Fast Algorithms for max independent set

Nicolas Bourgeois, Bruno Escoffier, Vangelis Paschos, Johan van Rooij

Fast algorithms for MAX INDEPENDENT SET[*]

N. Bourgeois[1] B. Escoffier[1] V. Th. Paschos[1] J.M.M. van Rooij[2]

[1] LAMSADE, CNRS and Université Paris-Dauphine, France
{bourgeois,escoffier,paschos}@lamsade.dauphine.fr

[2] Department of Information and Computing Sciences
Universiteit Utrecht, The Netherlands, johanvr@cs.uu.nl

January 26, 2010

Abstract

MAX INDEPENDENT SET is a paradigmatic problem in theoretical computer science and numerous studies tackle its resolution by exact algorithms with non-trivial worst-case complexity, both in the general case and in the case of bounded degree graphs. Here, we improve several of these results. We first get an algorithm in $O^*(1.0854^n)$ for graphs of average degree 3 using a case analysis based upon a detailed case analysis using several reduction rules. Then we propose a generic method showing how improvement of the worst-case complexity for MAX INDEPENDENT SET in graphs of average degree d entails improvement of it in any graph of average degree greater than d and, based upon it, we tackle MAX INDEPENDENT SET by improving its complexity in graphs of average degree 4, 5 and 6. Finally, we combine this method with measure and conquer techniques to get improved running times for general graphs. The best computation bounds obtained are $O^*(1.1571^n)$, $O^*(1.1918^n)$ and $O^*(1.2070^n)$, for graphs of maximum degree 4, 5 and 6 respectively, and $O^*(1.2125^n)$ for general graphs. These results improve upon the best known results for these cases.

1 Introduction

Very active research has been recently conducted around the development of optimal algorithms for **NP**-hard problems with non-trivial worst-case complexity (see the seminal paper [10] 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.

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 the best published worst-case complexity bound is, to our knowledge, the $O^*(1.2210^n)$ bound [5]. We also quote the $O^*(1.1889^n)$ bound claimed in the unpublished technical report [9].

Let $T(\cdot)$ be a super-polynomial and $p(\cdot)$ be a polynomial, both on integers. In what follows, using notations in [10], for an integer n, we express running-time bounds of the form $p(n) \cdot T(n)$ as $O^*(T(n))$, the star meaning that we ignore polynomial factors. We denote by $T(n)$ the worst-case time required to exactly solve the considered combinatorial optimization problem on an instance of size n. We recall (see, for instance, [4]) that, if it is possible to bound above $T(n)$ by

[*]Work partially supported by the French Agency for Research under the DEFIS program TODO, ANR-09-EMER-010.

a recurrence expression of the type $T(n) \leq \sum T(n-r_i)+O(p(n))$, we have $\sum T(n-r_i)+O(p(n)) = O^*(\alpha(r_1, r_2, \ldots)^n)$ where $\alpha(r_1, r_2, \ldots)$ is the largest root of the function $f(x) = 1 - \sum x^{-r_i}$.

In this paper we consider MAX INDEPENDENT SET in general graphs, but also on (connected) graphs with "small" average degree, more precisely with average degree at most 3, 4, 5 and 6, and on graphs of small bounded degree.

Dealing with MAX INDEPENDENT SET in graphs of maximum degree 3, faster and faster algorithms have been devised for optimally solving this problem. Let us quote the $O^*(1.1259^n)$ algorithm in [1], the $O^*(1.1254)$ algorithm in [3], the $O^*(1.1120)$ algorithm in [6], the $O^*(1.1034^n)$ algorithm in [7], the $O^*(1.0977^n)$ algorithm by [2] and, finally, the recent $O^*(1.0892^n)$ algorithm in [8]. As a first result, we improve in this paper the bound of [8] down to $O^*(1.0854^n)$ by proposing a finer and more detailed case analysis based upon powerful reduction rules (Section 2). Our result remains valid also for connected graphs of average degree bounded by 3.

We then propose a generic method extending improvements of the worst-case complexity for MAX INDEPENDENT SET in graphs of average degree d to graphs of average degree greater than d. This "bottom-up" method of carrying improvements of time-bounds for restrictive cases of a problem to less restrictive ones (the latter including the former) is, as far as we know, a new method that could be very useful for strengthening time-bounds not only for MAX INDEPENDENT SET but also for other graph-problems where local worst configurations appear when maximum degree is small.

In order to informally sketch the method, suppose that one knows how to solve the problem 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 (namely on its average degree), *and on the value* γ_d (see Section 3). In other words, the form of the running time we look for 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 yields improvements for higher average degrees. A direct application of this method leads, for instance, for MAX INDEPENDENT SET in graphs with average degree 4, to an upper complexity bound that already slightly outperforms the best known bound of [1] (Section 3). This result is further improved down to $O^*(1.1571^n)$ (Section 4) with a more refined case analysis.

Finally, in section 5, we combine measure and conquer techniques with ours to compute MAX INDEPENDENT SET within running time $O^*(1.2125^n)$ on general graphs, thus improving the best known published result of $O^*(1.2210^n)$ [5]. Furthermore, in graphs of maximum degree at most 5 and 6, we provide bounds of $O^*(1.1918^n)$ and $O^*(1.2070^n)$, respectively, that improve upon the corresponding bounds of $O^*(1.2023^n)$ and $O^*(1.2172^n)$ in graphs of maximum degree 5 and 6 by [5].

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

2 Graphs of average degree at most 3

We propose in this section an $O^*(1.0854^n)$ branch-and-reduce algorithm for the MAXIMUM INDEPENDENT SET problem on connected graphs of average degree at most 3 (or for graphs where each connected component has average degree at most 3).

Informally, our algorithm works as follows. First it applies a series of well known reduction rules that simplify the instance without branching. Next, it looks for vertex separators (i.e., for sets of vertices the removal of which disconnect the graph) of size 1 or 2 in the graph and uses them to further simplify the instance. Then, 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 branches on a vertex in the graph: in one branch it is taken in the independent set and in the other branch it is discarded. We will show that we can always branch in a way that allows a worst case running time of $O^*(1.0854^n)$.

