Information Processing Letters 116 (2016) 590–594

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

An FPT-algorithm for modifying a graph of bounded treewidth to decrease the size of its dominating set using minimum modification ☆

Mehdy Roayaei [a], Mohammadreza Razzazi [a,b,*]

[a] Department of Computer Engineering and Information Technology, AmirKabir University of Technology, Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran

ARTICLE INFO

Article history:
Received 26 October 2015
Received in revised form 2 April 2016
Accepted 2 April 2016
Available online 7 April 2016
Communicated by M. Chrobak

Keywords:
Dominating set
Fixed-parameter tractability
Tree decomposition
Computational Complexity

ABSTRACT

In this paper, we study the problem of modifying a graph such that the resulting graph has a dominating set of size at most k. Solving this problem determines the minimum number of edges to be added to a given graph such that at most k vertices can dominate all vertices. We show that this problem is NP-hard, and therefore, has no polynomial-time algorithm (unless $P = NP$). Also, we show that the problem is FPT parameterized by the treewidth of the input graph. Furthermore, we show that the problem parameterized by k is W[2]-hard and belongs to W[P].

1. Introduction

One of the most well-known problems in networks, especially in computer networks, is Minimum Dominating Set: Given an undirected graph $G = (V, E)$, a dominating set is a subset S of its vertices such that for all vertices $v \in V$, either $v \in S$ or it has a neighbor in S. The problem of computing a dominating set of minimum size is called Minimum Dominating Set. The number of vertices in the smallest dominating set for G is called the domination number, $\gamma(G)$.

Although Dominating Set is W[2]-complete with respect to the parameter domination number [5], it is FPT with respect to the parameter treewidth [15]. Also, it has been shown that Dominating Set parameterized by the domination number is FPT for graphs with bounded degeneracy [14] and planar graphs [7].

This problem has diverse applications in the real world, including dominating queens, set of representatives, bus routing, computer communication networks, radio stations, social network theory, land surveying, and kernels which have been illustrated in [1] and [2]. In most of the applications, by computing a dominating set, we are actually locating our resources such as the server computers in a computer network or bus stops in a transportation network; so, minimizing the size of the dominating set is directly related to minimizing the cost of the required resources on the dominating nodes.

In this paper, we consider the situation in which there is some threshold on the number of our resources; therefore, all vertices must be dominated using at most k vertices. We call it *Domination Number Improvement*. The solution of this problem determines the minimum number of edges to be added to a given graph so that at most k vertices can dominate all vertices. Domination Number Improvement is interesting when there is some threshold

☆ This research was in part supported by a grant from School of Computer Science, Institute for Research in Fundamental Sciences (IPM) (No. CS1393-8-1).

* Corresponding author at: P.O. Box 15875-4413, #424 Hafez Avenue, AmirKabir University of Technology, Tehran, Iran. Tel.: +98 2164542732.
E-mail addresses: mroayaei@aut.ac.ir (M. Roayaei), razzazi@aut.ac.ir (M. Razzazi).

http://dx.doi.org/10.1016/j.ipl.2016.04.002
0020-0190/

on the size of the dominating set because of a limitation on the cost or on the number of available resources [12] or when adding new edges between vertices of a graph is less expensive than adding new resources [13].

Definition 1 *(Domination Number Improvement).* Given an undirected graph $G = (V, E)$ and an integer k, find the minimum number of edges, the set $\hat{E}$, such that the graph $\hat{G} = (V, E \cup \hat{E})$ has domination number $\gamma(\hat{G}) = k$.

We show that this problem is NP-hard. Furthermore, we present an algorithm that runs in polynomial time for fixed treewidth of the input graph. More precisely, we show that Domination Number Improvement is FPT parameterized by the treewidth of the input graph. Also, we show that Domination Number Improvement parameterized by k is W[2]-hard and belongs to W[P]. W[P] is the class of all parameterized problems that can be reduced to Weighted Circuit Satisfiability by a parameterized reduction [17].

