Algorithmica (2012) 62:382–415
DOI 10.1007/s00453-010-9460-7

Fast Algorithms for MAX INDEPENDENT SET

Nicolas Bourgeois · Bruno Escoffier · Vangelis T. Paschos · Johan M.M. van Rooij

Received: 19 June 2009 / Accepted: 4 October 2010 / Published online: 19 October 2010

Abstract We first propose a method, called "bottom-up method" that, informally, "propagates" improvement of the worst-case complexity for "sparse" instances to "denser" ones and we show an easy though non-trivial application of it to the MIN SET COVER problem. We then tackle MAX INDEPENDENT SET. Here, we propagate improvements of worst-case complexity from graphs of average degree d to graphs of average degree greater than d. Indeed, using algorithms for MAX INDEPENDENT SET in graphs of average degree 3, we successively solve MAX INDEPENDENT SET in graphs of average degree 4, 5 and 6. Then, we combine the bottom-up technique with measure and conquer techniques to get improved running times for graphs of maximum degree 5 and 6 but also for general graphs. The computation bounds obtained for MAX INDEPENDENT SET are $O^*(1.1571^n)$, $O^*(1.1895^n)$ and $O^*(1.2050^n)$, for graphs of maximum (or more generally average) degree 4, 5 and 6 respectively, and $O^*(1.2114^n)$ for general graphs. These results improve upon the best known results for these cases for polynomial space algorithms.

Keywords Bottom-up method · Max independent set · Exact algorithms

Research supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010. Parts of this paper have been presented at SWAT'10, LNCS 6139 and at TAMC'10, LNCS 6108.

N. Bourgeois · B. Escoffier (✉) · V.T. Paschos
LAMSADE, Université Paris-Dauphine, Place du Maréchal de Lattre de Tassigny,
75775 Paris Cédex 16, France
e-mail: escoffier@lamsade.dauphine.fr

N. Bourgeois
e-mail: bourgeois@lamsade.dauphine.fr

V.T. Paschos
e-mail: paschos@lamsade.dauphine.fr

J.M.M. van Rooij
Department of Information and Computing Sciences, Universiteit Utrecht, Utrecht, The Netherlands
e-mail: johanvr@cs.uu.nl

1 Introduction

Very active research has been recently conducted around the development of exact algorithms for **NP**-hard problems with non-trivial worst-case complexity (see the seminal paper [20] for a survey on both methods used and results obtained). Among the problems studied in this field, MAX INDEPENDENT SET (and particular versions of it) is one of those that have received a very particular attention and mobilized numerous researchers.

We propose in Sect. 2 a generic method that propagates improvements of worst-case complexity from "sparse" instances to "denser" (less sparse) ones, where the density of an instance is proper to the problem handled and refers to the average value of some parameter of the input. We call this method "bottom-up method". The basic idea here has two ingredients: (i) the choice of the recursive complexity measure and (ii) a way to insure that on "denser" instances, a good branching (with respect to the chosen complexity measure) occurs.

We first illustrate our method on MIN SET COVER. Given a ground set $\mathcal{U}$ on n elements and a set-system $\mathcal{S}$ over $\mathcal{U}$, MIN SET COVER consists of determining a minimum-size subsystem $\mathcal{S}'$ covering $\mathcal{U}$. Here, the density of a MIN SET COVER-instance is defined to be the average cardinality of the sets in the set-system $\mathcal{S}$. Application of the method to MIN SET COVER is rather direct (and might be improved using a deeper analysis) but it produces quite interesting results. As we show in Sect. 2.1, it outerperforms the results of [18] in instances with average-set sizes $6, 7, 8, \ldots$. Let us note that we are not aware of results better than those given here.

We next handle the MAX INDEPENDENT SET problem. Given a graph $G = (V, E)$, MAX INDEPENDENT SET consists of finding a maximum-size subset $V' \subseteq V$ such that for any $(v_i, v_j) \in V' \times V'$, $(v_i, v_j) \notin E$. For this problem, [9] proposes an algorithm with worst-case complexity bound $O^*(1.2201^n)$ using polynomial space. Very recently this has been improved using a new branching and a more involved (computer-aided) case analysis down to $O^*(1.2132^n)$ in [13]. All the results we present here are polynomial space algorithms. We also quote the $O^*(1.2108^n)$ time bound in [16] using exponential space (claimed to be improved down to $O^*(1.1889^n)$ in the technical report [17], still using exponential space). Dealing with MAX INDEPENDENT SET in sparse graphs faster and faster algorithms have been devised for optimally solving this problem. The most important milestones of these improvements are the papers of [1, 4, 5, 7, 10, 14]. In particular, in graphs of maximum degree 3, the best known result is the recent $O^*(1.0892^n)$ algorithm in [15]. As a first result, we improve in this article (Sect. 3) the bound of [15] down to $O^*(1.08537^n)$ by proposing a finer and more detailed case analysis based upon powerful reduction rules. Our result remains valid also for connected graphs of average degree bounded above by 3. For MAX INDEPENDENT SET, density of the input graph is measured by its average degree. So, the bottom-up method here extends improvements of the worst-case complexity in graphs of average degree d to graphs of average degree greater than d.

In order to informally sketch this method in the case of MAX INDEPENDENT SET, suppose that one knows how to solve it on graphs with average degree d in time $O^*(\gamma_d^n)$. Solving the problem on graphs with average degree $d' > d$ is based

upon two ideas. We first look for a running time expression of the form $\alpha^m \beta^n$, where α and β depend both on the input graph (more precisely on its average degree), *and on the value* γ_d (see Sect. 2). In other words, the form of the running time we seek is parameterized by what we already know on graphs with smaller average degrees. Next, according to this form, we identify particular values d_i (not necessarily integer) of the average degree that ensure that a "good" branching occurs. This allows us to determine a good running time for increasing values of the average degree. Note also that a particular interest of this method lies in the fact that any improvement on the worst-case complexity on graphs of average degree 3 immediately entails improvements for higher average degrees. A direct application of this method, for instance, in MAX INDEPENDENT SET in graphs with average degree 4, leads to an upper complexity bound of $O^*(1.1707^n)$ (Sect. 2.2). This result is further improved in Sect. 4 down to $O^*(1.1571^n)$ with a more refined case analysis; that outperforms the bound of $O^*(1.1643^n)$ that can be derived from [13] (let us note that the bound in graphs of degree at most d is obtained as $O^*(1.2201^{nw_d})$, where w_d is the weight associated to vertices of degree d in the measure and conquer analysis of [13]; It might be possible that better bounds could be achieved with this method, but this is not straightforward, since reduction rules may create vertices of arbitrarily large degree, see Sect. 5). We also provide bounds for (connected) graphs with average degree at most 5 and 6.

The idea of propagating results from particular cases to more general ones is classical and old in mathematics and in theoretical computer science. For instance, many approximability preserving reductions are based upon this general idea. In the domain of exact algorithms, a natural instantiation of it, is the propagation of fast solutions for sparse instances to denser ones. Several methods of such propagation have been developed mainly in [6, 8, 11, 12, 19]. Most of them are designed for counting satisfiability models and are very different from the ours. Our method has some similarities with that of [6], although they are written and presented differently, and applied in different settings. The method of [6] is used for counting models for 2 and 3SAT formulæ, while the ours is designed and applied for solving optimization problems. Furthermore, the method presented here has a general scope and can be applied for several optimization problems as we show in the sequel.

In Sect. 5, we combine measure and conquer with bottom-up in order to show that MAX INDEPENDENT SET can be optimally solved in $O^*(1.2114^n)$ in general graphs, thus improving the best known published result of $O^*(1.2132^n)$ [13] (for polynomial space algorithms). Furthermore, in graphs of maximum degree at most 5 and 6, we provide bounds of $O^*(1.1895^n)$ and $O^*(1.2050^n)$, respectively, that improve upon the respective $O^*(1.1948^n)$ and $O^*(1.2084^n)$ bounds of [13].

Note that using previously known bounds for solving MAX INDEPENDENT SET in graphs of degree 3 (worse than $O^*(1.08537^n)$), the bottom-up method would also lead to improved results (with respect to those known in the literature). We illustrate this point in Table 1 where several results are given for graphs of degree 4, 5, 6 and in general graphs depending on the bounds used for graphs of degree 3.

Eventually more interesting than the improvements themselves is the fact that these bounds are obtained via a method that, once some possibly long case analysis has been performed on instances of small density, it is directly applicable for getting results on higher density instances, even on general ones (i.e., when no bound

Table 1 MAX INDEPENDENT SET results for graphs of degree 4, 5, 6 and general graphs with starting points several complexity bounds for graphs of degree 3

Degree 3	Degree 4	Degree 5	Degree 6	General graphs
1.08537	1.1571	1.1895	1.2050	1.2114
1.0892 [15]	1.1594	1.1908	1.2060	1.2120
1.0919 [21]	1.1614	1.1918	1.2168	1.2125

is considered on the density). We think that this method deserves further attention and insight, since it might be used to solve also other problems as it is already shown in Sect. 2.1.

Throughout this paper, we will denote by $N(u)$ and $N[u]$ the neighborhood and the closed neighborhood of a vertex u in a graph, respectively ($N(u) = \{v \in V : (u, v) \in E\}$ and $N[u] = N(u) \cup \{u\}$).

2 The Bottom-Up Method

The bottom-up method relies on two stepping stones:

1. the choice of the recursive complexity measure; in Sect. 2.1 we show how the choice of a smart measure for MIN SET COVER leads to non trivial time bounds for instances of bounded average set-cardinalities;
2. a way to guarantee that once an upper complexity bound is got on instances of density d, a good branching (with respect to the chosen complexity measure) occurs on instances of density greater that d; the importance of this point is illustrated in Sect. 2.2 for MAX INDEPENDENT SET.

In what follows in this section, we apply the bottom-up method to MIN SET COVER and to MAX INDEPENDENT SET.

2.1 The Importance of Recursive Complexity Measure: The Case of MIN SET COVER

Let us consider the MIN SET COVER problem with ground set $\mathcal{U} = \{x_1, \ldots, x_n\}$ and set system $\mathcal{S} = \{S_1, \ldots, S_m\}$. We show that the bottom-up method easily applies to get improved time bounds in instances with sets of average (or maximum) size d, for any $d \geq 5$. In what follows, we will denote by p the number of edges in the bipartite representation of the instance, i.e., $p = \sum_{i=1}^{m} |S_i|$. Given an instance $(\mathcal{S}, \mathcal{U})$ of MIN SET COVER, its bipartite representation is a bipartite graph $B(S, U, E)$ with $|U| = n$, $|S| = m$ and $E = \{(s_i, u_j) : s_i \in S, u_j \in U, x_j \in S_i\}$. Note that the average set-cardinality corresponds to the average degree of the S-vertices of this graph.

Lemma 1 *The algorithm in* [18] *exactly solves* MIN SET COVER *in time* $O^*(1.55^m)$, $O^*(1.63^m)$, $O^*(1.71^m)$ *and* $O^*(1.79^m)$ *in instances with sets of average size* 5, 6, 7 *and* 8, *respectively.*

Proof Denoting by n_i the number of sets of size i and by m_j the number of elements of frequency j (the frequency of an element is the number of sets in which it is contained, or, in other words, frequencies are the degrees of the U-vertices of B), [18]

gives an algorithm running in time $O^*(1.2302^{k(I)})$ where $k(I) = \sum_{i\geq 1} w_i n_i + \sum_{j\geq 1} v_j m_j$; w_i is the weight associated to a set of size i and v_j is the weight associated to an element of frequency j. It is easy to see that, by convexity (the weights w_i are $(0, 0.3755, 0.7509, 0.9058, 0.9720, 0.9982)$ for $i = 1, \ldots, 6$ and 1 for $i \geq 7$), if the average size of sets is d, then $\sum_{i\geq 1} w_i n_i \leq m w_d$. Moreover, note that $v_j/j \leq v_3/3$ for all $j \geq 1$ (the weights v_j are $(0, 0.3755, 0.7509, 0.9058, 0.9720, 0.9982)$ for $j = 1, \ldots, 6$ and 1 for $j \geq 7$), hence:

$$\sum_{j\geq 1} v_j m_j \leq \frac{v_3}{3} \sum_{j\geq 1} j m_j = \frac{dmv_3}{3}$$

We so get $k(I) \leq m(w_d + v_3 d/3)$ (this bound is tight, if all sets have size d and all elements have frequency 3). □

Let us consider an instance with $p > dm$. The bottom-up method assumes that we know how to solve the problem in instances with sets of average size d, in time $O^*(\gamma_d^m)$. It seeks a complexity of the form $O^*(\gamma_d^m y^{p-dm})$ (valid by hypothesis for $p = dm$). Consider a set S of size $s \geq d + 1$. We branch on S. If we do not take it in the solution, we remove one set and $s \geq d + 1$ edges; suppose that we take S in the solution and that each element in its neighborhood has frequency at least 3 (the other cases regarding frequency are examined below). Then we remove one set and (at least) $3s \geq 3(d + 1)$ edges. Hence, for the complexity to be valid we have to choose y such that:

$$\begin{aligned} &\gamma_d^m y^{p-dm} \geq \gamma_d^{m-1} y^{p-(d+1)-d(m-1)} + \gamma_d^{m-1} y^{p-3(d+1)-d(m-1)} \\ &\iff \quad 1 \geq \gamma_d^{-1} y^{-1} + \gamma_d^{-1} y^{-(2d+3)} \end{aligned}$$

Taking for instance $\gamma_5 = 1.55$ (Lemma 1), this is true for $y = 1.043$. Let us check the other cases:

- If there exists an element j of frequency 1 in S, we have to take S and we remove one set and $s \geq d + 1$ edges. This does not create any problem as long as:

$$\gamma_d^{m-1} y^{(p-d-1)-d(m-1)} \leq \gamma_d^m y^{p-dm}$$

  i.e., $y \leq \gamma_d$.
- Otherwise, if there is an element j of frequency 2, denote by S' the other set j belongs to. Then, either we take S and remove 1 set and at least $2(d + 1)$ edges, or we remove S and take S', removing so 2 sets and at least $d + 2$ edges. Hence, we have to verify that the value of y computed in the general case verifies:

$$1 \geq \gamma_d^{-1} y^{-(d+2)} + \gamma_d^{-2} y^{-2+d}$$

Then, if sets have average size $d+1$, since $p = (d+1)m$, the complexity is $O^*(\gamma_{d+1}^m)$ with $\gamma_{d+1} = \gamma_d \times y$. Starting from $\gamma_5 = 1.51$, the recurrences give $\gamma_6 = 1.61$, $\gamma_7 = 1.66$ (with $y = 1.031$) and $\gamma_8 = 1.70$ (with $y = 1.024$).

