A Faster Solving of the Maximum Independent Set Problem for Graphs with Maximal Degree 3

Igor Razgon

ABSTRACT. We present an $O(1.1034^n)$ algorithm computing a maximum independent set of a graph with maximal degree 3. This result improves currently best upper bound of $O(1.1255^n)$ for the problem obtained by Chen et al [2].

1 Introduction

Maximum independent set problem (MIS) is one of the most extensively studied problems in the area of exact algorithms. The best existing algorithms are proposed by Fomin et al [5] ($O(1.2210^n)$, polynomial space) and Robson [7] ($O(1.1893^n)$, exponential space). A special case of the problem is finding a MIS for a graph with max-degree 3 (MIS-3). Currently, the best upper bound for the problem was achieved by Chen et al [2]. Their algorithms takes $O(1.1255^n)$. This is a slight improvement of the $O(1.1259^n)$ bound provided by Beigel [1].

In this paper we propose a "branch-and-prune" algorithm that solves a MIS-3 problem in time $O(1.1034^n)$. This upper bound is achieved by combination of three main ingredients. First of all, the size of the residual graph is measured in the number of vertices of degree 3, without taking into account the rest of the vertices. Using a non-standard measure is a strategy that was suggested in [4] and then successfully applied in [5, 3, 6] for exact solving of various intractable problems. The second ingredient is simplification of the residual graph after each decision (selecting or removing of a vertex) made by the algorithm. The graph obtained as a result of the simplification is cubic, which allows easy application of the above non-standard measure. The ideas of simplification are similar to those that appear in [2]. The last ingredient is a compact classification of a variety of cases that can happen as a result of selection or removing of a vertex. According to the proposed classification, only four cases are considered: a vertex cut of size 3, a cycle of length 4, a cycle of length 3, and the case when none of the above three happens.

The rest of the paper is structured as follows. Section 2 presents the necessary notations, Section 3 describes the algorithm, Section 4 introduces the complexity analysis.

2 Notations

A simple undirected graph $G = (V, E)$ is referred in this paper as *a graph*, $V(G)$ and $E(G)$ denote the set of vertices and edges of G, respectively. For $u \in V(G)$, we denote by $N_G(u)$ the set of neighbours of u, $N_G(u) \cup \{u\}$ is denoted by $N^+_G(u)$. For $S \subseteq V(G)$, we denote by $N_G(S)$ the set of neighbours of the vertices of S that do not belong to S; $N^+_G(S)$ denotes $N_G(S) \cup S$. The subscript may be omitted if there is no risk of confusion. We denote by $G \setminus S$ the graph induced by $V(G) \setminus S$. If S consists of a single vertex u, we write $G \setminus u$ rather than $G \setminus \{u\}$. If $S = V(G_1)$, where G_1 is a subgraph of G, we write $G \setminus G_1$ rather than $G \setminus V(G_1)$. Set S is a maximum independent set of G if the vertices of S are mutually non-adjacent and the set S is largest subject to this property. We denote by $mis(G)$ the size of a MIS of G.

Finally, we introduce a non-standard definition which will simplify presentation of the proposed algorithm. Given a graph G with max-degree 3, we refer to a connected component of G with at most 20 vertices a *small* component.

3 The Algorithm

In this section we present a function $FindIndep$ that, given a graph G with max-degree 3, returns a MIS of G. The pseudocode is presented in Algorithm 1, 2, and 3, and 4. The function $FindIndep(G)$ checks whether G satisfies one of the predefined conditions. The first condition satisfied by G determines the operations to be performed.

If a condition checked in Algorithm 1 is satisfied then either a MIS of G is found efficiently or there is a subset of vertices of G to be included into any MIS of G. The latter happens when G has a small component or G has a vertex cut of size at most 2, after removing of which a small component is obtained (in case of vertex cut of size 2, the union of all small components is required to have at least 2 vertices). The decisions made if anyone of these conditions is satisfied are *nonbranching* in the sense that $FindIndep(G)$ is applied recursively only once. The case of vertex cut of size 2 requires checking of a number of subcases, therefore we describe it as a separate function $ProcessTwoCut$ (Algorithm 2).

