Theoretical Computer Science 755 (2019) 48–64

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Parameterized approximation algorithms for some location problems in graphs ☆

Arne Leitert [a,*], Feodor F. Dragan [b]

[a] Department of Computer Science, Central Washington University, Ellensburg, WA 98926, USA
[b] Department of Computer Science, Kent State University, Kent, OH 44240, USA

ARTICLE INFO

Article history:
Received 29 April 2018
Accepted 30 June 2018
Available online 14 July 2018
Communicated by V.Th. Paschos

Keywords:
Graph algorithms
Connected r-domination
Connected p-center
Layering partition
Tree-length
Tree-breadth

ABSTRACT

We develop efficient parameterized, with additive error, approximation algorithms for the (Connected) r-Domination problem and the (Connected) p-Center problem for unweighted and undirected graphs. Given a graph G, we show how to construct a (connected) $\big(r + \mathcal{O}(\mu)\big)$-dominating set D with $|D| \leq |D^*|$ efficiently. Here, D^* is a minimum (connected) r-dominating set of G and μ is our graph parameter, which is the *tree-breadth* or the *cluster diameter in a layering partition* of G. Additionally, we show that a $+\mathcal{O}(\mu)$-approximation for the (Connected) p-Center problem on G can be computed in polynomial time. Our interest in these parameters stems from the fact that in many real-world networks, including Internet application networks, web networks, collaboration networks, social networks, biological networks, and others, and in many structured classes of graphs these parameters are small constants.

1. Introduction

The (Connected) r-Domination problem and the (Connected) p-Center problem, along with the p-Median problem, are among basic facility location problems with many applications in data clustering, network design, operations research – to name a few. Let $G = (V, E)$ be an unweighted and undirected graph. Given a radius $r(v) \in \mathbb{N}$ for each vertex v of G, indicating within what radius a vertex v wants to be served, the *r-Domination problem* asks to find a set $D \subseteq V$ of minimum cardinality such that $d_G(v, D) \leq r(v)$ for every $v \in V$. The *Connected r-Domination problem* asks to find an r-dominating set D of minimum cardinality with an additional requirement that D needs to induce a connected subgraph of G. When $r(v) = 1$ for every $v \in V$, one gets the classical (Connected) Domination problem. Note that the Connected r-Domination problem is a natural generalisation of the Steiner Tree problem (where each vertex t in the target set has $r(t) = 0$ and each other vertex s has $r(s) = \mathrm{diam}(G)$). The connectedness of D is important also in network design and analysis applications (e.g. in finding a small backbone of a network). It is easy to see also that finding minimum connected dominating sets is equivalent to finding spanning trees with the maximum possible number of leaves.

The (closely related) *p-Center problem* asks to find in G a set $C \subseteq V$ of at most p vertices such that the value $\max_{v \in V} d_G(v, C)$ is minimised. If, additionally, C is required to induce a connected subgraph of G, then one gets the *Connected p-Center problem*.

☆ Results of this paper were partially presented at the COCOA 2017 conference [24].

* Corresponding author.
E-mail addresses: arne.leitert@cwu.edu (A. Leitert), dragan@cs.kent.edu (F.F. Dragan).

https://doi.org/10.1016/j.tcs.2018.06.049
0304-3975/

The domination problem is one of the most well-studied NP-hard problems in algorithmic graph theory. To cope with the intractability of this problem, it has been studied both in terms of approximability (relaxing the optimality) and fixed-parameter tractability (relaxing the runtime). The Domination problem is notorious in the theory of fixed-parameter tractability (see, e.g., [15,26] for an introduction to parameterized complexity). It was the first problem to be shown W[2]-complete [15], and it is hence unlikely to be FPT, i.e., unlikely to have an algorithm with runtime $f(k)n^c$ for f a computable function, k the size of an optimal solution, c a constant, and n the number of vertices of the input graph. Similar results are known also for the connected domination problem [22]. From the approximability prospective, a logarithmic approximation factor can be found by using a simple greedy algorithm, and finding a sublogarithmic approximation factor is NP-hard [27]. The problem is in fact Log-APX-complete [18] and it is unlikely that there is a good FPT approximation algorithm for it (see [6] and [7]).

The p-Center problem is known to be NP-hard on graphs. However, for it, a simple and efficient factor-2 approximation algorithm exists [21]. Furthermore, it is a best possible approximation algorithm in the sense that an approximation with factor less than 2 is proven to be NP-hard (see [21] for more details). The NP-hardness of the Connected p-Center problem is shown in [28]. For the edge-weighted variant of the p-center problem, [19] shows that a $(2-\epsilon)$-approximation is W[2]-hard for parameter p and NP-hard for graphs of highway dimension $h = \mathcal{O}(\log^2 n)$, while also offering a 3/2-approximation algorithm of running time $2^{\mathcal{O}(ph\log h)}n^{\mathcal{O}(1)}$. A variant of the p-center problem, called the (k,r)-*center problem* is investigated in [3,13,25]. It asks whether a given graph G has at most k vertices (called centers) such that every other vertex of G is within distance at most r from some center. It is shown [25] that the (k,r)-center problem parameterized by the number k of centers is W[1]-hard in the L_∞ metric. From the positive side, [3] gives an $\mathcal{O}\big((2r+1)^{\mathrm{tw}}n\big)$ time algorithm for the (k,r)-center problem and a randomized $\mathcal{O}\big((2r+2)^{\mathrm{tw}}n^{\mathcal{O}(1)}\big)$ time algorithm for the connected (k,r)-center problem in n-vertex graphs of tree-width tw. Additionally, the (k,r)-center problem, parameterized by k and r, is fixed-parameter tractable (FPT) on planar graphs, i.e., it admits an algorithm of complexity $2^{\mathcal{O}(r\log r)\sqrt{k}}n^{\mathcal{O}(1)}$ [13]. Moreover, the same type of FPT algorithms can be designed for the more general class of map graphs (see [13] for details).

Recently, in [10], a new type of approximability result (call it a *parameterized approximability* result) was obtained: there exists a polynomial time algorithm which finds in an arbitrary graph G having a minimum r-dominating set D a set D' such that $|D'| \leq |D|$ and each vertex $v \in V$ is within distance at most $r(v) + 2\delta$ from D', where δ is the hyperbolicity parameter of G (see [10] for details). We call such a D' an $(r+2\delta)$-*dominating set* of G. Later, in [17], this idea was extended to the p-Center problem: there is a quasi-linear time algorithm for the p-Center problem with an additive error less than or equal to six times the input graph's hyperbolicity (i.e., it finds a set C' with at most p vertices such that $\max_{v\in V} d_G(v,C') \leq \min_{C\subseteq V,|C|\leq p} \max_{v\in V} d_G(v,C) + 6\delta$). We call such a C' a $+6\delta$-*approximation for the p-Center problem*.

In this paper, we continue the line of research started in [10] and [17]. Unfortunately, the results of [10,17] are hardly extendable to connected versions of the r-Domination and p-Center problems. It remains an open question whether similar approximability results parameterized by the graph's hyperbolicity can be obtained for the Connected r-Domination and Connected p-Center problems. Instead, we consider two other graph parameters: the *tree-breadth* ρ and the *cluster diameter* Δ *in a layering partition* (formal definitions will be given in the next sections). Both parameters (like the hyperbolicity) capture the metric tree-likeness of a graph (see, e.g., [2] and papers cited therein). As demonstrated in [2], in many real-world networks, including Internet application networks, web networks, collaboration networks, social networks, biological networks, and others, as well as in many structured classes of graphs the parameters δ, ρ, and Δ are small constants.

We show here that, for a given n-vertex, m-edge graph G, having a minimum r-dominating set D and a minimum connected r-dominating set C:

- an $(r+\Delta)$-dominating set D' with $|D'| \leq |D|$ can be computed in linear time;
- a connected $(r+2\Delta)$-dominating set C' with $|C'| \leq |C|$ can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\Delta\big)$ time (where $\alpha(n)$ is the inverse Ackermann function);
- a $+\Delta$-approximation for the p-Center problem can be computed in linear time;
- a $+2\Delta$-approximation for the connected p-Center problem can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\min(\Delta,p)\big)$ time.

Furthermore, given a tree-decomposition with breadth ρ for G:

- an $(r+\rho)$-dominating set D' with $|D'| \leq |D|$ can be computed in $\mathcal{O}(nm)$ time;
- a connected $\big(r+5\rho\big)$-dominating set C' with $|C'| \leq |C|$ can be computed in $\mathcal{O}(nm)$ time;
- a $+\rho$-approximation for the p-Center problem can be computed in $\mathcal{O}(nm\log n)$ time;
- a $+5\rho$-approximation for the Connected p-Center problem can be computed in $\mathcal{O}(nm\log n)$ time.

To compare these results with the results of [10,17], notice that, for any graph G, its hyperbolicity δ is at most Δ [2] and at most two times its tree-breadth ρ [9], and the inequalities are sharp.

Note that, for split graphs (graphs in which the vertices can be partitioned into a clique and an independent set), δ and ρ are at most 1, and Δ is at most 2. Additionally, as shown in [11], there is (under reasonable assumptions) no polynomial-time algorithm to compute a sublogarithmic-factor approximation for the (Connected) Domination problem in split graphs. Hence, there is no such algorithm even for constant δ, ρ, and Δ.

One can extend this result to show that there is no polynomial-time algorithm $\mathcal{A}$ which computes, for any constant c, a $+c\log n$-approximation for split graphs. Hence, there is no polynomial-time $+c\Delta\log n$-approximation algorithm in general. Consider a given split graph $G=(C\cup I,E)$ with n vertices where C induces a clique and I induces an independent set. Create a graph $H=(C_H\cup I_H,E_H)$ by, first, making n copies of G. Let $C_H=C_1\cup C_2\cup\ldots\cup C_n$ and $I_H=I_1\cup I_2\cup\ldots\cup I_n$. Second, make the vertices in C_H pairwise adjacent. Then, C_H induces a clique and I_H induces an independent set. If there is such an algorithm $\mathcal{A}$, then $\mathcal{A}$ produces a (connected) dominating set $D_{\mathcal{A}}$ for H which has at most $2c\log n$ more vertices than a minimum (connected) dominating set D. Thus, by pigeonhole principle, H contains a clique C_i for which $|C_i\cap D_{\mathcal{A}}|=|C_i\cap D|$. Therefore, such an algorithm $\mathcal{A}$ would allow to solve the (Connected) Domination problem for split graphs in polynomial time.

After an extended abstract of these results was published in [24], we learned about new results (including approximation results) obtained in [23] for the (k,r)-center problem. For any $r\geq 1$, an algorithm that solves the problem in $\mathcal{O}\big((3r+1)^{\mathrm{cw}}n^{\mathcal{O}(1)}\big)$ time, where cw is the clique-width of the input graph, as well as a tight SETH lower bound matching this algorithm's performance are presented. Furthermore, algorithms are presented that, for any $\epsilon>0$, run in time $\mathcal{O}\big((\mathrm{tw}/\epsilon)^{\mathcal{O}(\mathrm{tw})}n^{\mathcal{O}(1)}\big)$, $\mathcal{O}\big((\mathrm{cw}/\epsilon)^{\mathcal{O}(\mathrm{cw})}n^{\mathcal{O}(1)}\big)$ and return a $(k,(1+\epsilon)r)$-center if a (k,r)-center exists. Although these approximation results have a flavor of our approximation results in a sense that they keep k unchanged and extend the value of r, the algorithms of [23] are exponential in tw | cw and have a multiplicative approximation factor $(1+\epsilon)$; our algorithms are low-polynomial and have additive approximation surpluses. On the other hand, the approximation results of [23] cannot really be compared with ours, as there are graphs (e.g. large cliques) with tree-length 1 and unbounded tree-width and there are graphs (e.g. large cycles) with tree-width 2 and unbounded tree-length.

2. Preliminaries

All graphs occurring in this paper are connected, finite, unweighted, undirected, without loops, and without multiple edges. For a graph $G=(V,E)$, we use $n=|V|$ and $m=|E|$ to denote the cardinality of the vertex set and the edge set of G, respectively.

The *length* of a path from a vertex v to a vertex u is the number of edges in the path. The *distance* $d_G(u,v)$ in a graph G of two vertices u and v is the length of a shortest path connecting u and v. The distance between a vertex v and a set $S\subseteq V$ is defined as $d_G(v,S)=\min_{u\in S}d_G(u,v)$. For a vertex v of G and some positive integer r, the set $N^r_G[v]=\{u\mid d_G(u,v)\leq r\}$ is called the *r-neighbourhood* of v. The *eccentricity* $\mathrm{ecc}_G(v)$ of a vertex v is $\max_{u\in V}d_G(u,v)$. For a set $S\subseteq V$, its eccentricity is $\mathrm{ecc}_G(S)=\max_{u\in V}d_G(u,S)$.

