Exact computation of maximum induced forest

Igor Razgon *

Computer Science Department, University College Cork, Ireland
i.razgon@cs.ucc.ie

Abstract. We propose a backtrack algorithm that solves a generalized version of the Maximum Induced Forest problem (MIF) in time $O^*(1.8899^n)$. The MIF problem is complementary to finding a minimum Feedback Vertex Set (FVS), a well-known intractable problem. Therefore the proposed algorithm can find a minimum FVS as well. To the best of our knowledge, this is the first algorithm that breaks the $O^*(2^n)$ barrier for the general case of FVS. Doing the analysis, we apply a more sophisticated measure of the problem size than the number of nodes of the underlying graph.

1 Introduction

Exact exponential algorithms are techniques for solving intractable problems with better complexity than trivial brute-force exploring of all the possible combinations. Examples of such algorithms include: [9] for maximum independent set, [1] for chromatic number, [3] for 3-COLORABILITY, [2] for 3-SAT, [4] for dominating set, and others. A recent overview of exact algorithms is provided in [10].

In this paper we propose an $O^*(1.8899^n)$ exact algorithm for solving the following problem. Given a graph G and a subset K of its vertices, find a largest superset S of K such that the subgraph of G induced by S is acyclic. If $K = \emptyset$ then S is a Maximum Induced Forest (MIF) of G. The complement of S, $V(G)\setminus S$, is a minimum Feedback Vertex Set (FVS) of G, i.e. a set of vertices that participate in all the cycles of G. Computing a minimum FVS is a "canonical" intractable optimization problem, whose NP-complete version is mentioned in [7]. To the best of our knowledge, the proposed algorithm is the first that breaks the $O^*(2^n)$ barrier for the general case of FVS. Previous studies [6, 8] describe exact algorithms only for special cases of FVS.

The proposed algorithm computes MIF using the "branch-and-prune" strategy ([10], Section 4). Using this strategy for computing of MIF is not straightforward. The reason is that selection of a new vertex for MIF does not necessarily cause additional pruning: a vertex can be pruned only if it induces a cycle with the "already selected" vertices. For graphs with a large girth, many vertices must be selected before at least one can be discarded.

* I would like to thank Fedor Fomin, who inspired me to investigate the problem, and an anonymous reviewer who suggested me a way of improving the result reported in the first version of the paper.

To overcome this difficulty, we analyze complexity of the algorithm by applying a more sophisticated measure than the number of vertices of the residual graph, a strategy suggested in [5]. In particular, we observe that all the "remaining" vertices can be partitioned into the vertices that have neighbours with the already selected vertices and the vertices that do not have them. We associate the vertices of the former class with weight 1 and the vertices of the latter class with weight 1.565, a constant guessed by a computational procedure. The proposed measure is the sum of weights of all the vertices of the residual graph. Further analysis yields an upper bound $O^*(1.50189^m)$, where m is the value of the applied measure for the input graph G. We then demonstrate that $m \leq 1.565n$, where n is the number of vertices of G, which results in an upper bound of $O^*((1.50189^{1.565})^n)$. Taking into account that $1.8898 < 1.50189^{1.565} < 1.8899$, this bound is transformed to $O^*(1.8899^n)$ by rounding.

The rest of the paper is organized as follows. Section 2 introduces the necessary terminology. Section 3 presents the proposed algorithm. Section 4 proves correctness of the algorithm and provides complexity analysis [1].

2 Preliminaries

A simple undirected graph is referred in this paper as *a graph*. A set of vertices of a graph G is denoted by $V(G)$. Given $S \subseteq V(G)$, we denote by $G[S]$ the subgraph of G induced by S and by $G \setminus S$ the subgraph of G induced by $V(G) \setminus S$. If S consists of a single vertex v, we write $G \setminus v$ rather than $G \setminus \{v\}$. Two vertices v and w of G are *S-connected* if they are adjacent or if there is a path $v, p_1, \ldots, p_m, w$, where $\{p_1, \ldots p_m\} \subseteq S$.

The set S is a maximum induced forest (MIF) if $G[S]$ is acyclic and S is the largest set subject to this property [2]. In addition, we introduce the notion of a T-MIF.

Definition 1. *Let $T \subseteq V(G)$. A T-MIF of G is a largest superset S of T such that $G[S]$ is acyclic.*

Clearly, the definition makes sense only when $G[T]$ is acyclic. Observe that a $\emptyset$-MIF of G is an ordinary MIF.

To present complexity of algorithms, we use the O^* notation [10], which suppresses the polynomial factor. For example, $O(n^2 2^n)$ is written as $O^*(2^n)$.

3 The algorithm

In this section we present an algorithm for computing a MIF of a given graph. We start with extending our notation.