Theorem 1 MIN SET COVER *is solvable in times* $O^*(1.61^m)$, $O^*(1.66^m)$ *and* $O^*(1.70^m)$ *in instances with sets of average size* 6, 7 *and* 8, *respectively.*

Interestingly enough, the bound of $O^*(1.2302^{m(w_d+v_3 d/3)})$ obtained in Lemma 1 is bigger than 2^m for $d \geq 11$ while using the bottom-up method, it can be shown that $\gamma_d < 2$ for any d (for instance, $\gamma_{100} < 1.98$). Of course, the analysis conducted in [18] did not handle in a particular way the case of bounded size sets.

2.2 Making a Good Branching: The Case of MAX INDEPENDENT SET

MAX INDEPENDENT SET has been studied in several articles both in the general case and in upper bounded degree cases; in these articles, well known reduction rules have been proposed, among which:

1. (*Degree 0 and 1*) If v is a vertex of degree at most 1, then v can be taken in the solution without branching. This is trivial if v is an isolated vertex. If v has degree 1, then taking its neighbor u is never interesting.
2. (*Vertex folding*) If a vertex v has degree 2, let u_1 and u_2 be its two neighbors. Then it is never interesting to take only one vertex among u_1, u_2 (take v instead). If they are adjacent we take v without branching, otherwise we can remove from the graph v, u_1 and u_2 and add a new vertex u_{12} whose neighborhood is $N(u_1) \cup N(u_2) \setminus \{v\}$, without branching (this reduction is called vertex folding, see for instance [9]).
3. (*Domination*) If v and u are such that $N[u] \subseteq N[v]$, then we way that u dominates v, and we can remove v from the graph without branching.

Note that after the first two reductions rules have been performed, each vertex in the resulting graph has degree at least 3.

The basic ingredient of the bottom-up method is an efficient algorithm working in graphs with low average degree. For MAX INDEPENDENT SET, several articles consider the case of graphs of degree 3. The best known bound published so far is the $O^*(1.0892^n)$ bound in [15]. As a first result, we improve this bound down to $O^*(1.08537^n)$, valid for (connected) graphs of average degree at most 3. Our algorithm is a branching algorithm. It exploits several well known reduction rules (including the ones mentioned above) that simplify the instance, and then looks for a suitable structure in the graph and branches on it: it generates a series of subproblems that are solved recursively and from which the largest solution is returned. We analyse it using as a complexity measure $m - n$, as in previous articles [2, 10], and get a running time (valid in any graph) in $O^*(1.1780^{m-n})$, thus $O^*(1.08537^n)$ in graphs of average degree at most 3.

Theorem 2 *There exists an algorithm solving* MAX INDEPENDENT SET *on connected graphs of average degree at most* 3 *in* $O^*(1.08537^n)$ *time and linear space.*

The detailed analysis is quite long and technical. We give it in Sect. 3.

Let us now show how we can use this algorithm and the bottom-up method for solving MAX INDEPENDENT SET in graphs of degree higher than 3. All the calculi

will be done using the bound of 1.08537, but as pointed out in the introduction interesting results would be also get using a worse bound (Table 1). In order to use efficiently the complexity measure introduced for MIN SET COVER (Sect. 2.1), we prove that, in a graph whose average degree is bounded by some constant (possibly non-integer), we are sure that there exists a rather dense local configuration we can branch on. More precisely, if the average degree is greater than d, this implies that we can find a vertex v with at least $f(d)$ edges incident to some neighbor of v, for some increasing function f.

Let us give an example. In the following we will assume that reductions rules mentioned above have been performed, i.e., the graph does not contain neither vertices of degree at most 2 nor dominated vertices. Then, trivially, if the graph has average degree greater than $d \in \mathbb{N}$, we know there exists a vertex v of degree (at least) $d+1$. Since no vertex is dominated, then there exist at least $f(d) = 2(d+1) + \lceil (d+1)/2 \rceil$ edges incident to some neighbors of v. Indeed, there exist $d+1$ edges incident to v, $d+1$ edges between a neighbor of v and a vertex not neighbor of v (one for each neighbor of v, to avoid domination) and, since each vertex has degree at least 3, at least $\lceil (3(d+1) - 2(d+1))/2 \rceil = \lceil (d+1)/2 \rceil$ other edges. Note that such relationships may be established even if d is non integer. For instance, we will see that if $d > 24/7$, then there exists a vertex of degree 5 or two adjacent vertices having degree 4, leading to $f(d) = 11$. This property linking the average degree to the quality of the branching is given in Lemma 2.

Then, for a given d, either the average degree is greater than d, and we can make an efficient branching (i.e., a branching that induces a recurrence relation leading to a lower time-bound), or it is smaller than d and we can use an algorithm tailored for low-degree graphs. Thus, Lemma 2 fixes a set of critical degrees (d_i) and we define step-by-step (from the smallest to the highest) algorithms $\texttt{STABLE}(d_i)$, that work on graphs of average degree d_i or less. With this lemma, we analyse the running time of these algorithms thanks to a measure, allowing to fruitfully use the existence of the dense local configurations mentioned above.

As for MIN SET COVER, if we know how to solve the problem in $O^*(\gamma_d^n)$ in graphs with average degree d, and that when the average degree is greater than d a good branching occurs, then we seek a complexity of the form $O^*(\gamma^n y^{2m-dn})$. This complexity measure is chosen because, by hypothesis, it is valid in graphs with average degree d. The recurrences given by the branching will produce the best possible value for y. This bottom-up analysis (from smaller to higher average degree) is detailed in Proposition 1.

At the end of the section, we mention some results obtained by a direct application of this method for graphs of average degree 4, 5 and 6.

Lemma 2 *There exists a specific sequence* $(\epsilon_{i,j}, f_{i,j})_{i \geq 4, j \leq i-2}$ *such that, if the input graph has average degree more than* $i - 1 + \epsilon_{i,j}$, *then the following branching is possible*: *either remove one vertex and* i *edges, or* $i+1$ *vertices and* (*at least*) $f_{i,j}$ *edges. For any* i, $\epsilon_{i,0} = 0$. *The following table gives the beginning of the sequence* $(\epsilon_{i,j}, f_{i,j})$:

$(\epsilon_{i,j}, f_{i,j})$	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$
$i=4$	$(0, 10)$	$(3/7, 11)$	$(3/5, 12)$		
$i=5$	$(0, 15)$	$(4/9, 16)$	$(4/7, 17)$	$(4/5, 18)$	
$i=6$	$(0, 20)$	$(5/23, 21)$	$(5/11, 22)$	$(20/37, 23)$	$(5/7, 24)$

Before giving the proof of the lemma, let us give an example, with $i=5$ and $j=2$. This lemma states that if the average degree is greater than $4+\epsilon_{5,2}=4+4/7$, then we can branch on a vertex v and either remove this vertex and 5 edges, or 6 vertices and (at least) 17 edges.

Proof Fix some vertex v_0 of maximum degree d, such that, for any other vertex v of degree d in the graph, $\sum_{w\in N(v)} d(w) \leq \sum_{w\in N(v_0)} d(w)$, and set $\delta = \sum_{w\in N(v_0)} d(w)$.

For $k \leq d$, let n_k be the number of vertices of degree k and m_{kd} be the number of edges (u, v) such that $d(u)=k$ and $d(v)=d$. For $k \leq d-1$, set $\alpha_k = m_{kd}/n_d$ and $\alpha_d = 2m_{dd}/n_d$. In other words, α_k is the average number of vertices of degree k that are adjacent to a vertex of degree d. Since folding or reduction rules remove vertices of degree at most 2, we fix $\alpha_k = 0$ for $k \leq 2$. Summing up inequalities on any vertex of degree d, we get (calculus details are omitted):

$$\sum_{k\leq d} k\alpha_k \leq \delta \tag{1}$$

$$\sum_{k\leq d} \alpha_k = d \tag{2}$$

Fix now $\epsilon = 2m/n - (d-1) \in]0, 1[$. Then, $\epsilon = \frac{\sum_{k\leq d}(k+1-d)n_k}{\sum_{k\leq d} n_k}$. This function is decreasing with n_k, for all $k < d$. Use some straightforward properties: $n_k \geq \frac{m_{kd}}{k}$, $\forall k < d$ and $dn_d = \sum_{k<d} m_{kd} + 2m_{dd}$. This leads to:

$$\epsilon \leq \frac{n_d - \sum_{k<d} \frac{(d-1-k)m_{kd}}{k}}{n_d + \sum_{k<d} \frac{m_{kd}}{k}} = \frac{1 - \sum_{k<d} \frac{(d-1-k)\alpha_k}{k}}{1 + \sum_{k<d} \frac{\alpha_k}{k}} \tag{3}$$

Clearly, when we discard v_0, we remove from the graph one vertex and d edges; when we add it, $d+1$ vertices are deleted. Now, let μ_2 be the minimal number of edges we delete when we add v_0 to the solution. Since there are at least $2d(v_0)$ edges between $N(v_0)$ and the rest of the graph, and thanks to inequalities (1) and (2), we get:

$$\mu_2 \geq 2d + \left\lceil \frac{\delta - 2d}{2} \right\rceil \geq 2d + \left\lceil \frac{\sum_{k\leq d}(k-2)\alpha_k}{2} \right\rceil \tag{4}$$

For $0 \leq j \leq i-2$, we consider the following programs $(P_{i,j})$:

$$\begin{cases} \max(\epsilon) \\ \text{subject to} \\ \quad (1), (2), (3), (4) \text{ and } \mu_2 \leq f_{i,j} - 1 \end{cases}$$

In other words, we look for the maximum value for ϵ such that it is possible that any vertex in the graph verifies $\mu_2 \leq f_{i,j} - 1$. Denote by $\epsilon_{i,j}$ this value. Equivalently, if the graph has degree higher than $i - 1 + \epsilon_{i,j}$, we remove at least $f_{i,j}$ edges (when taking some well chosen vertex). □

Let $d_{i,j} = i - 1 + \epsilon_{i,j}$. Now we use Lemma 2 to recursively define an algorithm STABLE($d_{i,j}$) solving MAX INDEPENDENT SET in a graph of average degree at most $d_{i,j}$. STABLE($d_{i,j}$) performs the usual preprocessing (described above at the beginning of the section) and branches on a vertex that maximizes the number of edges incident to its neighborhood. It repeats this step until the average degree is at most $d_{i,j-1}$, then it applies algorithm STABLE($d_{i,j-1}$). Note that if $j = 0$, we obviously use STABLE($d_{i-1,i-3}$) in graphs of average degree at most $d_{i-1,i-3}$ and the same result as in Proposition 1 holds.

Suppose that STABLE($d_{i,j-1}$) has a running time upper bounded by $\gamma_{i,j-1}^n$. Let $\nu_1 = 1$, $\mu_1 = i$, $\nu_2 = i + 1$ and $\mu_2 = f_{i,j-1}$.

Proposition 1 STABLE($d_{i,j}$) *has running time* $T(m,n) = O^*(\gamma_{i,j-1}^n y_{i,j}^{2m-d_{i,j-1}n})$, $y_{i,j}$ *being the smallest solution of the inequality*:

$$1 \geq \gamma_{i,j-1}^{-\nu_1} y^{-2\mu_1 + d_{i,j-1}\nu_1} + \gamma_{i,j-1}^{-\nu_2} y^{-2\mu_2 + d_{i,j-1}\nu_2}$$

In particular, the running time of STABLE($d_{i,j}$) *is* $O^*(\gamma_{i,j}^n)$ *where* $\gamma_{i,j} = \gamma_{i,j-1} \cdot y_{i,j}^{\epsilon_{i,j} - \epsilon_{i,j-1}}$.

