A Bottom-Up Method and Fast Algorithms for MAX INDEPENDENT SET*

Nicolas Bourgeois[1], Bruno Escoffier[1],
Vangelis Th. Paschos[1], and Johan M.M. van Rooij[2]

[1] LAMSADE, CNRS FRE 3234 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

Abstract. We first propose a new method, called "bottom-up method", that, informally, "propagates" improvement of the worst-case complexity for "sparse" instances to "denser" ones and we show an easy though non-trivial application of it to the MIN SET COVER problem. We then tackle MAX INDEPENDENT SET. Following the bottom-up method we propagate improvements of worst-case complexity from graphs of average degree d to graphs of average degree greater than d. Indeed, using algorithms for MAX INDEPENDENT SET in graphs of average degree 3, we tackle MAX INDEPENDENT SET in graphs of average degree 4, 5 and 6. Then, we combine the bottom-up technique with measure and conquer techniques to get improved running times for graphs of maximum degree 4, 5 and 6 but also for general graphs. The best computation bounds obtained for MAX INDEPENDENT SET are $O^*(1.1571^n)$, $O^*(1.1918^n)$ and $O^*(1.2071^n)$, for graphs of maximum (or more generally average) degree 4, 5 and 6 respectively, and $O^*(1.2127^n)$ for general graphs. These results improve upon the best known polynomial space results for these cases.

Keywords: Bottom-Up Method, Max Independent Set, Exact Algorithms.

1 Introduction

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

Here, we propose in Section 2 a generic method that propagates improvements of worst-case complexity from "sparse" instances to "denser" (less sparse) ones, where the density of an instance is proper to the problem handled and refers

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

H. Kaplan (Ed.): SWAT 2010, LNCS 6139, pp. 62–73, 2010.

to the average value of some parameter of its instance. We call this method "bottom-up method". The basic idea here has two ingredients: (i) the choice of the recursive measure of the instance and (ii) a way to ensure that on "denser" instances, a good branching (wrt. the chosen measure) occurs.

We then illustrate our method to MIN SET COVER. Given a finite ground set $\mathcal{U}$ and a set-system $\mathcal{S}$ over $\mathcal{U}$, MIN SET COVER consists of determining a minimum-size subsystem $\mathcal{S}'$ covering $\mathcal{U}$. Here, the density of an instance is defined to be the average cardinality of the sets in the set-system $\mathcal{S}$. Application of the method to MIN SET COVER is rather direct but it produces quite interesting results. As we show in Section 2 it outperforms the results of [9] in instances with average-set sizes 6, 7, 8, ... Note that we are not aware of results better than those given here.

We next handle the MAX INDEPENDENT SET problem. Given a graph $G = (V, E)$, MAX INDEPENDENT SET consists of finding a maximum-size subset $V' \subseteq V$ such that for any $(v_i, v_j) \in V' \times V'$, $(v_i, v_j) \notin E$. For this problem, [4] proposes an algorithm with worst-case complexity bound $O^*(1.2201^n)$[1]. All the results we present here are polynomial space algorithms. We also quote the $O(1.2108^n)$ time bound in [7] using exponential space (claimed to be improved down to $O(1.1889^n)$ in the technical report [8], still using exponential space). 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 recent $O^*(1.0892^n)$ time algorithm in [6], and the $O^*(1.0854^n)$ time algorithm by the authors of the article at hand [3]. For MAX INDEPENDENT SET density of a graph is measured by its average degree. So, the bottom-up method here extends improvements of the worst-case complexity in graphs of average degree d to graphs of average degree greater than d.

