Parameterized Approximation Algorithms for Some Location Problems in Graphs

Arne Leitert[1(✉)] and Feodor F. Dragan[2]

[1] Department of Computer Science, Central Washington University, Ellensburg, WA, USA
arne.leitert@cwu.edu
[2] Department of Computer Science, Kent State University, Kent, OH, USA
dragan@cs.kent.edu

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 generalization 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

X. Gao et al. (Eds.): COCOA 2017, Part II, LNCS 10628, pp. 348–361, 2017.
https://doi.org/10.1007/978-3-319-71147-8_24

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 minimized. If, additionally, C is required to induce a connected subgraph of G, then one gets the *Connected p-Center problem*.

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., [13,20] for an introduction to parameterized complexity). It was the first problem to be shown W[2]-complete [13], 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 [18]. 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 [21]. The problem is in fact Log-APX-complete [16] and it is unlikely that there is a good FPT approximation algorithm for it (see [5,6]).

The p-Center problem is known to be NP-hard on graphs. However, for it, a simple and efficient factor-2 approximation algorithm exists [17]. 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 [17] for more details). The NP-hardness of the Connected p-Center problem is shown in [22].

Recently, in [9], 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 [9] for details). We call such a D' an $(r + 2\delta)$*-dominating set* of G. Later, in [15], 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 [9,15]. Unfortunately, the results of [9,15] 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 $\varDelta$ 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 $(r+\varDelta)$-dominating set D' with $|D'| \leq |D|$ can be computed in linear time; a connected $(r+2\varDelta)$-dominating set C' with $|C'| \leq |C|$ can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\varDelta\big)$ time (where $\alpha(n)$ is the inverse Ackermann function); a $+\varDelta$-approximation for the p-Center problem can be computed in linear time; a $+2\varDelta$-approximation for the connected p-Center problem can be computed in $\mathcal{O}\big(m\,\alpha(n)\log\min(\varDelta, p)\big)$ 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 [9,15], notice that, for any graph G, its hyperbolicity δ is at most $\varDelta$ [2] and at most two times its tree-breadth ρ [8], 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 $\varDelta$ is at most 2. Additionally, as shown in [10], 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 $\varDelta$.

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 $+c\varDelta\log n$-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.

Due to space limitations, all proofs are omitted. Additionally, Sect. 4 is limited to the main ideas of our algorithm. A full version of the paper can be found in [19].

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 $N_G^r[v] = \{ u \mid d_G(u, v) \leq r \}$ 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.

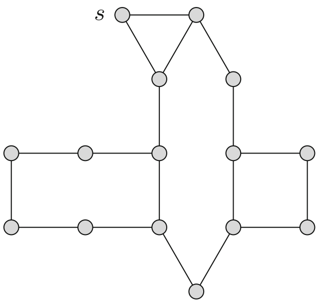
The *degree* of a vertex v is the number of vertices adjacent to it. For a vertex set S, let $G[S]$ denote the subgraph of G induced by S. A vertex set S is a *separator* for two vertices u and v in G if each path from u to v contains a vertex $s \in S$; in this case we say S *separates* u from v.

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 $B \subseteq N_G^\rho[v]$. 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.

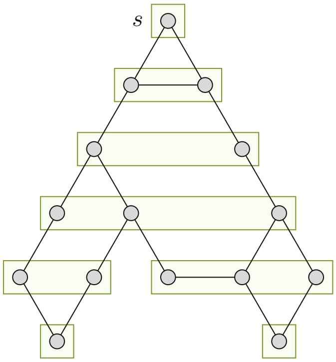
For a rooted tree T, let $\Lambda(T)$ denote the number of leaves of T. For the case when T contains only one node, let $\Lambda(T) := 0$. With α, we denote the inverse Ackermann function (see, e. g., [11]). It is well known that α grows extremely slowly. For $x = 10^{80}$ (estimated number of atoms in the universe), $\alpha(x) \leq 4$.

3 Using a Layering Partition

The concept of a *layering partition* was introduced in [4,7]. 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 $v \in C'$. Figure 1 gives an example for such a partition. A layering partition of a graph can be computed in linear time [7].



(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 [7].

