Computing Minimum Directed Feedback Vertex Set in $O^*(1.9977^n)$

Igor Razgon *

i.razgon@cs.ucc.ie

Computer Science Department, University College Cork, Ireland

In this paper we propose an algorithm which, given a directed graph G, finds the minimum directed feedback vertex set (FVS) of G in $O^*(1.9977^n)$ time and polynomial space. To the best of our knowledge, this is the first algorithm computing the minimum directed FVS faster than in $O(2^n)$. The algorithm is based on the branch-and-prune principle. The minimum directed FVS is obtained through computing of the complement, i.e. the maximum induced directed acyclic graph. To evaluate the time complexity, we use the measure-and-conquer strategy according to which the vertices are assigned with weights and the size of the problem is measured in the sum of weights of vertices of the given graph rather than in the number of the vertices.

1. Introduction

In this paper we consider the following problem: given a directed graph G, find the maximum acyclic subset (MAS) of G i.e. the largest subset of vertices of G inducing a directed acyclic graph (DAG). We propose an algorithm solving this problem in $O^*(1.9977^n)$ time and polynomial space. The complement of MAS is the minimum directed Feedback Vertex Set (FVS). The directed FVS problem is one of the "canonical" NP-hard optimization problems whose NP-complete version is mentioned in [8]. Thus the proposed algorithm solves the directed FVS problems as well. To the best of our knowledge, this is the first algorithm solving this problem faster than in $O(2^n)$. For the undirected version of the FVS problem, the $O(2^n)$ barrier has been broken by Razgon [9].

The proposed algorithm belongs to the area of exact exponential algorithms whose subject is design of algorithms solving intractable problems

*I dedicate this paper to my son Gabriel Razgon who was born on 14/04/2007, just one day before the ICTCS 2007 abstract submission deadline.

faster than brute-force enumeration of all the possibilities (see a tutorial of Woeginger [10] for introduction to the field). However, the directed FVS problem is considered challenging and interesting in other areas of Theoretical Computer Science, especially Parameterized Complexity and Approximation Algorithms. Therefore, the proposed result may be interesting to a broader audience of researchers as providing a new insight into the nature of the directed FVS problem.

The proposed algorithm is based on the branch-and-prune approach. That is, the algorithm selects a vertex v of the given graph G, finds the largest acyclic subset of G containing v and the largest one without v, and returns the larger of the above two. These two subsets are found by recursive application of the algorithm to the corresponding *residual* graphs. The $O(2^n)$ barrier can be easily broken if one shows that selection or removal of v necessarily causes removal of additional vertices from the respective residual graph. For example, for the Maximum Independent Set (MIS) problem that can be easily seen since selection of a non-isolated vertex causes removal of its neighbors. In our case, this is not always possible. For example, if graph G is strongly connected (that is, the problem cannot be divided into a number of independent subproblems) and has no cycles of size 2 (that is, selection of vertex v does not cause removal of an additional vertex) then many vertices may be selected or removed before at least one additional vertex can be eliminated from the residual graph. To overcome, the difficulty, we associate vertices with weights and measure the size of the problem as the sum of weights of the vertices rather than the number of vertices. If the given branching decision does not cause *real* elimination of additional vertices, the weights of the vertices which are likely to be removed in the future are decreased. This updating of weights "amortizes" the effect of vertex elimination among a number of iterations so that each branching decision gets "a small bit" of the effect sufficient for breaking the $O(2^n)$ barrier.

The above methodology of complexity analysis called *Measure-and-Conquer* is quite recent [1,2,4] but proved very successful in the last two years: it served as a basis of design and analysis of algorithms for such problems as Dominating Set [7], MIS [5], undirected FVS [3,9], connected Dominating Set [6].

