Theoretical Computer Science 511 (2013) 147–158

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

New parameterized algorithms for the edge dominating set problem

Mingyu Xiao [a,*], Ton Kloks [b], Sheung-Hung Poon [b]

[a] School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610054, China
[b] Department of Computer Science, National Tsing Hua University, Taiwan

ARTICLE INFO

Keywords:
Graph algorithms
Parameterized algorithms
Kernelization
Edge dominating set

ABSTRACT

An edge dominating set of a graph $G = (V, E)$ is a subset $M \subseteq E$ of edges in the graph such that each edge in $E - M$ is incident with at least one edge in M. In an instance of the parameterized edge dominating set problem, we are given a graph $G = (V, E)$ and an integer k, and we are asked to decide whether G has an edge dominating set of size at most k. In this paper, we show that the parameterized edge dominating set problem can be solved in $O^*(2.3147^k)$ time and polynomial space. We also show that this problem can be reduced to a quadratic kernel with $O(k^3)$ edges.

1. Introduction

The edge dominating set (EDS) problem, to find an edge dominating set of minimum size in a graph, is one of the basic problems highlighted by Garey and Johnson in their work on NP-completeness [6]. It is known that the problem is NP-hard even when the graph is restricted to planar or bipartite graphs of maximal degree 3 [17]. The problem in general graphs and in sparse graphs has been extensively studied in approximation algorithms [17,5,2]. Note that any maximal matching is a 2-approximation for the EDS problem. The 2-approximation algorithm for the weighted version of the EDS problem is considerably more complicated [5].

Recently, the EDS problem has drawn much attention from the exact and parameterized algorithms community. Randerath and Schiermeyer [10] designed an $O^*(1.4423^m)$ algorithm for the EDS problem, which was improved to $O^*(1.4423^n)$ by Raman et al. [11]. Here, we use the O^*-notation, which suppresses polynomial factors. We use n and m respectively as the number of vertices and number of edges in the graph.

Fomin et al. [4] improved Raman's result to $O^*(1.4082^n)$ by considering the treewidth of the graph. Van Rooij and Bodlaender [12] designed an $O^*(1.3226^n)$ algorithm by using the 'measure and conquer method'. Currently, the best result is the $O^*(1.3160^n)$ algorithm of Xiao and Nagamochi [16].

For the parameterized edge dominating set (PEDS) problem, with the parameter k being the size of the edge dominating set, Fernau [3] gives an $O^*(2.6181^k)$ algorithm. Fomin et al. [4] obtain an $O^*(2.4181^k)$-time and exponential-space algorithm based on dynamic programming on bounded treewidth graphs. Binkele-Raible and Fernau [1] further improve the running time to $O^*(2.3819^k)$.

Faster algorithms are known for graphs that have maximal degree 3. The EDS and PEDS problems in degree-3 graphs can be solved in $O^*(1.2721^n)$ time [14] and $O^*(2.1479^k)$ time, respectively [15].

In this paper, we present two new algorithms for the PEDS problem. The first is a simple and elegant algorithm that runs in $O^*(2.3715^k)$ time and polynomial space. We improve the running-time bound to $O^*(2.3147^k)$ by using a technique that deals with remaining graphs of maximal degree 3. We also design a linear-time algorithm that obtains a quadratic

* Corresponding author. Tel.: +86 15828165155; fax: +86 28 61831656.
E-mail addresses: myxiao@gmail.com (M. Xiao), kloks@cs.nthu.edu.tw (T. Kloks), spoon@cs.nthu.edu.tw (S.-H. Poon).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.06.022

kernel which is smaller than previously known kernels. Our algorithms for the PEDS problem are based on the technique of enumerating minimal vertex covers.

We introduce the idea for the algorithms in Section 2, and we introduce some basic techniques in Section 3. We present a simple algorithm for the PEDS problem in Section 4, and an improved algorithm in Section 5. We postpone the analysis of a subalgorithm to Section 6.

In Section 7, we discuss the problem kernel. Finally, in Section 8, we draw our conclusions, mention some related problems, and discuss possible further research.

2. Enumeration-based algorithms

As in many previous algorithms for the edge dominating set problem (see, e.g., [3,4,12,14]), our algorithms are based on the enumeration of minimal vertex covers.

Notice that the vertex set of an edge dominating set is a vertex cover. Conversely, let C be a minimal vertex cover, and let M be a minimum edge dominating set containing C in the set of its endpoints. Given C, M can be computed in polynomial time by computing a maximum matching in induced graph $G[C]$ and adding an edge for each unmatched vertex in C. This observation reduces the problem to that of finding the right minimal vertex cover C.

Thus, the idea is to enumerate all minimal vertex covers. Moon and Moser showed that the number of minimal vertex covers is bounded by $3^{n/3}$, and this shows that one can solve the EDS problem in $O(1.4423^n)$ time [7,8].

For the PEDS problem, we want to find an edge dominating set of size bounded by k. It follows that we need to enumerate minimal vertex covers of size only up to $2k$. We use a branch-and-reduce method to find the vertex covers. We fix some part of a minimal vertex cover and then we try to extend it with at most p vertices. Initially, $p = 2k$.

For a subset $C \subseteq V$ and an independent set $I \subseteq V \setminus C$ in G, an edge dominating set M is called a (C, I)-eds if

$$C \subseteq V(M) \quad \text{and} \quad I \cap V(M) = \emptyset.$$

In the search for the vertex cover $V(M)$ of a minimum (C, I)-eds M, we keep track of a partition of the vertices of G in four sets: C, I, U_1, and U_2. Initially, $C = I = U_1 = \emptyset$ and $U_2 = V$. The following conditions are kept invariant.

(1) I is an independent set in G, and
(2) each component of $G[U_1]$ is a clique component of $G[V \setminus (C \cup I)]$.

The vertices in $U_1 \cup U_2$ are called *undecided* vertices. We use a five-tuple

$$(G, C, I, U_1, U_2)$$

to denote the state described above. We let $q_i = |Q_i|$ denote the number of vertices of a clique component Q_i of $G[U_1]$. Van Rooij and Bodlaender proved the following lemma in [12].

Lemma 1. *If $U_2 = \emptyset$, then a minimum (C, I)-eds M of G can be found in polynomial time.*

When there are no undecided vertices in the graph, we can easily find a minimum (C, I)-eds. Lemma 1 tells us that clique components in the *undecided graph* $G[V \setminus (C \cup I)]$ do not cause trouble. We use some branching rules to deal with vertices in U_2.