For some function $r\colon V\to\mathbb{N}$, a vertex u is *r-dominated* by a vertex v (by a set $S\subseteq V$), if $d_G(u,v)\leq r(u)$ ($d_G(u,S)\leq r(u)$, respectively). A vertex set D is called an *r-dominating set* of G if each vertex $u\in V$ is r dominated by D. Additionally, for some non-negative integer ϕ, we say a vertex is *$(r+\phi)$-dominated* by a vertex v (by a set $S\subseteq V$), if $d_G(u,v)\leq r(u)+\phi$ ($d_G(u,S)\leq r(u)+\phi$, respectively). An *$(r+\phi)$-dominating set* is defined accordingly. For a given graph G and function r, the *(Connected) r-Domination* problem asks for the smallest (connected) vertex set D such that D is an r-dominating set of G.

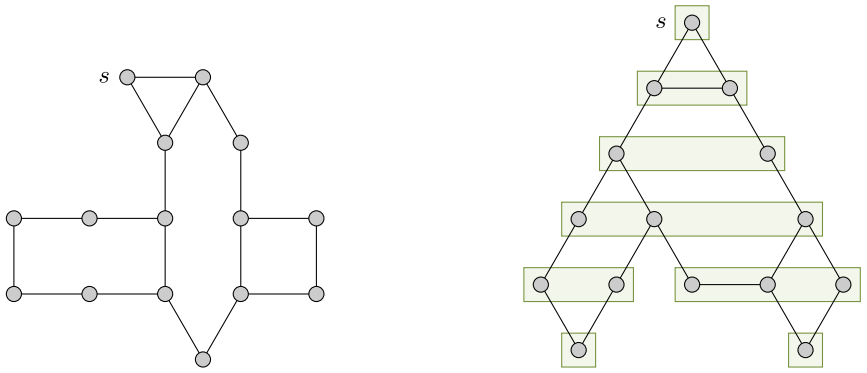
The *degree* of a vertex v is the number of vertices adjacent to it. For a vertex set S, let $G[S]$ denote the subgraph of G induced by S. A vertex set S is a *separator* for two vertices u and v in G if each path from u to v contains a vertex $s\in S$; in this case we say S *separates* u from v.

A *tree-decomposition* of a graph $G=(V,E)$ is a tree T with the vertex set $\mathcal{B}$ where each vertex of T, called bag, is a subset of V such that: (i) $V=\bigcup_{B\in\mathcal{B}}B$, (ii) for each edge $uv\in E$, there is a bag $B\in\mathcal{B}$ with $u,v\in B$, and (iii) for each vertex $v\in V$, the bags containing v induce a subtree of T. A tree-decomposition T of G has *breadth* ρ if, for each bag B of T, there is a vertex v in G with $B\subseteq N^{\rho}_G[v]$. The *tree-breadth* of a graph G is ρ, written as $\mathrm{tb}(G)=\rho$, if ρ is the minimal breadth of all tree-decomposition for G. A tree-decomposition T of G has *length* λ if, for each bag B of T and any two vertices $u,v\in B$, $d_G(u,v)\leq\lambda$. The *tree-length* of a graph G is λ, written as $\mathrm{tl}(G)=\lambda$, if λ is the minimal length of all tree-decomposition for G.

For a rooted tree T, let $\Lambda(T)$ denote the number of leaves of T. For the case when T contains only one node, let $\Lambda(T):=0$. With α, we denote the inverse Ackermann function (see, e.g., [12]). It is well known that α grows extremely slowly. For $x=10^{80}$ (estimated number of atoms in the universe), $\alpha(x)\leq 4$.

3. Using a layering partition

The concept of a *layering partition* was introduced in [5,8]. The idea is the following. First, partition the vertices of a given graph $G=(V,E)$ in distance layers $L_i=\{v\mid d_G(s,v)=i\}$ for a given vertex s. Second, partition each layer L_i into *clusters* in such a way that two vertices u and v are in the same cluster if and only if they are connected by a path only using vertices in the same or upper layers. That is, u and v are in the same cluster if and only if, for some i, $\{u,v\}\subseteq L_i$ and there is a path P from u to v in G such that, for all $j<i$, $P\cap L_j=\emptyset$. Note that each cluster C is a set of vertices of G, i.e., $C\subseteq V$, and all clusters are pairwise disjoint. The created clusters form a rooted tree $\mathcal{T}$ with the cluster $\{s\}$ as the root where each cluster is a node of $\mathcal{T}$ and two clusters C and C' are adjacent in $\mathcal{T}$ if and only if G contains an edge uv with $u\in C$ and $v\in C'$. Fig. 1 gives an example for such a partition. A layering partition of a graph can be computed in linear time [8].



(a) A graph G. (b) A layering partition $\mathcal{T}$ for G.

Fig. 1. Example of a layering partition. A given graph G (a) and the layering partition of G generated when starting at vertex s (b). Example taken from [8].

For the remainder of this section, assume that we are given a graph $G = (V, E)$ and a layering partition $\mathcal{T}$ of G for an arbitrary start vertex. We denote the largest diameter of all clusters of $\mathcal{T}$ as Δ, i.e., $\Delta := \max\{d_G(x, y) \mid x, y \text{ are in a cluster } C \text{ of } \mathcal{T}\}$. For two vertices u and v of G contained in the clusters C_u and C_v of $\mathcal{T}$, respectively, we define $d_{\mathcal{T}}(u, v) := d_{\mathcal{T}}(C_u, C_v)$.

Lemma 1. *For all vertices u and v of G, $d_{\mathcal{T}}(u, v) \leq d_G(u, v) \leq d_{\mathcal{T}}(u, v) + \Delta$.*

Proof. Clearly, by construction of a layering partition, $d_{\mathcal{T}}(u, v) \leq d_G(u, v)$ for all vertices u and v of G.

Next, let C_u and C_v be the clusters containing u and v, respectively. Note that $\mathcal{T}$ is a rooted tree. Let C' be the lowest common ancestor of C_u and C_v. Therefore, $d_{\mathcal{T}}(u, v) = d_{\mathcal{T}}(u, C') + d_{\mathcal{T}}(C', v)$. By construction of a layering partition, C' contains a vertex u' and vertex v' such that $d_G(u, u') = d_{\mathcal{T}}(u, u')$ and $d_G(v, v') = d_{\mathcal{T}}(v, v')$. Since the diameter of each cluster is at most Δ, $d_G(u, v) \leq d_{\mathcal{T}}(u, u') + \Delta + d_{\mathcal{T}}(v, v') = d_{\mathcal{T}}(u, v) + \Delta$. □

Theorem 2 below shows that we can use the layering partition $\mathcal{T}$ to compute an $(r + \Delta)$-dominating set for G in linear time which is not larger than a minimum r-dominating set for G. This is done by finding a minimum r-dominating set of $\mathcal{T}$ where, for each cluster C of $\mathcal{T}$, $r(C)$ is defined as $\min_{v \in C} r(v)$.

Theorem 2. *Let D be a minimum r-dominating set for a given graph G. An $(r + \Delta)$-dominating set D' for G with $|D'| \leq |D|$ can be computed in linear time.*

Proof. First, create a layering partition $\mathcal{T}$ of G and, for each cluster C of $\mathcal{T}$, set $r(C) := \min_{v \in C} r(v)$. Second, find a minimum r-dominating set $\mathcal{S}$ for $\mathcal{T}$, i.e., a set $\mathcal{S}$ of clusters such that, for each cluster C of $\mathcal{T}$, $d_{\mathcal{T}}(C, \mathcal{S}) \leq r(C)$. Third, create a set D' by picking an arbitrary vertex of G from each cluster in $\mathcal{S}$. All three steps can be performed in linear time, including the computation of $\mathcal{S}$ (see [4]).

Next, we show that D' is an $(r + \Delta)$-dominating set for G. By construction of $\mathcal{S}$, each cluster C of $\mathcal{T}$ has distance at most $r(C)$ to $\mathcal{S}$ in $\mathcal{T}$. Thus, for each vertex u of G, $\mathcal{S}$ contains a cluster $C_{\mathcal{S}}$ with $d_{\mathcal{T}}(u, C_{\mathcal{S}}) \leq r(u)$. Additionally, by Lemma 1, $d_G(u, v) \leq r(u) + \Delta$ for any vertex $v \in C_{\mathcal{S}}$. Therefore, for any vertex u, $d_G(u, D') \leq r(u) + \Delta$, i.e., D' is an $(r + \Delta)$-dominating set for G.

It remains to show that $|D'| \leq |D|$. Let $\mathcal{D}$ be the set of clusters of $\mathcal{T}$ that contain a vertex of D. Because D is an r-dominating set for G, it follows from Lemma 1 that $\mathcal{D}$ is an r-dominating set for $\mathcal{T}$. Clearly, since clusters are pairwise disjoint, $|\mathcal{D}| \leq |D|$. By minimality of $\mathcal{S}$, $|\mathcal{S}| \leq |\mathcal{D}|$ and, by construction of D', $|D'| = |\mathcal{S}|$. Therefore, $|D'| \leq |D|$. □

We now show how to construct a connected $(r + 2\Delta)$-dominating set for G using $\mathcal{T}$ in such a way that the set created is not larger than a minimum connected r-dominating set for G. For the remainder of this section, let D_r be a minimum connected r-dominating set of G and let, for each cluster C of $\mathcal{T}$, $r(C)$ be defined as above. Additionally, we say that a subtree T' of some tree T is an *r-dominating subtree of T* if the nodes (clusters in case of a layering partition) of T' form a connected r-dominating set for T.

The first step of our approach is to construct a minimum r-dominating subtree T_r of $\mathcal{T}$. Such a subtree T_r can be computed in linear time [16]. Lemma 3 below shows that T_r gives a lower bound for the cardinality of D_r.

Lemma 3. *If T_r contains more than one cluster, each connected r-dominating set of G intersects all clusters of T_r. Therefore, $|T_r| \leq |D_r|$.*

Proof. Let D be an arbitrary connected r-dominating set of G. Assume that T_r has a cluster C such that $C \cap D = \emptyset$. Because D is connected, the subtree of $\mathcal{T}$ induced by the clusters intersecting D is connected, too. Thus, if D intersects all leafs of T_r, then it intersects all clusters of T_r. Hence, we can assume, without loss of generality, that C is a leaf of T_r. Because T_r has at least two clusters and by minimality of T_r, $\mathcal{T}$ contains a cluster C' such that $d_{\mathcal{T}}(C', C) = d_{\mathcal{T}}(C', T_r) = r(C')$. Note that each path in G from a vertex in C' to a vertex in D intersects C. Therefore, by Lemma 1, there is a vertex $u \in C'$ with $r(u) = d_{\mathcal{T}}(u, C) < d_{\mathcal{T}}(u, D) \leq d_G(u, D)$. That contradicts with D being an r-dominating set.

Because any r-dominating set of G intersects each cluster of T_r and because these clusters are pairwise disjoint, it follows that $|T_r| \leq |D_r|$. □

As we show later in Corollary 5, each connected vertex set $S \subseteq V$ that intersects each cluster of T_r gives an $(r + \Delta)$-dominating set for G. It follows from Lemma 3 that, if such a set S has minimum cardinality, $|S| \leq |D_r|$. However, finding a minimum cardinality connected set intersecting each cluster of a layering partition (or of a subtree of it) is as hard as finding a minimum Steiner tree.

The main idea of our approach is to construct a minimum $(r + \delta)$-dominating subtree T_δ of $\mathcal{T}$ for some integer δ. We then compute a small enough connected set S_δ that intersects all cluster of T_δ. By trying different values of δ, we eventually construct a connected set S_δ such that $|S_\delta| \leq |T_r|$ and, thus, $|S_\delta| \leq |D_r|$. Additionally, we show that S_δ is a connected $(r + 2\Delta)$-dominating set of G.

For some non-negative integer δ, let T_δ be a minimum $(r + \delta)$-dominating subtree of $\mathcal{T}$. Clearly, $T_0 = T_r$. The following two lemmas set an upper bound for the maximum distance of a vertex of G to a vertex in a cluster of T_δ and for the size of T_δ compared to the size of T_r.

Lemma 4. *For each vertex v of G, $d_{\mathcal{T}}(v, T_\delta) \leq r(v) + \delta$.*

Proof. Let C_v be the cluster of $\mathcal{T}$ containing v and let C be the cluster of T_δ closest to C_v in $\mathcal{T}$. By construction of T_δ, $d_{\mathcal{T}}(v, C) = d_{\mathcal{T}}(C_v, C) \leq r(C_v) + \delta \leq r(v) + \delta$. □

Because the diameter of each cluster is at most Δ, Lemma 1 and Lemma 4 imply the following.