The rest of the paper is structured as follows. Section 2 introduces the necessary terminology. Section 3 presents the algorithm, proves its correctness, and describes intuitively why the algorithm breaks the $O(2^n)$ barrier. Section 4 presents complexity analysis of the algorithm which is, essen-

tially, formalization of the intuitive description given in Section 3. Due to the space constraints, some proofs or parts of them are omitted. †

2. Preliminaries

All graphs considered in the paper are directed graphs without loops and multiple arcs. Let G be a directed graph with the set of vertices $V(G)$ and the set of arcs $A(G)$. Let $v, w \in V(G)$. If $(w, v) \in A(G)$, we say that w is an *entering* neighbour of v and v is a *leaving* neighbour of w.

A subset S of $V(G)$ is a directed Feedback Vertex Set (FVS), if every directed cycle of G contains at least one vertex of S. We call the complement $V(G) \setminus S$ of S an *acyclic subset* of G because it induces an acyclic subgraph of G. A Maximum Acyclic Subset (MAS) is the complement of a minimum directed FVS.

Let $v \in V(G)$. Graph $G^C(v)$ is obtained from $G \setminus v$ as follows. For each entering neighbour u of v and for each leaving neighbour w of v, an edge from u to w is added (if there is no such edge in G). If u is both an entering and a leaving neighbour of v then u becomes a *loop vertex*. All loop vertices are removed from the resulting graph.

Let D be a subset of vertices of G. The graph $G^C(D)$ is defined recursively as follows. If $D = \emptyset$ then $G^C(D) = G$. Otherwise, $G^C(D) = (G^C(v))^C(D \setminus \{v\})$ for some $v \in D$. Observe that the definition of $G^C(D)$ makes sense only if D is acyclic in G: otherwise one of the vertices of D will be eventually removed as a loop vertex and there will be no possibility to finish up the recursive construction. We say that $G^C(D)$ is obtained from G as a result of *contraction* of vertices of D.

The complexity of the algorithm proposed in the paper is measured in terms of O^* notation [10], which suppresses polynomial factors. For example, $O(n^2 * 2^n)$ is transformed into $O^*(2^n)$.

3. The Algorithm

We present the algorithm for computing the MAS of G as a recursive procedure $GetMAS(G, R)$.

The parameter R is the function on $V(G)$ such that for $v \in V(G)$, $R(v)$ is the *role* of v. Initially, the role of each vertex is *UNMARKED* (*UM*). During the run of the algorithm, a vertex can change

†The preliminary (unpolished) version of the paper which contains all the proofs is available at http://www.cs.ucc.ie/ ir2/papers/mas1203.pdf

its role to *LEFT MARKED* (*LM*), *RIGHT MARKED* (*RM*), *WEAKLY LEFT MARKED* (*WLM*), *WEAKLY RIGHT MARKED* (*WRM*), *LEFT MARKED DISCONNECTED* (*LMD*), and *RIGHT MARKED DISCONNECTED* (*RMD*). The notion of roles is crucial for the complexity analysis because the vertices are assigned with weights according to their roles. As well, the roles guide the branching decisions made by the algorithm.

