OBNOXIOUS CENTERS IN GRAPHS*

SERGIO CABELLO† AND 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.

Key words. graph algorithms, facility location, planar graphs, parametric search, bounded treewidth

AMS subject classifications. 05C85, 68W05, 90B85

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, 2, 7, 11, 15, 23, 25, 26, 27] 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 [25] gave an algorithm with $O(n \log^2 n)$ worst-case time for trees. Faster algorithms are known for some special classes of trees [7, 25]. For cactus graphs, Zmazek and Žerovnik [27] 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 [24] 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 $O(\sqrt{n})$ [17] to solve the problem in roughly $O(n^{3/2})$ time.

Our results. In general, we follow an approach similar to Tamir [25], 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:

*To appear in SIAM J. Discrete Math. (2011). A preliminary version appeared in SODA 2007 [8].

†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 and program P1-0297.

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

- A covering problem in G can be solved constructing a shortest path tree in an augmented graph obtained by adding an apex to G.
- For arbitrary graphs, 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 $O(nm + n^2 \log n)$ worst-case time [24].
- For graphs with bounded treewidth, we give an algorithm to find an obnoxious center in $O(n \log n)$ worst-case time. Previously, algorithms using near-linear time were known only for trees (graphs with treewidth one), and they used $O(n \log^2 n)$ time [25].
- 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.

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 like Megiddo's [20] 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 of bounded treewidth is based on parametric search [18, 19]. However, an interesting point is our use of Cole's [12] 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. [14]. Our deterministic algorithm for planar graphs is based on the results and techniques developed by Fakcharoenphol and Rao [13] and Klein [16] for computing several shortest paths in planar graphs.

Organization of the paper. The rest of the paper is organized as follows. In the next section we give a formal definition of obnoxious centers, covering problem, and their relation, as well as a review of parametric search. In Section 3 we show how to reduce the associated decision problem to a single source shortest path problem, and also discuss how this easily leads to randomized algorithms. In Section 4 we study the case of graphs with bounded treewidth, and in Section 5 we deal with 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 vertex from a. In this setting, the larger the weight, the less relevant is the site. 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. Following Tamir [25], we study the decision version of the obnoxious center problem, which can be formulated as a covering problem.

Let $D(v, r) = \{a \in A(G) \mid \delta(a, v) \le r\}$ denote the closed 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 $A(G) = \bigcup_{v \in V(G)} D(v, r_v)$?

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 each disk as growing around its center 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 1. *Let a^* be an obnoxious center of G. The optimum value of the objective function is given by*

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

2.3. Parametric search. When solving optimization problems, it is common to look first at algorithms that solve the corresponding decision problem: is the optimal value smaller, larger, or equal, than a given value? Parametric search is a generic technique to transform a certain type of algorithms that solve the decision problem into an algorithm that solves the optimization problem. We explain the technique stepwise, starting from the basic idea contained in Megiddo [18].

Consider an optimization problem whose optimal value is t^*. Assume first that we have an algorithm A_d solving in time T_d the decision problem: given a parameter t, it decides whether t is smaller, larger, or equal to t^*. We now show how we can apply the algorithm A_d to the (unknown) value t^*. The difficulty is to decide which

[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.

branch the algorithm takes when it comes to a branching point. This amounts to checking the sign of a polynomial $p(t)$ evaluated at $t = t^*$. We can work around our ignorance of t^* by computing the roots $t_1, t_2, \dots$ of the polynomial $p(t)$ and using the decision algorithm to locate t^* between two consecutive roots of the polynomial $p(t)$. In this way, we know which branch the algorithm would follow if applied to t^*, and the algorithm can proceed. When we know which branch the algorithm follows for the optimal value t^*, then typically the problem is reduced to an interval where t^* must lie, and we can solve the problem restricted to values t within that interval. For this approach to work, two assumptions on the decision algorithm A_d are needed: the polynomials evaluated at branching points have degree bounded by a constant, in the input parameter t, and the solution can be computed if we know in which branch the decision algorithm finishes. The running time of this algorithm is $O(T_d^2)$: for each of the $O(T_d)$ branching points we have to call the decision algorithm $O(1)$ times.

As noted by Megiddo [19], a significant speedup can be obtained using parallel algorithms. Assume that we have a parallel algorithm A_p for the decision problem that makes T_p rounds (or time steps) of T_r parallel operations each. When simulating a step of the parallel algorithm, we can compute the roots of all T_r polynomials that are involved in one round, and make a binary search among them to find between which two consecutive roots t^* lies. For the binary search, we use a linear-time median-finding algorithm to locate the median of the roots, and then call the sequential algorithm A_d to decide in which side to recurse. Thus, in each round we can find the correct branch of the algorithm A_p for the value t^* in time $O(T_r + T_d \log T_r)$. Looking over all rounds, we conclude that in time $O(T_p(T_r + T_d \log T_r))$ we can find the branch followed by A_p if applied to the optimal value t^*.

Finally, a further improvement for a special type of parallel algorithms was realized by Cole [12]. This improvement can be applied when each computation of the parallel algorithm depends on the outcome of at most a constant number of previous computations. That is, we consider a digraph that encodes which operations have to be performed before other operations. If this digraph has bounded in-degree (or bounded out-degree) and the length of the longest path (which is T_p) is small, we can use a weighting technique so that in the binary search we can consider simultaneously roots of the polynomials from different rounds of the parallel algorithm. Thus, in some sense, this approach interleaves the steps of collecting roots and making the binary search across different rounds. We use and explain this technique in detail in Section 4.

