Improved Algorithms and Complexity Results for Power Domination in Graphs[*]

Jiong Guo[1], Rolf Niedermeier[1], and Daniel Raible[2]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, D-07743 Jena, Germany. {guo,niedermr}@minet.uni-jena.de
[2] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany. raibk@informatik.uni-tuebingen.de

Abstract. The Power Dominating Set problem is a variant of the classical domination problem in graphs: Given an undirected graph $G = (V, E)$, find a minimum $P \subseteq V$ such that all vertices in V are "observed" by vertices in P. Herein, a vertex observes itself and all its neighbors, and if an observed vertex has all but one of its neighbors observed, then the remaining neighbor becomes observed as well. We show that Power Dominating Set can be solved by "bounded-treewidth dynamic programs." Moreover, we simplify and extend several NP-completeness results, particularly showing that Power Dominating Set remains NP-complete for planar graphs, for circle graphs, and for split graphs. Specifically, our improved reductions imply that Power Dominating Set parameterized by $|P|$ is W[2]-hard and cannot be better approximated than Dominating Set.

1 Introduction

Domination is a central theme in graph theory. The basic problem is: given an undirected graph $G = (V, E)$, determine a minimum vertex set $D \subseteq V$ such that each $v \in V$ is contained in D or v is a neighbor of at least one vertex in D. The corresponding decision problem Dominating Set (DS) is NP-complete. Unless unlikely collapses in structural complexity theory occur, the polynomial-time approximation factor is $\Theta(\log n)$ [7]. Numerous variations of Dominating Set exist. For instance, the Connected Dominating Set additionally requires that the dominating set D induces a connected subgraph of G. Opposite to Dominating Set, Connected Dominating Set carries some form of *non-locality*: the correctness of the dominating set cannot be decided by locally checking every direct neighborhood. In this work, we study another "non-local" variant of domination which appears in electric power networks [8]—Power Dominating Set (PDS). This variant is motivated by monitoring an electric power network, where one is asked to place a minimum number of so-called *phase measurement units* (PMU) at some locations in the system to measure the state variables (for example, the voltage magnitude etc.). Intuitively, we have a more complex

---

degree of non-locality in PDS. Whereas Connected Dominating Set only refers to a property of the solution set, PDS has the non-locality in its domination mechanism: A vertex may dominate vertices at *arbitrary* distance when certain conditions are fulfilled.

For Dominating Set, we have one "observation rule" concerning vertices in the dominating set: these vertices observe (dominate) themselves and all their neighbors (and nothing else). The goal is to get all vertices observed by a minimum number of observers. By way of contrast, we have an additional observation rule in the case of PDS. This rule says that for an already observed vertex whose all but one neighbors are already observed, the one remaining neighbor becomes observed as well.[1] Note that the second observation rule brings in non-locality. The graph-theoretical and algorithmic study of PDS has been initiated by Haynes et al. [8]. They show that PDS is NP-complete for general graphs as well as for bipartite and chordal graphs. Moreover, they present a linear-time algorithm to solve PDS in trees. We improve their results in the following ways.

1. We present simplified and "stronger" NP-completeness proofs, giving all results of Haynes et al. in a less technical way and additionally implying NP-completeness also for planar graphs, circle graphs, and split graphs. Moreover, our simple reductions preserve parameterized complexity and approximability. So we can conclude that, in case of general graphs, PDS is W[2]-hard and it is only $\Theta(\log n)$-approximable unless unlikely collapses in structural complexity theory occur.[2]
2. We present a much simpler linear-time algorithm for PDS in trees than the one presented in [8].
3. Our main result is a dynamic programming algorithm for PDS in graphs of bounded treewidth, answering an open question of Haynes et al. [8]. Independently, fixed-parameter tractability with respect to parameter treewidth was also shown by Kneis et al. [10] using descriptive complexity tools. They express PDS in monadic second-order logic[3], which implies algorithms of highly super-exponential running time. The dependence of the combinatorial explosion on the parameter treewidth is non-elementary. In contrast, we describe and analyze a direct algorithm for PDS with much improved running time.

Let us return to the issue of non-locality. Demaine and Hajiaghayi [5], answering an open question from [1], showed that Connected Dominating Set can be solved by bounded-treewidth dynamic programs. It was not believed in [1] that such non-local properties as "connectedness" could be captured in this way. In case of PDS, the "even worse" degree of non-locality appears to make things even harder. Still, a dynamic programming solution exists. Due to the lack of space, some proofs are deferred to a long version of this extended abstract.