As in [2, 6], we use the quantity $m-n$ as complexity measure in the analysis of the algorithm, where m is the number of edges and n the number of vertices of the graph. The resulting upper bound on the running time of $O^*(\gamma^{m-n})$ implies a $O^*(\gamma^{0.5n})$ algorithm on graphs in which each connected component has average degree at most 3. The bound on the average degree of each connected component exists because, when connected components are trees, $m-n=-1$; these trees are removed by reduction rules possibly increasing $m-n$ to over $0.5n$ for the remaining graph.

2.1 Simple reduction rules and small separators

Before branching, our algorithm applies the following well known reduction rules. They are thoroughly described in many publications [2, 5, 6] and require no further explanation. Let I be the independent set that is being computed by our algorithm in the input graph G.

First, if G is disconnected, solve each connected component separately. Also, take degree-0 and 1 vertices in I and remove all the neighbors of the latter ones (degree-1 vertices). Moreover, if for any two vertices u, v: $N[u] \subseteq N[v]$, then we say that u *dominates* v and we remove v. This domination rule is correct because in any maximum independent set containing v, v can be replaced by u. Notice that domination forces degree 2 vertices with adjacent neighbors to be in I.

If there exists a vertex v of degree 2 with non-adjacent neighbors u, w, the algorithm removes v, merges u and w into a single vertex, and adds 1 to the size of I. This degree 2 rule is also called *vertex folding*. Its correctness is based upon the fact that if v is not in I, then we can put both neighbors in I because taking v gives an alternative of the same size to the fact of taking only one neighbor of v.

If these reduction rules do not apply, the graph is of minimum degree 3. The algorithm then follows the approach of [6] and looks for vertex separators of size at most 2. If the graph contains a vertex separator of size 1 or 2, recursively solve the smallest component and adjust the rest of the graph to the computed solution. For completeness, we have included the details from [6] in Section 2.1.1.

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. 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 can replace this local configuration with a smaller equivalent one. How this reduction rule works is explained in Section 2.1.2.

2.1.1 Small vertex separators

Following the approach of [6], we exploit vertex separators of size 1 and 2. Although using these separators is always beneficial, we only need separators related to a component of constant size to prove our time bound. In this case, the graph is reduced in constant time.

Let v be an articulation point of G (i.e., a vertex whose removal disconnects G) and let $C \subset V$ be the vertices of the smallest component (vertices in C only have edges to v or to other vertices in C). If the algorithm finds such an articulation point v, it recursively computes the maximum independent sets $I_{\not v}$ (those that do not contain v) in the subgraph $G[C]$ and I_v in the subgraph $G[C \cup \{v\}]$. Notice that $|I_v|$ can be at most 1 larger than $|I_{\not v}|$, and if this is the case then $v \in I_v$. If these sizes are the same, the algorithm recursively computes the maximum independent set I in $G[V \setminus (C \cup \{v\})]$ and returns $I \cup I_v$. This is correct since taking v in the independent set restricts the possibilities in $G[V \setminus (C \cup \{v\})]$ more, while it does not increase the maximum independent set in $C \cup \{v\}$. Furthermore, if $|I_v| = 1 + |I_{\not v}|$, then the algorithm computes the maximum independent set I in $G[V \setminus C]$ and returns $I \cup (I_v \setminus \{v\})$. This is also correct since adding v to C increases the size of the maximum independent set in $G[C]$ by 1; this choice is now left to the recursive call on $G[V \setminus C]$.

If the algorithm finds a 2-separator $\{u, v\}$ of a constant size component $C \subset V$, then it computes a maximum independent set in the four subgraphs induced by C and any combination of vertices from the separator. Let $I_{\not v,\not u}$ be the computed maximum independent set in $G[C]$, $I_{v,\not u}$ the computed maximum independent set in $G[C \cup \{v\}]$, $I_{\not v,u}$ the computed maximum independent set in $G[C \cup \{u\}]$, and $I_{v,u}$ the computed maximum independent set in $G[C \cup \{u, v\}]$. Now consider the following possible cases:

1. $|I_{v,u}| = |I_{\not v,\not u}| + 2$, and hence $|I_{v,\not u}| = |I_{\not v,u}| = |I_{\not v,\not u}| + 1$. The algorithm now computes a maximum independent set in $G[V \setminus C]$ and returns $I \cup J$ where J is the set from $\{I_{\not v,\not u}, I_{v,\not u}, I_{\not v,u}, I_{v,u}\}$ that agrees with I on u and v.

2. $|I_{v,\not u}| = |I_{\not v,u}| = |I_{v,u}| = |I_{\not v,\not u}| + 1$. Let G' be $G[V \setminus C]$ with an additional edge added between u and v. Similar to the previous case, the algorithm computes a maximum independent set in G' and returns $I \cup J$, where J is one of the four possible independent sets that agree on u and v.

3. $|I_{v,\not u}| = |I_{\not v,\not u}|$ and $|I_{\not v,u}| = |I_{v,u}| = |I_{\not v,\not u}| + 1$ (and the symmetric case). v can now safely be discarded since it does not help increasing the size of the independent set in $C \cup \{v\}$. The algorithm recursively computes maximum independent set I in $G[V \setminus (C \cup \{v\}]$ and returns $I \cup J$, where J is the independent set from $\{I_{\not v,\not u}, I_{\not v,u}\}$ that agrees on u.

4. $|I_{\not v,u}| = |I_{v,\not u}| = |I_{\not v,\not u}|$ and $|I_{v,u}| = |I_{\not v,\not u}| + 1$. Let G' be $G[V \setminus C]$ with u and v merged into a single vertex w. The algorithm makes a recursive call on G' returning I. If $w \in I$ then we return $(I \setminus \{w\}) \cup I_{v,u}$ and otherwise we return $I \cup I_{\not v,\not u}$.

5. $|I_{v,u}| = |I_{\not v,u}| = |I_{v,\not u}| = |I_{\not v,\not u}|$. Now it is safe to use $I_{\not v,\not u}$. We make a recursive call on $G[V \setminus (C \cup \{u, v\})]$ resulting in I and return $I \cup I_{\not v,\not u}$.