In order to informally sketch our bottom-up method in the case of MAX INDEPENDENT SET, 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 2). In other words, the form of the running time we seek is parameterized by what we already know on graphs with smaller average degrees. Next, according to this form, we identify particular values d_i (not necessarily integer) on 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 time

[1] In a very recent article [5] a $O^*(1.2132^n)$ time algorithm for MAX INDEPENDENT SET is proposed. Plugging this new result allows to further improve our results. We give the corresponding bounds at the end of Section 4.

bound of $O^*(1.1713^n)$ of [1] (Section 2). This result is further improved down to $O^*(1.1571^n)$ (Section 3) with a more refined case analysis. We also provide bounds for (connected) graphs with average degree at most 5 and 6.

In section 4, we combine measure and conquer with bottom-up to show that MAX INDEPENDENT SET can be optimally solved in time $O^*(1.2127^n)$ in general graphs, thus improving the $O^*(1.2201^n)$ bound [4]. Furthermore, in graphs of maximum degree at most 5 and 6, we provide time bounds of $O^*(1.1918^n)$ and $O^*(1.2071^n)$, respectively, that improve upon the respective $O^*(1.2023^n)$ and $O^*(1.2172^n)$ time bounds of [4][2].

We give the results obtained using the $O^*(1.0854^n)$ time bound of [3] for graphs of degree 3. Note that using previously known bounds for solving MAX INDEPENDENT SET in graphs of degree 3 (worse than $O^*(1.0854^n)$), the bottom-up method would also lead to improved results (with respect to those known in the literature). We illustrate this point in Table 1.

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

Degree 3	Degree 4	Degree 5	Degree 6	General graphs
1.08537	1.1571	1.1918	1.2071	1.2127
1.0892 [6]	1.1594	1.1932	1.2082	1.2135
1.0977 [2]	1.1655	1.198	1.213	1.217

Maybe more interesting than the improvements themselves is the fact that they are obtained via an original method that, once some, possibly long, case analysis has been performed on instances of small density, it is directly applicable for getting results higher density instances, even for general ones. We think that this method deserves further attention and insight, since it might be used to solve also other problems.

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 The Bottom-Up Method

In this section we present the method that relies on the following two stepping stones: (i) the choice of the recursive complexity measure, applied in a very simple way in Section 2.1 to the MIN SET COVER to get non trivial time bounds for instances of bounded average set-cardinalities, and (ii) a way to ensure that on instances of density greater that d, a good branching (wrt. the chosen complexity measure) occurs. This second point is illustrated in Section 2.2 for MAX INDEPENDENT SET.

[2] The bound in graphs of degree at most d is obtained as $O^*(1.2201^{nw_d})$, where w_d is the weight associated to vertices of degree d in the measure and conquer analysis in [4]; better bounds could maybe be achieved with this method, but this is not straightforward, since reduction rules may create vertices of arbitrarily large degree.

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

Let us consider the MIN SET COVER problem with ground set $\mathcal{U} = \{x_1, \cdots, x_n\}$ and set system $\mathcal{S} = \{S_1, \cdots, S_m\}$. We show that the bottom-up method easily applies to get algorithm with improved time bounds for instance with sets of average (or maximum) size d, for any $d \geq 5$. In what follows $p = \sum_{i=1}^{m} |S_i|$.

Lemma 1. *The algorithm in [9] solves* MIN SET COVER *in time resp.* $O^*(1.55^m)$, $O^*(1.63^m)$, $O^*(1.71^m)$, $O^*(1.79^m)$ *in instances with sets of average size 5, 6, 7 and 8.*

Proof. Denoting by n_i the number of sets of size i and m_j the number of elements of frequency j, [9] gives an algorithm working in time $O^*(1.2302^{k(I)})$ where $k(I) = \sum_{i\geq 1} w_i n_i + \sum_{j\geq 1} v_j m_j$. Here w_i is the weight associated to a set of size i, and v_j is the weight associated to an element of frequency j. It is easy to see that, by convexity[3], if the average size of sets is an integer d, then $\sum_{i\geq 1} w_i n_i \leq mw_d$. Moreover, note that $v_j/j \leq v_3/3$ for all $j \geq 1$[4], hence $\sum_{j\geq 1} v_j m_j \leq v_3/3 \sum_{j\geq 1} j m_j = dmv_3/3$. We get $k(I) \leq m(w_d + v_3 d/3)$ (this bound being tight if all sets have size d and all elements have frequency 3). □

