Algorithmica (2012) 64:535–563
DOI 10.1007/s00453-011-9546-x

Exact Algorithms for Edge Domination

Johan M.M. van Rooij · Hans L. Bodlaender

Received: 4 July 2008 / Accepted: 21 June 2011 / Published online: 6 July 2011

Abstract An edge dominating set in a graph $G = (V, E)$ is a subset of the edges $D \subseteq E$ such that every edge in E is adjacent or equal to some edge in D. The problem of finding an edge dominating set of minimum cardinality is NP-hard. We present a faster exact exponential time algorithm for this problem. Our algorithm uses $O(1.3226^n)$ time and polynomial space. The algorithm combines an enumeration approach of minimal vertex covers in the input graph with the branch and reduce paradigm. Its time bound is obtained using the measure and conquer technique. The algorithm is obtained by starting with a slower algorithm which is refined stepwisely. In each of these refinement steps, the worst cases in the measure and conquer analysis of the current algorithm are reconsidered and a new branching strategy is proposed on one of these worst cases. In this way a series of algorithms appears, each one slightly faster than the previous one, ending in the $O(1.3226^n)$ time algorithm. For each algorithm in the series, we also give a lower bound on its running time.

We also show that the related problems: minimum weight edge dominating set, minimum maximal matching and minimum weight maximal matching can be solved in $O(1.3226^n)$ time and polynomial space using modifications of the algorithm for edge dominating set. In addition, we consider the matrix dominating set problem which we solve in $O(1.3226^{n+m})$ time and polynomial space for $n \times m$ matrices, and the parametrised minimum weight maximal matching problem for which we obtain an $O^*(2.4179^k)$ time and space algorithm.

This research was partially supported by project BRICKS (Basic Research for Creating the Knowledge Society).

J.M.M. van Rooij (✉) · H.L. Bodlaender
Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
e-mail: jmmrooij@cs.uu.nl

H.L. Bodlaender
e-mail: hansb@cs.uu.nl

Keywords Edge dominating set · Minimum maximal matching · Exact algorithms · Exponential time algorithms · Measure and conquer

1 Introduction

Research on exponential time algorithms for finding exact solutions to NP-hard problems dates back to the sixties and seventies. Some natural problems such as independent set [27, 30], colouring [20] and Hamiltonian circuit [15] have been studied for a long time, while for other problems such as dominating set [9, 13, 28], treewidth [11] and feedback vertex set [26] exact exponential algorithms with non-trivial running times date from only recently.

There is renewed interest in these algorithms, also visible in a recent series of surveys on the matter [10, 16, 29, 35, 36]. An important new technique is measure and conquer [10, 13]. This technique allows us to derive better upper bounds on branch and reduce algorithms by using non-standard measures of instance size. It reminds us of the somewhat similar earlier approaches such as the approach of Kullmann to prove upper bound on search trees for 3-SAT instances [19] or Eppstein's quasiconvex analysis of backtracking algorithms [6].

In this paper, we consider the minimum edge dominating set problem. This problem is identical to the problem of finding a minimum dominating set in a line graph. While both the edge dominating set problem and the dominating set problem are NP-hard [37], in some ways the problem restricted to line graphs is easier. For instance, minimum dominating set is hard to approximate [7], while minimum edge dominating set is constant-factor approximable [3]. Also from the parametrised point of view, minimum dominating set most likely is not fixed parameter tractable (it is W[2]-complete [4]), while minimum edge dominating set is fixed parameter tractable [8]. In the setting of exact exponential time algorithms, it also seems that the edge dominating set problem is somewhat easier; the currently best known time bound for an exact algorithm for minimum dominating set is $O(1.4969^n)$ [32] (see also [13, 33, 34]), while in this paper we present an $O(1.3226^n)$ time algorithm for minimum edge dominating set.

The first exact algorithm for edge dominating set is from 2005 due to Randerath and Schiermeyer [28] who gave an algorithm of time complexity $O(1.4423^m)$. Raman et al. [25] gave an $O(1.4423^n)$ algorithm and recently Fomin et al. [12] improved this to $O(1.4082^n)$.

In this paper we further investigate the idea of enumerating minimal vertex covers in order to compute the minimum edge dominating set. Although this technique has already been used frequently on this problem, we were able to formulate reduction rules applied during the enumeration of the minimal vertex covers, which allow us to create a faster algorithm. These reduction rules are derived from the manner in which these vertex covers are used for solving the edge dominating set problem. Our first algorithm already improves on the literature by using these reduction rules, but no complicated techniques at all. The time bound for this algorithm is tightened considerably more by analysing it with measure and conquer. Furthermore the measure and conquer methodology allows us to create a series of improved algorithms for which we can derive even smaller upper bounds on their running times.

We also show that our ideas for minimum edge dominating set extend to minimum maximal matching and matrix dominating set, and with some more modifications also to minimum weight edge dominating set and minimum weight maximal matching. A consequence of our results also solves a problem left open by Fernau in [8] and gives an $O^*(2.4179^k)$ time algorithm for the parametrised minimum weight maximal matching problem.

We will first introduce some notation, concepts and the problems under study in Sect. 2. Then, in Sect. 3, we will show how we use minimal vertex covers to obtain an exact algorithm for the edge dominating set problem with a running time exponential in the number of vertices, not edges. In Sect. 4 we improve upon this algorithm by introducing reduction rules and a change in the branching strategy of the algorithm, which we later analyse using measure and conquer in Sect. 5. In Sect. 6 we further change the branching strategy of the algorithm and obtain an $O(1.3226^n)$ time and polynomial space algorithm. Finally in Sect. 7 we extend our results to weighted edge domination problems.

2 Preliminaries

Let $G=(V,E)$ be an n-node undirected simple graph. Let $G[V']$ be the subgraph of G induced by a subset $V' \subseteq V$ and let $L(G)$ be the line graph of a graph G: $L(G) = (E, \{\{e_1, e_2\} \mid \exists_{v \in V}\ v \in e_1 \wedge v \in e_2\})$. Let be $N(v)$ the open neighbourhood of a vertex $v \in V$, $N[v]$ be the closed neighbourhood of $v \in V$ ($N[v] = N(v) \cup \{v\}$), and, for a vertex $v \in V'$, let $N_{V'}(v)$, $N_{V'}[v]$ be the open, respectively closed, neighbourhoods of v in $G[V']$. $N_{V'}(V'')$ is an extension of this notation to neighbourhoods of $V'' \subseteq V$: $N_{V'}(V'') = (\bigcup_{v \in V''} N_{V'}(v)) \backslash V''$, $N_{V'}[V''] = \bigcup_{v \in V''} N_{V'}[v]$. For vertex $v \in V'$ with $V' \subseteq V$, we define the V'-degree of a vertex v to be the degree of v in $G[V']$.

A subset $D \subseteq V$ is a *dominating set* in a graph G if for every vertex $v \in V$ there exists a vertex $w \in D$ such that $v \in N[w]$; we say that w dominates v in this case. A *minimum dominating set* is a dominating set of minimum cardinality in a given graph. The *minimum edge dominating set problem* is: given a graph G, find a minimum dominating set in the line graph $L(G)$. Equivalently, look for the smallest *edge dominating set*: a subset $D \subseteq E$ such that every edge $e \in E$ is dominated by an edge $f \in D$, where f dominates e if e and f have an end point in common. For an edge weight function $\omega : E \to \mathbb{R}_{\geq 0}$ the *minimum weight edge dominating set problem* is: given a graph G, find an edge dominating set D of minimum total weight $\sum_{e \in D} \omega(e)$.

An *independent set* in a graph G is a subset $I \subseteq V$ such that no two distinct vertices $v, w \in I$ are adjacent to each other and a *vertex cover* in G is a subset $C \subseteq V$ such that every edge $e \in E$ is incident to some vertex $v \in C$. Notice that the complement of an independent set is a vertex cover and vice versa. A *maximal independent set* is an independent set such that for each $v \in V \backslash I$, $I \cup \{v\}$ is not an independent set. The complement of a maximal independent set is a *minimal vertex cover*, i.e., a vertex cover C such that for each $v \in C$, $C \backslash \{v\}$ is not a vertex cover.

A *matching* in a graph G is subset $M \subseteq E$ such that no two distinct edges $e, f \in M$ have an end point in common. A *maximum matching* is a matching of maximum cardinality, a *maximal matching* is a matching M such that for every $e \in E$, $M \cup \{e\}$

Algorithm 1 Simple Edge Dominating Set Algorithm

```
Input: a graph $G = (V, E)$
Output: a minimum edge dominating set in $G$
1: compute the set $\mathcal{C}$ of all minimal vertex covers in $G$
2: for all minimal vertex covers $C \in \mathcal{C}$ do
3:    let $C_i$ be the set of all isolated vertices in $G[C]$
4:    compute a minimum edge cover $C'$ in $G[C \backslash C_i]$
5:    let $D_C$ be $C'$ plus an extra edge for each vertex in $C_i$ containing this vertex
6: return the $D_C$ encountered of minimum cardinality
```

is not a matching, and a *perfect matching* is a matching M such that for every $v \in V$ there exists an edge $e \in M$ incident to v. Finally, an *edge cover* is a subset $C' \subseteq E$ such that every vertex $v \in V$ is incident to an edge $e \in C'$. Maximum matchings and minimum weight perfect matchings can be computed in polynomial time [5]. As a consequence an edge cover of minimum cardinality (*minimum edge cover*) can be computed in polynomial time also by computing a maximum matching and adding for each unmatched vertex an edge incident to it.

An interesting related problem is *minimum maximal matching*: given a graph G, find a maximal matching of minimum cardinality. This is equivalent to the *minimum independent edge dominating set problem*, where independence between edges is interpreted in terms of the line graph. For this problem we also consider the weighted variant *minimum weight maximal matching*: given a graph G, find a maximal matching of minimum total weight.

Another interesting related problem is *matrix dominating set*. In this problem, we are given an $n \times m$ 0–1 matrix and we need to find a set of 1 entries of minimum cardinality such that every 1 entry is on the same row or column as a selected 1-entry. As noted in [37] this problem is equivalent to *bipartite edge dominating set*: a matrix M that is an instance of matrix dominating set corresponds to the instance of bipartite edge dominating set where there exist a vertex for each row and each column, and an edge between a row vertex and a column vertex if and only if its corresponding entry in M is a 1.

3 Using Minimal Vertex Covers

We start by first giving a simple exact algorithm for the edge dominating set problem. This algorithm is based upon the following observation; see Algorithm 1.

