Exact Elimination of Cycles in Graphs

Henning Fernau[1], Daniel Raible[2],

[1] Univ.ersität Trier, FB 4—Abteilung Informatik, 54286 Trier, Germany
`fernau@uni-trier.de`
[2] Univ.ersität Trier, FB 4—Abteilung Informatik, 54286 Trier, Germany
`raible@uni-trier.de`

Abstract. One of the standard basic steps in drawing hierarchical graphs is to invert some arcs of the given graph to make the graph acyclic. We discuss exact and parameterized algorithms for this problem. In particular we examine a graph class called $(1, n)$-graphs, which contains cubic graphs. We discuss exact and parameterized algorithms, where we use a non-standard measure approach for the analysis. Especially the analysis of the parameterized algorithm is of special interest, as it is not an amortized analysis modelled by 'finite states' but is rather a 'top-down' amortized analysis. For $(1, n)$-graphs we achieve a running time of $\mathcal{O}^*(1.1871^m)$ and $\mathcal{O}^*(1.212^k)$, for cubic graphs $\mathcal{O}^*(1.1798^m)$ and $\mathcal{O}^*(1.201^k)$, respectively. As a by-product the the trivial bound of 2^n for Feedback Vertex Set on planar directed graphs is broken.

1 Introduction and Definitions

Our problem. The Sugiyama approach [2,3,22] is the most popular way to draw graphs in a hierarchical fashion. In the first phase of this approach, as few arcs as possible are re-oriented in order to make the graph acyclic. This re-orientation of arcs is equivalent to the Maximum Acyclic Subgraph problem, MAS, and to the Feedback Arcset Problem, FAS, see [2,3,22], which is on the list of 21 problems that was presented by Karp [15] in 1972 when showing the first $\mathcal{NP}$-complete problems. It has numerous further applications [11], ranging from program verification, VLSI to other network applications. More formally, we consider the following problem: Maximum Acyclic Subgraph MAS

Given a directed graph $G(V, A)$, and the parameter k.
We ask: Is there a subset $A' \subseteq A$, with $|A'| \geq k$, which is acyclic ?

In this paper, we deal with finding exact and parameterized algorithms for MAS. So, we continue studies on using exact algorithmics within the Sugiyama approach, see [6,7,9,20,21].

A new class of graphs. Mostly, we focus on a class of graphs that, to our knowledge, has not been previously described in the literature. Let us call a directed graph $G = (V, E)$ $(1, n)$-*graph* if, for each vertex $v \in V$, its indegree $d^+(v)$ obeys $d^+(v) \leq 1$ or its outdegree $d^-(v)$ satisfies $d^-(v) \leq 1$. In short, we can write this condition as follows: $\forall v \in V : \min\{d^+(v), d^-(v)\} \leq 1$. In particular, graphs of maximum degree three are $(1, n)$-graphs. Notice that MAS, restricted to cubic graphs, is still $\mathcal{NP}$-complete.

For some applications from graph drawing (e.g., laying out "binary decision diagrams" where vertices correspond to yes/no decisions) even the latter restriction is not so severe at all. Having a closer look at the famous paper of Nassi and Shneiderman [18] where they introduce structograms to aid structured programming (and restricting the use of GOTOs), the resulting class of flowchart graphs will be $(1, n)$-graphs.

The degree restriction to three has also been discussed in relation to approximation algorithms. Newman [19] showed that MAS can be approximated up to a factor of $\frac{12}{11}$. Having a closer look at her algorithm reveals that it also works for $(1, n)$-graphs with the same approximation factor. This largely improves on the general situation, where only a factor of 2 is known [3], although Eades, Lin and Smith [8] obtained an improvement displayed in Eq. (1).

Our framework: Parameterized Complexity. We will focus on studying MAS within this framework. A *parameterized problem* P is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a fixed alphabet and $\mathbb{N}$ is the set of all non-negative integers. Therefore, each instance of the parameterized problem P is a pair (I, k), where the second component k is called the *parameter*. The language $L(P)$ is the set of all YES-instances of P. We say that the parameterized problem P is *fixed-parameter tractable* [5] if there is an algorithm that decides whether an input (I, k) is a member of $L(P)$ in time $f(k)|I|^c$, where c is a fixed constant and $f(k)$ is a function independent of the overall input length $|I|$. We will also write $\mathcal{O}^*(f(k))$ for this run-time bound. Equivalently, one can define the class of fixed-parameter tractable problems as follows: strive to find a polynomial-time transformation that, given an instance (I, k), produces another instance (I', k') of the same problem, where $|I'|$ and k' are bounded by some function $g(k)$; in this case, (I', k') is also called a *(problem) kernel*.

Discussion of related results. It should be noticed that the "dual" problem of finding a minimum feedback arc set (in general graphs) is known to possess a factor $\log n \log\log n$-approximation, see [11], and hence shows an approximability behavior much worse than MAS. This might indicate that FAS is also hard from a parameterized perspective, although this is still unknown. Interestingly, it seems to be even unknown if FAS, restricted to cubic graphs, is constant-factor-approximable or if this problem is fixed-parameter tractable.

The complexity picture changes when one considers undirected graphs. The task of removing a minimum number of edges to obtain an acyclic graph can be accomplished in polynomial time basically by finding a spanning forest. The task of removing a minimum number of vertices to obtain an acyclic graph is (again) $\mathcal{NP}$-complete, but can be approximated to a factor of two and is known to be fixed-parameter tractable, see [1,4,11,14]. Also, exact (non-parameterized) algorithms have been derived for this problem [12].

Our contributions. Our main technical contribution of this paper is to derive a parameterized $\mathcal{O}^*(1.212^k)$-algorithm for MAS on $(1, n)$-graphs. However, notice that the analysis also works for graphs where only few vertices v show up with $\min\{d^+(v), d^-(v)\} > 1$, since one could first branch at their adjacent arcs and then quickly derive a situation when our analysis applies. In practice, our restriction should therefore not be considered harmful. We also derive exact algorithms for MAS on $(1, n)$-graphs.

Besides being a nice combinatorial problem on its own right, we think that our contribution is also interesting from the more general perspective of a development of tools for constructing efficient parameterized algorithms. Namely, the algorithm we present is of a quite simple overall structure, similar in simplicity as, e.g., the recently presented algorithms for Hitting Set [10]. But the analysis is quite intricate and seems to offer a novel way of amortized search tree analysis that might be applicable in other situations in parameterized algorithmics, as well. It appears to be the first ever application of the "measure & conquer" paradigm [13] in parameterized algorithmics. We mention that due to lack of space proofs can be found in the appendix.
We first link approximability and parameterized algorithmics by a simple but interesting observation. We call a maximization problem P a *set maximization problem* if its task, given instance I, is to identify a subset S (satisfying additional requirements) of a *ground set* M of maximum cardinality. The natural parameterized problem related to P (denoted by P_{par}) is to find, given (I, k) a subset S (satisfying additional requirements) of cardinality at least k.

Proposition 1. *If a set maximization problem P has a c-approximation, where the ratio is measured with respect to the whole ground set M that is part of the input I so that the size $|I|$ is measured in terms of the cardinality $|M|$ of M, then P_{par}, parameterized by k, has a kernel of size upper-bounded by ck.*

Both above mentioned approximations exhibit the required properties of proposition 1, entailing a $2k$-kernel for the general case of MAS, as well as a $\frac{12}{11}k$-kernel when restricted to $(1, n)$-graphs.

Corollary 1. MAS *is fixed-parameter tractable.*

More precisely, an easy exhaustive search can be run on the kernel, yielding an $\mathcal{O}^*(4^k)$-algorithm for the general case. For a planar graph $G = (V, E)$, we can slightly improve on this result by using the estimate

$$|S| \geq \frac{|E|}{2} + \frac{|V|}{6} \geq \frac{5|E|}{9} \tag{1}$$

for a solution S returned by the algorithm ELS from [8] (together with Euler's formula that yields $|E| \leq 3|V|$):

Corollary 2. MAS *on planar graphs can be solved in time $\mathcal{O}^*(3.445^k)$.*

Finally fixing terminology. We consider directed multigraphs $G(V, A)$ in the course of our algorithm, where V is the vertex set and A the arc set. From A to V we have two kinds of mappings: For $a \in A$, $init(a)$ denotes the vertex at the beginning of the arc a and $ter(a)$ the end. We distinguish between two kinds of arc-neighborhoods of a vertex v which are $E^+(v) = \{a \in A \mid ter(a) = v\}$ and $E^-(v) = \{a \in A \mid init(a) = v\}$. We have an in- and outdegree of a vertex, that is $d^+(v) = |E^+(v)|$ and $d^-(v) = |E^-(v)|$. We set $E(v) = E^+(v) \cup E^-(v)$ and $d(v) = |E(v)|$ called the degree of v. We also define a neighborhood for arcs a $N_A(a) := \{a_1, a_2 \in A \mid ter(a_1) = init(a), ter(a) = init(a_2)\}$ and for $A' \subseteq A$ we set $N_A(A') := \bigcup_{a' \in A'} N_A(a')$. For $V' \subseteq V$ we set $A(V') = \{a \in A \mid \exists u, v \in V', init(a) = u, ter(a) = v\}$. We call an arc (u, v) a *fork* if $d^-(v) \geq 2$ (but $d^+(v) = 1$) and a *join* if $d^+(u) \geq 2$ (but $d^-(u) = 1$). With $\mathcal{MAS}$, we refer to a set of arcs, which is acyclic and is a partial solution. A *undirected cycle* is an acyclic arc set, which is a cycle in the underlying undirected graph.