Consider the following simple branching rule. For any vertex $v \in U_2$, consider two branches that either include v in the vertex cover or exclude v from the vertex cover. In the first branch, we move v into C. In the second branch, we move v into I and move the set $N(v)$ of neighbors of v into C.

When we include a number of vertices in the vertex cover, we reduce the parameter p by the same value. Furthermore, in each branch we move any newly found clique component Q in $G[U_2]$ into U_1 and reduce p by $|V(Q)| - 1$. The reason is that each clique has at most one vertex that is not in the vertex cover.

Let $C(p)$ denote the running time of our algorithm to enumerate vertex covers up to size p. We obtain the following inequality (omitting polynomial factors).

$$C(p) \leq C(p - 1 - q_v) + C(p - |N(v)| - q_{N(v)}), \tag{1}$$

where q_v (respectively, $q_{N(v)}$) denotes the sum of $|V(Q)| - 1$ over all cliques Q in $G[U_2]$ that appear after removing v (respectively, $N(v)$) from U_2.

At worst, both q_v and $q_{N(v)}$ are 0. Then, we end up with the recurrence

$$C(p) \leq C(p - 1) + C(p - |N(v)|).$$

Notice that we can always branch on vertices of degree at least 2 in $G[U_2]$. Fernau [3] solves the edge dominating set problem in $O^*(1.6181^p) = O^*(2.6181^k)$ time in this manner. Namely, this time bound stems from the branching factor of the Fibonacci recurrence

$$C(p) \leq C(p - 1) + C(p - 2).$$

Fomin et al. [4] refine this as follows. Their algorithm first branches on vertices in $G[U_2]$ of degree at least 3. In the case where all the vertices in $G[U_2]$ have degree 1 or 2, they consider the treewidth of the graph. Their algorithm solves the

problem by dynamic programming if the treewidth is small and, otherwise, if the treewidth is large, the algorithm branches further on vertices of degree 2 in $G[U_2]$. This algorithm uses exponential space, and its running time depends on the running time of the dynamic programming algorithms.

The method of iteratively branching on vertices of maximum degree d is powerful when $d > 2$. Unfortunately, it seems that we cannot avoid some branchings on vertices of degree 2, especially when each component of $G[U_2]$ is a path that consists of two edges (called a 2-*path component*). We refer to the *worst case* as the case in which every component of $G[U_2]$ is a 2-path.

Our algorithms branch on vertices of maximum degree and on some other local structures in $G[U_2]$ until $G[U_2]$ has only 2-path components. When we are in the worst case, our algorithms deal with the graph in the following way. Let $P = v_0v_1v_2$ be a 2-path in $G[U_2]$. We say that P is *signed* if $v_1 \in V(M)$, and *unsigned* if $v_1 \notin V(M)$. We use an efficient way to enumerate all signed 2-paths in $G[U_2]$.

In the next section, we introduce our branching rules.

3. Branching rules

Besides the simple technique of branching on a vertex, we also use the following branching rules. Recall that, in our algorithm, once a clique component Q appears in $G[U_2]$, we move $V(Q)$ into U_1 and reduce p by $|V(Q)| - 1$.

3.1. Tails

Let the vertex v_1 have degree 2. Assume that v_1 has one neighbor v_0 of degree 1 and that the other neighbor v_2 has degree greater than 1. Then we call the path $v_0v_1v_2$ a *tail*.

In this paper, when we use the notation $v_0v_1v_2$ for a tail, we implicitly mean that the first vertex v_0 is the degree-1 vertex of the tail. *Branching on a tail* $v_0v_1v_2$ means that we branch by either including v_2 in the vertex cover or excluding v_2 from the vertex cover.

Lemma 2. *If $G[U_2]$ has a tail, then we can branch with the following recurrence,*

$$C(p) \leq 2C(p-2), \tag{2}$$

the branching factor of which is 1.4143.

Proof. Let the tail be $v_0v_1v_2$. In the branch where v_2 is included in C, $\{v_0, v_1\}$ becomes a clique component and is moved into U_1. Then p reduces by 1 from v_2 and by 1 from $\{v_0, v_1\}$. In the branch where v_2 is included in I, $N(v_2)$ is included in C. Since $|N(v_2)| \geq 2$, p also reduces by 2 in this branch. □

3.2. 4-Cycles

We say that *abcd* is a 4-*cycle* if there exist the four edges *ab*, *bc*, *cd*, and *da* in the graph. Xiao [13] used the following lemma to obtain a branching rule for the maximum independent set problem. In this paper, we use it for the edge dominating set problem.

Lemma 3. *Let abcd be a 4-cycle in graph G. Then any vertex cover in G contains either a and c or b and d.*

As our algorithm aims at finding a vertex cover, it branches on a 4-cycle *abcd* in $G[U_2]$ by including a and c in C or including b and d in C. Notice that we obtain the same recurrence as in Lemma 2.

4. A simple algorithm

Our first algorithm is described in Fig. 1. The search tree consists of two parts. First, we branch on tails, 4-cycles, and vertices of maximum degree in Lines 3–4 until every component in $G[U_2]$ is a 2-path. Second, we enumerate the unsigned 2-paths in $G[U_2]$. In each leaf of the search tree, we find an edge dominating set in polynomial time by Lemma 1. We return a smallest one.

4.1. Analysis

In order to show the correctness of the algorithm, we need to explain Line 6 and Line 8.

For each 2-path in $G[U_2]$, we need at least one edge to dominate it. So, we must have that $y \leq p$ and $y \leq k$. This explains the condition in Line 6.

It is also easy to see that for each unsigned 2-path we need at least two different edges to dominate it. Let p' be the number of unsigned 2-paths. In Line 8, we enumerate the possible sets $P' \subseteq P$ of unsigned 2-paths. Notice that

$$(y + p' \leq k \text{ and } y + p' \leq p) \Leftrightarrow p' \leq z.$$

Algorithm EDS(G, C, I, U_1, U_2, p)

Input: A graph $G = (V, E)$, and a partition of V into sets C, I, U_1, and U_2. Initially, $C = I = U_1 = \emptyset, U_2 = V$. Integer p; initially, $p = 2k$.
Output: An edge dominating set of size $\leq k$ in G if it exists.