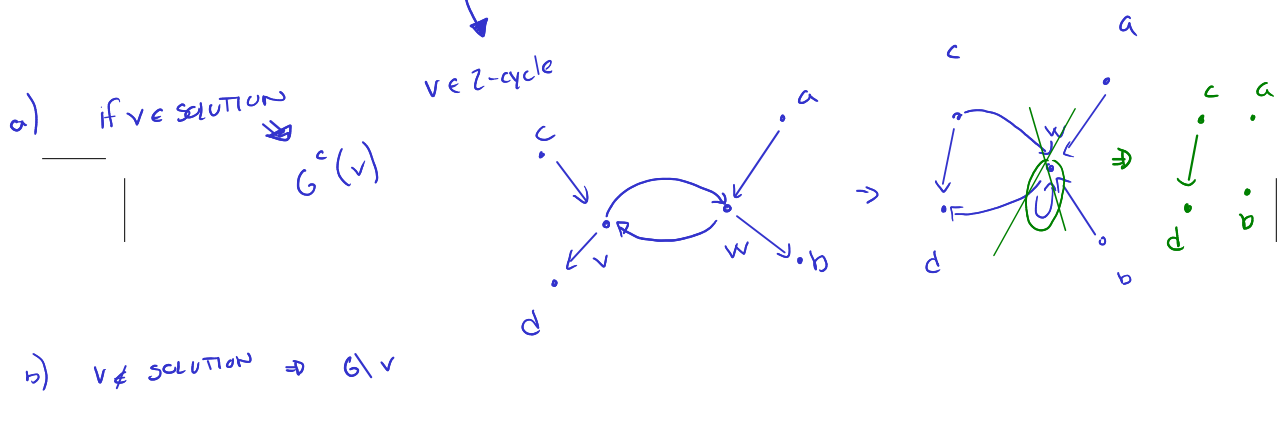
Let us denote by $V_X(G,R)$ the set of vertices v of G such that $R(v) = X$. In the further description of the algorithm, we frequently refer to the sets $V_{LM}(G,R) \cup V_{LMD}(G,R)$ and $V_{RM}(G,R) \cup V_{RMD}(G,R)$. For the sake of succinctness we denote these sets by $VL(G,R)$ and $VR(G,R)$, respectively. We refer to the vertices whose roles are *UM* as *unmarked* vertices and to the rest of the vertices as *marked* ones. Also, the vertices of $VL(G,R) \cup V_{WLM}(G,R)$, $VR(G,R) \cup V_{WRM}(G,R)$, $VL(G,R) \cup VR(G,R)$, $V_{WLM}(G,R) \cup V_{WRM}(G,R)$ are referred as *left-marked*, *right-marked*, *strongly marked*, and *weakly marked*, respectively.

Below we present the algorithm in the form of a list of items. Each item begins with the condition written in bold and associated with a short name in square brackets for easier reference. The condition is followed by the description of operations to be performed, if this condition is satisfied. The conditions are presented in the order they are checked by the algorithm. For each condition but the first one, it is assumed that this condition is checked only if all the previous conditions are not satisfied. The formal description is followed by intuitive explanation why the algorithm breaks the $O(2^n)$ barrier.

We assume that the first operation performed by $GetMAS(G,R)$ (prior to the operations described below) is the *balancing operation* ensuring that $|VL(G,R)|$ and $|VR(G,R)|$ differ by at most 3. In particular if $|VR(G,R)| - |VL(G,R)| > 3$ then arbitrary $|VR(G,R)| - (|VL(G,R)| + 3)$ vertices of $VR(G,R)$ are selected and their roles in R are changed to WRM. Symmetrically, if $|VL(G,R)| - |VR(G,R)| > 3$ then arbitrary $|VL(G,R)| - (|VR(G,R)| + 3)$ vertices of $VL(G,R)$ are selected an their roles in R are changed to WLM.

(1) **[C1] Graph G has at most 3 vertices.** Find a MAS of G efficiently and return it.
(2) **[C2] Graph G has a cycle of length 2.** Let v be a vertex participating in such a cycle. Return the largest set among $\{v\} \cup GetMAS(G^C(v), R')$ and $GetMAS(G \setminus v, R)$. ‡, where R' is computed

‡We assume that R is projected to the vertices of the graph given as the first parameter.



as follows. If v is unmarked then $R' = R$. If v is left-marked then R' is obtained from R by setting to WLM the roles of all unmarked entering neighbors of v. Finally, if v is right-marked then R' is obtained from setting to WRM the roles of all unmarked leaving neighbors of v.

(3) **[C3] Graph G has two or more strongly connected components.** Let v_1 and v_2 be vertices of different strongly connected components G_1 and G_2 that *preferably* belong to $V_{LMD}(G,R) \cup V_{RMD}(G,R)$. Let $T_1 = \{v_1, v_2\} \cup GetMAS(G^C(\{v_1, v_2\}), R)$, $T_2 = GetMAS(G \setminus \{v_1, v_2\}, R)$. Return $max(T_1 \cap V(G_1), T_2 \cap V(G_1)) \cup max(T_1 \cap V(G_2), T_2 \cap V(G_2)) \cup (T_1 \setminus (V(G_1) \cup V(G_2)))$, where $max(S_1, S_2)$ means the larger set of S_1 and S_2.