Corollary 5. *If a vertex set intersects all clusters of T_δ, it is an $\big(r + (\delta + \Delta)\big)$-dominating set of G.*

Lemma 6. $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$.

Proof. First, consider the case when T_δ contains only one cluster, i.e., $|T_\delta| = 1$. Then, $\Lambda(T_\delta) = 1$ and, thus, the statement clearly holds. Next, let T_δ contain more than one cluster, let C_u be an arbitrary leaf of T_δ, and let C_v be a cluster of T_r with maximum distance to C_u such that C_u is the only cluster on the shortest path from C_u to C_v in T_r, i.e., C_v is not in T_δ. Due to the minimality of T_δ, $d_{T_r}(C_u, C_v) = \delta$. Thus, the shortest path from C_u to C_v in T_r contains δ clusters (including C_v) which are not in T_δ. Therefore, $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$. □

Now that we have constructed and analysed T_δ, we show how to construct S_δ. First, we construct a set of shortest paths such that each cluster of T_δ is intersected by exactly one path. Second, we connect these paths with each other to form a connected set using an approach which is similar to Kruskal's algorithm for minimum spanning trees.

Let $\mathcal{L} = \{C_1, C_2, \ldots, C_\lambda\}$ be the leaf clusters of T_δ (excluding the root) with either $\lambda = \Lambda(T_\delta) - 1$ if the root of T_δ is a leaf, or with $\lambda = \Lambda(T_\delta)$ otherwise. We construct a set $\mathcal{P} = \{P_1, P_2, \ldots, P_\lambda\}$ of paths as follows. Initially, $\mathcal{P}$ is empty. For each cluster $C_i \in \mathcal{L}$, in turn, find the ancestor C_i' of C_i which is closest to the root of T_δ and does not intersect any path in $\mathcal{P}$ yet. If we assume that the indices of the clusters in $\mathcal{L}$ represent the order in which they are processed, then C_1' is the root of T_δ. Then, select an arbitrary vertex v in C_i and find a shortest path P_i in G from v to C_i'. Add P_i to $\mathcal{P}$ and continue with the next cluster in $\mathcal{L}$. Fig. 2 gives an example.

Lemma 7. *For each cluster C of T_δ, there is exactly one path $P_i \in \mathcal{P}$ intersecting C. Additionally, C and P_i share exactly one vertex, i.e., $|C \cap P_i| = 1$.*

Proof. Observe that, by construction of a layering partition, each vertex in a cluster C is adjacent to some vertex in the parent cluster of C. Therefore, a shortest path P in G from C to any of its ancestors C' only intersects clusters on the path from C to C' in $\mathcal{T}$ and each cluster shares only one vertex with P. It remains to show that each cluster intersects exactly one path.

Without loss of generality, assume that the indices of clusters in $\mathcal{L}$ and paths in $\mathcal{P}$ represent the order in which they are processed and created, i.e., assume that the algorithms first creates P_1 which starts in C_1, then P_2 which starts in C_2, and so on. Additionally, let $\mathcal{L}_i = \{C_1, C_2, \ldots, C_i\}$ and $\mathcal{P}_i = \{P_1, P_2, \ldots, P_i\}$.

To prove that each cluster intersects exactly one path, we show by induction over i that, if a cluster C_i of T_δ satisfies the statement, then all ancestors of C_i satisfy it, too. Thus, if C_λ satisfies the statement, each cluster satisfies it.

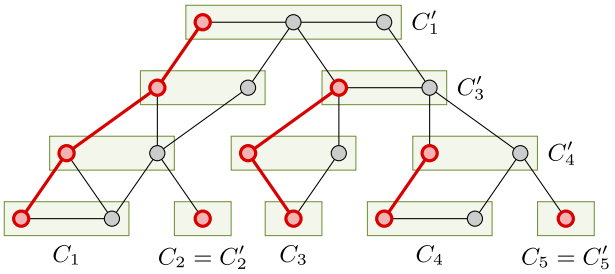


Fig. 2. Example for the set $\mathcal{P}$ for a subtree of a layering partition. Paths are shown in red. Each path P_i, with $1 \leq i \leq 5$, starts in the leaf C_i and ends in the cluster C'_i. For $i = 2$ and $i = 5$, P_i contains only one vertex. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

First, consider $i = 1$. Clearly, since P_1 is the first path, P_1 connects the leaf C_1 with the root of T_δ and no cluster intersects more than one path at this point. Therefore, the statement is true for C_1 and each of its ancestors.

Next, assume that $i > 1$ and that the statement is true for each cluster in $\mathcal{L}_{i-1}$ and their respective ancestors. Then, the algorithm creates P_i which connects the leaf C_i with the cluster C'_i. Assume that there is a cluster C on the path from C_i to C'_i in $\mathcal{T}$ such that C intersects a path P_j with $j < i$. Clearly, C'_i is an ancestor of C. Thus, by induction hypothesis, C'_i is also intersected by some path $P \neq P_i$. This contradicts with the way C'_i is selected by the algorithm. Therefore, each cluster on the path from C_i to C'_i in $\mathcal{T}$ only intersects P_i and P_i does not intersect any other clusters.

Because $i > 1$, C'_i has a parent cluster C'' in T_δ that is intersected by a path P_j with $j < i$. By induction hypothesis, each ancestor of C'' is intersected by a path in $\mathcal{P}_{i-1}$. Therefore, each ancestor of C_i is intersected by exactly one path in $\mathcal{P}_i$. □

Next, we use the paths in $\mathcal{P}$ to create the set S_δ. As first step, let $S_\delta := \bigcup_{P_i \in \mathcal{P}} P_i$. Later, we add more vertices into S_δ to ensure it is a connected set.

Now, create a partition $\mathcal{V} = \{V_1, V_2, \ldots, V_\lambda\}$ of V such that, for each i, $P_i \subseteq V_i$, V_i is connected, and $d_G(v, P_i) = \min_{P \in \mathcal{P}} d_G(v, P)$ for each vertex $v \in V_i$. That is, V_i contains the vertices of G which are not more distant to P_i in G than to any other path in $\mathcal{P}$. Additionally, for each vertex $v \in V$, set $P(v) := P_i$ if and only if $v \in V_i$ (i.e., $P(v)$ is the path in $\mathcal{P}$ which is closest to v) and set $d(v) := d_G\big(v, P(v)\big)$. Such a partition as well as $P(v)$ and $d(v)$ can be computed by performing a BFS on G starting at all paths $P_i \in \mathcal{P}$ simultaneously. Later, the BFS also allows us to easily determine the shortest path from v to $P(v)$ for each vertex v.

To manage the subsets of $\mathcal{V}$, we use a Union-Find data structure such that, for two vertices u and v, $\text{Find}(u) = \text{Find}(v)$ if and only if u and v are in the same set of $\mathcal{V}$. A Union-Find data structure additionally allows us to easily join two sets of $\mathcal{V}$ into one by performing a single Union operation. Note that, whenever we join two sets of $\mathcal{V}$ into one, $P(v)$ and $d(v)$ remain unchanged for each vertex v.

Next, create an edge set $E' = \{uv \mid \text{Find}(u) \neq \text{Find}(v)\}$, i.e., the set of edges uv such that u and v are in different sets of $\mathcal{V}$. Sort E' in such a way that an edge uv precedes an edge xy only if $d(u) + d(v) \leq d(x) + d(y)$.

The last step to create S_δ is similar to Kruskal's minimum spanning tree algorithm. Iterate over the edges in E' in increasing order. If, for an edge uv, $\text{Find}(u) \neq \text{Find}(v)$, i.e., if u and v are in different sets of $\mathcal{V}$, then join these sets into one by performing $\text{Union}(u, v)$, add the vertices on the shortest path from u to $P(u)$ to S_δ, and add the vertices on the shortest path from v to $P(v)$ to S_δ. Repeat this, until $\mathcal{V}$ contains only one set, i.e., until $\mathcal{V} = \{V\}$.

Algorithm 1 below summarises the steps to create a set S_δ for a given subtree of a layering partition subtree T_δ.

Lemma 8. *For a given graph G and a given subtree T_δ of some layering partition of G, Algorithm 1 constructs, in $\mathcal{O}\big(m\,\alpha(n)\big)$ time, a connected set S_δ with $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$ which intersects each cluster of T_δ.*

Proof (Correctness). First, we show that S_δ is connected at the end of the algorithm. To do so, we show by induction that, at any time, $S_\delta \cap V'$ is a connected set for each set $V' \in \mathcal{V}$. Clearly, when $\mathcal{V}$ is created, for each set $V_i \in \mathcal{V}$, $S_\delta \cap V_i = P_i$. Now, assume that the algorithm joins the set V_u and V_v in $\mathcal{V}$ into one set based on the edge uv with $u \in V_u$ and $v \in V_v$. Let $S_u = S_\delta \cap V_u$ and $S_v = S_\delta \cap V_v$. Note that $P(u) \subseteq S_u$ and $P(v) \subseteq S_v$. The algorithm now adds all vertices to S_δ which are on a path from $P(u)$ to $P(v)$. Therefore, $S_\delta \cap (V_u \cup V_v)$ is a connected set. Because $\mathcal{V} = \{V\}$ at the end of the algorithm, S_δ is connected eventually. Additionally, since $P_i \subseteq S_\delta$ for each $P_i \in \mathcal{P}$, it follows that S_δ intersects each cluster of T_δ.

Next, we show that the cardinality of S_δ is at most $|T_\delta| + \Delta \cdot \Lambda(T_\delta)$. When first created, the set S_δ contains all vertices of all paths in $\mathcal{P}$. Therefore, by Lemma 7, $|S_\delta| = \sum_{P_i \in \mathcal{P}} |P_i| = |T_\delta|$. Then, each time two sets of $\mathcal{V}$ are joined into one set based on an edge uv, S_δ is extended by the vertices on the shortest paths from u to $P(u)$ and from v to $P(v)$. Therefore, the size of S_δ increases by $d(u) + d(v)$, i.e., $|S_\delta| := |S_\delta| + d(u) + d(v)$. Let X denote the set of all edges used to join two sets of $\mathcal{V}$ into one at some point during the algorithm. Note that $|X| = |\mathcal{P}| - 1 \leq \Lambda(T_\delta)$. Therefore, at the end of the algorithm,

$$|S_\delta| = \sum_{P_i \in \mathcal{P}} |P_i| + \sum_{uv \in X} \big(d(u) + d(v)\big) \leq |T_\delta| + \Lambda(T_\delta) \cdot \max_{uv \in X} \big(d(u) + d(v)\big).$$

Algorithm 1: Computes a connected vertex set that intersects each cluster of a given layering partition.

Input: A graph $G = (V, E)$ and a subtree T_δ of some layering partition of G.
Output: A connected set $S_\delta \subseteq V$ that intersects each cluster of T_δ and contains at most $|T_\delta| + \big(\Lambda(T_\delta) - 1\big) \cdot \Delta$ vertices.

1 Let $\mathcal{L} = \{C_1, C_2, \ldots, C_\lambda\}$ be the set of clusters excluding the root that are leaves of T_δ.
2 Create an empty set $\mathcal{P}$.
3 **foreach** *cluster* $C_i \in \mathcal{L}$ **do**
4 Select an arbitrary vertex $v \in C_i$.
5 Find the highest ancestor C_i' of C_i (i.e., the ancestor which is closest to the root of T_δ) that is not flagged.
6 Find a shortest path P_i from v to an ancestor of v in C_i' (i.e., a shortest path from C_i to C_i' in G that contains exactly one vertex of each cluster of the corresponding path in T_δ).
7 Add P_i to $\mathcal{P}$.
8 Flag each cluster intersected by P_i.
9 Create a set $S_\delta := \bigcup_{P_i \in \mathcal{P}} P_i$.
10 Perform a BFS on G starting at all paths $P_i \in \mathcal{P}$ simultaneously. This results in a partition $\mathcal{V} = \{V_1, V_2, \ldots, V_\lambda\}$ of V with $P_i \subseteq V_i$ for each $P_i \in \mathcal{P}$. For each vertex v, set $P(v) := P_i$ if and only if $v \in V_i$ and let $d(v) := d_G(v, P(v))$.
11 Create a Union-Find data structure and add all vertices of G such that $\text{Find}(v) = i$ if and only if $v \in V_i$.
12 Determine the edge set $E' = \{uv \mid \text{Find}(u) \neq \text{Find}(v)\}$.
13 Sort E' such that $uv \leq xy$ if and only if $d(u) + d(v) \leq d(x) + d(y)$. Let $\langle e_1, e_2, \ldots, e_{|E'|} \rangle$ be the resulting sequence.
14 **for** $i := 1$ **to** $|E'|$ **do**
15 Let $uv = e_i$.
16 **if** $\text{Find}(u) \neq \text{Find}(v)$ **then**
17 Add the shortest path from u to $P(u)$ to S_δ.
18 Add the shortest path from v to $P(v)$ to S_δ.
19 $\text{Union}(u, v)$
20 Output S_δ.