---

[1] The original definition of Haynes et al. [8] is a bit more complicated and the equivalent definition presented here is due to Kneis et al. [10].

[2] Basically the same reduction was independently shown in [10].

[3] This confirms a statement made by Petr Hliněný in a discussion with Peter Rossmanith and Rolf Niedermeier at the IWPEC 2004 meeting in Bergen, Norway, September 2004.

2 Preliminaries

All graphs in this work are simple and without self-loops. For the definitions of the considered graph classes, we refer to [4]. For a vertex v in graph G, we denote by $N_G(v)$ the open neighborhood of v in G. By $N_G[v]$, we refer to the closed neighborhood of v. This naturally generalizes to $N_G(U)$ and $N_G[U]$ for U being a set of vertices. Moreover, for notion concerning approximation algorithms we refer to [2] and for parameterized complexity we refer to [6].

To define power domination in graphs, we use two (simplified) *observation rules* due to [10].

Observation Rule 1 (OR1): A vertex in the power domination set observes itself and all of its neighbors.

Observation Rule 2 (OR2): If an observed vertex v of degree $d \geq 2$ is adjacent to $d - 1$ observed vertices, then the remaining unobserved vertex becomes observed as well.

Given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, POWER DOMINATING SET (PDS) asks whether there is a set $P \subseteq V$ with $|P| \leq k$ which observes all vertices in V with respect to the two observation rules OR1 and OR2. Herein, P is called the *power dominating set* of G. Note that every connected graph of maximum vertex degree 2 has a power dominating set of size 1. The classical DOMINATING SET problem can be defined by simply omitting OR2.

Lemma 1. *If G is a connected graph with at least one vertex of degree three or higher, then there is always a minimum power dominating set which contains only vertices with degree at least three.*

Lemma 1 is due to Haynes et al. [8] who also show that, given an arbitrary power dominating set P for a connected graph with maximum degree at least three, one can construct in linear time a power dominating set P' with $|P'| \leq |P|$ containing only vertices with degree at least three.

The central concept of this work are tree decompositions of graphs and their use with respect to dynamic programming as, e.g., described in [3, 9].

Definition 1. *Let $G = (V, E)$ be a graph. A* tree decomposition *of G is a pair $\langle \{X_i \mid i \in I\}, T \rangle$, where each X_i is a subset of V, called a* bag, *and T is a tree with the elements of I as nodes. The following three properties must hold:*

1. $\bigcup_{i \in I} X_i = V$;

2. for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$;

3. for all $i, j, k \in I$, if j lies on the path between i and k in T, then $X_i \cap X_k \subseteq X_j$.

The width *of $\langle \{X_i \mid i \in I\}, T \rangle$ equals $\max\{|X_i| \mid i \in I\} - 1$. The* treewidth *of G is the minimum k such that G has a tree decomposition of width k.*

Definition 2. *A tree decomposition $\langle \{X_i \mid i \in I\}, T \rangle$ is called a* nice tree decomposition *if T is rooted and the following conditions are satisfied:*

1. Every node of the tree T has at most 2 children.

2. If a node i has two children j and k, then $X_i = X_j = X_k$ (in this case i is called a JOIN NODE).

3. If a node i has one child j, then one of the following holds:

(1) $|X_i| = |X_j| + 1$ *and* $X_j \subset X_i$ *(in this case i is called an* INSERT NODE*),*
(2) $|X_i| = |X_j| - 1$ *and* $X_i \subset X_j$ *(in this case i is called a* FORGET NODE*).*

Every tree decomposition can be transformed into a nice tree decomposition [9, Lemma 13.1.3].

Lemma 2. *Given a width-k tree decomposition with $O(n)$ nodes of a graph G, where n is the number of vertices of G, then we can find a nice tree decomposition of G that also has width k and $O(n)$ nodes in $O(n)$ time.*

3 Power Dominating Set vs. Dominating Set

Haynes et al. [8] show the NP-completeness of PDS by giving a reduction from 3-SAT. This also shows that PDS is NP-complete even when the input graph is bipartite or chordal. We show, for general graphs, that PDS is at least as hard to solve as DS by reducing DS to PDS. Moreover, this implies the NP-completeness of PDS also in case of bipartite, chordal, circle, and planar graphs, and the inapproximability and parameterized intractability results for DS also transfer to PDS in this way. The second reduction (from VERTEX COVER) in this section shows the NP-completeness of PDS for split graphs.