3. General Graphs.

Our running times will be expressed as a function of $T_{\text{sssp}}(G)$, 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 $T_{\text{sssp}}(G) = O(n \log n + m)$ time. Better results are known for some special classes of graphs.

Consider the problem of covering the graph G by balls $D(v, r_v)$, $v \in V(G)$. A useful concept for our subsequent discussion is the *coverage* $C(v)$ of a vertex $v \in V(G)$, defined as

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

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

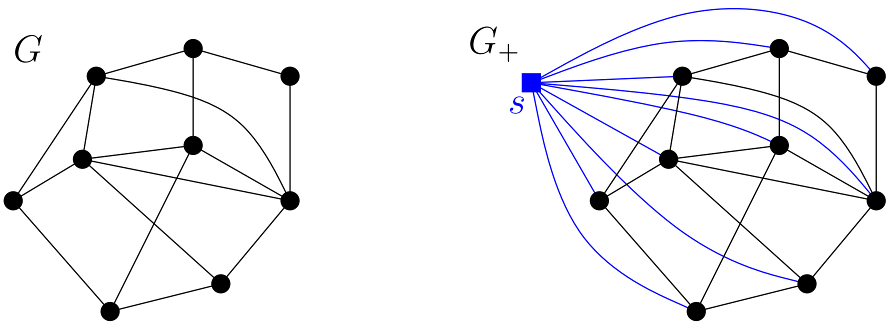

FIGURE 1. *Example showing how to obtain the graph G_+ from G.*

reflected in the following observation.

LEMMA 2. *For an $e = xy \in E(G)$, the edge $A(e)$ is covered by $\bigcup_{v \in V(G)} D(v, r_v)$ if and only if $\ell(e) \leq C(x) + C(y)$.*

Proof. We parameterize the edge $A(e)$ by the distance λ from x $(0 \leq \lambda \leq \ell(e))$. The point $a \in A(e)$ with parameter λ is covered if and only if

$$\min\{\lambda + \delta(x, v),\, \ell(e) - \lambda + \delta(y, v)\} \leq r_v \qquad \text{for some } v \in V(G),$$

which is equivalent to

$$\begin{aligned} 0 \geq& \min_{v \in V(G)} \min\{\lambda + \delta(x, v) - r_v,\, \ell(e) - \lambda + \delta(y, v) - r_v\} \\ =& \min\{\lambda + \min_{v \in V(G)} \{\delta(x, v) - r_v\},\, \ell(e) - \lambda + \min_{v \in V(G)} \{\delta(y, v) - r_v\} \\ =& \min\{\lambda - C(x),\, \ell(e) - \lambda - C(y)\}. \end{aligned}$$

Therefore, the edge $A(e)$ is covered if and only if

$$\min\{\lambda - C(x),\, \ell(e) - \lambda - C(y)\} \leq 0 \qquad \text{for all } 0 \leq \lambda \leq \ell(e),$$

which is equivalent to the condition $\ell(e) \leq C(x) + C(y)$. ◻

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 $V(G)$. See Figure 1.

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 $L = n \cdot \ell_{\max}$ on the length of any shortest path in G, where $\ell_{\max}$ is the length of the longest edge in G. Henceforth, we assume that $r_v \leq L$, $v \in V(G)$, 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. *The coverages $C(v)$ in G are related to the distances from s in G_+ as follows*:

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

Proof.

$$\begin{aligned}\delta_{G_+}(s,v) &= \min\{\, \ell(su) + \delta_{G_+}(u,v) \mid u \in V(G)\,\} \\ &= \min\{\, 2L - r_u + \delta_G(u,v) \mid u \in V(G)\,\} \\ &= 2L - \max\{\, r_u - \delta_G(u,v) \mid u \in V(G)\,\} = 2L - C(v). \qquad \square\end{aligned}$$

Combining Lemmas 2 and 3, we achieve the following:

PROPOSITION 4. *We can solve an instance of the covering problem in a graph G in $O(T_{\text{sssp}}(G_+))$ time.*

Proof. By the previous lemma, we can compute the coverages $C(v)$ for all $v \in V(G)$. Then, we use Lemma 2 for each edge $e \in E(G)$ to decide if $A(G)$ is covered or not. The first step requires $T_{\text{sssp}}(G_+)$ time, and the second step takes $O(|E(G)|)$ time. $\square$

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

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

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, $t_e = \min\{\, t \in \mathbb{R}_+ \mid A(e) \subset \mathcal{U}(t)\,\}$. The following result is straightforward.