(4) **[C4] $VL(G,R) = \emptyset$ or $VR(G,R) = \emptyset$.** We identify the following two subcases.

- **[C41] There is an *unmarked* or *weakly marked* vertex v with the in-degree at most 3 or the out-degree at most 3.** Let $v_1, \ldots, v_l$ ($l \leq 3$) be the set of all entering (or leaving) neighbors of v. The algorithm branches on selection of $v, v_1, \ldots, v_l$. In particular, $GetMAS(G,R)$ selects the largest set among $\{v\} \cup GetMAS(G^C(v), R)$, $\{v_1\} \cup GetMAS((G \setminus v)^C(v_1), R)$, $\ldots$, $\{v_l\} \cup GetMAS((G \setminus \{v, v_1, v_{l-1}\})^C(v_l), R)$.
- **[C42] The condition C41 is not satisfied** Select a vertex v *preferably* unmarked or weakly marked. Return the larger set of $\{v\} \cup GetMAS(G^C(v), R')$ and $GetMAS(G \setminus v, R)$ where $R' = R$ if $v \in VL(G,R) \cup VR(G,R)$ [§], otherwise R' is constructed as follows. Set the roles of all vertices of $G^C(v)$ to UM. Let $u_1, \ldots, u_4$ be any 4 entering neighbours of v and let $w_1, \ldots, w_4$ be any 4 leaving neighbours of G. These neighbours necessarily exist because the condition **C41** is not satisfied. Set the roles of $u_1 \ldots u_4$ in R' to be LM and the roles of $w_1, \ldots w_4$ to be RM. If v has entering neighbours other than $\{u_1, \ldots, u_4\}$ set their roles in R' to WLM. If there are leaving neighbours of v other than $\{w_1, \ldots w_4\}$, set their roles to WRM.

(5) **[C5] The conditions C1 to C4 are not satisfied.** We describe the case assuming that $|VL(G,R)| \leq |VR(G,R)|$. If $|VL(G,R)| > |VR(G,R)|$ then the behavior of $GetMAS(G,R)$ is symmetric with the difference that the vertices with roles RMD, RM, and WRM are con-

[§]In the complexity analysis, we show that the case where $v \in VL(G,R) \cup VR(G,R)$ never happens. We provide it here for the sake of completeness of the description.

sidered instead the vertices with roles LMD, LM, and WLM, respectively. As well, in the places where **entering** neighbors of **left-marked** vertices are mentioned, the **leaving neighbors** of the respective vertices are considered in the symmetric "right-marked" case. We consider three subcases of the given condition.

- **[C51] All the vertices of $VL(G,R)$ have role LMD or no vertex of $VL(G,R)$ has an unmarked entering neighbor.** ¶ Let v be an arbitrary vertex of $VL(G,R)$. Return the largest set among $\{v\} \cup GetMAS(G^C(v), R)$ and $GetMAS(G \setminus v, R)$.
- **[C52] There is a left-marked vertex v which does not belong to $V_{LMD}(G,R)$ and has at least 4 unmarked entering neighbors.** Let $v_1, \ldots, v_4$ be arbitrarily selected unmarked entering neighbors of v. Return the largest set among $\{v\} \cup GetMAS(G^C(v), R')$ and $GetMAS(G \setminus v, R)$ where R' differs from R in that the roles of $v_1, \ldots v_4$ are set to LM and the roles of the rest unmarked entering neighbors of v are set to WLM.
- **[C53] Conditions C51 and C52 are not satisfied.** Select a left-marked vertex v with the largest number of unmarked entering neighbors. Let X be the set of unmarked entering neighbors of v. For each acyclic subset $Y \subseteq X$, let $G_Y = (G \setminus (X \setminus Y))^C(Y)$ and return the largest set among $T_Y = Y \cup GetMAS(G_Y, R'(Y))$, where $R'(Y) = R$ if Y is non-empty. Otherwise, $R'(Y)$ is obtained from R by setting the role of v to LMD.