2 Preprocessing & Reduction Rules

Firstly, we can assume that our instance $G(V, A)$ forms a strongly connected component. Every arc not in such a component can be taken into a solution, and two solutions of two such components can be simply joined. We will mark those reduction rules that are sound for any graph by an asterisk in the following.

Preprocessing. In [11,16,19] a set of preprocessing rules is already mentioned:

Pre-1: $(*)$ For every $v \in V$ with $d^+(v) = 0$ or $d^-(v) = 0$, delete v and $E(v)$ and decrement k by $|E(v)|$.

Pre-2: $(*)$ For every $v \in V$ with $E(v) = \{(i, v), (v, o)\}$, delete v and $E(v)$ and introduce a new arc (i, o). Decrease k by one.

If in **Pre-2** $i = o$, a cycle of length one results. Basically, i.e., without referring to the adaptation to the parameterized setting as done in this paper, the reduction rules we used were known before; their power has been experimentally verified by Koehler in [16], where he showed, e.g., that the rules could completely solve one third of randomly created 10.000-vertex-digraphs, and on average the number of remaining vertices was reduced to one thirtieth. After carrying out **Pre-1** to **Pre-2** the resulting graph has no vertices of degree less than three.

Definition 1. *An arc g is an α-arc if it is a fork and a join.*

We need the next lemma, which is a sharpened version of [19, Lemma 2.1] and follows the same lines of reasoning.

Lemma 1. *Any two nondisjoint cycles in a $(1, n)$-graph with minimum degree at most 3 share an α-arc.*

We partition A in two parts, namely $A_\alpha = \{g \mid g \in A,\ g \text{ is an } \alpha\text{-arc}\}$ and $A_{gr} := A \setminus A_\alpha$. Let $G' := G[A_{gr}]$. By Lemma 1, the remaining cycles in G' must be arc disjoint. This justifies the next preprocessing rule.

Pre-3 In G delete the arc set of every cycle C in G'. For an arbitrary $a \in A(C)$ adjoin $A(C) \setminus \{a\}$ to $\mathcal{MAS}$ and decrease k by $|A(C)| - 1$.

After exhaustively applying Preprocess() (shown in Figure 1), every cycle has an α-arc. For $v \in V$ with $E^+(v) = \{a_1, \ldots, a_s\}$ ($E^+(v) = \{c\}$, resp.) and $E^-(v) = \{c\}$ ($E^-(v) = \{a_1, \ldots, a_s\}$), it is always better to delete c than one of $a_1, \ldots, a_s$. Therefore, we adjoin $a_1, \ldots, a_n$ to $\mathcal{MAS}$, adjusting k accordingly. Having applied this rule on every vertex, we adjoined A_{gr} to $\mathcal{MAS}$, and the remaining arcs are exactly A_α. So, the next task is to find $S \subseteq A_\alpha$ with $|\mathcal{MAS} \cup S| \geq k$ so that

```
Procedure:
Preprocess($\mathcal{MAS}$,$G(V,A)$,$k$):
1: repeat
2:     cont ← false
3:     for i=1 to 3 do
4:         apply Pre-i exhaustively.
5:         if Pre-i applied then
6:             cont ← true
7: until cont=false
8: return ($\mathcal{MAS}$,$G(V,A)$,$k$,$k$,1,1)
```

```
Procedure:
Reduce($\mathcal{MAS}$,$G(V,A)$),$k$,$k'$,$w_k$,$w_{k'}$):
1: repeat
2:     cont ← false
3:     for i=1 to 6 do
4:         apply RR-i exhaustively.
5:         if RR-i applied then
6:             cont ← true
7: until cont=false
8: return ($\mathcal{MAS}$,$G(V,A)$,$k$,$k'$,$w_k$,$w_{k'}$)
```

Fig. 1. The procedures Preprocess() and Reduce().

$G[\mathcal{MAS} \cup S]$ is acyclic. We have to branch on the α-arcs, deciding whether we take them into $\mathcal{MAS}$ or if we delete them. α-arcs which we take into $\mathcal{MAS}$ will be called *red*. For the purpose of measuring the complexity of the algorithm, we will deal with two parameters k and k', where k is the size of the solution and k' will be used for purposes of run-time estimation. The big difference is that we do not count the arcs in A_{gr} immediately into k'. For every branching on an α-arc, we want to count only a portion of them into k'. More precisely, upon first seeing an arc $b \in A_{gr}$ within the neighborhood $N_A(g)$ of an α-arc g we branch on, we will count b only partially, by an amount of ω, where $0 < \omega < 0.5$ will be determined later. So, we will have two weighting functions w_k and $w_{k'}$ for k and k' with $w_k(a) \in \{0, 1\}$ and $w_{k'}(a) \in \{0, (1-\omega), 1\}$ for $a \in A$, indicating each how much of the arc has not been counted into k, or k' respectively, yet. In the very beginning, we have $w_k(a) = w_{k'}(a) = 1$ for all $a \in A$. For a set $A' \subseteq A$, we define $w_{k'}(A') := \sum_{a' \in A'} w_{k'}(a')$ and $w_k(A)$ accordingly.

The preprocessing rules, together with the mentioned kernel of $\frac{12}{11}k$ arcs, gives us another simple brute-force algorithm for MAS: Within the kernel with m arcs, there could be at most $m/3$ arcs that are α-arcs. It is obviously sufficient to test all possible $2^{m/3} \leq 2^{(4/11)k} \approx 1.288^k$ many possibilities of choosing α-arcs into the (potential) feedback arc set, also beating the enumeration of all k-subsets of the $\frac{12}{11}k$-kernel in time $\mathcal{O}^*(1.367^k)$.

Reduction Rules. There is a set of reduction rules from [19], which we adapted and modified to deal with weighted arcs. Also, we define a (linear time checkable) predicate *contractible* for all $a \in A$.

$$\mathit{contractible}(a) = \begin{cases} 0 : w_k(a) = 1, \exists \text{ cycle } C \text{ with } a \in C \text{ and } w_k(C \setminus \{a\}) = 0 \\ 1 : \text{else} \end{cases}$$

The meaning of this predicate is the following: if $\mathit{contractible}(a) = 0$, then a is the only remaining arc of some cycle, which is not already determined to be put into $\mathcal{MAS}$. Thus, a has to be deleted. In the following, **RR-i-1** is always carried out exhaustively before **RR-i**.

RR-1 For $v \in V$ with $d^+(v) = 0$ or $d^-(v) = 0$, take $E(v)$ into $\mathcal{MAS}$, delete v and $E(v)$ and decrease k by $w_k(E(v))$ and k' by $w_{k'}(E(v))$.
RR-2 For $v \in V$ with $E(v) = \{a, b\}$ let $z = \arg\max\{w_{k'}(a), w_{k'}(b)\}$ and $y \in E(v) \setminus \{z\}$. If $\mathit{contractible}(y) = 1$, then contract y, decrement k by $w_k(y)$, k' by $w_{k'}(y)$. If y was red, then z becomes red.
RR-3 If for $g \in A$, we have $\mathit{contractible}(g) = 0$, then delete g.

α-arcs a which are not red are called *unprocessed* and are referred to as A^U_α. We classify the unprocessed α-arcs in *thin α-arcs*, which are contained in exactly one cycle, and *thick α-arcs*, which are contained in at least 2 cycles. Because G is strongly connected, there are no α-arcs which are contained in no cycle. We can distinguish them as follows: For every α-arc g, find the smallest cycle C_g which contains g via Breadth-First-Search (BFS). If g is contained in a second cycle C'_g, then there is an arc $a \in A(C_g)$ with $a \notin A(C'_g)$. So for every arc a of C_g, remove a and restart BFS. If we find this way another cycle, we know g is on two cycles. When running Algorithm 1, we guarantee that every unprocessed α-arc has value one for both weights, yielding:

RR-4 If $g \in A^U_\alpha$ is thin and $\mathit{contractible}(g) = 1$, then take g into $\mathcal{MAS}$ and decrease k by $w_k(g)$, k' by $w_{k'}(g)$ and set $w_k(g) \leftarrow 0$, $w_{k'}(g) \leftarrow 0$.
RR-5 If $a, b \in A$ form an undirected 2-cycle then delete $z = \arg\min\{w_{k'}(a), w_{k'}(b)\}$, decrease k by $w_k(z)$ and k' by $w_{k'}(z)$ and take z into $\mathcal{MAS}$.
RR-6 Having $(u, v), (v, w), (u, w) \in A$ (an undirected 3-cycle), delete (u, w), decrease k by $w_k((u, w))$ and k' by $w_{k'}((u, w))$, and take (u, w) into $\mathcal{MAS}$.