2. Main algorithm

We show that Domination Number Improvement can be reduced to k-Max Dominating Set [3] and vice versa. It is known that k-Max Dominating Set is NP-hard and it cannot be approximated with a ratio better than $1 - \frac{1}{e}$ [3]. Also, it is known that k-Max Dominating Set parameterized by the maximum number of dominated vertices is FPT [16].

Definition 2 *(k-Max Dominating Set).* Given an undirected graph $G = (V, E)$ and an integer k, find a subset $D \subseteq V$ with $|D| \leq k$, that maximizes the number of dominated vertices.

k-Max Dominating Set has applications in different networks, such as sensor networks, where bandwidth constraints limit the number of sensors we can choose to k [10], or social networks, where the goal is to find k vertices having maximum total influence in a social network [11].

Theorem 1. *An undirected graph $G = (V, E)$ and an integer k are given. There is a solution of size $n - d$ for k-Max Dominating Set, if and only if there is a solution of size d for Domination Number Improvement.*

Proof. It is clear that any instance for one problem is also an instance for the other. First, suppose that the subset $D \subseteq V$ with $|D| \leq k$ is a solution for k-Max Dominating Set that dominates $n - d$ vertices. For each non-dominated vertex, add an edge connecting it to a vertex in D. Clearly, the resulting graph has the domination number of size at most k. Therefore, the set of d new edges, namely S_d, is a solution for Domination Number Improvement.

Second, assume that the set S_d with $|S_d| = d$ is a solution for Domination Number Improvement. That is, adding the set S_d to the graph G converts it to the graph $\hat{G} = (V, E \cup S_d)$, which has a dominating set D of size at most k. Removing each edge of S_d from $\hat{G}$ can convert at most one dominated vertex to a non-dominated vertex. Therefore, after removing S_d from $\hat{G}$, the set D can dominate at least $n - d$ vertices. □

Corollary 1. *Domination Number Improvement is NP-hard.*

Corollary 2. *Any optimal solution of k-Max Dominating Set can be translated to an optimal solution of Domination Number Improvement in linear time.*

Typically, algorithms based on the tree decomposition technique consist of two major steps. The first step is responsible for finding a tree decomposition of bounded width (fixed treewidth) of the input graph, and the second step is responsible for solving the problem on that tree decomposition usually using the dynamic programming approach [4].

Definition 3 *(Tree decomposition).* Let $G = (V, E)$ be a graph. A *tree decomposition* of G is a pair $\langle \{X_i | i \in I\}, T \rangle$, where each X_i is a subset of V called a *bag*, and T is a tree with the elements of I as vertices. The following three properties must hold:

1. $\cup_{i \in I} X_i = V$;
2. for every edge $\{u, v\} \in E$, there is an $i \in I$ such that $\{u, v\} \subseteq X_i$; and
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 $\langle \{X_i | i \in I\}, T \rangle$ equals $max\{|X_i| : i \in I\} - 1$. The *treewidth* of G is the minimum ω such that G has a tree decomposition of width ω.

The problem of deciding whether a graph has treewidth at most ω and then finding its corresponding tree decomposition is NP-complete for general graphs [5]. However, this problem is FPT parameterized by the treewidth of the input graph, and there is an algorithm to find the tree decompositions of *small treewidth*, which runs in linear time for fixed treewidth of the input graph [6].

From now on, assume that a tree decomposition with the treewidth ω is given. Also, k is the upper bound for the size of the dominating set.

To reduce the time complexity of our algorithm, we take advantage of a special type of tree decomposition. The tree decomposition of a graph is not unique and there may be more than one tree decomposition for a graph, each of which has different properties. A special tree decomposition that has a simple structure but attractive properties is called *nice tree decomposition* [7]:

Definition 4 *(Nice tree decomposition).* A tree decomposition $\langle \{X_i | i \in I\}, T \rangle$ is a *nice tree decomposition* if it satisfies the following properties:

1. Every bag of the tree T has at most two children.
2. If a bag i has two children j and k, then $X_i = X_j = X_k$. X_i is called a *join bag*.
3. If a bag i has one child j, then either
   (a) $|X_i| = |X_j| + 1$ and $X_j \subset X_i$, X_i is called an *insert bag*, or
   (b) $|X_i| = |X_j| - 1$ and $X_i \subset X_j$, X_i is called a *forget bag*.