Let G be a graph and let T be a subset of its vertices. We recognize the following classes of vertices of $G \setminus T$.

[1] Due to space constraints, proofs of some technical lemmas are omitted.

[2] It is more convenient for us to represent a MIF as a set of vertices rather than a subgraph of G.

- *Boundary vertices* denoted by $Bnd(G,T)$. The set $Bnd(G,T)$ contains all vertices $v \in V(G \setminus T)$ such that v is adjacent to exactly one vertex of T.
- *Conflicting vertices* denoted by $Cnf(G,T)$. The set $Cnf(G,T)$ contains all vertices $v \in V(G \setminus T)$ such that v is adjacent to at least two vertices of the same connected component of $G[T]$.
- *Free vertices* denoted by $Free(G,T)$. The set $Free(G,T)$ contains all vertices $v \in V(G \setminus T)$ such that v is not adjacent to any vertex of T.

Now we are ready to introduce the algorithm $Main\_MIF$ (Algorithm 1). It gets as input a graph G and a subset K of $V(G)$. The algorithm returns (as we will prove further) a K-MIF of G. Clearly, setting K to $\emptyset$ will make the algorithm to return a MIF of G.

The algorithm $Main\_MIF$ starts with checking whether $G[K]$ is acyclic. If not, $FAIL$ is returned immediately (line 1 of Algorithm 1) because no K-MIF of G exists in this case. Otherwise, the function $Find\_MIF$ runs (line 2 of Algorithm 1).

Function $Find\_MIF$ is the main "search engine" of $Main\_MIF$. It is described in lines 3-31 of Algorithm 1. The function gets as input a graph G_1, and subsets T_1 and K_1 of $V(G_1)$. The function is supposed to return a $T_1 \cup K_1$-MIF of G_1 (provided that $G_1[T_1 \cup K_1]$ is acyclic).

If $T_1 \cup K_1 = V(G_1)$, $Find\_MIF$ returns $T_1 \cup K_1$ (lines 4 and 5 of Algorithm 1). Otherwise, the execution can be divided into four stages: selecting a vertex v of $V(G_1) \setminus (T_1 \cup K_1)$, a recursive call processing the case when v is added to $T_1 \cup K_1$, a recursive call processing the case when v is eliminated from G_1, and returning the maximum-size set among the ones returned by the above two recursive calls.

Selection of a vertex v is described in lines 7-11 of Algorithm 1. The vertex is taken from $Bnd(G_1,T_1)$ unless the set is empty. In this case, the function selects an arbitrary vertex that does not belong to $T_1 \cup K_1$.

Having selected a vertex v, the function adds it to $T_1 \cup K_1$ (line 12 of Algorithm 1). The addition is performed by function T_Update (Algorithm 2). Applying of T_Update in line 12 returns a triplet (G_2, T_2, K_2), in which $T_2 \cup K_2 = T_1 \cup K_1 \cup \{v\}$, v itself and all vertices of K_1 that are K_1-connected to v are "moved" to T_2, G_2 is obtained from G_1 by removing all vertices of $Cnf(G_1,T_2)$ because every one of them induces cycles being added to T_2. Function $Find\_MIF$ is applied recursively to (G_2,T_2,K_2) in line 13 and returns a set S_2.

The way a vertex v is selected and then added to $T_2 \cup K_2$ ensures that the inputs (G',T',K') of all recursive applications of $FindIndep$ have a number of invariant properties which are crucial for our analysis (Section 4). Two most important properties are that any connected component of G' contains at most one connected component of $G'[T']$, and that there are no edges between vertices of T' and vertices of K'.

Processing the case, where v is eliminated from G_1 (lines 14-29 of Algorithm 1), depends on the number of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v. If there is at most one such vertex, the function decides that

S_2 is a $T_1 \cup K_1$-MIF of G_1 and returns it (lines 15-16 of Algorithm 1). The case when there are exactly 2 such vertices is processed in lines 17-25. The set W of these two vertices is added to $T_1 \cup K_1$ by function K_Update (Algorithm 3). This function returns $FAIL$ if $G_1[T_1 \cup K_1 \cup W]$ contains cycles. In this case, $Find\_MIF$ returns S_2. Otherwise, $Find\_MIF$ is applied recursively to the triplet returned by K_Update, returns a set S_3, and the largest set among S_2 and S_3 is returned in line 22. If the number of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v is at least 3, $Find\_MIF$ returns the largest set among S_2 and S_3, where S_3 is returned by the recursive application of $Find\_MIF$ to $(G_1 \setminus v, T_1, K_1)$ (lines 26-28 of Algorithm 1).