Proposition 1 *If $D \subseteq E$ is an edge dominating set in $G = (V, E)$, then $C = \{v \in V \mid \exists_{e \in D}\ v \in e\}$ is a vertex cover in G.*

Proof For each $e \in E$, there is an edge $f \in D$ that dominates e, i.e., e and f have an end point in common. This endpoint belongs to C and therefore C is a vertex cover. □

Theorem 1 *Algorithm 1 solves the minimum edge dominating set problem in $O(1.4423^n)$ time.*

Proof We prove that the minimum size of an edge dominating set in G equals the minimum cardinality of a set D_C as computed by Algorithm 1 over all minimal vertex covers C in G. From this the correctness follows.

If C is a minimal vertex cover, then D_C is an edge dominating set, for if any edge is not dominated then both endpoints are not in C which contradicts C being a vertex cover. Therefore, the algorithm returns an edge dominating set. To see that it is of minimum size, consider a minimum edge dominating set D in G. By Proposition 1, its endpoints form a vertex cover C in G. From this vertex cover C, a minimum edge dominating set can be reconstructed by computing a minimum edge cover in $G[C]$. The vertex cover C does not need to be minimal, but for any minimal vertex cover $C_1 \subseteq C$, the edges incident to a vertex $v \in C \backslash C_1$ are all dominated by any choice of edges incident to the vertices in C_1. Thus, D_{C_1} constructed by Algorithm 1 from the minimal vertex cover C_1 dominates the same edges as D. And, it is not larger than D since the edge cover D_{C_1} needs to cover only a subset of the vertices in C. Hence, D_{C_1} is a minimum edge dominating set.

The running time is derived from the Moon and Moser bound [22] on the number of maximal independent sets, and hence minimal vertex covers in G: this number is bounded by $3^{n/3} < 1.4423^n$. Enumerating all minimal vertex covers can be done with only polynomial delay [17, 21], therefore Algorithm 1 has a running time of $O(1.4423^n)$. □

Following the notation of the proof, the smallest edge dominating set which contains $C \subseteq V$ as endpoints will be denoted by D_C from now on.

By using a standard technique from [14], this also gives an algorithm for the minimum maximal matching problem.

Corollary 1 *The minimum maximal matching problem can be solved by a modification of Algorithm 1 in $O(1.4423^n)$ time.*

Proof Take the minimum edge dominating set computed by Algorithm 1. Let $\{u, v\}$, $\{v, w\}$ be a pair of dominating edges incident to the same vertex v. By minimality there cannot be another dominating edge incident to w (remove $\{v, w\}$ for a smaller edge dominating set). Also, there must be a vertex x adjacent to w without any incident dominating edge, for otherwise the edge dominating set without $\{v, w\}$ would be a smaller edge dominating set. Hence, we can replace $\{v, w\}$ by $\{w, x\}$ obtaining an edge dominating set with one pair of not independent dominating edges less and repeating this process results in a minimum maximal matching. □

4 Exploiting Properties of Edge Dominating Sets

The $3^{n/3}$ upper bound on the number of minimal vertex covers in a graph G is tight; consider the family of graphs consisting of l triangles: these graphs have $3l$ vertices and 3^l minimal vertex covers. However, from the perspective of computing a minimum edge dominating set, this class of graphs is trivial: just pick an edge from each triangle.

In this section we use properties of edge dominating sets in order to enumerate fewer minimal vertex covers, avoiding situations of the type we just described, and in this way we introduce a faster algorithm than the simple algorithm of Sect. 3. These modifications are very simple, yet powerful enough to already improve upon the algorithm by Fomin et al. [12] which uses far more complicated techniques. First we will introduce a reduction rule and secondly we will introduce a more efficient branching strategy. Like Algorithm 1, the new algorithm enumerates a series of minimal vertex covers, and computes for each of these minimal vertex covers C the smallest edge dominating set D_C that contains the vertices C in its set of endpoints. To this end, it continuously keeps track of a partitioning of the vertices of G in three sets: a set C of vertices that must become part of the minimal vertex cover, a set I of vertices that may not become part of the minimal vertex cover (they are in the complementing maximal independent set), and a set U of vertices, which we call the set of *undecided* vertices. We denote such a state by the four-tuple (G, C, I, U).

We introduce the following rule:

Rule 1

```
**if** $G[U]$ contains a connected component $H$ which is a clique **then**
    let $\tilde{G}$ be the graph obtained from $G$ by adding a new vertex $v$ connected to all
    vertices in $H$
    $\tilde{C} := C \cup H \cup \{v\}$; $\tilde{U} := U \backslash H$
    recursively solve the problem $(\tilde{G}, \tilde{C}, I, \tilde{U})$ and let $D$ be the resulting edge dominating set
    **if** $D$ contains two distinct edges $\{u, v\}, \{v, w\}$ incident to $v$ **then**
        **return** $(D \backslash \{\{u, v\}, \{v, w\}\}) \cup \{\{u, w\}\}$
    **return** $D \backslash \{\{u, v\}\}$, where $\{u, v\}$ is the unique edge in $D$ incident to $v$
```

A simpler rule can be used for clique components of size 1 or 2. Isolated vertices in $G[U]$ can be put into I. K_2 components in $G[U]$ can be put into C if they have no neighbours in C in G, and they can be contracted after which we put the resulting vertex into C, otherwise.

Proof of Correctness After the recursive call the extra vertex v is incident to at least one edge in D, since $v \in \tilde{C}$. Also v is incident to at most two edges in D, for otherwise two such edges can be replaced by the edge joining the other endpoints which gives a smaller edge dominating set with $\tilde{C}$ as a subset of the set of endpoints.

All clique edges in the original graph are dominated if at most one clique vertex is not incident to a dominating edge. Therefore if D contains only one edge incident to v, removing this edge results in an edge dominating set in the original graph with C as a subset of its set of endpoints. Because D is of minimum cardinality (in $\tilde{G}$) and the returned set is of cardinality one smaller it must also be of minimum cardinality (in G): if it is not then adding the edge between the unique vertex of the clique that is not an endpoint in the edge dominating set and D results in a smaller alternative for D.

If D contains two edges incident to v, replacing these by the edge joining the other endpoints also results in such an edge dominating set in the original graph. This edge dominating set is also of minimal cardinality because adding any edge incident to v gives an alternative for D. □

Algorithm 2 Faster Edge Dominating Set Algorithm

```
Input: a graph $G = (V, E)$
Output: a minimum edge dominating set in $G$
 1: $I := \emptyset$; $C := \emptyset$; $U := V$
 2: if $G[U]$ contains a connected component $H$ which is a clique then
 3:    apply Rule 1
 4: else if a vertex $v$ of maximum degree in $G[U]$ has $U$-degree at least three then
 5:    create two subproblems and solve each one recursively:
 6:        1: $(G, C \cup N_U(v), I \cup \{v\}, U \backslash N_U[v])$    2: $(G, C \cup \{v\}, I, U \backslash \{v\})$
 7: else
 8:    for all minimal vertex covers $\hat{C}$ on $G[U]$ do
 9:        compute the candidate edge dominating set $D_{C \cup \hat{C}}$
10: return the smallest edge dominating set encountered
```

If Rule 1 does not apply, Algorithm 2 picks any undecided vertex $v \in U$ of maximum degree in $G[U]$ (maximum number of undecided neighbours in G). If v has U-degree at least three we branch on this vertex generating two subproblems. In one subproblem v is put in the independent set I; because no neighbour of v can also be in the independent set I these neighbours (at least three) are all put in the vertex cover C. In the other subproblem v is put the vertex cover C. We note that this may result in the construction of vertex covers which are not minimal, but all minimal vertex covers are enumerated in this way.

If v has U-degree smaller than three, $G[U]$ is of maximum degree at most two and due to Rule 1, $G[U]$ does not contain a connected component that is a clique. Therefore, $G[U]$ now consists of a collection of paths on at least three vertices and cycles on at least four vertices. In this case, Algorithm 2 enumerates all minimal vertex covers on these paths and cycles.

For each resulting partition of V in an independent set I and a vertex cover C, Algorithm 2 computes a candidate for the minimum edge dominating set D_C in the same way as Algorithm 1 and returns the candidate of minimum cardinality.

Theorem 2 *Algorithm 2 solves the minimum edge dominating set problem in $O(1.3803^n)$ time and polynomial space.*

Proof Correctness of the algorithm follows directly from the proof of Theorem 1 and the correctness of Rule 1.

Let $P(l)$ be the number of maximal independent sets on a path and $C(l)$ be the number of maximal independent sets on a cycle on l vertices. For each vertex in a maximal independent set I in a path, the next vertex in I must be at distance two or three; hence:

$$P(1) = 1 \qquad P(2) = 2 \qquad P(3) = 2 \quad \forall_{l \geq 4} : P(l) = P(l-2) + P(l-3)$$
$$l \geq 3 : P(l) \leq \beta^l \quad \text{where } \beta \text{ is the root of } 1 = \beta^{-2} + \beta^{-3} \text{ and } \beta < 1.33$$

The latter follows by induction after noting that it holds for $l \in \{3, 4, 5\}$.

For cycles on $l \leq 6$ vertices, a simple enumeration gives $C(l)$. For $l \geq 7$ consider an arbitrary vertex v on a cycle on l vertices. If v is in a maximal independent set I, then neither of its neighbours are, leaving $P(l-3)$ possibilities. If $v \notin I$, then one or both of its neighbours are in I. Each of the cases where one neighbour is in I leaves $P(l-6)$ possibilities because by maximality of I the neighbour of the neighbour of v that is not in I must belong to I. In the case that both neighbours are in I five vertices are fixed leaving $P(l-5)$ possibilities. Hence:

$$C(4)=2 \qquad C(5)=5 \qquad C(6)=5 \quad \forall_{l\geq 7}: C(l)=P(l-3)+P(l-5)+2P(l-6)$$

Let u be the number of undecided vertices in our problem instance (initially $u=n$), and $S(u)$ be the number of subproblems generated to solve an instance with $|U|=u$. We have:

$$S(u) \leq \begin{cases} S(u-1)+S(u-4) & \text{branch on a vertex of } U\text{-degree at least three} \\ P(l)S(u-l) & \text{enumerate minimal vertex covers in a path} \\ & \text{on } l \text{ vertices} \\ C(l)S(u-l) & \text{enumerate minimal vertex covers in a cycle} \\ & \text{on } l \text{ vertices} \end{cases}$$

Because of the branching on a vertex of degree three $S(u) \leq \alpha^u$, where α is the solution to $1=\alpha^{-1}+\alpha^{-4}$. For the enumeration of minimal vertex covers in paths $S(u) \leq \beta^l S(u-l) \leq \beta^l \alpha^{u-l} < \alpha^u$, because $\beta < \alpha$. And also for the enumeration of minimal vertex covers in cycles $S(u) < \alpha^u$, since the solution to $\gamma^u = C(l)\gamma^{u-l}$ converges to $\gamma = \beta$ when $l \to \infty$ and reaches its maximum on $l \geq 4$ when $l=5$; here $\gamma < 1.379 < \alpha < 1.3803$. The worst case over these three possibilities gives $S(u) \leq \alpha^u$ which results in the running time of $O(poly(n)\alpha^n)$ or $O(1.3803^n)$.