(1) **While** there is a clique component Q in $G[U_2]$ **do** move it into U_1 and reduce p by $|Q| - 1$.
(2) **If** $p < 0$ **then halt**.
(3) **While** there is a tail or 4-cycle in $G[U_2]$ **do** branch on it.
(4) **If** there is component Q of $G[U_2]$ that is not a 2-path **then** pick a vertex v of maximum degree in Q and branch on it.
(5) **Else** let P be the set of 2-paths in $G[U_2]$ and $y = |P|$.
(6) **If** $y > \min(p, k)$ **then halt**.
(7) Let $z = \min(p - y, k - y)$.
(8) **For** each subset $P' \subseteq P$ of size $0 \leq |P'| \leq z$ **do**
For each $v_0v_1v_2 \in P'$ **do** move $\{v_0, v_2\}$ into C and move v_1 into U_1;
For each $v_0v_1v_2 \in P - P'$ **do** move v_1 into C and move $\{v_0, v_2\}$ into U_1;
Compute the candidate edge dominating set M and return the smallest one. (Here $U_2 = \emptyset, C \cup I \cup U_1 = V$)

Fig. 1. Algorithm EDS(G, C, I, U_1, U_2, p).

Now, we analyze the running time of this algorithm. Lemma 1 guarantees that the subroutine in Line 9 runs in polynomial time. We focus on the exponential part of the running time. We prove a bound of the size of the search tree in our algorithm with respect to measure p.

First, we consider the running time of Lines 3–4.

Lemma 4. *Branching on tails, 4-cycles, and vertices of degree* ≥ 3 *yields the following recurrence,*

$$C(p) \leq C(p-1) + C(p-3), \tag{3}$$

with branching factor 1.4656.

Proof. If the algorithm branches on a tail or a 4-cycle, we have the upper bound given by (2). Otherwise, the algorithm branches on a vertex of maximum degree, and generates a recurrence covered by (3). Notice that (3) covers (2). This proves the lemma. □

Lemma 5. *If all components of the graph are paths and cycles, then the branchings of Algorithm* EDS *before Line* 5 *satisfy* (3).

Proof. If there is a path component of length greater than 2, then there is a tail, and the algorithm branches on it with (2).

If there is a component C_l which is an l-cycle in $G[U_2]$, the algorithm deals with it in this way: if the cycle is a 3-cycle, the algorithm moves it into U_1 without branching, since it is a clique.

If the cycle is a 4-cycle then, according to Lemma 3, our algorithm branches on it with (2).

If the cycle has length at least 5, our algorithm selects an arbitrary vertex v_0, and branches on it. Subsequently, it branches on the path that is created as long as the length of the path is greater than 2. When the cycle is a 5-cycle, we obtain the recurrence

$$C(p) \leq 3C(p-3),$$

with branching factor 1.4423.

When the cycle is a 6-cycle, we obtain the recurrence

$$C(p) \leq C(p-2) + C(p-3) + C(p-4),$$

with branching factor 1.4656.

The two recurrences above are covered by (3). Straightforward calculations show that, when the cycle has length ≥ 7, we also get a recurrence covered by (3). For brevity, we omit the details of this analysis. □

By Lemmas 4 and 5, we know that the running time of the algorithm, before it enters Line 5, is $O^*(1.4656^x)$, where x is the size of C upon entering the loop in Line 8. We now consider the time that is taken by the loop in Line 8, and then analyze the overall running time.

First, we derive a useful inequality.

Lemma 6. *Let* r *be a positive integer. Then, for any integer* $0 \leq i \leq \lfloor \frac{r}{2} \rfloor$,

$$\binom{r-i}{i} = O(1.6181^r). \tag{4}$$

Proof. Notice that

$$\binom{r-i}{i} \le \sum_{i=0}^{\lfloor \frac{r}{2} \rfloor} \binom{r-i}{i} = F(r+1),$$

where $F(r)$ is the rth Fibonacci number. We have $F(r) = O(1.6181^r)$. □

Now, we are ready to analyze the running time of the algorithm. It is clear that the loop in Line 8 takes less than $y\binom{y}{z}$ basic computations. First, assume that $x \le k$. We have that $z \le k - y$; thus $y + z \le k$. If we apply Lemma 6 with $r = y + z$, we find that the running time of the loop in Line 8 is $O^*(1.6181^k)$. By Lemmas 4 and 5, the running time of the algorithm is therefore bounded by $O^*(1.4656^x \cdot 1.6181^k) = O^*(2.3715^k)$.

Assume that $x > k$. We now use that $z \le p - y$; thus $y + z \le p$. By Lemma 6, the running time of Step 8 is $O^*(1.6181^p)$. Now, $p \le 2k - x$ and $x > k$. The running time of the algorithm is therefore bounded by

$$O^*(1.4656^x \cdot 1.6181^p) = O^*(2.3715^k).$$

We summarize the result in the following theorem.

Theorem 7. *Algorithm* EDS *solves the parameterized edge dominating set problem in* $O^*(2.3715^k)$ *time and polynomial space.*

5. An improvement

In this section, we present an improvement on Algorithm EDS. The improved algorithm is described in Fig. 2.

The search tree of this algorithm consists of three parts. First, we iteratively branch on vertices of degree ≥ 4 until $G[U_2]$ has no such vertices anymore (Line 3). Then, we partition the vertices in U_2 into two parts, $V(P)$ and U_2', where P is the set of 2-path components in $G[U_2]$ and $U_2' = U_2 \setminus V(P)$. Then the algorithm branches on vertices in U_2' until U_2' becomes empty (Lines 4–5). Finally, we enumerate the number of unsigned 2-paths in P (Line 9), and continue as in Algorithm EDS.

In Algorithm EDS1, a subroutine Branch3 deals with some components of maximum degree 3. It is called in Line 5. This is the major difference with Algorithm EDS. Algorithm Branch3 is described in Fig. 3. The algorithm contains several simple branching cases. They could be described in a shorter way, but we avoided doing that for analytic purposes.

We show the correctness of the condition in Line 7 of Algorithm EDS1. The variable p_0 in Algorithm PEDS1 marks the decrease of p by subroutine Branch3. Note that no vertices in $V(P)$ are adjacent to vertices in U_2'. Let M_1 be the set of edges in the solution with at least one endpoint in U_2', and let M_2 be the set of edges in the solution with at least one endpoint in $V(P)$. Then

$$M_1 \cap M_2 = \emptyset \quad \text{and} \quad |M_1| + |M_2| \le k \quad \text{and} \quad |M_1| \ge \frac{p_0}{2}.$$

Thus $|M_2| \le k - \frac{p_0}{2}$. The correctness of Algorithm EDS1 now follows, since the only difference is the subroutine Branch3.

