Obnoxious Centers in Graphs

Sergio Cabello* Günter Rote†

Abstract

We consider the problem of finding obnoxious centers in graphs. For arbitrary graphs with n vertices and m edges, we give a randomized algorithm with $O(n \log^2 n + m \log n)$ expected time. For planar graphs, we give algorithms with $O(n \log n)$ expected time and $O(n \log^3 n)$ worst-case time. For graphs with bounded treewidth, we give an algorithm taking $O(n \log n)$ worst-case time. The algorithms make use of parametric search and several results for computing distances on graphs of bounded treewidth and planar graphs.

1 Introduction

A central problem in locational analysis deals with the placement of new facilities that optimize a given objective function. In the obnoxious center problem, there is set of sites in some metric space, each with its own weight, and we want to place a facility that maximizes the minimum of the weighted distances from the given sites. The problem arises naturally when considering the placement of an undesirable facility that will affect the environment, or, in a dual setting, when searching for a place away from existing obnoxious facilities. Algorithmically, obnoxious facilities have received much attention previously; see [1, 6, 12, 21, 22, 23] and references therein.

In this paper, we consider the problem of placing a single obnoxious facility in a graph, either at its vertices or along its edges; this is often referred to as the continuous problem, as opposed to the discrete version, where the facility has to be placed in a vertex of G. A formal definition of the problem is given in Section 2.1. We use n, m for the number of vertices and edges of G, respectively.

Previous results. Subquadratic algorithms are known for the obnoxious center problem in trees and cacti. Tamir [21] gave an algorithm with $O(n \log^2 n)$ worst-case time for trees. There are faster algorithms for some special classes of trees [6, 21]. For cactus graphs, Zmazek and Žerovnik [23] gave an algorithm using $O(cn)$ time, where c is the number of different weights in the sites, and recently Ben-Moshe, Bhattacharya, and Shi [1] showed an algorithm using $O(n \log^3 n)$ time.

For general graphs, Tamir [20] showed how to solve the obnoxious center problem in $O(nm + n^2 \log n)$ time. We are not aware of other works for special classes of graphs. However, for planar graphs, it is easy to use separators of size $O(\sqrt{n})$ [14] to solve the problem in roughly $O(n^{3/2})$ time.

Our results. In general, we follow an approach similar to Tamir [21], using the close connection between the obnoxious center problem and the following covering problem: do a set of disks cover a graph? See Section 2.2 for a formal definition. A summary of our results is as follows:

- A covering problem in G can be solved constructing a shortest path tree in an augmented graph obtained by adding an apex to G. See Section 3.

- We give a randomized algorithm to find an obnoxious center in $O(m \log n + n \log^2 n)$ expected time. The best previous algorithm used $O(nm)$ worst-case time [20]. See Section 3.

- For graphs with bounded treewidth, we give an algorithm to find an obnoxious center in $O(n \log n)$ worst-case time. Previously, only algorithms for trees (graphs with treewidth one) were known, and they used $O(n \log^2 n)$ time [21]. See Section 4.

- For planar graphs, we give two algorithms to find an obnoxious center: one taking $O(n \log n)$ expected time and one taking $O(n \log^3 n)$ worst-case time. The best previous algorithm used roughly $O(n^{3/2})$ time, as discussed above. See Section 5.

A main difficulty in the obnoxious center problem is that it may have many local optima, since the objective depends on the *closest* neighbors of the placement. This is in contrast to the classical center problem, where we want to minimize the *maximum* weighted distance to the given sites. Thus, pruning techniques

*Department of Mathematics, IMFM, and Department of Mathematics, FMF, University of Ljubljana, Slovenia, sergio.cabello@fmf.uni-lj.si. Partially supported by the European Community Sixth Framework Programme under a Marie Curie Intra-European Fellowship, and by the Slovenian Research Agency, project J1-7218-0101.

†Freie Universität Berlin, Institut für Informatik, Takustraße 9, 14195 Berlin, Germany, rote@inf.fu-berlin.de.

like Megiddo's [17] solution to the classical problem do not seem fruitful here.

Randomized algorithms have not been considered previously in the context of obnoxious centers. Our randomized algorithm for general graphs is simple and easy to program, since it only uses linear programs in two variables and shortest paths in graphs, and it already improves the previous best bound by a factor of $n/\log n$.

Our approach for graphs with bounded treewidth is based on parametric search [15, 16]. However, an interesting point is our use of Cole's [9] speed-up technique: instead of applying it to a sorting network, as it is most common, we use it in a network arising from a tree decomposition of the graph. To make this approach fruitful and remove a logarithmic factor in the running time, we employ an alternative tree decomposition with logarithmic depth, but larger width. For example, we improve the previous running time for trees by considering a tree decomposition of width five and logarithmic depth.

Our randomized algorithm for planar graphs uses the shortest path algorithm by Henzinger et al. [11]. Our deterministic algorithm for planar graphs is based on the results and techniques developed by Fakcharoenphol and Rao [10] and Klein [13] for computing several shortest paths in planar graphs.

2 Preliminaries

2.1 Obnoxious Centers Let G be an undirected graph with n vertices, with a function $w\colon V(G) \to \mathbb{R}_+$ assigning positive *weights* to the vertices of G and a function $\ell\colon E(G) \to \mathbb{R}_+$ assigning lengths to the edges of G. We assume that w and ℓ are part of the graph G. The lengths of the edges naturally define a distance function $\delta_G\colon V(G) \times V(G) \to \mathbb{R}_+$, where $\delta_G(u,v)$ is the minimum length of all walks in G from u to v.