Consider the intuition behind the decisions made by the algorithm in lines 15-25. For this purpose, assume that $K_1 = \emptyset$. That is, we consider the cases where v have 1 or 2 neighbours that are not in T_1. In the former case, let $w \in V(G_1) \setminus T_1$ be the considered neighbour of v. Observe that it is safe to add v to T_2. Really, any T_1-MIF S of G_1 that does not contain v has to contain w (otherwise, we get contradiction to the maximality of S). In this case replacing w by v in S, we get another T_1-MIF of G_1.

Assume now that v is adjacent to vertices $w_1, w_2 \in V(G_1) \setminus T_1$ and that v does not belong to any T_1-MIF of G_1. Then, any T_1-MIF of G_1 contains both w_1 and w_2, otherwise, arguing as for the previous case, we get a contradiction with our assumption. A subtle question is where to add w_1 and w_2. The point is that the invariant property that every component of G_1 contains at most one component of $G_1[T_1]$ should not be violated for the inputs of the subsequent recursive calls of the $Find\_MIF$ function. To satisfy this requirement, for example, when none of w_1 and w_2 have neighbours in T_1, the function K_Update adds them to K_2, not to T_2. That is, even if $K = \emptyset$ in the original input, it can be transformed to a non-empty set in one of subsequent recursive calls of $Find\_MIF$. Thus the necessity to handle the case when v is adjacent to exactly two "remaining" vertices is what caused the author to consider a generalized version of the MIF-problem.

4 Analysis

The analysis of the $Main\_MIF$ algorithm is organized as follows. In Section 4.1 we introduce the notion of a *Fair Configuration* (FC) and prove a number of properties of FCs. In Section 4.2 we define a search tree generated by function $Find\_MIF$ with the nodes corresponding to the inputs of the recursive calls of $Find\_MIF$. We prove that all these inputs are FCs. Then, based on properties of FCs, we prove correctness of $Main\_MIF$ (section 4.3) and analyze its complexity (section 4.4). Due to space constraints, proofs of some technical lemmas are omitted.

4.1 Fair configurations and their properties

Definition 2. *Let G be a graph and let T and K be subsets of $V(G)$. A triplet (G, T, K) is a Fair Configuration (FC) if the following conditions hold:*

Algorithm 1 $Main\_MIF(G, K)$

1: **if** $G[K]$ contains cycles **then** Return $FAIL$
2: Return $Find\_MIF(G \setminus Cnf(G, K), \emptyset, K)$
3: **function** $Find\_MIF(G_1, T_1, K_1)$
4: **if** $T_1 \cup K_1 = V(G_1)$ **then**
5: Return $T_1 \cup K_1$
6: **else**
7: **if** $Bnd(G_1, T_1)$ is not empty **then**
8: Select a vertex $v \in Bnd(G_1, T_1)$
9: **else**
10: Select an arbitrary vertex $v \in G_1 \setminus (T_1 \cup K_1)$
11: **end if**
12: $(G_2, T_2, K_2) \leftarrow T\_Update(G_1, T_1, K_1, v)$
13: $S_2 \leftarrow Find\_MIF(G_2, T_2, K_2)$
14: **switch** The number of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v
15: **case** ≤ 1
16: Return S_2
17: **case** 2
18: Let W be the set of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v
19: **if** $K\_Update(G_1 \setminus v, T_1, K_1, W)$ does not return $FAIL$ **then**
20: $(G_3, T_3, K_3) \leftarrow K\_Update(G_1 \setminus v, T_1, K_1, W)$
21: $S_3 \leftarrow Find\_MIF(G_3, T_3, K_3)$
22: Return the largest set of S_2 and S_3
23: **else**
24: Return S_2
25: **endif**
26: **case** ≥ 3
27: $S_3 \leftarrow Find\_MIF(G_1 \setminus v, T_1, K_1)$
28: Return the largest set of S_2 and S_3
29: **end switch**
30: **end if**
31: **end function**

Algorithm 2 $T\_Update(G, T, K, v)$

1: Let S be the subset of vertices of K that are K-connected to v
2: $T' \leftarrow T \cup \{v\} \cup S$
3: $K' \leftarrow K \setminus (\{v\} \cup S)$
4: $G' \leftarrow G \setminus Cnf(G, T')$
5: Return (G', K', T')

Algorithm 3 $K\_Update(G, T, K, W)$

1: **if** $G[T \cup K \cup W]$ contains cycles **then** Return $FAIL$
2: **if** $W \cap Bnd(G, T) = \emptyset$ **then**
3: Return $(G \setminus Cnf(G, K \cup W), T, K \cup W)$
4: **else**
5: Let v_1 be a vertex of W that belongs to $Bnd(G, T)$
6: $(G', T', K') \leftarrow T\_Update(G, T, K, v_1)$
7: $\{v_2\} \leftarrow W \setminus \{v_1\}$
8: **if** $v_2 \in Bnd(G', T')$ **then**
9: Return $T\_Update(G', T', K', v_2)$
10: **else**
11: Return $(G' \setminus Cnf(G', K' \cup \{v_2\}), T', K' \cup \{v_2\})$
12: **end if**
13: **end if**

- $K \subseteq Free(G, T)$;
- $G[T]$ *and* $G[K]$ *are acyclic;*
- $Cnf(G, T) = Cnf(G, K) = \emptyset$;
- *every connected component of* G *contains at most one connected component of* $G[T]$.