Theorem 3.1. *Given a directed graph G and a function R assigning roles to the vertices of G, $GetMAS(G,R)$ finds a MAS of G taking a finite time and polynomial space.*

Now we shall describe intuitively why the algorithm breaks the $O(2^n)$ barrier. The branching rules corresponding to conditions **C2**, **C3**, and **C41** result in *immediate* pruning effect. In particular, the branching rule corresponding to **C2** removes from the residual graph at least one additional vertex on the selection branch, the branching rule corresponding to **C3** removes 2 vertices on both branches, the branching rule corresponding to condition **C41** selects a subset of $k+1$ vertices ($k \leq 3$) to be included to the returned MAS, spending only $k+1$ branches instead of 2^{k+1} ones.

¶Again, this case is provided for the sake of completeness only. In the next section we rule out the possibility of its appearance.

The pruning effect of branching rules corresponding to conditions **C42** and **C52** is based on our setting that vertices with roles LM or RM have smaller weight than unmarked vertices (the weakly marked vertices have the same weight as unmarked). As a result of the selection branch of the considered branching rules, some unmarked vertices acquire roles LM or RM reducing the size of the problem which is the sum of weights of vertices of the underlying graph. To be useful in decreasing of the overall complexity, this weight reduction should be compensated by the *real* pruning effect occurring later on during the processing. The idea of the compensation is based on the invariant stating that for any pair (G, R) to which $GetMAS$ is applied recursively during the processing, each left-marked vertex is an entering neighbor of each right-marked vertex (we prove this invariant in the next section). To understand why it is helpful, consider a sequence of vertex selection branches, with the first branch corresponding to condition **C42** and the rest corresponding to condition **C52**. Each of these vertex selection branches increases the number of left-marked and right-marked vertices so, if the sequence is long enough, the underlying graph can be partitioned into the left-marked and right-marked vertices. From this moment and until one of the partition classes wipes out, either condition **C2** or **C3** is satisfied which results in a *real* pruning effect. Really if there is an edge from a right-marked vertex to a left-marked vertex then the above invariant guarantees that the underlying graph has a directed cycle of length 2 satisfying condition **C2**. Otherwise, left-marked and right-marked vertices belong to different strongly connected components, that is condition **C3** is satisfied.

The above strategy has two major obstacles. The first obstacle occurs if, for example, there are no left-marked vertices and there are many vertices with roles RM. The most undesired event in this situation is applying the branching rule corresponding to condition **C42**. On the vertex selection branch all the vertices with roles RM are unmarked. Since there are many such vertices, this results in massive increase of the problem size which neutralizes the effect of previous weight reductions. To avoid this obstacle, we apply the balancing operation which guarantees that the size of $VL(G, R)$ and $VR(G, R)$ differ by at most 3. However this balancing operation turns out to be helpless if there are many consecutive calls of the vertex selection branch corresponding to condition **C52** applied to *the same* side, say to the left-marked vertices. In this case the pruning effect of weight reduction might be diminished by the subsequent balancing operation. To avoid this undesired effect, the vertex selection branch is applied to the *smaller side*,

i.e. if $|VL(G,R)| \leq |VR(G,R)|$ then new left-marked vertices are created otherwise new right-marked ones appear. Combining this "alternating" application of the vertex selection branch with the balancing operation has the desired effect of avoiding the considered obstacle.

The second obstacle that may occur is satisfaction of condition **C53**. In this case the weight reduction produced by the vertex selection rule of condition **C52** is insufficient for the complexity improvement. To avoid this obstacle the algorithm performs an ordinary branching on all combinations of unmarked entering (or leaving) neighbors of the specified vertex v and changes the role of v to LMD or RMD on the branch where all the considered neighbors are removed. This results in weight reduction on that branch due to our setting that vertices with roles LMD or RMD have the lowest weight. We prove in the next section that if all the left-marked vertices of (G,R) have roles LMD (or all right-marked vertices have role RMD) then left-marked vertices and right-marked ones belong to different strongly connected components, i.e. condition **C3** is satisfied. This ensures that multiple application of the above branching rule eventually result in a *real* pruning effect.