Theorem 1. *The reduction rules are sound.*

Proposition 2. *After the application of Reduce(), see Figure 1, we are left with a $(1, n)$-graph G with only thick α-arcs and with no directed or undirected 2- or 3-cycle.*

3 The Algorithm and its Analysis

3.1 The Algorithm

We are ready now to state our main Algorithm 1; observe that the handling of the second parameter k' is only needed for the run-time analysis and could be avoided when implementing the algorithm. Therefore, the branching structure of the algorithm is quite simple, as expressed in the following:

Algorithm 1 A parameterized algorithm for Maximum Acyclic Subgraph on $(1,n)$-graphs

```
1: $(\mathcal{MAS},G(V,A),k,k',w_k,w_{k'}) \leftarrow$ Preprocess$(\mathcal{MAS},G(V,A),k)$.
2: $\mathcal{MAS} \leftarrow A_{gr} \cup \mathcal{MAS}, k' \leftarrow k,\ k \leftarrow k - w_k(A_{gr}),\ w_k(A_{gr}) \leftarrow 0$
3: Sol3MAS$(\mathcal{MAS},G(V,A),k,k',w_{k'},w_k)$
Procedure: Sol3MAS$(\mathcal{MAS},G(V,A),k,k',w_k,w_{k'})$:
1: $(\mathcal{MAS},G(V,A),k,k',w_k,w_{k'}) \leftarrow$ Reduce$(\mathcal{MAS},G(V,A),k,k',w_k,w_{k'})$
2: if $k \leq 0$ then
3:   return YES
4: else if there is a component $C$ with at most 9 arcs then
5:   Test all possible solutions for $C$.
6: else if there is an unprocessed $\alpha$-arc $g$ then
7:   if not Sol3MAS$(\mathcal{MAS}\ ,G[A \setminus \{g\}],k,k',w_k,w_{k'})$ then
8:     $k \leftarrow k-1,\ k' \leftarrow k'-w_{k'}(g),\ w_k(g) \leftarrow w_{k'}(g) \leftarrow 0,\ \mathcal{MAS} \leftarrow \mathcal{MAS} \cup N_A(g) \cup \{g\}$.
9:     for all $a \in N_A(g)$ do
10:      Adjust $w_{k'}$, see Figure 2.
11:    return Sol3MAS$(\mathcal{MAS}\ ,G(V,A),k,k',w_k,w_{k'})$
12:  else
13:    return YES
14: else
15:   return NO
```

```
Adjust $w_{k'}$:
1: if $w_{k'}(a) = 1$ then
2:   if $\exists b \in (N_A(a) \setminus (N_A(g) \cup \{g\}))$ with $w_{k'}(b) = 0$ then
3:     $k' \leftarrow k'-1,\ w_{k'}(a) \leftarrow 0,\ k \leftarrow k - w_k(a), w_k(a) \leftarrow 0$  (case $a$.)
4:   else
5:     $k' \leftarrow k' - \omega,\ w_{k'}(a) \leftarrow (1-\omega),\ k \leftarrow k - w_k(a), w_k(a) \leftarrow 0$  (case $b$.)
6: else
7:   $k' \leftarrow k' - (1-\omega),\ w_{k'}(a) \leftarrow 0$.  (case $c$.)
```

Fig. 2. In case a. we set $w_{k'}(a) = 0$, because there will not be any other unprocessed neighboring α-arc of a. In case b., this might not be the case, so we count only a portion of ω. In case c., we will prove that $w_{k'}(a) = (1-\omega)$ and that there will be no other unprocessed neighboring α-arc of a, see Theorem 2.6

Lemma 2. *Branching in Alg. 1 either puts a selected α-arc a into $\mathcal{MAS}$, or it deletes a. Only if arcs are deleted, reduction rules will be triggered in the subsequent recursive call.This can can be also due to triggering* **RR-3** *after putting a into $\mathcal{MAS}$.*

3.2 Analysis

Basic Combinatorial Observations. While running the algorithm, $k \leq k'$. Now, substitute in line 2 of Sol3MAS of Algorithm 1 k by k'. If we run the algorithm, it will create a search tree $T_{k'}$. The search tree T_k of the original algorithm must be contained in $T_{k'}$, because $k \leq k'$. If $|T_{k'}| \leq c^{k'}$, then it follows that also $|T_k| \leq c^{k'} = c^k$, because in the very beginning, $k = k'$. So in the following, we will state the different recurrences derived from algorithm 1 in terms of k'. For a good estimate, we have to calculate an optimal value for ω.

α-arc g	*a.*	*b.*	*c.*
$\mathcal{MAS}$	1	ω	$(1-\omega)$
Deletion	1	$1+(1-\omega)$	$(1-\omega)$

Table 1. We summarize by which amount k' can be decreased for $a \in N_A(g)$, subject to if we take g into $\mathcal{MAS}$ or we delete g and to the case applying to a.

Theorem 2. *In every node of the search tree, after applying Reduce(), we have:*

1. *For all $a = (u,v) \in A_{gr}$ with $w_{k'}(a) = (1-\omega)$, there exists an incident red arc d, which is a fork or a join.*
2. *For all non-red $a = (u,v) \in A_{gr}$ with $w_{k'}(a) = 0$, we find red arcs $(u',u), (v,v')$. We will also say that a is* protected *(by the red arcs).*
3. *For all red arcs $d = (u,v)$ with $w_{k'}(d) = 0$, if we have only non-red arcs in $E(u) \setminus \{d\}$ ($E(v) \setminus \{d\}$, resp.), then d is a join (d is a fork).*
4. *For all red arcs $d = (u,v)$, if there is a non-red arc $a \in E^+(u) \setminus \{d\}$ ($a \in E^-(v) \setminus \{d\}$, resp.) with $w_{k'}(a) = 0$ and only non-red arcs in $E(u) \setminus \{a,d\}$ (in $E(v) \setminus \{a,d\}$, resp.), then d is a join (d is a fork).*
5. *For each red arc $d = (u,v)$ with $w_{k'}(d) = 0$ that is not a join (fork, resp.), if there is at least one red arc in $E(u) \setminus \{d\}$ (in $E(v) \setminus \{d\}$, resp.), then there is a fork (join, resp.) in $G[E(u)]$ ($G[E(v)]$).*
6. *For all $g \in A^U_\alpha$ and for all $a \in N_A(g)$, we have: $w_{k'}(a) > 0$.*

There, we use induction on the depth of the search tree. Clearly, all claims are trivially true for the original graph instance, i.e., the root node. Notice that each claim has the form $\forall a \in A(X(a) \implies Y(a))$. Here, X and Y express local situations affecting a. Therefore, we have to analyse how $X(a)$ could have been created by branching. According to Lemma 2, we have to discuss what happens (1) if a certain α-arc had been turned red and (2) if reduction rules were triggered. As a third point, we must consider the possibility that $X(a)$ is true both in the currently observed search tree node s and in its predecessor, but that $Y(a)$ was possibly affected upon entering s.

Estimating the running time for maximum degree 3 graphs. In Algorithm 1, depending in which case of Figure 2 we end up, we decrement k' by a different amount for each arc $a \in N_A(g)$ in the case that we put g into $\mathcal{MAS}$. We can be sure that we may decrement k' by at least $(1-\omega)$ for each neighbor $a \in N_A(g)$ due to the last property of the Theorem 2. If we put g into FAS, we delete g and we will delete $N_A(g)$ immediately by **RR-1**, and we can decrement k' accordingly (by $w_{k'}(N_A(g))$). Moreover, if case *b.* applies to $a \in N_A(g)$, we know that the two arcs $d, e \in (N_A(a) \setminus (N_A(g) \cup g\}))$ (observe that we do not have triangles) obey $w_{k'}(d)w_{k'}(e) > 0$. By deleting a, no matter whether **RR-1**

or **RR-2** applies to d and e (this depends on the direction of the arcs) we can decrement k' by an extra amount of at least $(1-\omega)$, cf. the handling of k' by these reduction rules. This is true even if $V(d), V(e) \subset V(N_A(g))$. Note that if $V(N_A(N_A(g))) \subseteq V(N_A(g))$, then $A(V(N_A(g)))$ is a component of 9 arcs (which are handled separately).
Let i denote the number of arcs $a \in N_A(g)$ for which case a. applies. In the analogous sense j stands for the case b. and q for c.. For every positive integer solution of $i + j + q = 4$, we can state a total of 15 recursions according to Table 1 depending on ω. A table with all 15 recursions $T_1, \ldots, T_{15}$ can be found in appendix C. For every T_i and for a fixed ω, we can calculate a constant $c_i(\omega)$ such that $T_i[k] \in \mathcal{O}^*(c_i(\omega)^k)$. We want to find a ω with subject to minimize $\max\{c_1(\omega), \ldots, c_{15}(\omega)\}$. We numerically obtained $\omega = 0.1687$ so that $\max\{c_1(\omega), \ldots, c_{15}(\omega)\}$ evaluates to 1.201. The dominating cases are when $i = 0, j = 4, q = 0$ (T_5) and $i = 4, j = 0, q = 0$ (T_{15}).