Theorem 1. POWER DOMINATING SET *is NP-complete for bipartite, chordal, circle, and planar graphs.*

Proof. Since it can be easily decided whether a vertex set P is a power dominating set, PDS is in NP. To show NP-hardness, we give a reduction from DS.

Given a DS-instance $G = (V, E)$, we construct a PDS-instance $G' = (V \cup V_1, E \cup E_1)$ simply by attaching newly introduced degree-1 vertices to all vertices from V.

For a dominating set D of G, we set $P := D$. By definition, all vertices from V are observed by OR1. Applying OR2 to every vertex in V, the vertices in V_1 become observed as well. Thus, P is a power dominating set of G'.

If G' has a power dominating set P with $|P| = k$, then we can assume due to Lemma 1 that each vertex of P has degree at least three. This implies that $P \cap V_1 = \emptyset$. The proof is by contradiction. Assume that P is not a dominating set of G. Then, there is a vertex $v \in V$ with $N_G[v] \cap P = \emptyset$. We also have that $N_{G'}[v] \cap P = \emptyset$. Vertex v can get observed in G' only by applying OR2 to one of its neighbors in G'. Denote this neighbor by u. Let v' denote the newly introduced degree-1 neighbor of v in G'. It is easy to see that u cannot be v'. Hence, $u \in V$. Furthermore, u has also a degree-1 neighbor $u' \in V_1$ in G'. Since $u \notin P$ and $u' \notin P$, u' can be observed only by applying OR2 to u. However, this is impossible since u has two unobserved neighbors v and u'. Thus, P cannot be a power domination set of G', yielding a contradiction.

It is easy to verify that the reduction preserves the properties "bipartite," "chordal," "circle," and "planar." Hence, the NP-completeness of PDS follows from the NP-completeness of DS for these graph classes [11]. □

Graph classes	DOMINATING SET	POWER DOMINATING SET
bipartite	NP-c	NP-c
chordal	NP-c	NP-c
circle	NP-c	NP-c
comparability	NP-c	NP-c
planar	NP-c	NP-c
split	NP-c	NP-c
AT-free	poly. time	
cocomparability	poly. time	
distance hereditary	poly. time	
dually chordal	poly. time	
interval	poly. time	
k-polygon ($k \geq 3$)	poly. time	
partial k-tree ($k \geq 1$)	poly. time	poly. time
permutation	poly. time	
strongly chordal	poly. time	

Table 1. The first column is from [11]. Partial k-trees are the same as graphs of bounded treewidth; the polynomial-time result for this row is shown in Section 4. Empty entries mean that this has not been studied yet.

The reduction in the proof of Theorem 1 was independently achieved in [10]. Observe that it is a gap-preserving as well as a parameterized reduction. This implies that all negative results with respect to the approximability and parameterized tractability with respect to the solution size valid for DS also are valid for PDS. It is hard to approximate PDS better than $\Theta(\log n)$ [7] and PDS is W[2]-hard with the size of the power dominating set as parameter [6].

Next, we show that PDS is NP-complete for split graphs. The reduction is from the NP-complete VERTEX COVER problem: Given an undirected graph $G = (V, E)$ and an integer $k \geq 1$, is there a set $C \subseteq V$ with $|C| \leq k$ such that each edge has at least one endpoint in C.

Theorem 2. POWER DOMINATING SET *is NP-complete on split graphs.*

Proof. For a VC-instance with graph $G = (V, E)$, we construct a split graph G' from G as follows. For each edge $e = \{u, v\} \in E$ we add a new vertex w_e and two edges between w_e and u and between w_e and v to G. We denote the set of the vertices w_e by V_E. Moreover, we introduce for each vertex $v \in V$ a new degree-1 vertex v' and an edge between v and v'. The set of these vertices is denoted by V_1. Finally, we complete the graph induced by V into a clique. Note that the subgraph of G' induced by the vertices in $V_e \cup V_1$ is an independent set. Thus, G' is a split graph. The proof for the claim that G has a vertex cover of size k iff G' has a power dominating set of size k uses a similar argument as the proof of Theorem 1. □

Altogether, Table 1 compares the computational complexity of PDS and DS.