Obviously, processing a nice tree decomposition is much simpler than an arbitrary tree decomposition. Fortunately, there is a useful result that makes it possible to transform an arbitrary tree decomposition to a nice tree decomposition [8]:

A tree decomposition of a graph G with the width ω and $O(n)$ bags is given, where n is the number of vertices of G. A nice tree decomposition of G that also has the width ω and $O(n)$ bags can be found in $O(n)$ time.

Since the number of bags in a tree decomposition is linear with respect to the vertices of the graph [9], we can assume that a nice tree decomposition of width ω which has $|I|$ bags is given as the input.

Let $X = \langle \{X_i | i \in I\}, T \rangle$ be a nice tree decomposition for the graph $G = (V, E)$. In every possible solution of k-Max Dominating Set, the vertices are classified into three disjoint classes. We assign different values to the vertices of each class; 0 to the vertices that have not been dominated, 1 to the vertices that have been dominated by other vertices (dominated vertices), and 2 to the vertices that are members of the dominating set (dominating vertices).

Using these values, any assignment to the graph vertices can be a possible solution for k-Max Dominating Set. Let $V = \{x_1, \ldots, x_n\}$. Also, assume that the vertices in the bags are ordered increasingly, i.e., $X_i = (x_{i_1}, \ldots, x_{i_{n_i}})$ with $i_1 \leq \ldots \leq i_{n_i}$. Therefore, the vector $C = (c_1, \ldots, c_{n_i}) \in \{0, 1, 2\}^{n_i}$ is an assignment for X_i.

Clearly, some of these assignments may not be valid and must be discarded. The causes of the invalidities may be different; therefore, they can be classified into different categories. In what follows, $\#_d(C) = |\{t \in \{1, \ldots, n_i\} : c_t = d\}|$ returns the number of vertices of value d in the assignment C and $\#_{d,r}(C) = |\{t \in \{1, \ldots, n_i\} : c_t = d \vee c_t = r\}|$ returns the number of vertices of value d or r in the assignment C.

- **Type 1**: An assignment $C \in \{0, 1, 2\}^{n_i}$ is invalid of type 1, if $(\exists s \in \{1, \ldots, n_i\} : c_s = 0) \wedge (\exists t \in \{1, \ldots, n_i\} : (x_{i_t} \in N(x_{i_s}) \wedge c_t = 2))$. In this case, the value of x_{i_s} is invalid according to the value of its neighbor.
- **Type 2**: An assignment $C \in \{0, 1, 2\}^{n_i}$ is invalid of type 2, if $\#_2(C) > k$. In this case, there cannot be any solution containing assignment C, because its domination number will be larger than the threshold k.
- **Type 3**: An assignment $C \in \{0, 1, 2\}^{n_i}$ is invalid of type 3, if $(\exists s \in \{1, \ldots, n_i\} : c_s = 1) \wedge (\nexists t \in \{1, \ldots, n_i\} : (x_{i_t} \in N(x_{i_s}) \wedge c_t = 2))$. In this case, the value of x_{i_s} is invalid in the bag X_i, but not necessarily in the whole tree.
- **Type 4**: An assignment $C \in \{0, 1, 2\}^{n_i}$ is invalid of type 4 for the threshold p, if $\#_2(C) > p$. In this case, p may be smaller than k.