The continuous center problem allows the center to be placed *on* an edge: we regard each edge $e = uv \in E(G)$ as a curve $A(e)$ of length $\ell(e)$ between u and v, containing a point at distance λ from u (and at distance $\ell(e) - \lambda$ from v), for every λ in the range $0 < \lambda < \ell(e)$. We denote by $A(G)$ the set of all points on all edges and vertices of G. We will use the notations $A(e)$ and $A(G)$ in order to emphasize that we mean the continuous set of points on the graph, as opposed to the edge e and the graph G as a discrete object. The distance function δ_G can be extended from the vertices to $A(G)$ in the natural way. When the graph G is understood and there is no possible confusion, we use δ instead of δ_G.

We can now define an objective function $\mathrm{COST}\colon A(G) \to \mathbb{R}_+$ as

$$\mathrm{COST}(a) = \min_{v \in V(G)} \{w(v) \cdot \delta(a,v)\},$$

which, for a point a, measures the weighted closest site vertex from a. Note that in this setting, the larger the weight, the less relevant is the point. In particular, if a vertex is irrelevant, then its weight is $+\infty$.[1] An *obnoxious center* is a point $a^* \in A(G)$ such that $\mathrm{COST}(a^*) = \max_{a \in A(G)} \mathrm{COST}(a)$.

2.2 Covering Problem and Decision Problem Let $D(v,r) = \{a \in A(G) \mid \delta(a,v) \le r\}$ denote the close disk with radius $r \ge 0$ and center $v \in V(G)$. Given radii r_v for all $v \in V(G)$, consider the following *covering problem*: does $\bigcup_{v \in V(G)} D(v, r_v)$ cover $A(G)$, or equivalently, is $A(G) = \bigcup_{v \in V(G)} D(v, r_v)$? We use $(G, \{(v, r_v) \mid v \in V(G)\})$ to represent an instance to the covering problem.

The decision problem associated to the obnoxious center problem asks, for a given value t, if $t \ge \mathrm{COST}(a^*)$. The decision problem corresponds to a covering problem where the radii of the disks are a function of the value t. To make this relation precise, we think of the disks as growing around their centers v with speed $1/w(v)$, and we define the union $\mathcal{U}(t) = \bigcup_{v \in V(G)} D(v, t/w(v))$ of the disks at time t. We have $\mathcal{U}(t) = \{\, a \in A(G) \mid \mathrm{COST}(a) \le t \,\}$, and therefore we obtain the following connection to obnoxious centers.

Lemma 2.1. *Let a^* be an obnoxious center of G. The optimum value of the objective function is given by*

$$\mathrm{COST}(a^*) = t^* := \min\{\, t \in \mathbb{R}_+ \mid \mathcal{U}(t) = A(G) \,\}. \quad \square$$

3 General Graphs

Our running times will be expressed as a function of $T_{\mathrm{sssp}}(G)$, the time needed to solve a single source shortest path problem in graph G with nonnegative edge lengths. It is well known that if G has n vertices and m edges, then $T_{\mathrm{sssp}}(G) = O(n \log n + m)$ time. Better results are known for some special classes of graphs.

Consider a covering instance $(G, \{(v, r_v) \mid v \in V(G)\})$. A useful concept for our subsequent discussion is the *coverage* $C(v)$ of a vertex $v \in V(G)$, defined as

$$C(v) = \max\{\, r_u - \delta(u,v) \mid u \in V(G) \,\}.$$

[1] Other authors use a different setting, namely assuming negative weights at the vertices and defining the cost as the maximum of the weighted distances. It is easy to see that these two models are equivalent.

Intuitively, the coverage of v is the maximum remaining "covering capacity" when trying to cover the graph by paths that pass through v. The relevance of coverages is reflected in the following observation.

LEMMA 3.1. *For an $e = uv \in E(G)$, the edge $A(e)$ is covered by $\bigcup_{v \in V(G)} D(v, r_v)$ if and only if $\ell(e) \le C(u) + C(v)$.* □

For any graph G, we define the graph G_+ as $(V(G) \cup \{s\}, E(G) \cup \{sv \mid v \in V(G)\})$, that is, G_+ is obtained from G by adding a new "apex" vertex s adjacent to all vertices $V(G)$. We will now show that all coverages $C(v)$ can be computed by a single-source shortest path computation in G_+. We define an upper bound $L = \max\{n \cdot \ell_{\max}\}$ on any shortest path in G, where $\ell_{\max}$ is the length of the longest edge in G. Henceforth, we assume that $r_v \le L$, $v \in V(G)$, as otherwise it is clear that G is covered.

Consider the graph G_+ where each edge already existing in G keeps the same length and each edge of the form sv has length $2L - r_v$. We have chosen the edges adjacent to s long enough such that the distance between two vertices $u, v \in V(G)$ is the same in G and in G_+, that is $\delta_G(u, v) = \delta_{G_+}(u, v)$ for any $u, v \in V(G)$.

LEMMA 3.2. *The coverages $C(v)$ in G are related to the distances from s in G_+ as follows*:

$$C(v) = 2L - \delta_{G_+}(s, v). \qquad \square$$

Combining Lemmas 3.1 and 3.2, we achieve the following:

PROPOSITION 3.1. *We can solve the covering problem in a graph G in $O(T_{\mathrm{sssp}}(G_+))$ time.* □

To study the relation to obnoxious centers, we need the coverage $C(v, t)$ as a function of $t \ge 0$,

$$C(v, t) = \max\{\, t/w(u) - \delta(u, v) \mid u \in V(G) \,\}.$$

This is an increasing piecewise linear function in t. For an edge $e = uv \in E(G)$, let t_e be the unique value satisfying $\ell(e) = C(u, t_e) + C(v, t_e)$. This is the first time when the edge $A(e)$ becomes covered, that is, $t_e = \min\{\, t \in \mathbb{R}_+ \mid A(e) \subset \mathcal{U}(t) \,\}$.