4   Dynamic Program for Graphs of Bounded Treewidth

Haynes et al. [8] give a linear-time algorithm for PDS in trees. As a warm-up, we start with a much simpler linear-time algorithm for these "treewidth-1 graphs." Without loss of generality, we assume that the input tree T is rooted at a degree-1 node r and the *depth* of a node v is defined as the length of the path between v and r. The algorithm follows a bottom-up strategy:

1. Sort the inner nodes of T in a list L according to the non-increasing order of their depths in T;

2. **while** $L \neq \{r\}$ **do**

2.1 $v \leftarrow$ the first node in L; $L \leftarrow L \setminus \{v\}$;

2.2 **if** v has at least two unobserved children **then**

2.2.1 $P \leftarrow P \cup \{v\}$;

2.2.2 Apply the two observation rules as long as possible;

3. **if** r is unobserved **then**

3.1 $P \leftarrow P \cup \{r\}$;

4. **return** P;

The linear running time of the algorithm is due to the fact that the algorithm examines for each inner node only its children. With proper data structures, the application of the observation rules can be performed in linear time.

Our linear-time algorithm for graphs of bounded treewidth uses the same strategy as the algorithms for DS [12, 1], i.e., bottom-up dynamic programming from the leaves to the root. In the following, we demonstrate that there are two difficulties associated with OR2 that make PDS "harder" than DS and which cannot be solved by a simple modification of the algorithms for DS.

Firstly, with OR2, there are more possibilities for a vertex to be observed: it can be observed by OR1 or OR2. More precisely, the application of OR2 implies that there is a certain "observation dependency" between two vertices, i.e., one vertex not in the power dominating set that becomes observed can make one of its neighbors observed. This observation dependency does not exist in DS. There, the domination status of one vertex that is not in the dominating set has no effect on the domination status of other vertices. Therefore, in order to describe this observation dependency in the dynamic programming, the three states defined in the algorithm for DS for the vertices in a bag, namely, "belonging to the dominating set," "already dominated at the current bag," and "not yet dominated at the current bag," are not sufficient. Secondly, only introducing further vertex states cannot settle the problem with the observation dependency. For example, assume that one defines the following additional state for the vertices in a bag: "already observed at the current bag by applying OR2 to one of its neighbors." Then, there could emerge a "cycle of observation dependencies:" Consider a simple cycle as the input graph, one might assign the new state "observed due to OR2" to each vertex of the cycle. This is "locally correct" but globally it is false because the reasoning is done in a circular fashion without "global justification."

Our answer to these two difficulties is to define, in addition to the vertex states, three states for the edges in the subgraph induced by the vertices in a

bag. In fact, these states give one of three possible *orientations* to an undirected edge $\{u, v\}$, orienting it from u to v, orienting it from v to u, or leaving it unoriented. These orientations express observation dependencies.

4.1 Valid Orientations of Undirected Graphs

Definition 3. *An* orientation *of an undirected graph* $G = (V, E)$ *is a graph* $D = (V, E_1 \cup E_2)$ *such that for each* $\{u, v\} \in E$ *there is either a directed edge* (u, v) *or* (v, u) *in* E_1 *or an undirected edge* $\{u, v\}$ *in* E_2, *and* $\{u, v\} \in E$ *for each directed edge* $(u, v) \in E_1$ *and undirected edge* $\{u, v\} \in E_2$. *The* indegree *of a vertex* v *in* D, *denoted by* $d^-(v)$, *is defined as* $|\{(u, v) \mid (u, v) \in E_1\}|$ *and the* outdegree *of* v, *denoted by* $d^+(v)$, *as* $|\{(v, u) \mid (v, u) \in E_1\}|$. *The subgraph* $D[V']$ *induced by* $V' \subseteq V$ *in* D *is called a* suborientation.

Note that in the standard graph theory literature, an orientation of an undirected graph G is a directed graph D where there is exactly one of (u, v) and (v, u) in D for each edge $\{u, v\}$ in G. Here, we abuse the term orientation to denote a graph that results from orienting only a subset of edges.