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, $d_{\mathcal{T}}(u, v) \leq d_G(u, v) \leq d_{\mathcal{T}}(u, v) + \Delta$.*

Theorem 1 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 1. *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.*

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 [14]. Lemma 2 below shows that T_r gives a lower bound for the cardinality of D_r.

Lemma 2. *If T_r contains more than one cluster, each connected r-dominating set of G intersects all clusters of T_r. Therefore, $|T_r| \leq |D_r|$.*

As we show later in Corollary 1, each connected vertex set $S \subseteq V$ that intersects each cluster of T_r gives an $(r + \varDelta)$-dominating set for G. It follows from Lemma 2 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 $(r + 2\varDelta)$-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 3. *For each vertex v of G, $d_{\mathcal{T}}(v, T_\delta) \leq r(v) + \delta$.*

Because the diameter of each cluster is at most $\varDelta$, Lemmas 1 and 3 imply the following.

Corollary 1. *If a vertex set intersects all clusters of T_δ, it is an $\big(r + (\delta + \varDelta)\big)$-dominating set of G.*

Lemma 4. $|T_\delta| \leq |T_r| - \delta \cdot \varLambda(T_\delta)$.

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 from a connected set using an approach which is similar to Kruskal's algorithm for minimum spanning trees.

Let $\mathcal{L} = \big\{C_1, C_2, \ldots, C_\lambda\big\}$ be the leaf clusters of T_δ (excluding the root) with either $\lambda = \varLambda(T_\delta) - 1$ if the root of T_δ is a leaf, or with $\lambda = \varLambda(T_\delta)$ otherwise. We construct a set $\mathcal{P} = \big\{P_1, P_2, \ldots, P_\lambda\big\}$ 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 $\mathcal{C}_i$ and find a shortest path P_i in G form v to C'_i. Add P_i to $\mathcal{P}$ and continue with the next cluster in $\mathcal{L}$. Figure 2 gives an example.

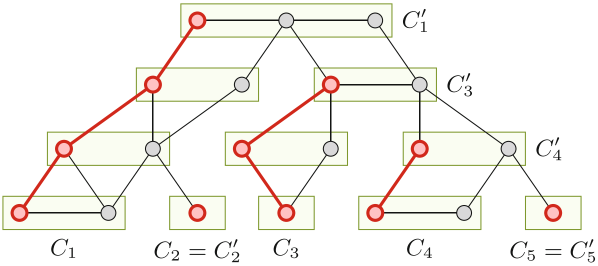


Fig. 2. Example for the set $\mathcal{P}$ for a subtree of a layering partition. Paths are shown in red. Each path P_i, with $1 \le i \le 5$, starts in the leaf C_i and ends in the cluster C'_i.For $i = 2$ and $i = 5$, P_i contains only one vertex.

Lemma 5. *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., $|C \cap P_i| = 1$.*

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 $\mathcal{V} = \{V_1, V_2, \dots, V_\lambda\}$ 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 $d(u) + d(v) \le d(x) + d(y)$.

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.

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.

Let $\mathcal{L} = \big\{C_1, C_2, \ldots, C_\lambda\big\}$ be the set of clusters excluding the root that are leaves of T_δ.
Create an empty set $\mathcal{P}$.
foreach *cluster* $C_i \in \mathcal{L}$ **do**
  Select an arbitrary vertex $v \in C_i$.
  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.
  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).
  Add P_i to $\mathcal{P}$.
  Flag each cluster intersected by P_i.
Create a set $S_\delta := \bigcup_{P_i \in \mathcal{P}} P_i$.
Perform a BFS on G starting at all paths $P_i \in \mathcal{P}$ simultaneously. This results in a partition $\mathcal{V} = \big\{V_1, V_2, \ldots, V_\lambda\big\}$ 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 $d(v) := d_G(v, P(v))$.
Create a Union-Find data structure and add all vertices of G such that $\mathrm{Find}(v) = i$ if and only if $v \in V_i$.
Determine the edge set $E' = \{\, uv \mid \mathrm{Find}(u) \neq \mathrm{Find}(v) \,\}$.
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.
for $i := 1$ **to** $|E'|$ **do**
  Let $uv = e_i$.
  if $\mathrm{Find}(u) \neq \mathrm{Find}(v)$ **then**
    Add the shortest path from u to $P(u)$ to S_δ.
    Add the shortest path from v to $P(v)$ to S_δ.
    $\mathrm{Union}(u, v)$
Output S_δ.

