problem instance to a collection of well-rounded instances. By Lemma 8, each such instance I has an $(m, 2)$-thin solution without crossings with an expected cost of at most $(1 + \epsilon)\mathrm{opt}_I$. By Lemma 9, there is a dynamic program which computes a solution for each I with an expected cost of at most $(1 + \epsilon)\mathrm{opt}_I$. That gives us a PTAS for $\mathrm{AR_P}$ with $k = \infty$, where additionally we can enforce the required number κ of connected components. ◀

4 $\mathrm{AR_P}$ and $\mathrm{AR_F}$ with Uniform Airport Costs

For simplifying the presentation, in this section we assume that there is only one airport required per component (even for $\mathrm{AR_P}$). Note that this is without the loss of generality, since all the airport costs are identical and we can just scale the $\mathrm{AR_P}$ instance down by a factor of 2. In this section we describe how to subdivide any given instance in the two-dimensional

Euclidean space into two classes of independent subinstances, and how to derive a PTAS for each of these classes. We assume without loss of generality that all airports have unit cost: for positive cost we can achieve this by scaling both the railway and airport costs by the same factor; otherwise we obtain an optimal solution by opening airports at all the vertices.

4.1 Preprocessing: Subdividing the Instance

The goal of this subsection is to simplify the input instance. We will describe how any given instance can be partitioned into a collection of smaller subinstances, such that each subinstance I_i can be bounded by an $\ell_i \times \ell_i$ bounding box, with $\ell_i \leq 1/\epsilon$. Furthermore, we will show that it suffices to solve each such subinstance independently. We will introduce a shifted *main grid* with cells of width $1/\epsilon$, and cut the instance along this main grid. We show that one can find shifting offsets such that it suffices to consider each cell of the main grid as a separate instance.

Let the original instance be inside an $L \times L$ bounding box, and let the point $(0, 0)$ be at the bottom left corner of the box. We divide the instance into subinstances, each bounded by a square of size $1/\epsilon \times 1/\epsilon$, as follows. Let $0 \leq \alpha, \beta < 1/\epsilon$ be real numbers. We introduce horizontal lines with a y-coordinate of $\beta + i \cdot 1/\epsilon$ and vertical lines with an x-coordinate of $\alpha + i \cdot 1/\epsilon$, where $i = 0, \ldots, \epsilon \cdot \lfloor L \rfloor$. We say that this creates a *main grid* with a shift (α, β). This main grid splits the bounding box into squares of size at most $1/\epsilon \times 1/\epsilon$. The following lemma implies that, for the case of unit airport costs, it suffices to consider each of these squares separately. In other words, in contrast to Arora's scheme for the Euclidean TSP where the bounding box has sides of length $L = O_\epsilon(n)$, for us it is enough to consider subinstances where the bounding box has sides of length $1/\epsilon = O_\epsilon(1)$.

▶ **Lemma 10.** *Let S be a solution of cost s to an instance I of either $1\mathrm{AR_F}$ or $1\mathrm{AR_P}$. Then there exists a shift (α, β) of the main grid defined above, and a solution S' of cost s' to I, obtained from S by removing some edges and adding new airports to maintain feasibility, such that* (i) *$s' \leq (1 + 2\epsilon)s$, and* (ii) *S' does not cross any line of the shifted main grid.*

Furthermore, we can compute in polynomial time a collection of shifts (α_i, β_i) such that at least one of them satisfies the above conditions.

For the rest of the section we may restrict our discussion to instances I that are within an $\ell \times \ell$ bounding square, with $\ell \leq 1/\epsilon$.

4.2 A PTAS for Sparse Subinstances

When dividing the original instance into subinstances, some subinstances may have optimal solutions with only a small number of components (at most $\frac{1}{\epsilon^7}$). This kind of instances can be handled with a slight adaption of Arora's scheme, as has already been observed by Asano et al. [6] for the capacitated vehicle routing problem. It can be easily seen that the adaptions of Arora's PTAS performed by Asano et al. [6] for CVRP can also be applied in our case.