The collection of minimal vertex covers constructed is not being stored, the enumeration search tree is traversed, therefore the algorithm uses only polynomial space. □

Remark 1 We cannot improve Algorithm 2 by putting more paths or cycles in the polynomial part of the algorithm (assuming $P \neq NP$). This is because one can show that the following problem is NP-hard. Given a graph $G=(V,E)$ and a set of marked vertices C, find a minimum edge dominating D set in G that satisfied the following two properties:

- all marked vertices are an endpoint of an edge in D.
- $G[V \setminus C]$ (this is $G[U]$ in our algorithms) is a collection of paths on at most three vertices.

Consider a SAT instance on variables $x_1, x_2, \ldots, x_n$ and clauses $c_1, c_2, \ldots, c_m$. We can safely assume that all variables x_i occur at least once as a positive literal and at least once as a negative literal.

Introduce a marked vertex (a vertex in C) for each clause. We will connect this clause vertex to gadgets representing the variables; this is done such that the edge between the gadget and this clause vertex will be selected in the minimum edge dominating if and only if the corresponding literal is *True*. Notice that because the clause vertex is marked it must be an endpoint of at least one edge in the required minimum edge dominating set.

Next, introduce a marked vertex (a vertex in C) incident to two edges for each variable. One of both edges must be selected since the variable vertex is a marked vertex, and which one is selected represents whether the variable is set to *True* or *False*. If the variable occurs only once as a positive or only once as a negative literal, we directly connect the corresponding edge to the vertex representing the corresponding clauses. Otherwise, we let the edge be incident to the middle vertex of an unmarked path on three vertices; these are the only unmarked vertices in our construction and these are of length at most tree as claimed. Each of the two endpoints of this path will be connected to a new marked vertex v_1 and v_2. These vertices v_1 and v_2 are both incident to one more edge which other endpoint we will assign soon. Suppose that the path is connected to the *True* edge of a variable vertex while this variable is set to *True*. In this case, the edges of the path are dominated by the selected edge of the variable vertex, and it is always optimal to pick the second edge (with so far unassigned other endpoint) of v_1 and v_2. If the current variable x_i occurs twice as a positive literal, we can now connect v_1 and v_2 to the corresponding clause vertices. Otherwise, we can add more of these path gadgets to increase the number of occurrences to any positive number. We repeat the same construction for the negative literals of the variable, and for all variables.

Since all marked vertices that are not clause vertices are non-adjacent, any minimum edge dominating set that contains all marked vertices as endpoints uses at least $|C| - m$ edges. It is not hard to see that such an minimum edge dominating set of size $|C| - m$ exists if and only if the corresponding SAT instance is satisfiable.

In this NP-hardness proof, the paths on three vertices can easily be replaced by cycles on four vertices.

5 Measure and Conquer

When we branch on a vertex of large degree in $G[U]$, not only will it be removed from U, it will also reduce the degrees of its neighbours in $G[U]$ in one branch, and it will reduce the degrees of the vertices at distance two in $G[U]$ in the other. Since we can deal with vertices of U-degree at most two (collections of paths and cycles in $G[U]$) in less time than we need for vertices of U-degree three or four, this reduction of the degrees means additional progress for the algorithm. In this section we show how we can keep track of this additional progress by using the *measure and conquer* technique [10, 13]. In combination with a slightly changed branching strategy on paths and cycles in $G[U]$ this leads to an improved time bound.

We first modify the enumeration of minimal vertex covers on paths and cycles: Algorithm 3 no longer enumerates all minimal vertex covers, but instead branches on the third vertex v of a path on at least four vertices and applies Rule 1. In one branch, v is put in the independent set resulting in the removal of four vertices: v, its neighbours, and the remaining isolated vertex. In the other branch, v is put in the vertex cover resulting in the removal of three vertices: v and the first two vertices of the path, since they now form a 2-clique in $G[U]$. Using this branching strategy

Algorithm 3 Third Edge Dominating Set Algorithm

```
Input: a graph $G = (V, E)$
Output: a minimum edge dominating set in $G$
1: $I := \emptyset$; $C := \emptyset$; $U := V$
2: if $G[U]$ contains a connected component $H$ which is a clique then
3:    apply Rule 1
4: else if a vertex $v$ of maximum degree in $G[U]$ has $U$-degree at least three
      or $G[U]$ contains a connected component $H$ which is a cycle on $l \geq 5$ vertices,
      $v \in H$ then
5:    create two subproblems and solve each one recursively:
6:       1: $(G, C \cup N_U(v), I \cup \{v\}, U \backslash N_U[v])$       2: $(G, C \cup \{v\}, I, U \backslash \{v\})$
7: else if $G[U]$ contains a connected component which is cycle on four vertices
   then
8:    Let $v_1, v_2, v_3, v_4$ be the vertices along the cycle and recursively solve the sub-
      problems:
9:       1: $(G, C \cup \{v_1, v_3\}, I \cup \{v_2, v_4\}, U \backslash \{v_1, v_2, v_3, v_4\})$
         2: $(G, C \cup \{v_2, v_4\}, I \cup \{v_1, v_3\}, U \backslash \{v_1, v_2, v_3, v_4\})$
10: else if $G[U]$ contains a connected component which is a path on $l \geq 4$ vertices
    then
11:    Let $v_1, v_2, v_3, v_4$ be the vertices at an end of the path and recursively solve the
       subproblems:
12:       1: $(G, C \cup \{v_2, v_4\}, I \cup \{v_1, v_3\}, U \backslash \{v_1, v_2, v_3, v_4\})$
          2: $(G, C \cup \{v_3\}, I, U \backslash \{v_3\})$
13: else if $G[U]$ contains a connected component which is a path on three vertices
    then
14:    Let $v$ be the middle vertex and recursively solve the subproblems:
15:       1: $(G, C \cup N_U(v), I \cup \{v\}, U \backslash N_U[v])$
          2: $(G, C \cup \{v\}, I \cup N_U(v), U \backslash N_U[v])$
    {Now: $U = \emptyset$, $C \uplus I = V$}
16: else
17:    compute the candidate edge dominating set $D_C$
18: return the smallest edge dominating set encountered
```

on paths, we break cycles on at least five vertices by branching in two subproblems: pick any vertex v and put v in the vertex cover or put v in the independent set and its neighbours in the vertex cover. Finally, we still enumerate all minimal vertex covers on remaining paths on three vertices or cycles on four vertices. This results in Algorithm 3.

We estimate the number of subproblems generated by branching on paths and cycles:

Lemma 1 *For Algorithm 3 and $l \geq 4$:*

1. *A cycle component C_l in $G[U]$ generates a maximum of $4^{l/6}$ subproblems.*
2. *A path component P_l in $G[U]$ generates a maximum of $4^{(l-1)/6}$ subproblems.*

Notice that we can repeatedly encounter these cycles and paths, and hence these numbers are multiplied. Therefore we need exact bounds on the number of subproblems generated in this way.

Proof (1) Let $P'(l)$, $C'(l)$ be the number of subproblems generated by Algorithm 3 when dealing with a path or cycle on l vertices, respectively. Derive the values of $P'(l)$ and $C'(l)$ for $l \leq 4$ directly and consider the following recurrence relation:

$$P'(1) = P'(2) = C'(3) = 1 \qquad P'(3) = P'(4) = C'(4) = 2$$
$$\forall_{l\geq 5} : P'(l) = P'(l-3) + P'(l-4) \qquad C'(l) = P'(l-1) + P'(l-3)$$

Let γ be the solution to $1 = \gamma^{-3} + \gamma^{-4}$. For $l \geq 4$, $P'(l) < \gamma^l$ follows by induction after noting that it holds for $l \in \{4, 5, 6, 7\}$. For $l \geq 10$ we have:

$$C'(l) < \gamma^{l-1} + \gamma^{l-3} = \gamma^l(\gamma^{-1} + \gamma^{-3}) = \left(\gamma\sqrt[l]{\gamma^{-1} + \gamma^{-3}}\right)^l < (4^{1/6})^l$$

using the fact that $\sqrt[l]{\gamma^{-1} + \gamma^{-3}}$ is decreasing and smaller than $4^{1/6}$ if $l \geq 10$. Direct computation shows that for $l < 10$: $C'(l) \leq 4^{l/6}$.

(2) For $l \geq 8$, $\gamma^{l/(l-1)}$ is decreasing and smaller than $4^{1/6}$, therefore:

$$P'(l) < \gamma^l = \left(\gamma^{l/(l-1)}\right)^{l-1} < (4^{1/6})^{l-1}$$

For $4 \leq l \leq 7$: $P'(l) \leq 4^{(l-1)/6}$, by direct computation. □

These estimates are tight: when Algorithm 3 branches on a C_6 component in $G[U]$ ($l = 6$), we indeed generate $4 = 4^{l/6}$ subproblems.

For the measure and conquer analysis we need a weight function $w : \mathbb{N} \to [0, 1]$ assigning weights $w(d)$ to vertices of degree d in $G[U]$. Instead of counting the number of undecided vertices to measure the progress of our algorithm, we will now use their total weight $k = \sum_{v\in U} w(\deg_{G[U]}(v))$ as a measure of complexity. This is justified by the fact that if we can show that our algorithm runs in $O(\alpha^k)$ time using weight function w, it will also run in $O(\alpha^n)$ time, since for any problem instance $k \leq n$.

Theorem 3 *Algorithm 3 solves the minimum edge dominating set problem in $O(1.3323^n)$ time and polynomial space.*

Proof Let $w : \mathbb{N} \to [0, 1]$ be the weight function assigning weight $w(\deg_{G[U]}(v))$ to vertices $v \in G[U]$. The algorithm removes all vertices of U-degree zero, therefore $w(0) = 0$. Let $\Delta w(i) = w(i) - w(i-1)$. Vertices with a larger U-degree should be given a larger weight, hence we demand: $\forall_{n\geq 1}\ \Delta w(n) \geq 0$. Furthermore we impose non-restricting steepness inequalities: $\forall_{n\geq 1}\ \Delta w(n) \geq \Delta w(n+1)$.