LEMMA 3.3. *The values $t_e, e \in E(G)$, have the following properties:*

1. *$t^* = \max\{\, t_e \mid e \in E(G) \,\}$;*
2. *for any two edges e, e', we have $t_e \le t_{e'}$ if and only if $A(e) \subset \mathcal{U}(t_{e'})$.* □

LEMMA 3.4. *For any edge $e = xy \in E(G)$, we can compute t_e in $O(T_{\mathrm{sssp}}(G))$ time.*

Proof. We parameterize the edge $A(e)$ by the distance λ from x ($0 \le \lambda \le \ell(e)$). Then the objective function value $\mathrm{COST}(a)$ of the point a with parameter λ is the minimum of the $2n$ linear functions

$$\{\, w(v) \cdot (\delta(v, x) + \lambda) \mid v \in V(G) \,\}$$
$$\cup \{\, w(v) \cdot (\delta(v, y) + \ell(e) - \lambda) \mid v \in V(G) \,\}$$

The set of distances $\delta(v, x)$ and $\delta(v, y)$ can be computed for all v by solving two shortest path problems with sources x and y, respectively, in $O(T_{\mathrm{sssp}}(G))$ time. The value λ that maximizes the lower envelope of $2n$ linear functions can then be found in $O(n)$ time as a linear programming problem in two variables [17]. □

THEOREM 3.1. *For a graph G with n vertices, the algorithm Obnoxious-Center-Randomized in Figure 1 finds an obnoxious center in $O(T_{\mathrm{sssp}}(G_+) \log n)$ expected time.*

Proof. Correctness is clear from Lemma 3.3: in steps 6–7, we exclude the edges e' with $t_{e'} \le t_{e_i}$. Thus, we compute increasing values $t_1, t_2, \ldots$ from $\{\, t_e \mid e \in E(G) \,\}$, and we maintain the invariant $E_i = \{\, e \in E(G) \mid t_e > t_i \,\}$. Therefore, when $E_i = \emptyset$ we have $t^* = t_i = \max\{t_e \mid e \in E(G)\}$, and it is clear that the edge e_i contains an obnoxious center. To bound the running time, we first show that the while-loop in lines 2–7 is iterated an expected number of $O(\log |E(G)|) = O(\log n)$ times. Indeed, if I_n denotes the expected number of remaining iterations when $|E_i| = n$, then we have the recurrence $I_n = 1 + \frac{1}{n} \sum_{i=1}^{n-1} I_i$, $I_1 = 1$, which solves by induction to

$$I_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} \le (1 + \ln n).$$

Finally, note that each iteration of the loop in lines 2–7 takes $O(T_{\mathrm{sssp}}(G_+))$ time because of Lemmas 3.4, 3.2, and 3.1. □

If G has n vertices and m edges, then G_+ has $O(n)$ vertices and $O(n+m)$ edges, and therefore $T_{\mathrm{sssp}}(G_+) = O(n \log n + m)$. Using the previous lemma we conclude the following.

COROLLARY 3.1. *For graphs with n vertices and m edges, we can solve the obnoxious center problem by a randomized algorithm in $O(n \log^2 n + m \log n)$ expected time.* □

An approach to obtain a deterministic algorithm that finds an obnoxious center would be to use parametric

Algorithm *Obnoxious-Center-Randomized*
Input: A graph G
Output: Computes t^* and finds an obnoxious center
1. $i \leftarrow 0$; $E_0 \leftarrow E(G)$;
2. **while** $E_i \neq \emptyset$
3. $\quad i \leftarrow i+1$;
4. $\quad e_i \leftarrow$ random edge in E_i;
5. $\quad$ compute t_{e_i} by Lemma 3.4;
6. $\quad E' \leftarrow \{e' \in E \mid A(e') \subset \mathcal{U}(t_{e_i})\}$;
7. $\quad E_i \leftarrow E_{i-1} \setminus E'$;
8. find the best point a in $A(e_i)$; $\quad$ (* $A(e_i)$ contains an obnoxious center *)
9. **return** t_{e_i} as t^* and a as an obnoxious center;

Figure 1: Algorithm *Obnoxious-Center-Randomized*

search [16], based on a parallel algorithm for the decision problem, i. e., the covering problem. However, the parallel algorithms that are known for single source shortest path do not provide any improvement over the current $O(nm)$ time bound by Tamir [20].

4 Graphs with Bounded Treewidth

4.1 Tree Decompositions and Treewidth We review some basic properties of tree decompositions and treewidth [3, 4].

Definition 1. *A* tree decomposition *of a graph G is a pair (X,T), with a collection $X = \{X_i \mid i \in I\}$ of subsets of $V(G)$ (called* bags*), and a tree $T = (I,F)$ with node set I, such that*

- $V(G) = \bigcup_{i \in I} X_i$;
- *For every $e = uv \in E(G)$, there is some bag X_i such that $u,v \in X_i$;*
- *For all $v \in V(G)$, the nodes $\{i \in I \mid v \in X_i\}$ form a connected subtree of T.*

The width *of a tree decomposition $(\{X_i \mid i \in I\}, T)$ is $\max_{i \in I} |X_i| - 1$. The* treewidth *of G is the minimum width over all tree decompositions of G.*

We will use the term *vertices* for the elements of $V(G)$ and *nodes* for the elements of $V(T)$.