Due to importance of the notion for the proposed analysis, we demonstrate it on an example.

Let G be the graph shown in Figure 1, the black circles represent the vertices of T, the crossed circles represent the vertices of K, the other vertices are represented by the white circles. Observe that (G, T, K) is a FC. Indeed, there are no edges between the vertices of T and K, both T and K induce acyclic subgraphs of G, no single vertex of $V(G) \setminus (T \cup K)$ makes cycles with T and K. Finally, every connected component of G contains at most one connected component of $G[T]$. Note that by definition of a FC, the last requirement is not necessary for $G[K]$. In our example, the component induced by vertices v_9 to v_{12} contains two components of $G[K]$.

Lemma 1. *Let* (G, T, K) *be a FC. Then* $T \cup Bnd(G, T) \cup Free(G, T) = V(G)$. [3]

Lemma 2. *Let* (G, T, K) *be a FC and let* $v \in V(G) \setminus (T \cup K)$. *Assume that one of the following properties holds:*

- $v \in Bnd(G, T)$;
- $Bnd(G, T) = \emptyset$.

Then $(G', T', K') = T\_Update(G, T, K, v)$ *is a FC.*

[3] In other words, any vertex of $V(G) \setminus T$ is adjacent to at most one vertex of T.

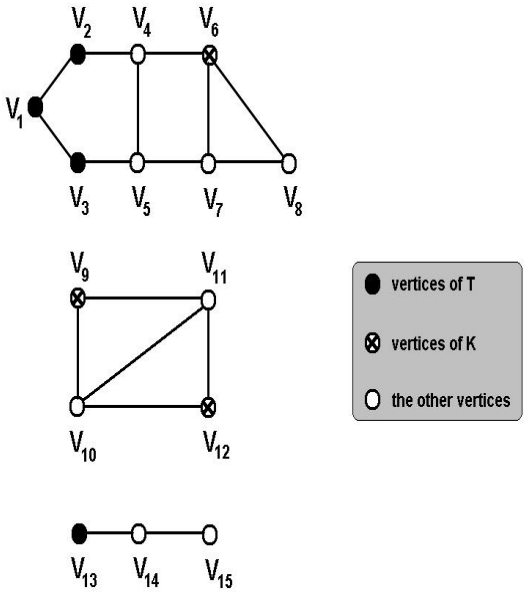

Fig. 1. A Fair Configuration (FC)

As a result of application of T_Update, some vertices change their "roles". This statement is described precisely in the following lemma.

Lemma 3. *Let (G, T, K) be a FC and let $v \in V(G)$ be a vertex such that $(G', T', K') = T\_Update(G, T, K, v)$ is a FC.*

Let $w \in V(G) \setminus (T \cup K \cup \{v\})$ be a vertex, which is K-connected to v. Then

- *$w \notin Free(G', T')$;*
- *in addition, if $w \in Bnd(G, T)$ then $w \notin V(G')$.*

Lemma 4. *Let (G, T, K) be a FC. Let $S \subseteq V(G) \setminus (T \cup K)$ and $W \subseteq Free(G, T) \setminus K$ be two disjoint sets. Then $(G \setminus S \setminus Cnf(G \setminus S, K \cup W), T, K \cup W)$ is a FC.*

4.2 The search tree *ST*

In this section we define a *search tree* ST explored by $Main\_MIF$. The root of the tree is associated with a triplet $(G \setminus Cnf(G, K), \emptyset, K)$, where G and K constitute the input of $Main\_MIF$. Assume that a node x of ST is associated with a triplet (G_1, T_1, K_1). The structure of the subtree rooted by x depends on the execution of $Find\_MIF(G_1, T_1, K_1)$. If $T_1 \cup K_1 = V(G_1)$ then x is a leaf. Otherwise, x has a child associated with the triplet returned by $T\_Update(G_1, T_1, K_1, v)$, where v is the vertex selected by $Find\_MIF(G_1, T_1, K_1)$ in lines 7-11 of Algorithm 1. It is the only child if $Find\_MIF(G_1, T_1, K_1)$ executes line 16 or line 24. If line 22 is executed then x has the additional child associated with

$K\_Update(G_1 \setminus v, T_1, K_1, \{v_1, v_2\})$; if line 28 is executed, the additional child is associated with the triplet $(G_1 \setminus v, T_1, K_1)$.