Let us consider an instance with $p > dm$. The bottom-up method assumes that we know how to solve the problem in instances with sets of average size d in time $O^*(\gamma_d^m)$. It seeks a complexity of the form $O^*(\gamma_d^m y^{p-dm})$; indeed, it is valid by hypothesis for $p = dm$, ie. on instances with sets of average size d. Let us consider a set S of size $s \geq d+1$. We branch on S. If we do not take it, we remove one set and $s \geq d+1$ edges; if we take it, suppose that each element in S has frequency at least 3. Then we remove 1 set and (at least) $3s \geq 3(d+1)$ edges. Hence, for the complexity to be valid we have to choose y such that $\gamma_d^m y^{p-dm} \geq \gamma_d^{m-1} y^{p-(d+1)-d(m-1)} + \gamma_d^{m-1} y^{p-3(d+1)-d(m-1)}$, or equivalently $1 \geq \gamma_d^{-1} y^{-1} + \gamma_d^{-1} y^{-(2d+3)}$.

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

- If there is an element j of frequency 1 in S, we have to take S and we remove one set and $s \geq d+1$ edges. This does not create any problem as long as $\gamma_d^{m-1} y^{(p-d-1)-d(m-1)} \leq \gamma_d^m y^{p-dm}$, i.e., $y \leq \gamma_d$.
- Otherwise, if there is an element j of frequency 2 which is in S and S', then either we take S and remove 1 set and at least $2(d+1)$ edges, or we remove S and take S', and we remove 2 sets and at least $d+2$ edges. So we have to check that the value of y computed in the general case verifies $1 \geq \gamma_d^{-1} y^{-(d+2)} + \gamma_d^{-2} y^{-2+d}$.

Then if sets have average size $d+1$, since $p = (d+1)m$, the complexity is $O^*(\gamma_{d+1}^m)$ with $\gamma_{d+1} = \gamma_d \times y$. Starting from $\gamma_5 = 1.55$, the recurrences give $\gamma_6 = 1.61$, $\gamma_7 = 1.66$ (with $y = 1.031$) and $\gamma_8 = 1.70$ (with $y = 1.024$).

[3] w_i's are (0,0.3755,0.7509,0.9058,0.9720,0.9982) for $i = 1, \cdots, 6$ and 1 for $i \geq 7$.

[4] v_j's are (0, 0.2195, 0.6714, 0.8766, 0.9569, 0.9882) for $i = 1, \cdots, 6$ and 1 for $i \geq 7$.

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

Interestingly enough, the time bound of $O^*(1.2302^{m(w_d+v_3 d/3)})$ obtained in Lemma 1 is bigger than 2^m for $d \geq 11$ while using the previous method, it can be shown that $\gamma_d < 2$ for any d (for instance $\gamma_{100} < 1.98$). Of course, the analysis conducted in [9] is not oriented towards the bounded size case, so better results might be obtained; we will say a few fords in conclusion on the links between this complexity measure and the one of measure and conquer.

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

In order to use efficiently the previous complexity measure for MAX INDEPENDENT SET, we prove that, 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. Let us give an example. In the independent set problem, there are two well known reductions rules that allow to remove without branching vertices of degree at most 2[5]. In the following we will assume that these reductions rules has been performed, i.e., the graph does not contain any vertex of degree at most 2. Then, trivially, if the graph has average degree greater than $d \in \mathbb{N}$, we know there exists a vertex v of degree at least $d+1$. If we assume that no vertex is dominated[6], then there exist at least $f(d) = 2(d+1) + \lceil (d+1)/2 \rceil$ edges incident to some neighbor of v. Indeed, there exist $d+1$ edges incident to v, $d+1$ edges between a neighbor of v and a vertex not neighbor of v (one for each neighbor of v, to avoid domination) and, since each vertex has degree at least 3, at least $\lceil (3(d+1) - 2(d+1))/2 \rceil = \lceil (d+1)/2 \rceil$ other edges. Note that such relationships may be established even if d is non integer. For instance, we will see that if $d > 24/7$, then there exists a vertex of degree 5 or two adjacent vertices having degree 4, leading to $f(d) = 11$. This property linking the average degree to the quality of the branching is given in Lemma 2.