Definition 4. *A* dependency path *in an orientation* $D = (V, E_1 \cup E_2)$ *is a subgraph of* D *consisting of a sequence of vertices and edges* $v_1, e_1, v_2, e_2, \ldots, v_i$, $i \geq 3$, *satisfying:*
(1) for all $1 \leq j, k \leq i$, $v_j = v_k \Leftrightarrow j = k$,
(2) for all $1 \leq j \leq i - 1$, *either* $e_j = (v_j, v_{j+1}) \in E_1$ *or* $e_j = \{v_j, v_{j+1}\} \in E_2$,
(3) and for all $1 \leq j \leq i - 1$, *at least one of* e_j *and* e_{j+1} *is from* E_1.
The vertices v_1 *and* v_i *are called the* tail endpoint *and the* head endpoint *of the dependency path, respectively. A* dependency cycle *in an orientation is a dependency path with an edge between* v_1 *and* v_i *which can be undirected only if* $(v_1, v_2) \in E_1$ *and* $(v_{i-1}, v_i) \in E_1$; *otherwise, it is directed. A* directed path *is a dependency path with only directed edges.*

Observe that a dependency path contains at least one directed edge.

Definition 5. *A* valid orientation $D = (V, E_1 \cup E_2)$ *of an undirected graph* $G = (V, E)$ *is an orientation such that* $\forall v \in V : d^-(v) \leq 1$, $\forall v \in V : d^-(v) = 1 \Rightarrow d^+(v) \leq 1$, *and there is no dependency cycle in* D. *We call the set of vertices with* $d^-(v) = 0$ *the* origin *of* D.

See Figure 1 for an example of a valid orientation. Note that one can easily decide in $O(|E_1 \cup E_2|)$ time whether an orientation D is valid and, if so, find the origin of D. The following lemma is also easy to show.

Lemma 3. *Let* $D = (V, E_1 \cup E_2)$ *denote a valid orientation with origin* $O \subseteq V$.
(1) For each vertex $v \in (V \setminus O)$, *there is exactly one directed path from the vertices in* O *to* v.
(2) Two directed paths from O *to two vertices in* $V \setminus O$ *are vertex-disjoint except for their tail endpoints in* O.
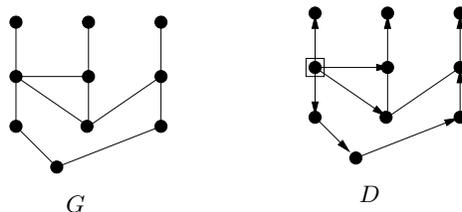
Fig. 1. An example of a valid orientation D of an undirected graph G. The origin O of D contains only one vertex, marked with a rectangular box.

The following orientation problem is a reformulation of PDS: Given an undirected graph $G = (V, E)$ and an integer $k \geq 0$, VALID ORIENTATION WITH MINIMUM ORIGIN (VOMO) asks whether there is a vertex subset $O \subseteq V$ with $|O| \leq k$ such that G has a valid orientation with O as origin.

Lemma 4. *An undirected graph G has a power dominating set P iff G has a valid orientation with P as origin.*

With VOMO and Lemma 4, we have an alternative formulation of PDS. The advantage of VOMO is that, by giving each undirected edge an orientation, a dependency cycle, which corresponds to a cycle of observation dependencies, can be easily detected in the dynamic programming on the tree decomposition.

4.2 Dynamic Programming on Tree Decompositions

Given an undirected, connected graph $G = (V, E)$ with $V := \{v_1, v_2, \ldots, v_n\}$ and a nice tree decomposition $\langle \{X_i \mid i \in V(T)\}, T \rangle$ of G with treewidth k. Let T_i denote the subtree of T rooted at node i and G_i denote the subgraph of G induced by the vertices in the bags in T_i, i.e., $G_i := G[\bigcup_{j \in V(T_i)} X_j]$. Furthermore, we use Y_i to denote $(\bigcup_{j \in V(T_i)} X_j) \setminus X_i$ and set $G_i' := G[X_i]$.

Definition of states: During the bottom-up process of our dynamic programming algorithm, it computes for a bag X_i the possible valid orientations of the subgraph G_i and stores the origin sizes of the valid orientations. Here, the valid orientations of G_i are characterized by the *bag states*. A bag state s is a combination of the states for all ordered pairs of vertices in X_i, the states of vertices in X_i, and the states of edges in G_i'. In the following, we will define the states for vertex pairs, for vertices, and for edges, respectively. Herein, we use $s(uv)$, $s(v)$, and $s(e)$ to denote the state of an ordered pair of vertices $v \in X_i$ and $u \in X_i$, the state of vertex $v \in X_i$, and the state of an edge $e \in E(G_i')$.

