

A Contraction Algorithm for finding Minimal Feedback Sets

Henning Koehler

Massey University, Department of Information Systems
Private Bag 11222, Palmerston North, New Zealand
h.koehler@massey.ac.nz

Abstract

We present a contraction algorithm for finding a minimal set of arcs or vertices that break all cycles in a digraph. The algorithm is (essentially) an extension to the contraction algorithm for the feedback vertex set problem introduced by Levy and Lew (Levy & Lew 1988). The algorithm's time complexity of $O(m \log n)$ is preserved, while allowing both (weighted) feedback vertices and arcs. As the transformation from feedback arc set to feedback vertex set graph increases the graph's size, the new algorithm becomes both faster and more powerful than the original one when applied to feedback arc set problems. We will show that the algorithm works well for reducible flow graphs, as it preserves the structure and can easily be combined with the algorithm proposed by Ramachandran (Ramachandran 1988).

1 Introduction

A *feedback arc set* (FAS) in a directed graph is a set of arcs such that the graph becomes acyclic if all the arcs in the set are removed. Similarly, a *feedback vertex set* (FVS) in a directed graph is a set of vertices such that the graph becomes acyclic if all the vertices in the set (and adjacent arcs) are removed. The challenge is to find a minimal feedback set, that is a set with minimal cardinality in the unweighted case, and minimal total weight in the weighted case. While those problems are also of interest for undirected graphs, we will only consider digraphs here.

The FAS and FVS problems are closely related, and are both NP-complete (Karp 1972). Even, Naor, Schieber and Sudan (Even, Naor, Schieber & Sudan 1998) showed how to perform reductions between the FAS and FVS problem which preserve solutions and their cost. Levy and Low (Levy & Lew 1988) introduced a contraction algorithm for the (unweighted) FVS problem which runs in $O(m \log n)$, graphs for which it finds a minimal feedback set have been named "completely contractible" graphs. While this includes the class of *reducible flow graphs* (RFG), the reduction from FAS to FVS does not preserve the reducibility property, so algorithms that solve the FVS problem on RFGs cannot be used to solve the FAS problem on this class of graphs. The fastest known algorithm for the FAS problem on RFGs is due to Ramachandran (Ramachandran 1988), which is also used for solving the weighted FVS problem on RFGs.

Copyright ©2005, Australian Computer Society, Inc. This paper appeared at the 28th Australasian Computer Science Conference, The University of Newcastle, Newcastle, Australia. Conferences in Research and Practice in Information Technology, Vol. 38. V. Estivill-Castro, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

In section 2 we show how FVS and FAS graphs can be transformed into each other. In section 3 we present a modified version of the original contraction algorithm which offers better worst-case performance when used for solving the FAS problem, and (more importantly) preserves the reducibility property, but does not extend the class of graphs for which it finds a minimal feedback set. We shall then extend this modified version in section 4 by adding new contraction steps and allowing for both feedback vertices and arcs. In section 5 we show how our algorithm can be used in solving the FAS problem on reducible flow graphs. Implementation and time complexity issues will be addressed in section 6. In section 7 we document some experimental results on RFGs and describe the algorithm used for generating them.

2 Reduction between FAS and FVS Problem

We present reductions between the FAS and FVS problem, taken from (Festa, Pardalos & Resende 1999). Both reductions can be performed in linear time.

$[FAS \rightarrow FVS]$: Replace each arc $(v_0 \rightarrow v_2)$ by a new corresponding vertex v_1 and two arcs $(v_0 \rightarrow v_1)$ and $(v_1 \rightarrow v_2)$. The weights assigned to the new vertices are the weights of the corresponding arcs, while the weights of the old vertices are set to infinity. See figure 1 for an example.

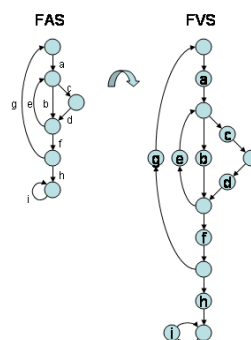


Figure 1: $[FAS \rightarrow FVS]$

$[FVS \rightarrow FAS]$: Replace each vertex v_0 by a new corresponding arc $(v_1 \rightarrow v_2)$ and two vertices v_1, v_2 . The incoming arcs of v_0 become the incoming arcs of v_1 while the outgoing arcs of v_0 become the outgoing arcs of v_2 . The weights assigned to the new arcs are the weights of the corresponding vertices, while the old arcs are set to infinity.

Given those reductions, it is easy to see that every FAS corresponds to a FVS of the same weight and vice versa. Both reductions introduce vertices or arcs with infinite weight, which the contraction algorithms

need to deal with. In the following we shall refer to vertices and arcs with infinite weight as ∞ -vertices and ∞ -arcs.

3 Contraction Algorithms for the FVS Problem

In this section we shall first describe the algorithm by Levy and Low, and then a variation of it, which is (almost) equally powerful. In the following we will refer to those algorithms as *LL-contraction* and *FVS-contraction*. While the LL-contractions were designed for unweighted graphs, they can easily be extended for weighted graphs, and we shall present them in this extended version.

At the base of the algorithms, there are two operations which both remove a vertex v from a graph G .

- *remove*(v): removes v from the graph (along with all adjacent arcs) and adds it to the (partial) feedback set
- ∞ -*remove*(v): removes v from the graph and adds arcs from each predecessor of v to each successor, provided they don't already exist (we don't allow parallel arcs)

We now describe the contraction steps based on those operations. The central idea is that after each step the graph becomes smaller (while the partial feedback set may increase), and every minimal feedback set for the contracted graph can be combined with the partial feedback set to obtain a minimal feedback set of the original graph¹. Both algorithms have the following steps in common:

- *loop*(v): if there exists a loop ($v \rightarrow v$) then *remove*(v)
- *in0*(v): if v has no incoming arcs then ∞ -*remove*(v)
- *out0*(v): if v has no outgoing arcs then ∞ -*remove*(v)
- *in1*(v): if v has only one incoming arc $e : (v' \rightarrow v), v \neq v'$ and $weight(v') \leq weight(v)$ then ∞ -*remove*(v)
- *out1*(v): if v has only one outgoing arc $e : (v \rightarrow v'), v \neq v'$ and $weight(v') \leq weight(v)$ then ∞ -*remove*(v)

LL- and FVS-contraction differ in only a single step, which in either case handles ∞ -vertices. We will call the following step a LL-contraction step, although it is usually performed right after the reduction from a FAS graph and considered part of this reduction.

- ∞ -*reduce*(v): if v has infinite weight then ∞ -*remove*(v)

Figure 2 shows an example LL-contraction.

For FVS-contraction, this is replaced by a different step:

- ∞ -*cycle*(e): if $e : (v \rightarrow v_\infty)$ has an inverse arc ($v_\infty \rightarrow v$) and v_∞ has infinite weight then *remove*(v)

An FVS-reduction for the same example graph (with some contraction steps combined) is given in figure 3.