Claim. *For each edge $uv \in X$, $d(u) + d(v) \leq \Delta$.*

Proof (Claim). To represent the relations between paths in $\mathcal{P}$ and vertex sets in $\mathcal{V}$, we define a function $f : \mathcal{P} \to \mathcal{V}$ such that $f(P_i) = V_j$ if and only if $P_i \subseteq V_j$. Directly after constructing $\mathcal{V}$, f is a bijection with $f(P_i) = V_i$. At the end of the algorithm, after all sets of $\mathcal{V}$ are joined into one, $f(P_i) = V$ for all $P_i \in \mathcal{P}$.

Recall the construction of $\mathcal{P}$ and assume that the indices of the paths in $\mathcal{P}$ represent the order in which they are created. Assume that $i > 1$. By construction, the path $P_i \in \mathcal{P}$ connects the leaf C_i with the cluster C_i' in T_δ. Because $i > 1$, C_i' has a parent cluster in T_δ that is intersected by a path $P_j \in \mathcal{P}$ with $j < i$. We define P_j as the *parent* of P_i. By Lemma 7, this parent P_j is unique for each $P_i \in \mathcal{P}$ with $i > 1$. Based on this relation between paths in $\mathcal{P}$, we can construct a rooted tree $\mathbb{T}$ with the node set $\{x_i \mid P_i \in \mathcal{P}\}$ such that each node x_i represents the path P_i and x_j is the parent of x_i if and only if P_j is the parent of P_i.

Because each node of $\mathbb{T}$ represents a path in $\mathcal{P}$, f defines a colouring for the nodes of $\mathbb{T}$ such that x_i and x_j have different colours if and only if $f(P_i) \neq f(P_j)$. As long as $|\mathcal{V}| > 1$, $\mathbb{T}$ contains two adjacent nodes with different colours. Let x_i and x_j be these nodes with $j < i$ and let P_i and P_j be the corresponding paths in $\mathcal{P}$. Note that x_j is the parent of x_i in $\mathbb{T}$ and, hence, P_j is the parent of P_i. Therefore, P_i ends in a cluster C_i' which has a parent cluster C that intersects P_j. By properties of layering partitions, it follows that $d_G(P_i, P_j) \leq \Delta + 1$. Recall that, by construction, $d(v) = \min_{P \in \mathcal{P}} d_G(v, P)$ for each vertex v. Thus, for each edge uv on a shortest path from P_i to P_j in G (with u being closer to P_i than to P_j), $d(u) + d(v) \leq d_G(u, P_i) + d_G(v, P_j) \leq \Delta$. Therefore, because $f(P_i) \neq f(P_j)$, there is an edge uv on a shortest path from P_i to P_j such that $f\big(P(u)\big) \neq f\big(P(v)\big)$ and $d(u) + d(v) \leq \Delta$. □

From the claim above, it follows that, as long as $\mathcal{V}$ contains multiple sets, there is an edge $uv \in E'$ such that $d(u) + d(v) \leq \Delta$ and $\text{Find}(u) \neq \text{Find}(v)$. Therefore, $\max_{uv \in X} \big(d(u) + d(v)\big) \leq \Delta$ and $|S_\delta| \leq |T_\delta| + \big(\Lambda(T_\delta) - 1\big) \cdot \Delta$. □

Proof (Complexity). First, the algorithm computes $\mathcal{P}$ (line 2 to line 8). If the parent of each vertex from the original BFS that was used to construct $\mathcal{T}$ is still known, $\mathcal{P}$ can be constructed in $\mathcal{O}(n)$ total time. After picking a vertex v in C_i, simply follow the parent pointers until a vertex in C_i' is reached. Computing $\mathcal{V}$ as well as $P(v)$ and $d(v)$ for each vertex v of G (line 10) can be done with single BFS and, thus, requires at most $\mathcal{O}(n + m)$ time.

Recall that, for a Union-Find data structure storing n elements, each operation requires at most $\mathcal{O}\big(\alpha(n)\big)$ amortised time. Therefore, initialising such a data structure to store all vertices (line 11) and computing E' (line 12) requires at most $\mathcal{O}\big(m\,\alpha(n)\big)$ time. Note that, for each vertex v, $d(v) \leq |V|$. Thus, sorting E' (line 13) can be done in linear time using counting sort. When iterating over E' (line 14 to line 19), for each edge $uv \in E'$, the Find-operation is called twice and the Union-operation is called at most once. Thus, the total runtime for all these operations is at most $\mathcal{O}\big(m\,\alpha(n)\big)$.

Let $P_u = \{u, \ldots, x, y, \ldots, p\}$ be the shortest path in G from a vertex u to $P(u)$. Assume that y has been added to S_δ in a previous iteration. Thus, $\{y, \ldots, p\} \subseteq S_\delta$ and, when adding P_u to S_δ, the algorithm only needs to add $\{u, \ldots, x\}$. Therefore, by using a simple binary flag to determine if a vertex is contained in S_δ, constructing S_δ (line 9, line 17, and line 18) requires at most $\mathcal{O}(n)$ time.

In total, Algorithm 1 runs in $\mathcal{O}\big(m\,\alpha(n)\big)$ time. □

Because, for each integer $\delta \geq 0$, $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$ (Lemma 8) and $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$ (Lemma 6), we have the following.

Corollary 9. *For each $\delta \geq \Delta$, $|S_\delta| \leq |T_r|$ and, thus, $|S_\delta| \leq |D_r|$.*

To the best of our knowledge, there is no algorithm known that computes Δ in less than $\mathcal{O}(nm)$ time. Additionally, under reasonable assumptions, computing the diameter or radius of a general graph requires $\Omega(n^2)$ time [1]. We conjecture that the runtime for computing Δ for a given graph has a similar lower bound.

To avoid the runtime required for computing Δ, we use the following approach shown in Algorithm 2 below. First, compute a layering partition $\mathcal{T}$ and the subtree T_r. Second, for a certain value of δ, compute T_δ and perform Algorithm 1 on it. If the resulting set S_δ is larger than T_r (i.e., $|S_\delta| > |T_r|$), increase δ; otherwise, if $|S_\delta| \leq |T_r|$, decrease δ. Repeat the second step with the new value of δ.

One strategy to select values for δ is a classical binary search over the number of vertices of G. In this case, Algorithm 1 is called up-to $\mathcal{O}(\log n)$ times. Empirical analysis [2], however, have shown that Δ is usually very small. Therefore, we use a so-called *one-sided* binary search.

Consider a sorted sequence $\langle x_1, x_2, \ldots, x_n \rangle$ in which we search for a value x_p. We say the value x_i is at position i. For a one-sided binary search, instead of starting in the middle at position $n/2$, we start at position 1. We then processes position 2, then position 4, then position 8, and so on until we reach position $j = 2^i$ and, next, position $k = 2^{i+1}$ with $x_j < x_p \leq x_k$. Then, we perform a classical binary search on the sequence $\langle x_{j+1}, \ldots, x_k \rangle$. Note that, because $x_j < x_p \leq x_k$, $2^i < p \leq 2^{i+1}$ and, hence, $j < p \leq k < 2p$. Therefore, a one-sided binary search requires at most $\mathcal{O}(\log p)$ iterations to find x_p.

Because of Corollary 9, using a one-sided binary search allows us to find a value $\delta \leq \Delta$ for which $|S_\delta| \leq |T_r|$ by calling Algorithm 1 at most $\mathcal{O}(\log \Delta)$ times. Algorithm 2 below implements this approach.

Algorithm 2: Computes a connected $(r + 2\Delta)$-dominating set for a given graph G.

Input: A graph $G = (V, E)$ and a function $r\colon V \to \mathbb{N}$.
Output: A connected $(r + 2\Delta)$-dominating set D for G with $|D| \leq |D_r|$.

```
Create a layering partition 𝒯 of G.
For each cluster C of 𝒯, set r(C) := min_{v∈C} r(v).
Compute a minimum r-dominating subtree T_r for 𝒯 (see [16]).
One-Sided Binary Search over δ, starting with δ = 0
    Create a minimum δ-dominating subtree T_δ of T_r (i.e., T_δ is a minimum (r + δ)-dominating subtree for 𝒯).
    Run Algorithm 1 on T_δ and let the set S_δ be the corresponding output.
    if |S_δ| ≤ |T_r| then
        Decrease δ.
    else
        Increase δ.
Output S_δ with the smallest δ for which |S_δ| ≤ |T_r|.
```

Theorem 10. *For a given graph G, Algorithm 2 computes a connected $(r + 2\Delta)$-dominating set D with $|D| \leq |D_r|$ in $\mathcal{O}(m\,\alpha(n) \log \Delta)$ time.*

Proof. Clearly, the set D is connected because $D = S_\delta$ for some δ and, by Lemma 8, the set S_δ is connected. By Corollary 9, for each $\delta \geq \Delta$, $|S_\delta| \leq |T_r|$. Thus, for each $\delta \geq \Delta$, the binary search decreases δ and, eventually, finds some δ such that $\delta \leq \Delta$ and $|S_\delta| \leq |T_r|$. Therefore, the algorithm finds a set D with $|D| \leq |D_r|$. Note that, because $D = S_\delta$ for some $\delta \leq \Delta$ and because S_δ intersects each cluster of T_δ (Lemma 8), it follows from Lemma 4 that, for each vertex v of G, $d_{\mathcal{T}}(v, D) \leq r(v) + \Delta$ and, by Lemma 1, $d_G(v, D) \leq r(v) + 2\Delta$. Thus, D is an $(r + 2\Delta)$-dominating set for G.

Creating a layering partition for a given graph and computing a minimum connected r-dominating set of a tree can be done in linear time [16]. The one-sided binary search over δ has at most $\mathcal{O}(\log \Delta)$ iterations. Each iteration of the binary search requires at most linear time to compute T_δ, $\mathcal{O}(m\,\alpha(n))$ time to compute S_δ (Lemma 8), and constant time to decide whether to increase or decrease δ. Therefore, Algorithm 2 runs in $\mathcal{O}(m\,\alpha(n) \log \Delta)$ total time. □

4. Using a tree-decomposition

Theorem 2 and Theorem 10 respectively show how to compute an $(r + \Delta)$-dominating set in linear time and a connected $(r + 2\Delta)$-dominating set in $\mathcal{O}(m\,\alpha(n) \log \Delta)$ time. It is known that the maximum diameter Δ of clusters of any layering partition of a graph approximates the tree-breadth and tree-length of this graph. Indeed, for a graph G with $\mathrm{tl}(G) = \lambda$, $\Delta \leq 3\lambda$ [14].

Corollary 11. *Let D be a minimum r-dominating set for a given graph G with $\mathrm{tl}(G) = \lambda$. An $(r + 3\lambda)$-dominating set D' for G with $|D'| \leq |D|$ can be computed in linear time.*

Corollary 12. *Let D be a minimum connected r-dominating set for a given graph G with $\mathrm{tl}(G) = \lambda$. A connected $(r + 6\lambda)$-dominating set D' for G with $|D'| \leq |D|$ can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\lambda\big)$ time.*

In this section, we consider the case when we are given a tree-decomposition with breadth ρ and length λ. We present algorithms to compute an $(r + \rho)$-dominating set as well as a connected $\big(r + \min(3\lambda, 5\rho)\big)$-dominating set in $\mathcal{O}(nm)$ time.

For the remainder of this section, assume that we are given a graph $G = (V, E)$ and a tree-decomposition $\mathcal{T}$ of G with breadth ρ and length λ. We assume that ρ and λ are known and that, for each bag B of $\mathcal{T}$, we know a vertex $c(B)$ with $B \subseteq N_G^{\rho}[c(B)]$. Let $\mathcal{T}$ be minimal, i.e., $B \nsubseteq B'$ for any two bags B and B'. Thus, the number of bags is not exceeding the number vertices of G. Additionally, let each vertex of G store a list of bags containing it and let each bag of T store a list of vertices it contains. One can see this as a bipartite graph where one subset of vertices are the vertices of G and the other subset are the bags of $\mathcal{T}$. Therefore, the total input size is in $\mathcal{O}(n + m + M)$ where $M \leq n^2$ is the sum of the cardinality of all bags of $\mathcal{T}$.