If none of the above conditions is satisfied, $FindIndep(G)$ calls a function $ProcessBranches(G)$, which makes a *branching* decision, i.e. applies $FindIndep(G)$ to two different subgraphs of G and returns the largest of

the two resulting sets. Before looking at the pseudocode, let us extend our terminology.

Observe first that if $ProcessBranches(G)$ is called, graph G has the following properties: there are no isolated vertices and vertices of degree 1, any vertex of degree 2 is adjacent to two nonadjacent vertices of degree 3, there are no two vertices of degree 2 adjacent to the same pair of vertices. This observation enables us to define a graph $C(G)$ obtained from G by replacing each vertex of degree 2 adjacent to vertices u and v by an edge between u and v. We call the edges of $C(G)$ that appear in G the *normal* edges and the edges that replace degree-two vertices the *odd* edges. A cycle in $C(G)$ all edges of which are normal is called a *normal cycle*. Otherwise it is called an *odd cycle*.

$ProcessBranches(G)$ tests whether G possesses a number of properties. The first property detected determines the operation to be performed. For most of the cases, $ProcessBranches(G)$ calls function $VertexBranch$ with G and a particular vertex u as parameters. The function $VertexBranch$ returns the largest one of two independent sets. The *first* set includes u, all the vertices connected to u by paths of odd edges, and the set returned by $FindIndep$ applied to the residual graph. The *second* set is returned by $FindIndep(G \setminus u)$.

THEOREM 1. *Let G be a graph with max-degree 3. Then $FindIndep(G)$ returns a MIS of G.*

Proof. Omitted due to space constraints. ■

4 Complexity Analysis

In this section we analyze complexity of $FindIndep(G)$ for a graph G with max-degree 3, by evaluation of the number of recursive calls done during execution of $FindIndep(G)$. Let us extend first the terminology. We say that G is a *non-trivial* graph if $FindIndep(G)$ calls $ProcessBranches(G)$. A graph G' is a *successor* of G if $FindIndep(G')$ is called during processing of $FidIndep(G)$. If, in addition, G' is nontrivial, it is a *non-trivial successor* (NS) of G. Further, we say that G' is a *first non-trivial successor* (FNS) of G, if G has no NS G'' such that G' is a NS of G''. Note that G can have at most two FNSes because $FindIndep(G)$ generates a binary search tree. If G has two FNSes, we call the one explored by the first (chronologically) branch, the *left* FNS of G and denote it by G_L. Accordingly, the other one is called the *right* FNS of G and denoted by G_R.

LEMMA 2. *Let G be a nontrivial graph such that $ProcessBranches(G)$ executes line 2. Assume that G has 2 FNSes. Then $|V(C(G_L))| \leq |V(C(G))| - 8$, $|V(C(G_R))| \leq |V(C(G))| - 8$.*

Algorithm 1 $FindIndep(G)$

if G has no vertices of degree 3 **then**
  Find a MIS of G efficiently and return it
elseif G has a small component G' **then**
  Let S be a MIS of G'
  Return $S \cup FindIndep(G \setminus G')$
elseif G has an articulation point u such that $G \setminus u$ has a small component G' **then**
  if $mis(G' \setminus N(u)) = mis(G')$ **then**
    Let S be a MIS of $G' \setminus N(u)$
    Return $S \cup FindIndep(G \setminus G')$
  else
    Let S be a MIS of G'
    Return $S \cup FindIndep((G \setminus G') \setminus u)$
  endif
elseif G has a vertex cut $\{u_1, u_2\}$ such that the union of small components G' of $G \setminus \{u_1, u_2\}$ contains at least 2 vertices **then**
  Return $ProcessTwoCut(G, G', u_1, u_2)$
else Return $ProcessBranches(G)$
endif