5.1. Analysis of Algorithm EDS1

Lemma 8. *The branchings of Algorithm* Branch3 *satisfy the recurrence*

$$C(p) \le C(p-2) + 2C(p-3), \tag{5}$$

with branching factor 1.5214.

To avoid distraction from our main discussions, we delay the proof of Lemma 8 to Section 6.

Algorithm EDS1 first branches on vertices of degree at least 4. These branchings of the algorithm satisfy

$$C(p) \le C(p-1) + C(p-4), \tag{6}$$

the branching factor of which is 1.3803.

Recall that the subroutine Branch3 reduces p by p_0. The analysis without the subroutine is similar to the analysis of Algorithm EDS in Section 4.1, except that k is replaced by $k - \frac{p_0}{2}$ and that formula (3) is replaced by formula (6). Thus, without the subroutine Branch3, the algorithm has running time proportional to

$$(1.3803 \cdot 1.6181)^{k-\frac{p_0}{2}} = 2.2335^{k-\frac{p_0}{2}}.$$

By Lemma 8, the running time of the algorithm is therefore bounded by

$$O^*(2.2335^{k-p_0/2} \cdot 1.5214^{p_0}) = O^*(2.2335^{k-p_0/2} \cdot 2.3147^{p_0/2}) = O^*(2.3147^k).$$

This proves the following theorem.

Theorem 9. *Algorithm* EDS1 *solves the parameterized edge dominating set problem in* $O^*(2.3147^k)$ *time and polynomial space.*

Algorithm EDS1(G, C, I, U_1, U_2, p)

Input: A graph $G = (V, E)$, and a partition of V into sets C, I, U_1, and U_2. Initially, $C = I = U_1 = \emptyset, U_2 = V$. Integer p; initially, $p = 2k$.
Output: An edge dominating set of size $\leq k$ in G if it exists.

(1) **While** there is a clique component Q in $G[U_2]$ **do** move it into U_1 and decrease p by $|Q| - 1$.
(2) **If** $p < 0$ **halt**.
(3) **While** there is a vertex v of degree ≥ 4 in $G[U_2]$ **do** branch on it.
(4) Let P denote the set of 2-path components in $G[U_2]$ and $U_2' = U_2 \setminus V(P)$. Let $y = |P|$ and $p' = p$.
(5) **While** $U_2' \neq \emptyset$ and $p \geq 0$ **do**
$(G, C, I, U_1, U_2, p) = Branch3(G, C, I, U_1, U_2', \cup V(P), p)$.
(6) Let $p_0 = p' - p$.
(7) **If** $y > \min(p, k - p_0/2)$ **halt**.
(8) Let $z = \min(p - y, k - p_0/2 - y)$.
(9) **For** each subset $P' \subseteq P$ of size $0 \leq |P'| \leq z$ **do**
For each $v_0v_1v_2 \in P'$ **do** move $\{v_0, v_2\}$ into C and move v_1 into U_1;
For each $v_0v_1v_2 \in P - P'$ **do** move v_1 into C and move $\{v_0, v_2\}$ into U_1.
(10) Compute the candidate edge dominating set M and return the smallest one. (Here $U_2 = \emptyset, C \cup I \cup U_1 = V$.)

Fig. 2. Algorithm EDS1(G, C, I, U_1, U_2, p).

Algorithm Branch3(G, C, I, U_1, U_2', V(P), p)

(1) **If** there is a clique component in $G[U_2']$ **then** move it to U_1.
(2) **If** there is a 2-path component $v_0v_1v_2$ in $G[U_2']$ **then** branch on v_1.
(3) **If** $U_2' \neq \emptyset$ **then**
3.1 **If** there is a degree-3 vertex v adjacent to two degree-1 vertices in $G[U_2']$ **then** branch on v.
3.2 **Elseif** there is a tail $v_0v_1v_2$ such that v_2 is a degree-2 vertex in $G[U_2']$ **then** branch on the tail.
3.3 **Elseif** there is a degree-3 vertex v adjacent to one degree-1 vertex in $G[U_2']$ **then** branch on v.
3.4 **Elseif** there is a tail $v_0v_1v_2$ such that v_2 is a degree-3 vertex in $G[U_2']$ **then** branch on the tail.
3.5 **Elseif** there is a 4-cycle containing at least a degree-2 vertex in $G[U_2']$ **then** branch on it.
3.6 **Elseif** there is a 4-cycle in $G[U_2']$ **then** branch on it.
3.7 **Elseif** there is a degree-3 vertex v adjacent to any degree-2 vertex in $G[U_2']$ **then** branch on v.
3.8 Pick a maximum vertex v in $G[U_2']$ and branch on it.
(4) **Return** $(G, C, I, U_1, V(P), p)$.

Note. If some 2-path component $v_0v_1v_2$ is created in 3.1–3.7, then the algorithm branches on v_1 in the next iteration. We may combine the branch of the 2-path into the branch in 3.1–3.7 to analyze the running time.

Fig. 3. Algorithm Branch3(G, C, I, U_1, U_2', V(P), p).

6. Analysis of Algorithm Branch3

In this section, we complete the analysis of Algorithm Branch3 presented in Fig. 3 by proving Lemma 8.

Initially, $G[U_2']$ contains no component that is a 2-path. We prove that, in each line of Step 3, Algorithm Branch3 branches with (5), or with a better recurrence, without leaving any newly created 2-path components. To be exact, some 2-path components may be created, but they are removed immediately by an application of Line 2 in the following step. In the analysis, we merge these operations into one recurrence. We limit the number of 2-paths that are created in each step to prove the upper bound on the running time.

Lemma 10. *The branching in Line 3.1 of Algorithm* Branch3 *(together with the branching on all 2-paths that are created) generates*

$$C(p) \leq C(p-2) + 2C(p-3), \tag{7}$$

with branching factor 1.5214.

Proof. Assume that v is a degree-3 vertex with two degree-1 neighbors in $G[U_2']$. The algorithm selects v and branches; either it includes v in C or it includes v in I (and adds $\{u_1, u_2, u_3\}$, which are the neighbors of v in $G[U_2']$, to C). We are interested in the number of 2-paths that are created in each branch. In the first branch, at most one 2-path component is created. If this occurs, then the second branch creates no 2-path.