4.1. Preprocessing

Before approaching the (Connected) r-Domination problem, we compute a subtree $\mathcal{T}'$ of $\mathcal{T}$ such that, for each vertex v of G, $\mathcal{T}'$ contains a bag B with $d_G(v, B) \leq r(v)$. We call such a (not necessarily minimal) subtree an *r-covering subtree of $\mathcal{T}$*.

Let T_r be a minimum r-covering subtree of $\mathcal{T}$. We do not know how to compute T_r directly. However, if we are given a bag B of $\mathcal{T}$, we can compute the smallest r-covering subtree T_B which contains B. Then, we can identify a bag B' in T_B for which we know it is a bag of T_r. Thus, we can compute T_r by computing the smallest r-covering subtree which contains B'.

The idea for computing T_B is to determine, for each vertex v of G, the bag B_v of $\mathcal{T}$ for which $d_G(v, B_v) \leq r(v)$ and which is closet to B. Then, let T_B be the smallest tree that contains all these bags B_v. Algorithm 3 below implements this approach.

Additionally to computing the tree T_B, we make it a rooted tree with B as the root, give each vertex v a pointer $\beta(v)$ to a bag of T_B, and give each bag B' a counter $\sigma(B')$. The pointer $\beta(v)$ identifies the bag B_v which is closest to B in T_B and intersects the r-neighbourhood of v. The counter $\sigma(B')$ states the number of vertices v with $\beta(v) = B'$. Even though setting β and σ as well as rooting the tree are not necessary for computing T_B, we use it when computing an $(r + \rho)$-dominating set later.

Algorithm 3: Computes the smallest r-covering subtree T_B of a given tree-decomposition $\mathcal{T}$ that contains a given bag B of $\mathcal{T}$.

```
1 Make T a rooted tree with the bag B as the root.
2 Create a set 𝓑 of bags and initialise it with 𝓑 := {B}.
3 For each bag B' of T, set σ(B') := 0 and determine d_T(B', B).
4 For each vertex u, determine the bag B(u) which contains u and has minimal distance to B.
5 foreach u ∈ V do
6     Determine a vertex v such that d_G(u, v) ≤ r(u) and d_T(B(v), B) is minimal and let B_u := B(v).
7     Add B_u to 𝓑, set β(u) := B_u, and increase σ(B_u) by 1.
8 Output the smallest subtree T_B of T that contains all bags in 𝓑.
```

Lemma 13. *For a given tree-decomposition $\mathcal{T}$ and a given bag B of $\mathcal{T}$, Algorithm 3 computes an r-covering subtree T_B in $\mathcal{O}(nm)$ time such that T_B contains B and has a minimal number of bags.*

Proof (Correctness). Note that, by construction of the set $\mathcal{B}$ (line 5 to line 7), $\mathcal{B}$ contains a bag B_u for each vertex u of G such that $d_G(u, B_u) \leq r(u)$. Thus, each subtree of $\mathcal{T}$ which contains all bags of $\mathcal{B}$ is an r-covering subtree. To show the correctness of the algorithm, it remains to show that the smallest r-covering subtree of $\mathcal{T}$ which contains B has to contain each bag from the set $\mathcal{B}$. Then, the subtree T_B constructed in line 8 is the desired subtree.

By properties of tree-decompositions, the set of bags which intersect the r-neighbourhood of some vertex u induces a subtree T_u of $\mathcal{T}$. That is, T_u contains exactly the bags B' with $d_G(u, B') \leq r(u)$. Note that $\mathcal{T}$ is a rooted tree with B as the root. Clearly, the bag $B_u \in \mathcal{B}$ (determined in line 6) is the root of T_u since it is the bag closest to B. Hence, each bag B' with $d_G(u, B') \leq r(u)$ is a descendant of B_u. Therefore, if a subtree of $\mathcal{T}$ contains B and does not contain B_u, then it also cannot contain any descendant of B_u and, thus, contains no bag intersecting the r-neighbourhood of u. □

Proof (Complexity). Recall that $\mathcal{T}$ has at most n bags and that the sum of the cardinality of all bags of $\mathcal{T}$ is $M \leq n^2$. Thus, line 3 and line 4 require at most $\mathcal{O}(M)$ time. Using a BFS, it takes at most $\mathcal{O}(m)$ time, for a given vertex u, to determine a vertex v such that $d_G(u, v) \leq r(u)$ and $d_{\mathcal{T}}\big(B(v), B\big)$ is minimal (line 6). Therefore, the loop starting in line 5 and, thus, Algorithm 3 run in at most $\mathcal{O}(nm)$ total time. □

Lemma 14 and Lemma 15 below show that each leaf $B' \neq B$ of T_B is a bag of a minimum r-covering subtree T_r of $\mathcal{T}$. Note that both lemmas only apply if T_B has at least two bags. If T_B contains only one bag, it is clearly a minimum r-covering subtree.

Lemma 14. *For each leaf $B' \neq B$ of T_B, there is a vertex v in G such that B' is the only bag of T_B with $d_G(v, B') \leq r(v)$.*

Proof. Assume that Lemma 14 is false. Then, there is a leaf B' such that, for each vertex v with $d_G(v, B') \leq r(v)$, T_B contains a bag $B'' \neq B'$ with $d_G(v, B'') \leq r(v)$. Thus, for each vertex v, the r-neighbourhood of v is intersected by a bag of the tree-decomposition $T_B - B'$. This contradicts with the minimality of T_B. □

Lemma 15. *For each leaf $B' \neq B$ of T_B, there is a minimum r-covering subtree T_r of $\mathcal{T}$ which contains B'.*

Proof. Assume that T_r is a minimum r-covering subtree which does not contain B'. Because of Lemma 14, there is a vertex v of G such that B' is the only bag of T_B which intersects the r-neighbourhood of v. Therefore, T_r contains only bags which are descendants of B'. Partition the vertices of G into the sets $V^{\uparrow}$ and $V^{\downarrow}$ such that $V^{\downarrow}$ contains the vertices of G which are contained in B' or in a descendant of B'. Because T_r is an r-covering subtree and because T_r only contains descendants of B', it follows from properties of tree-decompositions that, for each vertex $v \in V^{\uparrow}$, there is a path of length at most $r(v)$ from v to a bag of T_r passing through B' and, thus, $d_G(v, B') \leq r(v)$. Similarly, since T_B is an r-covering subtree, it follows that, for each vertex $v \in V^{\downarrow}$, $d_G(v, B') \leq r(v)$. Therefore, for each vertex v of G, $d_G(v, B') \leq r(v)$ and, thus, B' induces an r-covering subtree T_r of $\mathcal{T}$ with $|T_r| = 1$. □

Algorithm 4 below uses Lemma 15 to compute a minimum r-covering subtree T_r of $\mathcal{T}$.

Algorithm 4: Computes a minimum r-covering subtree T_r of a given tree-decomposition $\mathcal{T}$.

```
1 Pick an arbitrary bag B of T.
2 Determine the subtree T_B of T using Algorithm 3.
3 if |T_B| = 1 then
4     Output T_r := T_B.
5 else
6     Select an arbitrary leaf B' ≠ B of T_B.
7     Determine the subtree T_{B'} of T using Algorithm 3.
8     Output T_r := T_{B'}.
```

Lemma 16. *Algorithm 4 computes a minimum r-covering subtree T_r of $\mathcal{T}$ in $\mathcal{O}(nm)$ time.*

Proof. Algorithm 4 first picks an arbitrary bag B and then uses Algorithm 3 to compute the smallest r-covering subtree T_B of $\mathcal{T}$ which contains B. By Lemma 15, for each leaf B' of T_B, there is a minimum r-covering subtree T_r which contains B'. Thus, performing Algorithm 3 again with B' as input creates such a subtree T_r.

Clearly, with exception of calling Algorithm 3, all steps of Algorithm 4 require only constant time. Because Algorithm 3 requires at most $\mathcal{O}(nm)$ time (see Lemma 13) and is called at most two times, Algorithm 4 runs in at most $\mathcal{O}(nm)$ total time. □

Algorithm 4 computes T_r by, first, computing T_B for some bag B and, second, computing $T_{B'} = T_r$ for some leaf B' of T_B. Note that, because both trees are computed using Algorithm 3, Lemma 14 applies to T_B and $T_{B'}$. Therefore, we can slightly generalise Lemma 14 as follows.

Corollary 17. *For each leaf B of T_r, there is a vertex v in G such that B is the only bag of T_r with $d_G(v, B) \leq r(v)$.*

4.2. r-Domination

In this subsection, we use the minimum r-covering subtree T_r to determine an $(r + \rho)$-dominating set S in $\mathcal{O}(nm)$ time using the following approach. First, compute T_r. Second, pick a leaf B of T_r. If there is a vertex v such that v is not dominated and B is the only bag intersecting the r-neighbourhood of v, then add the center of B into S, flag all vertices u with $d_G(u, B) \leq r(u)$ as dominated, and remove B from T_r. Repeat the second step until T_r contains no more bags and each vertex is flagged as dominated. Algorithm 5 below implements this approach. Note that, instead of removing bags from T_r, we use a reversed BFS-order of the bags to ensure the algorithm processes bags in the correct order.

Theorem 18. *Let D be a minimum r-dominating set for a given graph G. Given a tree-decomposition with breadth ρ for G, Algorithm 5 computes an $(r + \rho)$-dominating set S with $|S| \leq |D|$ in $\mathcal{O}(nm)$ time.*

Algorithm 5: Computes an $(r+\rho)$-dominating set S for a given graph G with a given tree-decomposition $\mathcal{T}$ with breadth ρ.

```
1  Compute a minimum r-covering subtree T_r of T using Algorithm 4.
2  Give each vertex v a binary flag indicating if v is dominated. Initially, no vertex is dominated.
3  Create an empty vertex set S_0.
4  Let ⟨B_1, B_2, ..., B_k⟩ be the reverse of a BFS-order of T_r starting at its root.
5  for i = 1 to k do
6      if σ(B_i) > 0 then
7          Determine all vertices u such that u has not been flagged as dominated and that d_G(u, B_i) ≤ r(u). Add all these vertices into a new set X_i.
8          Let S_i = S_{i-1} ∪ {c(B_i)}.
9          For each vertex u ∈ X_i, flag u as dominated, and decrease σ(β(u)) by 1.
10     else
11         Let S_i = S_{i-1}.
12 Output S := S_k.
```

Proof (Correctness). First, we show that S is an $(r+\rho)$-dominating set for G. Note that a vertex v is flagged as dominated only if S_i contains a vertex $c(B_j)$ with $d_G(v, B_j) \leq r(v)$ (see line 7 to line 9). Thus, v is flagged as dominated only if $d_G(v, S_i) \leq d_G\big(v, c(B_j)\big) \leq r(v) + \rho$. Additionally, by construction of T_r (see Algorithm 3), for each vertex v, T_r contains a bag B with $\beta(v) = B$, $\sigma(B)$ states the number of vertices v with $\beta(v) = B$, and $\sigma(B)$ is decreased by 1 only if such a vertex v is flagged as dominated (see line 9). Therefore, if G contains a vertex v with $d_G(v, S_i) > r(v) + \rho$, then v is not flagged as dominated and T_r contains a bag B_i with $\beta(v) = B_i$ and $\sigma(B_i) > 0$. Thus, when B_i is processed by the algorithm, $c(B_i)$ will be added to S_i and, hence, $d_G(v, S_i) \leq r(v) + \rho$.

Let $V_i^S = \{\, u \mid d_G(u, B_j) \leq r(u), c(B_j) \in S_i \,\}$ be the set of vertices which are flagged as dominated after the algorithm processed B_i, i.e., each vertex in V_i^S is $(r+\rho)$-dominated by S_i. Similarly, for some set $D_i \subseteq D$, let $V_i^D = \{\, u \mid d_G(u, D_i) \leq r(u) \,\}$ be the set of vertices dominated by D_i. To show that $|S| \leq |D|$, we show by induction over i that, for each i, (i) there is a set $D_i \subseteq D$ such that $V_i^D \subseteq V_i^S$, (ii) $|S_i| = |D_i|$, and (iii) if, for some vertex v, $\beta(v) = B_j$ with $j \leq i$, then $v \in V_i^S$.