Algorithm 2 $ProcessTwoCut(G, G_2, u_1, u_2)$

if $mis(G_2 \setminus (N(u_1) \cup N(u_2))) = mis(G_2)$ **then**
  $S_1 \leftarrow FindIndep(G \setminus G_2)$
  Let S_2 be a MIS of $G_2 \setminus (N(u_1) \cup N(u_2))$
  Return $S_1 \cup S_2$
elseif $mis(G_2 \setminus N(u_1)) < mis(G_2 \setminus N(u_2)) = mis(G_2)$ **then**
  $S_1 \leftarrow FindIndep((G \setminus G_2) \setminus u_1)$
  Let S_2 be a MIS of $G_2 \setminus N(u_2)$
  Return $S_1 \cup S_2$
elseif $mis(G_2 \setminus N(u_2)) < mis(G_2 \setminus N(u_1)) = mis(G_2)$ **then**
  This case is symmetric to the previous one
elseif $mis(G_2 \setminus N(u_1)) = mis(G_2 \setminus N(u_2)) = mis(G_2)$ **then**
  $G' \leftarrow (V(G \setminus G_2), E(G \setminus G_2) \cup \{\{u_1, u_2\}\})$
  $S_1 \leftarrow FindIndep(G')$
  Let S_2 be a MIS of $G_2 \setminus N(S_1)$
  Return $S_1 \cup S_2$
elseif $mis(G_2 \setminus (N(u_1) \cup N(u_2))) \leq mis(G_2) - 2$ **then**
  $S_1 \leftarrow FindIndep((G \setminus G_2) \setminus \{u_1, u_2\})$
  Let S_2 be a MIS of G_2
  Return $S_1 \cup S_2$
else
  $G' \leftarrow (V(G \setminus G_2) \cup \{w\}, E(G \setminus G_2) \cup \{\{u_1, w\}, \{u_2, w\}\})$ (w is not a vertex of G)
  $S_1 \leftarrow FindIndep(G')$
  if $\{u_1, u_2\} \subseteq S_1$ **then**
    Let S_2 be a MIS of $G_2 \setminus (N(u_1) \cup N(u_2))$
  else
    Let S_2 be a MIS of G_2
    $S_1 \leftarrow S_1 \setminus \{w, u_1, u_2\}$
  endif
  Return $S_1 \cup S_2$
endif

Algorithm 3 $ProcessBranches(G)$

if there is a cut $\{u_1, u_2, u_3\}$ of $C(G)$ such that $C(G) \setminus \{u_1, u_2, u_3\}$ has a small component and the union G of all small components contains at least 5 vertices **then**
  Return $VertexBranch(G, u_1)$ (we assume w. l. o. g. that in $\{u_1, u_2, u_3\}$, u_1 has the largest number of neighbours outside G')
elseif $C(G)$ has a normal rectangle (u_1, u_2, u_3, u_4) (vertices are listed according to their appearance in the cycle) **then**
  $S_1 \leftarrow \{u_1\} \cup FindIndep(G \setminus N^+(u_1))$
  $S_2 \leftarrow FindIndep(G \setminus \{u_1, u_3\})$
  Return $max(S_1, S_2)$
elseif $C(G)$ has an odd rectangle (u_1, u_2, u_3, u_4) (we assume w.l.o.g. that u_1 is incident to at least one odd edge) **then**
  $VertexBranch(G, u_1)$
elseif $C(G)$ has a normal triangle (u_1, u_2, u_3) **then**
  Let v_1 be the outside vertex adjacent to u_1 in $C(G)$
  if $\{u_1, v_1\}$ is a normal edge
    $S_1 \leftarrow \{u_1\} \cup FindIndep(G \setminus N^+(u_1))$
    $S_2 \leftarrow \{v_1\} \cup FindIndep(G \setminus N^+(v_1))$
    Return $max(S_1, S_2)$
  else
    $VertexBranch(G, u_1)$
  endif
elseif $C(G)$ has an odd rectangle (u_1, u_2, u_3) (we assume w.l.o.g. that u_1 is incident to at least one odd edge) **then**
  $VertexBranch(G, u_1)$
else
  Select an arbitrary vertex u of $C(G)$ (if available, incident to an odd edge)
  $VertexBranch(G, u)$
endif

Algorithm 4 $VertexBranch(G, v)$

1: Let T be the set of vertices of $C(G)$ including v and the vertices that are connected to v by odd edges
2: **if** T is not an independent set in G **then**
3: Return $FindIndep(G \setminus v)$
4: **endif**
5: Return $max(T \cup FindIndep(G \setminus N^+(T)), FindIndep(G \setminus v))$

