Maintenance of transitive closures and transitive reductions of graphs

MAINTENANCE OF TRANSITIVE CLOSURES AND TRANSITIVE REDUCTIONS OF GRAPHS

J.A. La Poutré and J. van Leeuwen

RUU-CS-87-25
November 1987

Rijksuniversiteit Utrecht

Vakgroep informatica

Padualaan 14 3584 CH Utrecht
Corr. adres: Postbus 80.089, 3508 TB Utrecht
Telefoon 030-531454
The Netherlands

MAINTENANCE OF TRANSITIVE CLOSURES AND TRANSITIVE REDUCTIONS OF GRAPHS

Technical Report RUU-CS-87-25
November 1987

Department of Computer Science
University of Utrecht
P.O. Box 80.089, 3508 TB Utrecht
The Netherlands

An extended abstract of this paper appeared in the Proceedings of the 13th Int. Workshop on "Graph-Theoretic Concepts in Computer Science" (WG '87), June 29 - July 1, 1987, Kloster Banz/Staffelstein (FRG), published as : Lecture Notes in Computer Science, Vol. 314, Springer-Verlag, Berlin, 1988, pp. 106 - 120.

Full manuscript completed : November 1987.

Technical Report completed : July 1989.

Contents

1 Introduction 3

2 Definitions 4

3 Maintaining transitive closures and transitive reductions: basic solutions 5

3.1 Edge insertions 5

3.2 Edge deletions: an algorithm for the acyclic case 9

4 Transitive reduction and the component input matrix 15

4.1 Transitive reductions of acyclic graphs 15

4.2 General Graphs 17

4.3 Transitive reductions of general graphs 19

5 The General Problem 22

6 Edge insertions 22

6.1 Procedure insert (i, j) 23

6.2 Correctness proof 23

6.3 Complexity 32

6.3.1 Analysis of procedure insert 32

6.3.2 Analysis of procedure adapt 33

6.3.3 Analysis of procedure joincomponents 33

6.3.4 Complexity for q insertions 34

7 Edge deletions 34

7.1 Procedure delete(i, j) 35

7.2 Comment and correctness proof 35

7.3 Complexity 53

7.3.1 Analysis of procedure delete 53

7.3.2 Analysis of procedure adjust 54

7.3.3 Analysis of procedure componentsbreak 55

7.3.4 Analysis of procedure resetcomponents 55

7.3.5 Analysis of procedure adjusttransitivereduction 56

7.3.6 Analysis of procedure disconnect 56

7.3.7 Analysis of procedure valueNC 56
7.3.8 Analysis of procedure neutralNC 57
7.3.9 Complexity of q deletions 57

8 Conclusion and remarks **58**

MAINTENANCE OF TRANSITIVE CLOSURES AND TRANSITIVE REDUCTIONS OF GRAPHS

J.A. La Poutré and J. van Leeuwen

1 Introduction

Let $G =< V, E >$ be a directed graph, $G^* =< V, E^* >$ its transitive closure and $G^- =< V, E^- >$ its transitive reduction (cf. [1]). Let E^* and E^- be represented by incidence matrices. Suppose edges are inserted in and deleted from G one at a time. We consider the problem of efficiently updating G^* and G^- each time an edge is inserted or deleted.

Ibaraki and Katoh [2] presented two algorithms that update G^* when edge insertions and deletions are considered separately. Their insertion algorithm takes $O(|V|^3)$ time for q consecutive insertions and their deletion algorithm takes $O(|V|^2(|E_{\text{old}}| + |V|))$ time for q consecutive deletions (where the subscript 'old' refers to the original graph before the q deletions). A more careful analysis of their algorithms yields an $O(|E^*_{\text{new}}| \cdot |V|)$ time bound for q consecutive insertions (where the subscript 'new' refers to the result graph after the q insertions) and an $O(|E^*_{\text{old}}| \cdot |V| + |E^*_{\text{old}}| \cdot |E_{\text{old}}|)$ time bound for q consecutive deletions.

In this paper we present more efficient algorithms for the same problems, and for maintaining the transitive reduction of G as well.