Theorem 3. *Algorithm 1 solves* MAS *on graphs G with $\Delta(G) \leq 3$ in $\mathcal{O}^*(1.201^k)$.*

Measuring the run time in terms of $m := |A|$ the same way is also possible. Observe that if we take an α-arc into FAS, we can decrement m by one more. By adjusting $T_1, \ldots, T_{15}$ according to this and by choosing $\omega = 0.2016$, it follows:

Theorem 4. *On graphs G with $\Delta(G) \leq 3$* MAS *is solvable in $\mathcal{O}^*(1.1798^m)$.*

Note that if we run this exact algorithm on the $\frac{12}{11}k$-kernel, we finish in $\mathcal{O}^*(1.1977^k)$, which is slightly better than the pure parameterized algorithm. We will obtain a better bound for the search tree by a precedence rule, aiming to improve recurrence T_5. If we branch on an α-arc g according to this recurrence, for all $a \in N_A(g)$ we have $w_{k'}(a) \geq (1-\omega)$. Such α-arcs will be called α_5*-arcs*. We add the following rule: branch on α_5-arcs with least priority. Let $l := |A_\alpha^U|$.

Lemma 3. *Branching on an α_5-arc, we can assume: in the underlying undirected graph,*

$$\left\lfloor \frac{1}{1+4(1-\omega)} k' \right\rfloor \leq l < \left\lceil \frac{1}{4(1-\omega)} k' \right\rceil .$$

Employing this lemma, we can find a good combinatorial estimate for a brute-force search at the end of the algorithm. This allows us to conclude:

Theorem 5. MAS *is solvable in time $\mathcal{O}^*(1.1960^k)$ on maximum-degree-3-graphs* .

Estimating the running time for $(1, n)$-graphs. There is a difference to maximum degree 3 graphs, namely the entry for case b. in case of deletion in Table 1. For $a \in N_A(g)$ it might be the case that $|N_A(a) \setminus (N_A(g) \cup \{g\}| > 3$, so that when we delete g and afterwards a by **RR1** that whether **RR-1** nor **RR-2** applies (due to the lack of sources, sinks or degree two vertices). We call this case b'. Remember, case b. refers to the same setting but with $|N_A(a) \setminus (N_A(g) \cup \{g\}| = 2$. Thus the mentioned entry should be 1 here. As long as $|N_A(g)| \geq 6$ the reduction in k' is great enough for the modified table, but for the other cases we must argue more detailed. We introduce two more reduction rules, the first already mentioned in [19].

RR-7 Contract and adjoin to $\mathcal{MAS}$ any $(u, v) \in A$ with $d^+(u) = d^+(v) = 1$ ($d^-(u) = d^-(v) = 1$, resp.). If (u, v) was red the unique arc $a := (x, u)$ ((v, y), resp.) will be red. Decrease k' by $\min\{w_{k'}((u, v)), w_{k'}(a)\}$ and set $w_{k'}(a) \leftarrow \max\{w_{k'}((u, v)), w_{k'}(a)\}$. Proceed similarly with (v, y).
RR-8 For a red $g' \in A_\alpha$ with $w_{k'}(g') > 0$, set $k' \leftarrow k' - w_{k'}(g')$ and $w_{k'}(g') \leftarrow 0$.

Lemma 4. ***RR-7*** *and* ***RR-8*** *are sound and do not violate Theorem 2.*

We add the following rules to Algorithm 1:

1. After Reduce(), first apply **RR-7** and then **RR-8** exhaustively.
2. Prefer α-arcs g such that $|N_A(g)|$ is maximal for branching.
3. Forced to branch on $g \in A_\alpha^U$ with $|N_A(g)| = 5$, choose an α-arc with the least occurrences of case b'

Lemma 5. *We will never have to branch on a $g = (u, v) \in A_\alpha^U$ with $|N_A(g)| = 5$ and 5 occurrences of case b'*

Lemma 6. *When we branch on $g = (u, v) \in A_\alpha^U$ with $|N_A(g)| = 4$ there is no case b' occurrence.*

Let x, y, z denote the occurrences of cases $a.$, b' and $c.$. To upperbound the branchings according to α-arcs g with $|N_A(g)| \geq 6$, we put up all recurrences resulting from integer solutions of $x + y + z = 6$. Note herefore we use the modification of table 1. To upperbound branchings with $|N_A(g)| = 5$ we put up all recurrences obtained from integer solutions of $x + y + z = 5$, except when $x = z = 0$ and $y = 5$ due to Lemma 5. Additionally we have to put up the recurrence covering the case where we have 4 occurrences of case b' and one of case $b.$. To upperbound the case where $|N_A(g)| = 4$ the recurrences derived from table 1 for the integer solutions of $x + y + z = 4$ suffice due to Lemma 6

Theorem 6. *On $(1, n)$-graphs with m arcs,* MAS *is solvable in time $\mathcal{O}^*(1.1871^m)$ ($\omega = 0.2392$) and $\mathcal{O}^*(1.212^k)$ ($\omega = 0.21689$), respectively.*

Corollary 3. Feedback Vertex Set *on directed maximum degree 3 graphs can be solved in $\mathcal{O}^*(1.282^n)$ and on planar directed graphs in $\mathcal{O}^*(1.986^n)$.*

3.3 Reparameterization

Mahajan, Raman and Sidkar [17] have discussed a rather general setup for reparameterization of problems according to a “guaranteed value.” In order to use their framework, we only need to exhibit a family of example graphs where Newman’s approximation bound is sharp.

Corollary 4. *For any $\epsilon > 0$, the following question is not fixed-parameter tractable unless $\mathcal{P} = \mathcal{NP}$: Given a cubic directed graph $G(V, E)$ and a parameter k, does G possess an acyclic subgraph with at least $\left(\frac{11}{12} + \epsilon\right)|E| + k$ many arcs ?*

4 Conclusions

We presented exact and parameterized algorithms for MAS. How well can they work in practice ? In exact algorithmics, one of the yardsticks commonly used is the so-called klam value, see [5]. This means that we look for a value of k (or m in our case) such that for the exponential run time estimate c^k, we find that $c^k \leq 10^{20}$. We end up with a klam value of about 300, which indicates that, at least for hiearchichal drawings that need to be nice but need not be immediately displayed (i.e., there is some time to process a nice layout), one can afford using exact algorithms in the first phase of the Sugiyama approach for graphs of moderate size. We have to underline once more that our run time estimates are pure worst-case bounds. Suderman [20] showed quite convincingly that (even for parameterized algorithms with much worse worst-case run-time bounds) exact algorithmics may lead to practical algorithms in the context of graph drawing. Coming back to our problem, notice that it is quite known that the heuristics that splits the graph in strongly connected components does al-

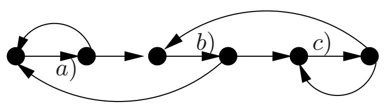

Fig. 3. A bad example for a well-known heuristics

ready a good job on practical instances, and this heuristic is built-in into our approach by assuming always working on strongly connected graphs. We also refer again to the paper of Koehler [16] who showed how well the preprocessing rules work on random graphs.

Let us conclude with an example that shows that quite some arcs can be "saved" with the exact approach as opposed to the often used heuristics of Eades, Lin and Smyth (ELM for short): In Figure 3, ELM would possibly pick arc b) first, because the incident vertices of a), b) and c) are all equal with respect to the difference between the in- and outdegree. Afterwards, of course, a) and c) have to be deleted, too. So, the ratio between the arcs deleted this way and the optimal number is $\frac{2}{3}$. Admittedly, the ratio between the cardinality of the maximum acyclic subgraph and the solution of the heuristic is not so bad: $\frac{6}{7}$. Furthermore, Newman's algorithm would (possibly) yield a solution no better than ELM. Similar but larger graphs displaying the same ratios are easily found.