Proof. Let G' be as defined in line 1 of Algorithm 3. Let G^* be a NS of G. If $C(G^*)$ contains at least one vertex of G' then, depending on presence of u_2 and u_3 in G^*, the latter is not connected or has a vertex cut of size at most 2 separating a small component, in contradiction to non-triviality of G^*. It follows that both $C(G_L)$ and $C(G_R)$ do not contain any vertex of G', consequently $|V(C(G_L))| \leq |V(C(G))|-5$ and $|V(C(G_R))| \leq |V(C(G))|-5$. Actually, 5 in the above inequalities can be replaced by 6 because neither $C(G_L)$ nor $C(G_R)$ contain u_1. That is, if G' contains 7 or more vertices, there is nothing to prove, so assume that $5 \leq |V(G')| \leq 6$.

Note that u_1 has at least one neighbour v_1 in $C(G)$ that does not belong to $V(G') \cup \{u_2, u_3\}$ for otherwise, $\{u_2, u_3\}$ would constitute a vertex cut of size 2 separating a small component, in contradiction to the non-triviality of G. If u_1 has another neighbour $v_2 \notin V(G') \cup \{u_2, u_3\}$, the lemma easily follows because neither of v_1 and v_2 is contained in either of $V(C(G_L))$ and $V(C(G_R))$.

If v_1 is the only "outside" neighbour then, by the selection of u_1, two other vertices u_2 and u_3 also have outside neighbours. Denote these neighbours by v_2 and v_3 (note that v_1, v_2, and v_3 do not coincide because in that case they constitute a vertex cut of size smaller than 3). After removing of u_1, v_2 and v_3 comprise a cut of size 2 which means that not only the vertices of G' but also u_2 and u_3 are excluded from G_L and G_R, which proves the desired inequalities. ■

LEMMA 3. *Let G be a nontrivial graph such that $ProcessBranches(G)$ executes lines 4-6 or line 8 and let (u_1, u_2, u_3, u_4) be the rectangle picked by the function. Then there are vertices $v_1, \ldots, v_4, w_1, w_4$ that together with the rectangle form a subgraph of $C(G)$ isomorphic to the one shown in Figure 1 a)* [1]

[1] Note that the lemma does not claim that this graph is induced by these vertices. Additional edges, if exist, do not relevant to the complexity analysis.

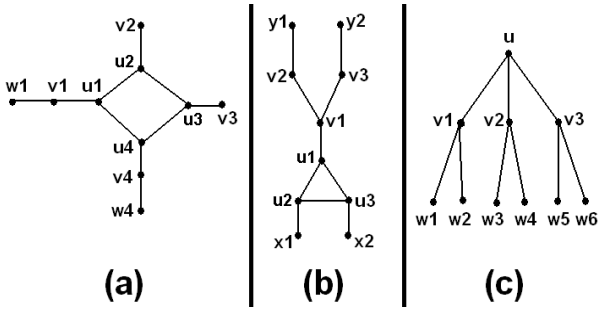

Figure 1. The considered subgraphs of $C(G)$

Proof. Observe that in the considered case the conditions related to vertex cuts of size 2 and 3 are not satisfied.

It follows that each vertex of the rectangle has the third neighbour outside the rectangle and that all these neighbours are distinct. Otherwise, if, for example u_1 and u_3 are adjacent or some two vertices have the common third neighbour, we obtain the case of vertex cut of size 2 or 3 separating a small component. Denote the neighbours of u_1 to u_4 by v_1 to v_4, respectively.

Now, let us concentrate on vertices v_1 and v_4. Observe that there must be two *distinct* "outside" vertices w_1 and w_4 adjacent to v_1 and v_4, respectively. If none of v_1 and v_4 is adjacent to an outside vertex then v_2 and v_3 form a cut of size 2 separating a small component. If there is only one outside vertex w adjacent to either of v_1 and v_4 (or to both of them) then $\{w, v_2, v_3\}$ is a cut of size 3 that disconnects a component of size 6. ■

LEMMA 4. *Let G be a nontrivial graph such that $ProcessBranches(G)$ executes lines 4-6 or line 8 and let (u_1, u_2, u_3, u_4) be the rectangle picked by the function. Assume that G has two nontrivial successors. Then one of the following two statements holds:*

- $|V(C(G_L))| \leq |V(C(G))| - 8$ *and* $|V(C(G_R))| \leq |V(C(G))| - 8$.
- $|V(C(G_L))| \leq |V(C(G))| - 10$ *and* $|V(C(G_R))| \leq |V(C(G))| - 6$.