LEMMA 5. *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 $t_e \leq t_{e'}$ if and only if $A(e) \subset \mathcal{U}(t_{e'})$. $\square$

LEMMA 6. *For any edge $e = xy \in E(G)$, we can compute t_e in $O(T_{\text{sssp}}(G))$ time.*

Proof. We parameterize the edge $A(e)$ by the distance λ from x ($0 \leq \lambda \leq \ell(e)$). Then the time when the point a with parameter λ is covered is given by 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 $O(T_{\text{sssp}}(G))$ 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 [20]. $\square$

THEOREM 7. *For a graph G with n vertices, the algorithm Obnoxious-Center-Randomized in Figure 2 finds an obnoxious center in $O(T_{\text{sssp}}(G_+)\log n)$ expected time.*

Proof. Correctness is clear from Lemma 5: in steps 7–8, we exclude the edges e' with $t_{e'} \leq t_{e_i}$. 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. Actually, once we know t^* we can also compute in $O(T_{\text{sssp}}(G_+))$ time all obnoxious centers: we compute the coverages $C(v,t^*)$ for all vertices $v \in V(G)$ and observe that every edge uv with $\ell(uv) = C(u,t^*) + C(v,t^*)$ contains an obnoxious center at distance $C(u,t^*)$ from u.

To bound the running time, we first show that the while-loop in lines 3–8 is iterated an expected number of $O(\log |E(G)|) = O(\log n)$ times. Indeed, if I_n denotes the

Algorithm *Obnoxious-Center-Randomized*
Input: A graph G
Output: Computes t^* and finds an obnoxious center

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

FIGURE 2. *Algorithm* Obnoxious-Center-Randomized

expected number of remaining iterations when $|E_i| = n$, then we have the recurrence

$$I_n = 1 + \frac{1}{n}\sum_{i=1}^{n-1} I_i, \qquad I_1 = 1,$$

which solves to $I_n \leq (1 + \ln n)$ by induction:

$$\begin{aligned} I_n &= 1 + \frac{1}{n}\sum_{i=1}^{n-1} I_i \leq 1 + \frac{1}{n}\sum_{i=1}^{n-1}(1 + \ln i) \\ &\leq 1 + \frac{1}{n}\int_1^n (1 + \ln x)\,dx = 1 + \frac{1}{n}(n \ln n - 0) = 1 + \ln n. \end{aligned}$$

Finally, note that each iteration of the loop in lines 3–8 takes $O(T_{\text{sssp}}(G_+))$ time: lines 4, 5 and 8 take $O(m)$ time, line 6 takes $O(T_{\text{sssp}}(G))$ time because of Lemma 6, and line 7 takes $O(T_{\text{sssp}}(G_+) + m) = O(T_{\text{sssp}}(G_+))$ because we can compute the coverages of each vertex using Lemma 3, and then apply Lemma 2 for each edge. ☐

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

COROLLARY 8. *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 search [19], 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 [24].

4. Graphs with Bounded Treewidth.

4.1. Tree Decompositions and Treewidth. We review some basic properties of tree decompositions and treewidth. See [4, 5] for a more comprehensive treatment.

DEFINITION 9. *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 [3] gives an algorithm to construct in linear time a tree decomposition of minimum width. Furthermore, Bodlaender and Hagerup [6] 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 10. *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 stated in Lemma 10. Therefore, we could replace Bodlaender's algorithm [3] by Reed's algorithm [22], which takes $O(n \log n)$ time but is simpler.

Chaudhuri and Zaroliagis [10, 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 11. *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, $i \in I$, can be calculated in $O(k^3 n)$ time.*

4.2. A Decision Algorithm and a Parametric Search Algorithm. Tamir showed in [25] that the covering problem is solvable in $O(n)$ time 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 [10] showed that a shortest path tree in a graph with bounded treewidth can be constructed in linear time, Proposition 4 leads to the following result.

LEMMA 12. *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.* □

Theorem 7 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 [9] for computing a shortest path tree in parallel. Moreover, our modification also applies the technique of Cole [12], to obtain a speed-up when later applying parametric search [19]. This leads to an algorithm using $O(n \log n)$ time in the worst case. We now 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