4. Analysis

We start the analysis from introducing additional terminology. Let G_{IN} be the input graph whose MAS we are interested to compute. Let R_{IN} be the function assigning role UM to each vertex of G_{IN}. Recall that $GetMAS(G_{IN},R_{IN})$ is the initial application of the considered algorithm. The set of *legal pairs* explored by $GetMAS(G_{IN},R_{IN})$ includes (G_{IN},R_{IN}) and all pairs (G,R) to which $GetMAS$ is recursively applied during the run of $GetMAS(G_{IN},R_{IN})$.

Let (G',R') be a legal pair. Recall that the first operation performed by $GetMAS(G',R')$ is the balancing of (G',R') producing as a result the pair (G,R) for which $|VL(G,R)|$ and $|VR(G,R)|$ differ by at most 3 (if the condition is true regarding (G',R') then $(G',R') = (G,R)$). Then the appropriate type of recursive branching is selected regarding (G,R). We call (G,R) a *balanced pair* (BP). Note that $GetMAS(G',R') = GetMAS(G,R)$. If (G,R) satisfies condition **C1**, we call (G,R) an *atomic balanced pair*.

The set of BPs explored by $GetMAS(G_{IN},R_{IN})$ can be naturally represented as a search tree. The root of the tree is (G_{IN},R_{IN}) (this is a BP since all the vertices of G_{IN} are unmarked in R_{IN}). Let (G,R) be a node of the tree. If (G,R) is atomic then this node is a leaf. Otherwise, depending on the condition satisfied by (G,R), $GetMAS(G,R)$ produces legal pairs

$(G'_1, R'_1), \ldots, (G'_k, R'_k)$ to which $GetMAS$ is recursively applied. (For example, if (G, R) satisfies condition **C2** then the produced pairs are $(G^C(v), R')$ and $(G \setminus v, R)$, where v and R' are as shown in the description of the algorithm.) The pairs $(G_1, R_1), \ldots, (G_k, R_k)$ obtained as a result of balancing of $(G'_1, R'_1), \ldots, (G'_k, R'_k)$, respectively, are the *children* of (G, R). Accordingly, (G, R) is the *parent* of $(G_1, R_1), \ldots, (G_k, R_k)$. Now, we recursively define the notion of *descendants*. If (G, R) is atomic, it is the only descendant of itself. Otherwise, the set of descendants of (G, R) include (G, R) and the union of descendants of the children of (G, R). [‖] If a descendant of (G, R) is an atomic BP, we call it an *atomic descendant* of (G, R).

The crucial step of the analysis is ruling out the possibility of application of the branching rule corresponding to condition **C51**.

Theorem 4.1. *No BP (G, R) satisfies condition **C51** and causes $GetMAS(G, R)$ to check this condition.*

In order to prove the theorem, we need two additional lemmas.

Lemma 4.1. *For each BP, each left-marked vertex is an entering neighbor of each right-marked vertex.*

Lemma 4.2. *Let (G, R) be any BP such that both $VL(G, R)$ and $VR(G, R)$ are nonempty. Let $v \in V_{LMD}(G, R)$, let u be an entering neighbor of v such that u and v belong to the same strongly connected component. Then u is a left marked vertex of (G, R). Analogously, if $v \in V_{RMD}(G, R)$ and u is a leaving neighbor of v in the same strongly connected component then u is a right-marked vertex of (G, R).*

Proof of theorem 4.1. Assume by contradiction that there is a BP (G, R) that satisfies condition **C51** and causes $GetMAS(G, R)$ to check this condition. We assume that condition **C51** is satisfied regarding the left-marked vertices of (G, R), the case with right-marked vertices is symmetric. Since $GetMAS(G, R)$ checks condition **C51**, the earlier conditions **C1** ... **C4** are not satisfied regarding (G, R). That is, both $VL(G, R)$ and $VR(G, R)$ are nonempty and G is a strongly connected graph. In particular, there is a path in G from a right-marked vertex to a left-marked vertex. This path necessarily contains an edge (u, v) such that v is left-marked and u is not. If u is right-marked then by Lemma 4.1, u and v constitute a cycle