Proof. Based on Lemma 3, we consider only the configuration shown in Figure 1 a). Assume that the rectangle is normal. Then lines 4-6 of Algorithm 3 are performed. It follows that all of $u_1, \ldots, u_4$ and $v_1, \ldots, v_4$ do not belong to $V(C(G_L))$. When u_1 and u_3 are excluded, the degree of u_2 and u_4 is 1. It follows that all of $u_1, \ldots, u_4$ and $v_1, \ldots, v_4$ do not belong to $V(C(G_R))$. Thus the first statement holds in the considered case.

Assume now that the edge $\{u_1, u_4\}$ is odd. Clearly, neither of the vertices in Figure 1 a) belongs to $V(C(G_L))$. Also, the vertices $u_1, \dots u_4, v_1, v_4$ do not belong to $V(C(G_R))$, implying the second statement.

Note that the rectangle is either even or odd and in the latter case assuming any edge to be odd does not restrict generality (we could prove, for example, the existence of the outside neighbours for u_2 and u_3 and then assume that $\{u_2, u_3\}$ is odd). Hence, the lemma follows. ■

LEMMA 5. *Assume that G is a nontrivial graph and that $ProcessBranches(G)$ picks a triangle u_1, u_2, u_3 in $C(G)$. Then there are vertices $v_1, v_2, v_3, x_1, x_2, y_1, y_2$ that together with u_1, u_2, u_3 form a subgraph of $C(G)$ isomorphic to the one shown in Figure 1 b).*

Proof. Observe that if $ProcessBranches(G)$ picks a triangle then $C(G)$ contains no rectangle. Let v_1 be the neighbour of u_1 in $C(G)$ outside of the triangle and x_1 and x_2 be the respective "outside" neighbours of u_2 and u_3. Note that v_1, x_1, x_2 are pairwise different because otherwise $C(G)$ contains a rectangle. Let v_2 and v_3 be the remaining two neighbours of v_1. Note that the same reason of appearing a rectangle forbids v_2 and v_3 to coincide with x_1 and x_2. Further, both v_2 and v_3 have neighbours that do not belong to the already considered vertices, just to avoid being $\{x_1, x_2\}$ a vertex cut separating a small component. Moreover, it is impossible that there is only one outside vertex y adjacent to v_2 or v_3 or both because y together with x_1 and x_2 constitute a 3-cut. Hence each of v_2 and v_3 has a "personal" neighbour y_1 and y_2, respectively. ■

LEMMA 6. *Assume that G is a nontrivial graph with two FNSes and that $ProcessBranches(G)$ picks a triangle u_1, u_2, u_3 in $C(G)$. Then $|V(C(G_L))| \leq |V(C(G))|-10$ and $|V(C(G_R))| \leq |V(C(G))|-6$ or $|V(C(G_L))| \leq |V(C(G))|-8$ and $|V(C(G_R))| \leq |V(C(G))| - 8$ or $|V(C(G_L))| \leq |V(C(G))| - 9$ and $|V(C(G_R))| \leq |V(C(G))| - 7$.*

Proof. Based on Lemma 6 we consider only configuration presented in Figure 1 b).

Assume first that lines 11-14 of Algorithm 3 are executed. Then $u_1, \dots, u_3, x_1, x_2, v_1, \dots, v_3$ are excluded from $V(C(G_L))$, $u_1, \dots, u_3, y_1, y_2, v_1, \dots, v_3$ are excluded from $V(C(G_R))$, hence $|V(C(G_L))| \leq |V(C(G))|-8$ and $|V(C(G_R))| \leq |V(C(G))|-8$.