Lemma 5. *ST is of finite size.*

Lemma 6. *The triplet associated with every node of ST is a FC.*

4.3 Correctness Proof.

In this section we will prove correctness of $Main\_MIF$ by demonstrating that $Main\_MIF(G, K)$ returns a K-MIF of G.

Lemma 7. *Let (G, T, K) be a FC and let $v \in V(G) \setminus (T \cup K)$. Let S be a $T \cup K$-MIF of G. Then either $v \in S$ or any cycle in $G[S \cup \{v\}]$ involves a vertex $w \in V(G) \setminus (T \cup K \cup \{v\})$, which is K-connected to v.*

Proof. Assume that $v \notin S$ and let $v, v_1, \ldots v_m$ be a cycle of $G[S \cup \{v\}]$ (clearly, v participates in any cycle of $G[S \cup \{v\}]$ because $G[S]$ is acyclic). If either v_1 or v_m belongs to $V(G) \setminus (T \cup K)$, we are done. Otherwise, note that $\{v_1, v_m\} \not\subseteq T$ because v cannot be connected to more than one vertex of T, by Lemma 1. It follows also that $\{v_1, \ldots, v_m\} \not\subseteq K$ because the opposite would mean that $v \in Cnf(G, K)$. Assume without loss of generality that $v_1 \in K$. Let i be the smallest index such that $v_i \notin K$, while $v_{i-1} \in K$. Note that by definition of a FC $v_i \notin T$ (because existence of an edge between K and T would follow otherwise). Thus $v_i \in V(G) \setminus (T \cup K)$ and the path $v, v_1, \ldots, v_i$, all intermediate vertices of which belong to K, certifies that v_i is K-connected to v. ■

Lemma 8. *Let G be a graph, $T \subset V(G)$ such that $G[T]$ is acyclic, and $U \subseteq V(G) \setminus T$. Assume that no T-MIF of G intersects with U. Then any T-MIF of $G \setminus U$ is a T-MIF of G.*

Lemma 9. *Let (G, T, K) be a FC and let $v \in V(G) \setminus (T \cup K)$. Let $(G', T', K') = T\_Update(G, T, K, v)$. Assume that at least one $T \cup K$-MIF of G contains v. Then any $T' \cup K'$-MIF of G' is a $T \cup K$-MIF of G.*

Lemma 10. *Let (G, T, K) be a FC. Let $v \in V(G) \setminus (T \cup K)$ be a vertex, which is K-connected to at most one vertex of $G \setminus (T \cup K \cup \{v\})$. Let $(G', T', K') = T\_Update(G, T, K, v)$. Then any $T' \cup K'$-MIF of G' is a $T \cup K$-MIF of G.*

Proof. If at least one $T \cup K$-MIF of G contains v, the statement follows from Lemma 9. Otherwise, let S be a $T \cup K$-MIF of G. By Lemma 7, every cycle of $G[S \cup \{v\}]$ contains a vertex of $V(G) \setminus (T \cup K \cup \{v\})$, which is K-connected to v. By the condition of the lemma, there is at most one such a vertex, say, w. Therefore w participates in all the cycles of $G[S \cup \{v\}]$, and removing of w breaks all the cycles. Clearly, $S \cup \{v\} \setminus \{w\}$ is a $T \cup K$-MIF of G containing v, in contradiction to our assumption. ■

Lemma 11. *Let (G, T, K) be a FC. Let $v \in V(G) \setminus (T \cup K)$ be a vertex, which is K-connected to exactly two vertices v_1 and v_2 of $G \setminus (T \cup K \cup \{v\})$. Then at least one of the following two statements is true.*

- *Let $(G_1, T_1, K_1) = T\_Update(G, T, K, v)$. Then any $T_1 \cup K_1$-MIF of G_1 is a $T \cup K$-MIF of G.*
- *$K\_Update(G \setminus v, T, K, \{v_1, v_2\})$ does not return $FAIL$. Moreover, let $(G_2, T_2, K_2) = K\_Update(G \setminus \{v\}, T, K, \{v_1, v_2\})$. Then any $T_2 \cup K_2$-MIF of G_2 is a $T \cup K$-MIF of G.*