For the base case, let $S_0 = D_0 = \emptyset$. Then, $V_0^S = V_0^D = \emptyset$ and all three statements are satisfied. For the inductive step, first, consider the case when $\sigma(B_i) = 0$. Because $\sigma(B_i) = 0$, each vertex v with $\beta(v) = B_i$ is flagged as dominated, i.e., $v \in V_{i-1}^S$. Thus, by setting $S_i = S_{i-1}$ (line 11) and $D_i = D_{i-1}$, all three statements are satisfied for i. Next, consider the case when $\sigma(B_i) > 0$. Therefore, G contains a vertex u with $\beta(u) = B_i$ and $u \notin V_{i-1}^S$. Then, the algorithm sets $S_i = S_{i-1} \cup \{c(B_i)\}$ and flags all such u as dominated (see line 7 to line 9). Thus, $u \in V_i^S$ and statement (iii) is satisfied. Let d_u be a vertex in D with minimal distance to u. Thus, $d_G(d_u, u) \leq r(u)$, i.e., d_u is in the r-neighbourhood of u. Note that, because $u \notin V_{i-1}^S$ and $V_{i-1}^D \subseteq V_{i-1}^S$, $d_u \notin D_{i-1}$. Therefore, by setting $D_i = D_{i-1} \cup \{d_u\}$, $|S_i| = |S_{i-1}| + 1 = |D_{i-1}| + 1 = |D_i|$ and statement (ii) is satisfied. Recall that $\beta(u)$ points to the bag closest to the root of T_r which intersects the r-neighbourhood of u. Thus, because $\beta(u) = B_i$, each bag $B \neq B_i$ with $d_G(u, B) \leq r(u)$ is a descendant of B_i. Therefore, d_u is in B_i or in a descendant of B_i. Let v be an arbitrary vertex of G such that $v \notin V_{i-1}^S$ and $d_G(v, d_u) \leq r(v)$, i.e., v is dominated by d_u but not by S_{i-1}. Due to statement (iii) of the induction hypothesis, $\beta(v) = B_j$ with $j \geq i$, i.e., B_j cannot be a descendant of B_i. Partition the vertices of G into the sets $V_i^{\uparrow}$ and $V_i^{\downarrow}$ such that $V_i^{\downarrow}$ contains the vertices which are contained in B_i or in a descendant of B_i. If $v \in V_i^{\downarrow}$, then there is a path of length at most $r(v)$ from v to B_j passing through B_i. If $v \in V_i^{\uparrow}$, then, because $d_u \in V_i^{\downarrow}$, there is a path of length at most $r(v)$ from v to d_u passing through B_i. Therefore, $d_G(v, B_i) \leq r(v)$. That is, each vertex r-dominated by d_u, is $(r+\rho)$-dominated by some $c(B_j) \in S_i$. Therefore, because $S_i = S_{i-1} \cup \{c(B_i)\}$ and $D_i = D_{i-1} \cup \{d_u\}$, $v \in V_i^S \cap V_i^D$ and, thus, statement (i) is satisfied. □

Proof (Complexity). Computing T_r (line 1) takes at most $\mathcal{O}(nm)$ time (see Lemma 16). Because T_r has at most n bags, computing a BFS-order of T_r (line 4) takes at most $\mathcal{O}(n)$ time. For some bag B_i, determining all vertices u with $d_G(u, B_i) \leq r(u)$, flagging u as dominated, and decreasing $\sigma\big(\beta(u)\big)$ (line 7 to line 9) can be done in $\mathcal{O}(m)$ time by performing a BFS starting at all vertices of B_i simultaneously. Therefore, because T_r has at most n bags, Algorithm 5 requires at most $\mathcal{O}(nm)$ total time. □

4.3. Connected r-domination

In this subsection, we show how to compute a connected $(r+5\rho)$-dominating set and a connected $(r+3\lambda)$-dominating set for G. For both results, we use almost the same algorithm. To identify and emphasise the differences, we use the label (♡) for parts which are only relevant to determine a connected $(r+5\rho)$-dominating set and use the label (◇) for parts which are only relevant to determine a connected $(r+3\lambda)$-dominating set.

For the remainder of this subsection, let D_r be a minimum connected r-dominating set of G. For (♡) $\phi = 3\rho$ or (◇) $\phi = 2\lambda$, let T_ϕ be a minimum $(r+\phi)$-covering subtree of $\mathcal{T}$ as computed by Algorithm 4.

The idea of our algorithm is to, first, compute T_ϕ and, second, compute a small enough connected set C_ϕ such that C_ϕ intersects each bag of T_ϕ. Lemma 19 below shows that such a set C_ϕ is an $\big(r+(\phi+\lambda)\big)$-dominating set.

Lemma 19. *Let C_ϕ be a connected set that contains at least one vertex of each leaf of T_ϕ. Then, C_ϕ is an $\big(r+(\phi+\lambda)\big)$-dominating set.*

Proof. Clearly, since C_ϕ is connected and contains a vertex of each leaf of T_ϕ, C_ϕ contains a vertex of every bag of T_ϕ. By construction of T_ϕ, for each vertex v of G, T_ϕ contains a bag B such that $d_G(v,B) \leq r(v)+\phi$. Therefore, $d_G(v,C_\phi) \leq r(v)+\phi+\lambda$, i.e., C_ϕ is an $\big(r+(\phi+\lambda)\big)$-dominating set. □

To compute a connected set C_ϕ which intersects all leaves of T_ϕ, we first consider the case when T_ρ contains only one bag B. In this case, we can construct C_ϕ by simply picking an arbitrary vertex $v \in B$ and setting $C_\phi = \{v\}$. Similarly, if T_ρ contains exactly two bags B and B', pick a vertex $v \in B \cap B'$ and set $C_\phi = \{v\}$. In both cases, due to Lemma 19, C_ϕ is clearly an $\big(r+(\phi+\lambda)\big)$-dominating set with $|C_\phi| \leq |D_r|$.

Now, consider the case when T_ϕ contains at least three bags. Additionally, assume that T_ϕ is a rooted tree such that its root R is a leaf.

4.3.1. Notation

Based on its degree in T_ϕ, we refer to each bag B of T_ϕ either as leaf, as *path bag* if B has degree 2, or as *branching bag* if B has a degree larger than 2. Additionally, we call a maximal connected set of path bags a *path segment* of T_ϕ. Let $\mathbb{L}$ denote the set of leaves, $\mathbb{P}$ denote the set of path segments, and $\mathbb{B}$ denote the set of branching bags of T_ϕ. Clearly, for any given tree T, the sets $\mathbb{L}$, $\mathbb{P}$, and $\mathbb{B}$ are pairwise disjoint and can be computed in linear time.

Let B and B' be two adjacent bags of T_ϕ such that B is the parent of B'. We call $S = B \cap B'$ the *up-separator* of B', denoted as $S^\uparrow(B')$, and a *down-separator* of B, denoted as $S^\downarrow(B)$, i.e., $S = S^\uparrow(B') = S^\downarrow(B)$. Note that a branching bag has multiple down-separators and that (with exception of R) each bag has exactly one up-separator. For each branching bag B, let $\mathcal{S}^\downarrow(B)$ be the set of down-separators of B. Accordingly, for a path segment $P \in \mathbb{P}$, $S^\uparrow(P)$ is the up-separator of the bag in P closest to the root and $S^\downarrow(P)$ is the down separator of the bag in P furthest from the root. Let ν be a function that assigns a vertex of G to a given separator. Initially, $\nu(S)$ is undefined for each separator S.

4.3.2. Algorithm

Now, we show how to compute C_ϕ. We, first, split T_ϕ into the sets $\mathbb{L}$, $\mathbb{P}$, and $\mathbb{B}$. Second, for each $P \in \mathbb{P}$, we create a small connected set C_P, and, third, for each $B \in \mathbb{B}$, we create a small connected set C_B. If this is done properly, the union C_ϕ of all these sets forms a connected set which intersects each bag of T_ϕ.

Note that, due to properties of tree-decompositions, it can be the case that there are two bags B and B' which have a common vertex v, even if B and B' are non-adjacent in T_ϕ. In such a case, either $v \in S^\downarrow(B) \cap S^\uparrow(B')$ if B is an ancestor of B', or $v \in S^\uparrow(B) \cap S^\uparrow(B')$ if neither is ancestor of the other. To avoid problems caused by this phenomenon and to avoid counting vertices multiple times, we consider any vertex in an up-separator as part of the bag above. That is, whenever we process some segment or bag $X \in \mathbb{L} \cup \mathbb{P} \cup \mathbb{B}$, even though we add a vertex $v \in S^\uparrow(X)$ to C_ϕ, v is not contained in C_X.

Processing Path Segments. First, after splitting T_ϕ, we create a set C_P for each path segment $P \in \mathbb{P}$ as follows. We determine $S^\uparrow(P)$ and $S^\downarrow(P)$ and then find a shortest path Q_P from $S^\uparrow(P)$ to $S^\downarrow(P)$. Note that Q_P contains exactly one vertex from each separator. Let $x \in S^\uparrow(P)$ and $y \in S^\downarrow(P)$ be these vertices. Then, we set $\nu\big(S^\uparrow(P)\big) = x$ and $\nu\big(S^\downarrow(P)\big) = y$. Last, we add the vertices of Q_P into C_ϕ and define C_P as $Q_P \setminus S^\uparrow(P)$. Let $C_\mathbb{P}$ be the union of all sets C_P, i.e., $C_\mathbb{P} = \bigcup_{P\in\mathbb{P}} C_P$.

Lemma 20. $|C_\mathbb{P}| \leq |D_r| - \phi \cdot \Lambda\big(T_\phi\big)$.

Proof. Recall that T_ϕ is a minimum $(r+\phi)$-covering subtree of $\mathcal{T}$. Thus, by Corollary 17, for each leaf $B \in \mathbb{L}$ of T_ϕ, there is a vertex v in G such that B is the only bag of T_ϕ with $d_G(v,B) \leq r(v)+\phi$. Because D_r is a connected r-dominating set, D_r intersects the r-neighbourhood of each of these vertices v. Thus, by properties of tree-decompositions, D_r intersects each bag of T_ϕ. Additionally, for each such v, D_r contains a path D_v with $|D_v| \geq \phi$ such that D_v intersects the r-neighbourhood of v, intersects the corresponding leaf B of T_ϕ, and does not intersect $S^\uparrow(B)$ ($S^\downarrow(B)$ if $B = R$). Let $D_\mathbb{L}$ be the union of all such sets D_v. Therefore, $|D_\mathbb{L}| \geq \phi \cdot \Lambda\big(T_\phi\big)$.

Because D_r intersects each bag of T_ϕ, D_r also intersects the up- and down-separators of each path segment. For a path segment $P \in \mathbb{P}$, let x and y be two vertices of D_r such that $x \in S^\uparrow(P)$, $y \in S^\downarrow(P)$, and for which the distance in $G[D_r]$ is minimal. Let D_P be the set of vertices on the shortest path in $G[D_r]$ from x to y without x, i.e., $x \notin D_P$. Note that, by construction, for each $P \in \mathbb{P}$, D_P contains exactly one vertex in $S^\downarrow(P)$ and no vertex in $S^\uparrow(P)$. Thus, for all $P, P' \in \mathbb{P}$, $D_P \cap D_{P'} = \emptyset$. Let $D_\mathbb{P}$ be the union of all such sets D_P, i.e., $D_\mathbb{P} = \bigcup_{P\in\mathbb{P}} D_P$. By construction, $|D_\mathbb{P}| = \sum_{P\in\mathbb{P}} |D_P|$ and $D_\mathbb{L} \cap D_\mathbb{P} = \emptyset$. Therefore, $|D_r| \geq |D_\mathbb{P}| + |D_\mathbb{L}|$ and, hence,

$$\sum_{P\in\mathbb{P}} |D_P| \leq |D_r| - |D_\mathbb{L}| \leq |D_r| - \phi \cdot \Lambda\big(T_\phi\big).$$

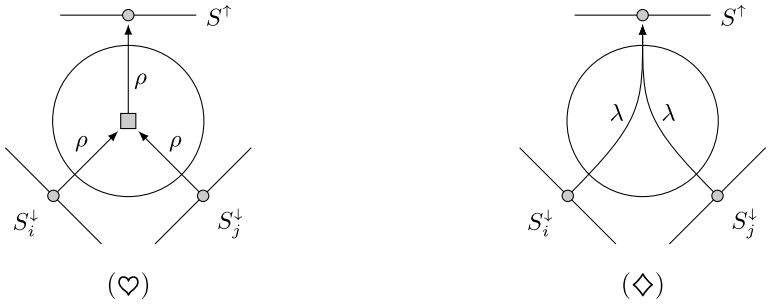


Fig. 3. Construction of the set C_B for a branching bag B.