Assume now that line 16 is executed. Clearly neither of vertices in Figure 1 belong to G_L. Also, $u_1, \dots, u_3, v_1, \dots, v_3$ do not belong to G_R, consequently $|V(C(G_L))| \leq |V(C(G))| - 10$ and $|V(C(G_R))| \leq |V(C(G))| - 6$.

For the case of line 19, note that x_1 and x_2 must have distinct neighbours z_1 and z_2 which are different from $u_1, \dots, u_3, v_1, \dots, v_3$, otherwise, analogously to the argumentation in Lemmas 3 and 5, we can prove existence of a vertex cut separating a small component.

Assume w.l.o.g, that the edge $\{u_1, u_2\}$ is odd. Now we split the proof into the case where at least one of the remaining edges of the triangle is odd and the case when both the remaining edges are normal. In the former case all of $u_1, \dots, u_3, x_1, x_2, v_1, \dots, v_3, z_1, z_2$ do not belong to G_L, $u_1, \dots, u_3, v_1, x_1, x_2$ do not belong to G_R, hence $|V(C(G_L))| \leq |V(C(G))|-10$ and $|V(C(G_R))| \leq |V(C(G))| - 6$.

If the latter case, all of $u_1, \dots, u_3, x_1, x_2, v_1, \dots, v_3, z_1$ do not belong to G_L. In the branch that excludes u_1, the vertex u_3 remains of degree 1, which causes excluding of x_2 and z_2 by non-branching calls. To summarize $u_1, u_2, u_3, v_1, x_1, x_2, z_2$ do not belong to $V(C(G_R))$, what causes $|V(C(G_L))| \leq |V(C(G))| - 9$ and $|V(C(G_R))| \leq |V(C(G))| - 7$. ■

LEMMA 7. *Let G be a nontrivial graph and assume that $ProcessBranches(G)$ executes lines 21-22 of Algorithm 3 and assume that G has 2 FNSes. Then one of the following statements holds.*

1. $|V(C(G_L))| \leq |V(C(G))| - 10$ *and* $|V(C(G_R))| \leq |V(C(G))| - 6$.
2. $|V(C(G_L))| \leq |V(C(G))| - 10$ *and* $|V(C(G_R))| \leq |V(C(G))| - 10$.
3. $|V(C(G_L))| \leq |V(C(G))| - 10$ *and* $|V(C(G_R))| \leq |V(C(G))| - 4$. *If G_R has two FNSes then* $|V(C((G_R)_L))| \leq |V(C(G_R))| - 10$ *and* $|V(C((G_R)_R))| \leq |V(C(G_R))| - 6$ *or* $|V(C((G_R)_L))| \leq |V(C(G_R))|-9$ *and* $|V(C((G_R)_R))| \leq |V(C(G_R))|-7$ *or* $|V(C((G_R)_L))| \leq |V(C(G_R))|-8$ *and* $|V(C((G_R)_R))| \leq |V(C(G_R))|-8$

Proof. First of all note that if lines 21-22 are executed then $C(G)$ has no triangles and no rectangles. Then a vertex u together with its neighbours v_1, v_2, v_3 and together with their neighbours form a subgraph of $C(G)$ isomorphic to the one shown in Figure 1 c). If u is incident to an odd edge, say, $\{u, v_1\}$ then all the vertices shown in Figure 1 c) do not belong to G_L. Also $u, v_1, \dots, v_3, w_1, w_2$ do not belong to G_R. Thus if u is incident to an odd edge, the first statement holds.

Assume now that all the edges incident to u are normal. By selection of u this means that all the edges of G are normal. Clearly, $|V(C(G_L))| \leq |V(C(G))| - 10$. Further, observe that after removing of u, three odd edges

are created. If there are no odd edges in $C(G_R)$ then the ends of these odd edges do not belong to $C(G_R)$ i.e. $|V(C(G_R))| \leq |V(C(G))| - 10$ as in the second statement. Otherwise, one of the conditions *before* line 19 of Algorithm 3 is satisfied or a vertex incident to an odd edge is selected in line 21. By Lemmas 2, 4, 6, and by the first statement of the present lemma, we get the last three inequalities. ■

THEOREM 8. *For a graph G with n vertices and max-degree 3, $FindIndep(G)$ returns a MIS of G in $O(1.1034^n)$.*

Proof. We define a search tree $ST(G)$ generated by $FindIndep(G)$. The nodes of the tree are associated with the graphs to which $FindIndep$ is recursively applied during processing of $FindIndep(G)$. Assume w.l.o.g. that G is nontrivial. Then the root of $ST(G)$ is associated with G. Let x be a node of $ST(G)$ associated with graph G'. If G' has no NSes then x is a leaf. Otherwise the children of x correspond to the FNSes of G'.