¹This can easily be verified for each contraction step, as has been done by Levy and Low (Levy & Lew 1988)

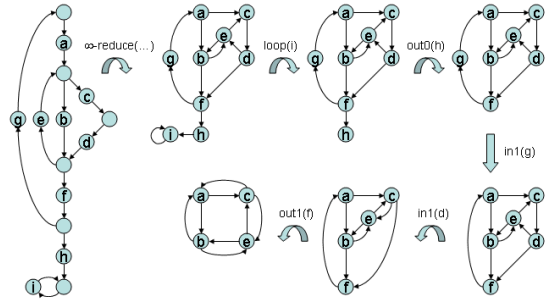


Figure 2: LL-contraction

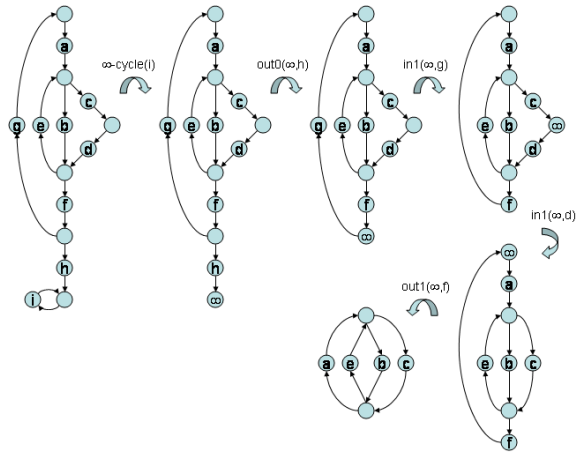


Figure 3: FVS-contraction

The benefit of the ∞ -*reduce* step is the following: When there is a cycle in which only one vertex has finite weight (we shall call this an ∞ -cycle), then this vertex must be in every feedback set. The ∞ -*reduce* step contracts such cycles into a loop which the *loop* contraction can handle. Since the ∞ -vertices can have arbitrary in- and outdegree, and the number of arcs to be added is the product of in- and outdegree, executing the ∞ -*reduce* step in a concrete implementation can be time consuming. Alternatively, we could look for such cycles, but detecting cycles of arbitrary length can be very time consuming as well. However, in every FVS graph that originates from a FAS graph with no ∞ -arcs, there are no adjacent ∞ -vertices and thus all ∞ -cycles have a length of at most two, and can be handled by the *loop* and ∞ -*cycle* steps. Applicability of these steps needs only be checked for each new arc added (or shifted), and each such check can be performed in constant time. Thus the overall complexity is not increased by those steps.

Levy and Low (Levy & Lew 1988) showed that the contraction steps possess the *finite Church-Rosser* (FCR) property, which is to say: if we apply those contraction steps in arbitrary order until no further contractions are possible, we always end up with isomorphic graphs². This is a desirable property, for if there exists a sequence of reduction steps that reduces G to some final reduced graph (preferably the empty one), we can easily find one.

Following (Levy & Lew 1988) we introduce additional notation. Let $\Rightarrow \subset S \times S$ be a relation on a set S . Let $x \Rightarrow y$ denote $(x, y) \in \Rightarrow$. Then we denote the reflexive transitive closure of \Rightarrow by \Rightarrow^* . The completion of \Rightarrow , denoted $\hat{\Rightarrow}$,

²While Levy and Lew only showed this for unweighted graphs, the extension to weighted graphs is straight-forward.

is $\left\{ (x, y) \mid x \xrightarrow{*} y \text{ and there is no } z \text{ with } y \Rightarrow z \right\}$. We write $p_0 \xrightarrow{i} p_i$ if there exists a sequence $p_0 \Rightarrow p_1 \Rightarrow \dots \Rightarrow p_i$. The pair (S, \Rightarrow) is said to be finite if for each p in S , there is a constant k_p such that if $p \xrightarrow{i} q$, then $i \leq k_p$. We say that (S, \Rightarrow) is finite Church-Rosser if it is finite, and $\hat{\Rightarrow}$ is a function, i.e. $p \hat{\Rightarrow} q$ and $p \hat{\Rightarrow} r$ implies $q = r$.

To show that a relation is finite Church-Rosser, the following theorem from (Aho, Sethi & Ullman 1972) is useful:

Lemma 1 *Let \Rightarrow be a relation on set S . Then (S, \Rightarrow) is FCR if and only if it is finite, and for all p in S , if $p \Rightarrow p_1$ and $p \Rightarrow p_2$, then there is some q such that $p_1 \xrightarrow{*} q$ and $p_2 \xrightarrow{*} q$.*

We now show that the FVS-contractions possess the finite Church-Rosser property. Let $GRAPHHS$ be the set of finite directed graphs, and $G \Rightarrow G'$ iff G can be transformed into G' using one of the FVS-contraction steps. We say that two contractions c_1 and c_2 are *in conflict*, if the execution of one prevents the execution of the other, or if the contracted graph varies depending on the order of execution.

Theorem 2 *(GRAPHHS, \Rightarrow) is finite Church-Rosser (FCR).*

Proof: Obviously it is finite since every $G \in GRAPHHS$ is finite and each contraction step reduces the number of vertices. Levy and Low have already shown that \Rightarrow without ∞ -cycle contraction is FCR, so we only need to check cases where this contraction step is involved. We may assume $G \Rightarrow G_1$ is induced by ∞ -cycle(e), $e : (v \rightarrow v_\infty)$ and consider the different cases for $G \Rightarrow G_2$.

- *loop(v'):* If $v' \notin \{v, v_\infty\}$ then there is no conflict, i.e. $G_1 \xrightarrow{loop(v')} G'$ and $G_2 \xrightarrow{\infty\text{-cycle}(e)} G'$. Otherwise, since v_∞ is an ∞ -vertex it can't have a loop, so $v' = v$ and $G_1 = G_2$.
- *in0(v'), out0(v'):* If $v' \notin \{v, v_\infty\}$ then there is no conflict, and $v' \in \{v, v_\infty\}$ can't happen since v and v_∞ have in- and outdegree at least one.
- *in1(v'), out1(v'):* If $v' \notin \{v, v_\infty\}$ then there is no conflict. $v' = v$ can't happen since the predecessor/successor of v would be v_∞ which is of larger weight. If $v' = v_\infty$ then ∞ -remove(v_∞) creates a loop at v while $remove(v)$ reduces the in-/outdegree of v_∞ to zero, and thus $G_1 \xrightarrow{loop(v)} G'$ and $G_2 \xrightarrow{in0/out0(v_\infty)} G'$.
- *∞ -cycle($v' \rightarrow v'_\infty$):* Due to their weights $v' = v_\infty$ or $v'_\infty = v$ is impossible. If $v' \neq v$ then there is no conflict. If $v' = v$ then $G_1 = G_2$. ■

Due to the FCR property, the graph G' resulting from contraction operations on G (until no further contraction is possible) is unique, and we shall call it the LL-contracted or FVS-contracted graph of G , respectively, or simply the *contracted graph* if the contraction algorithm used is clear. We now show that FVS-contraction is (almost) as powerful as LL-contraction.