Recall that, for each $P \in \mathbb{P}$, the sets C_P and D_P are constructed based on a path from $S^{\uparrow}(P)$ to $S^{\downarrow}(P)$. Since C_P is based on a shortest path in G, it follows that $|C_P| = d_G\big(S^{\uparrow}(P), S^{\downarrow}(P)\big) \leq |D_P|$. Therefore,

$$|C_{\mathbb{P}}| \leq \sum_{P \in \mathbb{P}} |C_P| \leq \sum_{P \in \mathbb{P}} |D_P| \leq |D_r| - \phi \cdot \Lambda\big(T_\phi\big). \quad \square$$

Processing Branching Bags. After processing path segments, we process the branching bags of T_ϕ. Similar to path segments, we have to ensure that all separators are connected. Branching bags, however, have multiple down-separators. To connect all separators of some bag B, we pick a vertex s in each separator $S \in \mathcal{S}^{\downarrow}(B) \cup \big\{S^{\uparrow}(B)\big\}$. If $\nu(S)$ is defined, we set $s = \nu(S)$. Otherwise, we pick an arbitrary $s \in S$ and set $\nu(S) = s$. Let $\mathcal{S}^{\downarrow}(B) = \{S_1, S_2, \ldots\}$, $s_i = \nu(S_i)$, and $t = \nu\big(S^{\uparrow}(B)\big)$. We then connect these vertices as follows. (See Fig. 3 for an illustration.)

($\heartsuit$) Connect each vertex s_i via a shortest path Q_i (of length at most ρ) with the center $c(B)$ of B. Additionally, connect $c(B)$ via a shortest path Q_t (of length at most ρ) with t. Add all vertices from the paths Q_i and from the path Q_t into C_ϕ and let C_B be the union of these paths without t.

($\diamondsuit$) Connect each vertex s_i via a shortest path Q_i (of length at most λ) with t. Add all vertices from the paths Q_i into C_ϕ and let C_B be the union of these paths without t.

Let $C_{\mathbb{B}}$ be the union of all created sets C_B, i.e., $C_{\mathbb{B}} = \bigcup_{B \in \mathbb{B}} C_B$.

Before analysing the cardinality of $C_{\mathbb{B}}$ in Lemma 22 below, we need an axillary lemma.

Lemma 21. *For a tree T which is rooted in one of its leaves, let b denote the number of branching nodes, c denote the total number of children of branching nodes, and l denote the number of leaves. Then, $c + b \leq 3l - 1$ and $c \leq 2l - 1$.*

Proof. Assume that we construct T by starting with only the root and then step by step adding leaves to it. Let T_i be the subtree of T with i nodes during this construction. We define b_i, c_i, and l_i accordingly. Now, assume by induction over i that Lemma 21 is true for T_i. Let v be the leaf we add to construct T_{i+1} and let u be its neighbour.

First, consider the case when u is a leaf of T_i. Then, u becomes a path node of T_{i+1}. Therefore, $b_{i+1} = b_i$, $c_{i+1} = c_i$, and $l_{i+1} = l_i$. Next, assume that u is path node of T_i. Then, u is a branch node of T_{i+1}. Thus, $b_{i+1} = b_i + 1$, $c_{i+1} = c_i + 2$, and $l_{i+1} = l_i + 1$. Therefore, $c_{i+1} + b_{i+1} = c_i + b_i + 3 \leq 3(l_i + 1) - 1 = 3l_{i+1} - 1$ and $c_{i+1} = c_i + 2 \leq 2(l_i + 1) - 1 = 2l_{i+1} - 1$. It remains to check the case when u is a branch node of T_i. Then, $b_{i+1} = b_i$, $c_{i+1} = c_i + 1$, and $l_{i+1} = l_i + 1$. Thus, $c_{i+1} + b_{i+1} = c_i + b_i + 1 \leq 3l_i - 1 + 1 \leq 3l_{i+1} - 1$ and $c_{i+1} = c_i + 1 \leq 2l_i - 1 + 1 \leq 2l_{i+1} - 1$. Therefore, in all three cases, Lemma 21 is true for T_{i+1}. $\square$

Lemma 22. $|C_{\mathbb{B}}| \leq \phi \cdot \Lambda\big(T_\phi\big)$.

Proof. For some branching bag $B \in \mathbb{B}$, the set C_B contains ($\heartsuit$) a path of length at most ρ for each $S_i \in \mathcal{S}^{\downarrow}(B)$ and a path of length at most ρ to $S^{\uparrow}(B)$, or ($\diamondsuit$) a path of length at most λ for each $S_i \in \mathcal{S}^{\downarrow}(B)$. Thus, ($\heartsuit$) $|C_B| \leq \rho \cdot \big|\mathcal{S}^{\downarrow}(B)\big| + \rho$ or ($\diamondsuit$) $|C_B| \leq \lambda \cdot \big|\mathcal{S}^{\downarrow}(B)\big|$. Recall that $\mathcal{S}^{\downarrow}(B)$ contains exactly one down-separator for each child of B in T_ϕ and that $C_{\mathbb{B}}$ is the union of all sets C_B. Therefore, Lemma 21 implies the following.

$$\begin{aligned} |C_{\mathbb{B}}| &\leq \sum_{B \in \mathbb{B}} |C_B| \\ (\heartsuit) \quad &\leq \rho \cdot \sum_{B \in \mathbb{B}} \big|\mathcal{S}^{\downarrow}(B)\big| + \rho \cdot |\mathbb{B}| \leq 3\rho \cdot \Lambda\big(T_\phi\big) - 1 \end{aligned}$$

$$(\diamondsuit) \quad \leq \lambda \cdot \sum_{B \in \mathbb{B}} \left|\mathcal{S}^{\downarrow}(B)\right| \leq 2\lambda \cdot \Lambda\left(T_\phi\right) - 1$$
$$\leq \phi \cdot \Lambda\left(T_\phi\right) - 1. \quad \square$$

Properties of C_ϕ. We now analyse the created set C_ϕ and show that C_ϕ is a connected $(r+\phi)$-dominating set for G.

Lemma 23. *C_ϕ contains a vertex in each bag of T_ϕ.*

Proof. Clearly, by construction, C_ϕ contains a vertex in each path bag and in each branching bag. Now, consider a leaf L of T_ϕ. L is adjacent to a path segment or branching bag $X \in \mathbb{P} \cap \mathbb{B}$. Whenever such an X is processed, the algorithm ensures that all separators of X contain a vertex of C_ϕ. Since one of these separators is also the separator of L, it follows that each leaf L and, thus, each bag of T_ϕ contains a vertex of C_ϕ. $\square$

Lemma 24. $|C_\phi| \leq |D_r|$.

Proof. Note that, for each vertex u we add to C_ϕ, we also add u to a unique set C_X for some $X \in \mathbb{P} \cap \mathbb{B}$. The exception is the vertex v in $S^{\downarrow}(R)$ which is added to no such set C_X. It follows from our construction of the sets C_X that there is only one such vertex v and that $v = \nu\left(S^{\downarrow}(R)\right)$. Thus, $|C_\phi| = |C_{\mathbb{P}}| + |C_{\mathbb{B}}| + 1$. Now, it follows from Lemma 20 and Lemma 22 that

$$|C_\phi| \leq |D_r| - \phi \cdot \Lambda\left(T_\phi\right) + \phi \cdot \Lambda\left(T_\phi\right) - 1 + 1 \leq |D_r|. \quad \square$$

Lemma 25. *C_ϕ is connected.*

Proof. First, note that, by maximality, two path segments of T_ϕ cannot share a common separator. Also, note that, when processing a branching bag B, the algorithm first checks if, for any separator S of B, $\nu(S)$ is already defined; if this is the case, it will not be overwritten. Therefore, for each separator S in T_ϕ, $\nu(S)$ is defined and never overwritten.

Next, consider a path segment or branching bag $X \in \mathbb{P} \cup \mathbb{B}$ and let S and S' be two separators of X. Whenever such an X is processed, our approach ensures that C_ϕ connects $\nu(S)$ with $\nu(S')$. Additionally, observe that, when processing X, each vertex added to C_ϕ is connected via C_ϕ with $\nu(S)$ for some separator S of X.

Thus, for any two separators S and S' in T_ϕ, C_ϕ connects $\nu(S)$ with $\nu(S')$ and, additionally, each vertex $v \in C_\phi$ is connected via C_ϕ with $\nu(S)$ for some separator S in T_ϕ. Therefore, C_ϕ is connected. $\square$

From Lemma 23, Lemma 24, Lemma 25, and from applying Lemma 19 it follows:

Corollary 26. *C_ϕ is a connected $\left(r + (\phi + \lambda)\right)$-dominating set for G with $|C_\phi| \leq |D_r|$.*

Implementation. Algorithm 6 below implements our approach described above. This also includes the case when T_ϕ contains at most two bags.

Theorem 27. *Algorithm 6 computes a connected $\left(r + (\phi + \lambda)\right)$-dominating set C_ϕ with $|C_\phi| \leq |D_r|$ in $\mathcal{O}(nm)$ time.*

Proof. Since Algorithm 6 constructs a set C_ϕ as described above, its correctness follows from Corollary 26. It remains to show that the algorithm runs in $\mathcal{O}(nm)$ time.

Computing T_ϕ (line 2) can be done in $\mathcal{O}(nm)$ time (see Lemma 16). Picking a vertex u in the case when T_ϕ contains at most two bags (line 3 to line 6) can be easily done in $\mathcal{O}(n)$ time. Recall that T_ϕ has at most n bags. Thus, splitting T_ϕ in the sets $\mathbb{L}$, $\mathbb{P}$, and $\mathbb{B}$ can be done in $\mathcal{O}(n)$ time.

Determining all up-separators in T_ϕ can be done in $\mathcal{O}(M)$ time as follows. Process all bags of T_ϕ in an order such that a bag is processed before its descendants, e.g., use a preorder or BFS-order. Whenever a bag B is processed, determine a set $S \subseteq B$ of flagged vertices, store S as up-separator of B, and, afterwards, flag all vertices in B. Clearly, S is empty for the root. Because a bag B is processed before its descendants, all flagged vertices in B also belong to its parent. Thus, by properties of tree-decompositions, these vertices are exactly the vertices in $S^{\uparrow}(B)$. Clearly, processing a single bag B takes at most $\mathcal{O}(|B|)$ time. Thus, processing all bags takes at most $\mathcal{O}(M)$ time. Note that it is not necessary to determine the down-separators of a (branching) bag. They can easily be accessed via the children of a bag.

Processing a single path segment (line 11 and line 12) can be easily done in $\mathcal{O}(m)$ time. Processing a branching bag B (line 13 to line 19) can be implemented to run in $\mathcal{O}(m)$ time by, first, determining $\nu(S)$ for each separator S of B and, second, running a BFS starting at v (defined in line 15) to connect v with each vertex $\nu(S)$. Because T_ϕ has at most n bags, it takes at most $\mathcal{O}(nm)$ time to process all path segments and branching bags of T_ϕ.

Therefore, Algorithm 6 runs in $\mathcal{O}(nm)$ total time. $\square$

Algorithm 6: Computes (♡) a connected $(r + 5\rho)$-dominating set or (◇) a connected $(r + 3\lambda)$-dominating set for a given graph G with a given tree-decomposition $\mathcal{T}$ with breadth ρ and length λ.

(♡) Set $\phi := 3\rho$.
(◇) Set $\phi := 2\lambda$.
Compute a minimum $(r + \phi)$-covering subtree T_ϕ of $\mathcal{T}$ using Algorithm 4.
if *T_ϕ contains only one bag B* **then**
  Pick an arbitrary vertex $u \in B$, output $C_\phi := \{u\}$, and **stop**.
if *T_ϕ contains exactly two bags B and B'* **then**
  Pick an arbitrary vertex $u \in B \cap B'$, output $C_\phi := \{u\}$, and **stop**.
Pick a leaf of T_ϕ and make it the root of T_ϕ.
Split T_ϕ into a set $\mathbb{L}$ of leaves, a set $\mathbb{P}$ of path segments, and a set $\mathbb{B}$ of branching bags.
Create an empty set C_ϕ.
foreach $P \in \mathbb{P}$ **do**
  Find a shortest path Q_P from $S^\uparrow(P)$ to $S^\downarrow(P)$ and add its vertices into C_ϕ.
  Let $x \in S^\uparrow(P)$ be the start vertex and $y \in S^\downarrow(P)$ be the end vertex of Q_P. Set $v\big(S^\uparrow(P)\big) := x$ and $v\big(S^\downarrow(P)\big) := y$.
foreach $B \in \mathbb{B}$ **do**
  If $v\big(S^\uparrow(B)\big)$ is defined, let $u := v\big(S^\uparrow(B)\big)$. Otherwise, let u be an arbitrary vertex in $S^\uparrow(B)$ and set $v\big(S^\uparrow(B)\big) := u$.
  (♡) Let $v := c(B)$ be the center of B.
  (◇) Let $v := u$.
  Find a shortest path from u to v and add its vertices into C_ϕ.
  foreach $S_i \in \mathcal{S}^\downarrow(B)$ **do**
    If $v(S_i)$ is defined, let $w_i := v(S_i)$. Otherwise, let w_i be an arbitrary vertex in S_i and set $v(S_i) := w_i$.
    Find a shortest path from w_i to v and add the vertices of this path into C_ϕ.
Output C_ϕ.