The decisive point in the dynamic programming is to detect the dependency cycles in the orientations D_i for G_i while reaching bag X_i. The dependency cycles inside suborientation $D_i[X_i]$ can be detected by exhaustive search in $D_i[X_i]$ which contains at most k vertices. However, the detection of dependency cycles over some vertices in Y_i depends on, for each pair of vertices u and v in X_i with $u \neq v$, the information about the dependency paths between u and v

in $D_i[Y_i \cup \{u, v\}]$. We solve this problem by defining for each pair of vertices in X_i several states reflecting the information about such dependency paths. Herein, let $V(p)$ be the set of the vertices on the dependency path p. We use p_{uv} to denote a dependency path with u and v as the tail and head endpoints, respectively. The path p_{uv} is called a dependency path *from u to v*.

Definition 6. *There are four types of dependency paths from u to v, p_{uv}, in an orientation D:*

Type 1: The first edge (between u and its successor on p_{uv}) as well as the last edge (between v and its predecessor on p_{uv}) of p_{uv} are directed edges.

Type 2: The first edge of p_{uv} is a directed edge but the last edge is not.

Type 3: The last edge of p_{uv} is a directed edge but the first edge is not.

Type 4: The last edge as well as the first edge of p_{uv} are undirected edges.

In addition, the function g maps a dependency path p to one of the four types, i.e., $g(p) \in \{$Type 1, Type 2, Type 3, Type 4$\}$.

Observe that, if there are two dependency paths p_{uv} and p_{vu} in a valid orientation D, then $g(p_{uv}) \neq$ Type 1 and $g(p_{vu}) \neq$ Type 1, and at least one of p_{uv} and p_{vu} is Type 4 or one is Type 2 and the other is Type 3; otherwise, there is a dependency cycle in the suborientation $D[V(p_{uv}) \cup V(p_{vu})]$ and we say that there is a "path type conflict." The states for an ordered vertex pair uv with $u \neq v$ in X_i are defined according to the possible types of dependency paths from u to v in $D_i[Y_i \cup \{u, v\}]$: there are 16 states for an ordered vertex pair uv according to the 16 possible subsets of $\{$Type 1, Type 2, Type 3, Type 4$\}$. With these vertex pair states, we can simply detect a dependency cycle at a bag X_i: check, for each ordered vertex pair uv with $u \in X_i$ and $v \in X_i$, whether there is a dependency path from v to u in $D_i[X_i]$ whose type together with one type in $s(uv)$ builds a path type conflict.

Next, we define five vertex states $s(v)$ for every vertex v in a bag X_i:

- $s(v) = 1$: there is exactly one directed edge from a vertex in Y_i to v and no directed edge from v to vertices in Y_i;
- $s(v) = 2$: there is exactly one directed edge from a vertex in Y_i to v and there is exactly one directed edge from v to a vertex in Y_i;
- $s(v) = 3$: there is no directed edge between v and the vertices in Y_i;
- $s(v) = 4$: there are at least two directed edges from v to the vertices in Y_i and no directed edge from the vertices in Y_i to v;
- $s(v) = 5$: there is exactly one directed edge from v to a vertex in Y_i and no directed edge from the vertices in Y_i to v.

Furthermore, we define three edge states $s(e)$ for the edges $e = \{u, v\}$ in G_i':

- $s(e) = uv$: edge e is directed from u to v;
- $s(e) = vu$: edge e is directed from v to u;
- $s(e) = \perp$: edge e is undirected.

As a consequence, for a bag X_i with $|X_i| \leq k$, we have at most $16^{k^2} \cdot 5^k \cdot 3^{k^2}$ bag states. In the following, we say that a valid orientation D is *under the restriction of a bag state s of the bag X_i* if D satisfies the following conditions:

– the orientations in D of the edges in G'_i coincide with their states in s;
– for each ordered vertex pair uv with $u \in X_i$ and $v \in X_i$, the types of the dependency paths from u to v in $D[Y_i \cup \{u, v\}]$ coincide with $s(uv)$; and
– for each vertex $v \in X_i$, the orientations of the edges between v and vertices in Y_i coincide with $s(v)$.