For graphs with bounded treewidth, Bodlaender [2] gives an algorithm to construct in linear time a tree decomposition of minimum width. Furthermore, Bodlaender and Hagerup [5] show that the tree can be assumed to be a binary tree of height $O(\log n)$, at the expense of a constant factor in terms of the width:

Lemma 4.1. *Let k_0 be a fixed constant. For graphs with n vertices and treewidth at most k_0, we can construct in linear time a tree decomposition (X,T) of width at most $3k_0+2$, whose tree T is a rooted binary tree of height $O(\log n)$ with $O(n)$ nodes.*

In fact, for our solution to the obnoxious center problem, we will spend $O(n \log n)$ time, but we only need to construct *once* a tree-decomposition as described in Lemma 4.1. Therefore, we could replace Bodlaender's algorithm [2] by Reed's algorithm [19], which takes $O(n \log n)$ time but is simpler.

Chaudhuri and Zaroliagis [8, Lemma 3.2] have shown that all distances between pairs of vertices in the same bag can be computed in linear time (even if negative edges are permitted):

Lemma 4.2. *Let (X,T) be a tree decomposition of width k for a graph G with n vertices. Then the distances $\delta_G(u,v)$ for all pairs of vertices u,v that belong to a common bag X_i can be calculated in $O(k^3 n)$ time.*

4.2 A Decision Algorithm and a Parametric Search Algorithm Tamir [21] showed that the coverage problem is solvable in $O(n)$ when the graph is a tree. We will now generalize this result to graphs with bounded treewidth. Note that if G has treewidth at most k_0, then G_+ has treewidth at most k_0+1: from a tree decomposition for G of width k_0 we can obtain a tree decomposition for G_+ of width k_0+1 by adding the special vertex s to all bags. Since Chaudhuri and Zaroliagis [8] showed that a shortest path tree in graphs with bounded treewidth can be constructed in linear time, Proposition 3.1 leads to the following result.

LEMMA 4.3. *Let k_0 be a fixed constant. For a graph with n vertices and treewidth at most k_0, we can solve the covering problem in $O(n)$ time.* $\square$

Theorem 3.1 gives a randomized algorithm with $O(n \log n)$ expected time for graphs with bounded treewidth. We next show how to achieve the same time bound deterministically. The approach is to use a modification of the parallel algorithm by Chaudhuri and Zaroliagis [7] for computing a shortest path tree in parallel. Moreover, our modification also applies the technique of Cole [9], to obtain a speed-up when later applying parametric search [16]. This leads to an algorithm using $O(n \log n)$ time in the worst case. We next provide the details.

The idea of the algorithm is to utilize the structure of the tree-decomposition to construct in G_+ a shortest path tree from s. First we compute the distances between all pairs of vertices in the same bag. After that, we can compute shortest paths from s in an upward sweep along T followed by a downward sweep. We compute shortest paths (as most algorithms do) by maintaining vertex labels $d(v)$ and carrying out a sequence of *relaxation* operations

$$d(v) := \min\{d(v), d(u) + \ell(u, v)\}.$$

Classically, $\ell(u, v)$ is the length of the edge uv. However, we will apply this operation to two vertices u, v belonging to the same bag, and we will use the precomputed distance $\delta_G(u, v) = \delta_{G_+}(u, v)$ in G:

$$d(v) := \min\{d(v), d(u) + \delta_G(u, v)\}. \tag{1}$$

The algorithm *Decision-Tree-Width* in Figure 3 is complicated and slower than our previous approach, but it is better for the parametric search framework. It uses a directed graph H given by

$$\begin{aligned} V(H) &= \{\, (u, v, i) \mid i \in I,\ u, v \in X_i,\ u \neq v \,\}, \\ E(H) &= \{\, ((u, v, i), (u', v', j)) \mid u, v \in X_i,\ u', v' \in X_j, \\ &\qquad j \text{ parent of } i \text{ in } T \,\}. \end{aligned}$$

Each vertex $(u, v, i) \in V(H)$ is identified with the relaxation $d(v) = \min\{d(v), d(u) + \delta_G(u, v)\}$ (1) that has to be made when considering the bag X_i, see Figure 2. An edge $((u, v, i), (u', v', j))$ in H indicates some order in which relaxations (u, v, i) and (u', v', j) have to take place: in the bottom-up part (lines 8–18), we always perform relaxation (u, v, i) before (u', v', j), and in the top-down part (lines 20–30) we always perform relaxation (u', v', j) before (u, v, i). Therefore, in the bottom-up part (lines 8–18), a relaxation (u, v, i) is performed only when the relaxations of its predecessors $\Gamma^-(u, v, i)$ in H have been performed, and an analogous statement holds for the top-down part with respect to the successors $\Gamma^+(u, v, i)$. The algorithm maintains the set A of *active* relaxations, from which a subset A' is selected for execution (lines 13 and 25). When the algorithm is carried out within the framework of parametric search, the selection of A' is beyond the control of the algorithm; it is only guaranteed that the total weight of the executed relaxations is at least half of the weight of all active relaxations. The correctness of the algorithm does not depend on the order in which the relaxations in lines 15 and 27 are carried out. Note that the same pair u, v can be relaxed several times during one sweep, as part of different bags X_i. The correctness of the algorithm follows from tools commonly used in tree-width; here we only state it.

LEMMA 4.4. *The algorithm Decision-Tree-Width correctly decides if $\mathcal{U}(t)$ covers $A(G)$.* $\square$

Note that the value of L computed in line 3 is actually completely irrelevant. Changing L amounts to adding a constant to all variables $d(v)$, and this constant cancels in all operations of the algorithm, including the final test in line 32. Setting $L = 0$ corresponds to choosing negative lengths for the arcs sv.

LEMMA 4.5. *The algorithm Decision-Tree-Width performs $O(\log n)$ iterations of the while-loops in lines 12 and 24.*

Proof. The proof applies the ideas of Cole's speed-up technique [9]. We only analyze the while-loop in line 12; similar argument applies to the while-loop in line 24. Each vertex $(u, v, i) \in V(H)$ gets assigned a weight $W(u, v, i)$ that depends on the depth of node i in T and the bound k_0 for the treewidth of G. We use $W(A)$ for the sum of the weights over vertices $(u, v, i) \in A,\ A \subseteq V(H)$.