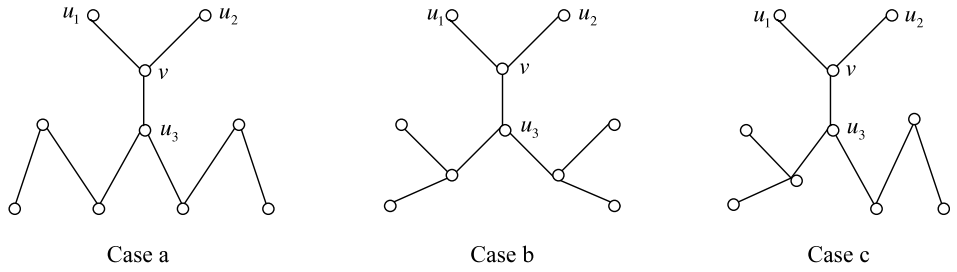

Fig. 4. The three possible graphs for Case (0, 2).

Let the pair (a, b) denote that there are a 2-paths created in the first branch and b 2-paths created in the second branch. Then the possible values for (a, b) are (0, 0), (1, 0), (0, 1), and (0, 2).

Once a 2-path component is created, the algorithm branches on it. In the first case, this gives a recurrence $C(p) \leq C(p-1) + C(p-3)$, and it leaves no 2-path component. In the second case, the algorithm branches with

$$C(p) \leq C(p-1-1) + C(p-1-2) + C(p-3) = C(p-2) + 2C(p-3),$$

and it leaves no 2-paths. In the third case, the algorithm branches with

$$\begin{aligned} C(p) &\leq C(p-1) + C(p-3-1) + C(p-3-2) \\ &= C(p-1) + C(p-4) + C(p-5), \end{aligned}$$

the branching factor of which is 1.4971. This case leaves no 2-paths. It is easy to see the above three recurrences are covered by (7).

When the fourth case occurs, there are only three possible cases for the component that contains v. We illustrate the three cases a, b, and c in Fig. 4.

In Case a, the first branch after deleting v has a path of length 6, and the algorithm will branch on a tail, and so on. We will get the following recurrence by removing all the vertices in the path:

$$\begin{aligned} C(p) &\leq C(p-2-2) + C(p-2-2) + C(p-2-1) + C(p-2-2) \\ &= C(p-3) + 3C(p-4). \end{aligned}$$

In the second branch, the algorithm branches further on two 2-paths, with the recurrence

$$\begin{aligned} C(p) &\leq C(p-1-1) + C(p-1-2) + C(p-2-1) + C(p-2-2) \\ &= C(p-2) + 2C(p-3) + C(p-4). \end{aligned}$$

Summarizing, we get

$$C(p) \leq C(p-4) + 4C(p-5) + 2C(p-6) + C(p-7),$$

with branching factor 1.4876.

In Case b, the first branch after deleting v causes the algorithm to branch further on a degree-3 vertex, and so on. We obtain the recurrence

$$\begin{aligned} C(p) &\leq C(p-1-1) + C(p-1-3) + C(p-3-1) + C(p-3-2) \\ &= C(p-2) + 2C(p-4) + C(p-5). \end{aligned}$$

In the second branch of Case b, the algorithm branches further on two 2-paths. Putting these together, we obtain

$$C(p) \leq C(p-3) + 3C(p-5) + 3C(p-6) + C(p-7),$$

with branching factor 1.5042.

In Case c, in the first branch after deleting v the algorithm branches on a degree-3 vertex, and so on. This yields

$$\begin{aligned} C(p) &\leq C(p-1-2) + C(p-1-2) + C(p-3-1) + C(p-3-2) \\ &= 2C(p-3) + C(p-4) + C(p-5). \end{aligned}$$

In the second branch of Case c, the algorithm branches on two 2-paths. If we take them together, we get

$$C(p) \leq 2C(p-4) + 2C(p-5) + 3C(p-6) + C(p-7),$$

with branching factor 1.4941.

Since the branching factor of (7) is 1.5214, it follows that (7) covers all the cases.

This proves the lemma. □

Lemma 11. *In Line 3.2 of Algorithm* Branch3 *(together with the branching on all 2-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

Proof. Assume that $v_0v_1v_2$ is the tail and that v_2 is a degree-2 vertex in $G[U_2']$. The algorithm branches on v_2 by including it in C or including it in I (and including its neighbors in C).

We consider the number of 2-path components that are created in each branch. If the component that contains the tail is a 5-path, then the algorithm branches on it according to

$$\begin{aligned} C(p) &\leq C(p-2-1) + C(p-2-2) + C(p-3) \\ &= 2C(p-3) + C(p-4). \end{aligned}$$

Otherwise, it is impossible to create a 2-path component after removing v_2, since there is now no degree-3 vertex adjacent to two degree-1 vertices.

Also, there are at most two 2-path components created in the second branch after Line 3.1. If only one 2-path component is created, the algorithm branches according to

$$\begin{aligned} C(p) &\leq C(p-2) + C(p-2-1) + C(p-2-2) \\ &= C(p-2) + C(p-3) + C(p-4). \end{aligned}$$

If two 2-path components are created, the algorithm branches with

$$\begin{aligned} C(p) &\leq C(p-2) + C(p-2-2) + 2C(p-2-3) + C(p-2-4) \\ &= C(p-2) + C(p-4) + 2C(p-5) + C(p-6). \end{aligned}$$

All the recurrences above are weaker than $C(p) \leq C(p-2) + 2C(p-3)$. This proves the lemma. □

Lemma 12. *In Line 3.3 of Algorithm* Branch3 *(together with the branching on all 2-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

Proof. Assume that v is a degree-3 vertex having one degree-1 neighbor in $G[U_2']$. The algorithm branches on v by including it in C or including it in I (and including its neighbors in C). Since Line 3.1 and Line 3.2 no longer apply, we can simply assume that in $G[U_2']$ there is no longer a degree-3 vertex adjacent to two degree-1 vertices, nor is there any tail $v_0v_1v_2$ with v_2 being a degree-2 vertex.

Under this assumption, we analyze the number of 2-path components created in each branch.