References

1. V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal of Discrete Mathematics*, 12:289–297, 1999.

2. O. Bastert and C. Matuszewski. Layered drawings of digraphs. In M. Kaufmann and D. Wagner, editors, *Drawing Graphs: Methods and Models*, number 2025, pages 87–120. Springer, 2001.
3. B. Berger and P. W. Shor. Approximation algorithms for the maximum acyclic subgraph problem. In *ACM-SIAM Symposium on Discrete Algorithms SODA*, pages 236–243, 1990.
4. F. Dehne, M. R. Fellows, M. Langston, F. Rosamond, and K. Stevens. An $O(2^{O(k)}n^3)$ FPT algorithm for the undirected feedback vertex set problem. In L. Wang, editor, *Computing and Combinatorics COCOON*, volume 3595 of *LNCS*, pages 859–869. Springer, 2005.
5. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
6. V. Dujmović, H. Fernau, and M. Kaufmann. Fixed parameter algorithms for one-sided crossing minimization revisited. In G. Liotta, editor, *Graph Drawing, 11th International Symposium GD 2003*, volume 2912 of *LNCS*, pages 332–344. Springer, 2004. A journal version has been accepted to Journal on Discrete Algorithms.
7. V. Dujmović and S. Whitesides. An efficient fixed parameter tractable algorithm for 1-sided crossing minimization. *Algorithmica*, 40:15–32, 2004.
8. P. Eades, X. Lin, and W. F. Smyth. A fast & effective heuristic for the feedback arc set problem. *Information Processing Letters*, 47:319–323, 1993.
9. H. Fernau. Two-layer planarization: improving on parameterized algorithmics. *Journal of Graph Algorithms and Applications*, 9:205–238, 2005.
10. H. Fernau. Parameterized algorithms for HITTING SET: the weighted case. In T. Calamoneri, I. Finocchi, and G. F. Italiano, editors, *Conference on Algorithms and Complexity CIAC*, volume 3998 of *LNCS*, pages 332–343. Springer, 2006.
11. P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, volume Supplemenent Volume A, pages 209–258. Kluwer Academic Publishers, 1999. Also AT&T Technical Report No. 99.2.2.
12. F. V. Fomin, S. Gaspers, and A. V. Pyatkin. Finding a minimum feedback set in time $\mathcal{O}(1.7548^n)$. In H. L. Bodlaender and M. Langston, editors, *International Workshop on Parameterized and Exact Computation IWPEC*, volume 4169 of *LNCS*, pages 184–191. Springer, 2006.
13. F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and conquer: domination – a case study. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP*, volume 3580 of *LNCS*, pages 191–203. Springer, 2005.
14. J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke. Improved fixed-parameter algorithms for two feedback set problems. In *Proceedings of the 9th Workshop on Algorithms and Data Structures WADS*, volume 3608 of *LNCS*, pages 158–168. Springer, 2005.
15. R. M. Karp. *Reducibility Among Combinatorial Problems*, pages 85–103. Plenum, 1972.
16. H. Koehler. A contraction algorithm for finding minimal feedback sets. In V. Estivill-Castro, editor, *Twenty-Eighth Australasian Computer Science Conference ACSC*, volume 38 of *CRPIT*, pages 165–174, Newcastle, Australia, 2005. ACS.
17. M. Mahajan, V. Raman, and Sikdar S. Parameterizing MAXNP problems above guaranteed values. In H. L. Bodlaender and M. Langston, editors, *International Workshop on Parameterized and Exact Computation IWPEC*, volume 4169 of *LNCS*, pages 38–49. Springer, 2006.
18. I. Nassi and B. Shneiderman. Flowchart techniques for structured programming. *ACM SIGPLAN Notices*, 12, 1973.

19. A. Newman. The maximum acyclic subgraph problem and degree-3 graphs. In M. X. Goemans, K. Jansen, J. D. P. Rolim, and L. Trevisan, editors, *RANDOM-APPROX*, volume 2129 of *LNCS*, pages 147–158. Springer, 2001.
20. M. Suderman. *Layered Graph Drawing*. PhD thesis, McGill University, Montréal, 2005.
21. M. Suderman and S. Whitesides. Experiments with the fixed-parameter approach for two-layer planarization. *Journal of Graph Algorithms and Applications*, 9(1):149–163, 2005.
22. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Trans. Systems Man Cybernet.*, 11(2):109–125, 1981.

A More an general graphs

In the introduction, we claimed that one could use our algorithm (including its analysis) also in the case when only a few vertices violate the $(1, n)$-condition. We give more details in this section.

Let v be a vertex with $\ell := \min\{d^+(v), d^-(v)\} \geq 2$. Without loss of generality, assume that $\ell = d^+(v) \leq d^-(v)$. Let $a_1, \ldots, a_\ell$ be the arcs pointing towards v. Then, we branch on all 2^ℓ bitvectors $(b_1, \ldots, b_\ell)$ that indicate whether or not to take a_i into $\mathcal{MAS}$. More precisely, we take a_i into $\mathcal{MAS}$ iff $b_i = 1$. We discuss the situation immediately after this branching for a specific bitvector $.b = (b_1, \ldots, b_\ell)$. Of course, we can modify our graph instance by deleting those arcs a_i for which $b_i = 0$. Define $B = \sum b_i$. If $B \leq 1$, then we would find $\min\{d^+(v), d^-(v)\} \leq 1$ as required for $(1, n)$-graphs. If $B > 1$, let us split v into v_1 and v_2 as follows:

1. Add a new arc (v_1, v_2) known to belong to $\mathcal{MAS}$.
2. Then, replace those a_i with $b_i = 1$ by $(init(a_i), v_1)$ and
3. replace all arcs of the form (v, u) by (v_2, u).
4. Finally, remove v (and all incident arcs).

In addition, one could now exhaustively apply the first two reduction rules. It is straightforward to see that our construction yields an equivalent instance of MAS, Moreover, it has (at least) one vertex less that violates the $(1, n)$-condition. Therefore, after only a few bad branches, we arrive at a situation where our algorithm (with the corresponding analysis) applies.

B Omitted Proofs

B.1 Proof of Proposition 1

Proof. Consider the parameterized problem with input (I, k). So, I is an instance of P. Start the c-approximation on I, obtaining a solution set S. If $|S| \geq k$, we are done. Otherwise, we know that $\frac{|M|}{|S|} \leq c$, and so $|I| = |M| \leq c \cdot |S| < c \cdot k$.

B.2 Reduction Rules & Preprocessing Correctness

Lemma 7. **Pre-1** *and* **RR-1** *are correct.*

Proof. A vertex $v \in V$ with $d^+(v) = 0$ or $d^-(v) = 0$ cannot be entered and left by a cycle, so the incident arcs are not part of any cycle.

Lemma 8. **Pre-2** *and* **RR-2** *are sound.*

Proof. For a vertex v with $E(v) = \{a, b\}$, we have to delete at most one arc from $\{a, b\}$ in order to cut a cycle. So we can take one into $\mathcal{MAS}$ and contract it. In the case of **RR-2**, we must check if the arc we want to contract is not the last remaining arc on a cycle, which is not in $\mathcal{MAS}$. This check is done be *contractible*(). If so, we have to delete it.
Also **RR-2** differs from the one in [19] by the fact that red arcs are dominant. It is possible that we create an new α-arc (w, v) by this rule, (w, v) being red. This is justified by the following observations. This type of application of **RR-2** to (w, v) and a red arc (z, t) is only possible, because at the time when (z, t) became red, there was, w.l.o.g., a directed path $P = z, t, u_1, \ldots, u_j, w, v$, and during the algorithm we deleted again and again those arcs incident to the u_i's which are not on P, see Figure 4.

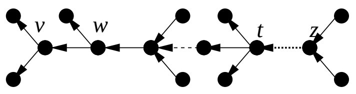

Fig. 4.

At the point when we want to merge (w, v) and (z, t), think of unwrapping P. To destroy any cycle passing through P, we have to delete at most one arc. But by already branching on (z, t) as a kind of representative of P, we have already decided that this arc should not be deleted, so we do not have to consider to delete (w, v) anymore, which means we do not have to branch here.

Lemma 9. **RR-3** *and* **RR-4** *are sound.*

Proof. If an α-arc is not contractible, it must be deleted because it is the only arc not in $\mathcal{MAS}$ for some cycle, so **RR-3** is correct.
If $g \in A_\alpha^U$ is thin, it can cut only one cycle C. Because it is contractible, there must be another α-arc g' which is able to cut C (possibly also some other cycle). We take g into $\mathcal{MAS}$ because it is no worse to delete g' than to delete g.

Lemma 10. RR-5 *is sound.*

Proof. Let $u, v \in V$ be the endpoints of an undirected 2-cycle. W.l.o.g., there are arcs a_1, a_2 with $init(a_1) = init(a_2) = u$ and $ter(a_1) = ter(a_2) = v$. Because having no vertices of degree less than three and the $(1, n)$-property, there are distinct arcs (c, u) and (v, b). Clearly, it is better to delete one of these arcs than to delete a_1 and a_2 (we have to delete both because they form an undirected 2-cycle). So by deleting, w.l.o.g., a_1, eventually we trigger **RR-2** on a_2. If, w.l.o.g, (c, u) will be contracted and a_1 deleted, we also should exchange these properties in a post-processing step, i.e., we delete (c, v) and take a_1 into $\mathcal{MAS}$.

Lemma 11. RR-6 *is sound.*

Proof. If we have $(u, v), (v, w), (u, w) \in A$, there must be also $(a, u), (w, b) \in A$ and w.l.o.g., $(c, v) \in A$, because of the absence of vertices of degree less than three. It is always better to delete (w, b) or (a, u) than to delete $(u, v), (v, w), (u, w)$. Also, any cycle C passing through (u, w) passes also through $(u, v), (v, w)$. So we have to take care only of cycles passing through $(u, v), (v, w)$. This justifies the deletion of (u, w). By deleting (u, w), **RR-2** eventually will be applied to (u, v) and (v, w). Again, if eventually (v, w) will be deleted and (w, b) taken into $\mathcal{MAS}$, we should exchange these properties in a post-processing step. The same is true for (u, v) and (a, u).

B.3 Proof of Theorem 2

Proof. We proceed by induction on the depth of the search tree. Clearly, all claims are trivially true for the original graph instance, i.e., at the root node. As induction hypothesis, we assume that the claim is true for all search tree nodes up to depth n. Let us discuss a certain search tree node s at depth $n+1$. Let $G = (V, A)$ be the graph instance associated with s. Let k and k' be the parameter values at node s. Let $\bar{s}$ be the immediate predecessor node of s in the search tree. We will refer with $\bar{G} = (\bar{V}, \bar{A})$, $\bar{k}$, $\bar{k}'$ to the corresponding instance and parameter values.

Exemplarily, we will give a very detailed proof of the very first assertion. The other parts can be similarly shown, so that we only indicate the basic steps of a complete formal proof.