4.3 A PTAS for Dense Subinstances

This subsection copes with the subinstances that are not covered above, i. e., for which any optimal solution has a large number of connected components (more than $\frac{1}{\epsilon^7}$). The idea is to start with a solution for $\mathrm{AR_F}^\infty$ ($\mathrm{AR_P}^\infty$, respectively), and appropriately divide each connected component of the solution into *chunks/subpaths*, of roughly ϵk points each. We show that there exists a solution, not much more expensive than the optimal one, that contains all the edges within chunks/subpaths. Since in each such solution the number of edges between different chunks/subpaths is a small constant for each component, and each component has a total cost of at least 1 (due to the airport cost), we can connect the chunks/subpaths in a way that keeps the increase in cost within the limits.

Our algorithm for handling the dense subinstances consists of the following steps.

Step 1: Creating a grid. We partition the $\ell \times \ell$ bounding box into **cells**, by creating an ϵ^2-grid. Note that by Lemma 10 we have $\ell \leq 1/\epsilon$, and therefore the total number of cells is bounded from above by $1/\epsilon^6$.

Step 2: Splitting components. In this step we construct a solution OPT' which is not necessarily feasible, but which will be needed for Step 3. The construction of OPT' assumes that we know the exact number X of components in OPT. This is without the loss of generality, as we can run the algorithm for all choices of X (from 1 to n) and output the minimum cost solution obtained.

AR_F: We run an exact algorithm for $\text{AR}_\text{F}{}^\infty$ (see Theorem 4) with $\kappa = X$. This procedure returns a forest $F = \{T_1, T_2, \ldots, T_X\}$.

AR_P: We run a PTAS for $\text{AR}_\text{P}{}^\infty$ (see Theorem 2) with $\kappa = X$. This procedure returns a collection of X paths.

Note that in both cases $\text{opt}' \leq (1+\epsilon)\text{opt}$, by Theorems 2 and 4.

Step 3: Cutting into chunks/subpaths.

AR_F: For each tree $T_i \in F$, we partition it into chunks by calling a procedure called $\text{GetChunks}(T_i)$, described below. As we will see in Lemma 11, each of the returned chunks will span least ϵk and at most $6\epsilon k$ vertices, except perhaps one smaller chunk per tree.

AR_P: For each path of the solution returned by the PTAS used in Step 2, we split the path into subpaths of ϵk points each, except perhaps one shorter subpath per component.

Step 4: Associate chunks with cells.

AR_F: We associate each chunk c_i with a cell, denoted by $\text{cell}(c_i)$. We do this by selecting an arbitrary vertex $p \in c_i$, and associating c_i with the cell in which p lies.

AR_P: We associate each of the two endpoints of each subpath with the cell that contains it.

Step 5: Assembling chunks/subpaths into a solution.

AR_F: For each cell c we repeatedly collect arbitrary chunks associated with c to create a single connected component, until we obtain a component of size between $(1-6\epsilon)k$ and k, or until we run out of chunks. We connect these chunks into one connected component by adding edges within c, connecting any two vertices of the different chunks.

AR_P: We start with a cell that contains an endpoint of a not yet considered subpath, follow the subpath to its other endpoint v and connect it to an endpoint v' of some other unconsidered subpath that is associated with $\text{cell}(v)$ (if such a v' exists). We do this until either there is no other subpath starting at $\text{cell}(v)$, or the current component has between $(1-\epsilon)k$ and k many vertices. We repeat the above as long as there are unconsidered subpaths in the instance.