[‖] We admit that a node is a descendant of itself in order to ensure that an atomic node has exactly one descendant which will be convenient for the complexity computation.

of size 2, which satisfied condition **C2**, a contradiction. If u is unmarked then the second part of condition **C51** is not satisfied regarding left-marked vertices (v has an unmarked entering neighbor of u). It remains to assume that $v \in V_{LMD}(G,R)$ but this contradicts Lemma 4.2. □

Let w, wm, wmd be 3 real numbers so that $w > wm > wmd$. Let (G,R) be a BP. We assign each vertex v of G a weight $W_R(v)$ according to the role of v in (G,R). In particular, if $v \in V_{UM}(G,R) \cup V_{WLM}(G,R) \cup V_{WRM}(G,R)$ then $W_R(v) = w$. If $v \in V_{LM}(G,R) \cup V_{RM}(G,R)$ then $W_R(v) = wm$. Finally, if $v \in V_{LMD} \cup V_{RMD}(v)$ then $W_R(v) = wmd$. The weight $W_R(G)$ of G is the sum of weights of its vertices. The following theorem provides an upper bound on the number of atomic descendants of (G,R) depending on $W_R(G)$.

Theorem 4.2. *Let (G,R) be a BP. Assume that $w = 1, wm = 0.925, wmd = 0.884$. ** Let $m = W_R(G)$. Then the number of atomic descendants of (G,R) is at most 1.9977^m.*

Proof. The proof is by induction on the sequence of balanced pairs sorted in the reverse chronological order (we are allowed to consider this sequence due to the finite number of generated BPs as verified by Theorem 3.1). The first BP in this sequence is atomic, hence the theorem trivially holds for that pair (this pair is the only atomic descendant of itself). Consider a BP which is not the first in the sequence assuming that the theorem holds for all the previous pairs. Since the theorem trivially holds for all atomic BPs, we may assume that (G,R) is not atomic. In this case the number of atomic descendants of (G,R) is the sum of the numbers of atomic descendants of the children of (G,R). Let $(G_1,R_1),\ldots,(G_k,R_k)$ be the children of (G,R). Denote $W_{R_1}(G_1),\ldots,W_{R_k}(G_k)$ by $m_1,\ldots,m_k$, respectively. By the induction assumption, the number of atomic descendants of each (G_i,R_i) is at most 1.9977^{m_i}. We are going to show that $1.9977^{m_1} + \cdots + 1.9977^{m_k} \leq 1.9977^m$ which will finish the proof of the theorem. The rest of the proof follows all the conditions checked by the algorithm and shows that the theorem holds regarding (G,R) if it satisfies the given conditions. All these conditions are analyzed in a similar manner. Due to the space constraints, we cannot provide the analysis of all the conditions. Hence, we prove the analysis of the most non-trivial condition, which occurs when conditions **C1**, ..., **C4**

**The values are obtained by a computer program which guessed all the triplets of values from 0 to 1 with interval 0.001 with the objective to minimize c where c is the resulting constant, 1.9977 in considered case.

are not satisfied and condition **C52** is satisfied. According to Theorem 4.1, the condition **C51** cannot be satisfied in the considered case, hence the algorithm performs the operations associated with condition **C52**. Then (G, R) has two children (G_1, R_1) and (G_2, R_2) corresponding to selection and non-selection of the specified vertex v. Due to our agreement that $|VL(G, R)| \leq |VR(G, R)|$, v is a left-marked vertex in (G, R). Condition **C52** explicitly forbids v to have role LMD, hence v has role LM or WLM in (G, R). We prove these two subcases separately.