1. Consider an arc $a = (u, v) \in A_{gr}$ with $w_{k'}(a) = (1 - \omega)$. This situation could have been due to three reasons:
(A) In node $\bar{s}$, we branched at an arc $\bar{d} \in N_A(\bar{a})$ with $w_{\bar{k}'}(\bar{a}) = 1$, see Figure 5. We consider here the case that $\bar{d}$ is turned red. Namely, according to case

b. of the procedure "Adjust", $w_{k'}(a) = (1 - \omega)$. Since we only branch at α-arcs, $\bar{d}$ is even both a fork and a join. As detailed in (B), $\bar{a}$ could give rise to $a \in V$ by a sequence of **RR-2**-applications in possible combination with other rule applications, such that $w_{k'}(a) = (1 - \omega)$. As described in (C), $\bar{d}$ will yield, as a red neighbor of $\bar{a}$, again by a (possibly empty) sequence of reduction rule applications, in particular of **RR-2**-applications, a fork or join that is neighbor of a in G as required.
(B) In node $\bar{s}$, we branched at some arc c. We consider here the case that (possibly due to an application of **RR-3**) b is deleted. This triggers some reduction rules. How could the situation have been created by reduction rule applications ? The only possibility is to (eventually) use **RR-2**. In that case, there would have been two neighbored arcs a', a'' in $\bar{G}$ with $\max\{w_{\bar{k}'}(a'), w_{\bar{k}'}(a'')\} = (1 - \omega)$. Hence, at least one of these arcs, say a', actually carried the weight $(1 - \omega)$. Moreover, since $a \in A_{gr}$, $a', a'' \in \bar{A}_{gr}$. In actual fact, there could have been a whole cascade of **RR-2**. applications along a path P in $\bar{G}$ (P consists of a sequence of subsequently neighbored arcs from $\bar{A}_{gr}$), eventually leading to a, but by an easy inductive argument one can see that there must have been some $\bar{a} \in \bar{A}_{gr}$ within this cascade to which the induction hypothesis applies, so that we conclude that, in $\bar{G}$, $\bar{a}$ has a neighboring arc $\bar{d}$ that is a fork or a join. Since $\bar{d}$ is red, $\bar{d}$ is not on the path P. Since d is a fork or a join, it cannot be neighbor to two subsequent arcs from the path P. Therefore, w.l.o.g., $\bar{a}$ is the first arc on P (without predecessors on P), and $\bar{d}$ is a fork. We will show in (C) that the fork $\bar{d}$ will eventually lead to a fork d that is neighbor of a in G.
(C) We consider the scenario that already in node $\bar{s}$, $w_{\bar{k}'}(a) = (1 - \omega)$. By induction hypothesis, assume that (w.l.o.g.) $a = (u, v)$ has a neighbored red fork $\bar{d} = (u', u)$. If $\bar{d}$ is deleted by using reduction rules, then u would have (intermediately) in-degree zero, so that **RR-1** triggers on a, contradicting our very scenario in G that we are discussing. Therefore, the local situation could only change by applications of **RR-2** involving $\bar{d}$. If those mergers refer to neighbors of $\bar{d}$ via the tip of $\bar{d}$, then either a is directly deleted or merged with $\bar{d}$. Both possibilities would destroy the scenario we discuss, since a would disappear. Therefore, such mergers could be only via the tail of $\bar{d}$. Since $\bar{d}$ is red, a merger with $\bar{d}$ will be red, as well. Moreover, this merger would be also a fork. Again by an easy induction, one can conclude that the neighbor d of a in G that results from a sequence of mergers using rule **RR-2** on a path ending at $\bar{d}$ in $\bar{G}$ would be a red fork as required.

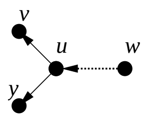

Fig. 5.

2. We will actually prove points 2. through 4. by a parallel induction.
   To improve readability of our main argument, we refrain from giving all possible details how the employment of **RR-2** may affect (but not drastically change) the situation in particular. These discussions are very similar to the ones given under item 1.

   How can $a = (u, v) \in A_{gr}$ with $w_{k'}(a) = 0$ have been created?

   Firstly, it could be due to a **RR2-**contraction with a non-red arc t with $w_{k'}(t) = 0$. But then, either a or t was not protected, which is a contradiction to the induction hypothesis.

   Secondly, it could be due to branching on a neighboring α-arc b, say $b = (v, w)$, in two different ways:
   (1) either we branched at b at a point of time when $w_{\bar{k}'}(a) = (1 - \omega)$ (case *c.* of Procedure "Adjust"), or
   (2) we branched at b when $w_{\bar{k}'}(a) = 1$ (case *a.*)
   In case (1), there must have been another red arc e incident to a by item 1, see Figure 6.1). of our property list. e is not incident to v, since it is a fork or a join. Hence, $e = (y, u)$. This displays the two required red arcs (namely b and e) in this case.
   In case (2), a was created by case *a.* of Procedure "Adjust." Obviously, b is red after branching. Since we have branched according to case a, there is another arc h incident with a (but not with b) such that $w_{\bar{k}'}(h) = 0$. There are four subcases to be considered:
   (a) $h = (u, u')$ is not red, see Figure 6.2.a). By induction, there must be a red fork arc (u'', u). Hence, a is protected.
   (b) $h = (u, u')$ is red, see Figure 6.2.b). Consider the all other arcs incident to u. Since we are dealing with reduced instances, they must be of the form $c = (u'', u)$, since otherwise u would be a sink. Now, by item 3. (induction), one of these arcs must be red. Therefore, a is protected.
   (c) $h = (u', u)$ is not red. All other arcs incident to u could be of the form (u'', u), see Figure 6.2.c.1). Since h must be protected, by induction, a should be red, contradicting our assumption on a. Therefore, all these arcs are of the form (u, u''), see Figure 6.2.c.1), and one of them, say c, is the red arc protecting h. This situation contradicts item 4. by induction, since c is neither fork nor join.
   (d) $h = (u', u)$ is red, see Figure 6.2.d). Hence, a will be protected.
3. How could d have been created?
   If it had been created by branching, then there are two cases: (1) d was put into $\mathcal{MAS}$; (2) d was neighbor of an arc b which we put into $\mathcal{MAS}$.
   In case (1), the claim is obviously true. In case (2), let, w.l.o.g., u be the common neighbor of b and d, see Figure 7.1). After putting b into $\mathcal{MAS}$, there will be a red arc (namely b), incident to u, so that there could be only non-red arcs incident with v that have the claimed property by induction. If d has been created by reduction rules, it must have been through **RR-2**. So, there have been (w.l.o.g.) two arcs (u, w) and (w, v) with $w_{\bar{k}'}$-weights zero. W.l.o.g., assume that (u, w) is red. If (w, v) is red, see Figure 7.2), then the

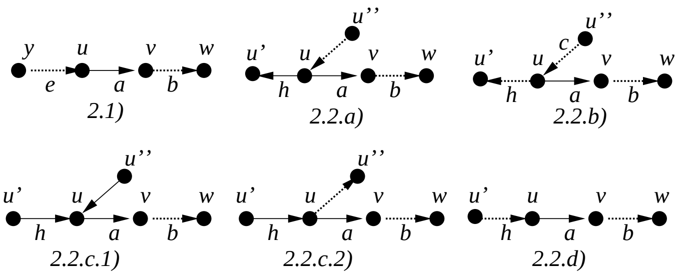

Fig. 6.

claim holds by induction. If (w, v) is not red, see Figure 7.3), then (w, v) must be protected.

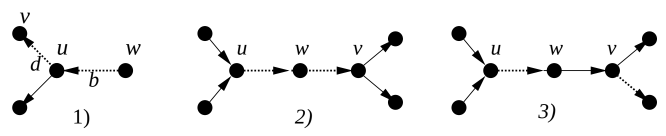

Fig. 7.

4. Again, we discuss cases how a with $w_{k'}(a) = 0$ could have been created. We can exclude that a has been created by using reduction rules as in the proof of item 2.
If a has been created by branching, then the claim is obviously true if we branched on d. Hence, we can assume that we actually branched on another (now red) arc b in the neighborhood of a. Without loss of generality, we have the following setting: $d = (u, v)$, $a = (y, u)$, $b = (x, y)$, see Figure 8. If, before branching, $w_{\bar{k}'}(a) = (1 - \omega)$, then by item 1., d is a join. If the situation before branching was that $w_{\bar{k}'}(a) = 1$, then, by claim 3., $w_{\bar{k}'}(d) > 0$ (or item 4. is shown). Since we are branching on b due to case a. of 'Adjust' and we can assume that $w_{\bar{k}'}(d) > 0$, we must have another $q \in N_A(a) \cap N_A(d)$ with $w_{\bar{k}'}(q) = 0$, since otherwise we would not find $w_{\bar{k}'}(a) = 1$. Hence, we can apply the induction hypothesis of claim 2: if q would not point towards u, it would not be protected, since a is not red. By the $(1, n)$-property, all other arcs in $N_A(a) \cap N_A(d)$ must point towards the common endpoint of a and d. Hence, in all cases, d is a join. The other case (in which d would be a fork) is symmetric.
5. We again discuss the possibilities that may create a red d with $w_{k'}(d) = 0$.
If d was created by taking it into $\mathcal{MAS}$ during branching, then d would be both fork and join in contrast to our assumptions.
If we branch in the neighborhood of d, then the claim could be easily verified directly.