For each bag $X_i = \left(x_{i_1}, \ldots, x_{i_{n_i}}\right)$, consider the table T_i in the following format:

$$T_i = \begin{bmatrix} d_i(C_1^i, 0) & d_i(C_1^i, 1) & \cdots & d_i(C_1^i, k) \\ d_i(C_2^i, 0) & d_i(C_2^i, 1) & \cdots & d_i(C_2^i, k) \\ \vdots & \vdots & \vdots & \\ d_i(C_{3^{n_i}}^i, 0) & d_i(C_{3^{n_i}}^i, 1) & \cdots & d_i(C_{3^{n_i}}^i, k) \end{bmatrix}$$

Each row of the table T_i corresponds to an assignment for the vertices in X_i. $d_i(C_l^i, p)$ stores the maximum number of dominated and dominating vertices of the graph visited up to the current step of the algorithm under the restriction of using the l-th assignment C_l^i of 3^{n_i} possibilities for the bag X_i and the threshold p for the size of the dominating set.

Initialization. For each assignment C and threshold p for a *leaf bag* X_i, $d_i(C, p)$ is initialized as follows. It takes $O(3^\omega \cdot (k + \omega^2))$ time to initialize each leaf bag.

$$d_i(C, p) = \begin{cases} -\infty & C \text{ is invalid of type 2, 3, 4} \\ \#_{1,2}(C) & \text{o.w.} \end{cases} \quad (1)$$

Note that invalid assignments of type 1 are not considered here and are treated the same as valid assignments. Since the problem is a maximization problem, for every assignment C, which is invalid of type 1, there is always another assignment C', in which for each $t \in \{1, \cdots, n_i\}$: $c_t = 0$ that causes the invalidity of type 1 in C, $c'_t = 1$. C' is not invalid of type 1 and certainly dominates more vertices than C using the same number of dominating vertices. Thus, in a comparison between C' and C, C' will be selected as a better assignment.

Recursion. In the recursion step, the table of each bag is updated using the tables of its children. Also, for each assignment, its corresponding row in a table is populated form left to right. Depending on the type of the bag being updated, the recursion rule will be different:

- **Forget bag:** Suppose i is a forget bag with child bag j. Without loss of generality, assume that $X_i = \left(x_{i_1}, \cdots, x_{i_{n_i}}\right)$ and $X_j = \left(x_{i_1}, \cdots, x_{i_{n_i}}, x\right)$. For all $C \in \{0, 1, 2\}^{n_i}$ and $p \in \{0, 1, \cdots, k\}$, $d_i(C, p)$ is updated as follows. It takes $O(3^\omega \cdot k)$ time for each bag.

$$d_i(C, p) = max_{d \in \{0,1,2\}} \{d_j(C \times d, p)\} \quad (2)$$

  The symbol $\times$ in $d_j(C \times d, p)$ denotes appending value d to an assignment C.
- **Insert bag:** Suppose i is an insert bag with child bag j. Again, without loss of generality, assume that $X_i = \left(x_{i_1}, \cdots, x_{i_{n_j}}, x\right)$ and $X_j = \left(x_{i_1}, \cdots, x_{i_{n_j}}\right)$. Depending on the value of the vertex x, $d_i(C, p)$ is updated for all C and p as follows:

$$d_i(C \times 0, p) = d_j(C, p) \quad (3)$$

$$d_i(C \times 1, p) = \begin{cases} d_j(C, p) + 1 & \exists x_{j_q} \in (N(x) \cap X_j) : c_q = 2 \\ -\infty & \text{o.w.} \end{cases} \quad (4)$$

  Let $\phi_j(x) = \{x_{j_t} : x_{j_t} \in (N(x) \cap X_j) \wedge c_t = 1\}$. Then, the function $\phi(C) = C' = \left(c'_1, c'_2, \cdots, c'_{n_j}\right)$ is defined as follows:

$$c'_t = \begin{cases} 0 & \text{if } x_{j_t} \in \phi_j(x) \\ c_t & \text{o.w.} \end{cases}$$

$$d_i(C \times 2, p) = \begin{cases} -\infty & \#_2(C) \geq p \\ d_j(C', p-1) + |\phi_j(x)| + 1 & \text{o.w.} \end{cases} \quad (5)$$

It takes $O(3^\omega \cdot (k+w))$ time for each bag.