Proof. Assume that the first statement does not hold. By Lemma 9, no $T \cup K$-MIF of G contains v. Let S be a $T \cup K$-MIF of G. According to Lemma 7, any cycle in $G[S \cup \{v\}]$ involves a vertex of $G \setminus (T \cup K \cup \{v\})$, which is K-connected to v. If S contains only one such vertex, say, v_1, then removing v_1 breaks all the cycles and $S \cup \{v\} \setminus \{v_1\}$ is a $T \cup K$-MIF of G in contradiction to our assumption. It follows that $\{v_1, v_2\} \subseteq S$. Clearly, S is a $T \cup K \cup \{v_1, v_2\}$-MIF of G because otherwise we get a contradiction with being S a $T \cup K$-MIF of G. It follows that any $T \cup K \cup \{v_1, v_2\}$-MIF of G is a $T \cup K$-MIF of G. Then, by Lemma 8, any $T \cup K \cup \{v_1, v_2\}$-MIF of $G \setminus v$ is a $T \cup K$-MIF of G. Consequently $(G \setminus v)[T \cup K \cup \{v_1, v_2\}]$ is acyclic and hence $K\_Update(G, T, K, \{v_1, v_2\})$ does not return $FAIL$. Furthermore, it follows from the description of K_Update that $T_2 \cup K_2 = T \cup K \cup \{v_1, v_2\}$ and G_2 is obtained from $G \setminus v$ by removing vertices that make cycles with $T \cup K \cup \{v_1, v_2\}$ in $G \setminus v$. Thus, any $T_2 \cup K_2$-MIF of G_2 is a $T \cup K$-MIF of G by Lemma 8 and the above reasoning. ■

Theorem 1. *For any triplet (G_1, T_1, K_1) associated with a node of ST, $Find\_MIF(G_1, T_1, K_1)$ returns a $T_1 \cup K_1$-MIF of G_1.*

Proof. Let $x_1, x_2, \ldots$ be an order of nodes of ST such that children are ordered before their parents; existence of such an order follows from Lemma 5. The proof is by induction on the sequence. We also use the fact that the triplet associated with every node x of ST is a FC (Lemma 6).

Clearly, the statement holds for all leaves of ST and, in particular, for x_1. Consider a non-leaf node x_i, assuming validity of the theorem for all nodes placed before, and denote the FC associated with x_i by (G_1, T_1, K_1). The FCs associated with the children of x_i are exactly the inputs of the recursive calls performed by $Find\_MIF(G_1, T_1, K_1)$. By the induction assumption, these recursive calls work properly.

If the vertex v picked by $Find\_MIF(G_1, T_1, K_1)$ is K_1-connected to exactly one vertex of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$, the correctness follows from Lemma 10; in the case of two vertices, the correctness follows from Lemma 11; in the case of three or more vertices, the correctness follows from Lemmas 8 and 9. ■

Corollary 1. *Let G be a graph and $K \subseteq V(G)$. If $G[K]$ is acyclic, $Main\_MIF(G, K)$ returns a K-MIF of G.*

4.4 Complexity analysis

In this section we analyse the complexity of $Main\_MIF$ by deriving the upper bound on the number of nodes of ST. For the complexity analysis, we associate with every node x of ST the measure $Y(x) = c|Free(G_1, T_1) \setminus K_1| + |Bnd(G_1, T_1)|$, where (G_1, T_1, K_1) is the triplet associated with x, $c = 1.565$. In other words, the elements of $Free(G_1, T_1) \setminus K_1$ are assigned with weight c, the elements of $Bnd(G_1, T_1)$ are assigned with weight 1, $Y(x)$ is the sum of all the weights. The complexity analysis is structured as follows. For a given two nodes x and z such that x is the parent of z, we evaluate $Y(x) - Y(z)$. Based on the evaluation, we obtain an upper bound on the number of nodes of the subtree rooted at x. This upper bound is $O^*(\alpha(c)^{Y(x)})$, where $\alpha(c)$ is the constant depending on c. For $c = 1.565$, $\alpha(c) = 1.50189$. Then we notice that for the root node r, the value of $Y(r)$ is at most cn, where n is the number of vertices of the original graph. Thus we obtain the upper bound $O^*(1.50189^{1.565n})$. Taking into account that $1.8898 < 1.50189^{1.565} < 1.8899$, the upper bound obtained after rounding the base of the exponent is $O^*(1.8899^n)$.

The constant $c = 1.565$ was guessed by a binary search computational procedure that explored the range from 1.005 to 2 by steps of 0.005 and for every considered constant c computed $\alpha(c)^c$. The smallest value of this expression was obtained for $c = 1.565$.