Proof The running time claimed is valid for graphs of average degree $d_{i,j-1}$. If the graph has average degree greater than $d_{i,j-1}$, thanks to Lemma 2, we branch on a vertex where we remove either ν_1 vertices and μ_1 edges or ν_2 vertices and (at least) μ_2 edges. Then, the running time is valid as long as y is such that $T(m,n) \geq T(m - \mu_1, n - \nu_1) + T(m - \mu_2, n - \nu_2) + p(m,n)$ (for some polynomial p). This gives the recurrence relation claimed by proposition's statement.

Since $2m \leq d_{i,j}n$ in a graph of average degree at most $d_{i,j}$, the running time $O^*(y_{i,j}^n)$ follows. Of course, we need to initialize the recurrence, for example with $\gamma_{4,0} = 1.0854$ in graphs of average degree $d_{4,0} = 3$ (i.e., with the basis of the running time of our algorithm for graphs of degree 3 described in Sect. 3).

For completeness, we need to pay attention to the fact that, once the branching has been performed, reduction rules might be applied in order to remove vertices of degree at most 2 in the remaining graph. For instance, if a separated tree on ν vertices is created, reduction rules will remove this tree hence, in all, ν vertices and $\nu - 1$ edges. A vertex folding removes 2 vertices and (at least) 2 edges. All reduction rules remove $\nu \geq 1$ vertices and at least $\nu - 1$ edges. We have to check that these operations do not increase $T(m,n)$, i.e., $T(m,n) \geq T(m - \nu + 1, n - \nu)$. So, we have to check that $y^{d_{i,j-1}-2+2/\nu} \leq \gamma_{i,j-1}$. Of course, it depends on the complexity $\gamma_{i,j-1}$ of the algorithm used for low-degree graphs. In the following, we will always verify that these inequalities are fulfilled when computing y. □

To conclude this section, let us present some of the results we can obtain as corollaries of Proposition 1:

- An algorithm running in time $O^*(1.1707^n)$ for graphs of average degree 4. It is directly obtained using Proposition 1 and our algorithm described in Sect. 3 that runs in time $O^*(1.1781^{m-n})$. Indeed:
  - In graphs of average degree $d_{4,1} = 3 + 3/7$, $m - n = 5/7$; hence this algorithm works in time $O^*(1.1781^{5n/7}) = O^*(1.1243^n)$.
  - Then, thanks to Lemma 2, if the average degree is greater than $d_{4,1} = 3 + 3/7$, we reduce the graph (at least) either by 1 vertex and 4 edges or by 5 vertices and 11 edges. Then, as explained in Proposition 1, the recurrence shows that the running time is $O^*(1.1396^n)$ in graphs of average degree $d_{4,2} = 3 + 3/5$. Indeed, we get $y = 1.0821$ as a solution of the equation: $1 = 1.1243^{-1}y^{-2\times4+(3+3/7)} + 1.1243^{-5}y^{-2\times11+(3+3/7)\times5}$, and the running time is $O^*(1.1243^n y^{((3+3/5)-(3+3/7))n}) = O^*(1.1396^n)$ in graphs of average degree $3 + 3/5$.
  - Similarly, if the average degree is greater than $d_{4,2}$, we reduce the graph either by one vertex and 4 edges or by 5 vertices and 12 edges. Thanks to Proposition 1, we get a running time of $O^*(1.1707^n)$ in graphs of average degree 5.
- An algorithm running in time $O^*(1.2000^n)$ for graphs of average degree 5 based upon the algorithm in $O^*(1.1571^n)$ for graphs of average degree 4 of Theorem 3 (Sect. 4). Thanks to Lemma 2, we know that we can reduce (at least) the graph either by one vertex and five edges, or by 6 vertices and 15, 16, 17 and 18 edges in graphs of average degree greater than $4, 4+4/9, 4+4/7$ and $4+4/5$, respectively. We get successively algorithms working in times $O^*(1.1786^n)$, $O^*(1.1840^n)$, $O^*(1.1929^n)$ and $O^*(1.2000^n)$ in graphs of average degree $4 + 4/9$, $4 + 4/7$, $4 + 4/5$ and 5, respectively.
- An algorithm running in time $O^*(1.2098^n)$ for graphs of average degrees 6 based upon the algorithm in $O^*(1.1895^n)$ for graphs of average degree 5 of Corollary 1 (Sect. 5). Thanks to Lemma 2, we can reduce (at least) the graph either by one vertex and 6 edges, or by 7 vertices and by 20, 21, 22, 23 and 24 edges in graphs of average degree greater than $5, 5 + 5/23, 5 + 5/11, 5 + 20/37$ and $5 + 5/7$, respectively. We get successively algorithms working in times $O^*(1.1947^n)$, $O^*(1.1999^n)$, $O^*(1.2016^n)$, $O^*(1.2049^n)$ and $O^*(1.2098^n)$ in graphs of average degree $5 + 5/23, 5 + 5/11, 5 + 20/37, 5 + 5/7$ and 6, respectively.

It is worth noticing that these results are obtained by a direct application of the method proposed; they will be further improved in the rest of the paper, using more involved case analysis or finer techniques, but already they outperform the best bounds known so far, showing that the bottom-up method presented has a certain interest.

3 An $O^*(1.08537^n)$ MAX INDEPENDENT SET-Algorithm for Graphs of Average Degree at Most 3

3.1 Overview of the Algorithm

Our algorithm has the following form. First it applies a series of well known reduction rules that simplify the instance. Secondly, it looks for vertex separators of size 1 or 2 in the graph and uses them to further simplify the instance. Thirdly, it exploits any

separator that consists of the closed neighborhood of a single degree 3 vertex that separates a tree from the rest of the graph. Finally, the algorithm looks for a suitable structure in the graph and branches on it: it generates a series of subproblems that are solved recursively and from which the largest solution is returned.

Similar to [2, 10], we use the number of edges m minus the number of vertices n as a measure of progress in the analysis of the algorithm. The resulting upper bound on the running time of $O^*(\alpha^{m-n})$ implies a $O^*(\alpha^{0.5n})$ algorithm on graphs in which each connected component has average degree at most 3. The requirement on the average degree of each connected component exist because trees have measure $m - n = -1$; these trees are removed by the reduction rules and can increase $m - n$ to over $0.5n$ for the remaining graph.

3.1.1 Simple Reduction Rules and Small Separators

Our algorithm applies the following well known reduction rules. We have already seen in Sect. 2 that dominated vertices and vertices of degree at least 2 may be removed without branching (either directly or by vertex folding). If these reduction rules do not apply, the graph is of minimum degree 3. The algorithm then follows the approach of [10] and looks for vertex separators of size at most 2.

> *Small Separators:* If the graph contains a vertex separator (i.e., a vertex set the removal of which disconnects the graph) of size 1 or 2, then recursively solve the smallest component and adjust the rest of the graph to the computed solution.

The proof can be found in [10].

Finally, if G is of maximum degree 4, the algorithm looks at local configurations in which the closed neighborhood of a vertex separates a tree from the rest of the graph.

> *Tree Separators:* If in a maximum degree 4 graph the closed neighborhood of a single degree 3 vertex v separates a tree from the rest of the graph, then we replace this local configuration with a smaller equivalent one.

This rule is a slight improvement of the one in [2, Sect. 4]. Details can be found in [3, Sect. 3].

Before considering the branching of our algorithm, we look at the effect of the reduction rules to the $m - n$ measure. The measure is invariant under Rules 1 (for degree 1) and 2 of Sect. 2.2; they remove as much edges as vertices. Rules 1 (for degree 0) increases the measure, but we always treat this (one vertex) tree separately. After application of these rules, the domination and small/tree separator Rules just mentioned decrease $m - n$ by at least 2.

3.1.2 The Branching of the Algorithm

Let us start by giving the main idea as to why our algorithm is faster those published previously. On maximum degree 3 graphs, most previous branching algorithms first branch on vertices that are on a cycle of length 3 or 4 since, as one can easily observe, this leads to a smaller number of generated subproblems. Thereafter, these

algorithms give long subcase analyses to prove that efficiently enough branchings exist for graphs without such cycles.

We use these cycles not only to restrict the worst case to small cycle free graphs, but to improve the branching on these graphs as well. As we reduce low degree vertices, the only maximum degree 3 graphs that our algorithm considers are 3-regular. If no small cycles exist, hence we are forced to perform a relatively inefficient branching on a vertex v, then vertices "near" of v get degree 2 in the subproblems generated. These vertices are folded by Rule 2 (Sect. 2.2), resulting in new vertices of degree at least 4. If new small cycles emerge by this folding, they must contain the new higher degree vertices: this combination allows for very good branching counterbalancing the inefficient branching we started with. If, on the other hand, no new small cycles emerge, then the new higher degree vertices are spread out in the graph, that are enough to allow us to use them for branching in a way that counterbalances the initial inefficient branching even more.

If no reduction rule applies, our algorithm branches producing several subproblems that are solved recursively. In one branch, a vertex is taken in the current solution I and hence it is removed together with its neighborhood. In the other branch, the vertex is decided not to be in I and hence it is removed.

The description of the branching of our algorithm is divided over the proofs of four lemmata. In the analysis, $T(k)$ is the number of subproblems generated when branching on a graph G with $m - n = k$.

We start with non 3-regular graphs, with extra attention to small cycles if the maximum degree is 4.

Lemma 3 *If G has a vertex of degree at least* 5, *then* $T(k) \leq T(k-4) + T(k-7)$.

Lemma 4 *If G is of maximum degree* 4 *but not* 3*-regular, then*:

1. *either G has a vertex of degree* 4 *that is part of a* 3*- or* 4*-cycle also containing at least one degree* 3 *vertex, and there are no* 3*- or* 4*-cycles containing only degree* 3 *vertices; in this case,* $T(k) \leq T(k-5) + T(k-6)$, *or* $T(k) \leq 2T(k-8) + 2T(k-12)$;
2. *or, G has a vertex of degree* 4 *that is part of a triangle containing at least one degree* 3 *vertex, and there is no constraint on the degree* 3 *vertices; in this case,* $T(k) \leq T(k-4) + T(k-6)$, *or* $T(k) \leq 2T(k-8) + 2T(k-12)$;
3. *or, finally, the previous items do not apply and then* $T(k) \leq T(k-3) + T(k-7)$.

If G is 3-regular and contains 3- or 4-cycles, we branch efficiently.

Lemma 5 *If G is* 3*-regular and contains a* 3*- or* 4*-cycle, then* $T(k) \leq T(k-4) + T(k-5)$.

In the remaining case, handled by Lemma 6, we use Lemma 4 to counterbalance the less efficient branching.

Lemma 6 *If G is* 3*-regular and* 3*- and* 4*-cycle free, then* $T(k) \leq T_1(k-2) + T_3(k-5)$, *or a better sequence of branchings exist. Here, we apply situations* 1 *and* 3 *from Lemma* 4 *to the branches denoted by T_1 and T_3, respectively.*

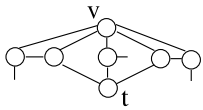


Fig. 1 Vertices of degree at least 5

Taking Lemmata 3, 4, 5 and 6 proves Theorem 2. The worst case of all branchings from Lemmata 3 to 6 is $T(k) \leq T(k-8) + 2T(k-10) + T(k-12) + 2T(k-14)$. This recurrence relation is formed by combining Lemmata 6 and 4 and leads to a running time of $O^*(1.17802^k)$. On average degree 3 graphs this is $O^*(1.17802^{0.5n}) = O^*(1.08537^n)$.

3.2 Proofs of Lemmata 3 to 6

What remains is to prove Lemmata 3 to 6. Lemmata 3 and 5 allow short proofs; they are given first. These proofs also give an idea of the proof of Lemma 4 as its proof uses similar ideas although involving a much more extensive case analysis.

Recall that $m \in V$ is a mirror of $v \in V$ if $G[N(v) \setminus N(m)]$ is a clique (see for example [9]), or equivalently, if every combination of two non-adjacent vertices in $N(v)$ contains a vertex from $N(m)$. Whenever the algorithm branches on v and discards it, it can discard all its mirrors as well. This is because at least two neighbors of v should be in I in this branch because taking v is an alternative of equal size to taking only one.

Using the $m - n$ measure, we have to be careful when trees are separated from G since they have complexity -1. This will not happen in branches where at most two vertices are removed because of the small separators Rule, given in Sect. 3.1.1. The same holds for branches where the neighborhood of a single degree 3 vertex (sometimes through domination) is removed in a maximum degree 4 graph because of the tree separators Rule (Sect. 3.1.1). When trees can be separated, we often bound quantity $m - n$ by counting the number of *external edges*. When considering removal of a set of vertices $S \subset V$ from G, external edges are the edges connecting $G[S]$ to the rest of G.

3.2.1 Proof of Lemma 3

Consider a vertex v fitting conditions of the lemma (Fig. 1) and assume that one branches on v. If v is discarded, then one vertex and at least 5 edges are removed, giving $T(k-4)$. If v is taken in I, then $N[v]$ is removed. All vertices in $N(v)$ have at least one neighbor outside of $N[v]$ by domination. In the worst case, $N(v)$ consists of degree 3 vertices, hence there are at most two edges in $G[N(v)]$ and at least six external edges. If no trees are created, this amounts to 13 edges and 6 vertices giving $T(k-7)$. Furthermore, if there are more external edges because either a vertex in $N(v)$ has degree 4 or more, or there are fewer edges in $G[N(v)]$, then the number of external edges increases giving $T(k-7)$, or better, depending on the fact that trees are separated from G or not.