Lemma 3 *Let G be a vertex-weighted graph with no adjacent ∞ -vertices, and G' obtained by applying any number of LL-contractions or FVS-contractions to G , i.e. $G \xrightarrow{*} G'$. Then G' does not contain adjacent ∞ -vertices either.*

Proof: We have to show that none of the reduction steps introduces an arc between two ∞ -vertices. The only steps that introduce new arcs are *in1*, *out1* and ∞ -reduce. The *in1* contraction has $weight(predecessor(v)) \leq weight(v)$ as precondition, and since they are adjacent and thus can't both have infinite weight, the predecessor must be finite. A similar argument holds for *out1*, and the predecessors and successors of an ∞ -reduced vertex must be finite as well. ■

Theorem 4 *Let G be a vertex-weighted graph with no adjacent ∞ -vertices. Let further G_{LL} be the LL-contracted, and G_{FVS} the FVS-contracted graphs of G . Then G_{FVS} can be transformed into G_{LL} by applying ∞ -reduce to all ∞ -vertices in G_{FVS} . In particular, G_{LL} is empty iff G_{FVS} is empty.*

Proof: It is easy to see that if we combine LL-contraction and FVS-contraction (that is, allow both ∞ -reduce and ∞ -cycle), then the FCR property is preserved and the ∞ -cycle contraction is redundant, i.e. the resulting $(LL \cup FVS)$ -contracted graph is G_{LL} . Thus we only need to show that after applying ∞ -reduce to all ∞ -vertices in G_{FVS} , no further contractions are possible.

Every ∞ -vertex in G_{FVS} has in- and outdegree at least two. Thus the ∞ -reduce steps never reduce the in- or outdegree of their predecessors or successors below two, so no *in0*, *out0*, *in1* or *out1* steps are enabled. Also, due to the lemma above, we know that G_{FVS} contains no ∞ -cycles, and while ∞ -reduce can reduce the length of ∞ -cycles, it can't create new ones. Thus *loop* contraction isn't possible either. ■

4 Extended Contraction Algorithm

In the following we allow for both feedback arcs and feedback vertices. This does not increase the problem scope, as either of the reductions could be used to transform it into a pure FVS or FAS problem, but it allows us to extend our FVS-contractions in an efficient manner.

As arcs can be merged or 'converted' into vertices during the reduction and vice versa, we shall uniformly call them *feedback objects* (FO), and we associate a set $R(o)$ of FOs of the original graph with every FO o , and call them the FOs *represented* by o . The weight of an FO is the total weight of the FOs it represents. Whenever we state that we add an FO to the feedback set, we actually add the FOs represented by it. Since FOs with infinite weight will never be added to the feedback set, we need not keep track of the FOs they represent and thus assign $R(o) := \{\infty\}$ whenever o has infinite weight. At the beginning every FO represents itself, but this can change via $subst(o, \{o_1, \dots, o_n\})$ which assigns $R(o) := R(o_1) \cup \dots \cup R(o_n)$ and $R(o_1), \dots, R(o_n) := \{\infty\}$. If $n = 1$ we will write $subst(o, o_1)$ for $subst(o, \{o_1\})$, while $subst(o, \infty)$ means $R(o) := \{\infty\}$.

We now describe our extended algorithm, which we denote as *FS-contraction*.

- *loop-arc(e):* if $e : (v_\infty \rightarrow v_\infty)$ is a loop and v_∞ an ∞ -vertex then remove e from the graph and add it to the feedback set
- *loop(v):* if v contains a loop of infinite weight then remove v from the graph and add it to the feedback set
- *in0(v):* if v has no incoming arcs then remove v from the graph

- $out0(v)$: if v has no outgoing arcs then remove v from the graph
- $in1 - fin(v)$: if v has only one incoming arc e and $weight(e) < weight(v)$ then $subst(v, e)$
- $out1 - fin(v)$: if v has only one outgoing arc e and $weight(e) < weight(v)$ then $subst(v, e)$
- $in1 - inf(v)$: if v has only one incoming arc $e : (v' \rightarrow v), v \neq v', e$ has infinite weight and $weight(v') \leq weight(v)$ then remove v and e from the graph and move all outgoing arcs of v to become outgoing arcs of v'
- $out1 - inf(v)$: if v has only one outgoing arc $e : (v \rightarrow v'), v \neq v', e$ has infinite weight and $weight(v') \leq weight(v)$ then remove v and e from the graph and move all incoming arcs of v to become incoming arcs of v'
- $\infty - cycle(e)$: if $e : (v \rightarrow v_\infty)$ has an inverse arc $e' : (v_\infty \rightarrow v)$ and e, e' and v_∞ have infinite weight then remove v from the graph and add it to the feedback set
- $\infty - mark(e)$: if $e : (v_1 \rightarrow v_2)$ has finite weight such that $weight(e) \geq weight(v_1)$ or $weight(e) \geq weight(v_2)$ then $subst(e, \infty)$
- $io1(v)$: if v has only one incoming arc $(v_1 \rightarrow v)$ and one outgoing arc $(v \rightarrow v_2)$, both of infinite weight, then remove v from the graph and add an arc $e : (v_1 \rightarrow v_2)$ instead, and $subst(e, v)$
- $parallel(e_1, e_2)$: if $e_1 : (v_1 \rightarrow v_2)$ and $e_2 : (v_1 \rightarrow v_2)$ are parallel arcs then $subst(e_1, \{e_1, e_2\})$ and remove e_2 from the graph.

As lemma 7 demonstrates, the new steps introduced (with the exception of $io1$ and $parallel$) correspond to FVS-steps, modified to apply to feedback arcs, and do not make the algorithm more powerful. The $io1$ step is a more technical tool which makes the transformations from FS to FVS graphs reversible (i.e. the transformed graph can be $io1$ -contracted back to the original graph) and can enable the $parallel$ step. The $parallel$ step makes the contraction more powerful: as figure 4 shows, the example graph from earlier can be contracted completely using FS-contraction.

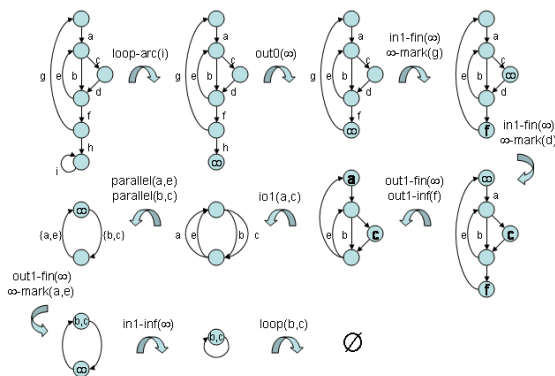


Figure 4: FS-contraction

In order to compare FS-contractions with FVS-contractions, we modify our definition of the FVS-contractions slightly by treating removal of extra parallel arcs as an extra step:

- $\infty - par(e_1, e_2)$: if $e_1 : (v_1 \rightarrow v_2)$ and $e_2 : (v_1 \rightarrow v_2)$ are parallel ∞ -arcs then remove e_2 from the graph.