Lemma 12. *Let x be a non-leaf node of ST associated with a triplet (G_1, T_1, K_1), let a node z associated with a triplet (G_2, T_2, K_2) be a child of x, and let $v \in V(G_1) \setminus (T_1 \cup K_1)$. Then the following statements hold.*

1. *If $(G_2, T_2, K_2) = T\_Update(G_1, T_1, K_1, v)$ then $Y(z) \leq Y(x) - ((c-1)|W| + 1)$, where W is the set of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v.*
2. *If $(G_2, T_2, K_2) = K\_Update(G_1 \setminus v, T_1, K_1, \{v_1, v_2\})$, where $\{v_1, v_2\} \subseteq V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$, then $Y(z) \leq Y(x) - 3$.*
3. *If $(G_2, T_2, K_2) = (G_1 \setminus v, T_1, K_1)$ then $Y(z) \leq Y(x) - 1$.*

Proof. Let z be a child of x associated with (G_2, T_2, K_2) and assume that $(G_2, T_2, K_2) = T\_Update(G_1, T_1, K_1, v)$. Recall that all the triplets associated with the nodes of ST are FCs (Lemma 6). By definition of W and Lemma 1, the vertices of W can be partitioned into two subsets, $W_1 \subseteq Free(G_1, T_1) \setminus K_1$ and $W_2 \subseteq Bnd(G_1, T_1)$.

Observe that $|Free(G_2, T_2) \setminus K_2| \leq |Free(G_1, T_1) \setminus K_1| - |W_1|$. Really, $Free(G_2, T_2) \setminus K_2$ are the vertices of $V(G_2) \setminus (T_2 \cup K_2)$ that do not have neighbors in T_2. Taking into account that $T_1 \subset T_2$, $T_1 \cup K_1 \subset T_2 \cup K_2$, and $V(G_2) \subseteq V(G_1)$, it is clear that $Free(G_2, T_2) \setminus K_2 \subseteq Free(G_1, T_1) \setminus K_1$. Further, applying Lemma 3, we obtain that $Free(G_2, T_2) \setminus K_2 \subseteq (Free(G_1, T_1) \setminus K_1) \setminus W_1$. Considering that $W_1 \subseteq Free(G_1, T_1) \setminus K_1$, we get the desired inequality. A vertex of W_1 can be either removed from G_2 or added to $Bnd(G_2, T_2)$. In the former case the value of $Y(z)$ is decreased by c with respect to $Y(x)$, in the second case $Y(z)$ is decreased only by $c - 1$, because the weight of the vertex is changed from c to

1. We evaluate the maximal possible weight of $Y(z)$, hence we can assume that all vertices of W_1 are moved to $Bnd(G_2, T_2)$, decreasing $Y(z)$ by $(c-1)|W_1|$.

The vertices of W_2 are removed from G_2 by Lemma 3 and vertex v is moved from $Bnd(G_1, T_1)$ to T_2. These transformations decrease the value of $Y(z)$ with respect to $Y(x)$ by $|W_2|+1$. Combining the above argumentation, and taking into account that $c = 1.565$, we see that $Y(z) \leq Y(x)-((c-1)*|W_1|+|W_2|+1) \leq Y(x) - ((c-1)|W|+1)$, proving the first statement.

If the condition of the second statement holds then $|Free(G_2, T_2) \setminus K_2| + |Bnd(G_2, T_2)| \leq |Free(G_1, T_1) \setminus K_1| + |Bnd(G_1, T_1)| - 3$ because v is removed from G_2, v_1 and v_2 are moved to $T_2 \cup K_2$. Removing of anyone of these vertices decreases $Y(z)$ by at least 1. The second statement immediately follows.

The last statement is immediate when we observe that $|Free(G_2, T_2) \setminus K_2| + |Bnd(G_2, T_2)| \leq |Free(G_1, T_1) \setminus K_1| + |Bnd(G_1, T_1)| - 1$ if the condition of the last statement holds. ■

Let m be an integer such that there is a node x of ST with $Y(x) = m$. We denote by $F(m)$ the maximum possible number of nodes of the subtree rooted at x.

Lemma 13. *For any node x of ST, $F(Y(x))$ is bounded by $O^*(1.50189^{Y(x)})$.*

Proof. Let (G_1, T_1, K_1) be the triplet associated with x. Recall that $Y(x) = c|Free(G_1, T_1) \setminus K_1| + |Bnd(G_1, T_1)|$. Clearly, $Y(x) \geq 0$.

Assume that $Y(x) = 0$. It is only possible when $V(G_1) = T_1 \cup K_1$. According to Algorithm 1, x is a leaf, hence the lemma holds for this case.

Assume now that $Y(x) > 0$. Clearly, x is a non-leaf. If x has only one child z then z is necessarily associated with $T\_Update(G_1, T_1, K_1, v)$ for some $v \in V(G_1) \setminus (T_1 \cup K_1)$. By Lemma 12, $Y(z) \leq Y(x) - 1$ (the equality holds when v is not K_1-connected to any vertex of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$). In this case, $F(Y(x)) = F(Y(z)) + 1 = F(Y(x) - l) + 1$, where $l \geq 1$.