It remains to handle the special case where the minimum amount of 13 edges is removed and a separate tree is created. This tree will be a single degree 3 vertex t since, otherwise, there exists a 2-separator in $N(v)$. Notice that v is a mirror

of t. We branch on t. Taking t in I leads to the removal of 4 vertices and 9 edges: $T(k-5)$. Furthermore, discarding t and v leads to the removal of 8 edges and 2 vertices: $T(k-6)$. This branching with $T(k) \leq T(k-5) + T(k-6)$ is better than the required $T(k) \leq T(k-4) + T(k-7)$.

3.2.2 *Proof of Lemma 5*

Let a, b and c form a 3-cycle in G. Assume that one of these vertices, say a, has a third neighbor v that is not part of a 3-cycle. The algorithm branches on v. In one branch, v is taken in I and 9 edges and 4 vertices are removed: $T(k-5)$. In the other branch, v is discarded and, by domination, a is taken in I resulting in the removal of 8 edges and 4 vertices: $T(k-4)$.

This covers the 3-cycles, unless all third neighbors of a, b and c are in a 3-cycle also. Moreover, they should be in different 3-cycles because, otherwise, there would exist a size 2 vertex separator. We branch on a. In the branch where a is discarded, domination results in the fact that its third neighbor is taken in I giving $T(k-4)$ as before. In the other branch, a is taken in I and, by domination, the third neighbors of b and c are taken in I also. This removes their corresponding 3-cycles completely: at least 16 edges and 10 vertices are removed. We notice that a tree can be separated from G but, in this case, we still have $T(k-5)$ or better.

Finally, suppose that G is 3-cycle free and let v be a vertex on a 4-cycle. Any vertex opposite to c on a 4-cycle is a mirror of v. We branch on v. In one branch, we take v in I and 3-cycle freeness results in the removal of 9 edges and 4 vertices: $T(k-5)$. In the other one, we discard v and all its mirrors. This results in the removal of 6 edges and 2 vertices, if v has only one mirror, and possibly more, if v has two or three mirrors: $T(k-4)$. Again trees can be separated from G, but then v must have three mirrors. There is only one local configuration representing this case in which 12 edges and 7 vertices are removed, which is more than enough.

3.2.3 *Proof of Lemma 6*

Before giving the proof, let us note that, if there are no 3- or 4-cycles in G, that the algorithm can exploit, it is mandatory to perform a $T(k) \leq T(k-2) + T(k-5)$ branching on a degree 3 vertex v. This would give a running time much worse than the one we try to prove. However, Lemma 6 shows that either such a branching results in the creation of local structures that, thereafter, allow the much more efficient branching described in Lemma 4, or there are more straightforward ways to perform even better branchings.

The proof of Lemma 6 strongly relies on the fact that we assume G to be 3- and 4-cycle free, and therefore that after branching any new 3- or 4-cycle must involve folded vertices. It is based upon an extensive analysis of possible local structures that are the result of vertex folding after taking or discarding v.

Let us now proceed to the proof of the lemma. In a 3-regular graph without 3- or 4-cycles, the algorithm is forced to perform a less efficient branching. If we consider branching on any vertex v, the branch where v is taken in I results in the removal of 9 edges and 4 vertices, while the other branch results in the removal of 3 edges and 1 vertex: $T(k) \leq T(k-2) + T(k-5)$.

Notice that our reduction rules guarantee that no trees can be separated from G since we branch on a degree 3 vertex in a maximum degree 3 graph. We can also assume that no other reduction rules than Rules 1 (for degree 1) and 2 of Sect. 2.2 can be applied after branching. In fact, if such a reduction rule fires in the branch where we discard v, then we have the sufficient relation: $T(k) \leq T(k-4)+T(k-5)$. If such a rule fires only in the branch where we take v in I, then we follow the proof for the other branch below and obtain the sufficient relation: $T(k) \leq T_3(k-2)+T(k-7)$.

Let G be as in the statement of the lemma and let v be any vertex of G with neighbors x, y and z. We systematically consider the possible local neighborhoods around v and observe what happens to these local neighborhoods when branching on v. Because of 3- and 4-cycle freeness, v is the only common neighbor of any two vertices from the set $\{x, y, z\}$. With the same argument, there cannot exist adjacent vertices within $N(x)$, $N(y)$ or $N(z)$. There can, however, exist at most six adjacencies between vertices from two different neighborhoods. These adjacencies are important in the branch where v is discarded. Here, the remaining vertices in the closed neighborhoods of x, y and z are folded, resulting in three new vertices. The adjacencies between the old neighborhoods $N(x)$, $N(y)$ and $N(z)$ determine the new local structure on which we will branch next. Notice that, whenever there are two adjacencies between the same neighborhoods, then vertex folding after discarding v leads to the removal of an additional edge because we do not allow double edges.

We begin by showing that we can easily deal with cases involving more than three of these adjacencies between the neighborhoods of x, y and z. If there are six, then we are looking at a connected component of constant size that can be solved in constant time. If there are five, then there are only two external edges out of $N[x] \cup N[y] \cup N[z]$ and hence there exists a small separator. Finally, if there are four such adjacencies, then they cannot all be between the same two neighborhoods as this creates a 4-cycle. The alternative is that all three neighborhoods are adjacent. In the branch where we discard v, the neighborhoods of x, y and z are folded resulting in two degree 3 vertices between which a double edge is removed that are in a 3-cycle with a degree 4 vertex. This allows us to apply Lemma 4 case 2 to obtain the following recurrence relations with a better branching behavior than we are proving: $T(k) \leq T_3(k-5) + T(k-3-4) + T(k-3-6)$, or $T(k) \leq T_3(k-5) + 2T(k-3-8) + 2T(k-3-12)$.

We will continue by showing that we can always obtain a $T_3(k-5)$ branch when taking v in I and discarding its neighbors. Removing $N(v)$, results in the creation of six degree 2 vertices that will be folded. If any of these vertices are folded to degree 4 vertices, we can apply Lemma 4 and we are done.

Consider the case in which no degree 4 vertex is created. In this case, the degree 2 vertices must form a set of chains of even length. By the previous argument, we know that there are at most three adjacencies between the vertices in $(N(x) \cup N(y) \cup N(z)) \setminus \{v\}$. These vertices are the new degree 2 vertices, and therefore the only possible way for them to be divided in even length chains is when they form three pairs of adjacent vertices. In this particular local structure, v lies on three 5-cycles, each pair of which overlaps in v and in a different neighbor of v. We obtain the required branching behavior by deciding not to branch on v: either this connected graph G has a vertex with a different local configuration, or G has no such vertex

and it is isomorphic to a dodecahedron. We conclude the argument by noting that the dodecahedron has 20 vertices and can be solved in constant time.

What remains is the $T_1(k-2)$ branch when discarding v. In this branch, vertex folding results in three folded vertices. Because the graph is 3- and 4-cycle free before applying the lemma, a new 3- or 4-cycle created by folding must involve the folded vertices. If such a 3- or 4-cycle is created, we can apply Lemma 4. Notice that the folded vertices are of degree 4, unless folding results in the implicit removal of double edges between folded vertices. If all folded vertices are of degree 4, we can apply Lemma 4 case 1 obtaining our result of $T_1(k-2)$. If, on the other hand, additional edges are removed and we also consider the possibility that 3- or 4-cycles involving only degree 3 vertices are created, we can apply the slightly worse case 2 of Lemma 4 on the graph of measure $k-3$. This, results in the even better behavior of $T(k) \leq T_3(k-5) + T(k-3-4) + T(k-3-6)$, or $T(k) \leq T_3(k-5) + 2T(k-3-8) + 2T(k-3-12)$.

The cases in which no new 3- or 4-cycles are created by folding are treated next. These cases are handled by a closer inspection of each of the remaining possible local neighborhoods created by folding. Each time we let x', y' and z' be the result of folding the neighborhoods of x, y and z to single vertices, respectively. Notice that, by previous cases, there are at most three adjacencies between x', y' and z', no 3- or 4-cycles exist involving (folded) degree 4 vertices, and that for each case we already have a $T_3(k-5)$ branch when taking v. We will construct sequences of branchings that are efficient enough to obtain our running time.

When we consider to remove a set of vertices S from the graph we often speak of the related *external edges*. These external edges are the edges incident to the boundary of S, but not to the interior of S. We mostly need the number of these edges, in which case we count the number of endpoints. This, however, leads to counting edges in the boundary of S twice. We will call these twice counted edges *additional adjacencies*, and we will always consider how many additional adjacencies can exist before claiming how many edges are removed.

Three Non-adjacent Degree 4 Vertices

We will perform a $T(k) \leq T_3(k-3) + T(k-9)$ branching after discarding v using some additional information on the maximum independent set we need to compute in this branch. This reasoning is quite similar to exploiting mirrors. Namely, if v is discarded, we know that we need to pick at least two of the three neighbors of v: if we pick only one we could equally well have taken v which is done in the other branch already. This observation becomes slightly more complicated because we just folded the neighbors of v. The original vertex x is taken in the independent set if and only if the vertex x' is discarded in the reduced graph. Thus, the fact that we needed to pick at least two vertices from $N(v)$ results in being allowed to pick at most one vertex from the three degree 4 vertices created by folding the neighbors of v. Hence, picking any vertex from the three folded vertices allows us to discard the other two (Fig. 2).

If we discard x', we remove 4 edges and 1 vertex. Moreover, at least one degree 4 vertex remains in the graph after discarding x' giving $T_3(k-3)$, or at least one additional edge is removed by folding, resulting in $T(k-4)$ (which, in this case, is even

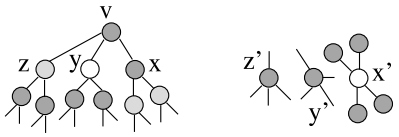


Fig. 2 Picking at most one vertex from the three degree 4 vertices created by folding the neighbors of v

better). If, in the other branch, we take x' in I, then we can discard all four degree 3 neighbors and both y' and z', resulting in the removal of 20 edges from which 16 external edges and 7 vertices. Because y' and z' are non adjacent and they can only be adjacent to a single neighbor of x' (or a 4-cycle would exist), there are at most two additional adjacencies and thus at least 12 remaining external edges. If at most two trees are separated from the graph, this gives the $T(k-20+7+2+2) = T(k-9)$ branch.

We will show that at most two trees can be separated from the graph. Whenever a tree is separated, we have some very specific local structures. Because of the 3- and 4-cycle freeness, every tree vertex t can only have neighbors that are at distance at least 3 away from each other in $G[V \setminus \{t\}]$. The only size 1 trees that can be separated are adjacent to both y' and z' and a neighbor of x' that is not adjacent to either y' or z'. There can be at most one such tree, since two of these trees adjacent to the same two vertices also create a 4-cycle. Furthermore, this tree can only exist if y' and z' are adjacent to different neighbors of x'. This, results in 9 remaining external edges that because of the small separators Rule (Sect. 3.1.1) can form only one larger tree. If there is no size 1 tree, larger trees use more external edges and hence there can also exist at most two of them.

In the above proof we have assumed that there are two additional adjacencies. There can, however, also exist fewer additional adjacencies leaving more external edges to form trees. In this case, the fewer additional adjacencies lead to additional edges compensating for the possible additional tree. Putting all the above together, we get $T(k) \leq T_3(k-2-3) + T(k-2-9) + T_3(k-5) \leq 2T(k-8) + T(k-11) + 2T(k-12)$.

Three Degree 4 Vertices only Two of which Are Adjacent

Identical to the previous case, we aim at performing a $T(k) \leq T_3(k-3) + T(k-9)$ branching (or better) after discarding v. In the worst case, this gives $T(k) \leq 2T(k-8) + T(k-11) + 2T(k-12)$. Without loss of generality, assume that x' is adjacent to y' while z' is not adjacent to neither of them.

If we discard x', we remove 4 edges and 1 vertex. Now, either a degree 4 vertex remains, giving $T_3(k-3)$, or an additional edge is removed by folding, giving the in this case the slightly better $T(k-4)$. If we take x' in I, we can also discard z' resulting in the removal of 17 edges 13 out of which are external, and 6 vertices. There can be at most one additional adjacency, namely between z' and a degree 3 neighbor of x'. Any tree vertex must again be adjacent to vertices that are at distance at least 3 away from each other in this structure. This can only be z' and any neighbor of x'. Hence, there cannot be any size 1 tree: it would need two neighbors of x' which entails a 4-cycle. Actually, there can be no tree at all since every tree leaf needs to

be adjacent to z' in order to avoid 4-cycles in $N(x')$, but this also implies a 4-cycle. Hence, we have $T(k-17+6+1) = T(k-10)$.

If there is no additional adjacency, there can again exist no 1-tree since it can be adjacent to at most one neighbor of x'. Larger trees remove enough external edges and lead to $T(k-9)$.