Consider an instance where the algorithm branches on a vertex v of maximum U-degree $d \geq 3$ with r_i neighbours of degree i in $G[U]$ ($d = \sum_{i=1}^{d} r_i$). If v is put in the vertex cover, it is removed from U and the U-degrees of all its neighbours in $G[U]$

are decreased by one. If v is placed in the independent set then $N_U[v]$ is removed from U, and the total sum of the degrees of the remaining vertices is reduced by at least d_2; here d_2 is a lower bound on the number of edges between $N_U[v]$ and vertices at distance two from v in $G[U]$:

$$d_2 = \left(\sum_{i=1}^{d} (i-1) r_i \right) \bmod 2 \quad \text{except when } d = r_3 = 3 \text{ then: } d_2 = 2$$

This follows from a parity argument: there must be an edge in $G[U]$ with only one endpoint in $N_U[v]$ if $1 \equiv \sum_{i=1}^{d} (i-1) r_i \pmod 2$. Also $N_U[v]$ cannot be a clique by Rule 1, hence if $d = r_d$ there must be at least two edges in $G[U]$ with only one endpoint in $N_U[v]$.

Altogether we conclude that the algorithm recurses on two instances which are reduced Δ_{indep} and Δ_{vc} in the measured complexity:

$$\Delta_{indep} = w(d) + \sum_{i=1}^{d} r_i w(i) + d_2 \Delta w(d) \qquad \Delta_{vc} = w(d) + \sum_{i=1}^{d} r_i \Delta w(i)$$

Let $S(k)$ be the number of subproblems generated to solve a problem of measured complexity k. For all $d \geq 3$ and $(d = \sum_{i=1}^{d} r_i)$ we have a recurrence relation of the form:

$$S(k) \leq S(k - \Delta_{indep}) + S(k - \Delta_{vc})$$

We define $q(w)$ to be the functional mapping a weight function to the solution of this entire set of recurrence relations.

By Lemma 1, an l-cycle or l-path generates a maximum of $4^{l/6}$, respectively $4^{(l-1)/6}$, subproblems. An l-cycle has a measured complexity of at least $l \cdot w(2)$ and a path on l vertices has a measured complexity of at least $(l-1) \cdot w(2)$, since $\Delta w(1) \geq \Delta w(2)$ and hence $2w(1) \geq w(2)$. Therefore, in an instance where the vertices in cycle components and path components on at least four vertices in $G[U]$ have measured complexity k'; the removal of these vertices from U by Algorithm 3 results in a maximum of $4^{k'/6w(2)}$ subproblems.

We now look for the optimal weight function $w : \mathbb{N} \to [0,1]$, satisfying the restrictions, such that the following maximum over the worst case behaviours of the different branch cases is minimum. We distinguish between the case where the maximum U-degree is three or more, the case where cycles and paths on at least four vertices are removed from $G[U]$, and the case where a path on three vertices is removed from $G[U]$.

$$S(k) \leq \left(\min_{w:\mathbb{N} \to [0,1]} \max\left\{ q(w), 4^{1/6w(2)}, 2^{1/(w(2)+2w(1))} \right\} \right)^k$$

By setting from some large enough integer $p \geq 3$: $\forall_{i \geq p} w(i) = 1$ we obtain a finite problem with the recurrence relations for $3 \leq d \leq p+1$ and $d = \sum_{i=1}^{d} r_i$. In this finite problem all recurrences with $d > p+1$ are dominated by those with $d = p+1$.

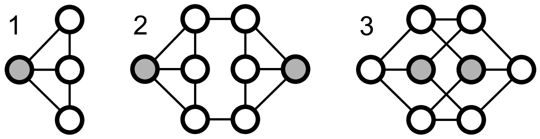


Fig. 1 Lower bound graphs

As a result we have obtained a quasiconvex program [6]. Solving this numerically, we obtain the optimal weights and a solution $\alpha < 1.3323$:

$$w(1) = 0.750724 \qquad w(2) = 0.914953 \quad \forall_{i\geq 3}\, w(i) = 1$$

Therefore an instance of measured complexity k generates less than 1.3323^k subproblems, leading to the upper bound on the running time of $O(1.3323^n)$. Since we do not store any subproblems, but just traverse an enumeration tree, we use only polynomial space. □

Since measure and conquer analyses only provide upper bounds on the running time of an algorithm, it is useful to consider lower bounds also.

Proposition 2 *The worst case running time of Algorithm 3 is* $\Omega(1.3160^n)$.

Proof Consider the class of graphs consisting of l disjoint copies of Graph 1, in Fig. 1. On each individual copy, Algorithm 3 can branch on the leftmost vertex resulting in two subproblems: one where this entire copy is removed from U and one where a path of length three remains in $G[U]$. This leads to a total of three subproblems for each copy of the graph. Therefore Algorithm 3 generates $3^l = 3^{n/4} > 1.3160^n$ subproblems on this class of graphs. This proves the $\Omega(1.3160^n)$ lower bound. □

6 Improving the Worst Cases

It is often a good idea to reconsider the quasiconvex program obtained from a measure and conquer analysis. The quasiconvex function optimised in the proof of Theorem 3 equals the maximum over the solutions to a series of recurrence relations: one for each subcase considered. The solution to this quasiconvex program is an optimal point $x \in \mathbb{R}^p$. In x, or any other feasible point in $\mathbb{R}^p$, some of the solutions to the individual recurrence relations are tight to the maximum. If one slightly varies the weights at this optimum x, the solutions to these tight recurrence relations increase (by optimality of x). If we now change our algorithm in such a way that it handles such a tight subcase in a more efficient way, the corresponding recurrence relation changes: its solution becomes smaller. In this case we can move out of x to a new optimum, with a necessarily smaller maximum over the solutions of the recurrence relations. This results in a smaller upper bound on the running time. This idea was first introduced in [33] for the design of algorithms for minimum dominating set and was named *design by measure and conquer*. This technique will be used in this section to obtain a series of algorithms that improve upon Algorithm 3.

The quasiconvex program associated with Algorithm 3 (see the proof of Theorem 3) gives the following tight worst cases:

1. $d = 3, r_2 = 2, r_3 = 1$, i.e., we have a vertex of maximum U-degree three, with two neighbours in $G[U]$ of U-degree two and one neighbour in $G[U]$ of U-degree three.
2. $d = 3, r_3 = 3$: we have a vertex of maximum U-degree three, with three neighbours in $G[U]$ of U-degree three.
3. a connected component in $G[U]$ is a path on three vertices.

We can improve upon the first two cases, while improving upon the third seems hard (see Remark 1). Consider the first case. In this case, v is a vertex of maximum U-degree where the algorithm branches on. It has degree three with two neighbours $u_1, u_2 \in U$ of U-degree two and one neighbour $u_3 \in U$ of U-degree three. In our analysis of Sect. 5, we had a lower bound d_2 on the number of edges between $N_U[v]$ and the vertices and distance two from v; for this case we had $d_2 = 0$. We can now consider two subcases.

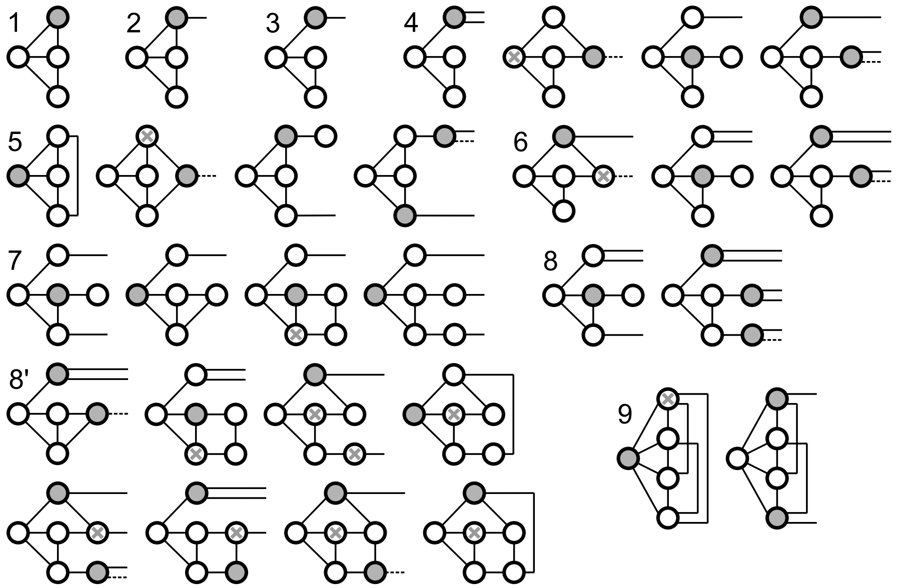
In the first subcase v, u_1, u_2 and u_3 form a connected component in $G[U]$, isomorphic to Subgraph 1 in Fig. 2. Algorithm 3 branches on v. We modify this now, by instead branching on one of the U-degree two vertices, e.g., u_1. In both subproblems that are obtained after branching on u_1, the vertices of the subgraph that remain in U form a clique in $G[U]$, and so are dealt with by Rule 1. Therefore the entire subgraph disappears from $G[U]$ after one branching step and application of Rule 1, while previously, we had a path on three vertices remaining in $G[U]$ in one subproblem that required another branching step.

In the second subcase u_1, u_2 and/or u_3 are adjacent to vertices in $U \backslash \{v, u_1, u_2, u_3\}$. If we branch on v, then these vertices will have their U-degrees reduced by one in one branch, implying a larger progress than estimated in Sect. 5: by a parity argument we can use $d_2 = 2$ as a new lower bound on the number of edges between $N_U[v]$ and the rest of $G[U]$.

Thus we modify the algorithm and split this case in two subcases in the measure and conquer analysis. The optimum of the resulting quasiconvex program proves an upper bound on the running time of $O(1.3315^n)$ for this modified algorithm.

Arguments, similar to the argument given above for the case $d = 3, r_2 = 2, r_3 = 1$ can be given in a large number of other cases as well. This leads to a series of improvement steps, and a series of algorithms: each algorithm slightly improves upon the previous. We give the other improvements on the algorithms in a more schematic manner. Listed in the order in which they appear as worst cases in the improvement series, we introduce the following set of alternative branching strategies to Algorithm 3. The numbering corresponds to the subgraphs of $G[U]$ drawn in Fig. 2. We refer to the vertex on which the previous algorithm could branch as v (leftmost vertex in the Fig. 2), and we denote the reductions in measured complexity of the subproblems generated by the alternative branching strategy by $\Delta_1, \Delta_2, \ldots$. At each subcase, we either increase the current lower bound d_2 on the number of edges between $N_U[v]$ and the rest of $G[U]$, or find other means of increasing the lower bound on the total reductions in measure when branching (Δ_{indep} and Δ_{vc}). See Table 1 for the upper bounds on the running times of the individual algorithms in the series.