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 $A(G)$

Construct a binary, rooted tree decomposition $(\{X_i \mid i \in I\}, T)$ for G of width at most $3k_0 + 2$ and height $O(\log n)$;
for all bags X_i, compute and store $\delta_G(u, v)$ for all $u, v \in X_i$;
$L \leftarrow n \cdot \max_{e \in E(G)} \ell(e)$;
$d(v) \leftarrow 2L - t/w(v)$ for all $v \in V(G)$;
Construct the directed graph H;
$z \leftarrow 2(3k_0 + 3)(3k_0 + 2)$; $(*$ upper bound on indegree/outdegree in H $*)$
$(*$ Start bottom-up traversal of T $*)$
$W(u, v, i) \leftarrow (2z)^{\mathrm{depth}(i)}$ for all $(u, v, i) \in V(H)$;
$A \leftarrow \{\, (u, v, i) \in V(H) \mid i$ a leaf in $T\,\}$; $(*$ Relaxations that are active $*)$
$D \leftarrow \emptyset$; $(*$ Relaxations that are done $*)$
$B \leftarrow V(H) \setminus A$; $(*$ Relaxations that are waiting: not active, not done $*)$
while $A \neq \emptyset$
  $A' \leftarrow$ some subset of A such that $W(A') \geq W(A)/2$;
  for $(u, v, i) \in A'$ $(*$ in arbitrary order $*)$
    $d(v) \leftarrow \min\{d(v), d(u) + \delta_G(u, v)\}$; $(*$ Perform relaxations $*)$
  $D \leftarrow D \cup A'$;
  $A_{\mathrm{new}} \leftarrow \{\, (u, v, i) \in B \mid \Gamma^-(u, v, i) \subset D\,\}$;
  $A \leftarrow (A \setminus A') \cup A_{\mathrm{new}}$; $B \leftarrow B \setminus A_{\mathrm{new}}$;
$(*$ End bottom-up traversal of T, start top-down traversal $*)$
$W(u, v, i) \leftarrow (1/2z)^{\mathrm{depth}(i)}$ for all $(u, v, i) \in V(H)$;
$A \leftarrow \{\, (u, v, r) \in V(H) \mid r$ the root of $T\,\}$; $(*$ Relaxations that are active $*)$
$D \leftarrow \emptyset$; $(*$ Relaxations that are done $*)$
$B \leftarrow V(H) \setminus A$; $(*$ Relaxations that are waiting: not active, not done $*)$
while $A \neq \emptyset$
  $A' \leftarrow$ some subset of A such that $W(A') \geq W(A)/2$;
  for $(u, v, i) \in A'$ $(*$ in arbitrary order $*)$
    $d(v) \leftarrow \min\{d(v), d(u) + \delta_G(u, v)\}$; $(*$ Perform relaxations $*)$
  $D \leftarrow D \cup A'$;
  $A_{\mathrm{new}} \leftarrow \{\, (u, v, i) \in B \mid \Gamma^+(u, v, i) \subset D\,\}$;
  $A \leftarrow (A \setminus A') \cup A_{\mathrm{new}}$; $B \leftarrow B \setminus A_{\mathrm{new}}$;
$(*$ End top-down traversal of T. Now, $d(v) = \delta_{G_+}(s, v)$ for all $v \in V(G)$. $*)$
return $\bigwedge_{uv \in E(G)} (\ell(uv) \leq 4L - d(u) - d(v))$

FIGURE 3. *Decision algorithm for the covering problem in graphs of treewidth at most k_0. $\mathrm{depth}(i)$ 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 $(u, v, i) \in A$.*

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

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

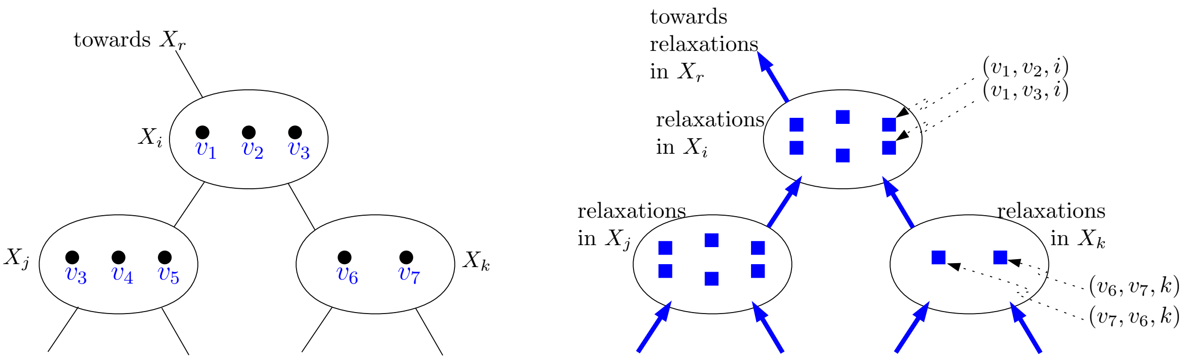

FIGURE 4. *The directed graph H used in the algorithm. Left: portion of the tree decomposition of G rooted at r. Right: Portion of the graph H; the thick edges indicate that there is a directed edge between any vertex in one bag and any vertex in the other bag.*

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:

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

Consider the algorithm *Decision-Tree-Width* in Figure 3. Although it is more complicated and inefficient than our previous approach, it is more suitable for using it in the parametric search framework, as described below. It uses a directed graph H defined by

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

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 4. Moreover, each vertex $(u, v, i) \in V(H)$ has a weight $W(u, v, i)$ associated to it, whose value is different for the top-down and the bottom-up parts (lines 8 and 20). The weights are irrelevant for the correctness of the algorithm but they affect its efficiency. We ignore the weights for the time being.

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. 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. To show the correctness of the algorithm we will use the following basic observation about tree-decompositions; see [10, Lemma 3.1] for a similar statement:

LEMMA 13. *For every path from u to v in G there is a subsequence of its vertices*

$u = u_0, u_1, \ldots, u_r = v$ and a sequence of distinct nodes $X_1, X_2, \ldots, X_r$ that lie on a path in T such that $u_{i-1}, u_i \in X_i$.