Then, for a given d, either the average degree is greater than d, and we can make an efficient branching (i.e., a branching that induces a recurrence relation leading to a lower time-bound), or it is not and we can use an algorithm tailored for low-degree graphs. Thus, Lemma 2 fixes a set of critical degrees (d_i) and we define step-by-step (from the smallest to the highest) algorithms $\texttt{STABLE}(d_i)$, that work on graphs of average degree d_i or less. With this lemma, we analyse the running time of these algorithms thanks to a measure allowing to fruitfully use the existence of the dense local configurations mentioned above. As for MIN

[5] A vertex of degree at most 1 should be taken in the solution. If a vertex v has two neigbors u_1 and u_2, take v if u_1 and u_2 are adjacent, otherwise remove v, u_1, u_2 from the graph and add a new vertex u_{12} whose neighborhood is $N(u_1) \cup N(u_2) \setminus \{v\}$ (this reduction is called vertex folding, see for instance [4]).

[6] u dominates v if $N[u] \subseteq N[v]$. In this case, it is never interesting to take v.

SET COVER, if we know how to solve the problem in time $O^*(\gamma_d^n)$ 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 $O^*(\gamma_d^n y^{2m-dn})$. 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 2. *There exists a specific sequence $(\epsilon_{i,j}, f_{i,j})_{i\geq 4, j\leq i-2}$ such that, if the input graph has average degree more than $i-1+\epsilon_{i,j}$, then the following branching is possible: either remove 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 (details are omitted):

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

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

Fix now $\epsilon = 2m/n - (d-1) \in]0,1[$. Then, $\epsilon = \frac{\sum_{k\leq d}(k+1-d)n_k}{\sum_{k\leq d} n_k}$. This function is decreasing with $n_k, \forall k < d$. 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:

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

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

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

For $0 \leq j \leq i-2$, we 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 $\epsilon_{i,j}$ for ϵ such that it is possible that any vertex in the graph verifies $\mu_2 \leq f_{i,j} - 1$. Equivalently, if the graph has degree higher than $i - 1 + \epsilon_{i,j}$, we remove at least $f_{i,j}$ edges (when taking some well chosen vertex). □

Let $d_{i,j} = i - 1 + \epsilon_{i,j}$. Now we use Lemma 2 to recursively define an algorithm $\texttt{STABLE}(d_{i,j})$ solving MAX INDEPENDENT SET in a graph of average degree at most $d_{i,j}$. $\texttt{STABLE}(d_{i,j})$ performs the usual preprocessing (described above at the beginning of the section) and branches on a vertex that maximizes the number of edges incident to its neighborhood. It repeats this step until the average degree is at most $d_{i,j-1}$, then it applies algorithm $\texttt{STABLE}(d_{i,j-1})$[7].

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

Proposition 1. *$\texttt{STABLE}(d_{i,j})$ runs in time $T(m,n) = O^*\left(\gamma^n_{i,j-1} y_{i,j}^{2m - d_{i,j-1}n}\right)$, where $y_{i,j}$ is the smallest solution of the inequality:*

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

In particular, $T(m,n) = O^*(\gamma^n_{i,j})$ 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 2 we branch on a vertex where we remove either ν_1 vertices and μ_1 edges or ν_2 vertices and (at least) μ_2 edges. Then the running time is valid as long as y is such that $T(m,n) \geq T(m-\mu_1, n-\nu_1) + T(m-\mu_2, n-\nu_2) + p(m,n)$ (for some polynomial p). This gives the recurrence relation claimed by proposition's statement.