We first show that in each iteration, the weight $W(A)$ of active relaxations decreases at least by a factor $3/4$: the relaxations A' that are carried out remove one half of A's weight. Each relaxation $(u, v, i) \in A'$ that is carried out may make at most $|\Gamma^+(u, v, i)| \leq z$ relaxations active. However, by the choice of weights, the total weight of these successor relaxations is at most half the weight of (u, v, i). Thus, $W(A)$ is reduced to at most

$$\begin{aligned} W(A) - W(A') + \frac{W(A')}{2} &\leq W(A) - \frac{W(A')}{2} \\ &\leq \frac{3}{4} \cdot W(A). \end{aligned}$$

It follows that the number of iterations is bounded by $\log_{4/3} W_0/W_{\min}$, where W_0 is the initial weight $W(A)$

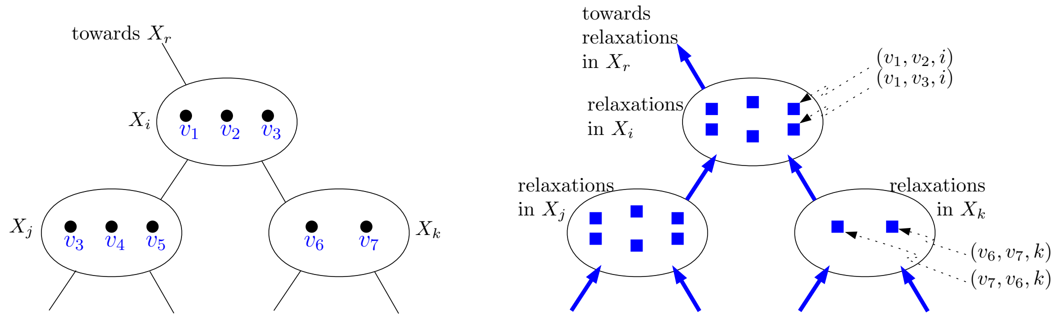

Figure 2: The directed graph H used in the algorithm. Left: portion of the tree decomposition of G rooted at r. Right: Portion of the graph H; the thick edges indicate that there is a directed edge between any vertex in one bag and any vertex in the other bag.

and $W_{\min}$ is the minimum weight of a non-empty set A. In our case, the weights are integers and $W_{\min} \geq 1$. The graph H has a total of $O(nz)$ nodes, each of weight at most $(2z)^h$, where $h = O(\log n)$ is the height of the tree. Thus, the number of iterations is bounded by

$$\begin{aligned} \log_{4/3} \frac{W_0}{W_{\min}} &\leq \log_{4/3}(O(nz)(2z)^h) \\ &= O(\log n + \log z + h \log 2z) \\ &= O(\log n). \end{aligned}$$

□

THEOREM 4.1. *Let k_0 be a fixed constant. For any graph with n vertices and treewidth at most k_0, we can find an obnoxious center in $O(n \log n)$ time.*

Proof. We apply parametric search to transform the decision Algorithm *Decision-Tree-Width* into an optimization algorithm. Consider running Algorithm *Decision-Tree-Width* for the (unknown) optimal value t^*. Starting with the interval $[t_0, t_1] = [-\infty, \infty]$, we maintain an interval $[t_0, t_1]$ such that $t^* \in [t_0, t_1]$ and all decisions that Algorithm *Decision-Tree-Width* has performed so far are identical for any $t \in [t_0, t_1]$. Instead of storing a single value $d(v)$ for each $v \in V(G)$, we keep a linear function in t, $d(v, t)$, which is initialized in line 4.

In lines 13 and 25, we have a set A of active relaxations (u, v, i) that we can perform. For each $(u, v, i) \in A$, let $t_{(u,v,i)}$ be the unique root of the linear equation $d(v, t) = d(u, t) + \delta_G(u, v)$. We then compute, in linear time, the weighted median $\hat{t}$ of these roots, with the weights $W(u, v, i)$ as given by the algorithm. We use the decision algorithm for this fixed value $\hat{t}$ to decide whether $t^* \leq \hat{t}$ or $t^* \geq \hat{t}$. This reduces the interval $[t_0, t_1]$ and decides a subset A' of the relaxations A that wait for a decision. The weight of A' is at least half the weight of A.

Thus, we can carry out one iteration of the loop 12–18 or the loop 24–30 for the the unknown value t^* in $O(n)$ time by a single-source shortest path computation; see Lemma 4.3. The additional overhead for computing the median and maintaining the sets A, B, D is linear. The number of iterations is $O(\log n)$ by Lemma 4.5. This leads to a total of $O(n \log n)$ time for the two while-loops.

The remaining operations can be carried out in $O(n)$ time: Since G has treewidth at most k_0, we spend $O(n)$ time in line 1 because of Lemma 4.1. The distances in line 2 can be computed in linear time by Lemma 4.2. These operations are independent of the value t and need to be carried out only once. □

5 Planar Graphs

First, we provide the background that will be used in our deterministic algorithm. Our algorithms are explained in Section 5.3. For the randomized algorithm, Sections 5.1 and 5.2 are not needed.

5.1 Distances We describe results concerning distances in planar graphs that will be used, starting with the following particular form of the results due to Klein.