Proof. Let $u_0 = u$. Look at the subtree T_{u_0} of nodes containing u_0 and the subtree T_v containing v. If they overlap we are done. Otherwise let X_1 be the node of T_{u_0} closest to T_v, and $X_1' \notin T_{u_0}$ be the adjacent node on the path to T_v. The edge (X_1, X_1') splits the tree into two components S and S'. We select some vertex u_1 on the path that belongs both to X_1 and to X_1'. (Such vertex must exist because $X_1 \cup X_1'$ is a cutset in G.) Then we have $u_0, u_1 \in X_1$, satisfying the statement of the lemma. We also have $u_1 \in X_1'$, and we can proceed by induction from u_1. ∎

LEMMA 14. *The algorithm Decision-Tree-Width correctly decides if $\mathcal{U}(t)$ covers $A(G)$.*

Proof. As in Proposition 4, we only need to show that in line 31 the algorithm computes shortest distances $d(v)$ from s to all vertices v in the graph G_+. Line 4 initializes $d(v)$ to the length of the edges sv. Because of Lemma 13, a shortest path from s to v has a subsequence of vertices $u = u_0, u_1, \ldots, u_r = v$ and a sequence of distinct nodes $X_1, X_2, \ldots, X_r$ that lie on a path in T such that $u_{i-1}, u_i \in X_i$. The path in T containing $X_1, X_2, \ldots, X_r$ consists of a bottom-up and a top-down part, and therefore the algorithm performs the relaxations $d(u_i) = \min\{d(u_i), d(u_{i-1}) + \delta_G(u_i, u_{i-1})\}$ in the order $i = 1 \ldots r$. Therefore, at the end $d(v) = d(u_r) = \delta_{G+}(s, u_r)$. ∎

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

We now turn our attention to the efficiency of the algorithm and the role of weights.

LEMMA 15. *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 [12]. We only analyze the while-loop in line 12; a similar argument applies to the while-loop in line 24. We use $W(A)$ for the sum of the weights over vertices $(u, v, i) \in A,\ A \subseteq V(H)$.

The value $z = 2(3k_0 + 3)(3k_0 + 2)$ (line 6) is an upper bound on the maximum outdegree of H because each bag X_i has at most two descendants in T. The weight $W(u, v, i)$ of node (u, v, i) is set to $(2z)^{\mathrm{depth}(i)}$ (line 8), where depth(i) is the depth of node i in T. These values are chosen such that the following property holds: the weight of the successors $\Gamma^+(u, v, i)$ of vertex (u, v, i) is at most half the weight of (u, v, i), that is, $W(\Gamma^+(u, v, i)) \le W(u, v, i)/2$.

We can now 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 makes a subset of $\Gamma^+(u, v, i)$ active relaxations. However, the total weight of these successor relaxations is at most $W(u, v, i)/2$. Thus, $W(A)$ is reduced to at most

$$W(A) - W(A') + W(A')/2 \le W(A) - W(A')/2 \le W(A) \cdot \tfrac{3}{4}.$$

It follows that the number of iterations is bounded by $\log_{4/3} W_0/W_{\min}$, where W_0 is the initial weight $W(A)$ and $W_{\min}$ is the minimum weight of a non-empty set A. In our case, the weights are integers and $W_{\min} \ge 1$. 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

$$\log_{4/3} \frac{W_0}{W_{\min}} \le \log_{4/3}(O(nz)(2z)^h) = O(\log n + \log z + h \log 2z) = O(\log n). \qquad \square$$

THEOREM 16. *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 the coefficients of 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 carry out. Such a relaxation involves the comparison between $d(v, t)$ and $d(u, t) + \delta_G(u, v)$, whose outcome depends on t. For each $(u, v, i) \in A$, define the *critical value* $t_{u,v,i}$ as 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 $t^* \le \hat{t}$ or $t^* \ge \hat{t}$. This settles the comparisons for those relaxations $A' \subseteq A$ whose critical value is $\ge \hat{t}$ or those whose critical value is $\le \hat{t}$, respectively, and we can perform these relaxations. The weight of A' is at least half the weight of A. The interval $[t_0, t_1]$ is reduced to $[t_0, \min\{t_1, \hat{t}\,\}]$ or to $[\max\{t_0, \hat{t}\,\}, t_1]$.

Thus, we can carry out one iteration of the loop 12–18 or the loop 24–30 for the unknown value t^* in $O(n)$ time by solving the decision problem for $\hat{t}$; see Lemma 12. 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 15. This leads to a total of $O(n \log n)$ time for the two while-loops.

Line 32 involves a final comparison. Recall that we are looking for the smallest value t which makes the algorithm return *true*. Thus, we simply have to compute the smallest value t such that all conditions $\ell(uv) \le 4L - d(u, t) - d(v, t)$ are fulfilled, i. e., we compute the optimum value t^* as the maximum of the roots of all equations $\ell(uv) = 4L - d(u, t) - d(v, t)$, $uv \in E(G)$. (This value is automatically guaranteed to lie in the current interval $[t_0, t_1]$, by the correctness of the decision algorithm on which this interval is based.) The edge uv where the maximum is achieved is the edge on which the obnoxious center a^* is placed. Its location can be computed from $d(u)$ and $d(v)$ as the last point on the edge that remains uncovered as t approaches t^* from below.

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 10. The distances in line 2 can be computed in linear time by Lemma 11. These operations are independent of the value t and need to be carried out only once. $\square$

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.