Three Degree 4 Vertices on a Path

Again, we combine application of Lemma 4 to the branch where we take v in I, with a $T(k) \leq T_3(k-3) + T(k-9)$ (or better) branching after discarding v. This again leads to $T(k) \leq 2T(k-8) + T(k-11) + 2T(k-12)$. Let y' be adjacent to both x' and z', and let x' and z' be non-adjacent.

If we discard x', we remove 4 edges and 1 vertex while z' remains of degree 4 giving the $T_4(k-3)$. If we take x' in I, we can also discard z' resulting in the removal of 16 edges, 11 out of which are external, and 6 vertices. Notice that in the last branch there cannot exist additional adjacencies, since they imply 3- or 4-cycles. There also cannot exist trees consisting of 1 or 2 vertices because tree leaves can only be adjacent to z' and a degree 3 neighbor of x'. Finally, any larger tree decreases the number of external edges enough to obtain $T(k-16+6+1) = T(k-9)$.

Two Degree 3 Vertices and a Non-adjacent Degree 4 Vertex

We now have a graph of measure $k-3$ with two degree 3 vertices, say y' and z', and a degree 4 vertex x'. Furthermore, y' and z' are adjacent, but they are not adjacent to x'. Among these vertices, x' cannot be involved in any 3- or 4-cycle, or we apply Lemma 4 case 2 as discussed previously.

We branch on x'. This leads to $T(k-3-3)$ when discarding x'. Similar to the above cases, we can still discard both y' and z' when taking x' in I. Therefore, taking x' leads to removing 17 edges, 12 out of which are external, and 7 vertices. If there is an additional adjacency, this is between y' or z' and a neighbor of x'. In this case, there can exist at most one tree since y' and z' together have only 3 external edges left and every tree leaf can be adjacent to at most one neighbor of x' (otherwise a 4-cycle with x' would exist). This leads to $T(k-3-17+7+1+1) = T(k-11)$. If there is no additional adjacency, every tree leaf can still be adjacent to no more than one neighbor of x', which together with the 4 external edges of y' and z' lead to at most 2 trees and $T(k-11)$. We so obtain $T(k) \leq T_3(k-5) + T(k-6) + T(k-11)$.

Two Degree 3 Vertices Adjacent to a Degree 4 Vertex

We again have a graph of measure $k-3$ with two degree 3 vertices y' and z' and a degree 4 vertex x' all of them resulting from folding. Furthermore, y' is adjacent to x' and z', while x' and z' are non-adjacent. Among these vertices, x' cannot be involved in any 3- or 4-cycle, since we then apply Lemma 4 case 2 as discussed previously.

Similarly to the previous case, we branch on x' getting $T(k-3-3)$ when discarding x', and when taking x' in I we discard y' and z' as well. In this second branch, 14 edges and 6 vertices are removed giving $T(k-3-8)$ if there are no trees separated. Thus, in this case, we obtain $T(k) \leq T_3(k-5) + T(k-6) + T(k-11)$ as before.

If trees are separated, observe that every tree leaf can again be adjacent to at most one neighbor of x', and hence all tree leaves must be adjacent to z'. Also observe that the third neighbor of y' cannot be adjacent to x' or any of its neighbors. Since z' has only two external edges, this means that the only tree that can exist is a size 2 tree with both leaves connected to z' and a different neighbor of x' not equal to y' (or else, z dominates a tree vertex). Notice that this implies a 3-cycle involving the tree and z'. In this case, we branch on y'. When taking y', we remove 10 edges and 4 vertices: $T(k-3-6)$. When discarding y', the tree forms a 3-cycle in which, by dominance, z' is taken in I. Since we can take at most one of the folded vertices, this also results in that x' is discarded. In all, we conclude in the removal of 11 edges and 4 vertices, and in this very specific structure no trees can exist: $T(k-3-6)$. We so obtain the better $T(k) \leq T_3(k-5) + 2T(k-9)$.

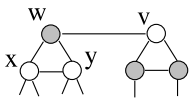
Three Degree 4 Vertices that Form a Clique

We treat this last subcase in an entirely different way. Let $N(x) = \{v, a, b\}$, $N(y) = \{v, c, d\}$ and $N(z) = \{v, e, f\}$. From the general proof of the $T_3(k-5)$ branch when taking v in I, we know that if there are three adjacencies between the three neighborhoods $N(x)$, $N(y)$ and $N(z)$ then these are not pairwise distributed over the six possible vertices. This was proved by deciding to branch on another vertex which is always possible unless G is isomorphic to a dodecahedron. Hence, we know that at least one vertex from $\{a, \ldots, f\}$ has a neighbor in both other neighborhoods. Also, since there cannot be more than three adjacencies between these neighborhoods, we know that there can be at most two such vertices with neighbors in both other neighborhoods, and if there are two, then they must be adjacent.

We begin with the case where one vertex has neighbors in both other neighborhoods. Without loss of generality, let this vertex be a and let $N(a) = \{x, c, e\}$ and let d and f be adjacent. We now branch on x instead of v. If we discard x, the neighborhoods of a, b and v are folded to single vertices and we implicitly remove a double edge coming from the old edges (c, y) and (e, z). Furthermore, $N(b)$ is folded to a degree 4 vertex giving a $T_3(k-3)$ branch. If we take x in I and discard $N(x)$, then $c - y$ and $e - z$ are chains of two degree 2 vertices that are replaced by a single edge. Since these chains end in d and f and the neighbors of b are non adjacent, a degree 4 vertex must be created giving $T_3(k-5)$ in the other branch. Putting all this together results in the sufficiently efficient recurrence of $T(k) \leq T_3(k-3) + T_3(k-5) \leq T(k-6) + T(k-8) + T(k-10) + T(k-12)$.

We conclude the proof of the lemma with the last case involved. Without loss of generality let both a and c have neighbors in both other neighborhoods. Set $N(a) = \{x, c, e\}$ and $N(c) = \{y, a, f\}$. We obtain the same branching of $T(k) \leq T_3(k-3) + T_3(k-5)$ when branching on x. If we discard x, the edges $\{c, f\}$ and $\{e, z\}$ lead to a double edge between the folded neighborhoods $N(a)$ and $N(v)$. Also, $N(b)$ will become a degree 4 vertex giving the $T_3(k-3)$ branch. In the other branch we take x in I again leading to two chains of degree 2 vertices that are replaced by single edges. In this case, these edges are incident to c and f. By the same argument as before, the neighbors of b will be folded to at least one degree 4 vertex getting $T_3(k-5)$.

Fig. 3 Vertex v has degree 3



3.2.4 Proof of Lemma 4

The proof of Lemma 6 in the previous section, and thus also the running time of our algorithm, greatly relies on the fact that we can perform very efficient branching on graphs with degree 4 vertices. Moreover, they require even more efficient branching when every 3- and 4-cycle contains a degree 4 vertex. In this section, we describe the branching that satisfies these requirements.

The proof of Lemma 4 uses the fact that, in a 3-cycles containing a degree 3 vertex, we can often branch in such a way that we can apply the domination Rule 3 (Sect. 2.2). And, in a 4-cycles containing a degree 3 vertex, we can use the fact that the vertex opposite the degree 3 vertex is a mirror of this vertex. Actually, the only 4-cycle in which this does not happen is a 4-cycle consisting only of degree 4 vertices. These observations are similar to those used in the proof of Lemmata 5 and 3. In contrast to these two lemmata, the next proof uses a much larger case analysis.

We start the proof of Lemma 4 by noticing that our reduction rules guarantee that no trees can be separated from G when we branch on a degree 3 vertex. Furthermore, no trees are separated from G when discarding a vertex that, by domination, leads to a single degree 3 vertex to be taken in I.

We first consider all possible 3-cycles containing both degree 3 and 4 vertices. Then we consider all possible 4-cycles containing both degree 3 and 4 vertices in a 3-cycle free graph. In each such subcase, cases 1 and 2 from the statement of the lemma are proved. Thereafter, the remaining case 3 will be proved.

3-Cycles with Two Degree 4 Vertices and a Degree 3 Vertex

Let x, y, w be a 3-cycle in the graph with $d(x) = d(y) = 4$ and $d(w) = 3$; also, let v be the third neighbor of w. Notice that discarding v entails domination which results in that w is taken in the maximum independent set.

If v is of degree 4, discarding it and taking w leads to the removal 11 edges and 4 vertices: $T(k-7)$. Taking v and removing $N[v]$ results in the removal of 3 edges incident to w and of at least 8 more edges and 5 vertices. If, in this last case, all neighbors of v are of degree 3, then there exist at most 6 external edges not incident to w. In this case, there can be at most one tree since the neighbors of w are fixed and cannot form a tree because then there would exist a small separator. We obtain $T(k-6+1)$. If not all neighbors of v are of degree 3, then the degree 4 neighbors of v entail removal of even more edges, compensating for any possible additional tree.

If v is of degree 3 (Fig. 3), discarding it and taking w instead, leads to the removal of at least 10 edges and 4 vertices: $T(k-6)$. In this case, if v is not part of another 3-cycle or v has a degree 4 neighbor (case 1 of the lemma), taking v removes at least 9 edges and 4 vertices: $T(k-5)$. On the other hand, if v is part of a 3-cycle of degree 3 vertices (case 2 of the lemma), taking v removes 9 edges and 4 vertices: $T(k-4)$.

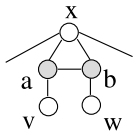


Fig. 4 Triangles with one degree 4 vertex and two degree 3 vertices

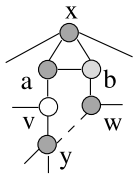


Fig. 5 Vertex v, has a degree 4 neighbor y

3-Cycles with One Degree 4 Vertex and Two Degree 3 Vertices

When there exists only one degree 4 vertex, the situation gets a lot more complicated. Let x, a and b be the 3-cycle vertices with $d(x) = 4$ and $d(a) = d(b) = 3$, also let v be the third neighbor of a, and let w be the third neighbor of b (Fig. 4). By domination, v and w are not adjacent to x, and $v \neq w$. If v and w are adjacent, we can safely discard x reducing the graph. This follows from the fact that if we pick v, we would also pick b, and if we discard v, its mirror b is also discarded which results in that a is picked. In both cases, a neighbor of x is in a maximum independent set, and hence x can safely be discarded. So, we assume that v and w are non-adjacent.

If v or w, say v, is of degree 4, taking v removes at least 11 edges and 5 vertices, but since there are 6 external edges there can be a tree: $T(k-5)$. Furthermore, if there are more external edges (less edges in $N(v)$), the number of removed edges increases. Discarding v and, by domination, taking a, leads to the removal of 10 edges and 4 vertices: $T(k-6)$. So, from now on, we can assume that v and w are of degree 3.

Consider the case where v or w, say v, has a degree 4 neighbor y (Fig. 5). Suppose that y does not form a 3-cycle with v, then taking v removes at least 10 edges and 4 vertices: $T(k-6)$. Discarding v and, by domination, taking a removes at least 9 edges and 4 vertices: $T(k-5)$. If, on the other hand, y forms a 3-cycle with v, then we branch on w. If, in this case, w has a degree 4 neighbor, or is not involved in a 3-cycle (case 1 of the lemma), then taking w, results as before in $T(k-5)$. Discarding w by domination, results in taking b; this, again by domination, results in taking v. In total 15 edges are removed, 7 out of which are external, and 7 vertices. Because of the small separators, there can be at most 2 additional adjacencies, leaving, at worst, 3 external edges and $T(k-6)$. Note that trees are beneficial over additional adjacencies. This leaves the case where w has only degree 3 neighbors with which it forms a 3-cycle (case 2 of the lemma). In this case, taking w leads to $T(k-4)$ which now is enough. So, we can assume that v and w are of degree 3 and have no degree 4 neighbors.

Suppose that v or w, say v, is part of a 3-cycle (Fig. 6). Notice that we are now in case 2 of the lemma. We branch on w. In the first branch, we take w and the worst case arises when w is also part of a 3-cycle; 8 edges and 4 vertices are removed: $T(k-4)$. In the other branch, we discard w and, by domination, b and v are put in I removing a total of at least 14 edges, 6 out of which are external, and 7 vertices.

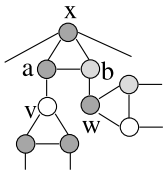


Fig. 6 Vertex v is part of a triangle

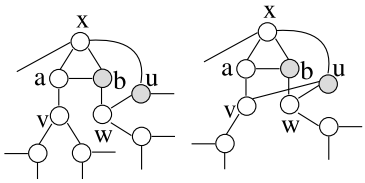


Fig. 7 Vertex w, has a neighbor $u \neq a, b$ that is adjacent to x

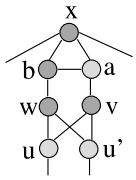


Fig. 8 Vertices v and w are adjacent to both of $u, u' \in X$

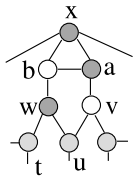
Because of the small separator Rule (Sect. 3.1.1), the external edges can form at most one additional adjacency or tree, leading to $T(k-6)$. So, at this point, we can also assume that v and w are not part of any 3-cycle.