THEOREM 5.1. (KLEIN [13]) *For a given embedded plane graph G with n vertices, and k vertex pairs $(u_1, v_1), \ldots, (u_k, v_k)$, with all vertices u_i on a common face, we can compute the distances $\delta_G(u_j, v_j)$, $1 \leq j \leq k$, in $O(n \log n + k \log n)$ time.*

The previous theorem, together with the techniques developed by Fakcharoenphol and Rao [10] imply the following result, which is of independent interest:

LEMMA 5.1. *Let G be an embedded planar graph with n vertices and let α be a Jordan curve passing through c vertices and crossing no edge of G. We can compute*

Algorithm *Decision-Tree-Width*

Input: A graph G with treewidth at most k_0 and a value t

Output: Decides if $\mathcal{U}(t)$ covers $A(G)$

1. Construct a binary, rooted tree decomposition $(\{X_i \mid i \in I\}, T)$ for G of width at most $3k_0 + 2$ and height $O(\log n)$;

2. for all bags X_i, compute and store $\delta_G(u,v)$ for all $u, v \in X_i$;

3. $L \leftarrow n \cdot \max_{e \in E(G)} \ell(e)$;

4. $d(v) \leftarrow 2L - t/w(v)$ for all $v \in V(G)$;

5. Construct the directed graph H;

6. $z \leftarrow 2(3k_0 + 3)(3k_0 + 2)$; (∗ upper bound on indegree and outdegree in H ∗)

7. (∗ Start bottom-up traversal of T ∗)

8. $W(u,v,i) \leftarrow (2z)^{\text{depth}(i)}$ for all $(u,v,i) \in V(H)$;

9. $A \leftarrow \{\, (u,v,i) \in V(H) \mid i \text{ a leaf in } T \,\}$; (∗ Relaxations that are active ∗)

10. $D \leftarrow \emptyset$; (∗ Relaxations that are done ∗)

11. $B \leftarrow V(H) \setminus A$; (∗ Relaxations that are waiting: not active, not done ∗)

12. **while** $A \neq \emptyset$

13. $A' \leftarrow$ some subset of A such that $W(A') \geq W(A)/2$;

14.   **for** $(u,v,i) \in A'$ (∗ in arbitrary order ∗)

15. $d(v) \leftarrow \min\{d(v), d(u) + \delta_G(u,v)\}$; (∗ Perform relaxations ∗)

16. $D \leftarrow D \cup A'$;

17. $A_{\text{new}} \leftarrow \{\, (u,v,i) \in B \mid \Gamma^-(u,v,i) \subset D \,\}$;

18. $A \leftarrow (A \setminus A') \cup A_{\text{new}}$; $B \leftarrow B \setminus A_{\text{new}}$;

19. (∗ End bottom-up traversal of T, start top-down traversal ∗)

20. $W(u,v,i) \leftarrow (1/2z)^{\text{depth}(i)}$ for all $(u,v,i) \in V(H)$;

21. $A \leftarrow \{\, (u,v,r) \in V(H) \mid r \text{ the root of } T \,\}$; (∗ Relaxations that are active ∗)

22. $D \leftarrow \emptyset$; (∗ Relaxations that are done ∗)

23. $B \leftarrow V(H) \setminus A$; (∗ Relaxations that are waiting: not active, not done ∗)

24. **while** $A \neq \emptyset$

25. $A' \leftarrow$ some subset of A such that $W(A') \geq W(A)/2$;

26.   **for** $(u,v,i) \in A'$ (∗ in arbitrary order ∗)

27. $d(v) \leftarrow \min\{d(v), d(u) + \delta_G(u,v)\}$; (∗ Perform relaxations ∗)

28. $D \leftarrow D \cup A'$;

29. $A_{\text{new}} \leftarrow \{\, (u,v,i) \in B \mid \Gamma^+(u,v,i) \subset D \,\}$;

30. $A \leftarrow (A \setminus A') \cup A_{\text{new}}$; $B \leftarrow B \setminus A_{\text{new}}$;

31. (∗ End top-down traversal of T. Now, $d(v) = \delta_{G_+}(s,v)$ for all $v \in V(G)$ ∗)

32. **return** $\bigwedge_{uv \in E(G)} (d(u) + d(v) \leq 4L + \ell(uv))$

Figure 3: depth(i) refers to the depth of node i in T. For any set $A \subset V(H)$, its weight $W(A)$ is defined as the sum of $W(u,v,i)$ over all $(u,v,i) \in A$.

in $O(n \log n + c^2 \log^2 c)$ time the distances $\delta_G(u,v)$ for all $u, v \in V(C)$.

Proof. Let V_α be the vertices that lie on α. The curve α splits the graph into an interior part I and an exterior part E; the vertices V_α belong to both parts, and they lie on a common face in each part. By Theorem 5.1, we compute in $O(n \log n + c^2 \log n)$ time the distances $\delta_I(u,v), \delta_E(u,v)$ for all $u, v \in V_\alpha$. Henceforth, only the vertices V_α and the distances δ_E, δ_I between them are used.

We now describe a data structure that is used to maintain the distances δ_E while running Dijkstra's algorithm. Assume that each vertex $v \in V_\alpha$ has a label $d_E(v) \geq 0$, and we have a set of inactive vertices $S_E \subset V_\alpha$. Fakcharoenphol and Rao [10, Section 4] give a technique to construct a data structure $DS(E)$ that *implicitly* maintains labels $d_E(v),\ v \in V_\alpha$, and supports the following operations in $O(\log^2 c)$ amortized time:

- `Relax`(v, d_v): we set $d_E(v) = d_v$ and (implicitily) update the labels of all other vertices using $d_E(u) = \min\{d_E(u), d_v + \delta_E(v,u)\}$ for all $u \in V_\alpha$. This operation requires that the values $\delta_E(v,u),\ u \in V_\alpha$, are available.
- `FindMin`(): returns the value $d_E^0 = \min_{v \in V_\alpha \setminus S_E} d_E(v)$.
- `ExtractMin`() returns $v_E = \arg\min_{v \in V_\alpha \setminus S_E} d_E(v)$ and makes it inactive: $S_E = S_E \cup \{v_E\}$.