Since $2m \leq d_{i,j}n$ in a graph of average degree at most $d_{i,j}$, the running time $O^*(y^n_{i,j})$ 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 in [3]).

For completeness, we need to pay attention to the fact that, once the branching has been performed, reduction rules might be applied in order to remove vertices of degree at most 2 in the remaining graph. For instance, if a separated tree on ν vertices is created, reduction rules will remove this tree hence, in all, ν vertices

[7] If $j = 0$ of course we use $\texttt{STABLE}(d_{i-1,i-3})$ in graphs of average degree at most $d_{i-1,i-3}$ and the same result as in Proposition 1 holds.

and $\nu - 1$ edges. Note that all reduction rules remove $\nu \geq 1$ vertices and at least $\nu - 1$ edges. We have to check that these operations do not increase $T(m, n)$, i.e., $T(m, n) \geq T(m - \nu + 1, n - \nu)$, or $y^{d_{i,j-1}-2+2/\nu} \leq \gamma_{i,j-1}$. □

To conclude this section, let us note that as direct applications of Proposition 1 we obtain (details are omitted) an algorithm running in time $O^*(1.1707^n)$ for graphs of average degree 4 (slightly outperforming the bound of $O^*(1.1713^n)$ by [1]), and algorithms running in times $O^*(1.2000^n)$ and $O^*(1.2114^n)$ for graphs of average degrees 5 and 6, respectively (based upon the algorithm in time $O^*(1.1571^n)$ for graphs of average degree 4 of Theorem 1). 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.

3 Refined Case Analysis for Graphs of Average Degree 4

In Lemma 2 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 1. *It is possible to solve* MAX INDEPENDENT SET *on graphs with maximum (or even average) degree* 4 *with running time* $O^*(1.1571^n)$.

Proof (Sketch). Based upon what has been discussed in Section 2, 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.

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

In what follows, we consider the latter case, *i.e.*, 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 4 cases, depending on the configuration of $N(v)$. Here, we consider that no trees are created while branching (the case of trees is not detailed here due to lack of space).

Case 1. All the neighbors of v have degree 4.

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

But there is only one possibility with no domination and only 12 edges incident to vertices in $N(v)$: when u_1, u_2, u_3, u_4 is a 4-cycle. In this case, we can reduce the graph before branching. Any optimal solution cannot contain more than two vertices from the cycle. If it contains only one vertex, then replacing it by v does not change its size. Finally, there exist only three disjoint possibilities: keep u_1 and u_3, keep u_2 and u_4 or keep only v. 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 is 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 are both part of a triangle.

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. If $|N(y) \cap N(z)| = 1$, then their common neighbor has degree 4. 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 $y = 1.0581$.

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

Case 4. u_1 has degree 4 and only one outer edge.

Since Case 1 does not occur, we can assume that there is a vertex (say u_4) of degree 3. Since Case 3 does not occur, u_4 has no inner edge. Hence, u_1 is adjacent to both u_2 and u_3. Then, there are only two possibilities. If there are no other inner edges, since Case 3 does not occur, u_2 and u_3 have two outer edges, and we have in all 13 edges. This gives once again $\nu_1 = 1$, $\mu_1 = 4$, $\nu_2 = 5$ and $\mu_2 \geq 13$. Otherwise, there is an edge between u_2 and u_3. Then, v, u_1, u_2, u_3 form a 4-clique. We branch on u_4. If we take u_4, we delete $\nu_1 = 4$ vertices and (at least) $\mu_1 = 9$ 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.

The remaining part of the proof (not given here) has to deal with the case when some trees are created while branching and to verify that performing a reduction rule (such as a vertex folding) does not increase the measure. □

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

In this section we devise algorithms for graphs of maximum degree 5, 6 and general graphs. The algorithms follow the same line as the one devised in [4]. 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, by taking into account that creating a vertex of high degree by vertex folding may decrease the total weight. As a final result, we combine the two previous ideas to get a general algorithm in time $O^*(1.2127^n)$ in Theorem 3.