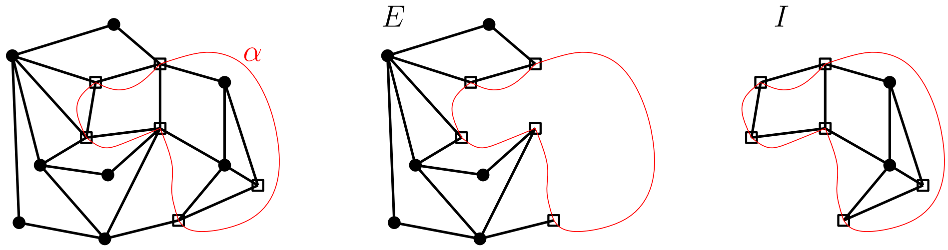

FIGURE 5. *Left: a curve α that is disjoint from the interior of the edges, and the set V_α marked with squares. Center and right: the two subgraphs E, I defined by α.*

THEOREM 17 (Klein [16]). *For a given embedded plane graph G with n vertices, and k vertex pairs $(u_1, v_1), \dots, (u_k, v_k)$, with all vertices u_i on a common face, we can compute the distances $\delta_G(u_j, v_j)$, $1 \le j \le k$, in $O(n \log n + k \log n)$ time.*

The previous theorem, together with the techniques developed by Fakcharoenphol and Rao [13] imply the following result, which is of independent interest; see Figure 5.

LEMMA 18. *Let G be an embedded planar graph with n vertices and let α be a Jordan curve passing through a subset V_α of c vertices and disjoint from the interior of the edges of G. We can compute in $O(n \log n + c^2 \log^2 c)$ time the distances $\delta_G(u, v)$ for all $u, v \in V_\alpha$.*

Proof. The curve α splits the graph into an interior part I and an exterior part E; the vertices V_α belong to both parts. See Figure 5. In each of the subgraphs, the vertices V_α lie on a face. By Theorem 17, 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 will use a modification of Dijkstra's algorithm to compute distances from a source $u \in V_\alpha$ to all vertices of V_α. The algorithm is based on a data structure $DS(E)$ whose properties we describe first. Assume that each vertex $v \in V_\alpha$ has a label $d_E(v) \ge 0$, and we have a set of active vertices $S_E \subset V_\alpha$. Fakcharoenphol and Rao [13, 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 (implicitly) 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 $d_E^0 = \min_{v \in S_E} d_E(v)$.
- `ExtractMin`$()$ returns $v_E = \arg\min_{v \in S_E} d_E(v)$ and makes it inactive: $S_E = S_E \setminus \{v_E\}$.

A similar data structure $DS(I)$ is built for maintaining labels d_I. In this case, the calls to `Relax`(v, d_v) in $DS(I)$ use the distances δ_I instead of δ_E.

Fix a vertex $u \in V_\alpha$. We now show how to compute $\delta_G(u, v)$ for all $v \in V_\alpha$ running Dijkstra's algorithm. However, the way to choose the next vertex is slightly more involved. We initialize data structures $DS(E)$ and $DS(I)$ with $S_E = S_I = V_\alpha$, and labels $d_E(u) = d_I(u) = 0$ and $d_E(v) = d_I(v) = \infty$ for all $v \in V_\alpha \setminus \{u\}$. This finishes the initialization. We now proceed like in Dijkstra's algorithm, but for each vertex we have to consider the minimum of the labels d_E, d_I, corresponding to the shortest connection arriving from E or from I, respectively. More precisely, at each round we call `FindMin`$()$ in $DS(E)$ and $DS(I)$. Let us assume that $d_E^0 \le d_I^0$; the other case is symmetric. We then call `ExtractMin`$()$ in $DS(E)$, return $d_E(v_E)$ as the value $\delta_G(u, v_E)$, call $\texttt{Relax}(v_E, d_E(v_E))$ in $DS(E)$ and $DS(I)$, and start a new round of Dijkstra's algorithm. The procedure finishes when S_E or S_I are empty. (As the algorithm is stated, each vertex $v \in V_\alpha$ is actually extracted twice in succession: after extracting it from DS_E or DS_I for the first time, it will immediately be found by $\texttt{FindMin}()$ in the other structure. One could improve the interface of $DS(\cdot)$ to avoid this.)

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. $\square$

5.2. Decompositions. We use the hierarchical decomposition of planar graphs as given by Fakcharoenphol and Rao [13]. Let G be an embedded plane graph. 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 uv 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 that is not a facial walk of 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 L and R, using a Jordan curve α_P that passes through vertices of P but does not cross any edge of P, until pieces with $O(1)$ vertices are obtained. The vertices V_{α_P} crossed by α_P go to both parts L and R. If any part has several connected components, we treat each separately; for simplicity we assume that both L, R are connected. Note that the vertices V_{α_P} form part of a hole in L, R, and that a boundary vertex of L or R is a boundary vertex of P or a vertex of V_{α_P}. We denote this recursive decomposition by (Π, T_Π), where Π is the collection of pieces appearing through the decomposition and T_Π is a rooted binary tree with a node for each piece $P \in \Pi$, with the node G as root, and with edges from piece P to pieces L, R whenever subpieces L, R are the pieces arising from partitioning piece P.

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 [21]: given a piece P_i with weights in the vertices, we can find 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 19. *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. As a consequence, the sums $\sum m_i$ and $\sum b_i^2$ over the pieces $P_i \in \Pi$ at *all* levels are $O(n \log n)$. This (weaker) statement is implicit in [13], but we give a detailed proof here.

Proof. First, we argue that the total number of vertices in the curves used through the decomposition, counted with multiplicity, is $O(n)$. Let $A(n)$ be the number of vertices in all curves used through the hierarchical decomposition of a piece of size n.