If x has two children, z_1 and z_2, one of them, say z_1, is necessary associated with the triplet returned by $T\_Update(G_1, T_1, K_1, v)$. The node z_2 is associated either with $K\_Update(G_1 \setminus v, T_1, K_1, \{v_1, v_2\})$ or with $(G_1 \setminus v, T_1, K_1)$.

In the former case, $\{v_1, v_2\}$ is the set of vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$ that are K_1-connected to v. By the first part of Lemma 12, $Y(z_1) \leq Y(x) - (2(c-1)+1)$, by the second part of the same lemma, $Y(z_2) \leq Y(x) - 3$. Substituting $c = 1.565$, we obtain $F(Y(x)) = F(Y(z_1)) + F(Y(z_2)) + 1 = F(Y(x) - l_1) + F(Y(x) - l_2) + 1$, where $l_1 \geq 2.13$, $l_2 \geq 3$.

In the latter case, it follows from the description of Algorithm 1 that v is K_1-connected to at least 3 vertices of $V(G_1) \setminus (T_1 \cup K_1 \cup \{v\})$. Consequently, $Y(z_1) \leq Y(x) - (3(c-1)+1)$ and $Y(z_2) \leq Y(x) - 1$ by the first and the last parts of Lemma 12. Arguing as for the previous two cases, we obtain $F(Y(x)) = F((Y(x) - l_1) + F(Y(x) - l_2) + 1$, where $l_1 \geq 2.695$, $l_2 \geq 1$.

The last recursive relation for $F(Y(x))$ yields the worst upper bound. Taking into account that the upper bound is exponential, we can ignore the additive constant because it contributes only a polynomial factor to the resulting bound. The

upper bound following from the expression $F(Y(x)) = F(Y(x)-1) + F(Y(x)-2.695)$ is $O^*(\beta^{Y(x)})$, where β is the largest root of the equation $\beta^{2.695} = \beta^{1.695}+1$. A simple computation shows that $1.50188 < \beta < 1.50189$. ■

Theorem 2. *The $Main\_MIF$ algorithm, applied to a graph G with n vertices, takes $O^*(1.8899^n)$ time and a polynomial space.*

Proof. Let x be the root node of ST. It is associated with the triplet $(G, \emptyset, K)$. Then $Y(x) = c|Free(G,\emptyset)\backslash K| + |Bnd(G,\emptyset)| = c|Free(G,\emptyset)\backslash K| \leq cn$, where $c = 1.565$. It follows that ST has $O^*(1.50189^{1.565n}) < O^*(1.8899^n)$ nodes. The upper bound on the time-complexity of $Main\_MIF$ can be obtained by summing up the bounds on the processing time spent to every node of ST. Observe that processing of a node includes all the operations performed by $Find\_MIF$ except the recursive calls (whose processing time is related to other nodes of ST). The total time of these operations can be bounded by a polynomial multiplied to a number of nodes of ST. The resulting polynomial is suppressed by the O^* notation, hence $O^*(1.8899^n)$ is an upper bound on the time complexity of $Main\_MIF$.

Observe that $Find\_MIF$ has a polynomial space complexity because it is a backtrack-like procedure without explicit recording of results related to intermediate recursive calls. ■

References

1. J. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters*, 32(6):547–556, November 2004.
2. E. Dantsin, A. Goerdt, E. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic (2-2/(k+1))$^{\mathrm{n}}$ algorithm for k-sat based on local search. *Theor. Comput. Sci.*, 289(1):69–83, 2002.
3. D. Eppstein. Improved algorithms for 3-coloring, 3-edge coloring and constraint satisfaction. In *SODA-2001*, pages 329–337, 2001.
4. F. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: Domination - a case study. In *ICALP*, pages 191–203, 2005.
5. 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.
6. F. Fomin and A. Pyatkin. Finding minimum feedback vertex set in bipartite graphs. In *Report N 291, Department of Informatics, University of Bergen*, 2005.
7. R. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, 1972.
8. V. Raman, S. Saurabh, and S. Sikdar. Improved exact exponential algorithms for vertex bipartization and other problems. In *ICTCS*, pages 375–389, 2005.
9. J. Robson. Algorithms for maximum independent sets. *Journal of Algorithms*, 7:425–440, 1986.
10. G. Woeginger. Exact algorithms for NP-hard problems: A survey. In *Combinatorial Optimization: "Eureka, you shrink", LNCS 2570*, pages 185–207, 2003.