Suppose that v or w, say w, has a neighbor $u \neq a, b$ that is adjacent to x (Fig. 7). We branch on v. In the branch where we discard v, a is picked by domination and we still have $T(k-5)$. In the branch where we take v, b will become a degree 2 vertex with neighbors x and w that will be folded. Notice that both x and w are adjacent to u, and hence this folding removes an additional edge: $T(k-6)$. The only case in which the above does not hold is when v and w are both a neighbor of u. We reduce this exceptional case by noting that the tree separators Rule (Sect. 3.1.1) fires when considering branching on u (without actually branching on u of course) because this would create the size 2 tree $\{a, b\}$. Hence, we can also assume that v and w have no neighbors besides a and b that are adjacent to x.

The rest of the analysis of this case consists of three more subcases depending on the number of vertices in $X = (N(w) \cup N(v)) \setminus \{a, b\}$. Because of the degrees of w and v: $2 \leq |X| \leq 4$.

If $|X| = 2$, v and w are adjacent to both vertices in X (Fig. 8). Notice that if we take v in I, it is optimal to also pick w and vice-versa. We use this and branch by taking both v and w in I, or discarding both of them. If we take both v and w in I, 11 edges are removed and 6 vertices: $T(k-5)$. If we discard both v and w, then we can take a in the independent set and remove 11 edges and 5 vertices: $T(k-6)$. With this special kind of branching, we have to check that we do not separate trees from G. This cannot be the case when taking both v and w in I since there are only 4 external edges. Furthermore, this can also not be the case when discarding both v and w. Then, only two tree leaves are formed which are either adjacent, which results in a small separator, or adjacent to the only degree 2 vertices (neighbors of x), which

Fig. 9 The case $|X| = 3$



also results in a small separator or even in a constant size component. Furthermore, there cannot exist additional adjacencies because, in this case, there also exist a small separator.

If $|X| = 3$, let $u \in X$ be the common neighbor of v and w and let $t \in X$ be the third neighbor of w (Fig. 9). We branch on t. If we take t in I, we also take b by domination. This results in the removal of 7 vertices and 15 edges, if t has a degree 4 neighbor, or if there is no 3-cycle involving t. Otherwise, only 14 edges are removed. We have the required $T(k-6)$ or $T(k-5)$ since there can be at most 8 external edges with this number of removed edges, and hence at most 2 additional adjacencies or trees. If we discard t, 3 edges and 1 vertex are removed and the folding of w results in the merging of vertices b and u. The new vertex can be discarded directly since it is dominated by a resulting in an additional removal of 4 edges and 1 vertex. This leads to $T(k-5)$ in total. Furthermore, we cannot separate trees in this way since there can be at most one vertex of degree less than 2 (adjacent to t and u, but no to w) which cannot become an isolated vertex. Depending on whether t is in a 3-cycle, we are in case 1 or 2 or the lemma and we have a good enough branching.

Finally, if $|X| = 4$, all neighbors of v and w are disjoint. We branch on v. If we take v in I, we remove 9 edges and 4 vertices, and if we discard v, we take a and again remove 9 edges and 4 vertices. This $T(k) \leq T(k-5) + T(k-5)$ branching is not good enough, and therefore we look to the result of both branches.

If we take a in I, then w is folded, resulting in the removal of an additional edge if its neighbors have another common neighbor. In this case, we are done. But if this is not the case, the folding of w results in a degree 4 vertex. In the other branch where we take v, b is folded resulting in another degree 4 vertex. We now apply the worst case of this lemma (case 3) inductively to our two $T(k-5)$ branches and obtain $T(k) \leq 2T(k-8) + 2T(k-12)$ as in the lemma.

Note that $T(k) \leq T(k-5) + T(k-5)$ has a smaller solution than $T(k) \leq 2T(k-8) + 2T(k-12)$. However, after the bad branch in a 3-regular graph of Lemma 6, the second recurrence gives a better solution when applied in the $T_1(k-2)$ branch. This is because it is a composition of three branchings that are all much better than the bad 3-regular graph branching: the composition has more "weight".

4-cycles in which a Degree 4 Vertex is a Mirror of a Degree 3 Vertex

Let x be the degree 4 vertex that is a mirror of the degree 3 vertex v, let a and b be their common neighbors, and let w be the third neighbor of v (Fig. 10). If we branch on v and take v in I, we remove at least 9 edges and 4 vertices. When we discard v and also x because it is a mirror of v, we remove 7 edges and 2 vertices. This gives the insufficient $T(k) \leq T(k-5) + T(k-5)$. Notice that if we discard v and

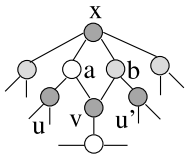


Fig. 10 Vertex x is a degree 4 vertex that is a mirror of the degree 3 vertex v

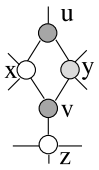


Fig. 11 Vertex v, has a third degree 4 neighbor z

x, there can be no trees, since then v and x would form a small separator. Also, any additional adjacency (a and b adjacent) results in 3-cycles involving degree 3 and 4 vertices which are handled in the previous cases of this proof.

First assume that a, b or w is of degree 4; then, we have $T(k-6)$ or better when taking v in I. Hence, we can assume that a, b and w are of degree 3. We can also assume that a and b have no common neighbor outside this 4-cycle: if they had such a neighbor, then the tree separators Rule (Sect. 3.1.1) should fire on a with possible size 1 tree b.

Let u and u' be the third neighbors of a and b, respectively. When discarding v and x, both a and b are taken in I and u and u' are discarded. This means that 13 edges, 7 out of which are external, and 6 vertices are removed. If u and u' are vertices of degree 3, then the only possible additional adjacencies are those between u and u', or between u or u' and v. But there can be only one additional adjacency because, otherwise, there exists a small separator. So, we end up removing 12 edges, 5 out of which are external, and 6 vertices which cannot create trees: $T(k-6)$. Finally, if u or u' are of degree 4 then the additional edges removed compensate for any possible additional adjacency or separated tree.

4-cycles that Contain Degree 3 and 4 Vertices, While no Degree 4 Vertex is a Mirror of a Degree 3 Vertex

This can only be the case if the cycle consists of two degree 4 vertices x, y and two degree 3 vertices u, v with x and y not adjacent. There are no other adjacencies than the cycle between these vertices because then we would apply branchings from the analysis of 3-cycles.

Suppose that either u or v, say v, has a third degree 4 neighbor z (Fig. 11). Notice that this neighbor cannot be adjacent to either x or y. If we branch on v and take v in I, we remove 12 edges and 4 vertices. When we discard v and its mirror u, we remove 6 edges and 2 vertices. Thus, we have $T(k) \leq T(k-8) + T(k-4)$, giving a better branching behavior than required by this lemma. So, we can assume that u and v have no degree 4 neighbors not on the 4-cycle.

Let w be the degree 3 neighbor of v. It is not adjacent to u since such adjacency would imply that x and y are mirrors of w. Since w cannot be adjacent to x or y,

taking v in I results in the removal of 11 edges and 4 vertices: $T(k-7)$. Discarding v and u leads to the removal of 6 edges and 2 vertices and hence to a graph of measure $k-4$. In this case, x and y will be folded. Because they are not adjacent to other created degree 2 vertices (in the opposite, there would exist 3-cycles involving degree 3 and 4 vertices), a vertex of degree at least 4 is created, or at least one additional edge is removed. It is also possible that a reduction rule different from Rules 1 and 2 (Sect. 2.2) fires on the new graph. In this last case we have $T(k-4-2)$, giving $T(k) \leq T(k-6) + T(k-7)$.

If an additional edge is removed, then this leads to $T(k) \leq T(k-5) + T(k-7)$. Otherwise, if both options do not apply, we apply the worst case of this lemma inductively to our new degree 4 vertex and obtain $T(k) \leq 2T(k-7) + T(k-11)$. These recurrences are sufficient to prove our running time.

A Degree 4 Vertex that is not Involved in any 3- or 4-cycle with any Degree 3 Vertex

We finally arrive at case 3 of our lemma. Let x be this degree 4 vertex. If all its neighbors are of degree 3, branching on it results in $T(k) \leq T(k-7) + T(k-3)$. In this case, no trees can be separated since any tree leaf is of degree at least 3 before branching and, therefore, it must have at least 2 neighbors in $N(x)$ to become a leaf. But in this last case, there exist 4-cycles with degree 3 and 4 vertices on it which contradicts our assumption.

If x has degree 4 neighbors, still no trees will be separated unless at least three neighbors of x are of degree 4 and every tree leaf was of degree 4 before branching. In this case, the additional degree 4 vertices entail the removal of more edges. If x has three neighbors of degree 4, then at least 13 edges are removed, 7 out of which are external. This can lead to at most one tree and thus gives $T(k-7)$, as required. If there are more external edges, more edges will be removed compensating for any possible separated tree. Finally, if x has four degree 4 neighbors, we remove at least 12 edges, 4 out of which are external, always getting $T(k-7)$. Again, the existence of any separated tree implies more external edges that are removed and compensate for the tree.

4 Refined Case Analysis for Graphs of Average Degree 4

In Lemma 2 (Sect. 2.2), we have shown the existence of local dense configurations when the graph has average degree more than 3. For instance, if it has average degree at least $4+4/7$, then we can branch on a vertex v and either remove this vertex and 5 edges, or 6 vertices and (at least) 15 edges. In this section, we apply a similar method, by performing a deeper analysis, to compute the running time of an algorithm for graphs of average degree 4. Indeed, we prove the following theorem.

Theorem 3 *It is possible solve* MAX INDEPENDENT SET *on graphs with maximum* (*or even average*) *degree* 4 *with running time* $O^*(1.1571^n)$.

Proof Based upon Sect. 2, we seek a complexity of the form $O^*(\gamma^n y^{2m-3n})$, where $\gamma = 1.08537$, valid for graphs of average degree 3. We assume that our graph has $m > 3n/2$ edges. In particular, there is a vertex of degree at least 4.

Fig. 12 $G[N(v)]$ is a 4-cycle

Assume that we perform a branching that reduces the graph by either ν_1 vertices and μ_1 edges, or by ν_2 vertices and μ_2 edges. Then, by recurrence, our complexity formula is valid for y being the largest root of the following equality: $1 = \gamma^{-\nu_1} y^{-2\mu_1+3\nu_1} + \gamma^{-\nu_2} y^{-2\mu_2+3\nu_2}$. Then, either there exists a vertex of degree at least 5, or the maximum degree is 4. In the former case, we reduce the graph either by $\nu_1 = 1$ vertex and $\mu_1 = 5$ edges, or by $\nu_2 = 6$ vertices and $\mu_2 \geq 13$ edges, leading to $y = 1.0596$.

In what follows, we consider that the graph has maximum degree 4, and we denote by u_1, u_2, u_3 and u_4 the four neighbors of some vertex v. We call inner edge an edge between two vertices in $N(v)$, and outer edge an edge (u_i, x) where $x \notin \{v\} \cup N(v)$. We study four cases, depending on the configuration of $N(v)$. We consider that no trees are created while branching since, as we show later, trees' occurrence is never problematic.

Case 1: All the Neighbors of v Have Degree 4

This case is easy. Indeed, if there are at least 13 edges incident to vertices in $N(v)$, by branching on v we get $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 5$ and $\mu_2 \geq 13$. This gives $y = 1.0658$.

But there is only one possibility with no domination and only 12 edges incident to vertices in $N(v)$: when u_1, u_2, u_3, u_4 is a 4-cycle. In this case, we can reduce the graph before branching. Any optimal solution cannot contain more than two vertices from the cycle. If it contains only one vertex, then replacing it by v does not change its size. Finally, there exist only three disjoint possibilities: keep u_1 and u_3, keep u_2 and u_4 or keep only v (Fig. 12). Hence, we can replace $N(v) \cup \{v\}$ by only two adjacent vertices u_1u_3 and u_2u_4, such that u is adjacent to u_1u_3 (resp., u_2u_4) if and only if u is adjacent to u_1 or to u_3 (resp., to u_2 or to u_4).

Case 2: All the Neighbors of v Have at Least 2 Outer Edges

If one of them has degree 4, then there are at least 13 edges removed when taking v, and we get again $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 5$ and $\mu_2 \geq 13$.

Otherwise, once v is removed, any u_i now has degree 2. Note that when folding a vertex of degree 2, we reduce the graph by 2 vertices and 2 edges (if the vertex dominates another one, this is even better). Since any two vertices u_i cannot be adjacent to each other, we can remove 8 vertices and at least 8 edges by folding u_1, u_2, u_3, u_4. Indeed, if for instance, u_1 dominates its neighbors (its two neighbors being adjacent), we remove 3 vertices and at least 5 edges which is even better. Removing 8 vertices and at least 8 edges is very interesting since it leads to $\nu_1 = 9$, $\mu_1 = 12$, $\nu_2 = 5$, $\mu_2 = 12$, and $y = 1.0420$.