In each case, we decide whether discarding u and/or v is optimal. If they cannot be discarded, we let the recursive call on the larger component decide on their membership of the maximum independent set.

2.1.2 Details of the tree separators rule

Whenever the closed neighborhood of a degree 3 vertex separates trees from the rest of the graph in a maximum degree 4 graph, we claim that we can replace the instance by a smaller equivalent one. In this section, we give a proof of this claim.

Let a, b, c be the neighbors of v. Notice that they all have at least one edge not incident to v or to the tree T, because otherwise there exists a small vertex separator. Hence there are at least three and at most six edges between $N(v)$ and T, and there exists at most one edge in $G[N(v)]$.

First consider the case where there exists an edge in $N(v)$ and hence $|T| \leq 2$. In this case, the maximum independent set in $G[N[v] \cup T]$ is of size 2, and hence we can safely pick those vertices in I that pose no restrictions on the remaining graph: v and one vertex from T. This is easy to see if T is a single vertex, namely taking v or T in I forbids taking a, b and c, and because of the edge in $N(v)$ it is not possible to take all tree vertices of $N(v)$. If $|T| = 2$ and without loss of generality a is connected to both vertices in T, then we cannot take three vertices if we take v because this forbids taking a, b and c while there is an edge in T. Also, we cannot take three vertices if we take a because this forbids taking v and T while there is an edge between b and c. The remaining vertices form a 4-cycle, and thus I has at most two vertices from $N(v) \cup T$.

Secondly, if there is no edge in $N(v)$ and $|T| \leq 2$, then we merge $N[v] \cup T$ to a single vertex and add 2 to the size of I. This is similar to vertex folding. Namely, the only maximum independent set in $G[N[v] \cup T]$ equals $N(v)$, while if we do not take these three vertices we can safely pick the non-restricting size 2 option consisting of v and one vertex from T. Merging the local structure to a single vertex postpones this choice. This is clearly correct if $|T| = 1$ since taking v or T in I forbids taking any vertex in $N(v)$. Furthermore, if $|T| = 2$, then taking v in I again forbids taking any vertex in $N(v)$ and taking any vertex of T in I forbids taking anything except for v and one of its neighbors: this again does not allow three vertices to be picked.

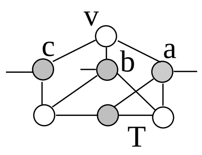

Figure 1: T consists of three vertices and at least two neighbors of v, a and b, have two tree neighbors.

Finally, if $|T| \geq 3$, then we can safely pick v and a maximum independent set in $G[T]$. Let us first look at the case where $|T| = 3$. At least two neighbors of v, say a and b have two neighbors in T (Figure 1). If we would have taken a in I, then we would forbid all vertices in $N(v) \cup T$ except for b, c and one vertex in T. By adjacency, we can take only two of these vertices making our initial choice a safe alternative. The same argument goes for taking b in I. Thus, with a and b discarded, we can safely pick the degree 1 vertex v and hence also the maximum independent set in $G[T]$ in I. What remains is the case $|T| = 4$. If T has three leaves, then there is only one way to pick four vertices in I: v and the three leaves of T. This does not restrict any choice in the rest of the graph and hence is optimal. Otherwise, T is a 4-vertex path and all local configurations of $G[N[v] \cup T]$ have a maximum independent set of size 3. Again, picking v and a maximum independent set in $G[T]$ is a safe choice.

Now let us look at the effect of the reduction rules to the $m - n$ measure. The measure is invariant under the degree 1 and 2 rules: they remove as many edges as vertices. The degree 0 rule increases the measure, but we always treat this (one vertex) tree separately. Furthermore, after application of these rules, the domination and small/tree separator rules decrease $m - n$ by at least 2.

2.2 The branching of the algorithm

If no reduction rule applies and no small separator exists, then our algorithm branches, producing several subproblems that are solved recursively. In one branch, a vertex is taken in I and hence it is removed together with its neighborhood. In the other, the vertex is not taken in I and is removed. We thoroughly analyse the local structures involved, removing more vertices and edges when possible.

The description of the branching of our algorithm is described by means of four lemmata given in what follows. Before describing the branching, we need the concept of a *mirror* (see for example [5]). A vertex $v' \in V$ is a mirror of $v \in V$ if $G[N(v) \setminus N(v')]$ is a clique, or equivalently, every combination of two non-adjacent vertices in $N(v)$ contains a vertex from $N(v')$. 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 since taking v is an alternative of equal size to taking only one.

In the analysis, we denote by $T(k)$ the number of subproblems generated when branching on a graph G of complexity $m - n = k$. 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 separator rule. 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. 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.

We now state the four lemmata concerning the branching of our algorithm. Let us recall that vertex foldings may increase the degrees of the vertices of the remaining graph. In the statements of the lemmata below, the graph G is indeed the remaining graph after some of its vertices are fixed or folded. We first deal with the case of (current) graphs of maximum degree 4 (Lemma 1), next with graphs of degree at least 5 (Lemma 2); finally, we settle in Lemmata 3 and 4 the case of graphs of maximum degree 3.

Lemma 1 *If G is of maximum degree 4, then either one of the following occurs:*

1. *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, then: $T(k) \leq T(k-5) + T(k-6)$ or $T(k) \leq 2T(k-8) + 2T(k-12)$.*
2. *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 is no constraint on the degree 3 vertices, then: $T(k) \leq T(k-4)+T(k-6)$ or $T(k) \leq 2T(k-8) + 2T(k-12)$.*
3. *The previous do not apply and then $T(k) \leq T(k-3) + T(k-7)$.*

The proof of Lemma 1, strongly relies on the fact that we can perform very efficient branching on graphs with degree 4 vertices. Moreover, we 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 1 uses the fact that in a 3-cycle containing a degree 3 vertex we can often branch in such a way that we can apply the domination rule. Furthermore, in a 4-cycle containing a degree 3 vertex, we can use that the vertex opposite the degree 3 vertex is a mirror of this vertex. Actually, the only 4-cycles in which this does not happen are 4-cycles consisting only of degree 4 vertices. These observations are similar to those used in the proof of Lemma 4, but in contrast, the proof of Lemma 1 given just below uses a more detailed case analysis.