Lemma 6. *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.*

Because, for each integer $\delta \geq 0$, $|S_\delta| \leq |T_\delta| + \Delta \cdot \Lambda(T_\delta)$ (Lemma 6) and $|T_\delta| \leq |T_r| - \delta \cdot \Lambda(T_\delta)$ (Lemma 4), we have the following.

Corollary 2. *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 $\Omega\big(n^2\big)$ 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 2, 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 $\mathcal{T}$ of G.
For each cluster C of $\mathcal{T}$, set $r(C) := \min_{v \in C} r(v)$.
Compute a minimum r-dominating subtree T_r for $\mathcal{T}$ (see [14]).
One-Sided Binary Search *over δ, starting with $\delta = 0$*
  Create a minimum δ-dominating subtree T_δ of T_r (i. e., T_δ is a minimum $(r + \delta)$-dominating subtree for $\mathcal{T}$).
  Run Algorithm 1 on T_δ and let the set S_δ be the corresponding output.
  if $|S_\delta| \leq |T_r|$ **then**
    Decrease δ.
  else
    Increase δ.
Output S_δ with the smallest δ for which $|S_\delta| \leq |T_r|$.

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

4 Using a Tree-Decomposition

Theorems 1 and 2 respectively show how to compute an $(r+\varDelta)$-dominating set in linear time and a connected $(r+2\varDelta)$-dominating set in $\mathcal{O}\big(m\,\alpha(n)\log\varDelta\big)$ time. It is known that the maximum diameter $\varDelta$ 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 $\mathrm{tl}(G)=\lambda$, $\varDelta \leq 3\lambda$ [12].

Corollary 3. *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 4. *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 graph $G=(V,E)$ and a tree-decomposition $\mathcal{T}$ of G with known breadth ρ and length λ. Additionally, we assume that, for each bag B of $\mathcal{T}$, we know a vertex $c(B)$, called *center of* B, with $B \subseteq N_G^{\rho}[c(B)]$. 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.

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 $\mathcal{T}$*.

Lemma 7. *One can compute a minimum r-covering subtree T_r of $\mathcal{T}$ in $\mathcal{O}(nm)$ time.*

Next, we use a 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.

Theorem 3. *Let D be a minimum r-dominating set for a given graph G. Given a tree-decomposition with breadth ρ for G, one can compute an $(r+\rho)$-dominating set S with $|S| \leq |D|$ in $\mathcal{O}(nm)$ time.*

Now, 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 $(\heartsuit)$

for parts which are only relevant to determine a connected $(r+5\rho)$-dominating set and use the label $(\diamondsuit)$ for parts which are only relevant to determine a connected $(r+3\lambda)$-dominating set.

For $(\heartsuit)$ $\phi = 3\rho$ or $(\diamondsuit)$ $\phi = 2\lambda$, let T_ϕ be a minimum $(r+\phi)$-covering subtree of $\mathcal{T}$. 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.

Notation. Let T_ϕ be a rooted tree such that its root R is a leaf. 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.

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 phenomena 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 $Q_P \setminus S^\uparrow(P)$.

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 $S \in \mathcal{S}^\downarrow(B) \cup \{S^\uparrow(B)\}$. 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 $t = \nu(S^\uparrow(B))$. 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.

($\diamondsuit$) 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.

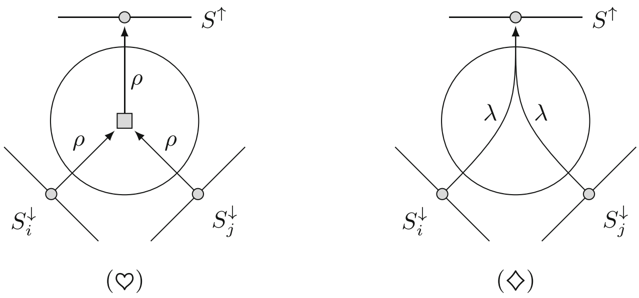


Fig. 3. Construction of the set C_B for a branching bag B.