- **Join bag:** Suppose i is a join bag with two children j and k such that $X_i = X_j = X_k = \left(x_{i_1}, \cdots, x_{i_{n_i}}\right)$. Let $C = \left(c_1, \cdots, c_{n_i}\right) \in \{0,1,2\}^{n_i}$ be an assignment for X_i. Then, $d_i(C, p)$ is updated for all C and p as follows:

$$d_i(C, p) = max\{d_j(C', k_1) + d_k(C'', k_2)\} - \#_2(C), \quad (6)$$

where $k_1 + k_2 - \#_2(C) = p$ and $k_1, k_2 \geq \#_2(C)$, and $C' = \left(c'_1, \cdots, c'_{n_i}\right), C'' = \left(c''_1, \cdots, c''_{n_i}\right) \in \{0,1,2\}^{n_i}$, such that:
– $c_t \in \{0,2\} \Longrightarrow c'_t = c''_t = c_t$, and
– $c_t = 1 \Longrightarrow c'_t, c''_t \in \{0,1\} \wedge c'_t \neq c''_t$.
The idea behind (6) is that it is not necessary for $x \in X_i$ to be dominated by both assignments C' and C'', but one of them is sufficient.
There are $2^{\#_1(C)}$ pairs (C', C'') that divide C, and $2^{n_i - q}\binom{n_i}{q}$ assignments C with $\#_1(C) = q$. Also, there are p pairs (k_1, k_2) that divide k. Thus, updating the whole table T_i takes $\Sigma_{p=0}^{k} \Sigma_{q=0}^{n_i} \binom{n_i}{q} \cdot 2^{n_i - q} \cdot p \cdot 2^q = O(k^2 \cdot 4^\omega)$ time.

Lemma 1. *Recursive formulas (2)–(6) assign the correct values to the variable $d_i(C, p)$.*

Proof. In the case of a forget bag, the recursive formula (2), no new vertex is added to the parent bag; thus, no new invalidity can emerge and no non-dominated vertex can be dominated.

In the case of an insert bag x, different scenarios occur depending on the value of x. If $x = 0$, the recursive formula (3), only the invalidity of type 1 may occur, which is treated as a valid assignment. Since the new vertex has not been dominated, the same value as child j is assigned to $d_i(C \times 0, p)$. When $x = 1$, the recursive formula (4), invalidity of type 3 may occur if $\nexists x_{j_q} \in (N(x) \cap X_j) : c_q = 2$, which is handled explicitly by (4). Otherwise, the number of dominated vertices in child j is added by 1. When $x = 2$, the recursive formula (5), invalidities of type 1, 2 or 4 may occur and invalidity of type 3 may disappear. As in the previous steps, the invalidity of type 1 is not considered. The invalidities of type 2 and 4 are separately handled in (5). In the case of the invalidity of type 3, an invalid assignment that contains a vertex $x_{j_s} : c_s = 1 \wedge (\nexists x_{j_t} : x_{j_t} \in (N(x_{j_s}) \cap X_j) \wedge c_t = 2)$ becomes valid if $x \in N(x_{j_s})$. To handle such a situation, we have replaced the assignment C by the assignment C'. Thus, it is necessary to add the number of vertices whose values have been changed, that is $|\phi_j(x)|$, to the number of vertices that have been dominated by the assignment C'.