Proof. We start the proof 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 causes domination which results in w being taken in the maximum independent set.

If v is of degree 4, discarding v 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 at least 8 more edges and 5 vertices. If in this last case all neighbors of v are of degree 3, then there are 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 be 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 cause even more edges to be removed compensating for any possible additional tree.

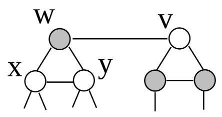

Figure 2: Vertex v has degree 3.

If v is of degree 3 (Figure 2), discarding it and taking w instead, leads to the removal of 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: $T(k-4)$.

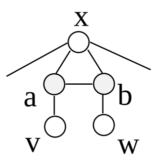

Figure 3: Triangles with one degree 4 vertex and two degree 3 vertices.

3-cycles with one degree 4 vertex and two degree 3 vertices

When there is 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 (Figure 3). By domination, we know that v and w are not adjacent to x, and that $v \neq w$. 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 a being 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.

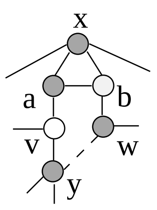

Figure 4: Vertex v, has a degree 4 neighbor y.

Consider the case where v or w, say v, has a degree 4 neighbor y (Figure 4). 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 from which 7 external edges and 7 vertices. Because of the small separators, there can be at most 2 additional adjacencies in the worst case leaving 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 only leads to $T(k-4)$ which now is enough. So, we can assume v and w to be of degree 3 and have no degree 4 neighbors.

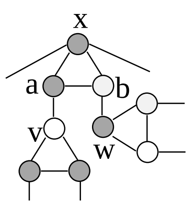

Figure 5: Vertex v is part of a triangle.

Suppose that v or w, say v, is part of a 3-cycle (Figure 5). 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 from which 6 external edges and 7 vertices. Because of the small separator rules, 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.

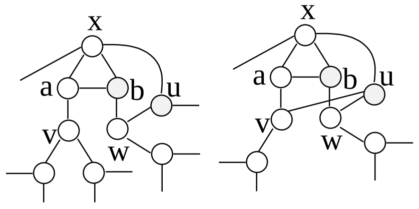

Figure 6: Vertex w, has a neighbor $u \neq a, b$ that is adjacent to x.

Suppose that v or w, say w, has a neighbor $u \neq a, b$ that is adjacent to x (Figure 6). 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 fires when considering branching on u (without actually branching on u of course) because this would create the size 2 tree $\{a, b\}$. Hence, now we can also assume that v and w have no neighbors besides a and b that are adjacent 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: $2 \leq |X| \leq 4$.

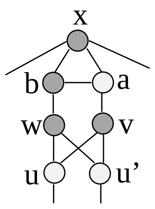

Figure 7: Vertices v and w are adjacent to both of $u, u' \in X$.

If $|X| = 2$, v and w are adjacent to both vertices in X (Figure 7). 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. 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 also results in a small separator or even a constant size component. Furthermore, there cannot exist additional adjacencies because then there also exist a small separator.

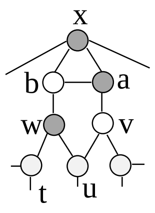

Figure 8: The case $|X| = 3$.

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 (Figure 8). 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, 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.

We remark 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 3 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 (Figure 9). 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 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.

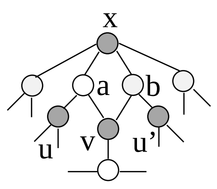

Figure 9: Vertex x is a degree 4 vertex that is a mirror of the degree three vertex v.

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 would have such a neighbor, then the tree separators rule fires 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 form which 7 external edges 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 is a small separator. So we end up removing 12 edges from which 5 external edges and 6 vertices which cannot create trees: $T(k-6)$. Finally, if u or u' is 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.

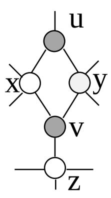

Figure 10: Vertex v, has a third degree 4 neighbor z.

Suppose that either u or v, say v, has a third degree 4 neighbor z (Figure 10). 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 behaviour 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 that 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 a graph of complexity $k-4$. In this case, x and y will be folded. Because they are not adjacent to other created degree 2 vertices (then there 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 the degree 0, 1 or 2 rules 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 worse 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 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 result in the removal of more edges. If x has three neighbors of degree 4, then at least 13 edges are removed from which 7 are external edges. 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 from which 4 external edges again leading to $T(k-7)$. Again, the existence of any separated tree implies more external edges that are removed and compensate for the tree. The proof of Lemma 1 is now completed. $\square$

Lemma 2 *If G has a vertex v of degree at least 5, then $T(k) \leq T(k-4) + T(k-7)$.*

Proof. Consider such a vertex v (Figure 11) and assume that one branches on v. If v is discarded, one vertex is removed and at least 5 edges are removed, giving $T(k-4)$. If v is taken in I, $N[v]$ 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 sums 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.

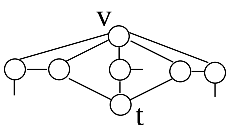

Figure 11: Vertices of degree at least 5.

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 $T(k) \leq T(k-4) + T(k-7)$. □

Lemma 3 *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 exists*[1]. *Here, we can apply items 1 and 3 from Lemma 1 to the branches denoted by T_1 and T_3, respectively.*

If there are no 3- or 4-cycles in G, that the algorithm can exploit, it is forced 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 3 shows that either such a branching results in the creation of local structures that, thereafter, allow the much more efficient branching described in Lemma 1, or there are more straightforward ways to perform even better branchings.

The proof of Lemma 3 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.

Proof. 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, and 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 the degree 1 and 2 rules can be applied after branching. Namely, if such a reduction rule fires in the branch where we discard v, then we have the sufficient relation of $T(k) \leq T(k-4) + T(k-5)$. 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 of $T(k) \leq T_3(k-2) + T(k-7)$.

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

[1]A sequence of consecutive branchings leading to a better recurrence, i.e., deriving a better time-bound.