We now describe the algorithm $\text{GetChunks}(T)$ (Algorithm 1) that splits a tree T into chunks. The *height* $h(v)$ of a vertex v is defined as the minimum number of edges on a path between v and a leaf (if v is a leaf then $h(v) = 0$). For some vertex v, let *subtree*(v, T) be the (downward) subtree of T rooted at v, and let $\text{childOf}(v, T)$ be the set of children of v in T. Processing the vertices of the tree bottom-up, the algorithm identifies an internal vertex v such that the subtree of T rooted at v contains at least ϵk and at most $6\epsilon k$ vertices. In

```
Algorithm 1: GetChunks($T$)
  Input: A tree $T$ rooted at $r$
  Output: A set of chunks
          comprising $T$
  $CH := \emptyset$;
  while $T \neq \emptyset$ do
      $chunk :=$ Cut-A-Chunk($T$);
      $CH := CH \cup \{chunk\}$;
      $T := T \setminus chunk$;
  end
  return $CH$
```

```
Algorithm 2: Cut-A-Chunk($T$)
  Input: A tree $T$ rooted at $r$
  Output: A subtree of $T$
  for every vertex $v$ of $T$ do
      value($v$) := 1
  end
  $i := 0$;
  while $|T| > 6\epsilon k$ do
      if there exists a vertex $v$ with $h(v) = i$
       and $value(v) \geq \epsilon k$ then
          return $subtree(v, T)$
      else
          for every vertex $v$ with $h(v) = i + 1$
           do
              $value(v) :=$
               $\sum_{v' \in \text{childOf}(v,T)} value(v') + 1$
          end
      end
      $i := i + 1$;
  end
  return $T$
```

Lemma 11 we will show that such a vertex v always exists (its existence can be derived by the fact that kissing number in the Euclidean plane is 6). The corresponding subtree rooted at v is then removed from T and the whole process is repeated. The algorithm ends when the whole tree T contains at most $6\epsilon k$ vertices, and this last chunk may contain less than ϵk many vertices.
We show the following property of the solutions output by the algorithm GetChunks(T).

▶ **Lemma 11.** *The algorithm* GetChunks(T) *outputs a partition of the tree T into chunks of size in the interval $[\epsilon k, 6\epsilon k]$, and possibly one smaller chunk.*

We are now ready to prove the following result.

▶ **Lemma 12.** *The algorithm presented in Steps* 1-5 *above is a PTAS for dense instances of 1*AR_F *and 1*AR_P*.*

Proof. We first observe that all the steps of the algorithm run in polynomial time.

Denote by Opt_F (Opt_P) an optimal solution for a given instance of 1AR_F (1AR_P, respectively). Let Alg_F (Alg_P) be the solutions output by the algorithm, and Alg'_F (Alg'_P) the intermediate, and possibly infeasible, solutions constructed in Step 2 of the algorithm for 1AR_F (1AR_P, respectively). When a statement applies to both AR_P and AR_F then we skip the subscript P and F. Recall that $\tilde{a}(S)$ and $r(S)$ refer to the airport cost (which is equal to the number of connected components of S), and the edge cost of a solution S.

First, consider Step 2 of the algorithm and the respective solutions Alg'_F and Alg'_P. As Opt_F (Opt_P) is a feasible solution for the $\text{AR}_\text{F}{}^\infty$ ($\text{AR}_\text{F}{}^\infty$, respectively) problem, and in Step 2 the algorithm outputs an exact solution for $\text{AR}_\text{F}{}^\infty$ (a $(1+\epsilon)$-approximate solution for

$\mathrm{AR_P}^\infty$, respectively), we obtain $\mathrm{alg}' \le (1+\epsilon)\mathrm{opt}$. As we assumed that $\textsc{Alg}'$ has the same number of connected components as $\textsc{Opt}$ (i.e., $\tilde{a}(\textsc{Alg}') = \tilde{a}(\textsc{Opt})$), we get

$$r(\textsc{Alg}') \le (1+\epsilon) r(\textsc{Opt}) + \epsilon \tilde{a}(\textsc{Opt}) \ . \tag{1}$$

In Step 3 the algorithm removes some edges from the solution $\textsc{Alg}'$, splitting the connected components into chunks (subpaths). We now turn our attention to Step 5 of the algorithm, i.e., reassembling the chunks (subpaths) into a solution for $1\mathrm{AR_F}$ ($1\mathrm{AR_P}$). In this step we repeatedly merge arbitrary chunks (subpaths) until we end up with a connected component of size between $(1-6\epsilon)k$ and k ($(1-\epsilon)k$ and k, respectively), or until we run out of chunks (subpaths) in the current cell c. Since we can only run out of chunks (subpaths) once per cell, we know that we can have at most $1/\epsilon^6$ components with a size less than $(1-6\epsilon)k$ in the final solution. As we are in the unit airport cost setting and we consider a dense problem instance, we have $\tilde{a}(\textsc{Opt}) > 1/\epsilon^7$, which gives us

$$\tilde{a}(\textsc{Alg}) \le (1 + O(\epsilon))\tilde{a}(\textsc{Opt}) \ . \tag{2}$$