In the case of a join bag, there can be three different values for $x_{i_s} \in X_i$. If $x_{i_s} = 0$ or $x_{i_s} = 2$, the assignments C' and C'' must also assign the same value to the variable x_{i_s}. However, in the case of $x_{i_s} = 1$, it is sufficient for x_{i_s} to be dominated by either C' or C''. This is where we may use the value corresponding to an invalid assignment of type 1. By changing the value of a vertex to 0 in a child's bag, its assignment may be converted to an invalid assignment of type 1; however, by treating it as a valid assignment in the initialization step, this situation is handled. Since each vertex of value 1 in C is repeated either in the assignment C' or C'', it is sufficient to subtract $\#_2(C)$ from the sum of $d_j(C', k_1) + d_k(C'', k_2)$. □

Extraction of the optimal solution. Let r be the root of T. Then $max_{1 \leq l \leq 3^{n_r} : c_l = \{0,1,2\}^{n_r}} \{d_r(C_l, k)\}$ will be the optimal value of k-Max Dominating Set, i.e. the maximum number of vertices that can be dominated in a graph G using at most k dominating vertices. Also, by recording the selected assignments of child bags next to each assignment of their parent, we can construct the dominating set D of the optimal value. This step of the algorithm takes $O(3^\omega + |I| \cdot w)$ time.

Theorem 2. *Domination Number Improvement is FPT parameterized by the treewidth of the input graph.*

Proof. Given a graph $G = (V, E)$ and its tree decomposition, which has width ω and $|I|$ bags, we have presented an algorithm to find a subset $D \subseteq V$ with $|D| \leq k$, that maximizes the number of dominated vertices. According to Lemma 1 and the recursive formulas (2)–(6), the assignment C_l in $max_{1 \leq l \leq 3^{n_r} : c_l = \{0,1,2\}^{n_r}} \{d_r(C_l, k)\}$ is an optimal solution for k-Max Dominating Set. Considering the complexity of each step of the algorithm, it is not hard to see that the time complexity of the algorithm is $O((k^2 + \omega^2) \cdot 4^\omega \cdot |I|)$, and since $|I| = O(n)$, is $O((k^2 + \omega^2) \cdot 4^\omega \cdot n)$. Using Corollary 2 of Theorem 1, we can find the minimum number of edges such that adding them to the graph G improves its domination number to the value k in linear time. Since the running time is polynomially bounded in k, which in turn is upper-bounded by n, hence, Domination Number Improvement is FPT parameterized by the treewidth of the input graph. □

Theorem 3. *Domination Number Improvement parameterized by k is W[2]-hard and belongs to W[P].*

Proof. To show that it is in W[P], we reduce k-Max Dominating Set to k-Weighted Circuit Satisfiability [17]: Given a graph $G(V, E)$ and integers k and t, we create a decision circuit C, such that there is a dominating set of size k in G that dominates t vertices if and only if C have a weight k satisfying assignment. For each vertex $x_j \in V$, create an input in_j for C, which is assigned 1 if the vertex x_j belongs to the dominating set, 0 otherwise.

Before creating C, we create a circuit C_n with n inputs and $n+1$ outputs $o_0, o_1, \cdots, o_n$, such that o_i is 1 if exactly i of the inputs are 1. We recursively create C_n using C_{n-1} with n outputs $y_0, y_1, \cdots, y_{n-1}$ as shown in Fig. 1. Then, for each vertex $x_j \in V$, consider an OR gate which as input receives the inputs corresponding to the adjacent vertices of x_j. Connect the output of this OR gate to the input i_j of C_n. Furthermore, connect o_t to the output of C. Obviously, the resulting circuit C has polynomial size and has a weight k assignment if and only if k-Max Dominating Set

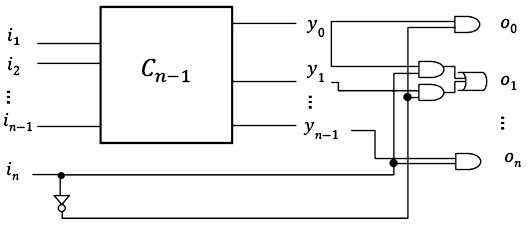

Fig. 1. Illustration of reduction to k-Weighted Circuit Satisfiability.

has a dominating set of size k that dominates t vertices. Hence, k-Max Dominating Set, and thus Domination Number Improvement are in W[P].