1. $d = 3, r_2 = 2, r_3 = 1$. See introductory example. The subcase tight to $d_2 = 0$ is handled more efficiently by branching on a U-degree two vertex. This results in



The leftmost vertex in every subgraph corresponds to a vertex we could branch on in Algorithm 3 *and grey vertices* represent more efficient alternatives. If multiple vertices are grey, simultaneously branch on these vertices generating four or eight subproblems. *Crossed vertices* represent vertices branched on directly hereafter, but only in the subproblems where this induces extra 1, 2 or 3-cliques. Sometimes small path components remain in a subproblem; these are immediately branched upon also.
Unfinished edges always connect to vertices outside the drawn subgraph, and there are no other edges in $G[U]$ between vertices with at least one drawn endpoint. Dashed edges are optional.

Fig. 2 More efficient branching strategies on possible subgraphs of $G[U]$

Table 1 Bounds on the running times of the algorithms in the improvement series

Strategies	None	1	1–2	1–3	1–4	1–5	1–7	1–8	1–9
$O(x^n)$	1.3323	1.3315	1.3296	1.3280	1.3265	1.3248	1.3240	1.3228	1.3226
$\Omega(x^n)$	1.3160	1.2968	1.2968	1.2968	1.2968	1.2753	1.2753	1.2753	1.2753

$\Delta_1 = \Delta_2 = 2w(3) + 2w(2)$. All other subcases have at least two edges with only one endpoint in $N_U[v]$, thus: $d_2 = 2$.

2. $d = 3, r_2 = 1, r_3 = 2$. If there is a unique edge in $G[U]$ with only one endpoint u in $N_U[v]$, then u has U-degree three. Branch on u and apply Rule 1 to any 3-clique remaining in $G[U]$. Because the other vertex incident to this unique edge has weight at least $w(1)$, and when its U-degree is reduced by one this reduces its weight by at least $\Delta w(3)$, we derive $\Delta_1 = 3w(3) + w(2) + w(1)$, $\Delta_2 = 3w(3) + w(2) + \Delta w(3)$. For the other subcases with $d = 3, r_2 = 1, r_3 = 2$ the number of edges from $N_U[v]$ to the rest of $G[U]$ is at least 3 by a parity argument. Hence, we can now use $d_2 = 3$ for these cases.
3. $d = 3, r_2 = 3$. Similar to Case 2: $\Delta_1 = w(3) + 3w(2) + w(1)$, $\Delta_2 = w(3) + 3w(2) + \Delta w(3)$. For the remaining subcases, we have that $d_2 = 3$.

4. $d = 3, r_2 = 2, r_3 = 1$. Case 1 reappears; consider four more subcases representing $d_2 = 2$.
   (a) Both edges with only one endpoint in $N_U[v]$ are incident to the same vertex $u \in N_U(v)$. Branch on u and apply Rule 1 if possible; this is similar to Cases 2 and 3. $\Delta_1 = 2w(3) + 2w(2) + 2w(1)$, $\Delta_2 = 2w(3) + 2w(2) + 2\Delta w(3)$.

   Let $u, w \in N_U(v)$ be incident to the edges with only one endpoint in $N_U[v]$ such that u has U-degree two and w has U-degree three.
   (b) Both edges in $G[U]$ with only one endpoint in $N_U[v]$ are incident to the same vertex $x \notin N_U(v)$. Branch on x, and if it is put in the vertex cover also branch on v. Because paths on two vertices are removed from $G[U]$: $\Delta_1 = \Delta_2 = \Delta_3 = 2w(3) + 3w(2)$.
   (c) If the vertex outside of $N_U[v]$ adjacent to w in $G[U]$ has U-degree one, branch on w. This represents a different case of the quasiconvex program: $d = 3, r_1 = r_2 = r_3 = 1$.
   (d) If the vertex $x \notin N_U[v]$ that is adjacent to w in $G[U]$, has U-degree two or three, branch simultaneously on x and u. $\Delta_1 = 2w(3) + 3w(2) + \min(w(2), 2w(1))$, $\Delta_2 = \Delta_3 = 2w(3) + 3w(2) + w(1) + \Delta w(3)$, $\Delta_4 = 2w(3) + 3w(2) + 2\Delta w(3)$. For Δ_1 we use the minimum extra reduction over the subcase where two edges with one endpoint in $N_U[v]$ are incident to the same vertex or to two different vertices outside $N_U[v]$.

   For all other subcases with $d = 3, r_2 = 2, r_3 = 1$ we now have $d_2 = 4$.
5. $d = 3, r_3 = 3$. Because of Rule 1: $d_2 = 2$. In Sect. 7 we discuss variants of our algorithm for which we do not have a reduction rule dealing with this subcase. Therefore we consider the subcase with $d_2 = 0$ as if the reduction rule was not in our algorithm: remove it using two subproblems by branching on any vertex. $\Delta_1 = \Delta_2 = 4w(3)$.

   The rest of this case is identical to Subcases 4(b–d), with $\Delta_1 = \Delta_2 = \Delta_3 = 4w(3) + w(2)$ in Subcase (b), and $\Delta_1 = 4w(3) + w(2) + \min(w(2), 2w(1))$, $\Delta_2 = \Delta_3 = 4w(3) + w(2) + w(1) + \Delta w(3)$, $\Delta_4 = 4w(3) + w(2) + 2\Delta w(3)$ in Subcase (d). For all remaining subcases set $d_2 = 4$.
6. $d = 3, r_2 = 1, r_3 = 2$. As we handled Case 2 earlier, we have $d_2 = 3$. Suppose that the U-degree two neighbour of v is adjacent to another neighbour of v in $G[U]$. See Case 7 when this extra condition does not apply. Let T be the 3-clique (triangle) in $G[U]$ containing v.
   (a) A vertex $u \neq v$ in $G[U]$ is a neighbour of two vertices in $N_U(v)$. Branch on the neighbour of v incident to two edges with one endpoint in $N_U[v]$. In the subproblem where u is not removed also branch on u. $\Delta_1 = 3w(3) + 2w(2) + w(1)$, $\Delta_2 = \Delta_3 = 3w(3) + 2w(2) + \Delta w(3)$.
   (b) In $G[U]$ a vertex $u \in T$ has a U-degree one neighbour: branch on u.
   (c) In the remaining subcase branch on both vertices in $N_U(T)$. $\Delta_1 = 3w(3) + 2w(2) + \min(w(2) + w(1), 3w(1))$, $\Delta_2 = 3w(3) + 2w(2) + 2w(1) + \Delta w(3)$, $\Delta_3 = 3w(3) + 2w(2) + w(1) + 2\Delta w(3)$, $\Delta_4 = 3w(3) + 2w(2) + 3\Delta w(3)$. Notice that since $G[U]$ is simple, the minimum in the formula for Δ_1 does not need to consider $w(3)$: not all edges with only one endpoint drawn in Fig. 2 can be incident to the same vertex.

7. $d = 3, r_2 = 1, r_3 = 2$, again $d_2 = 3$ as we handled Case 2 earlier. Because of Case 6 suppose that the U-degree two neighbour of v is not adjacent to another neighbour of v in $G[U]$. Let T be the 3-clique in $G[U]$ with $v \in T$.
   (a) If any vertex in $N_U(T)$ has U-degree one, branch on its neighbour in T.
   (b) If any vertex in $N_U(T)$ has U-degree three, we proceed to Case 8, where we let v be the vertex in T that is a neighbour to this U-degree three vertex. This case is illustrated in Fig. 2, Case 8 (not Case 7).

   For all other subcases $N_U(T)$ consists of vertices of U-degree two.
   (c) A vertex in $u \in U$ is adjacent to two vertices in T. Branch on the vertex in T not adjacent to u. $\Delta_1 = 3w(3) + 2w(2) + \Delta w(3)$, $\Delta_2 = 3w(3) + w(2) + \Delta w(2)$.
   (d) Two vertices in U adjacent to T are adjacent to each other. Branch on a vertex $u \in T$ adjacent to one of these two vertices. When u is put in the vertex cover branch on the other vertex adjacent to one of these two vertices. $\Delta_1 = \Delta_2 = 3w(3) + 2w(2) + \Delta w(2)$, $\Delta_3 = 2w(3) + 2w(2) + \Delta w(3) + \Delta w(2)$.
   (e) The U-degree two neighbour of v is adjacent to a neighbour of a vertex in T in $G[U]$. Notice that this case is isomorphic to Subcase (d) as the triangle T is adjacent to three degree two vertices, two of which from a 4-cycle with T. Hence, this case can be dealt with similarly.
   (f) Left is the subcase where no vertices in U neighbouring T are adjacent: branch on v.

   Together with Case 6 this allows us to add an additional $2(\Delta w(2) - \Delta w(3))$ to Δ_{indep} for $d = 3, r_2 = 1, r_3 = 2$. This is because the only remaining subcase is Subcase 7(f) where putting v in the independent set gives at least two vertices of U-degree two whose U-degree is reduced, in contrast to the original analysis, where we did not have the U-degrees of these vertices specified.
8. $d = 3, r_3 = 3$ with $d_2 = 4$ since we handled Case 5 earlier. We consider many subcases.

   If not all vertices neighbouring the triangle T containing v in $G[U]$ are different, or they are adjacent to each other, then we again give alternative ways of branching. These specific cases are shown visually as Case 8 in Fig. 2: the first picture corresponds to a vertex being adjacent to two vertices of T; the other pictures on the top row correspond to T having two neighbours of degree two; and the pictures on the bottom row represent neighbourhoods of T with at most one degree two vertex. We will not explicitly state the recurrences representing these cases: they can be derived easily in a way similar to the above analysis.

   We assume all three vertices that are neighbours of T to be different and non-adjacent.
   (a) Again if any U-degree one vertex is a neighbour of a vertex $u \in T$, branch on u.
   (b) Otherwise, if at most one of the vertices adjacent to T in $G[U]$ has degree two, branch on all three vertices neighbouring T in $G[U]$ simultaneously. In the worst case, one vertex has degree two resulting in:
   $\Delta_1 = 5w(3) + w(2) + \min(w(3) + w(2), w(3) + 2w(1), 2w(2) + w(1), w(2) + 3w(1), 5w(1))$, $\Delta_2 = \Delta_3 = 5w(3) + w(2) + \min(w(2) + w(1), 3w(1)) + 2\Delta w(3)$, $\Delta_4 = 5w(3) + w(2) + w(1) + 4\Delta w(3)$, $\Delta_5 = 5w(3) + w(2) +$

$\min(2w(2), w(2) + 2w(1), 4w(2)) + \Delta w(3)$, $\Delta_6 = \Delta_7 = 5w(3) + w(2) + 2w(1) + 3\Delta w(3)$, $\Delta_8 = 5w(3) + w(2) + 5\Delta w(3)$.