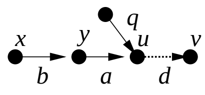

Fig. 8.

Finally, d could be obtained from merging two arcs $e = (u, w), f = (w, v)$ with $w_{\bar{k}'}(e) = w_{\bar{k}'}(f) = 0$. If both e and f are red, the claim follows by induction. If only f is red and e is non-red, then consider the neighboring arc h of e, with h being red by claim 2. and directed as $h = (z, u)$ and the other arcs $j_1, \ldots, j_i$ also incident to u, see Figure 9.1). If all $j_1, \ldots, j_i$ are of the form (u, u'), see Figure 9.2), then h is a fork and the claim is true.. If all $j_1, \ldots, j_i$ are of the form (u', u), see Figure 9.3), then e is a join and so is the resulting d. Thus the premise is falsified. Because of the $(1, n)$-property, there are no other possibilities for $j_1, \ldots, j_i$.

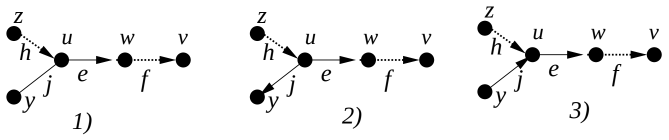

Fig. 9.

6. Assume the contrary. Discuss a neighbor arc a of g with $w_{k'}(a) = 0$.
   If a is not red, then g must be red due to item 2.
   If a is red, then discuss another arc b that is incident to the common endpoint of a and g.
   If b is not red, then the situation contradicts item 3. due to the direction of b. So, b must be red. This picture contradicts item 5.

B.4 Proof of Lemma 3

Proof. If $l \geq \left\lceil \frac{1}{4(1-\omega)} k' \right\rceil$ then by deleting A^U_α, we decrement k' by at least $l \cdot 4(1-\omega) \geq k'$, returning YES. If $l < \left\lfloor \frac{1}{1+4(1-\omega)} k' \right\rfloor$ then by taking A^U_α into $\mathcal{MAS}$ we decrement k' by at most $l \cdot (1 + 4(1-\omega)) < k'$, returning NO.

B.5 Proof of Theorem 5

Proof. So using Lemma 3, in general we can find $b \geq 1$ such that $l = \left\lceil \frac{1}{4(1-\omega)} k' - b \right\rceil$. Again, if we decided to delete A^U_α we decrement k' by at least $l \cdot 4(1-\omega)$, so that afterwards $k' \leq b4(1-\omega)$. If $k' \geq k > 0$, we have to step back and take some arcs of A^U_α into $\mathcal{MAS}$. For any such arc we can decrement k' by one more than by deleting it. Finally, we have to find at most $\lceil b4(1-\omega)\rceil$ arcs from A^U_α, which we can take in to $\mathcal{MAS}$ without causing any cyclicity. For this we have $\binom{l}{\lceil b4(1-\omega)\rceil}$ choices, which is biggest for $l = 2\lceil b4(1-\omega)\rceil$. So for $b = \frac{1}{4(1-\omega)(9-8\omega)}k'$ this can be upper bounded asymptotically by $\mathcal{O}^*\left(\binom{\frac{2}{4(1-\omega)(9-8\omega)}k'}{\frac{1}{4(1-\omega)(9-8\omega)}k'}\right) \subseteq \mathcal{O}^*\left(4^{\frac{1}{4(1-\omega)(9-8\omega)}k'}\right)$. We mention that we have to take care of the case where $k' = l$. In this case we have to check whether $G[\text{MAS}\cup A^U_\alpha]$ is acyclic and give the appropriate answer. Then the above mentioned run time for recurrence T_5 can be assumed. For $\omega = 0.2012$ we get an improved run time of $\mathcal{O}^*(1.1960^k)$, where recurrences T_4 and T_{15} are dominating. Further note that we can not make use of Lemma 3 when we measure the running time in terms of m.

B.6 Proof of Lemma 4

Proof. It should be rather obvious that applying **RR-8** is sound and does not interfere with Theorem 2.
Let us discuss the following scenario: Assume arcs (u, v) and (v, w) such that v and w have indegree one. Hence, **RR-7** could apply. Before applying **RR-7**, we find: (u, v) is a fork. (v, w) is not a join. (v, w) is a fork.

Hence, after applying **RR-7**, (u, v) is a fork. It will be an α-arc iff (u, v) was an α-arc before applying **RR-7**.

The soundness of the rule follows by induction. As well as the fact that the assertions of Theorem 2 will also hold after applying **RR-7**.

Namely, if some arc (that is not removed by rule **RR-7**) was neighbor of some fork or of some join or of some red arc before applying **RR-7**, this will be true after applying **RR-7**.

Moreover, observe that through applying **RR-7** or **RR-8**, no other reduction rule (numbered up to 6) could be triggered.

B.7 Proof of Lemma 5

Proof. Suppose the contrary holds. W.l.o.g. $d^-(v) = 2$. For (v, a) case b' must match. This means that $d^-(a) = 1$ and $d^+(v) \geq 3$ or otherwise **RR-1** or **RR-2** could be applied to a after the deletion of (v, a). For the distinct arc $b = (a, y)$ we must have $d^-(y) > 1$ or otherwise **RR-7** could be applied. Thus b must be an non-red α-arc (otherwise **RR-8** could be applied due to having case b' and therefore $w_{k'}(b) > 0$.). Also we must have $|N_A(b)| = 5$ and at most 4 case b' and at least one case b. occurrences. This contradicts the choice of the α-arc g, because we would have preferred b.

B.8 Proof of Lemma 6

Proof. Suppose the contrary holds. W.l.o.g. take an arc (v, a). As in the previous lemma we can deduce that $d^-(a) = 1$ and $d^+(v) \geq 3$ and that there is a distinct non-red α-arc (a, y) with $|N_A((a, y))| \geq 5$. This is a contradiction to the choice of g.

B.9 Proof of Corollary 3

Proof. Given a graph $G(V, A)$ use the construction (a well-known transformation form Directed Feedback Vertex Set to Directed Feedback Arc Set) in appendix A to obtain an instance $G'(V', A')$. Every $v \in V$ has a corresponding arc $a_v \in A'$. Due to the $(1, n)$-property of G', these new introduced arcs will be reduced away by **RR-2**, if we run Algorithm 1 on G'. Thus it suffices to run the Algorithm on G. For every α-arc $(u, v) \notin \mathcal{MAS}$, choose u or v for deletion. The remaining vertices are a minimum feedback vertex set. From $|A| \leq \frac{3}{2}|V|$ follows that Algorithm 1 solves this problem in $\mathcal{O}^*(1.1798^{\frac{3}{2}n})$.
Given a planar graph $G(V, A)$, again use the construction in appendix A to obtain an instance $G'(V', A')$. G' now has the $(1, n)$-property and we have $|A'| \leq 4n$. Thus the running of Algorithm 1 is $\mathcal{O}^*(1.1871^{4n})$.

B.10 Proof of corollary 4

Proof. Consider $G_r(V_r, E_r)$, $r \geq 2$, with $V_r = \{(i, j) \mid 0 \leq i < r, 0 \leq j \leq 7\}$, and E_r contains two types of arcs:
1. $((i, j), (i, (j + 1) \bmod 8)$ for $0 \leq i \leq r$.
2a. $((i, j), ((i + 1) \bmod r, (1 - j) \bmod 8)$ for $0 \leq i < r$ and $j = 1, 2$.
2b. $(((i + 1) \bmod r, (1 - j) \bmod 8, (i, j))$ for $0 \leq i < r$ and $j = 3, 4$.
See figure 10 for an example of the case where $r = 2$. The graph is cubic and has $8r$ vertices, thus it has $12r$ arcs. Its α-arcs are $((i, 0), (i, 1)$ and $((i, 4), (i, 5))$ for $0 \leq i < r$. Since we have to destroy all "rings" as described by the arcs from 1., any feasible solution to this MAS-instance requires r arcs to go into the feedback arc set. However, also r arcs suffice, namely $((0, 4), (0, 5))$ and $((i, 0), (i, 1))$ for $0 < i < r$. This gives the "tight example" as required in [17] to conclude:

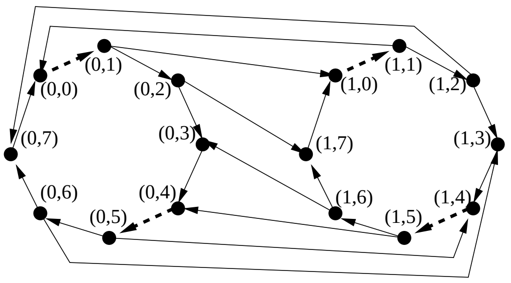

Fig. 10. $G_2(V_r, A_r)$

C Recursions and running times

C.1 The maximum degree three case

In Table 2 we state the 15 recurrences necessary for solving the parameterized version of MAS on maximum degree 3 graphs, which we needed to estimate the run time in theorem 3. They are derived from the positive integer solutions of $i + j + q = 4$. Similarly, Table 3 displays the recurrences for the exact, non-parameterized case (measured in m).