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 $\{x, y, z\}$. By the same argument, there cannot exist adjacent vertices within $N(x)$, $N(y)$ or $N(z)$. There can, however, be 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. 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 there are two situations possible. Either, there are two pairs of these adjacencies between the same two neighborhoods, or all three neighborhoods are adjacent. In the first case, discarding v causes vertex folding to remove two additional edges since double edges are removed implicitly: this gives $T(k) \leq T(k-4) + T(k-5)$. In the second case, the neighborhoods of x, y and z are folded resulting in two degree 3 vertices between which a double edge is removed and a degree 4 vertex. Branching on this degree 4 vertex afterwards gives us $T(k) \leq T(k-5) + T_3(k-3-4)$. After filling in the worst case (case 3) of Lemma 1, we obtain two recurrence relations with a better branching behaviour than we need to prove.

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 1 and we are done. Consider the case in which no degree 4 vertices are 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 a different neighbor of v. We obtain the required branching behaviour 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 equals the dodecahedron. We finish 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 this 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 1. 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 1 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 1 on the graph of complexity $k-3$. This results in the even better behaviour 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 end points. 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 we have 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 (Figure 12).

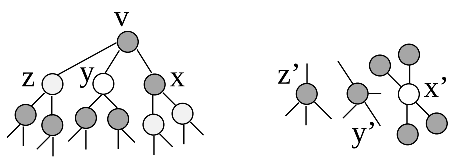

Figure 12: Picking at most one vertex from the three degree 4 vertices created by folding the neighbors of v.

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 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 external edges remain. 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 three 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, it 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 can form only one larger tree. If there is no size 1 tree, larger trees use more external edges and hence there can also be at most two of them.

In the above proof we have assumed that there are two additional adjacencies. There can, however, also be 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. Altogether, we have $T(k) \leq T_3(k-2-3) + T(k-2-9) + T_3(k-5) \leq 2T(k-8) + T(k-11) + 2T(k-12)$.

Three degree 4 vertices only two of which are adjacent

Identical to the previous case, we aim to perform 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 x' or y'.

If we discard x', we remove 4 edges and 1 vertex. Now, either a degree 4 vertex remains giving the $T_3(k-3)$, or an additional edge is removed by folding giving the in this case slightly better $T(k-4)$. If we take x' in I, we can also discard z' resulting in the removal of 17 edges from which 13 external edges 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 distance at least 3 away from each other in this structure. This can only be both z' and any neighbor of x'. Hence, there cannot be any size 1 tree: it would need two neighbors of x' which causes 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 be no 1-tree since it can be adjacent to at most one neighbor of x'. Larger trees remove enough external edges to prove $T(k-9)$.

Three degree 4 vertices on a path

Again, we combine applying Lemma 1 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 from which 11 external edges 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 a degree 4 vertex

We now have a graph of complexity $k-3$ with two degree 3 vertices, say y' and z', and a degree 4 vertex x'. Furthermore, y' and z' are adjacent but not adjacent to x'. Of these vertices x' cannot be involved in any 3- or 4-cycle, or we apply Lemma 1 case 2 as discussed with the general approach.

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 from which 12 external edges 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 be 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 exists). 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 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 complexity $k-3$ with two degree 3 vertices y', z' and a degree 4 vertex x' which are all the result of folding. Furthermore, y' is adjacent to x' and z' while x' and z' are non-adjacent. Of these vertices, x' cannot be involved in any 3- or 4-cycle since we then apply Lemma 1 case 2 as discussed with the general approach.

Similar to the previous case, we branch on x' giving $T(k-3-3)$ 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 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 x' being discarded. In total, this results in the removal of 11 edges and 4 vertices, and in this very specific structure no trees can exist: $T(k-3-6)$. We now 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 equals the dedocahedron. 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. Together this gives 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 end our proof with the last case involved. Without loss of generality let both a and c have neighbors in both other neighborhoods: let $N(a) = \{x, c, e\}$ 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 giving $T_3(k-5)$. The proof of Lemma 3 is now completed. □

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

Proof. Let a, b, 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, 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 are 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 that 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. □

Putting the four previous lemmata together, we have the following result.

Theorem 1 *There is an algorithm for the maximum independent set problem on connected graphs of average degree at most 3, running in time $O^*(1.0854^n)$.*

Proof. Considering all branchings from Lemmata 1 up to 2, we have $T(k) \leq T(k-8) + 2T(k-10) + T(k-12) + 2T(k-14)$ in the worst case. This recurrence relation is formed by combining Lemmata 3 and 1 and leads to a running time of $O^*(1.1781^k)$. On average degree three graphs this is $O^*(1.1781^{0.5n}) = O^*(1.0854^n)$. □

3 A general bottom-up method

In this section we present an original method that, given an algorithm working in graphs of average degree d, produces algorithms with non-trivial computation bounds for graphs of average degree greater than d. The method relies on the following two ideas.

First, in a graph whose average degree is bounded from below by some constant (possibly not an 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. For instance, trivially, if the graph has average degree greater than $d \in \mathbb{N}$, we know there exists a vertex v of degree $d+1$. If we assume that 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 5.

Then, for a given d, either 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 not and we can use an algorithm tailored for low-degree graphs. Thus, Lemma 5 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. The second idea is to analyze the running time of these algorithms thanks to a measure, allowing to fruitfully use the existence of the dense local configurations mentioned above. 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 it is by hypothesis valid in graphs with average degree d. The recurrences given by the branching will give 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 5 *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 1 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):

$$\sum_{k \leq d} k\alpha_k \leq \delta \qquad (1)$$

$$\sum_{k \leq d} \alpha_k = d \qquad (2)$$

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 $n_k, \forall k < d$. 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:

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

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 remaining of the graph, and thanks to inequalities (1) and (2), we get:

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

For $0 \leq j \leq i-2$, we now consider the following programs $(P_{i,j})$: $\max(\epsilon)$ under constraints (1),(2),(3), (4) and $\mu_2 \leq f_{i,j} - 1$. In other words, we look for the maximal value for ϵ such that it is possible that any vertex in the graph verifies $\mu_2 \leq f_{i,j} - 1$. The result is $\epsilon_{i,j}$. 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 5 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($d_{i,j}$) computes the usual preprocessing and branch 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($d_{i,j-1}$)[2].