Let $u_0u_1u_2$ be a 2-path created after removing v (or $N(v)$). Then there are at least two edges between $\{u_0, u_1, u_2\}$ and v (or $N(v)$); otherwise, before branching on v, the condition of Line 3.1 or Line 3.2 holds. This implies that, after removing v, at most one 2-path component is created.

If a 2-path component is created in the first branch, then no 2-path component is created in the other branch. Therefore, the branching of the algorithm satisfies

$$\begin{aligned} C(p) &\leq C(p-1-1) + C(p-1-2) + C(p-3) \\ &= C(p-2) + 2C(p-3). \end{aligned}$$

If no 2-path component is created in the first branch, then at most two 2-path components are created in the second branch. In the worst case, we first branch on the degree-3 vertex v and then branch on two 2-path components in the second branch with

$$C(p) \leq C(p-2) + 2C(p-3) + C(p-4).$$

Therefore, we get

$$C(p) \leq C(p-1) + C(p-5) + 2C(p-6) + C(p-7), \tag{8}$$

with branching factor 1.5181. It follows that the worst recurrence is

$$C(p) \leq C(p-2) + 2C(p-3). \quad \square$$

Lemma 13. *In Line 3.4 of Algorithm* Branch3 *(together with the branching on all 2-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

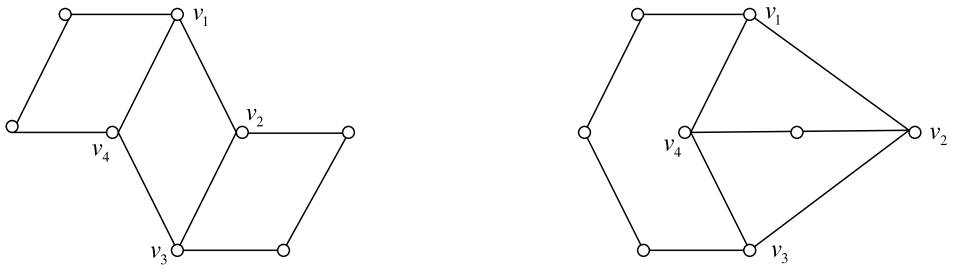

Fig. 5. The two possible graphs in the proof of Lemma 15.

Proof. The proof of Lemma 13 is similar to the proof of Lemma 12. □

Lemma 14. *In Line 3.5 of Algorithm* Branch3 *(together with the branching on all* 2*-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

Proof. Let $v_1v_2v_3v_4$ be a 4-cycle in $G[U_2']$, where v_1 is assumed to be a degree-2 vertex, without loss of generality. The algorithm branches by including v_1 and v_3 or by including v_2 and v_4 in C. Note that after Line 3.4 there are no more degree-1 vertices in $G[U_2']$. Since v_1 is a degree-2 vertex, in each branch at most one 2-path component is created. Therefore, the algorithm branches first with $C(p) \leq 2C(p-2)$ for the 4-cycle, and then it possibly branches further on a 2-path in each branch. This leads to the recurrence

$$\begin{aligned} C(p) &\leq C(p-2-1) + C(p-2-2) + C(p-2-1) + C(p-2-2) \\ &= 2C(p-3) + 2C(p-4), \end{aligned}$$

with branching factor 1.4946. This is covered by

$$C(p) \leq C(p-2) + 2C(p-3). \quad \square$$

Lemma 15. *In Line 3.6 of Algorithm* Branch3 *(together with the branching on all* 2*-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

Proof. Similarly to the proof of Lemma 14, we will prove that at most one 2-path component is created in each branch. Let $v_1v_2v_3v_4$ be a 4-cycle in $G[U_2']$; the algorithm branches by either including v_1 and v_3 or including v_2 and v_4 in C. In the first branch, after deleting v_1 and v_3 if two 2-path components are created, the four neighbors of v_1 and v_3 will become degree-1 vertices, since the graph before the branching has no degree-1 vertex. The only two possible configurations of the component containing the 4-cycle are shown in Fig. 5: each of them has a 4-cycle containing a degree-2 vertex, which contradicts with Line 3.5 no longer applying. It is also impossible to create two or more 2-path components in the second branch. The following part of the proof follows the proof of Lemma 14. □

Lemma 16. *In Line 3.7 of Algorithm* Branch3 *(together with the branching on all* 2*-paths that are created), the algorithm branches with*

$$C(p) \leq C(p-2) + 2C(p-3).$$

Proof. Assume that v is a degree-3 vertex adjacent to at least one degree-2 neighbor in $G[U_2']$. The algorithm branches on v by including it in C or I.

In the first branch, no 2-path is created; otherwise, there is a 4-cycle in $G[U_2']$ before the branching on v.

In the second branch, where $N(v)$ is moved into C, there are at most two 2-path components created. Note that, for any 2-path component $u_0u_1u_2$ that is created after removing $N(v)$ there are at least two edges between $\{u_0, u_1, u_2\}$ and $N(v)$, and at least one vertex in $N(v)$ is a degree-2 vertex. Therefore, we have (8) as an upper bound. □

Now we are ready to complete the proof of Lemma 8.

Proof. Notice that Lemmas 10–16 guarantee that, if any of the Lines 3.1–3.7 are called, the algorithm branches according to formula (5).

Next, we analyze the branch in Line 3.8. In this step, the induced subgraph $G[U_2']$ has only two kinds of component: each component is either a 3-regular graph without any 4-cycle or a cycle containing at least five edges.

If there is a 3-regular component, the algorithm selects an arbitrary vertex v, and branches on it. According to the analysis in the proof of Lemma 15, no 2-path components are created after removing v. In the branch where $N(v)$ is removed, at most one 2-path is created. Note that, for any 2-path component $u_0u_1u_2$, created after removing $N(v)$, there are five edges between

$\{u_0, u_1, u_2\}$ and $N(v)$, because each vertex is a degree-3 vertex before the branching. In the worst case, the algorithm still branches with

$$\begin{aligned} C(p) &\le C(p-1) + C(p-3-1) + C(p-3-2) \\ &= C(p-1) + C(p-4) + C(p-5). \end{aligned}$$

This is weaker than formula (5).