Let $T(n)$ be the maximal number of nodes of $ST(G)$ for a graph G with $|V(C(G))| = n$. The complexity of $FindIndep(G)$ can be represented as $T(n)$ multiplied to a polynomial. Taking into account that $T(n)$ is exponential in n, the complexity of $FindIndep(G)$ can be represented as $T(n)$.

If $C(G)$ has a vertex cut of size 3 that separates a union of small components with at least 5 vertices then, by Lemma 2, the size of $ST(G)$ is at most $T(n-8)+T(n-8)+1$, where the first two items bound the sizes of subtrees rooted by the FNSes of G and the constant stands for the root node. Analogously, if $C(G)$ has a rectangle or a triangle then the size of $ST(G)$ can be bounded by $T(n-10)+T(n-6)+1$ or $T(n-9)+T(n-7)+1$ (Lemmas 4 and 6). If $C(G)$ has no appropriate cut of size 3, no triangles, and no rectangles then by Lemma 7, the size of $ST(G)$ can be bounded by $T(n-10)+T(n-10)+1$ or by $T(n-10)+T(n-14)+T(n-10)+2$ or by $T(n-10)+T(n-12)+T(n-12)+2$ or by $T(n-10)+T(n-13)+T(n-11)$. In the expression $T(n-10)+T(n-14)+T(n-10)+2$, the first three items bound the number of nodes in $ST(G_L)$, $ST((G_R)_L)$, and $ST((G_R)_R)$, respectively; the last item represents the nodes of $ST(G)$ associated with G and G_R. The last two expressions can be interpreted analogously.

Taking into account that the size of $ST(G)$ is bounded by $T(n)$, we obtain that $T(n) = T(n-8)+T(n-8)$ or $T(n) = T(n-10)+T(n-6)$ or $T(n) = T(n-9)+T(n-7)$ or $T(n) = T(n-10)+T(n-10)$ or $T(n) = T(n-10)+T(n-14)+T(n-10)$ or $T(n) = T(n-10)+T(n-12)+T(n-12)$ or $T(n) = T(n-10)+T(n-13)+T(n-11)$. The constants in the right-hand side of the expressions are omitted because they contribute only a polynomial factor to the resulting exponential function. Due to the same reason, we do not consider the cases where G has only one FNS. Each one

of the obtained recursive formulas corresponds to an exponential function. In particular if $T(n) = T(n-k_1)+\cdots+T(n-k_l)$ then $T(n) = O(c^n)$, where c is the largest solution of the equation $1 = 1/c^{k_1} + \cdots + 1/c^{k_l}$. It can be shown that $T(n)$ is expressed by the recursive formula that corresponds to a function with the largest base of the exponents. A simple computation shows that this formula is $T(n) = T(n-10) + T(n-14) + T(n-10)$, and the resulting exponential function is $T(n) = O(1.1034^n)$. ■

BIBLIOGRAPHY

[1] Richard Beigel. Finding maximum independent sets in sparse and general graphs. In *SODA*, pages 856–857, 1999.

[2] Jianer Chen, Iyad A. Kanj, and Ge Xia. Labeled search trees and amortized analysis: Improved upper bounds for NP-Hard problems. *Algorithmica*, 43(4):245–273, 2005.

[3] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination - a case study. In *ICALP*, pages 191–203, 2005.

[4] F. Fomin, F. Grandoni, and D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS*, 87:47–77, 2005.

[5] F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: A simple $O(2^0.288n)$ independent set algorithm. In *SODA 2006*, 2006.

[6] Igor Razgon. Exact computation of maximum induced forest. In *SWAT 2006*, pages 160–171, 2006.

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

Igor Razgon

University College Cork, Ireland

i.razgon@cs.ucc.ie