The first four reductions correspond to the degree two vertex being put in the independent set and the last four reductions correspond to it being put in the vertex cover.

The only case that remains is the case where two vertices adjacent to T have degree two, and we can assume that these are not adjacent to v. Similar to the previous case, we add an additional $2(\Delta w(2) - \Delta w(3))$ to Δ_{indep} in the recurrence representing branching on v since at least two vertices at distance two from v now have degree two. We remark that this improved case remains a worst case in the quasiconvex program.

9. $d = 4, r_4 = 4$. If all vertices in $N_U[v]$ are pairwise adjacent we have a clique that can be filtered out by Rule 1. It can also be removed using three subproblems by branching on any two vertices: $\Delta_1 = \Delta_2 = \Delta_3 = 5w(4)$.

   If there are two edges in $G[U]$ with only one endpoint in $N_U[v]$ then we branch on both vertices in $N_U[v]$ incident to these edges. $\Delta_1 = 5w(4) + \min(w(2), 2w(1))$, $\Delta_2 = \Delta_3 = 5w(4) + w(1) + \Delta w(4)$, $\Delta_4 = 5w(4) + 2\Delta w(4)$. This results in $d_2 = 4$ for all other subcases.

Algorithm 4 Let Algorithm 4 be the modification of Algorithm 3 using all the alternative branching strategies discussed above and illustrated in Fig. 2.

Theorem 4 *Algorithm 4 solves the minimum edge dominating set problem in $O(1.3226^n)$ time and polynomial space.*

Proof Reconsider the quasiconvex program used to prove the running time of Theorem 3 and modify the values of Δ_{indep} as justified by the above case analysis. This gives:

$$\Delta_{indep} = w(d) + \sum_{i=1}^{d} r_i w(i) + d_2 \Delta w(d) + \begin{cases} 2(\Delta w(2) - \Delta w(3)) & \text{if } d = 3, r_2 = 1, r_3 = 2 \\ & \text{or } d = r_3 = 3 \\ 0 & \text{otherwise} \end{cases}$$

$$d_2 = \left\lceil \left(\sum_{i=1}^{d} (i-1) r_i \right) \bmod 2 \right\rceil + \begin{cases} 4 & \text{if } d = 3, r_2 = 2, r_3 = 1 \quad \text{(Cases 1 and 4)} \\ 2 & \text{if } d = 3, r_2 = 1, r_3 = 2 \quad \text{(Case 2; also 6, 7)} \\ 2 & \text{if } d = r_2 = 3 \quad \text{(Case 3)} \\ 2 & \text{if } d = r_3 = 3 \quad \text{(Case 5; also 8)} \\ 4 & \text{if } d = r_4 = 4 \quad \text{(Case 9)} \end{cases}$$

Furthermore, we add additional recurrences corresponding to the alternative branching strategies for all the subcases listed above. In order to keep the problem finite, set $w(i) = 1$ for $i \geq p$ for some $p \geq 4$ (see Remark 2).

The solution to this modified quasiconvex program gives a running time of $O(1.3226^n)$ for Algorithm 4 using weights:

$$w(1) = 0.779307 \qquad w(2) = 0.920664 \qquad w(3) = 0.997121 \quad \forall_{i \geq 4}\, w(i) = 1$$

The modified algorithm uses only polynomial space for the same reason as in Theorem 3. □

Remark 2 If we would have set $w(i) = 1$ for all $i \geq 3$ as in the proof of Theorem 3 the recurrence relation for $d = r_4 = 4$ becomes independent of the weight function w: $w(i)$ for $i < 3$ does not occur in its formulas. The solution to this recurrence relation is close to $\alpha = 1.3247$ and is independent of w. Hence, $w(3)$ needs to be variable in order to get any solution below 1.3247.

As a consequence we also obtained the following results (see Sect. 2 and Corollary 1):

Corollary 2 *The minimum maximal matching problem can be solved by a modification of Algorithm* 4 *in* $O(1.3226^n)$ *time and polynomial space.*

Corollary 3 *The matrix dominating set problem can be solved by modification of Algorithm* 4 *in* $O(1.3226^{n+m})$ *time and polynomial space on* $n \times m$ *matrices.*

For the matrix dominating set problem a slightly simpler algorithm would suffice since there cannot be any odd cycles in a bipartite graph; removing isolated vertices and 2-cliques from $G[U]$ by a reduction rule would suffice.

We note that the previous fastest algorithm for matrix dominating set by Fomin et al. is faster than the algorithm for minimum edge dominating set on which it was based [12]. Fomin et al. obtain this improvement by noticing that a bipartite graph contains less that $3^{n/3}$ minimal vertex covers. We cannot use this improvement here, because our approach does not use a subroutine that enumerates all minimal vertex covers.

The proof of the lower bound on the running time of Algorithm 3 is no longer valid after introducing the first alternative branching strategy. We prove different lower bounds for the algorithms in the improvement series (also see Table 1):

Proposition 3 *The worst case running time of the* i*-th algorithm in the improvement series is* $\Omega(1.2968^n)$ *if* $i \leq 4$ *and* $\Omega(1.2753^n)$ *if* $i \geq 5$.

Proof If $i \leq 4$, consider the class of graphs consisting of l disjoint copies of Graph 2 in Fig. 1. In this case, the i-th algorithm in the series can branch on the rightmost grey vertex v. When v is put in the independent set, we are left with Subgraph 1 of Fig. 2 in $G[U]$ which generates two subproblems. When v is put in the vertex cover, the algorithm can branch on the leftmost grey vertex. This results in either a path on three vertices (two subproblems) or a cycle of length six (four subproblems) remaining in $G[U]$. Altogether this leads to a total of eight subproblems for each copy consisting of eight vertices. Since $8^l = 8^{n/8} > 1.2968^n$, these algorithms run in time $\Omega(1.2968^n)$.

If $i \geq 5$, then the i-th algorithm in the series uses alternative branching strategies 1 up to $i \geq 5$. In this case, the previous lower bound is no longer valid since the algorithm can no longer branch on the grey vertices: an alternative is introduced by alternative branching strategy 5. Now consider Graph 3 of Fig. 1 in which the algorithm can branch on one of the grey vertices. In the subproblem where this vertex is put in the independent set a star shaped graph remains in $G[U]$ which generates two subproblems by branching on its centre vertex. In the other subproblem the algorithm can branch on the other grey vertex resulting a cycle of length six in $G[U]$ or the removal of this copy of the graph from U. This gives a total sum of seven subproblems on a graph on eight vertices. Since $7^l = 7^{n/8} > 1.2753^n$, these algorithms run in time $\Omega(1.2753^n)$. $\square$

Remark 3 Considering more subcases and deriving more alternative branching strategies could further reduce the running time of the algorithm. But if we continue in the above fashion, we cannot improve beyond $O(1.3214^n)$. This is because, if we branch on a vertex of maximum degree d, then the number of edges d_2 between $N_U[v]$ and the rest of $G[U]$ is bounded from above by $d_2 \leq \sum_{i=1}^{d}(i-1)r_i$. If we solve our quasiconvex program using these maximum values for d_2, while discarding the extra reductions imposed by alternative branching strategies 7 and 8 (these are dominated by increasing d_2), we obtain the running time bound of $O(1.3214^n)$. This makes further subcase analyses almost not worth the effort.

The lower bound of $\Omega(1.2753^n)$ would remain valid for these further improved algorithms. This is the case because d_2 is maximal for the vertices we branch on in the second part of the proof of Proposition 3: non of their neighbours are neighbours of each other.

7 Weighted Edge Domination

Now let us consider the weighted variants of minimum edge dominating set and minimum maximal matching. Proposition 1 still applies when considering these weighted problems, while other properties exploited by our algorithm need more careful consideration. In this section, we introduce modifications of the algorithm of the previous section that solve these weighted problems with the same upper bound on running time. For both variants, we need a slightly different approach.

7.1 Minimum Weight Edge Dominating Set

Let us first look at the polynomial part of the algorithm at the leaves of the search tree. In the unweighted case it is sufficient to compute a minimum edge cover in $G[C]$, but this does not extend to the weighted case; using edges between a vertex in the independent set I and a vertex in the vertex cover C possibly leads to a smaller total weight. To deal with this we notice that the *minimum weight edge cover* problem is solvable in polynomial (cubic) time [23] by using matching techniques.

First consider the *minimum weight generalised edge cover* problem: in a graph G cover a specified subset of the vertices $C \subseteq V$ by a set of edges of minimum total weight. This problem is solvable in cubic time too [24] in the following way [8]. Create the graph G' with vertex set $C \cup \{v\}$, where v is a new vertex. The edges of G' are the edges of $G[C]$ to which we add an edge $\{u, v\}$ for all $u \in C$ with degree zero in $G[C]$ or for which u has an edge in G whose weight is smaller than the weight of each edge incident to u in $G[C]$. The weight of a new edge $\{u, v\}$ will be the minimum weight of all edges incident to u in G.

Fernau [8] used this to prove the following proposition that we will need for our algorithms. We give the proof for completeness.

Proposition 4 (Fernau [8]) *The minimum weight generalised edge cover in G has weight equal to the minimum of the weights of the edge covers in $G[C]$ and G'.*

Proof The minimum weight generalised edge cover in G has weight equal to the minimum weight edge cover in $G[C]$ or G' depending on whether edges with endpoints in $V \backslash C$ are used. This will equal the one with smallest weight, since if no edges incident to a vertex in $V \backslash C$ are used in the minimum weight generalised edge cover in G then the minimum weight edge cover in G' will have greater weight than the one in $G[C]$ (more needs to be covered). Equivalently, if some of these edges are used, then we obtain a solution with smaller weight by using them and hence the minimum weight edge cover in $G[C]$ will have larger weight than the one in G'. □

We now consider Rule 1. Rule 1 no longer applies to the weighted case, and cannot be easily adapted to this case, as it is not possible to assign weights to the new edges it introduces such that we obtain an equivalent instance. However, in the case of cliques of size at most three, the following modified rules can be used:

Rule 2 Put isolated vertices in $G[U]$ in the independent set I.

Rule 3

```
if $G[U]$ contains a connected component $H$ which is a clique of size two or three then
    let $e$ be an edge of minimum weight in $H$
    let $\tilde{G}$ be $G$ with a new vertex $v$ connected by edges of weight $\omega(e)$ to all vertices in $H$
    $\tilde{C} := C \cup H \cup \{v\}$, $\tilde{U} := U \backslash H$
    recursively solve the problem $(\tilde{G}, \tilde{C}, I, \tilde{U})$ and let $D$ be the resulting edge dominating set
    if $D$ contains two distinct edges $f, g$ incident to $v$ then
        return $(D \backslash \{f, g\}) \cup \{e\}$
    return $D \backslash \{f\}$, where $f$ is the unique edge in $D$ incident to $v$
```

Notice that for 2-cliques we can equivalently contract its edge and connect the newly obtained vertex by an edge with weight equal to the contracted edge's weight to a new vertex. This new vertex does not need to be covered by the generalised edge cover.