For this, every arc in a FVS-graph is considered an ∞ -arc. We further define a weaker version of our FS-contraction, denoted WFS -contraction, which contains all FS-contractions except $parallel$ which is replaced by the weaker $\infty - par$ step. Since WFS-contraction contains all steps of FVS-contraction, it is at least as powerful. As we will show, WFS-contraction is actually equivalent to FVS-contraction, so the only improvement comes from merging parallel arcs. Since WFS-contractions are defined on graphs which allow weighted arcs while FVS-contractions require graphs with unweighted (∞ -weighted) arcs, we describe a transformation $T : FS \rightarrow FVS$ which converts the former graphs into the latter, which is essentially an extension to the reduction $[FAS \rightarrow FVS]$.

$[T : FS \rightarrow FVS]$: Replace each arc $e : (v_0 \rightarrow v_2)$ with finite weight by a new corresponding vertex v_1 and two arcs $(v_0 \rightarrow v_1)$ and $(v_1 \rightarrow v_2)$, and $subst(v_1, e)$.

If no WFS-contractions can be applied to a graph, we shall call it WFS -complete, and analogous for other contraction algorithms.

Lemma 5 *If graph G is WFS -complete, then $T(G)$ is FVS -complete.*

Proof: Assume the contrary, that is a FVS-contraction step can be applied to $T(G)$. If the vertex/arc it is applied to already existed in G , it is easily seen that the same contraction step could be applied to G , which means G would not be WFS -complete. So the vertex/arc must have been introduced by T . Since newly introduced vertices have no loops and in- and outdegree of one, $loop, in0, out0$ are impossible. If $in1/out1$ could be applied to v , then the weight of its predecessor/successor must be $\leq weight(v)$, and thus $\infty - mark$ could be applied to $T^{-1}(v)$ in G . If $\infty - cycle$ could be applied to $e : (v \rightarrow v_\infty)$ in $T(G)$, then $loop - arc$ could be applied to $T^{-1}(v) : (v_\infty \rightarrow v_\infty)$. That covers all cases. ■

Lemma 6 *Let G_1 and G_2 be WFS -complete graphs. If $T(G_1) = T(G_2)$ then $G_1 = G_2$.*

Proof: Let V be the vertex set of $T(G_1)$ and $V_1, V_2 \subset V$ be the sets of vertices newly introduced by applying T to G_1 and G_2 , respectively. To show that $V_1 = V_2$, which would prove the lemma, we first note that $io1$ can be applied to each $v \in V_1 \cup V_2$. It therefore suffices to show that $io1$ cannot be applied to any vertex $v_0 \notin V_1 \cap V_2$. Without loss of generality $v_0 \notin V_1$. Since T does not change the in- or outdegree of any vertex, v_0 has only one incoming and outgoing arc e_{in}, e_{out} in G_1 as well. But then, depending on the weights of v_0, e_{in} and e_{out} , either $in1/out1 - fin(v_0), \infty - mark(e_{in}/e_{out})$ or $io1(v_0)$ could be applied to G_1 , which contradicts the assumption that G_1 be WFS -complete. ■

Lemma 7 *If graph G can be WFS -contracted into G' , then $T(G)$ can be FVS -contracted into $T(G')$.*

Proof: It is sufficient to show that this holds for each single WFS-contraction step. We do this by simply providing the corresponding FVS-contractions. It is easily seen that the lemma holds in each case.

WFS-contraction	\rightsquigarrow	FVS-contraction
$loop(v)$		$loop(v)$
$loop - arc(e)$		$\infty - cycle(T(e))$
$\infty - cycle(e)$		$\infty - cycle(e)$
$in0/out0(v)$		$in0/out0(v)$
$in1/out1 - fin(v)$		$in1/out1(v)$
$in1/out1 - inf(v)$		$in1/out1(v)$
$\infty - mark(e)$		$in1/out1(T(e))$
$io1(v)$		\emptyset
$\infty - par(e_1, e_2)$		$\infty - par(e_1, e_2)$

We can now show:

Theorem 8 (a) *WFS-contractions is finite Church-Rosser.*

(b) Let G_{WFS} be the WFS-contracted graph of G and G_{FVS} the FVS-contracted graph of $T(G)$. Then $T(G_{WFS}) = G_{FVS}$.

Proof: Let G be WFS-contractible to the WFS-complete graphs G_1 and G_2 . Then $T(G)$ can be FVS-contracted into the FVS-complete graphs $T(G_1)$ and $T(G_2)$. Since FVS-contraction is FCR, $T(G_1) = T(G_2) = G_{FVS}$ and thus $G_1 = G_2 = G_{WFS}$, which proves the theorem. ■

While part (b) is the result we were after in the first place, as it makes clear where the advantage of FS-contraction lies compared to FVS-contraction (and thus compared to LL-contraction as well), part (a) is useful in showing that FS-contraction is FCR as well. The proof follows the same schema as theorem 2, and we will use the same notation as there (with $G \Rightarrow G'$ now indicating that G' can be derived from G using a FS-contraction step).

Theorem 9 *FS-contraction is finite Church-Rosser.*

Proof: We get FS-contraction by adding the *parallel* step to WFS-contraction and removing $\infty - par$. Since $\infty - par$ is redundant, its removal needs not concern us. Since WFS-contraction is FCR, we only need to check the cases where the new *parallel* step is involved. We may assume $G \Rightarrow G_1$ is induced by *parallel*(e_1, e_2) for $e_1, e_2 : (v_1 \rightarrow v_2)$ and consider the different cases for $G \Rightarrow G_2$. The new arc resulting from *parallel*(e_1, e_2) shall be denoted as e_{new} .

- *loop - arc*(e): If $e \notin \{e_1, e_2\}$ then there is no conflict. If $e \in \{e_1, e_2\}$, say $e = e_1$, then $G_1 \xrightarrow{loop-arc(e_{new})} G'$ and $G_2 \xrightarrow{loop-arc(e_2)} G'$.
- *loop*(v): If $v \notin \{v_1, v_2\}$ then there is no conflict. Otherwise $G_1 \xrightarrow{loop(v)} G_2$.
- *in0/out0*(v): If $v \notin \{v_1, v_2\}$ then there is no conflict. Otherwise $G_1 \xrightarrow{in0/out0(v)} G_2$.
- *in1/out1 - fin*/ ∞ (v): e_1, e_2 cannot be the single incoming/outgoing arc, thus there is no conflict.
- $\infty - cycle$ ($e : v \rightarrow v_\infty$): If $v \notin \{v_1, v_2\}$ then there is no conflict. Let $v \in \{v_1, v_2\}$. If $e \notin \{e_1, e_2\}$ then $G_1 \xrightarrow{\infty-cycle(e)} G_2$, otherwise $G_1 \xrightarrow{\infty-cycle(e_{new})} G_2$.
- $\infty - mark$ (e): If $e \notin \{e_1, e_2\}$ then there is no conflict. Otherwise $G_1 \xrightarrow{\infty-mark(e_{new})} G'$ and $G_2 \xrightarrow{parallel(e_1, e_2)} G'$.