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

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

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

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 5 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 $T(m,n) \leq T(m - \mu_1, n - \nu_1) + T(m - \mu_2, n - \nu_2) + p(m,n)$ (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 proved in Section 2).

For completeness, we need to pay attention to reduction rules and creation of trees when branching. Note that an additional reduction of ν vertices and $\mu \geq \nu$ edges is not problematic for $T(m,n) \geq T(m - \nu, n - \nu)$, i.e., $y^{d_{i,j_1} - 2} \leq \gamma_{i,j-1}$, which has to be verified when computing y. In order to deal with trees, note also that removing a tree corresponds to a reduction of ν vertices and $\nu - 1$ edges. This is not problematic as soon as $y^{d_{i,j-1} - 2 + 2/\nu} \leq \gamma_{i,j-1}$. Of course, it depends on the value we fix for low-degree algorithms. Then, we will always pay attention to the fact that these inequalities are fulfilled when computing y. □

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

- An algorithm working in time $O^*(1.1707^n)$ for graphs of average degree 4, slightly outperforming the best known bound of $O^*(1.1713^n)$ of [1]. It is directly obtained using Proposition 1 and our previous algorithm of Section 2 that works in graphs in time $O^*(1.1781^{m-n})$. Indeed:

[2]If $j = 0$ of course we 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.

  - 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 5, 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[3], the recurrence shows that the running time is $O^*(1.1396^n)$ in graphs of average degree $d_{4,2} = 3 + 3/5$.
  - Similarly, if the average degree is greater than $d_{4,2}$, we reduce the graph either by 1 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 working 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 2 (Section 4), that outperforms the best known bound of [5]. Thanks to Lemma 5, we know that we can reduce (at least) the graph either by 1 vertex and 5 edges, or by 6 vertices and respectively 15, 16, 17 and 18 edges in graphs of average degree greater than 4, $4 + 4/9$, $4 + 4/7$ and $4 + 4/5$. We get successively algorithms working in time $O^*(1.1786^n)$, $O^*(1.1840^n)$, $O^*(1.1929^n)$ and $O^*(1.2000^n)$ in graphs of average degree $4 + 4/9$, $4 + 4/7$, $4 + 4/5$ and 5.

- An algorithm working in time $O^*(1.2114^n)$ for graphs of average degrees 6 based upon the algorithm in $O^*(1.1918^n)$ for graphs of average degree 5 of Theorem 3 (Section 5), that outperforms the best known bound of [5]. Thanks to Lemma 5, we know that we can reduce (at least) the graph either by 1 vertex and 6 edges, or by 7 vertices and respectively 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 $5 + 5/7$. We get successively algorithms working in time $O^*(1.1968^n)$, $O^*(1.2018^n)$, $O^*(1.2035^n)$ $O^*(1.2066^n)$ and $O^*(1.2113^n)$ in graphs of average degree $5 + 5/23$, $5 + 5/11$, $5 + 20/37$, $5 + 5/7$ and 6.

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 techniques, but already outperform the best known bounds so far, leading us thinking that the method presented has a certain interest.

4 Refined case analysis for graphs of average degree 4

In Lemma 5 we have shown the existence of local dense configurations when the graph has average degree more than 3. For instance, we have seen that 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, in order to prove the following theorem.

Theorem 2 *It is possible to compute a solution to* MAX INDEPENDENT SET *on graphs with maximum (or even average) degree* 4 *with running time* $O^*(1.1571^n)$.

Proof. Based upon the discussion in Section 3, we seek a complexity of the form $O^*(\gamma^n y^{2m-3n})$, where $\gamma = 1.0854$, 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.

[3]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 $O^*\left(1.1243^n y^{((3+3/5)-(3+3/7))n}\right) = O^*\left(1.1396^n\right)$ in graphs of average degree $3 + 3/5$.

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 $y = 1.0596$ (or $\nu_1 = 4$, $\mu_1 = 9$, $\nu_2 = 2$, $\mu_2 = 8$, which is even better), see Lemma 2. In the latter case, Lemma 1 gives a set of possible reductions that can be plugged into the previous equation to compute y, but we can do much better now, thanks to our complexity measure, using the fact that, informally, removing a lot of vertices might be also good.

In what follows, we consider that the graph has maximum degree 4, and we denote 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 $y = 1.0658$.

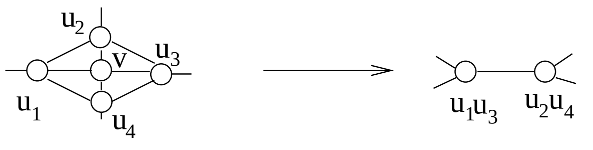

Figure 13: $G[N(v)]$ is a 4-cycle

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 (Figure 13). 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 $y = 1.0420$.

Case 3: u_1 has degree 3 and only one outer edge

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 Figure 14).

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 (Figure 14a). If $|N(y) \cap N(z)| = 1$, then their common neighbor has degree 4 (Figure 14b). 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 Figure 14c.

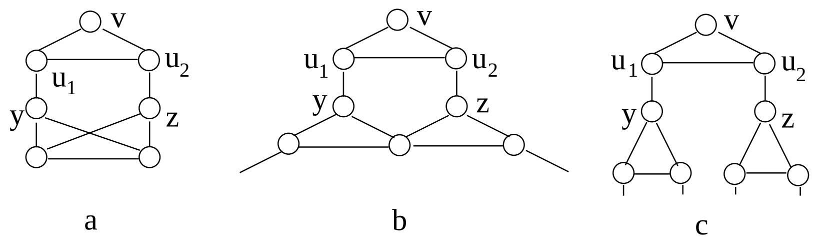

Figure 14: Case 3.

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 $\mu_2 \geq 13$.

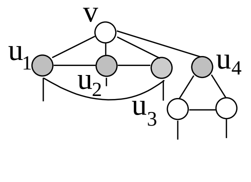

Figure 15: Vertices v, u_1, u_2, u_3 form a 4-clique

Otherwise, there is an edge between u_2 and u_3. Then, v, u_1, u_2, u_3 form a 4-clique (Figure 15).