Proof of Correctness Rule 2 is correct, because all edges incident to an isolated vertex in $G[U]$ have their other endpoint in C and hence will be dominated by an edge incident to this endpoint.

Observe that the edges of the clique H in Rule 3 are dominated in G if at most one vertex in H is not incident to a dominating edge. Thus if one edge in D is incident to v, the returned set is an edge dominating set in G. If two edges $\{u, v\}, \{v, w\}$ in D are incident to v, then e is incident to u or w, because H consists of no more than three vertices. Therefore, as the returned set contains e, we have that it is an edge dominating set in G also.

The returned set is of total weight $(\sum_{d \in D} \omega(d)) - \omega(e)$ and therefore it has minimum weight. This is because if there is an edge dominating set D' in G of smaller weight then we can add an edge e' with weight $\omega(e)$ to D' obtaining a minimum weight edge dominating set in $\tilde{G}$ of smaller total weight than D. Here, e' is the edge joining the one vertex in H not incident to an edge in D' with v, or any edge incident to v if no such vertex exists. □

Algorithm 5 Let Algorithm 5 be obtained from Algorithm 4 by replacing Rule 1 by Rules 2 and 3.

Theorem 5 *Algorithm 5 solves the minimum weight edge dominating set problem in $O(1.3226^n)$ time and polynomial space.*

Proof Correctness follows in exactly the same way as in Theorem 2, based on Theorem 1 and the correctness of the Rules 2 and 3.

The running time is dominated by the exponential number of subproblems generated, because for each partitioning of V in a minimal vertex cover and a maximal independent set, the algorithm computes the minimum weight edge dominating set containing the vertex cover in its set of endpoints in polynomial time. The only difference in the number of subproblems generated compared to Theorem 4 is the removal of cliques of size four and larger by a reduction rule. We have considered these subcases, which are removed by Rule 1 but not by Rules 2 and 3, in the list of alternative branching strategies of Sect. 6. Therefore the upper bound on the running time in the proof of Theorem 4 remains valid for our modified algorithm. □

7.2 Minimum Weight Maximal Matching

We have given modified versions of Algorithm 4 for both the minimum maximal matching problem, and the minimum weight edge dominating set problem. These modifications cannot be combined to construct an algorithm for the minimum weight maximal matching problem (minimum weight independent edge dominating set problem) since the transformation of Corollary 1 does not preserve edge weights.

However, for a minimal vertex cover C in a graph G we can construct the minimum weight maximal matching containing C in its set of endpoints. To this end

Algorithm 6 Algorithm for Minimum Weight Generalised Independent Edge Cover

```
Input: a graph $G = (V, E)$ and a subset of its vertices $C \subseteq V$
Output: a minimum weight generalised independent edge cover of $C$ in $G$ if one
    exists
1: if $G$ has an odd number of vertices then
2:     add a new vertex $v$ to $G$ ($v \notin C$)
3: for all $v, w \in V \setminus C$, $v \neq w$ do
4:     add a new edge between $v$ and $w$ to $G$ with zero weight
5: if there exists a minimum weight perfect matching $P$ in $G$ then
6:     return $P$ with all edges between vertices not in $C$ removed
7: return false
```

we consider the *minimum weight generalised independent edge cover* problem: in a graph G cover a specified subset of the vertices $C \subseteq V$ by a set of edges of minimum total weight such that no two edges are incident to the same vertex.

Proposition 5 *Algorithm 6 solves the minimum weight generalised independent edge cover problem in polynomial time.*

Proof The returned edge set is a generalised independent edge cover of C in G since it is a matching and it contains all vertices in C in its set of endpoints.

Consider any generalised independent edge cover D of C in G. We notice that we can extend D to a perfect matching P' in G with the added edges and the possibly added vertex v because all vertices not incident to an edge in D are adjacent as they are not in C, and there are an even number of vertices as we add v if this was not the case. This perfect matching P' has the same total weight as D since the added edges all have zero weight.

The returned generalised independent edge cover has the same weight as the computed perfect matching P. Because P is of minimum total weight, and all generalised independent edge covers of C in G can be transformed into a matching of equal total weight by using the above construction, the returned set is a minimal weight generalised independent edge cover of C in G. *False* is only returned if no generalised independent edge cover of C exists in G. □

Now we again only have to consider our reduction rule:

Rule 4

```
if $G[U]$ contains a connected component $H$ which is a clique then
    let $\tilde{G}$ be $G$ with a new vertex $v$ connected by edges of zero weight to all vertices
    in $H$
    $\tilde{C} := C \cup H$; $\tilde{I} := I \cup \{v\}$; $\tilde{U} := U \setminus H$
    recursively solve the problem $(\tilde{G}, \tilde{C}, \tilde{I}, \tilde{U})$ and let $D$ be the resulting edge dom-
    inating set
    if $D$ contains an edge $e$ incident to $v$ then
        return $D \setminus \{e\}$
    return $D$
```

Proof of Correctness In a clique a maximum of one vertex is allowed not to be incident to a dominating edge. Since all vertices in H are put in $\tilde{C}$, and in $\tilde{C}$ at most one edge can be incident to v, the returned edge set is an independent edge dominating set. This returned independent edge dominating set has the same total weight as D. Therefore it is of minimal total weight: if an independent edge dominating set of smaller weight would exist, a minimum weight maximal matching in $\tilde{G}$ with smaller weight than D can be constructed. □

Algorithm 7 Let Algorithm 7 be obtained from Algorithm 4 by replacing Rule 1 by Rule 4.

Theorem 6 *Algorithm 7 solves the minimum weight maximal matching problem in* $O(1.3226^n)$ *time and polynomial space.*

Proof Identical to Theorem 5 using Proposition 5 and the proof of correctness of Rule 4. □

7.3 Parametrised Minimum Weight Maximal Matching

The results from Sect. 7.2 also solve a question raised by Fernau [8] for the parametrised version of this problem. He asks whether vertex cover structures can be exploited to obtain efficient parametrised algorithms for this problem. We combine Algorithm 6 with the parametrised algorithm from [12]. This gives us the fastest known algorithm for this problem: Algorithm 8. The algorithm uses the *global application of width parameters approach* of [12], which combines a branching approach with pathwidth based techniques.

The *parameterised minimum weight maximal matching* is defined: given a graph G and a parameter k, find a minimum weight maximal matching in G of weight at most k. In order to compare weights to the parameter, it is required that for every input edge e: $\omega(e) \geq 1$. Alternatively one could ask for the minimum weight maximal matching if it consists of at most k edges.

First we need some basic results on pathwidth, path decompositions and dynamic programming. For the definitions of these concepts see for example [2, 18]. We start with the following result due to Fomin et al. [12].

Lemma 2 (Fomin et al. [12]) *If in any node of the branching tree of Algorithm 8, $G[U]$ has maximum degree two, then the pathwidth of G is bounded by $|C| + 2$ and a path decomposition of this width can be found in polynomial time.*

Proof Let C, I, U be the partitioning of the vertices of G in a node of the branching tree. $G[U]$ has maximum degree two; therefore it has a path decomposition of width at most two. Because I is an independent set, and non of the neighbours of the vertices in I are in U, the pathwidth of $G[U \cup I]$ equals the pathwidth of $G[U]$. Such a path decomposition of $G[U \cup I]$: $(X_1, X_2, \dots, X_r)$ of width at most two can be computed in polynomial time. Now add C to every X_i to obtain a path decomposition of G of width at most $|C| + 2$. □

Algorithm 8 Algorithm for Parametrised Minimum Weight Maximal Matching

```
Input: a graph $G = (V, E)$ and a parameter $k$
Output: a minimum weight maximal matching of weight at most $k$ in $G$ if one exists
1: $I := \emptyset$; $C := \emptyset$; $U := V$
2: if $|C| > 2k$ then
3:     return false
4: else if $|C| \leq 0.8036k$ and $G[U]$ is of maximum degree two then
5:     compute the minimum weight maximal matching $M$ in $G$ by dynamic programming over a path decomposition of $G$ by using Lemmas 2 and 3
6:     stop the algorithm: do not backtrack!
7:     return $M$ if it is of total weight at most $k$ or false otherwise
8: else if a vertex $v$ of maximum degree in $G[U]$ has $U$-degree at least two then
9:     create two subproblems and solve each one recursively:
10:        1: $(G, C \cup N_U(v), I \cup \{v\}, U \backslash N_U[v])$    2: $(G, C \cup \{v\}, I, U \backslash \{v\})$
11: else
12:     exhaustively apply Rule 4 {this results in: $U = \emptyset$}
13:     Let $M$ be a minimum weight generalised independent edge cover of $(G, C)$ (Algorithm 6)
14:     return $M$ if it is of total weight at most $k$ or false otherwise
```

Lemma 3 *Given a graph G and a path decomposition $(X_1, X_2, \ldots, X_r)$ of G of width at most p, the minimum weight maximal matching of G can be found in $O^*(3^p)$*[1] *time and space.*

Proof We only sketch the proof, which uses standard dynamic programming techniques for tree and path decompositions. (E.g., compare with [1, 2, 31].) In $O(n)$ time, we can transform the path decomposition into a nice path decomposition, see [18]. This is a path decomposition with $r = O(n)$, and for each $i \geq 2$, there is a $v \in V$ with $X_i = X_{i-1} \cup \{v\}$ or $X_{i-1} = X_i \cup \{v\}$. Now, for each i, $1 \leq i \leq r$, let $V_i = X_1 \cup \cdots \cup X_i$, and let $G_i = G[V_i]$ be the subgraph of G, induced by V_i. A *partial solution* in G_i is a set of edges $F \subseteq E \cap (V_i \times V_i)$, such that for each $v \in V_i - X_i$, either v is an endpoint of an edge in F, or all the neighbours of v are endpoints of edges in F. An *extension* of a partial solution F is a maximal matching obtained by adding edges not in G_i to F.

Each vertex v in X_i has a *state* with respect to partial solution F: state A if v is endpoint of an edge in F; state B if v is not endpoint of an edge in F and v has a neighbour in $V_i - X_i$ that is not endpoint of an edge in F, and state C otherwise. When a vertex $v \in X_i$ has state B then an extension of F must include an edge with v as endpoint. The *characteristic* of a partial solution F is the collection of states for all $v \in X_i$, so there are at most 3^{p+1} states for each X_i. It can be shown that partial solutions with the same characteristic are equivalent, in the sense that an extension of one also gives an extension of the other; the same edges are added.