A similar data structure $DS(I)$ can be built for the distances δ_I, instead of δ_E.

For a fixed start vertex $u \in V_\alpha$, we compute $\delta_G(u,v)$ for all $v \in V_\alpha$ by Dijkstra's algorithm, using the data structures $DS(I)$ and $DS(E)$ in parallel. For example, when selecting the next vertex to be processed, we call `FindMin`() in both data structures $DS(E)$ and $DS(I)$, and then apply `ExtractMin`() to the data structure that has produced the smaller value. There are $|V_\alpha| = c$ iterations in Dijkstra's algorithm, and we spend $O(\log^2 c)$ amortized time per iterations. Therefore, for a fixed $u \in V_\alpha$, we can compute in $O(c \log^2 c)$ time the values $\delta_G(u,v)$ for all $v \in V_\alpha$. Applying this procedure for each $u \in V_\alpha$, the result follows. □

5.2 Decompositions We use the (hierarchical) decomposition of planar graphs as given by Fakcharoenphol and Rao [10]. Let G be an embedded plane graph. We assume that G is a triangulation, since we can add edges of infinite length without harm. A *piece* P is a connected subgraph of G; we assume in P the embedding inherited from G. A vertex v in P is a *boundary vertex* if there is some edge uv in G with u not in P. The *boundary* ∂P of P is the set of its boundary vertices. A *hole* in a piece P is a facial walk of P such that, in G, the removal of its vertices disconnects G. Note that the boundary of a piece P is contained in its holes.

The decomposition starts with G as a single "piece" and recursively partitions each piece P into two parts P_l and P_r, using a Jordan curve α_P that passes through vertices but does not cross any edge of P, until pieces consisting of a single edge are obtained. The vertices V_{α_P} crossed by α_P go to both parts P_l and P_r. If any part has several connected components, we treat each separately; for simplicity we assume that both P_l, P_r are connected. The vertices V_{α_P} form part of a hole in P_l, P_r. We denote the recursive decomposition by (Π, T_Π), where Π is a collection of pieces and T_Π is a rooted binary tree.

For a piece $P_i \in \Pi$, let m_i be its number of vertices and let b_i be the number of its boundary vertices. The curve α that is used to partition P_i comes from Miller's results [18]: given a planar graph with weights in the vertices, it finds in $O(m_i)$ time a curve α passing through $O(\sqrt{m_i})$ vertices and crossing no edge such that each side of α has at most $2/3$ of the total weight. In the hierarchical decomposition, we make rounds, where each *round* consists successively of a balanced separation of the vertices, a balanced separation of the boundary vertices, and a balanced separation of the holes. Therefore, in each round we decompose a piece into 8 subpieces. This hierarchical decomposition has the following properties.

LEMMA 5.2. *The hierarchical decomposition (T_Π, Π) that we have described can be constructed in $O(n \log n)$ time, has $O(\log n)$ levels, each piece has $O(1)$ holes, and*

$$\sum_{P_i \in \Pi_d} m_i = O(n), \qquad \sum_{P_i \in \Pi_d} b_i^2 = O(n),$$

where Π_d is the set of pieces at depth d in T_Π. □

We apply for each piece $P \in \Pi$ Lemma 5.1 to α_P and Theorem 5.1, once per hole, and obtain:

LEMMA 5.3. *Let (Π, T_Π) be a hierarchical decomposition. We can compute in $O(n \log^3 n)$ time the distances $\delta_P(u,v)$ between every pair of vertices $u, v \in \partial P \cup V_{\alpha_P}$, for all pieces $P \in \Pi$.* □

5.3 Algorithms If G is a planar graph, then the graph G_+ defined in Section 3 is a so-called apex graph, and it has separators of size $O(\sqrt{n})$: a planar separator [14] of G plus the apex s is a separator in G_+. Moreover, since we know the apex of G_+ beforehand, a separator in G_+ can be computed in linear time, and the results by Henzinger et al. [11] imply that

$T_{\text{sssp}}(G_+) = O(n)$. From Lemma 3.1 and Theorem 3.1, we conclude the following.

THEOREM 5.2. *For planar graphs with n vertices, we can decide in $O(n)$ worst-case time any covering instance. Moreover, we can find an obnoxious center in $O(n \log n)$ expected time.* □

We next move on to our deterministic algorithm. For this, we design another algorithm for the decision problem that is suitable for parametric search.

THEOREM 5.3. *In a planar graph G with n vertices, we can find an obnoxious center in $O(n \log^3 n)$ time.*

Proof. We construct a hierarchical decomposition (Π, T_Π) of G as discussed in the previous section. For each piece $P \in \Pi$, we compute and store the distances described in Lemma 5.3.

We now design an algorithm to solve the decision problem, that is, given a value t, we want to decide if $\mathcal{U}(t)$ covers $A(G)$ or not. Like in Section 3, we consider the graph G_+, where each edge sv has length $2L - t/w(v)$, and we are interested on computing the distances $\delta_{G_+}(s, v)$ for all $v \in V(G)$. For each piece $P \in P$, let P_+ be the graph obtained by adding the edges sv, $v \in V(P)$.