- *io1*(v): Since $v \notin \{v_1, v_2\}$ there is no conflict.
- *parallel*(e_3, e_4): If $\{e_3, e_4\} \cap \{e_1, e_2\} = \emptyset$ then there is no conflict. If $\{e_3, e_4\} = \{e_1, e_2\}$ then $G_1 = G_2$. Otherwise, say $e_3 = e_1$, $G_1 \xrightarrow{parallel(e_4, e_{new})} G'$ and $G_2 \xrightarrow{parallel(e_2, e_{new})} G'$. ■

5 Reducible Flow Graphs

Most classes of graphs for which polynomial time algorithms for solving the FVS/FAS problem are known are artificial classes, i.e. defined by the algorithm solving the problem. One exception is the class of reducible flow graphs (RFGs), which (despite the name) arises naturally in the modelling of the control structure of computer programs³. The unweighted FVS problem on those graphs can be solved in linear time (Shamir 1979), and they are completely LL-contractible (i.e. can be LL-contracted to the empty graph) as well. While the FAS problem can be reduced to a FVS problem, the reduction does not preserve the RFG property (or, more precisely, the $\infty - reduce$ step doesn't preserve it), so those algorithms cannot be applied to solve the FAS problem on RFGs. The current state of the art for this problem class is the approach introduced by Ramachandran (Ramachandran 1988), who showed that the FAS problem on RFGs can be reduced to the minimum cut problem in flow networks, such that the known polynomial-time algorithms for this problem can be applied. This leads to a time complexity of $O(\min(mn^{5/3}, m^2))$ in the unweighted, and $O(mn^2 \log(n^2/m))$ in the weighted case.

While not all RFGs are FS-contractible, we will show that FS-contraction "works well" for contracting RFGs, and that the contracted graphs are "very similar" to RFGs such that Ramachandran's algorithm can be applied to them. Since FS-contraction is very fast, this should provide a significant speed improvement for most graphs. We introduce the following contraction steps which will be useful in the definition of graph classes and are very similar to some of the FS-contractions:

- $T_1(e)$: If $e : (v \rightarrow v)$ is a loop then remove it ($\sim loop - arc$)
- $T_2(e)$: If $e : (v_1 \rightarrow v_2)$ is the only incoming arc of $v_2 \neq v_1$ then remove e and v_2 and shift the outgoing arcs of v_2 to v_1 ($\sim in1$)
- $T_3(e)$: If $e : (v_1 \rightarrow v_2)$ is the only outgoing arc of $v_1 \neq v_2$ then remove e and v_1 and shift the incoming arcs of v_1 to v_2 ($\sim out1$)
- *par*(e_1, e_2) : If $e_1, e_2 : (v_1 \rightarrow v_2)$ are parallel arcs then remove e_2 ($\sim parallel$)
- *in0*(v) : If v has no incoming arcs then remove it
- *sub*(e/v) : Remove e or v from the graph

For any set C of contraction steps we will call a graph C -contractible iff it can be contracted to a single vertex (or the empty graph) by applying contractions from C . Using this notation, we define several graph classes.

³RFGs can be characterized as having a "root" vertex such that every vertex is reachable (\rightsquigarrow directed path) from the root, and every cycle contains a single "entry vertex", i.e. removal of that vertex makes all vertices in the cycle unreachable. This corresponds to disallowing jumps into the middle of a loop with "GOTO" statements.

Weakly Reducible Flow Graph	(WRFG)
Reducible Flow Graph	(RFG)
Weakly Reducible Graph	(WRG)
Strongly Reducible Graph	(SRG)
Component Reducible Flow Graph	(CRFG)

The definition for RFGs is equivalent to the one given in (Hecht & Ullman 1974), with the difference being that the removal of parallel arcs is made explicit. The *maximal strongly connected components* (MSCs) of a graph are the maximal subgraphs such that there is a directed path between each pair of vertices (in both directions).

WRFG	$:\Leftrightarrow$	$\{T_1, T_2\}$ -contractible
RFG	$:\Leftrightarrow$	$\{T_1, T_2, par\}$ -contractible
WRG	$:\Leftrightarrow$	$\{T_1, T_2, in0\}$ -contractible
SRG	$:\Leftrightarrow$	$\{T_1, T_2, in0, par\}$ -contractible
CRFG	$:\Leftrightarrow$	every MSC is a RFG

While not all RFGs were FS-contractible, we will show a weaker result, namely that all unweighted WRFGs are WFS-contractible. Since the only contraction step lacking in the definition of WRFGs towards RFGs is the *par* step, and FS-contraction merges all parallel arcs (although this destroys the unweighted property), it seems likely that a large number of RFGs are FS-contractible, or at least can be contracted significantly. Our second goal is to show that the FS-contracted graphs of RFGs are CRFGs. Thus when FS-contraction doesn't contract the graph completely, we can partition the remaining graph into its MSCs (which requires only linear time (Tarjan 1972) and preserves minimal feedback sets) and then apply Ramachandran's algorithm to solve the feedback set problems on them.

Lemma 10 *The contraction systems $\{T_1, T_2, in0\}$, $\{T_1, T_2, in0, par\}$, $\{T_3\}$, $\{sub\}$ are finite Church-Rosser.*

Proof: Easy to show using lemma 1. ■

Let \Rightarrow_1 and \Rightarrow_2 be two relations on a set S . We say that \Rightarrow_1 and \Rightarrow_2 *commute*, if for all $p, p_1, p_2 \in S$ with $p \Rightarrow_1^* p_1$ and $p \Rightarrow_2^* p_2$ there exists $q \in S$ such that $p_1 \Rightarrow_2^* q$ and $p_2 \Rightarrow_1^* q$.

Lemma 11 *Let \Rightarrow_1 and \Rightarrow_2 be two commuting relations. If $p \Rightarrow_1^* p_1$, $p \Rightarrow_2^* p_2$ and p_1 is \Rightarrow_2 -complete (i.e. there is no q with $p_1 \Rightarrow_2 q$), then $p_2 \Rightarrow_1^* p_1$.*

Proof: Follows directly from the definition of commutativity ($q = p_1$). ■

The graph classes WRG and SRG have been defined as being contractible to the empty graph using certain contractions (\Rightarrow_1). Since the empty graph (p_1) cannot be contracted any further, the above lemma can be used to show that the WRG and SRG properties are invariant under other contractions (\Rightarrow_2), provided that the contraction sets commute. To show commutativity we need the following lemma, taken from (Rosen 1973, Lemma 3.6). In this, \Rightarrow^- denotes the reflexive closure of \Rightarrow .

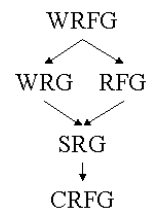
Lemma 12 (*Commutativity Lemma*) *Let \Rightarrow_1 and \Rightarrow_2 be two relations on a set S . If for all $p, p_1, p_2 \in S$ with $p \Rightarrow_1 p_1$ and $p \Rightarrow_2 p_2$ there exists $q \in S$ such that $p_1 \Rightarrow_2^* q$ and $p_2 \Rightarrow_1^- q$, then \Rightarrow_1 and \Rightarrow_2 commute⁴.*

⁴For finite relations this holds with \Rightarrow_1^* instead of \Rightarrow_1^- .