By Lemma 11 and Step 5 of the algorithm, no component in the resulting solutions for $\mathrm{AR_F}$ and $\mathrm{AR_P}$ has size more than k. This, along with the structure of the constructed solutions (the assembled chunks form a tree, and the assembled subpaths form a path), implies the feasibility of the solution output by the algorithm.

The assembling in Step 5 is performed by adding at most one new edge per each chunk (subpath), and each added edge has cost of at most $\sqrt{2}\epsilon^2$. As each connected component of $\textsc{Alg}'$ has been split into at most $1/\epsilon$ chunks (subpaths), that gives us

$$r(\textsc{Alg}) \le r(\textsc{Alg}') + \epsilon\sqrt{2}\tilde{a}(\textsc{Alg}') = r(\textsc{Alg}') + \epsilon\sqrt{2}\tilde{a}(\textsc{Opt}) \ . \tag{3}$$

The above inequalities imply the lemma, because

$$\begin{aligned}\mathrm{alg} = \tilde{a}(\textsc{Alg}) + r(\textsc{Alg}) &\le (1+O(\epsilon))\tilde{a}(\textsc{Opt}) + r(\textsc{Alg}) \\ &\le (1+O(\epsilon))\tilde{a}(\textsc{Opt}) + r(\textsc{Alg}') \le (1+O(\epsilon))\mathrm{opt}.\end{aligned}$$

◀

4.4 A PTAS for $1\mathrm{AR_F}$ and $1\mathrm{AR_P}$

We will now show how the above results can be combined in order to prove Theorem 5.

Proof of Theorem 5. Given an arbitrary instance of $1\mathrm{AR_F}$ or $1\mathrm{AR_P}$ we first apply the preprocessing step (see Lemma 10). The problem instance is split into a collection of independent subinstances, such that each subinstance I is contained in a bounding box of size $1/\epsilon \times 1/\epsilon$. Each instance is either sparse or dense. Since we do not know which subinstances are sparse and which are dense (as this definition depends on the structure of an optimal solution for I), we will run both the algorithm for sparse instances (i.e., the PTAS from Section 4.2) and the algorithm for dense instances (i.e., the PTAS from Section 4.3) on each subinstance, and return the solution with lower cost for each subinstance. That will yield a PTAS for the original input instance. ◀

5 Open Problems

Two-dimensional Euclidean setting. In this paper we resolved the complexity of the $\mathrm{AR_P}^\infty, \mathrm{AR_F}^\infty, 1\mathrm{AR_F}$ and $1\mathrm{AR_P}$ problems in the two-dimensional Euclidean setting. It would be interesting to know how well the general $\mathrm{AR_P}$ and $\mathrm{AR_F}$ problems can be approximated in this setting. Another open problem is generalizing the PTAS for $1\mathrm{AR_F}$ and $1\mathrm{AR_P}$ to work for more general families of airport costs. In particular, as stated earlier in this paper, a generalization of the PTAS for $1\mathrm{AR_P}$ to a constant range of airport costs would yield a PTAS for Euclidean CVRP, which is an important open problem.

Other metrics. Another interesting question is getting positive and negative approximability results for the $\mathrm{AR_P}$ and $\mathrm{AR_F}$ problems in other metrics, for example for planar graphs, H-minor free graphs, or for metric graphs.

Other problems in the Airport and Railway framework. Another open question is to investigate other problems in the Airport and Railway framework introduced in our paper. It is not difficult to come up with several different specific requirements on the connected components, that model natural scenarios.