First, we make a bottom-up traversal of T_Π. The objective is, for each piece P, to find the values $\delta_{P_+}(s, v)$ for $v \in \partial P \cup V_{\alpha_P}$. At the bottommost level, each piece P has constant size, and we can compute the values $\delta_{P_+}(s, v), v \in V(P)$ in $O(1)$ time. For any piece P with two subpieces Q, R we use that, for any $v \in \partial P \cup V_{\alpha_P}$, the value $\delta_{P_+}(s, v)$ is given by

$$(2) \quad \min \left\{ \begin{array}{l} \min\{\delta_{Q_+}(s, u) + \delta_P(u, v) \mid u \in \partial Q)\} \\ \min\{\delta_{R_+}(s, u) + \delta_P(u, v) \mid u \in \partial R)\} \end{array} \right\}.$$

At the end of the traversal, we obtain the values $\delta_{G_+}(s, v)$ for all $v \in V_{\alpha_G}$.

Then, we make a top-down traversal of T_Π. The objective is, for each piece P, to find the values $\delta_{G_+}(s, v)$ for $v \in V_{\alpha_P}$. At the root, we obtained this data from the bottom-top traversal. For every piece P which is a child of another piece Q and for any $v \in \partial P \cup V_{\alpha_P}$, the value $\delta_{G_+}(s, v)$ is given by

$$(3) \quad \min \left\{ \begin{array}{l} \delta_{P_+}(s, v) \\ \min\{\delta_{G_+}(s, u) + \delta_P(u, v) \mid u \in \partial P)\} \end{array} \right\}.$$

The values $\delta_{G_+}(s, u)$ for $u \in \partial P$ are available because $\partial P \subseteq \partial Q \cup V_{\alpha_Q}$. At the end of the traversal, we have the values $\delta_{G_+}(s, v)$ for all $v \in V(G)$ because every vertex is boundary of some piece.

This finishes the description of the decision algorithm. We analyze its running time in view of applying parametric search [16]. The hierarchical decomposition and the use of Lemma 5.3 is done once at the beginning and takes $O(n \log^3 n)$ time. In a piece P, using equation (2) or (3) for each of its $O(b + \sqrt{m})$ vertices in $\partial P \cup V_{\alpha_P}$ takes $O((b + \sqrt{m})^2) = O(b^2 + m)$ time. Therefore, for all pieces $\Pi_d \subset \Pi$ at depth d of T_Π we spend $\sum_{P_i \in \Pi_d} O(b_i^2 + m_i) = O(n)$ time during the algorithm. Moreover, note that in the bottom-up (or the top-down) traversal, it does not matter in what order the $O(n)$ operations concerning the pieces Π_d are made.

We have seen that after $O(n \log^3 n)$ time, the decision problem can be solved with $O(\log n)$ rounds, each round involving $O(n)$ operations that can be made in arbitrary order. Standard parametric search [16] leads to an optimization algorithm making $O(\log n)$ rounds, where each round uses $O(n)$ time plus the time used to solve $O(\log n)$ decision problems. The decision problem is a covering problem, and Theorem 5.2 leads to $O(n \log n)$ time per level, for a total of $O(n \log^2 n)$ time over all levels. Note that the dominating term in the running time comes from Lemma 5.3. □

References

[1] B. Ben-Moshe, B. K. Bhattacharya, and Q. Shi. Efficient algorithms for the weighted 2-center problem in a cactus graph. In *Algorithms and Computation: ISAAC 2005*, volume 3827 of *Lecture Notes in Computer Science*, pages 693–703. Springer-Verlag, 2005.

[2] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[3] H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings MFCS'97*, volume 1295 of *Lecture Notes in Computer Science*, pages 19–36. Springer-Verlag, 1997.

[4] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comput. Sci.*, 209:1–45, 1998.

[5] H. L. Bodlaender and T. Hagerup. Parallel algorithms with optimal speedup for bounded treewidth. *SIAM J. Comput.*, 27(6):1725–1746, 1998.

[6] R. E. Burkard, H. Dollani, Y. Lin, and G. Rote. The obnoxious center problem on a tree. *SIAM Journal on Discrete Mathematics*, 14:498–509, 2001.

[7] S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part II: Optimal parallel algorithms. *Theoretical Computer Science*, 203:205–223, 1998.

[8] S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27:212–226, 2000.

[9] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[10] J. Fakcharoenphol and S. Rao. Planar graphs, negative weight edges, shortest paths, and near linear time. *J. Comput. Syst. Sci.*, 72:868–889, 2006.

[11] M.R. Henzinger, P.N. Klein, S. Rao, and S. Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997.

[12] M. J. Katz, K. Kedem, and M. Segal. Improved algorithms for placing undesirable facilities. *Computers and Operations Research*, 29:1859–1872, 2002.

[13] P. N. Klein. Multiple-source shortest paths in planar graphs. In *Proc. 16th ACM-SIAM Sympos. Disc. Alg. (SODA)*, pages 146–155, 2005.

[14] R. J. Lipton and R. E. Tarjan. A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–189, 1979.

[15] N. Megiddo. Combinatorial optimization with rational objective functions. *Math. Oper. Res*, 4(4):414–424, 1979.

[16] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[17] N. Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM J. Comput.*, 12:759–776, 1983.

[18] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *J. Comput. Syst. Sci.*, 32:265–279, 1986.

[19] B.A. Reed. Finding approximate separators and computing tree width quickly. In *STOC '92: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 221–228, 1992.

[20] A. Tamir. Improved complexity bounds for center location problems on networks by using dynamic data structures. *SIAM J. Discrete Math.*, 1:377–396, 1988.

[21] A. Tamir. Obnoxious facility location on graphs. *SIAM J. Discrete Math.*, 4:550–567, 1991.

[22] A. Tamir. Locating two obnoxious facilities using the weighted maximin criterion. *Operations Research Letters*, 34:97–105, 2006.

[23] B. Zmazek and J. Žerovnik. The obnoxious center problem on weighted cactus graphs. *Discrete Appl. Math.*, 136:377–386, 2004.