In the bottom-up dynamic programming, we use a mapping A_i for each bag X_i, which stores, for each bag state, the minimum size of the origins of all possible valid orientations of G_i under the restriction of the bag state. Due to the following easy-to-prove lemma, in the computation of the values for A_i, we do not count vertices v for the size of an origin with $d^-(v) = 0$ and $d^+(v) \leq 1$.

Lemma 5. *For a connected, undirected graph $G = (V, E)$, there is always a valid orientation with origin $O \subseteq V$ such that each $v \in O$ has at least two neighbors in $V \setminus O$ which are not neighbors of other vertices in O.*

In the following, we use valid(G'_i, s_i) to denote the procedure deciding whether the edge states $s_i(e)$ of the edges e in G'_i form a valid orientation of G'_i. Since G'_i contains at most k vertices, valid(G'_i, s_i) needs at most $O(k!)$ time by enumerating all cycles in G'_i. Procedure valid(G'_i, s_i) returns true if s_i implies a valid orientation of G'_i; otherwise, it returns false.

Initialization: For each leaf node i with bag X_i of the tree decomposition T, we initialize A_i as follows. For each bag state s_i, we set $A_i(s_i) := +\infty$ if

$$(\exists v \in X_i : s_i(v) \neq 3) \vee (\exists uv : u \in X_i \wedge v \in X_i \wedge s_i(uv) \neq \emptyset) \vee (\text{valid}(G'_i, s_i) = \text{false});$$

and, otherwise,

$$A_i(s_i) := |\{v \in X_i \mid \exists e = \{v, u\} \in E(G'_i) : \exists e' = \{v, w\} \in E(G'_i) :$$
$$u \neq w \wedge s_i(e) = vu \wedge s_i(e') = vw\}|.$$

Updating process: After the initialization, we visit the bags of our tree decomposition bottom-up from the leaves to the root, evaluating the corresponding mappings in each step. For each bag state s_i of a bag X_i, do the following:

– Check whether the edge states contained in s_i form a valid orientation of G'_i.
– Compute the set S^{s_i} containing the "compatible bag states" s_j (or "compatible bag state pairs" for join nodes) of the child bag X_j. The formal definition of compatible bag states (and compatible bag state pairs) will be given individually for forget, insert, and join nodes.
– For each compatible bag state s_j (or each compatible bag state pair) in S^{s_i}, check whether a valid orientation for G_j under the restriction of s_j is a valid orientation for G_i under the restriction of s_i.
– Based on the mappings A_j for all compatible bag states (or bag state pairs) in S^{s_i}, evaluate $A_i(s_i)$.

Due to the lack of space, we only treat the (most interesting) case of "Insert Nodes" of the tree decomposition here.

Insert Nodes: Suppose that node i is an insert node with child node j, $X_i := \{x_{j_1}, \ldots, x_{j_{n_j}}, x\}$ and $X_j := \{x_{j_1}, \ldots, x_{j_{n_j}}\}$.

Procedure valid(G'_i, s_i) can decide whether the edge states contained in s_i form a valid orientation of G'_i in $O(k!)$ time by enumerating all cycles. For each bag state s_i of node i, we consider the set S^{s_i} containing the compatible bag states s_j of node j which satisfy

$$(\forall e \in E(G'_j) : s_i(e) = s_j(e)) \wedge (\forall v : s_i(v) = s_j(v)) \wedge (\forall uv : s_i(uv) = s_j(uv)),$$

where $u, v \in X_j$.

Note that the introduction of a new vertex x does not change the number of directed edges between a vertex $v \in X_j$ and the vertices in Y_i; thus, $s_i(v) = s_j(v)$. The types of the dependency paths remain the same for each ordered vertex pair uv with $u \in X_j$ and $v \in X_j$, i.e., $s_i(uv) = s_j(uv)$. Moreover, since there is no edge between the new vertex x in X_i and the vertices in Y_i, we set $S^{s_i} := \emptyset$ for bag states s_i if $s_i(x) \neq 3$ or if there is a vertex $v \in X_j$ with $s_i(vx) \neq \emptyset$ or $s_i(xv) \neq \emptyset$.