[1]Here we use the O^* notation which suppresses not only constant but all polynomial parts of the running time.

The algorithm thus tabulates for each characteristic the minimum weight of a partial solution F with that characteristic. Simple but tedious case analysis shows that we can compute, for each i, $1 \leq i \leq r$, in $O^*(3^p)$ time this table for X_i, given the table for X_{i-1}. Inspection of the table for X_r gives the answer: take the minimum value over all characteristics without state B vertices, and without adjacent vertices both with state C. □

Now we can prove the following running time for Algorithm 8 in a way similar to [12].

Theorem 7 *Algorithm 8 solves the parametrised minimum weight maximal matching problem in $O^*(2.4179^k)$ time and space.*

Proof Correctness is trivial if a path decomposition is constructed by the algorithm: it then ignores any branching done and outputs a minimum weight maximal matching of G if it is of small enough weight. If no path decomposition is constructed the algorithm considers all minimal vertex covers of size at most $2k$. If a maximal matching is returned, it is of minimum weight for the same reason as in Theorem 6. If no maximal matching is computed, then $|C| > 2k$ for all minimal vertex covers C in G, and because $\forall_{e \in E} w(e) \geq 1$ any maximal matching is of weight at least k.

Let $S(k)$ be the number of subproblems generated to solve a problem with parameter k, and let $\alpha = 0.4018$. If we branch on a vertex of U-degree at least three, the behaviour of the algorithm corresponds to the recurrence relation $S(k) \leq S(k - \frac{1}{2}) + S(k - 1\frac{1}{2})$. This is because the algorithm stops if $|C| > 2k$ and $|C|$ increases by one in one branch and by at least three in the other. Solving the recurrence relation leads to a running time of this part of the algorithm of $O^*(2.1480^k)$.

Now suppose that during the execution of the algorithm a path decomposition of width p is computed. This happens when the maximum degree in $G[U]$ first becomes at most two. Then a minimum weight maximal matching in G is computed in $O^*(3^{|C|}) \leq O^*(3^{2\alpha k}) \leq O^*(2.4179^k)$ time. This leads to a total running time of $O^*(2.1480^k + 2.4178^k) = O^*(2.4179^k)$.

If no path decomposition is computed, we have that $|C| > 2\alpha k$ in every branch in which the maximum degree in $G[U]$ first becomes at most two. Hereafter, the algorithm performs a series of branchings on degree two vertices according to the recurrence relation $S(k) \leq S(k - \frac{1}{2}) + S(k - 1)$ ($|C|$ increases by one or two). This recurrence relation solves to $O^*(2.6180^k)$. When the maximum degree in $G[U]$ becomes one, the minimum weight maximal matching for this branch is computed in polynomial time. In the first branching steps until $|C| > 2\alpha k$, the algorithm branches on vertices of degree at least three generating at most $2.1480^{\alpha k}$ subproblems with parameter at most $k - \alpha k$. Hereafter, it branches on other vertices solving these subproblems in time at most $2.6180^{k-\alpha k}$. Together, this leads to a total running time of $O^*(2.1480^{\alpha k} 2.6180^{k-\alpha k}) = O^*(2.4179^k)$.

Notice that α was chosen in such a way that $2.1480^{\alpha k} 2.6180^{k-\alpha k} = 3^{2\alpha k}$. □

8 Conclusion and Further Research

We have presented $O(1.3226^n)$ time and polynomial space algorithms for minimum edge dominating set, minimum weight edge dominating set, minimum maximal matching (minimum independent edge dominating set), and minimum weight maximal matching. These algorithms are obtained by using a vertex cover structure on the input graph, special branching strategies and reduction rules applied to simple instances and the iterative improvement of a measure and conquer analysis. We have also created an $O(1.3226^{n+m})$ algorithm for matrix dominating set and an $O^*(2.4179^k)$ algorithm for parametrised minimum weight maximal matching.

It would be interesting to see if there are more related problems, for example minimum (weight) total edge dominating set, to which similar methods can be applied. This is not straightforward as in this total domination problem, every dominating edge does not dominate itself and thus needs to share an endpoint with another dominating edge. This makes it appear hard to design algorithms in which the total edge dominating set is based on a matching for this problem.

We note that our algorithms have their running times expressed in the number of vertices n in the input graph G, instead of the number of edge m in G. As an interesting research topic, we mention the analysis and design of exact algorithms for edge domination (and other problems), where we focus on the running time as function of the number of edges m. See the discussion about complexity parameters in [35].

We realise that the upper bounds on the running times proved using measure and conquer are not tight. Therefore we also derived lower bounds. It is an interesting question what the exact worst case behaviour of our algorithms is. We know of no method that can derive such exact worst case behaviour systematically, but if this is possible it would be very interesting to see whether this results in a similarly iterative improvement methodology.

References

1. Alber, J., Bodlaender, H.L., Fernau, H., Kloks, T., Niedermeier, R.: Fixed parameter algorithms for dominating set and related problems on planar graphs. Algorithmica **33**, 461–493 (2002)
2. Bodlaender, H.L.: Treewidth algorithmic techniques and results. In: Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS 1997. Lecture Notes in Computer Science, vol. 1295, pp. 19–36. Springer, Berlin (1997)
3. Carr, R., Fujito, T., Konjevod, G., Parekh, O.: A 2 1/10-approximation algorithm for a generalization of the weighted edge-dominating set problem. J. Comb. Optim. **5**, 317–326 (2001)
4. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness. Congr. Numer. **87**, 161–178 (1992)
5. Edmonds, J.: Paths, trees, and flowers. Can. J. Math. **17**, 445–467 (1965)
6. Eppstein, D.: Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. ACM Trans. Algorithms **2**, 492–509 (2006)
7. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**, 634–652 (1998)

8. Fernau, H.: Edge dominating set: efficient enumeration-based exact algorithms. In: Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006. Lecture Notes in Computer Science, vol. 4169, pp. 140–151. Springer, Berlin (2006)
9. Fomin, F.V., Kratsch, D., Woeginger, G.J.: Exact (exponential) algorithms for the dominating set problem. In: Proceedings 30th International Workshop on Graph-Theoretic Concepts in Computer Science, WG 2004. Lecture Notes in Computer Science, vol. 3353, pp. 245–256. Springer, Berlin (2004)
10. Fomin, F.V., Grandoni, F., Kratsch, D.: Some new techniques in design and analysis of exact (exponential) algorithms. Bull. Eur. Assoc. Theor. Comput. Sci. **87**, 47–77 (2005)
11. Fomin, F.V., Todinca, I., Kratsch, D., Villanger, Y.: Exact algorithms for treewidth and minimum fill-in. SIAM J. Comput. **38**, 1058–1079 (2008)
12. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica **54**, 181–207 (2009)
13. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. ACM **56** (2009)
14. Harary, F.: Graph Theory. Addison-Wesley, Reading (1969)
15. Held, M., Karp, R.: A dynamic programming approach to sequencing problems. J. SIAM **10**, 196–210 (1962)
16. Iwama, K.: Worst-case upper bounds for ksat. Bull. Eur. Assoc. Theor. Comput. Sci. **82**, 61–71 (2004)
17. Johnson, D.S., Yannakakis, M., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. **27**, 119–123 (1988)
18. Kloks, T.: Treewidth. Computations and Approximations. Lecture Notes in Computer Science, vol. 842. Springer, Berlin (1994)
19. Kullmann, O.: New methods for 3-SAT decision and worst-case analysis. Theor. Comput. Sci. **223**(1–2), 1–72 (1999)
20. Lawler, E.L.: A note on the complexity of the chromatic number problem. Inf. Process. Lett. **5**, 66–67 (1976)
21. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. SIAM J. Comput. **9**, 558–565 (1980)
22. Moon, J.W., Moser, L.: On cliques in graphs. Isr. J. Math. **3**, 23–28 (1965)
23. Plesník, J.: Equivalence between the minimum covering problem and the maximum matching problem. Discrete Math. **49**, 315–317 (1984)
24. Plesník, J.: Constrained weighted matchings and edge coverings in graphs. Discrete Appl. Math. **92**, 229–241 (1999)
25. Raman, V., Saurabh, S., Sikdar, S.: Efficient exact algorithms through enumerating maximal independent sets and other techniques. Theory Comput. Syst. **42**, 563–587 (2007)
26. Razgon, I.: Exact computation of maximum induced forest. In: Proceedings of the 10th Scandinavian Workshop on Algorithm Theory, SWAT 2006. Lecture Notes in Computer Science, vol. 4059, pp. 160–171. Springer, Berlin (2006)
27. Robson, J.M.: Algorithms for maximum independent sets. J. Algorithms **7**, 425–440 (1986)
28. Schiermeyer, I.: Efficiency in exponential time for domination-type problems. Discrete Appl. Math. **156**, 3291–3297 (2008)
29. Schöning, U.: Algorithmics in exponential time. In: Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science, STACS 2005. Lecture Notes in Computer Science, vol. 3404, pp. 36–43. Springer, Berlin (2005)
30. Tarjan, R.E., Trojanowski, A.: Finding a maximum independent set. SIAM J. Comput. **6**, 537–546 (1977)
31. Telle, J.A., Proskurowski, A.: Algorithms for vertex partitioning problems on partial k-trees. SIAM J. Discrete Math. **10**, 529–550 (1997)
32. van Rooij, J.M.M.: Exact exponential-time algorithms for domination problems in graphs. Ph.D. thesis, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands (2011)
33. van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer: a faster exact algorithm for dominating set. In: Proceedings of the 25st Symposium on Theoretical Aspects of Computer Science, STACS 2008, pp. 657–668 (2008)
34. van Rooij, J.M.M., Nederlof, J., van Dijk, T.C.: Inclusion/exclusion meets measure and conquer: exact algorithms for counting dominating sets. In: Proceedings of the 17th Annual European Symposium on Algorithms, ESA 2009. Advanced Research in Computing and Software Science, Lecture Notes in Computer Science, vol. 5757, pp. 554–565. Springer, Berlin (2009)

35. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Combinatorial Optimization: “Eureka, You Shrink”. Lecture Notes in Computer Science, vol. 2570, pp. 185–207. Springer, Berlin (2003)
36. Woeginger, G.J.: Space and time complexity of exact algorithms: some open problems. In: Proceedings 1st International Workshop on Parameterized and Exact Computation, IWPEC 2004. Lecture Notes in Computer Science, vol. 3162, pp. 281–290. Springer, Berlin (2004)
37. Yannakakis, M., Gavril, F.: Edge dominating sets in graphs. SIAM J. Appl. Math. **38**, 364–372 (1980)