In each round we divide the piece in up to 8 pieces of size $m_k,\ k = 1 \ldots 8$, and we have introduced $O(\sqrt{n})$ new vertices. Thus, we have the recurrence $A(n) = O(\sqrt{n}) + \sum_{i=k}^{8} A(m_k)$, with $\sum_{k=1}^{8} m_k = n + O(\sqrt{n})$ and $m_k \leq 2n/3 + O(\sqrt{n}),\ k = 1 \ldots 8$. It follows by induction that $A(n) = O(n)$.

We can now see that $\sum_{P_i \in \Pi_d} m_i = O(n)$. Each vertex of $V(G)$ is either in only one piece of $P_i \in \Pi_d$, or in several. Clearly, we can have at most $n = |V(G)|$ of the former type. Each copy of the latter type appears when a curve that is used for partitioning passes through the vertex, and we can have at most $O(n)$ of those because of the previous paragraph.

In each round the number of vertices in the pieces decreases geometrically, and therefore we have a logarithmic number of levels. In each piece P_i we spend $O(m_i)$ time to find the curve α_{P_i} used to decompose P_i, and the $O(n \log n)$ running time follows. Regarding the holes per piece, in each round $O(1)$ new holes are introduced, but each subpiece gets at most $2/3$ of the holes of its parent. It follows that each piece has $O(1)$ holes.

It remains to bound $B_d := \sum_{P_i \in \Pi_d} (b_i)^2$. Consider a level $d \equiv 2 \pmod 3$ where in each piece we are about to partition the number of boundary vertices. Let P_0 be a piece in Π_d, and let $P_k,\ k = 1 \ldots 8$ be the eight subpieces resulting from P in a round; the nodes corresponding to $P_k,\ k = 1 \ldots 8$ are descendants of P_0 in T_Π. Each vertex of ∂P_0 goes to exactly one of $\partial P_k,\ k = 1 \ldots 8$, unless some splitting curve passes through it in the round, in which case it goes to several subpiece boundaries. Therefore, we have $\sum_{k=1}^{8} b_k \leq b_0 + O(\sqrt{m_0})$, and using that $b_k \leq (2/3)b_0 + O(\sqrt{m_0}),\ k = 1 \ldots 8$, we obtain

$$\begin{aligned} \sum_{k=1}^{8} (b_k)^2 &\leq ((2/3)b_0 + O(\sqrt{m_0}))^2 + ((1/3)b_0 + O(\sqrt{m_0}))^2 \\ &\leq \tfrac{5}{9}(b_0)^2 + O(m_0 + b_0\sqrt{m_0}). \end{aligned}$$

Therefore we have

$$\begin{aligned} B_d &= \sum_{P_i \in \Pi_d} (b_i)^2 \leq \sum_{P_i \in \Pi_{d-3}} \left(\tfrac{5}{9}(b_i)^2 + O(m_i + b_i\sqrt{m_i})\right) \\ &= \tfrac{5}{9} B_{d-3} + O\left(\sum_{P_i \in \Pi_{d-3}} m_i\right) + O\left(\sum_{P_i \in \Pi_{d-3}} b_i\sqrt{m_i}\right) \\ &\leq \tfrac{5}{9} B_{d-3} + O(n) + O\left(\sqrt{\sum_{P_i \in \Pi_{d-3}} b_i^2}\sqrt{\sum_{P_i \in \Pi_{d-3}} m_i}\right) \\ &= \tfrac{5}{9} B_{d-3} + O(n) + O(\sqrt{B_{d-3} n}), \end{aligned}$$

where we have used the Cauchy-Schwarz inequality in the last inequality. It follows now by induction that $B_d = O(n)$ for $d \equiv 2 \pmod 3$. For any d we have $B_d = O(B_{d-1}) = O(B_{d-2})$, and therefore the bound $B_d = O(n)$ extends to all d. □

Applying for each piece $P \in \Pi$ Lemma 18 to α_P and Theorem 17, once per hole, and obtain:

LEMMA 20. *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* $P \in \Pi$.

Proof. Consider a piece $P \in \Pi$ of size m and with b boundary vertices, and the corresponding curve α_P passing through $O(\sqrt{m})$ vertices V_{α_P}. The distances we want to find in P can be divided into two groups:

- The distances $\delta_P(u, v)$ where $u \in \partial P$ and $v \in (\partial P \cup V_{\alpha_P})$. Since P has $O(1)$ holes, ∂P is contained in $O(1)$ facial walks and we can compute these distances applying $O(1)$ times Theorem 17 in P, once for each hole. We then spend

$$
\begin{aligned}
O(m \log m + (|\partial P|)(|\partial P| + |V_{\alpha_P}|) \log m) &= O(m \log n + (b^2 + b\sqrt{m}) \log n) \\
&= O(m \log n + b^2 \log n)
\end{aligned}
$$

time.
- The distances $\delta_G(u, v)$ where $u, v \in V_{\alpha_P}$. These distances can be computed using Lemma 18, and we spend $O(m \log n + |V_{\alpha_P}|^2 \log^2 n) = O(m \log^2 n)$ time.