The most difficult task here is to decide whether a valid orientation of G_j under the restriction of a bag state in S^{s_i} is still a valid orientation of G_i under the restriction of s_i. Herein, we have to take into account the states of the edges incident to x in G'_i. Observe that the conditions for a valid orientation can be violated either by a vertex in X_i having x as neighbor or by a dependency cycle over vertex x. Thus, based on the information saved in $s_j(v)$ for vertices $v \in X_j$ and in $s_j(uv)$ for the ordered vertex pairs with $u \in X_j$ and $v \in X_j$, this task can be fulfilled by firstly checking for each vertex v in X_i having x as neighbor whether v violates the vertex-degree conditions of a valid orientation, i.e., $d^-(v) \leq 1$ and $d^-(v) = 1 \Rightarrow d^+(v) \leq 1$, by taking $s_j(v)$ and $s_i(e)$ for $e = \{v, x\}$ into account. Secondly, check for every two vertices u and v in X_i whether they now have a dependency path over x in $D_i[X_i]$ built by the edge states in s_i whose type together with one path type in $s_j(uv)$ or $s_j(vu)$ forms a path type conflict. It is easy to see that these two checks can be done in $O(k!)$ time by enumerating all dependency paths in $D_i[X_i]$. Finally, if for a bag state s_i there is no s_j in S^{s_i} which passes both checks, then $S^{s_i} := \emptyset$.

Evaluate the mapping A_i of X_i as follows: for each bag state s_i, set

$$A_i(s_i) := \min_{s_j \in S^{s_i}} \{A_j(s_j) + f_{s_i}(x) + |B|\},$$

where $f_{s_i}(x) = 1$ if there exist at least two edges $e = \{u, x\}$ and $e' = \{v, x\}$ in G'_i with $s_i(e) = xu$ and $s_i(e') = xv$; otherwise, $f_{s_i}(x) = 0$. The set B contains the vertices $v \in X_j$ which either have $s_j(v) = 5$ and exactly one edge $e = \{v, x\}$ in G'_i with $s_i(e) = vx$ or have $s_j(v) = 3$ and exactly two edges $e = \{v, x\}$ and $e' = \{u, v\}$ in G'_i with $s_i(e) = vx$ and $s_i(e') = vu$. Roughly speaking, $f_{s_i}(x) = 1$ covers the case that x has to be added to the origin and the set B contains the vertices which are in the origin of the valid orientation of G_i but not in the origin of the valid orientation of G_j. Note that, if $S^{s_i} = \emptyset$ for a s_i, then $A_i(s_i) = +\infty$.

When reaching the root of T, we can determine the minimum size of the origin of a valid orientation of G in the mappings A of the root node. Together with Lemma 4, we obtain our main result.

Theorem 3. *For an n-vertex graph with given width-k tree decomposition,* POWER DOMINATING SET *can be solved in $O(c^{k^{2}} \cdot n)$ time for a constant c.*

5 Conclusion

There are several avenues for future work. Table 1 contains several empty entries concerning the complexity of POWER DOMINATING SET in particular graph classes we did not address but which have been addressed for DOMINATING SET [11]. In particular, is there a "significant" difference between the complexities of DS and PDS? How do fixed-parameter tractability results for DS in planar graphs transfer to PDS? Are there nontrivial data reduction rules for PDS? So far, we are not aware of a graph class where DS is polynomial-time solvable and PDS is NP-complete or vice versa.

References

1. J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for Dominating Set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.
2. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation — Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.
3. H. L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proc. 22nd MFCS*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.
4. A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes: a Survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.
5. E. D. Demaine and M. Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs. In *Proc. 16th SODA*, pages 590–601, 2005.
6. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
7. U. Feige. A threshold of $\ln n$ for approximating Set Cover. *Journal of the ACM*, 45(4):634–652, 1998.
8. T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi, and M. A. Henning. Domination in graphs: applied to electric power networks. *SIAM Journal on Discrete Mathematics*, 15(4):519–529, 2002.
9. T. Kloks. *Treewidth: Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.
10. J. Kneis, D. Mölle, S. Richter, and P. Rossmanith. Parameterized power domination complexity. Technical Report AIB-2004-09, Department of Computer Science, RWTH Aachen, Dec. 2004.
11. D. Kratsch. Algorithms. In T. W. Haynes, S. T. Hedetniemi, and P. J. Slater, editors, *Domination in Graphs: Advanced Topics*, pages 191–231. Marcel Dekker, 1998.
12. J. A. Telle and A. Proskurowski. Practical algorithms on partial k-trees with an application to domination-like problems. In *Proc. 3rd WADS*, volume 709 of *LNCS*, pages 610–621. Springer, 1993.