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 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 the proof of the theorem, 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 will show in what follows that it is never problematic to create a tree while branching in the analysis in Theorem 2. We have shown in Section 2.1.2 that we can introduce a reduction rule that prevents the creation of trees when branching on a vertex of degree 3. Thus, 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. We 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 $N(v)$).

First, suppose 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 3 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 2 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 remainder 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 are 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 $2l$ 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 $y = 1.0503$).

Now, consider 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 $y = 1.0638$.

- 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, 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 consider now 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 we do not need to branch. Indeed, the tree has only two vertices t_1, t_2 that are of degree 3 (otherwise there would be 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 have to 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, u_3 and t_2 is adjacent to u_2, 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 2 vertices (v and t_1), or we take 3 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 $m_i \geq n_i$ have occurred. In order to simplify our notation we denote by $\bar{T}$ the complexity of this case. Running time now verifies:

$$\begin{aligned} T(m,n) &\leq \sum_i \bar{T}(m_i,n_i) = \sum_i \bar{T}(m - \sum_{j\neq i} m_j, n - \sum_{j\neq i} n_j) \\ &\leq \sum_i \bar{T}(m - \sum_{j\neq i}(m_j - n_j), n)c^{n-n_i} \leq \bar{T}(m,n)c^n \sum_i \frac{1}{c^{n_i}} \end{aligned}$$

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 2 is now completed. □

5 Final improvements and algorithm in general graphs

In this section we devise algorithms for graphs of maximum degree 5, 6 and general graphs. The algorithms follow the same line as the one devised in [5]. There, a branching is performed on a vertex of maximum degree, 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 two ways in order to improve the running time for MAX INDEPENDENT SET. First, we incorporate the fact that we have 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, modifying the set of constraints in the measure and conquer analysis (see Proposition 2) and leading to a better running time.

Then we improve the analysis of measure and conquer in graphs of maximum degree 5. Since vertex folding may produce vertices of degree higher than 5, it is not possible to restrict the study to vertices of degree at most 5. However, any vertex of degree at least 6 has been produced by a vertex folding. Depending on the weights, a vertex folding may decrease the total weight of the graph, fact that was not taken into account so far. In Proposition 3 we manage to include this decrease in the branching analysis in order to improve the time bound for MAX INDEPENDENT SET in graphs of degree 5. As a final result, we combine the two previous ideas to get a general algorithm in $O^*(1.2125^n)$ in Theorem 4.

Proposition 2 *It is possible to optimally solve* MAX INDEPENDENT SET *in* $O^*(1.2135^n)$.

Proof. According to former sections, we know that it is possible to compute MAX INDEPENDENT SET on graphs of maximum degree Δ with running time bounded above by $O^*(\gamma_\Delta^n)$, where $\gamma_3 = 1.0854$ and $\gamma_4 = 1.1571$.

The algorithm we propose is the same as in [5] 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 $\Delta \geq 5$ do
   (a) if there are several connected components, solve the problem separately on each connected component.
   (b) If a vertex v is dominated, remove it.

(c) If there is a vertex of degree 2, fold it.

(d) Otherwise select a vertex v of degree Δ which minimizes the number of inner edges (edges with both endpoints in $N(v)$) and branch on v (either take it or not).

2. If $\Delta \leq 2$ solve the problem in polynomial time.

3. If $\Delta = 3$ use algorithm of Section 2.

4. If $\Delta = 4$ use algorithm of Section 4.

For the running time analysis we use measure and conquer techniques as in [5]. 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 $d \geq 7$. 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 is $O^*(\gamma^n)$. Weights are optimized in order to find the best γ. For the analysis to be valid, weights must fulfill some constraints, such as the fact that reduction rules must not increase the total weight of the graph. As shown in [5] 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 $\{2, 3, \cdots, 8\}$ (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 2 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 for the analysis to be valid the weights have to satisfy:

$$\begin{aligned} & \gamma_3^n \left(\frac{\gamma_4}{\gamma_3}\right)^{(d-3)n} \leq \gamma^{w_3 n_3 + w_4 n_4} \\ \Leftrightarrow \quad & \log \gamma_3 + (d-3)(\log \gamma_4 - \log \gamma_3) \leq \log \gamma((4-d)w_3 + (d-3)w_4) \end{aligned}$$

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 [5]. 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 1d leads to decrease 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 worse one gives the running time $O^*(\gamma^W)$. Here, we do not need to consider the cases of branching on degree 3 and 4 anymore, since it is handled by our low degree algorithms. Hence, we only have the recurrence relation for $\Delta \geq 5$, allowing to choose more efficient weights leading to a better running time.

The best values we have found are $w_3 = 0.493$, $w_4 = 0.765$, $w_5 = 0.914$, $w_6 = 0.9777$, satisfying all the constraints and leading to $\gamma = 1.2135$. $\square$

Corollary 1 *On graphs of maximum degree* 6*,* MAX INDEPENDENT SET *can be computed with running time* $O^*(1.2083^n)$*. On graphs of maximum degree* 5*,* MAX INDEPENDENT SET *can be computed with running time* $O^*(1.1935^n)$*.*

Both these results are consequences of Proposition 2. Remark now that, for $v \in V$, $w(v) \leq w_\Delta n$. So, for the cases dealt with 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. Moreover, as mentioned before, it is possible to improve them slightly.

Theorem 3 *On graphs of maximum degree* 5*,* MAX INDEPENDENT SET *can be computed with running time* $O^*(1.1918^n)$*.*

Proof. In Theorem 3 we deal with graphs of graphs of maximum degree 5. We use the algorithm described in Proposition 2, but we perform a better analysis. As mentioned in Section 5, vertex folding may create vertices of degree higher than 5 so we have to take into account branching on vertices of degree higher than 5.