5. Implications for the *p*-center problem

The *(Connected) p-Center* problem asks, given a graph G and some integer p, for a (connected) vertex set S with $|S| \leq p$ such that S has minimum eccentricity, i.e., there is no (connected) set S' with $\mathrm{ecc}_G(S') < \mathrm{ecc}_G(S)$. It is known (see, e.g., [4]) that the p-Center problem and r-Domination problem are closely related. Indeed, one can solve each of these problems by solving the other problem a logarithmic number of times. Lemma 28 below generalises this observation. Informally, it states that we are able to find a $+\phi$-approximation for the p-Center problem if we can find a good $(r + \phi)$-dominating set.

Lemma 28. *For a given graph G, let D_r be an optimal (connected) r-dominating set and C_p be an optimal (connected) p-center. If, for some non-negative integer ϕ, there is an algorithm to compute a (connected) $(r + \phi)$-dominating set D with $|D| \leq |D_r|$ in $\mathcal{O}\big(T(G)\big)$ time, then there is an algorithm to compute a (connected) p-center C with $\mathrm{ecc}_G(C) \leq \mathrm{ecc}_G(C_p) + \phi$ in $\mathcal{O}\big(T(G) \log n\big)$ time.*

Proof. Let $\mathcal{A}$ be an algorithm which computes a (connected) $(r + \phi)$ dominating set $D = \mathcal{A}(G, r)$ for G with $|D| \leq |D_r|$ in $\mathcal{O}\big(T(G)\big)$ time. Then we can compute a (connected) p-center for G as follows. Make a binary search over the integers $i \in [0, n]$. In each iteration, set $r_i(u) = i$ for each vertex u of G and compute the set $D_i = \mathcal{A}(G, r_i)$. Then, increase i if $|D_i| > p$ and decrease i otherwise. Note that, by construction, $\mathrm{ecc}_G(D_i) \leq i + \phi$. Let D be the resulting set, i.e., out of all computed sets D_i, D is the set with minimal i for which $|D_i| \leq p$. It is easy to see that finding D requires at most $\mathcal{O}\big(T(G) \log n\big)$ time.

Clearly, C_p is a (connected) r-dominating set for G when setting $r(u) = \mathrm{ecc}_G(C_p)$ for each vertex u of G. Thus, for each $i \geq \mathrm{ecc}_G(C_p)$, $|D_i| \leq |C_p| \leq p$ and, hence, the binary search decreases i for next iteration. Therefore, there is an $i \leq \mathrm{ecc}_G(C_p)$ such that $D = D_i$. Hence, $|D| \leq |C_p|$ and $\mathrm{ecc}_G(D) \leq \mathrm{ecc}_G(C_p) + \phi$. □

From Lemma 28, the results in Table 1 and Table 2 follow immediately.

Table 1
Implications of our results for the p-Center problem.

Approach	Approx.	Time
Layering Partition	$+\Delta$	$\mathcal{O}(m \log n)$
Tree-Decomposition	$+\rho$	$\mathcal{O}(nm \log n)$

In what follows, we show that, when using a layering partition, we can achieve the results from Table 1 and Table 2 without the logarithmic overhead.

Theorem 29. *For a given graph G, a $+\Delta$-approximation for the p-Center problem can be computed in linear time.*

Table 2
Implications of our results for the Connected p-Center problem.

Approach	Approx.	Time
Layering Partition	$+2\Delta$	$\mathcal{O}(m\,\alpha(n)\log\Delta\log n)$
Tree-Decomposition	$+\min(5\rho, 3\lambda)$	$\mathcal{O}(nm\log n)$

Proof. First, create a layering partition $\mathcal{T}$ of G. Second, find an optimal p-center $\mathcal{S}$ for $\mathcal{T}$. Third, create a set S by picking an arbitrary vertex of G from each cluster in $\mathcal{S}$. All three steps can be performed in linear time, including the computation of $\mathcal{S}$ (see [20]).

Let C be an optimal p-center for G. Note that, by Lemma 1, C also induces a p-center for $\mathcal{T}$. Therefore, because S induces an optimal p-center for $\mathcal{T}$, Lemma 1 implies that, for each vertex u of G,

$$d_G(u,C) \leq d_G(u,S) \leq d_{\mathcal{T}}(u,\mathcal{S}) + \Delta \leq d_{\mathcal{T}}(u,C) + \Delta \leq d_G(u,C) + \Delta. \quad \square$$

Theorem 30. *For a given graph G, a $+2\Delta$-approximation for the connected p-Center problem can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\min(\Delta,p)\big)$ time.*

Proof. Recall Algorithm 2 for computing a connected $(r+2\Delta)$-dominating set. We create Algorithm 2* by slightly modifying Algorithm 2 as follows. In line 3, instead of computing an r-dominating subtree T_r of $\mathcal{T}$, compute an optimal connected p-center T_p of $\mathcal{T}$ (see [28]). Accordingly, in line 5, compute a δ-dominating subtree of T_p, check in line 7 if $|S_\delta| \leq |T_p|$ (i.e., if $|S_\delta| \leq p$), and output in line 11 the set S_δ with the smallest δ for which $|S_\delta| \leq p$.

Let S be the set computed by Algorithm 2*. As shown in the proof of Theorem 10, it follows from Lemma 8 and Corollary 9 that S is connected, $|S| \leq p$, and $S = S_\delta$ for some $\delta \leq \Delta$.

Now, let C be an optimal connected p-center for G. Clearly, by definition of C and by Lemma 1, $\mathrm{ecc}_G(C) \leq \mathrm{ecc}_G(S_\delta) \leq \mathrm{ecc}_{\mathcal{T}}(T_\delta) + \Delta$. Because T_δ is a δ-dominating subtree of T_p, $\mathrm{ecc}_{\mathcal{T}}(T_\delta) \leq \mathrm{ecc}_{\mathcal{T}}(T_p) + \delta$. Let T_C be the subtree of $\mathcal{T}$ induced by C, i.e., the subtree of $\mathcal{T}$ induced by the clusters which contain vertices of C. Then, because T_p is an optimal connected p-center for $\mathcal{T}$ and, clearly, $|T_C| \leq p$, $\mathrm{ecc}_{\mathcal{T}}(T_p) \leq \mathrm{ecc}_{\mathcal{T}}(T_C)$. Therefore, since $\delta \leq \Delta$, $\mathrm{ecc}_G(C) \leq \mathrm{ecc}_G(S_\delta) \leq \mathrm{ecc}_{\mathcal{T}}(T_C) + 2\Delta$ and, by Lemma 1, $\mathrm{ecc}_G(C) \leq \mathrm{ecc}_G(S_\delta) \leq \mathrm{ecc}_G(C) + 2\Delta$.

As shown in the proof of Theorem 10, the one-sided binary search of Algorithm 2* has at most $\mathcal{O}(\log\Delta)$ iterations. Because $|T_p| \leq p$, T_p contains a cluster with eccentricity at most $\lceil p/2 \rceil$ in T_p. Therefore, for any $\delta \geq \lceil p/2 \rceil$, $|T_\delta| = |S_\delta| = 1$ and, thus, the algorithm decreases δ. Hence, the one-sided binary search of Algorithm 2* has at most $\mathcal{O}(\log p)$ iterations. Therefore, the algorithm runs in at most $\mathcal{O}\big(m\,\alpha(n)\log\min(\Delta,p)\big)$ total time. $\square$

References

[1] A. Abboud, V.V. Williams, J. Wang, Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs, in: Proceedings of the 27th Annual ACM–SIAM Symposium on Discrete Algorithms, 2016, pp. 377–391.
[2] M. Abu-Ata, F.F. Dragan, Metric tree-like structures in real-life networks: an empirical study, Networks 67 (1) (2016) 49–68.
[3] G. Borradaile, H. Le, Optimal dynamic program for r-domination problems over tree decompositions, in: IPEC 2016, in: LIPIcs. Leibniz Int. Proc. Inform., vol. 63, 2017, pp. 8:1–8:23.
[4] A. Brandstädt, V. Chepoi, F.F. Dragan, The algorithmic use of hypertree structure and maximum neighbourhood orderings, Discrete Appl. Math. 82 (1–3) (1998) 43–77.
[5] A. Brandstädt, V. Chepoi, F.F. Dragan, Distance approximating trees for chordal and dually chordal graphs, J. Algorithms 30 (1999) 166–184.
[6] P. Chalermsook, M. Cygan, G. Kortsarz, B. Laekhanukit, P. Manurangsi, D. Nanongkai, D. Trevisan, From gap-ETH to FPT-inapproximability: clique, dominating set, and more manuscript, CoRR, arXiv:1708.04218, 2017.
[7] Y. Chen, B. Lin, The constant inapproximability of the parameterized dominating set problem, manuscript, CoRR, arXiv:1511.00075, 2015.
[8] V. Chepoi, F.F. Dragan, A note on distance approximating trees in graphs, European J. Combin. 21 (2000) 761–766.
[9] V.D. Chepoi, F.F. Dragan, B. Estellon, M. Habib, Y. Vaxes, Diameters, centers, and approximating trees of δ-hyperbolic geodesic spaces and graphs, in: Proceedings of the 24th Annual ACM Symposium on Computational Geometry, SoCG 2008, 2008, pp. 59–68.
[10] V. Chepoi, B. Estellon, Packing and covering δ-hyperbolic spaces by balls, in: APPROX-RANDOM 2007, in: Lecture Notes in Comput. Sci., vol. 4627, 2007, pp. 59–73.
[11] M. Chlebík, J. Chlebíková, Approximation hardness of dominating set problems in bounded degree graphs, Inform. and Comput. 206 (2008) 1264–1275.
[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, 3rd ed., MIT Press, 2009.
[13] E.D. Demaine, F.V. Fomin, M.T. Hajiaghayi, D.M. Thilikos, Fixed-parameter algorithms for (k,r)-center in planar graphs and map graphs, ACM Trans. Algorithms 1 (1) (2005) 33–47.
[14] Y. Dourisboure, F.F. Dragan, C. Gavoille, C. Yan, Spanners for bounded tree-length graphs, Theoret. Comput. Sci. 383 (1) (2007) 34–44.
[15] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, 1999.
[16] F.F. Dragan, HT-graphs: centers, connected r-domination and Steiner trees, Comput. Sci. J. Moldova 1 (2) (1993) 64–83.
[17] K. Edwards, K. Kennedy, I. Saniee, Fast approximation algorithms for p-centers in large δ-hyperbolic graphs, in: WAW 2016, in: Lecture Notes in Comput. Sci., vol. 10088, 2016, pp. 60–73.
[18] B. Escoffier, V.Th. Paschos, Completeness in approximation classes beyond APX, Theoret. Comput. Sci. 359 (1–3) (2006) 369–377.
[19] A.E. Feldmann, Fixed parameter approximations for k-center problems in low highway dimension graphs, in: ICALP 2015, in: Lecture Notes in Comput. Sci., vol. 9135, 2015, pp. 588–600.
[20] G.N. Frederickson, Parametric search and locating supply centers in trees, in: WADS 1991, in: Lecture Notes in Comput. Sci., vol. 519, 1991, pp. 299–319.
[21] T. Gonzalez, Clustering to minimize the maximum intercluster distance, Theoret. Comput. Sci. 38 (1985) 293–306.

[22] S. Guha, S. Khuller, Approximation algorithms for connected dominating sets, Algorithmica 20 (4) (1998) 374–387.
[23] I. Katsikarelis, M. Lampis, V.Th. Paschos, Structural parameters, tight bounds, and approximation for (k, r)-center, in: ISAAC 2017, in: LIPIcs. Leibniz Int. Proc. Inform., vol. 92, pp. 50:1–50:13, 2017.
[24] A. Leitert, F.F. Dragan, Parameterized approximation algorithms for some location problems in graphs, in: COCOA 2017, in: Lecture Notes in Comput. Sci., vol. 10628, 2017, pp. 348–361.
[25] D. Marx, Efficient approximation schemes for geometric problems?, in: ESA 2005, in: Lecture Notes in Comput. Sci., vol. 3669, 2005, pp. 448–459.
[26] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford Lecture Ser. Math. Appl., Oxford University Press, 2006.
[27] R. Raz, S. Safra, A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP, in: Proc. 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 475–484.
[28] W.C.-K. Yen, C.-T. Chen, The p-center problem with connectivity constraint, Appl. Math. Sci. 1 (27) (2007) 1311–1324.