Theorem 4. *For a given graph which has an unknown minimum connected r-dominating set D_r, one can compute a connected $(r+(\phi+\lambda))$-dominating set C_ϕ with $|C_\phi| \leq |D_r|$ in $\mathcal{O}(nm)$ time.*

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., [3]) 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 8 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 8. *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.*

From Lemma 8, the results in Tables 1 and 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	$+\rho$	$\mathcal{O}(nm \log n)$

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	$+\min(5\rho, 3\lambda)$	$\mathcal{O}(nm \log n)$

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

Theorem 5. *For a given graph G, a $+\Delta$-approximation for the p-Center problem can be computed in linear time.*

Theorem 6. *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.*

References

1. Abboud, A., Williams, V.V., Wang, J.: 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, pp. 377–391 (2016)
2. Abu-Ata, M., Dragan, F.F.: Metric tree-like structures in real-life networks: an empirical study. Networks **67**(1), 49–68 (2016)
3. Brandstädt, A., Chepoi, V., Dragan, F.F.: The algorithmic use of hypertree structure and maximum neighbourhood orderings. Discrete Appl. Math. **82**(1–3), 43–77 (1998)
4. Brandstädt, A., Chepoi, V., Dragan, F.F.: Distance approximating trees for chordal and dually chordal graphs. J. Algorithms **30**, 166–184 (1999)

5. Chalermsook, P., Cygan, M., Kortsarz, G., Laekhanukit, B., Manurangsi, P., Nanongkai, D., Trevisan, D.: From Gap-ETH to FPT-Inapproximability: Clique, Dominating Set, and More Manuscript, CoRR abs/1708.04218 (2017)
6. Chen, Y., Lin, B.: The Constant Inapproximability of the Parameterized Dominating Set Problem, Manuscript, CoRR abs/1511.00075 (2015)
7. Chepoi, V., Dragan, F.F.: A note on distance approximating trees in graphs. Eur. J. Comb. **21**, 761–766 (2000)
8. Chepoi, V.D., Dragan, F.F., Estellon, B., Habib, M., Vaxes, Y.: 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, pp. 59–68 (2008)
9. Chepoi, V., Estellon, B.: Packing and covering δ-hyperbolic spaces by balls. In: Charikar, M., Jansen, K., Reingold, O., Rolim, J.D.P. (eds.) APPROX/RANDOM -2007. LNCS, vol. 4627, pp. 59–73. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74208-1_5
10. Chlebík, M., Chlebíková, J.: Approximation hardness of dominating set problems in bounded degree graphs. Inf. Comput. **206**, 1264–1275 (2008)
11. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 3rd edn. MIT Press, Cambridge (2009)
12. Dourisboure, Y., Dragan, F.F., Gavoille, C., Yan, C.: Spanners for bounded tree-length graphs. Theor. Comput. Sci. **383**(1), 34–44 (2007)
13. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
14. Dragan, F.F.: HT-graphs: centers, connected r-domination and Steiner trees. Comput. Sci. J. Moldova **1**(2), 64–83 (1993)
15. Edwards, K., Kennedy, S., Saniee, I.: Fast approximation algorithms for p-centers in large δ-hyperbolic graphs. In: Bonato, A., Graham, F.C., Prałat, P. (eds.) WAW 2016. LNCS, vol. 10088, pp. 60–73. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49787-7_6
16. Escoffier, B., Paschos, V.T.: Completeness in approximation classes beyond APX. Theor. Comput. Sci. **359**(1–3), 369–377 (2006)
17. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theor. Comput. Sci. **38**, 293–306 (1985)
18. Guha, S., Khuller, S.: Approximation algorithms for connected dominating sets. Algorithmica **20**(4), 374–387 (1998)
19. Leitert, A., Dragan, F.F.: Parametrized Approximation Algorithms for some Location Problems in Graphs, Manuscript, CoRR abs/1706.07475 (2017)
20. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, Oxford (2006)
21. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 475–484 (1997)
22. Yen, W.C.-K., Chen, C.-T.: The p-center problem with connectivity constraint. Appl. Math. Sci. **1**(27), 1311–1324 (2007)