Acknowledgment. We could like to thank the anonymous reviewers for suggesting the use of an auxiliary vertex which simplified the proof of Theorem 4.

References

1 Anna Adamaszek, Artur Czumaj, and Andrzej Lingas. PTAS for k-tour cover problem on the plane for moderately large values of k. *Int. J. Found. Comput. Sci.*, 21(6):893–904, 2010. URL: `http://dx.doi.org/10.1142/S0129054110007623`, `doi:10.1142/S0129054110007623`.

2 Anna Adamaszek, Artur Czumaj, Andrzej Lingas, and Jakub Onufry Wojtaszczyk. Approximation schemes for capacitated geometric network design. In *Automata, Languages and Programming – 38th International Colloquium, ICALP 2011, Zurich, Switzerland, July 4-8, 2011, Proceedings, Part I*, pages 25–36, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-22006-7_3`, `doi:10.1007/978-3-642-22006-7_3`.

3 Sanjeev Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

4 Sanjeev Arora. Approximation schemes for NP-hard geometric optimization problems: a survey. *Math. Program.*, 97(1–2):43–69, 2003.

5 Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*, pages 106–113, 1998. URL: `http://doi.acm.org/10.1145/276698.276718`, `doi:10.1145/276698.276718`.

6 Tetsuo Asano, Naoki Katoh, Hisao Tamaki, and Takeshi Tokuyama. Covering points in the plane by k-tours: Towards a polynomial time approximation scheme for general k. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 275–283, 1997. URL: `http://doi.acm.org/10.1145/258533.258602`, `doi:10.1145/258533.258602`.

7 Marshall Wayne Bern and David Eppstein. Approximation algorithms for geometric problems. In Dorit Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, chapter 8, pages 296–345. PWS Publishing, 1996.

8 G.B. Dantzig and Ramser J.H. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

9 Aparna Das and Claire Mathieu. A quasipolynomial time approximation scheme for euclidean capacitated vehicle routing. *Algorithmica*, 73(1):115–142, 2015. URL: `http://dx.doi.org/10.1007/s00453-014-9906-4`, `doi:10.1007/s00453-014-9906-4`.

10 Friedrich Eisenbrand, Fabrizio Grandoni, Thomas Rothvoß, and Guido Schäfer. Connected facility location via random facility sampling and core detouring. *J. Comput. Syst. Sci.*, 76(8):709–726, 2010. URL: `http://dx.doi.org/10.1016/j.jcss.2010.02.001`, `doi:10.1016/j.jcss.2010.02.001`.

11 Inge Li Gørtz and Viswanath Nagarajan. Locating depots for capacitated vehicle routing. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques – 14th International Workshop, APPROX 2011, and 15th International Workshop, RANDOM 2011, Princeton, NJ, USA, August 17-19, 2011. Proceedings*, pages 230–241, 2011. URL: `http://dx.doi.org/10.1007/978-3-642-22935-0_20`, `doi:10.1007/978-3-642-22935-0_20`.

12 Tobias Harks, Felix G. König, and Jannik Matuschke. Approximation algorithms for capacitated location routing. *Transportation Science*, 47(1):3–22, 2013. URL: `http://dx.doi.org/10.1287/trsc.1120.0423`, `doi:10.1287/trsc.1120.0423`.

13 Raja Jothi and Balaji Raghavachari. Approximation algorithms for the capacitated minimum spanning tree problem and its variants in network design. *ACM Transactions on Algorithms (TALG)*, 1(2):265–282, 2005.

14 Christos H Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18(1):1–11, 1993.

15 R. Ravi and Amitabh Sinha. Approximation algorithms for problems combining facility location and network design. *Operations Research*, 54(1):73–81, 2006. URL: `http://dx.doi.org/10.1287/opre.1050.0228`, `doi:10.1287/opre.1050.0228`.

16 Alexander Schrijver. *Combinatorial Optimization.* Springer, 2003.

17 Chaitanya Swamy and Amit Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica*, 40(4):245–269, 2004. URL: `http://dx.doi.org/10.1007/s00453-004-1112-3`, `doi:10.1007/s00453-004-1112-3`.