Next, we assume that each component is a cycle containing at least five edges. We show that the algorithm can branch with a recurrence better than (5) to remove all the vertices in a cycle of length at least 5 out of U_2'. The algorithm selects an arbitrary vertex v in the cycle, and branches on it. In the remaining graph, a path is created, and the algorithm will further branch on trails, and so on. Straightforward manipulation shows that, when the cycle contains $5 \le l \le 10$ edges, we can get a recurrence weaker than (5) by removing all vertices in the cycle out of U_2' in each branch (a detailed proof can be derived from the analysis in [15]). For each cycle containing $l > 10$ edges, in the first branch after deleting v we can further branch on a tail with $C(p) \le 2C(p-2)$ without leaving a component of a 2-path in each branch. Therefore, we can branch on the cycle with

$$C(p) \le C(p-1-2) + C(p-1-2) + C(p-2) = C(p-2) + 2C(p-3),$$

and this leaves a path of length ≥ 3 in each branch, which can always be branched with a recurrence weaker than (5) by the above analysis.

Therefore, the branching of Algorithm Branch3 satisfies (5). □

7. Kernelization

A *kernelization algorithm* takes an instance of a parameterized problem and transforms it into an equivalent parameterized instance (called the *kernel*), such that the new parameter is at most the old parameter and the size of the new instance is a function of the new parameter.

For the parameterized edge dominating set problem, Prieto [9] presented a quadratic-time algorithm that finds a kernel with at most $4k^2 + 8k$ vertices by adapting 'crown reduction techniques'. Fernau [3] obtained a kernel with at most $8k^2$ vertices.

We present a new linear-time kernelization that reduces a parameterized edge dominating set instance (G, k) to another instance (G', k') such that

$$|V(G')| \le 2k'^2 + 2k', \qquad |E(G')| = O(k'^3) \quad \text{and} \quad k' \le k.$$

In our kernelization algorithm, we first find an arbitrary maximal matching M_0 in the graph in linear time. Let $m = |M_0|$. Then we may assume that $m \ge k+1$; otherwise, M_0 is a solution. Let

$$V_m = V(M_0) \quad \text{and} \quad V^* = V - V_m.$$

Since M_0 is a maximal matching, we know that V^* is an independent set. For a vertex $v_i \in V_m$, let $x_i = |V^* \cap N(v_i)|$. We call vertex $v_i \in V_m$ *overloaded*, if $m + x_i > 2k$. Let $A \subseteq V_m$ be the set of overloaded vertices.

Lemma 17. *Let M be an edge dominating set of size at most k. Then*

$$A \subseteq V(M).$$

Proof. If an overloaded vertex $v_i \notin V(M)$, then all neighbors of v_i are in $V(M)$. Note that at least one endpoint of each edge in M_0 (an arbitrary maximal matching) must be in $V(M)$, and that $V^* \cap N(v_i)$ and $V(M_0)$ are disjoint. Therefore, $|V(M)| \ge x_i + m$. Since v_i is an overloaded vertex, we have that $|V(M)| > 2k$. This implies that $|M| > k$, which is a contradiction. □

Lemma 17 implies that all overloaded vertices must be in the vertex set of the edge dominating set. We *annotate* these vertices to indicate that these vertices are in the vertex set of the edge dominating set.

We also annotate all vertices that have a neighbor of degree 1.

Our kernelization algorithm is presented in Fig. 6. In the algorithm, the set A' denotes the set of annotated vertices. The correctness of the algorithm follows from the following observations. Assume that there is a vertex u adjacent only to annotated vertices. Then we can delete it from the graph without increasing the size of the solution. The reason is this. Let ua be an edge that is in the edge dominating set of the original graph, where a is an annotated vertex. Then we can replace ua with another edge that is incident with a to get an edge dominating set of the new graph. This is formulated in the reduction rule in Line 4 of the algorithm. We add a new edge for each annotated vertex in Line 5 to enforce that the annotated vertices are selected in the vertex set of the edge dominating set.

It is easy to see that each step of the algorithm can be implemented in linear time. Therefore, the algorithm takes linear time.

We analyze the number of vertices in the new graph G' returned by Algorithm *Kernel*(G, k). Note that A' is a subset of V_m. Let $B = V_m - A'$. Let q be the number of edges between V^* and B. Then

$$q = \sum_{v_i \in B} x_i \le \sum_{v_i \in B} (2k - m) = |B|(2k - m).$$

Algorithm *Kernel*(G, k)

(1) Find a maximal matching M_0 in G.
(2) Let $A \subseteq V_m$ be the set of overloaded vertices and $\tilde{A} \subseteq V_m$ be the set of vertices having some degree-1 neighbor.
(3) Let $A' = A \cup \tilde{A}$.
(4) If there is a vertex $u \in V^*$ such that $N(u) \subseteq A'$, then delete u from G.
(5) For each vertex $w \in A'$, add a new vertex w' and a new edge $w'w$
(In the analysis we assume that the new vertex w' is in V^*).
(6) Return $(G', k' = k)$, where G' is the new graph.

Fig. 6. Algorithm *Kernel*(G, k).

Let

$$V_1^* = \bigcup_{v \in B} N(v) \cap V^* \quad \text{and} \quad V_2^* = V^* - V_1^*.$$

Each vertex in V_1^* is adjacent to a vertex in B. Since there are at most q edges between V_1^* and B, we have

$$|V_1^*| \leq q.$$

Notice that all vertices of V_2^* have neighbors only in A'. In Line 4, the algorithm deletes all vertices that have neighbors only in A'. In Line 5, the algorithm adds a new vertex w' and a new edge $w'w$ for each vertex w in A'. Thus V_2^* is the set of new vertices that are added in Line 5. This proves that

$$|V_2^*| = |A'| = 2m - |B|.$$

The total number of vertices in the graph is

$$\begin{aligned} |V_m| + |V_1^*| + |V_2^*| &\leq 2m + |B|(2k - m) + (2m - |B|) \\ &= 4m + |B|(2k - m - 1) \\ &\leq 4m + 2m(2k - m - 1) && \text{since } |B| \leq 2m \\ &= 2m(2k - m + 1) \\ &\leq 2k(k + 1) && \text{since } m \geq k + 1. \end{aligned}$$

Note that the maximal value of $2m(2k-m+1)$ as a function of m is attained for $m = k+\frac{1}{2}$. Thus the function $2m(2k-m+1)$ is decreasing for $m \geq k + 1$.

To obtain a bound for the number of edges, we partition the edge set into three disjoint sets.

(1) Let E_1 be the set of edges with two endpoints in V_m;
(2) let E_2 be the set of edges between A' and V^*; and
(3) let E_3 be the set of edges between B and V^*.