Lemma 13 *The contraction systems $\{T_1, T_2, in0\}$ and $\{T_1, T_2, in0, par\}$ commute with $\{T_3\}$ and $\{sub\}$.*

Proof: Easy to show using lemma 12. ■

Lemma 14 *The implication hierarchy of the graph classes above is the following:*



Proof: The only implication which is not immediately obvious is $SRG \Rightarrow CRFG$, thus we only need to show that every MSC of a SRG is a RFG. Since $\{T_1, T_2, in0, par\}$ commutes with $\{sub\}$, every subgraph of a SRG is again a SRG by lemma 11, i.e. every MSC can be contracted to a single vertex using $\{T_1, T_2, in0, par\}$. In a strongly connected graph with more than one vertex, every vertex has positive in- and outdegree, thus *in0* can't be applied. The contraction steps $\{T_1, T_2, par\}$ preserve strong connectivity, so *in0* won't be used, i.e. every MSC can be reduced to a single vertex using only $\{T_1, T_2, par\}$, i.e. is a RFG. ■

Using the implication hierarchy established above, we only need to prove that the SRG property is invariant under FS-contractions to reach our secondary goal of showing that the FS-contracted graph of a RFG is a CRFG. We will do that next.

Lemma 15 *The WRG and SRG properties are invariant under FS-contraction*

Proof: Since $\{T_1, T_2, in0\}$ and $\{T_1, T_2, in0, par\}$ commute with $\{T_3\}$ and $\{sub\}$, WRG and SRG are invariant under T_3 and *sub* by lemma 11, and trivially under T_2 . Now every applied FS-contraction step which changes the graph structure is equivalent to one of T_2, T_3, sub , and hence preserves WRG and SRG properties. ■

We are now ready for our main theorem.

Theorem 16 *Every unweighted WRG is WFS-contractible. In particular, every unweighted WRFG is WFS-contractible.*

Proof: For this proof we shall refer to an arc as *1-arc*, iff it is the only incoming arc for its target vertex (i.e. T_2 could be applied). We will use the following invariance properties:

- (I₁) All vertices have positive in- and outdegree.
- (I₂) No loops $e : (v \rightarrow v)$ exist.
- (I₃) There are no (directed) paths of 1-arcs of length 2.
- (I₄) 1-arcs have no inverse arcs.

Any non-empty graph G' for which I_1 and I_2 hold must contain at least 2 vertices. Thus any contraction algorithm which removes at most one vertex each step and for which I_1 and I_2 are invariant cannot contract G' to the empty graph. Now let G be any unweighted WRG and G_{WFS} its WFS-contracted graph. By lemma 15 G_{WFS} is a WRG, i.e. can be $\{T_1, T_2, in0\}$ -contracted to the empty graph. We have to show that G_{WFS} is the empty graph, and we will do so by proving that $I_1 - I_4$ (in particular I_1 and I_2)

hold for G_{WFS} and are invariant under $\{T_1, T_2, in0\}$ -contraction.

We first show that $I_1 - I_4$ hold for G_{WFS} . Since WFS -contraction doesn't introduce new weights (other than ∞), G_{WFS} is still unweighted, i.e. all arcs and vertices have weight 1 or ∞ . Since G_{WFS} is WFS -complete, all 1-arcs and their source vertex have infinite weight while their target vertex has weight 1 (otherwise $in1 - fin$ or $in1 - inf$ could be applied).

- (I_1): Otherwise $in0$ or $out0$ could be applied.
- (I_2): If $weight(v) = \infty$ then $loop - arc(e)$ could be applied, so let $weight(v) = 1$. If $weight(e) = 1$ we could apply $\infty - mark(e)$, otherwise $weight(e) = \infty$ and thus $loop(v)$ could be applied.
- (I_3): Let $v_1 \rightarrow v_2 \rightarrow v_3$ be a path of 1-arcs. Then v_2 would be both source and target of a 1-arc which is a contradiction since its weight can't be both 1 and ∞ .
- (I_4): Let $e : (v_1 \rightarrow v_2)$ be a 1-arc and $e' : (v_2 \rightarrow v_1)$ its invers. Since e is a 1-arc, $weight(e) = weight(v_1) = \infty$ and $weight(v_2) = 1$. Thus we could apply either $\infty - mark(e')$ or $\infty - cycle(e')$.

Next we have to show that $I_1 - I_4$ are invariant under $\{T_1, T_2, in0\}$ -contraction. Due to I_1 and I_2 neither T_1 nor $in0$ can be applied, so we may restrict ourselves to T_2 . Let $e : (v_1 \rightarrow v_2)$ be the 1-arc which T_2 is applied to.

- (I_1): Obvious.
- (I_2): The only arcs added (shifted) are the outgoing arcs of v_2 . Due to I_4 they don't have v_1 as incoming arc, i.e. aren't loops.
- (I_3): The only new paths of length 2 are those that use one of the new (shifted) arcs. Due to I_3 those aren't 1-arcs.
- (I_4): The only new 2-cycles are those that use one of the new (shifted) arcs. Due to I_3 those aren't 1-arcs. Their inverse arcs are incoming arcs of v_1 and again due to I_3 those aren't 1-arcs. ■

6 Implementation & Time Complexity

Levy and Lew have shown that LL-contraction without $\infty - reduce$ on unweighted graphs can be implemented to run in $O(m \log n)$, where m and n are the number of arcs and vertices, respectively. However, the transformation $FAS \rightarrow FVS$ combined with $\infty - reduce$ enlarges the size of the graph. Let G be the original FAS -graph and $G' := T_\infty(G)$ the transformed and $\infty - reduced$ graph of G with parameters m', n' , which we now determine. Obviously $n' = m$, and the average in- and out-degree of a vertex in G is $\frac{m}{n}$. Thus we have $m' \approx \left(\frac{m}{n}\right)^2 n = \frac{m^2}{n}$ as an average over all graphs, while in the worst case (where some vertices have maximal in- and outdegree n) we get $m' \approx n^2 \frac{m}{n} = nm$. The overall complexity of LL-contraction for (unweighted) FAS -graphs therefore becomes $O\left(\frac{m^2}{n} \log m\right)$ in the average, and $O(nm \log m)$ in the worst case.

We now show that FS -contraction can be implemented to run in $O(m \log n)$. An implementation of a contraction algorithm consists of two parts: the identification of a vertex/arc for which a reduction step can be applied, and the execution of such a reduction step. Let $s \leq n + 2m \in O(m)$ be the number of vertices plus the number of arcs plus the number of *finite*

arcs. Each FS -reduction step decreases s by one or more, and thus at most s steps are necessary. We assume that the graph G is represented in a form which allows us to perform the following operations in constant time: check the in- and outdegree of a vertex, retrieve an arc with given source and target vertex, remove or add an arc, remove an isolated vertex.