Proposition 2. MAX INDEPENDENT SET *can be solved in time* $O^*(1.2135^n)$.

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 $\gamma_4 = 1.1571$.

Our algorithm works as follows: while $\Delta \geq 5$, run the preprocessing (in particular, fold appropriate vertices and reduce mirrors) described in [4] and branch on any vertex of maximum degree. Once $\Delta \leq 4$, run the algorithms described in [3] (case of degree 3) and in Section 3 (case of degree 4).

We analyse the running time of this algorithm with the same measure and conquer techniques as in [4] modified as follows. First, we do not need to consider branching on vertices of degree 3 or 4, and this allows of course to choose much more efficient weights. On the other hand, we have to consider two additional constraints. Indeed, the bound on effective running time of the algorithm we use for degree 3 and 4 must be lower than the complexity we claim. Since we provided in Theorem 1 an algorithm in $O^*(\gamma_3^n(\gamma_4/\gamma_3)^{2m-3n})$, we have to verify that for any graph of maximum degree 4 or less and average degree $d = 3 + n_4/n$ (note that $n_3+n_4 = n$) $\gamma_3^n \left(\frac{\gamma_4}{\gamma_3}\right)^{(d-3)n} \leq \gamma^{w_3 n_3 + w_4 n_4}$, or equivalently $\log \gamma_3 + (d-3)(\log \gamma_4 - \log \gamma_3) \leq \log \gamma((4-d)w_3 + (d-3)w_4)$. Notice that $4-d$ and $d-3$ are nonnegative, thus this inequality is a consequence of $w_3 \geq \frac{\log \gamma_3}{\log \gamma}$ and $w_4 \geq \frac{\log \gamma_4}{\log \gamma}$. 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 $\gamma = 1.2135$. □

As consequences we get running times of $O^*(1.1935^n)$ and $O^*(1.2083^n)$ in graphs of maximum degree 5 and 6 (by the fact that $\sum_{v \in V} w(v) \leq w_\Delta n$). Moreover, as mentioned before, it is possible to improve them slightly.

Theorem 2. *On graphs of maximum degree* 5, MAX INDEPENDENT SET *can be solved with running time* $O^*(1.1918^n)$.

Proof. In graphs of maximum degree Δ, inequalities $w_d \leq 1$ for any $d > \Delta$ are not relevant anymore (since initially $\sum_{v \in V} w(v) \leq n$ as soon as $w_d \leq 1$ for $d \leq \Delta$). However, the values w_d must respect the constraint that folding does not increase the total weight of the graph. If we use weights $w_3^5 = 0.52161$, $w_4^5 = 0.83161$, $w_5^5 = 1$ and $w_6^5 = 1.16839$ and $w_d^5 = 1.33678$ for $d > 6$, all the constraints are satisfied and we get $\gamma_5 = 1.1918$. □

Theorem 3. *It is possible to solve* MAX INDEPENDENT SET *in time* $O^*(1.2127^n)$.

Proof. As discussed above, MAX INDEPENDENT SET is solvable on graphs of maximum degree Δ in time $O^*(\gamma_\Delta^n)$, where $\gamma_3 = 1.0854$, $\gamma_4 = 1.1571$ and $\gamma_5 = 1.1918$. Our algorithm works as follows: while $\Delta \geq 6$, run the same algorithm as in [4]. Once $\Delta \leq 5$, run algorithm described in Theorem 2 based upon our improvements for $\Delta \leq 5$. The additional constraint is now $\gamma_5^{w_3^5 n_3 + w_4^5 n_4 + n_5} \leq \gamma^{w_3 n_3 + w_4 n_4 + w_5 n_5}$, which is a consequence of $w_i \geq w_i^5 \frac{\log \gamma_5}{\log \gamma}$, $\forall i \leq 5$. The weights: $w_3 = 0.47459$, $w_4 = 0.75665$, $w_5 = 0.90986$, $w_6 = 0.9757$, $w_7 = 0.9994$ and $w_d = 1$ for $d \geq 8$ satisfy all the constraints and lead to $\gamma = 1.2127$. □

As a consequence of the proof of Theorem 3, we solve MAX INDEPENDENT SET in graphs of degree at most 6 in time $O^*(1.2127^{w_6 n}) = O^*(1.2071^n)$.