Assume first that $R(v) = LM$ The transformation from (G, R) removes v contributing wm to the decreasing of m_1. Next, 4 unmarked entering neighbors of v change their roles to LM which contributes $4(w - wm)$ to the decreasing of m_1. Observe that the subsequent balancing operation does not change roles of the vertices. Really, (G, R) is balanced and, by our agreement, $|VL(G, R)|$ is not greater than $|VR(G, R)|$. As a result of this transformation one strongly marked vertex (namely, v) is removed and four new ones appear. Clearly, the resulting difference between the number of strongly left-marked vertices and the strongly right-marked ones is not greater than 3. Thus $m_1 = m - wm - 4(w - wm) = m - 4w + 3wm$. The transformation from (G, R) to (G_2, R_2) removes v decreasing m_2 by wm. Since v is strongly marked, at most one vertex is made weakly marked by the subsequent balancing operation, which increases m_2 by at most $w - wmd$. In total $m_2 \leq m - wm + (w - wmd) = m - wm - wmd + w$. We obtain that $1.9977^{m_1} + 1.9977^{m_2} \leq 1.9977^{m-4w+3wm} + 1.9977^{m-wm-wmd+w} = 1.9977^m * (1.9977^{-4w+3wm} + 1.9977^{-wm-wmd+w}) < 1.9977^m * 0.9998$, getting the last inequality by the substitution of w, wm, and wmd with their values guessed by the statement of the theorem.

Assume now that $R(v) = WLM$. This time the removal of v and making 4 unmarked entering neighbors of v to have roles LM decreases m_1 by $w + 4(w - wm)$. However, the resulting number of strongly left-marked vertices may be greater by 4 than the number of strongly right-marked vertices (if initially $|VL(G, R)| = |VR(G, R)|$). Consequently, at most one strongly right-marked vertex can be made weakly marked by the subsequent balancing operation which increases m_1 by at most $w - wmd$. In total $m_1 \leq m - w - 4(w - wm) + w - wmd = m - 4w + 4wm - wmd$. The transformation from (G, R) to (G_2, R_2) removes v thus decreasing m_2 by w. Since w is a weakly marked vertex, its removal does not violate the difference between the number of strongly left-marked and strongly right-marked vertices in (G, R), hence the subsequent balancing operation does not change the roles of the vertices. That is, $m_2 = m - w$. Consequently,

$1.9977^{m_1} + 1.9977^{m_2} \leq 1.9977^{m-4w+4wm-wmd} + 1.9977^{m-w} = 1.9977^m * (1.9977^{-4w+4wm-wmd} + 1.9977^{-w}) < 1.9977^m * 0.968$. Thus we have verified that the theorem holds for both subcases of the considered case. □

Corollary 4.1. *There is an algorithm that finds the largest acyclic subset of the given graph G_{IN} in time $O^*(1.9977^n)$, where $n = |V(G_{IN})|$ and space polynomial in n.*

Acknowledgements

This work was supported by Science Foundation Ireland (Grant Number 05/IN/I886).

I would like to thank the reviewers for their very useful comments that allowed me to essentially improve the presentation of the final version of the paper.

References

1. D. Eppstein. Quasiconvex analysis of backtracking algorithms. In *SODA*, pages 788–797, 2004.
2. David Eppstein. Improved algorithms for 3-coloring, 3-edge-coloring, and constraint satisfaction. In *SODA*, pages 329–337, 2001.
3. F. Fomin, S. Gaspers, and A. Pyatkin. Finding a Minimum Feedback Vertex Set in time $O(1.7548^n)$. In *IWPEC 2006*, pages 184–191, 2006.
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 $O(2^{0.288})$ independent set algorithm. In *SODA*, pages 18–25, 2006.
6. F. Fomin, F. Grandoni, and D. Kratsch. Solving Connected Dominating Set faster than 2^n. In *FSTTCS*, pages 152–163, 2006.
7. F. Fomin, Fabrizio Grandoni, and Dieter Kratsch. Measure and conquer: Domination - a case study. In *ICALP*, pages 191–203, 2005.
8. M. Held nad R. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, 10:196–210, 1962.
9. I. Razgon. Exact Computation of Maximum Induced Forest. In *SWAT 2006, LNCS 4059*, pages 160–171, 2006.
10. G. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization*, pages 185–208, 2001.