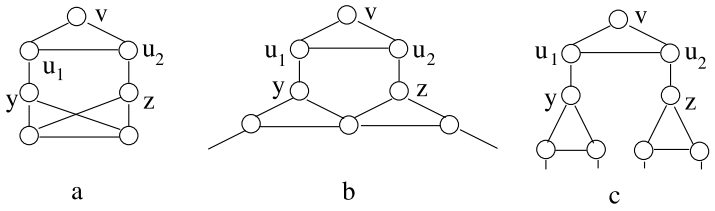


Fig. 13 Case 3

Case 3: u_1 Has Degree 3 and only One Outer Edge

Vertex u_1 has one inner edge, say (u_1, u_2). Let y be the third neighbor of u_1. We branch on y. Suppose first that u_2 has degree 3. If we take y we remove 4 vertices and (at least) 8 edges (there is at most one inner edge in $N(y)$); if we don't take y, then we remove also v and we remove globally 2 vertices and 7 edges.

Obviously, this is not sufficient. There exists an easily improvable case, when a neighbor of y has degree 4 (or when y itself has degree 4), or when the neighbors of y are not adjacent to each other. Indeed, in this case, there are at least 9 edges in $N(y)$, and we get $\nu_1 = 4$, $\mu_1 = 9$, $\nu_2 = 2$ and $\mu_2 \geq 7$, leading to $y = 1.0661$. Now, we can assume that y has degree 3, its three neighbors have also degree 3, and that the same holds for z, the neighbor of u_2. Furthermore, we assume that they both are part of a triangle (see Fig. 13).

We reason with respect to the quantity $|N(y) \cap N(z)|$. If $|N(y) \cap N(z)| = 2$, then either some neighbor of y has degree 4, or else v is a separator of size 1 (Fig. 13a). If $|N(y) \cap N(z)| = 1$, then their common neighbor has degree 4 (Fig. 13b). Finally, if $|N(y) \cap N(z)| = 0$, then at least a neighbor of, say, z is neither u_3 nor u_4. Hence, when discarding y, we take u_1, so remove u_2 and then add z to the solution. We get $\nu_1 = 4$, $\mu_1 = 8$, $\nu_2 = 7$ and $\mu_2 \geq 13$, leading to $y = 1.0581$; this situation is illustrated in Fig. 13c.

Suppose now that u_2 has degree 4. Then, when we don't take y, since we don't take v, u_1 has degree 1. Then, we can take it and remove u_2 and its incident edges. Then, when we don't take y, we remove in all 4 vertices and 10 edges. In other words, $\nu_1 = 4$, $\mu_1 = 8$, $\nu_2 = 4$ and $\mu_2 \geq 10$. This gives $y = 1.0642$.

Case 4: u_1 Has Degree 4 and only One Outer Edge

Since Case 1 does not occur, we can assume that there is a vertex (say u_4) of degree 3. Since Case 3 does not occur, u_4 has no inner edge. Hence, u_1 is adjacent to both u_2 and u_3. Then, there are only two possibilities.

If there are no other inner edges, since Case 3 does not occur, u_2 and u_3 have two outer edges, and we have in all 13 edges. This gives once again $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 5$ and $\mu_2 \geq 13$.

Otherwise, there is an edge between u_2 and u_3. Then, v, u_1, u_2, u_3 form a 4-clique (Fig. 14). We branch on u_4. If we take u_4, we delete $\nu_1 = 4$ vertices and (at least) $\nu_2 = 9$ edges (v has degree 4 and is not adjacent to other neighbors of u_4). If we

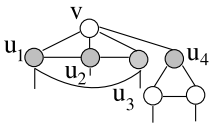


Fig. 14 Vertices v, u_1, u_2, u_3 form a 4-clique

discard u_4 then, by domination, we take v, and delete $\nu_2 = 5$ vertices and at least $\mu_2 = 12$ edges. So, $y = 1.0451$ and Case 4 is concluded.

To conclude, we have to verify that removing ν'_1 vertices and $\mu'_1 \geq \nu'_1$ edges (without branching) does not increase the running time. Indeed, this may occur when graph reductions are performed (such as a vertex folding for instance), but also in the previous analysis of the possible branchings, since it may happen that the real reduction removes $\nu_1 + \nu'_1$ vertices and $\mu_1 + \mu'_1$ edges, where $\mu'_1 \geq \nu'_1$. To get the result claimed, we have to verify that $\gamma^{n-\nu'_1} y^{2m-2\mu'_1-3n+3\nu'_1} \leq \gamma^n y^{2m-3n}$, or equivalently, that $y^{-2\mu'_1+3\nu'_1} \leq \gamma^{\nu'_1}$. This is trivially true as soon as $y \leq \gamma$, since $\nu'_1 \leq \mu'_1$. In other words, each time we remove ν'_1 vertices and at least ν'_1 edges, we reduce running time by a multiplicative factor $c^{\nu'_1}$ where $c = (y/\gamma) < 1$.

Similarly, a last issue we have to deal with is what happens if some branching disconnects our graph. In what follows, the case where some trees are created is handled.

Dealing with Trees

We show in what follows that it is never problematic to create a tree while branching, in the analysis in Theorem 3. This issue is important since removing a separate tree from the graph increases our complexity measure. Thus, for instance, we also show that the creation of trees is no more problematic when branching on a vertex of degree 3 for the algorithm in graphs of average degree 3 (see Sect. 3). Here, we consider the case where one or more trees are created when branching on a vertex v of degree 4 with neighbors u_1, u_2, u_3, u_4. Let $\mathcal{N}$ be the number of edges incident to some vertex in $N(v)$, and Θ (resp., Ω) be the set of edges, called inner edges, that have both endpoints in $N(v)$ (resp., the set of edges that have only one endpoint in $N(v)$).

Let us recall the concept of a *mirror* (see for example [9]). A vertex $m \in V$ is a mirror of $v \in V$, if $G[N(v) \setminus N(m)]$ is a clique, or equivalently, every combination of two non-adjacent vertices in $N(v)$ contains a vertex from $N(m)$. Whenever the algorithm branches on v and discards it, it can discard all its mirrors as well. This is because at least two neighbors of v should be in I in this branch since taking v is an alternative of equal size to taking only one.

Suppose first that one of the $l \geq 1$ trees created is a single vertex t. In this case, t is a mirror of v. When discarding v, we can also discard t removing 2 vertices and at least 7 edges. It is easy to see that no trees are separated. Indeed, this does not disconnect the graph since at least three u_i's are connected to the rest of the graph (the graph without $N[v]$ or the l trees), and each of the l trees is connected to at least one of these three u_i's while the fourth u_i is connected either to two trees or to some other u_i (or to the remainder of the graph). When taking v in I, we remove 5 vertices and $\mathcal{N} \geq |\Omega| \geq 4 + 3 + 3l$ edges (at least 3 edges per tree and 3 edges to the

rest of the graph). The removal of the trees removes l more vertices in the worst case (actually $l+k$ vertices and k edges, for some $k \geq 0$). In all, we have $\nu_1 = 2$, $\mu_1 = 7$, $\nu_2 = 5+l$ and $\mu_2 = 7+3l$, which is good for $l \geq 2$ (this gives $y = 1.0526$). If $l = 1$, then $\mathcal{N} \geq |\Omega| + \lceil (12 - |\Omega|)/2 \rceil \geq 10 + 1 = 11$, and hence the reduction we get is $\nu_1 = 2$, $\mu_1 = 7$, $\nu_2 = 6$ and $\mu_2 = 11$. This gives $y = 1.0630$.

Now, if $l \geq 2$ and each tree consists of at least 2 vertices, then there exist at least 4 edges linking each tree to $N(v)$. When taking v, we remove 5 vertices and at least $4+3+4l$ edges. When reducing the trees, we remove an additional amount of $2l$ vertices and l edges. In all, we remove $5+2l$ vertices and $7+5l$ edges. The worst case, of course, arises when $l = 2$, for which we have $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 9$ and $\mu_2 = 19$ (and $y = 1.0503$).

Consider now the final case where one tree T composed by at least 2 vertices is created while branching on v. Then, we have at least 4 edges linking $N(v)$ to T and 3 edges linking $N(v)$ to the remainder of the graph. In this case, $\mathcal{N} = |\Omega| + |\Theta| \geq 11 + |\Theta|$. When taking v, in the worst case we remove 7 vertices and $\mathcal{N}+1$ edges because we reduce a tree T of at least 2 vertices.

- If all neighbors of v have degree 4, then $\mathcal{N} \geq 11 + \lceil (16-11)/2 \rceil = 14$. In this case, $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 7$ and $\mu_2 = 15$. This gives $y = 1.0638$.
- For the rest of the cases, we first handle the case where *one neighbor of v has degree 3 and 3 neighbors have degree 4*. If there is at most one inner edge, then $|\Omega| \geq 13$ and $\mathcal{N} \geq 14$. Hence, the worst case is $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 7$ and $\mu_2 = 15$. Now, suppose there are two inner edges, and hence the tree consists of two degree 3 vertices t_1, t_2. If a vertex of the tree, say t_1, is a mirror of v, then we can discard t_1 as well when discarding v and we get $\nu_1 = 2$, $\mu_1 = 7$ (this does not create trees). With $\nu_2 = 7$ and $\mu_2 = 14$, it gives $y = 1.0466$. What remains are the three possibilities without a mirror. The first two possibilities occur when the two inner edges are (u_1, u_2) and (u_3, u_4). If one u_i, say u_3, is adjacent to both t_1 and t_2 (and then t_1 is adjacent to u_1 and t_2 to u_2), it is never interesting to take u_3 because we cannot take 3 vertices if we take u_3. We discard u_3 reducing the graph. The case where t_1 is adjacent to u_1 and u_3, and t_2 to u_2 and u_4 reduces as follows: we can replace the whole subgraph by two adjacent vertices u_1u_3 and u_2u_4 since either we take the two vertices v and t_1, or we take 3 vertices in the combinations u_1, u_3, t_2, or u_2, u_4, t_1. Finally, if the inner edges are (u_1, u_2) and (u_2, u_3), then to avoid having a mirror, u_2 must be adjacent to say t_1, also t_1 must be adjacent to u_4, and t_2 must be adjacent to u_1 and u_3. But, as previously, this case reduces by replacing the whole graph by two adjacent vertices u_1u_3 and u_2u_4.
- We now handle the case where *2 neighbors of v have degree 4, and 2 have degree 3*. There is at most one inner edge. If there is no inner edge, then $\mathcal{N} = |\Omega| = 14$ and $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 7$ and $\mu_2 = 15$. If there is one inner edge (u_1, u_2), and if u_3 or u_4 has degree 3, then we can fold two (non adjacent) vertices of degree 2 when discarding v. This gives $\nu_1 = 5$, $\mu_1 = 8$, $\nu_2 = 7$ and $\mu_2 = 14$ ($y = 1.0604$). If u_1 and u_2 have degree 3, then to avoid separators of size 2, u_1 is adjacent to say t_1 and u_2 is not adjacent to the tree. In this case, t_2 is a mirror of v and we get $\nu_1 = 2$, $\mu_1 = 7$, $\nu_2 = 7$ and $\mu_2 = 14$.
- We now consider the case where *one neighbor of v has degree 4 and the other neighbors of v have degree 3*. Because of the degrees, there cannot be more than

one inner edge. If there is no inner edge, then $\mathcal{N} \geq 13$ and, as previously, by folding the 3 (pairwise non adjacent) vertices of degree 3 when not taking v, we get $\nu_1 = 7$, $\mu_1 = 10$, $\nu_2 = 7$ and $\mu_2 = 14$ ($y = 1.0463$). If there is an inner edge, say (u_1, u_2), then no need to branch. Indeed, the tree has only two vertices t_1, t_2 that are of degree 3 (otherwise there would exist 12 edges in Ω). If, say, t_1 is adjacent to both u_1 and u_2, then to avoid domination, u_1 and u_2 must be incident to a fourth edge. If t_1 is adjacent to both u_3 and u_4, then it is never interesting to take t_1 and we discard it. Indeed, it is impossible to take t_1 plus 2 other vertices, and we can always take v and t_2. If t_1 is adjacent to u_1 and u_3, and t_2 is adjacent to u_2 and u_3, then at least 2 vertices among u_1, u_2, u_3 have degree 4 since 2 of them must be adjacent to the remainder of the graph. The only remaining case occurs when t_1 is adjacent to u_1 and u_3, and t_2 is adjacent to u_2 and u_4. In this case, we can replace the whole subgraph by two adjacent vertices u_1u_3 and u_2u_4. Indeed, either we take two vertices (v and t_1), or we take three vertices (either u_1, u_3, t_2, or u_2, u_4, t_1).

- Finally, if all neighbors of v have degree 3, then since $|\Omega| \geq 11$ and $|\Theta| = 0$, we have $\mathcal{N} = |\Omega| = 12$. In this case, when we do not take v in I, we have 4 vertices of degree 3 pairwise non adjacent. We can fold each of them (if there is domination this is even better) and delete 8 more vertices and edges. We get a worst case of $\nu_1 = 9$, $\mu_1 = 12$, $\nu_2 = 7$ and $\mu_2 = 14$ ($y = 1.0192$).