It is easily seen that the only critical FS -contraction steps are $in1, out1$ since for all other steps the number of operations performed is linear in the amount by which they reduce s . What makes $in1, out1$ critical is that they may require shifting of a large number of arcs while only decreasing s by two or three. For each such step where $v_1 \rightarrow v_2$ is the single incoming or outgoing arc, we have the choice whether to shift the arcs of v_1 to v_2 or vice versa. Levy and Lew showed that by always shifting the arcs of the less often shifted vertex⁵, no arc gets shifted more than $\log_2(n)$ times, such that an overall worst-case complexity of $O(m \log n)$ can be achieved.

As we do not want to check the entire graph for applicable contraction steps whenever it changes, we keep a collection C of contraction steps (plus the vertices/arcs to apply them to) to be checked and executed. For each vertex/arc, the check whether a particular contraction step can be applied to this vertex/arc can be performed in constant time. We want C to contain (at least) all applicable contractions, thus C gets initialized with all contraction steps applied to all vertices/arcs. Then, while C is not empty, we select one of the contractions in C and check whether it is applicable. If so we execute it, which involves the removal, addition and shifting of arcs, the removal of isolated vertices⁶, and the substitution of associated sets.

The execution of a contraction step might enable other contractions, so we need to update C . As the FS -contraction steps only require local properties (i.e. properties of adjacent arcs and vertices) to hold, we only need to check for contractions that apply to vertices/arcs local (adjacent) to the vertex or arc that got removed, added, shifted⁷ or weight-modified. Weight-modifications can be performed by the $in1 - fin, out1 - fin, \infty - mark$ and $parallel$ steps, or when the execution of an $in1 - inf(v)$ or $out1 - inf(v)$ step keeps the vertex v in the graph and substitutes its associated set with the associated set of its predecessor/successor (to avoid shifting the arcs of v). In the following, we list the contractions to be added to C for each basic operation on the graph⁸.

$remove(v_{isolated}) : -$
 $remove(v_1 \rightarrow v_2) :$
 $out0(v_1), in0(v_2),$
 $out1 - inf(v_1), in1 - inf(v_2),$
 $out1 - fin(v_1), in1 - fin(v_2),$
 $io1(v_1), io1(v_2)$
 $add(e : (v_1 \rightarrow v_2)) :$
 $loop - arc(e), loop(v_1)$
 $\infty - cycle(e), \infty - mark(e),$
 $parallel(e, \dots)$
 $increase - weight(e : (v_1 \rightarrow v_2)) :$
 $loop(v_1),$
 $out1 - inf(v_1), in1 - inf(v_2)$
 $io1(v_1), io1(v_2),$
 $\infty - cycle(e), \infty - mark(e)$

⁵Obviously it would be even better to shift the vertex with lower degree.

⁶We remove non-isolated vertices by removing adjacent arcs first.

⁷For the purpose of detecting enabled contractions we will treat the shifting of an arc as removing it and then adding a new one.

⁸Note that the weight of a vertex never increases while the weight of an arc never decreases.

$decrease - weight(v)$:
 $in1 - inf(successors(v))$,
 $out1 - inf(predecessors(v))$,
 $\infty - mark(adjacent - arcs(v))$

The critical cases here are those where the number of contractions to be added to C is not bounded by a constant, i.e. $add(e)$ and $decrease - weight(v)$. By executing *parallel* steps in C first, we can ensure that the set of contractions *parallel* (e, \dots) contains at most one element. The contractions added by $decrease - weight(v)$ require a more complicated handling⁹. For that we create a fibonacci heap (Fredman & Tarjan 1987) at each vertex v which stores its adjacent *critical* arcs. An arc e between v and a vertex u is critical (w.r.t. v) if a decrease in the weight of v could enable a contraction on it, i.e. if either $weight(e) < weight(v)$ ($\rightsquigarrow \infty - mark$), or $weight(e) = \infty$, e is the only incoming/outgoing arc of u , and $weight(u) < weight(v)$ ($\rightsquigarrow in/out1 - inf$).

The benefit of this heap is that it allows us to quickly identify all newly enabled contractions when the weight of v gets decreased. We can do so by checking the the critical arcs in order of their weight, starting with the highest-weight arc¹⁰. As soon as a check fails (i.e. an arc's weight is lower than the new weight of v) we can abort, knowing that all further checks would fail as well. Consequently only one unsuccessful check is performed, and the overall number of checks is proportional to the number of contractions executed, such that the time complexity is not increased.

We show that updates to the heaps only add a cost of $O(m \log n)$ by looking at the basic graph operations performed¹¹. Arcs are removed, added or weight-increased at most $O(m)$ times, and adding and removing elements to a fibonacci heap can be performed (amortized) in $O(\log n)$. Shifting of arcs due to $in/out1 - inf$ contractions require the merging of two heaps, which can be performed in constant time. Finally, arcs can become critical or non-critical (\rightsquigarrow infinite arcs) when the weight of one of their adjacent vertices decreases. Each time an arc becomes non-critical in that way, a new contraction is enabled, which happens at most $O(m)$ times (and can be detected along with the checking for new contractions). Infinite arcs can only become critical (w.r.t. the unchanged vertex u) when they are the only incoming/outgoing arc of the weight-decreased vertex, so each such weight-decrease makes at most one arc critical. At most $O(n)$ vertex weight-decreases can occur.

7 Experimental Results

We implemented the LL- and FS-contraction algorithms (without use of Fibonacci heaps) and tested them on some randomly generated arc-weighted RFGs. We considered RFGs of different sizes and densities, both weighted and unweighted (i.e. all arcs have the same weight). To apply LL-contraction to those graphs, we transformed them via $[FAS \rightarrow FVS]$. Since the $\infty - reduce$ step destroys the RFG property, making the contracted problem much harder to solve, we also considered LL-contraction without this step and denote it as WLL-

contraction. For each of the 12 test cases we randomly generated 1000 graphs, and applied FS-, LL- and WLL-contraction to them. As results we list the average number of vertices, arcs and finite feedback objects in the contracted graphs, as well as the number of complete contractions (i.e. where contracted graph = \emptyset) and average time taken¹².

Alg.	vert.	arcs	FOs	\emptyset	time
100/200 unweighted					
FS	2	4	2	814	5ms
LL	3	11	3	728	9ms
WLL	86	145	62	0	5ms
100/200 weighted					
FS	22	49	36	26	4ms
LL	38	108	38	7	9ms
WLL	105	166	81	0	4ms
100/500 unweighted					
FS	20	78	49	529	15ms
LL	52	268	52	502	44ms
WLL	578	975	482	0	5ms
100/500 weighted					
FS	84	339	291	0	8ms
LL	292	1378	292	0	28ms
WLL	584	980	488	0	5ms
1000/2000 unweighted					
FS	5	15	6	759	67ms
LL	9	48	9	643	131ms
WLL	864	1470	629	0	72ms
1000/2000 weighted					
FS	228	542	385	0	57ms
LL	397	1421	397	0	112ms
WLL	1065	1686	826	0	67ms
1000/5000 unweighted					
FS	125	509	289	454	179ms
LL	312	1838	312	410	769ms
WLL	5750	9714	4794	0	77ms
1000/5000 weighted					
FS	845	3429	2945	0	77ms
LL	2966	16149	2966	0	410ms
WLL	5828	9792	4872	0	75ms
10000/20000 unweighted					
FS	11	31	13	688	999ms
LL	19	160	19	539	2.24s
WLL	8678	14789	6320	0	951ms
10000/20000 weighted					
FS	2266	5431	3843	0	847ms
LL	3951	16643	3951	0	1.78s
WLL	10603	16792	8234	0	871ms
10000/50000 unweighted					
FS	666	2768	1517	365	3.75s
LL	1623	10074	1623	328	26.9s
WLL	57428	97071	47888	0	1.01s
10000/50000 weighted					
FS	8454	34376	29532	0	1.23s
LL	29672	177839	29672	0	7.59s
WLL	58226	97867	48690	0	1.10s