As mentioned in the introduction, a recent article provides an algorithm solving MAX INDEPENDENT SET in time $O^*(1.2132^n)$. Considering this new result

instead of the bound of [4], we obtain the following time bounds: $O^*(1.1895^n)$, $O^*(1.2050^n)$ and $O^*(1.2114^n)$ for respectively graphs of maximum degree 5, 6, and for general graphs.

5 Conclusion

The complexity measure of the method proposed in the paper may appear, at least for the two problems considered, as an adaptation of measure and conquer for bounded degree instances. Indeed, for example, $\gamma_d^m y^{2m-dn}$ can be easily written as $2^{\sum_i w_i n_i}$ where n_i is the number of vertices of degree i. However, first, the way we seek the complexity in the bottom up method actually specifies the strong links that have to be verified between weights in order to use as efficiently as possible an algorithm for lower degree graphs (or, more generally for low density instances). The second point is to exhibit and to fruitfully use the link between density and branching. A recursive application of the method then allows to take into account situations where a good branching necessarily occurs to derive good complexity bounds.

Though the precise links between bottom-up and measure and conquer is not very clear yet, at least these two points seem not to be considered in a usual measure and conquer analysis. Furthermore, the results obtained for MAX INDEPENDENT SET by using it, the best known until now, are interesting per se.

References

1. Beigel, R.: Finding maximum independent sets in sparse and general graphs. In: Proc. SODA'99, pp. 856–857 (1999)
2. Bourgeois, N., Escoffier, B., Paschos, V.T.: An $O^*(1.0977^n)$ exact algorithm for MAX INDEPENDENT SET in sparse graphs. In: Grohe, M., Niedermeier, R. (eds.) IWPEC 2008. LNCS, vol. 5018, pp. 55–65. Springer, Heidelberg (2008)
3. Bourgeois, N., Escoffier, B., Paschos, V.T., van Rooij, J.M.M.: Fast Algorithms for Max Independent Set in Graphs of Small Average Degree. CoRR, abs/0901.1563 (2009)
4. Fomin, F.V., Grandoni, F., Kratsch, D.: A measure and conquer approach for the analysis of exact algorithms. Journal of the ACM 56(5) (2009)
5. Kneis, J., Langer, A., Rossmanith, P.: A Fine-grained Analysis of a Simple Independent Set Algorithm. In: Proc. FSTTCS 2009, pp. 287–298 (2009)
6. Razgon, I.: Faster computation of maximum independent set and parameterized vertex cover for graphs with maximum degree 3. J. Discrete Algorithms 7, 191–212 (2009)
7. Robson, J.M.: Algorithms for maximum independent sets. J. Algorithms 7(3), 425–440 (1986)
8. Robson, J.M.: Finding a maximum independent set in time $O(2^{n/4})$. Technical Report 1251-01, LaBRI, Université de Bordeaux I (2001)
9. Van Rooij, J.M.M., Bodlaender, H.L.: Design by measure and conquer a faster algorithm for dominating set. In: STACS 2008, pp. 657–668 (2008)
10. Woeginger, G.J.: Exact algorithms for NP-hard problems: a survey. In: Jünger, M., Reinelt, G., Rinaldi, G. (eds.) Combinatorial Optimization - Eureka!, You Shrink! LNCS, vol. 2570, pp. 185–207. Springer, Heidelberg (2003)