Discussion above about trees occurrences has as corollary that we can assume that during algorithm's unraveling only connected components C_i each verifying $m_i \geq n_i$ occur. In order to simplify our notation we denote by $\bar{T}$ the complexity of this case. Running time now verifies:

$$T(m,n) \leq \sum_i \bar{T}(m_i, n_i) = \sum_i \bar{T}\left(m - \sum_{j\neq i} m_j, n - \sum_{j\neq i} n_j\right)$$

$$\leq \sum_i \bar{T}\left(m - \sum_{j\neq i} (m_j - n_j), n\right) c^{n-n_i} \leq \bar{T}(m,n) c^n \sum_i \frac{1}{c^{n_i}}$$

Since $c < 1$ (and $n_1 \geq 1$), for n large enough we have $c^n \sum_i \frac{1}{c^{n_i}} \leq 1$ and therefore $T(m,n) \leq \bar{T}(m,n)$. The proof of Theorem 3 is now completed. □

5 When Bottom-Up Meets Measure and Conquer: Final Improvements and an Algorithm in General Graphs

In this section we devise algorithms for graphs of maximum degree 5 and 6 as well as for general graphs. The algorithm follows the same line as the one devised in [13]. There, depending on the maximum degree of the graph, a branching is performed and a measure and conquer technique is used to analyse the running time: vertices of degree d receive a weight $w_d \geq 0$ which is non-decreasing with d (with $w_d = 1$ for $d \geq 7$ and $w_1 = w_2 = 0$). The running time of the algorithm is measured as a function of the total weight of the graph (initially smaller than n). In other words,

running times are expressed as $T(n) = O^*(c^{\sum_{i \in V} w_i})$, where $\sum_{i \in V} w_i \leq n$. When a branching on a vertex of degree d is done, the decreasing δ_d of the total weight of the graph is measured. Weights are then optimized in such a way that δ_d leads to the same complexity (neglecting polynomial terms) for any d. Weights are subject to some constraints, such as, for example, the fact that reduction rules must not increase the total weight of the graph.

Here, we modify the algorithm above and its analysis in order to improve the running time for MAX INDEPENDENT SET, by incorporating our efficient algorithms able to solve MAX INDEPENDENT SET in graphs of maximum degree 3 and 4. We will use these algorithms when the input graph has degree at most 4 (see Theorem 4). Note that this technique could be also used to improve for instance the running time of the algorithm in [9]; we will give the corresponding bounds at the end of the section.

Theorem 4 *It is possible to optimally solve* MAX INDEPENDENT SET *in* $O^*(1.2114^n)$.

Proof According to what has been discussed in Sect. 4, it is possible to solve MAX INDEPENDENT SET in graphs of maximum degree Δ with running time upper bounded by $O^*(\gamma_\Delta^n)$, where $\gamma_3 = 1.08537$ and $\gamma_4 = 1.1571$.

The algorithm we propose is the same as in [13] up to the fact that we use our algorithms for degree 3 and 4 when the maximum degree Δ of the graph G is 4 or less. Formally:

1. while the maximum degree $\Delta \geq 5$ do
   (a) apply reduction rules (let us note that the reduction rules in [13] include vertex folding and the removal of dominated vertices; another reduction is used, but since we use exactly the same procedure, it does not seem worth giving all the details, and we refer the interested reader to [13]);
   (b) if there are several connected components, solve the problem separately on each connected component;
   (c) otherwise, select a vertex v of degree Δ and perform a branching on it as in [13] (again, for more details the reader can be referred to [13]);
2. if $\Delta \leq 2$ solve the problem in polynomial time;
3. if $\Delta = 3$ use the algorithm described in Sect. 3;
4. if $\Delta = 4$ use the algorithm of Sect. 4.

For analysis of the running time, we use measure and conquer techniques as in [9, 13]. Each vertex v of degree d receives a weight $w_d \in [0, 1]$, non decreasing with d, with $w_d = 0$ for $d = 0, 1, 2$, and with $w_d = 1$ for $d \geq 8$. The analysis seeks a complexity in $O^*(\gamma^W)$ where W is the total weight of the graph. Since initially $W = \sum_{v \in V} w(v) \leq n$, the complexity bound is $O^*(\gamma^n)$. Weights are optimized in order to determine the best γ. In order that the analysis is valid, weights must fulfill some constraints, for instance, that reduction rules must not increase the total weight of the graph. As shown in [9, 13] the constraints are:

(c1) $w_{d_1} + w_{d_2} - w_{d_1+d_2-2} \geq 0$ for all d_1, d_2 in $\{2, 3, \ldots, 8\}$ (not increasing the weight while doing a vertex folding)
(c2) $w_i - w_{i-1} \geq w_{i+1} - w_i$ for $i \geq 3$ (useful for the branching analysis).

Here, we add two more constraints. Indeed, the recursive argument is that the problem is solvable in $O^*(\gamma^W)$. Since, if the graph has degree at most 4 we use our algorithm, the running time has to be valid in this case also. If the graph has degree at most 2 this is trivial, otherwise let n_3 and n_4 be the number of vertices of degree 3 and 4 in the graph ($n_3 + n_4 = n$ since vertices of degree 2 are removed by vertex foldings). The running time obtained in Theorem 3 for graphs of average degree between 3 and 4 is $O^*(\gamma_3^n y^{3m-2n}) = O^*(\gamma_3^n(\gamma_4/\gamma_3)^{2m-3n})$ where γ_3 and γ_4 are the complexity of our algorithms for average degree 3 and 4. Since $2m - 3n = (d-3)n$, where d is the average degree, in order that the analysis is valid the weights must satisfy:

$$\gamma_3^n \left(\frac{\gamma_4}{\gamma_3}\right)^{(d-3)n} \leq \gamma^{w_3 n_3 + w_4 n_4}$$

$$\iff \quad \log\gamma_3 + (d-3)(\log\gamma_4 - \log\gamma_3) \leq \log\gamma\,((4-d)w_3 + (d-3)w_4)$$

Notice that $4-d$ and $d-3$ are nonnegative numbers, thus this inequality is a consequence of the two following constraints that we add:

(c3) $w_3 \geq \frac{\log\gamma_3}{\log\gamma}$

(c4) $w_4 \geq \frac{\log\gamma_4}{\log\gamma}$.

With these constraints, the running time is valid for $\Delta \leq 4$. For $\Delta \geq 5$, we use the analysis in [13]. There, for each value of Δ, an analysis is done showing that in the worst case, branching on a vertex of degree Δ as in step (1) leads to a decrease of W either by μ_Δ or by ν_Δ. This gives a recurrence relation ($1 \leq \gamma^{-\mu_\Delta} + \gamma^{-\nu_\Delta}$) for all $\Delta \geq 3$, and the worst one specifies the running time $O^*(\gamma^W)$. Here, no need to consider the cases of branching on degrees 3 and 4 anymore, since they are handled by our low degree algorithms. Hence, we only need the recurrence relation for $\Delta \geq 5$.

The best values we have found are $w_3 = 0.45$, $w_4 = 0.7609$, $w_5 = 0.9045$, $w_6 = 0.9721$, $w_7 = 0.9977$, satisfying all the constraints and leading to $\gamma = 1.2114$. □

Corollary 1 *On graphs of maximum degree* 5, MAX INDEPENDENT SET *can be solved with running time* $O^*(1.1895^n)$. *On graphs of maximum degree* 6, MAX INDEPENDENT SET *can be solved with running time* $O^*(1.2050^n)$.

Both these results are consequences of Theorem 4. Remark that, for $v \in V$, $w(v) \leq w_\Delta n$. So, for the cases handled in Corollary 1, $\sum_{v\in V} w(v) \leq w_6 n$ and $\sum_{v\in V} w(v) \leq w_5 n$ in graphs with maximum degree $\Delta = 6$ and $\Delta = 5$, respectively.

Let us note that, using the algorithm in [9] that runs in time $O^*(1.2201^n)$, with a similar approach we obtain respectively $O^*(1.1918^n)$, $O^*(1.2071^n)$ and $O^*(1.2127^n)$ for solving MAX INDEPENDENT SET in graphs of maximum degree 5, 6 and in general graphs.

6 Conclusion

The complexity measure of the method proposed in the paper may appear, at least for the two problems considered, as an adaptation of measure and conquer for instances with bounded density. Indeed, for example, $\gamma_d^m y^{2m-dn}$ can be easily written

as $2^{\sum_i w_i n_i}$ where n_i is the number of vertices of degree i. However, two points must be underlined. First, the way we seek the complexity in the bottom-up method actually specifies the strong links that have to be verified between weights in order to use as efficiently as possible an algorithm for lower degree graphs (or, more generally for low density instances). The second point is to exhibit and to fruitfully use the link between density and branching. A recursive application of the method then allows to take into account situations where a good branching necessarily occurs to derive good complexity bounds.

Though the precise links between bottom-up and measure and conquer is not very clear yet, at least these two points seem not to be considered in a usual measure and conquer analysis. Furthermore, even if the bound of $O^*(1.1^n)$ mentioned in [20] is not yet beaten, the results obtained for MAX INDEPENDENT SET in this article, the best known until now, are interesting per se.

Acknowledgements The very useful comments and suggestions of three anonymous referees are gratefully acknowledged. Many thanks to one of them for having pointed out to us the paper of [6].

References

1. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: Proc. Symposium on Discrete Algorithms, SODA'99, pp. 856–857 (1999)
2. Bourgeois, N., Escoffier, B., Paschos, V.Th.: An $O^*(1.0977^n)$ exact algorithm for MAX INDEPENDENT SET in sparse graphs. In: Grohe, M., Niedermeier, R. (eds.) Proc. International Workshop on Exact and Parameterized Computation, IWPEC'08. Lecture Notes in Computer Science, vol. 5018, pp. 55–65. Springer, Berlin (2008)
3. Bourgeois, N., Escoffier, B., Paschos, V.Th., Van Rooij, J.M.M: Maximum independent set in graphs of average degree at most three in $O(1.08537^n)$. In: Kratochvíl, J., Li, A., Fiala, J., Kolman, P. (eds.) Proc. Conference on Theory and Applications of Models of Computation, TAMC'10. Lecture Notes in Computer Science, vol. 6108, pp. 373–384. Springer, Berlin (2010)
4. Chen, J., Liu, L., Jia, W.: Improvement on vertex cover for low-degree graphs. Networks **35**(4), 253–259 (2000)
5. Chen, J., Kanj, I.A., Xia, G.: Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. Algorithmica **43**(4), 245–273 (2005)
6. Dahllöf, V., Jonsson, P., Wahlström, M.: Counting models for 2SAT and 3SAT formulæ. Theor. Comput. Sci. **332**(1–3), 265–291 (2005)
7. Fomin, F.V., Høie, K.: Pathwidth of cubic graphs and exact algorithms. Inf. Process. Lett. **97**(5), 191–196 (2006)
8. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On two techniques of combining branching and treewidth. Algorithmica **54**(2), 181–207 (2009)
9. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure & conquer approach for the analysis of exact algorithms. J. Assoc. Comput. Mach. **56**(5), 1–32 (2009)
10. Fürer, M.: A faster algorithm for finding maximum independent sets in sparse graphs. In: Corea, J.R., Hevia, A., Kiwi, M. (eds.) Proc. Latin American Symposium on Theoretical Informatics, LATIN'06. Lecture Notes in Computer Science, vol. 3887, pp. 491–501. Springer, Berlin (2006)
11. Fürer, M., Kasiviswanathan, S.P.: Algorithms for counting 2-SAT solutions and colorings with applications. In: Kao, M.-Y., Li, X.-Y. (eds.) Proc. Algorithmic Aspects in Information and Management, AAIM'07. Lecture Notes in Computer Science, vol. 4508, pp. 47–57. Springer, Berlin (2007)
12. Goldberg, M.K., Spencer, T.H., Berque, D.A.: A low-exponential algorithm for counting vertex covers. In: Graph Theory, Combinatorics and Algorithms. Proc. 7th Quadrennial International Conference on the Theory and Application of Graphs, vol. 1, pp. 431–444. Wiley, New York (1992)
13. Kneis, J., Langer, A., Rossmanith, P.: A fine-grained analysis of a simple independent set algorithm. In: Kannan, R., Narayan Kumar, K. (eds.) Proc. Foundations of Software Technology and Theoretical Computer Science, FSTTCS'09. LIPIcs, vol. 4, pp. 287–298. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik (2009)

14. Kojevnikov, A., Kulikov, A.S.: A new approach to proving upper bounds for max-2-sat. In: Proc. Symposium on Discrete Algorithms, SODA'06, pp. 11–17 (2006)
15. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. J. Discrete Algorithms **7**, 191–212 (2009)
16. Robson, J.M.: Algorithms for maximum independent sets. J. Algorithms **7**(3), 425–440 (1986)
17. Robson, J.M.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université de Bordeaux I (2001)
18. van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer, a faster exact algorithm for dominating set. In: Albers, S., Weil, P. (eds.) Proc. International Symposium on Theoretical Aspects of Computer Science, STACS'08, pp. 657–668. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany (2008)
19. Wahlström, M.: A tighter bound for counting max-weight solutions to 2sat instances. In: Grohe, M., Niedermeier, R. (eds.) Proc. Parameterized and Exact Computation, Third International Workshop, IWPEC'08. Lecture Notes in Computer Science, vol. 5018, pp. 202–213. Springer, Berlin (2008)
20. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Juenger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization—Eureka! You shrink! Lecture Notes in Computer Science, vol. 2570, pp. 185–207. Springer, Berlin (2003)
21. Xiao, M.: New branching rules: improvements on independent set and vertex cover in sparse graphs. CoRR (2009). arXiv:0904.2712