As predicted by theorem 16, FS-contraction works quite well for unweighted RFGs, although it performs far worse in the weighted case. Our results also illustrate the dilemma of the $\infty - reduce$ step (which we solved by introducing $\infty - cycle$): it destroys the graph's structure, but at the same time it is vital to achieve good contraction ratios. While the FS-contracted graphs are significantly smaller than the LL-contracted ones, the number of finite FOs in them is only slightly less. The runtime increases almost linearly with the graph size.

A result that was unexpected is the observation that in the unweighted case, the ratio by which FS- and LL-contraction contract the graph becomes sig-

⁹At most $O(n)$ vertex shifts and vertex weight-decreases occur, and the average degree of a vertex is $\frac{m}{n}$. Thus we get an average complexity of $O(m)$ for both contraction step detection and execution, which makes a simpler implementation without special handling preferable for most applications.

¹⁰The heap-weight associated with an infinite critical arc e is the weight of its opposing vertex u .

¹¹For this we will *not* treat the shifting of arcs as removal and re-adding.

¹²The test was performed on a 1.8 GHz PC.

nificantly better with increasing graph size. An explanation for this might be found in the way the test graphs were generated, the algorithm for which is given next.

Using the terminology of (Hecht & Ullman 1974), we say that a vertex v_1 *dominates* v_2 iff every path from the root to v_2 contains v_1 . A *dag* of a flow graph is a maximal acyclic sub-flow graph. By (Hecht & Ullman 1974, Theorem 6) a flow graph is reducible iff it can be (uniquely) split into a dag and a set of back-arcs, such that for each back-arc $v_2 \rightarrow v_1$ the target v_1 dominates the source v_2 in the dag.

Any dag of a RFG can be interpreted as a partial order on the vertex set (with $v_1 < v_2$ iff there is a path from v_1 to v_2) which can be linearized, i.e. the vertices can be identified with integers $0 \dots n$ such that an arc $v \rightarrow v'$ is part of the dag iff $v < v'$ (and a back-arc otherwise). The central idea for constructing RFGs is to prescribe such a linear ordering and then randomly add arcs to it without violating the domination condition for back-arcs. In order to determine whether a vertex dominates another, we will use a "pre-dominator tree". For any RFG we define the *pre-dominator* of a vertex $v \neq \text{root}$ as the largest vertex $v' \neq v$ that dominates v (the pre-dominator tree then contains the arcs *pre-dominator* (v) $\rightarrow v$). The *max-dominator* of a (non-empty) set of vertices is the largest vertex that dominates them all.

We now describe the algorithm in pseudocode:

```

Input:   $n, da, ba$ 
Output: RFG with  $n+1$  vertices,  $n+da$  dag arcs
        and  $ba$  back arcs

add vertices  $0 \dots n$  to the graph
for  $t = 1 \dots n$  do
     $s := \text{random}(0 \dots t-1)$ 
    add arc  $s \rightarrow t$ 
     $\text{pre\_dom}[t] := s$ 
repeat  $da$  times
    add random arc  $s \rightarrow t$  with  $s < t$ 
     $\text{pre\_dom}[t] := \text{max\_dom}(\text{pre\_dom}[t], s)$ 
repeat  $ba$  times
     $s := \text{random}(1 \dots n)$ 
     $t := \text{random\_dom}(s)$ 
    add arc  $s \rightarrow t$ 

 $\text{max\_dom}(a, b)$ 
    while  $a \neq b$  do
        if  $a < b$  then  $b := \text{pre\_dom}[b]$ 
        else  $a := \text{pre\_dom}[a]$ 
    return  $a$ 

 $\text{random\_dom}(a)$ 
    while  $a > 0 \wedge \text{random}(\text{true}/\text{false})$  do
         $a := \text{pre\_dom}[a]$ 
    return  $a$ 

```

It should be clear from the explanation above that the graphs generated by the algorithm are indeed RFGs. The average time complexity of the algorithm can be shown to be $O(n + da \log n + ba)$, so that the time for generating the test graphs does not dominate the time for solving the feedback set problems on them.

It can be observed that in the graphs generated, the height of the pre-dominator tree is rather low, such that the back-arc are not spread evenly. This effect intensifies as the graphs grow larger, which might be the cause for the better contraction ratios.

References

- A.V. Aho, R. Sethi & J.D. Ullman (1972), Code Optimization and finite Church-Rosser systems, *in* 'Design and Optimization of Compilers', R. Rudin, Ed., Prentice-Hall, Eaglewood Cliffs, NJ, pp. 89–106
- R.M. Karp (1972), Reducibility among combinatorial problems, *in* 'Complexity of Computer Computations', R.E. Miller & J.W. Thatcher, Eds., New York: Plenum Press, pp. 85–103
- R. Tarjan (1972), Depth-first search and linear graph algorithms, *in* 'SIAM Journal on Computing', Vol. 1, pp. 146–160
- B.K. Rosen (1973), Tree-Manipulation Systems and Church-Rosser Theorems, *in* 'Journal of the Association for Computing Machinery', Vol. 20 No. 1, pp. 160–187
- M.S. Hecht & J.D. Ullman (1974), Characterization of Reducible Flow Graphs, *in* 'Journal of the Association for Computing Machinery', Vol. 21 No. 3, pp. 167–175
- A. Shamir (1979), A linear time algorithm for finding minimum cutsets in reduced graphs, *in* 'SIAM Journal on Computing', Vol. 8 No. 4, pp. 645–655
- M.L. Fredman & R.E. Tarjan (1987), Fibonacci Heaps and Their Uses in Improved Network Optimization Algorithms, *in* 'Journal of the ACM', Vol. 34, pp. 596–615
- H. Levy & L. Low (1988), A contraction algorithm for finding small cycle cutsets, *in* 'Journal of Algorithms', Vol. 9, pp. 470–493
- V. Ramachandran (1988), Finding a minimum feedback arc set in reducible flow graphs, *in* 'Journal of Algorithms', Vol. 9, pp. 299–313
- G. Even, S. Naor, B. Schieber & M. Sudan (1998), Approximating minimum feedback sets and multicutsets in directed graphs, *in* 'Algorithmica', Vol. 20, pp. 151–174
- P. Festa, P.M. Pardalos & M.G.C. Resende (1999), Feedback Set Problems, *in* 'Handbook of Combinatorial Optimization', D.Z. Du & P.M. Pardalos, Eds. Vol. 4, Kluwer Academic Publishers, pp. 209–258