To show W[2]-hardness, we reduce from Dominating Set, which is W[2]-complete with respect to k. Let G, k be an instance for Dominating Set. We create an instance G, k with 0 as the size of the solution for Domination Number Improvement. Obviously, if there is a solution of size 0 for Domination Number Improvement, there is a solution for Dominating Set and vice versa. □

Theorem 4. *Domination Number Improvement on planar graphs is FPT parameterized by the domination number of the input graph.*

Proof. Let G be the input graph. According to Theorem 2, Domination Number Improvement can be solved in $O((k^2+\omega^2)\cdot 4^{\omega}\cdot n)$ time. Additionally, we know that the treewidth of a planar graph with domination number $\gamma(G)$ is $O(\sqrt{\gamma(G)})$; and that such a tree decomposition can be found in $O(\sqrt{\gamma(G)}\cdot n)$ time [7]. Also, it is clear that $k \leq \gamma(G)$. □

3. Conclusion

In this paper, we showed that Domination Number Improvement is NP-hard. Then, we showed that this problem parameterized by the treewidth of the input graph is FPT. Also, we studied the parameterized complexity of this problem with respect to the threshold on the domination number of the resulting graph, and showed that Domination Number Improvement parameterized by k is W[2]-hard and belongs to W[P]. We remark that, according to Theorem 1, all results also hold for k-Max Dominating Set.

References

[1] T.W. Haynes, S. Hedetniemi, P.J. Slater, Fundamentals of Domination in Graphs, CRC Press, 1998.
[2] E. Carrizosa, J.B.G. Frenk, Dominating sets for convex functions with some applications, J. Optim. Theory Appl. 96 (1998) 281–295.
[3] E. Miyano, H. Ono, Maximum domination problem, in: Australas. Theory Symp. (CATS 2011), 2011.
[4] H.L. Bodlaender, Treewidth: algorithmic techniques and results, in: 22nd MFCS, 1997, pp. 19–36.
[5] R. Niedermeier, Invitation to Fixed Parameter Algorithms, Oxford University Press, 2006.
[6] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, SIAM J. Comput. 25 (1996) 1305–1317.
[7] J. Alber, H.L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, Algorithmica 33 (2002) 461–493.
[8] T. Kloks, Treewidth: Computations and Approximations, Lect. Notes Comput. Sci., 1994.
[9] E. Paisios, Generic Programing for Graph Problems Using Tree Decompositions, Vienna University of Technology, 2008.
[10] M.X. Cheng, L. Ruan, W. Wu, Achieving minimum coverage breach under bandwidth constraints in wireless sensor networks, in: INFOCOM 2005, 24th Annu. Jt. Conf. IEEE Comput. Commun. Soc., Proc. IEEE, 2005, pp. 2638–2645.
[11] K. Avrachenkov, P. Basu, G. Neglia, B. Ribeiro, D. Towsley, Online myopic network covering, CoRR, arXiv:1212.5035 [cs.SI], 2012.
[12] M.X. Cheng, L. Ruan, W. Wu, Coverage breach problems in bandwidth-constrained sensor networks, ACM Trans. Sens. Netw. 3 (2007) 12.
[13] M. Papagelis, F. Bonchi, A. Gionis, Suggesting ghost edges for a smaller world, in: Proc. 20th ACM Int. Conf. Inf. Knowl. Manag, ACM Press, New York, NY, USA, 2011, pp. 2305–2308.
[14] G. Philip, V. Raman, S. Sikdar, Solving dominating set in larger classes of graphs: FPT algorithms and polynomial kernels, in: Algorithms – ESA 2009, Springer, 2009, pp. 694–705.
[15] J.M.M. Van Rooij, H.L. Bodlaender, P. Rossmanith, Dynamic programming on tree decompositions using generalised fast subset convolution, in: Algorithms – ESA 2009, Springer, 2009, pp. 566–577.
[16] J. Kneis, D. Mölle, P. Rossmanith, Partial vs. complete domination: t-dominating set, in: SOFSEM 2007 Theory Pract. Comput. Sci., Springer, 2007, pp. 367–376.
[17] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Springer, 2013.