We define weights w_d for vertices of degree d, with the same set of constraints up to the following modification: we fix $w_5 = 1$ and do not impose anymore $w_d \in [0,1]$ for $d \geq 6$. Indeed, if the recurrence relations give a running time in $O^*(\gamma^W)$, then, since initially the graph has maximum degree 5 and $W = \sum_{v\in V} w(v) \leq n$, the running time is $O^*(\gamma^n)$. However, we cannot choose any value for w_d, we have to verify that a folding does not increase the total weight of the graph. Let v be a vertex of degree d higher than 5. It has been created by folding one or several vertices of degree 2. Let $(v_j)_{j\in J}$ be the set of vertices (of degree at most 5) that have been merged together (by the foldings) to create vertex v. When a folding is executed, two edges are deleted (together with the vertex of degree 2), thus $\sum_{j\in J}(d(v_j) - 2) = d$. Then, if we denote M_d the minimal decreasing obtained by foldings to create a vertex of degree d (formally $M_d = \min_{K:\sum_{k\in K}(k-2)=d; 3\leq k\leq\Delta} \sum_{k\in K} w_k$), we add the constraints $w_d \leq M_d$ for all $d > \Delta$.

This additional constraint ensures that the total weight of the graph does not increase while doing foldings. Now, we perform the same analysis as before. The fact that we can choose $w_d > 1$ for $d \geq 6$ leads to a better running time. This is not surprising since informally this is a way to take into account the progress of the algorithm made during a vertex folding.

If we use weights $w_3^5 = 0.5218$, $w_4^5 = 0.8317$, $w_5^5 = 1$ and $w_d^5 = M_d$, $d > 5$, all the constraints are satisfied and we get $\gamma_5 = 1.1918$. □

All the ingredients are now here and allow us to conclude the paper by giving the final theorem that improves the best published exact complexity result for MAX INDEPENDENT SET [5].

Theorem 4 *It is possible to compute* MAX INDEPENDENT SET *with running time* $O^*(1.2125^n)$*.*

Proof. We use the following straightforward modification of the algorithm in Proposition 2:

1. While $\Delta \geq 6$ do
   (a) if there are several connected components, solve the problem separately on each connected component.
   (b) If a vertex v is dominated, remove it.
   (c) If there is a vertex of degree 2, fold it.
   (d) Otherwise select a vertex v of degree Δ which minimizes the number of inner edges (edges with both endpoints in $N(v)$) and branch on v (either take it or not).
2. If $\Delta \leq 2$ solve the problem in polynomial time.
3. Otherwise use algorithm of Theorem 3.

The analysis of the algorithm is performed similarly as in the proof of Proposition 2. Now, we know that it is possible to compute MAX INDEPENDENT SET on graphs of maximum degree Δ with running time bounded above by $O^*(\gamma_\Delta^n)$, where $\gamma_3 = 1.0854$, $\gamma_4 = 1.1571$ and $\gamma_5 = 1.1918$.

Weights are non decreasing, between 0 and 1, and with $w_d = 0$ for $d \geq 2$, $w_d = 1$ for $d \geq 7$. Constraints (c1) and (c2) are exactly the same. Now we have to ensure that the running time $O^*(\gamma^W)$ is valid for graphs of maximum degree at most 5. Let n_3, n_4 and n_5 the number of vertices of degree 3, 4 and 5 respectively ($n_3 + n_4 + n_5 = n$ thanks to the vertex folding). If the algorithm in Theorem 3 has running time $O^*(\gamma_5^{w_3^5 n_3 + w_4^5 n_4 + n_5})$, then we have to introduce constraints that enforce:

$$\gamma_5^{w_3^5 n_3 + w_4^5 n_4 + n_5} \leq \gamma^{w_3 n_3 + w_4 n_4 + w_5 n_5}$$

This is a consequence of the following constraints:

(c'3) $w_3 \geq w_3^5 \frac{\log \gamma_5}{\log \gamma}$

(c'4) $w_4 \geq w_4^5 \frac{\log \gamma_5}{\log \gamma}$

(c'5) $w_5 \geq w_5^5 \frac{\log \gamma_5}{\log \gamma}$.

Now, we only have to consider branching of degree at least 6 in our recurrence relation leading to γ. The weights $w_3 = 0.4752$, $w_4 = 0.756$, $w_5 = 0.91$, $w_6 = 0.9763$ satisfy all the constraints and lead to $\gamma = 1.2125$. □

References

[1] R. Beigel. Finding maximum independent sets in sparse and general graphs. In *Proc. Symposium on Discrete Algorithms, SODA'99*, pages 856–857, 1999.

[2] N. Bourgeois, B. Escoffier, and V. Th. Paschos. An $O^*(1.0977^n)$ exact algorithm for MAX INDEPENDENT SET in sparse graphs. In M. Grohe and R. Niedermeier, editors, *Proc. International Workshop on Exact and Parameterized Computation, IWPEC'08*, volume 5018 of *Lecture Notes in Computer Science*, pages 55–65. Springer-Verlag, 2008.

[3] J. Chen, I. A. Kanj, and G. Xia. Labeled search trees and amortized analysis: improved upper bounds for NP-hard problems. In T. Ibaraki, N. Katoh, and H. Ono, editors, *Proc. International Symposium on Algorithms and Computation, ISAAC'03*, volume 2906 of *Lecture Notes in Computer Science*, pages 148–157. Springer-Verlag, 2003.

[4] D. Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *Proc. Symposium on Discrete Algorithms, SODA'01*, pages 329–337, 2001.

[5] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: a simple $O(2^{0.288n})$ independent set algorithm. In *Proc. Symposium on Discrete Algorithms, SODA'06*, pages 18–25, 2006.

[6] M. Fürer. A faster algorithm for finding maximum independent sets in sparse graphs. In J. R. Corea, A. Hevia, and M. Kiwi, editors, *Proc. Latin American Symposium on Theoretical Informatics, LATIN'06*, volume 3887 of *Lecture Notes in Computer Science*, pages 491–501. Springer-Verlag, 2006.

[7] I. Razgon. A faster solving of the maximum independent set problem for graphs with maximal degree 3. In *Proc. Algorithms and Complexity in Durham, ACiD'06*, pages 131–142, 2006.

[8] I. Razgon. Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. *J. Discrete Algorithms*, 7:191–212, 2009.

[9] J. M. Robson. Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université de Bordeaux I, 2001.

[10] G. J. Woeginger. Exact algorithms for NP-hard problems: a survey. In M. Juenger, G. Reinelt, and G. Rinaldi, editors, *Combinatorial Optimization - Eureka! You shrink!*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–207. Springer-Verlag, 2003.