Therefore, we find the relevant distances in a piece P using $O(m \log^2 n + b^2 \log n)$ time. Using the bounds in Lemma 19, we see that the total time we need is bounded by

$$
O\left(\sum_{P_i \in \Pi} (m_i \log^2 n + (b_i)^2 \log n) \right) = O(n \log^3 n). \qquad \square
$$

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 [17] 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. [14] imply that $T_{\text{sssp}}(G_+) = O(n)$. From Lemma 4 and Theorem 7, we conclude the following.

THEOREM 21. *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.* $\square$

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 22. *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 Section 5.2. For each piece $P \in \Pi$, we compute and store the distances described in Lemma 20.

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 in computing the distances $\delta_{G_+}(s, v)$ for all $v \in V(G)$. For each piece $P \in P$, let P_+ be the subgraph of G_+ obtained by adding to P 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 all $v \in (\partial P \cup V_{\alpha_P})$. Each piece P that corresponds to a leaf of T_Π has constant size, and we can compute the values $\delta_{P_+}(s, v), v \in V(P)$, in $O(1)$ time. For a piece P with two subpieces Q, R we have that, for any $v \in (\partial P \cup V_{\alpha_P})$,

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

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 $v \in (\partial P \cup V_{\alpha_P})$. At the root, we obtained this data from the bottom-top traversal. For a piece P which is a child of another piece Q and for any $v \in (\partial P \cup V_{\alpha_P})$ we have

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

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 $v \in \partial P$, P a leaf of T_Π. The distances $\delta_{G_+}(s,v)$ for the remaining vertices are found using (3), which holds for any vertex $v \in P$ in each piece corresponding to a leaf of T_Π.

This finishes the description of the decision algorithm. We analyze its running time in view of applying parametric search [19]. The hierarchical decomposition and the use of Lemma 20 is done once at the beginning and takes $O(n \log^3 n)$ time. In a piece P, using (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 in $T_p = O(\log n)$ rounds, each round involving $T_r = O(n)$ operations that can be made in arbitrary order. The operations that involve the input data are just additions and comparisons. Thus, the framework of standard parametric search [19] can be applied; see Section 2.3. It leads to an optimization algorithm making T_p rounds, where each round uses T_r time plus the time used to solve $O(\log T_r)$ decision problems. The decision problem is a covering problem, and by Theorem 21 it can be solved in $T_d = O(n \log n)$ time. The overall running time for parametric search is therefore $O(T_p(T_r + T_d \log T_r)) = O(n \log^2 n)$. The dominating term in the running time comes from Lemma 20. $\square$

6. Conclusions. We have proposed algorithms for finding obnoxious centers in graphs. It is worth noting the similarity between our solution for planar graphs and graphs with bounded treewidth. In both cases we use a decomposition of depth $O(\log n)$, compute some distances within each piece efficiently, and use a bottom-up and top-down pass. For planar graphs we cannot afford to deal with the whole piece when passing information between pieces, and therfore, we only look at the boundary.

We have described how to find an obnoxious center in trees in $O(n \log n)$ time. However, no superlinear lower bound is known for this problem. We conjecture that our solution for trees, and more generally for graphs of bounded treewidth, is asymptotically optimal.

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, vol. 3827 of Lecture Notes in Computer Science, Springer-Verlag, 2005, pp. 693–703.

[2] B. Ben-Moshe, M.J. Katz, and M. Segal, *Obnoxious facility location: complete service with minimal harm*, International J. of Computational Geometry and Applications, 10 (2000), pp. 581–592.

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

[4] ———, *Treewidth: Algorithmic techniques and results*, in Proceedings MFCS'97, vol. 1295 of Lecture Notes in Computer Science, Springer-Verlag, 1997, pp. 19–36.

[5] ———, *A partial k-arboretum of graphs with bounded treewidth*, Theor. Comput. Sci., 209 (1998), pp. 1–45.

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

[7] R. E. Burkard, H. Dollani, Y. Lin, and G. Rote, *The obnoxious center problem on a tree*, SIAM J. Discrete Math., 14 (2001), pp. 498–509.

[8] S. Cabello and G. Rote, *Obonoxious centers in graphs*, in Proc. 18th ACM-SIAM Sympos. Disc. Alg. (SODA), 2007, pp. 98–107.

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

[10] ———, *Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms*, Algorithmica, 27 (2000), pp. 212–226.

[11] R. L. Church and R. S. Garfinkel, *Locating an obnoxious facility on a network*, Transport. Science, 12 (1978), pp. 107–118.

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

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

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

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

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

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

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

[19] ———, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM, 30 (1983), pp. 852–865.

[20] ———, *Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems*, SIAM J. Comput., 12 (1983), pp. 759–776.

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

[22] B. A. Reed, *Finding approximate separators and computing tree width quickly*, in Proc. 24th ACM Sympos. Theory Computing (STOC), 1992, pp. 221–228.

[23] M. Segal, *Placing an obnoxious facility in geometric networks*, Nordic J. of Computing, 10 (2003), pp. 224–237.

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

[25] ———, *Obnoxious facility location on graphs*, SIAM J. Discrete Math., 4 (1991), pp. 550–567.

[26] ———, *Locating two obnoxious facilities using the weighted maximin criterion*, Operations Research Letters, 34 (2006), pp. 97–105.

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