It is easy to see that

$$|E_1| = O(m^2) = O(k^2) \quad \text{and} \quad |E_3| = q = |B|(2k - m) = O(k^2).$$

By the analysis above,

$$|E_2| \leq |A'| \cdot |V_1^*| + |V_2^*| \leq |A'|q + |V_2^*| \Rightarrow |E_2| = O(k^3).$$

Thus we obtain the following theorem.

Theorem 18. *Algorithm Kernel runs in linear time and linear space, and it returns a kernel with at most* $2k^2 + 2k$ *vertices and* $O(k^3)$ *edges.*

8. Conclusions

In this paper, we have presented several improved parameterized algorithms for the edge dominating set problem. Our algorithms can also be used to get improved parameterized algorithms for some related problems. There are standard techniques to reduce the *parameterized maximal matching problem* that finds a maximal matching of size k in a graph to the parameterized edge dominating set problem without increasing the input size and the parameter [17]. The *parameterized matrix domination problem* (given an $m \times n$ matrix M with entries being 0 or 1 and an integer k, the problem is to find a subset S of the 1-entries in M such that $|S| \leq k$ and every row and column of M contains at least one 1-entry in S) reduces directly to a parameterized edge dominating set problem in a bipartite graph [17,3]. Van Rooij and Bodlaender [12] also show that algorithms based on enumerating vertex covers for the parameterized edge dominating set problem can be extended to the *parameterized weighted edge dominating set problem* (to find an edge dominating set of size at most k in an edge-weighted

graph such that the total weight of the edges in the edge dominating set is minimized) without increasing the dominating part of the running time. By Theorem 9, we know that the above three related problems can also be solved in $O^*(2.3147^k)$ time and polynomial space, improving previous parameterized algorithms for them.

Our algorithms for the edge dominating set problem are based on enumerating vertex covers. Although there are many techniques that lead to fast algorithms for the vertex cover problem, only a few of the techniques can be used in the edge dominating set problem. To find a vertex cover, we use a branch-and-search method.

The bottleneck of our algorithm is to branch on connected components that are 2-paths, which implies the Fibonacci recurrence $C(p) \leq C(p-1) + C(p-2)$. In this paper, we obtained an improvement by reducing the number of the worst recurrences in the search tree. However, we cannot totally avoid bad branchings. Recently, Xiao and Nagamochi [15] showed that, when the graph is restricted to a graph with maximum degree 3, the worst recurrence can be improved to $C(p) \leq C(p-1) + C(p-3)$. It is still an open question whether we can obtain this improvement for general graphs.

Another interesting problem is to improve the kernel of the edge dominating set problem. The vertex cover problem has a linear kernel. Is there a kernel of linear size for the edge dominating set problem?

Acknowledgments

The first author was supported in part by Grant 60903007 of NSFC, China. The second author was supported in part by Grant 99-2218-E-007-016 of NSC, Taiwan. The third author was supported in part by Grant 97-2221-E-007-054-MY3 of NSC, Taiwan.

References

[1] D. Binkele-Raible, H. Fernau, Enumerate and measure: improving parameter budget management, in: Proceedings IPEC, in: LNCS, vol. 6478, Springer-Verlag, 2010, pp. 38–49.
[2] J. Cardinal, S. Langerman, E. Levy, Improved approximation bounds for edge dominating set in dense graphs, Theor. Comput. Sci. 410 (2009) 949–957.
[3] H. Fernau, Edge dominating set: efficient enumeration-based exact algorithms, in: H. Bodlaender, M. Langston (Eds.), Proceedings IWPEC, in: LNCS, vol. 4169, Springer-Verlag, 2006, pp. 142–153.
[4] F. Fomin, S. Gaspers, S. Saurabh, A. Stepanov, On two techniques of combining branching and treewidth, Algorithmica 54 (2) (2009) 181–207.
[5] T. Fujito, H. Nagamochi, A 2-approximation algorithm for the minimum weight edge dominating set problem, Discrete Appl. Math. 118 (3) (2002) 19–207.
[6] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, 1979.
[7] D. Johnson, M. Yannakakis, C. Papadimitriou, On generating all maximal independent sets, Inform. Process. Lett. 27 (3) (1988) 119–123.
[8] J.W. Moon, L. Moser, On cliques in graphs, Israel J. Math. 3 (1965) 23–28.
[9] E. Prieto, Systematic kernelization in FPT algorithm design, Ph.D. Thesis, The University of Newcastle, Australia, 2005.
[10] B. Randerath, I. Schiermeyer, Exact algorithms for minimum dominating set, Technical Report zaik 2005-501, Universität zu Köln, Germany, 2005.
[11] V. Raman, S. Saurabh, S. Sikdar, Efficient exact algorithms through enumerating maximal independent sets and other techniques, Theory Comput. Syst. 42 (3) (2007) 563–587.
[12] J.J.M. van Rooij, H.L. Bodlaender, Exact algorithms for edge domination, in: M. Grohe, R. Niedermeier (Eds.), Proceedings IWPEC, in: LNCS, vol. 5018, Springer-Verlag, 2008, pp. 214–225.
[13] M. Xiao, A simple and fast algorithm for maximum independent set in 3-degree graphs, in: Md.S. Rahman, S. Fujita (Eds.), Proceedings WALCOM, in: LNCS, vol. 5942, Springer-Verlag, 2010, pp. 281–292.
[14] M. Xiao, H. Nagamochi, Exact algorithms for annotated edge dominating set in cubic graphs. TR 2011-009. Kyoto University. 2011. A preliminary version appeared as: M. Xiao, Exact and parameterized algorithms for edge dominating set in 3-degree graphs, in: W. Wu, O. Daescu (Eds.), Proceedings COCOA, in: LNCS, vol. 6509, Springer-Verlag, 2010, pp. 387–400.
[15] M. Xiao, H. Nagamochi, Parameterized edge dominating set in cubic graphs, in: Proceedings FAW-AAIM, in: LNCS, vol. 6681, Springer-Verlag, 2011, pp. 100–112.
[16] M. Xiao, H. Nagamochi, A refined exact algorithm for edge dominating set, in: M. Agrawal, S.B. Cooper, A. Li (Eds.), TAMC 2012, in: LNCS, Vol. 7287, Springer-Verlag, 2012, pp. 360–372.
[17] M. Yannakakis, F. Gavril, Edge dominating sets in graphs, SIAM J. Appl. Math. 38 (3) (1980) 364–372.