No.	j	q	i	Derived recursion
1	0	0	4	$T[k] \le T[k-4] + T[k-5]$
2	0	1	3	$T[k] \le T[k-(4-\omega)] + T[k-(5-\omega)]$
3	0	2	2	$T[k] \le T[k-(4-2\omega)] + T[k-(5-2\omega)]$
4	0	3	1	$T[k] \le T[k-(4-3\omega)] + T[k-(5-3\omega)]$
5	0	4	0	$T[k] \le T[k-(4-4\omega)] + T[k-(5-4\omega)]$
6	1	0	3	$T[k] \le T[k-(5-\omega)] + T[k-(4+\omega)]$
7	1	1	2	$T[k] \le T[k-(5-2\omega)] + T[k-4]$
8	1	2	1	$T[k] \le T[k-(5-3\omega)] + T[k-(4-\omega)]$
9	1	3	0	$T[k] \le T[k-(5-4\omega)] + T[k-(4-2\omega)]$
10	2	0	2	$T[k] \le T[k-(6-2\omega)] + T[k-(3+2\omega)]$
11	2	1	1	$T[k] \le T[k-(6-3\omega)] + T[k-(3+\omega)]$
12	2	2	0	$T[k] \le T[k-(6-4\omega)] + T[k-3]$
13	3	0	1	$T[k] \le T[k-(7-3\omega)] + T[k-(2+3\omega)]$
14	3	1	0	$T[k] \le T[k-(7-4\omega)] + T[k-(2+2\omega)]$
15	4	0	0	$T[k] \le T[k-(8-4\omega)] + T[k-(1+4\omega)]$

Table 2.

C.2 The $(1, n)$-case

We state Table 5, which covers all recursions derived from integer solutions of $x + y + z = 5$, where x, y and z are the number of occurrences of cases $a.$, b' and $c.$ in Table 4. Table 6 covers all recursions derived from integer solutions of $x + y + z = 6$. Both tables refer to the parameterized version on $\mathcal{MAS}$ on $(1, n)$-graphs. To derive the corresponding tables when we measure in m, simply transform any $T[k] \le T[k-a] + T[k-b]$ to $T[m] \le T[m-(a+1)] + T[m-b]$.

Notice again that these tables are correct upper bounds in particular because we have shown in point 6. of Theorem 2 that we will always find some non-null value still attached to neighbors of arcs we branch on.

No.	j	q	i	Derived recursion
1	0	0	4	$T[m] \leq T[m-5] + T[m-5]$
2	0	1	3	$T[m] \leq T[m-(5-\omega)] + T[m-(5-\omega)]$
3	0	2	2	$T[m] \leq T[m-(5-2\omega)] + T[m-(5-2\omega)]$
4	0	3	1	$T[m] \leq T[m-(5-3\omega)] + T[m-(5-3\omega)]$
5	0	4	0	$T[m] \leq T[m-(5-4\omega)] + T[m-(5-4\omega)]$
6	1	0	3	$T[m] \leq T[m-(6-\omega)] + T[m-(4+\omega)]$
7	1	1	2	$T[m] \leq T[m-(6-2\omega)] + T[m-4]$
8	1	2	1	$T[m] \leq T[m-(6-3\omega)] + T[m-(4-\omega)]$
9	1	3	0	$T[m] \leq T[m-(6-4\omega)] + T[m-(4-2\omega)]$
10	2	0	2	$T[m] \leq T[m-(7-2\omega)] + T[m-(3+2\omega)]$
11	2	1	1	$T[m] \leq T[m-(7-3\omega)] + T[m-(3+\omega)]$
12	2	2	0	$T[m] \leq T[m-(7-4\omega)] + T[m-3]$
13	3	0	1	$T[m] \leq T[m-(8-3\omega)] + T[m-(2+3\omega)]$
14	3	1	0	$T[m] \leq T[m-(8-4\omega)] + T[m-(2+2\omega)]$
15	4	0	0	$T[m] \leq T[m-(9-4\omega)] + T[m-(1+4\omega)]$

Table 3.

α-arc g	$a.$	b'	$c.$
$\mathcal{MAS}$	1	ω	$(1-\omega)$
Deletion	1	1	$(1-\omega)$

Table 4.

No.	x	y	z	Derived recursion
1	5	0	0	$T[k] \leq T[k-5] + T[k-6]$
2	4	1	0	$T[k] \leq T[k-5] + T[k-(5+\omega)]$
3	4	0	1	$T[k] \leq T[k-(5-\omega)] + T[k-(6-\omega)]$
4	3	2	0	$T[k] \leq T[k-5] + T[k-(4+2\omega)]$
5	3	1	1	$T[k] \leq T[k-(5-\omega)] + T[k-5]$
6	3	0	2	$T[k] \leq T[k-(5-2\omega)] + T[k-(6-2\omega)]$
7	2	3	0	$T[k] \leq T[k-5] + T[k-(3+3\omega)]$
8	2	2	1	$T[k] \leq T[k-(5-\omega)] + T[k-(4+\omega)]$
9	2	1	2	$T[k] \leq T[k-(5-2\omega)] + T[k-(5-\omega)]$
10	2	0	3	$T[k] \leq T[k-(5-3\omega)] + T[k-(6-3\omega)]$
11	1	4	0	$T[k] \leq T[k-5] + T[k-(2+4\omega)]$
12	1	3	1	$T[k] \leq T[k-(5-\omega)] + T[k-(3+2\omega)]$
13	1	2	2	$T[k] \leq T[k-(5-2\omega)] + T[k-4]$
14	1	1	3	$T[k] \leq T[k-(5-3\omega)] + T[k-(5-2\omega)]$
15	1	0	4	$T[k] \leq T[k-(5-4\omega)] + T[k-(6-4\omega)]$
16	0	5	0	$T[k] \leq T[k-5] + T[k-(1+5\omega)]$
17	0	4	1	$T[k] \leq T[k-(5-\omega)] + T[k-(2+3\omega)]$
18	0	3	2	$T[k] \leq T[k-(5-2\omega)] + T[k-(3+\omega)]$
19	0	2	3	$T[k] \leq T[k-(5-3\omega)] + T[k-(4-\omega)]$
20	0	1	4	$T[k] \leq T[k-(5-4\omega)] + T[k-(5-3\omega)]$
21	0	0	5	$T[k] \leq T[k-(5-5\omega)] + T[k-(6-5\omega)]$

Table 5.

No.	x	y	z	Derived recursion
1	6	0	0	$T[k] \le T[k-6] + T[k-7]$
2	5	1	0	$T[k] \le T[k-6] + T[k-(6+\omega)]$
3	5	0	1	$T[k] \le T[k-(6-\omega)] + T[k-(7-\omega)]$
4	4	2	0	$T[k] \le T[k-6] + T[k-(5+\omega)]$
5	4	1	1	$T[k] \le T[k-(6-\omega)] + T[k-6]$
6	4	0	2	$T[k] \le T[k-(6-2\omega)] + T[k-(7-2\omega)]$
7	3	3	0	$T[k] \le T[k-6] + T[k-(4+3\omega)]$
8	3	2	1	$T[k] \le T[k-(6-\omega)] + T[k-(5+\omega)]$
9	3	1	2	$T[k] \le T[k-(6-2\omega)] + T[k-(6-\omega)]$
10	3	0	3	$T[k] \le T[k-(6-3\omega)] + T[k-(7-3\omega)]$
11	2	4	0	$T[k] \le T[k-(6)] + T[k-(3+4\omega)]$
12	2	3	1	$T[k] \le T[k-(6-\omega)] + T[k-(4+2\omega)]$
13	2	2	2	$T[k] \le T[k-(6-2\omega)] + T[k-5]$
14	2	1	3	$T[k] \le T[k-(6-3\omega)] + T[k-(6-2\omega)]$
15	2	0	4	$T[k] \le T[k-(6-4\omega)] + T[k-(7-4\omega)]$
16	1	5	0	$T[k] \le T[k-6] + T[k-(2+5\omega)]$
17	1	4	1	$T[k] \le T[k-(6-\omega)] + T[k-(3+3\omega)]$
18	1	3	2	$T[k] \le T[k-(6-2\omega)] + T[k-(4+\omega)]$
19	1	2	3	$T[k] \le T[k-(6-3\omega)] + T[k-(5-\omega)]$
20	1	1	4	$T[k] \le T[k-(6-4\omega)] + T[k-(6-3\omega)]$
21	1	0	5	$T[k] \le T[k-(6-5\omega)] + T[k-(7-5\omega)]$
22	0	6	0	$T[k] \le T[k-6] + T[k-(1+6\omega)]$
23	0	5	1	$T[k] \le T[k-(6-\omega)] + T[k-(2+4\omega)]$
24	0	4	2	$T[k] \le T[k-(6-2\omega)] + T[k-(3+2\omega)]$
25	0	3	3	$T[k] \le T[k-(6-3\omega)] + T[k-4]$
26	0	2	4	$T[k] \le T[k-(6-4\omega)] + T[k-(5-2\omega)]$
27	0	1	5	$T[k] \le T[k-(6-5\omega)] + T[k-(6-4\omega)]$
28	0	0	6	$T[k] \le T[k-(6-6\omega)] + T[k-(7-6\omega)]$

Table 6.