First we present an algorithm for updating both G^* and G^- in the case of edge insertions, that requires $O(|E_{\text{new}}|.|V|)$ time for q consecutive insertions. The algorithm employs an efficient search strategy (e.g. depth first search) for determining the entities to be updated. The approach is related to an algorithm presented by Rohnert [3] for updating least-cost paths in graphs.

We also present a new algorithm for updating G^* and G^- in case of edge deletions that requires $O(|E_{\text{old}}|.|V| + e^{\text{totcyc}}_{\text{old}} \cdot e^{\text{maxcyc}}_{\text{old}})$ time for q consecutive deletions, where $e^{\text{maxcyc}}_{\text{old}}$ denotes the maximum number of interior edges that are contained in a strongly connected component of G_{old} and $e^{\text{totcyc}}_{\text{old}}$denotes the total number of edges that are interior to any strongly connected component in G_{old} . (Hence, $e^{\text{maxcyc}}_{\text{old}} \leq e^{\text{totcyc}}_{\text{old}} \leq E_{\text{old}}$ and if G_{old} is acyclic then $e^{\text{maxcyc}}_{\text{old}} = e^{\text{totcyc}}_{\text{old}} = 0$.) The algorithm again uses an efficient search strategy (comparable to that used by Rohnert [3]), in combination with the use of auxiliary information about the number of ways one can arrive at nodes coming from other nodes. The information is updated by both the insertion and deletion algorithm.

The algorithms especially yield better time complexities (compared with those presented in [2]) for graphs with $E_{\text{old}} \ll |E^*|$ (e.g. planar graphs) and for graphs with relatively small components (i.e. graphs with $e^{\text{maxcyc}}_{\text{old}} = o(|E|)$). For example, for planar graphs our algorithms both take $O(|V|^2)$ time for any q consecutive applications, whereas the

algorithm presented in [2] take $O(|V|^3)$ time in this case. For acyclic graphs, our deletion algorithm takes $O(|E_{old}| \cdot |V|)$ $(= O(|V|^3))$ time for q consecutive deletions, whereas the deletion algorithm presented in [2] takes $O(|E^*_{old}|.|V| + |E^*_{old}|.|E_{old}|)$ $(= O(|V|^4))$ time for q consecutive deletions.

The paper is organized as follows. In Section 2 we define some notions and in Section 3 we present some restricted algorithms, as an introduction to the ultimate algorithms. In Section 4 notions with respect to strongly connected components are introduced and the transitive reduction of a graph is defined as in [1]. Section 5 gives a precise description of the problems to solve. In Section 6 and Section 7 the procedures for edge insertions and edge deletions are presented, including correctness proofs and complexity considerations. Finally, in Section 8 the results are condensed in some theorems and some concluding remarks are stated.

2 Definitions

Let $G =< V, E >$ be a directed graph with $n = \#V$. The adjacency matrix M of G is the $n \times n$ matrix with entries from $\{0, 1\}$ given by

$$M(i,j) = 1 \Longleftrightarrow (i,j) \in E$$

for $i, j \in V$. If $(i,j) \in E$, then we also write $i \longrightarrow j$. The array P of predecessor-sets w.r.t. G is defined by

$$P(j) = \{i \in V | (i,j) \in E\}$$

for $j \in V$. The array S of successor-sets w.r.t. G is defined by

$$S(i) = \{j \in V | (i,j) \in E\}$$

for $i \in V$.

For any two nodes $i, j \in V$ we write $i \xrightarrow[G]{*} j$ iff there is a (possibly empty) path from i to j in G . If $i \xrightarrow[G]{*} j$, node j is called to be reachable from i. We write $i \xrightarrow[G]{+} j$ if there is a non-empty path from i to j (i.e., a path of at least one edge). Graph G is said to be acyclic if

$$\bigwedge_{i,j \in V} [i \xrightarrow[G]{+} j \Longrightarrow i \neq j].$$

The transitive closure of G is the graph denoted by $G^* =< V, E^* >$, where

$$E^* = \{(i,j) \in V \times V | i \xrightarrow[G]{*} j\}.$$

The adjacency matrix of G^* is denoted by M^*_G. Clearly $M^*_G(i,j) = 1 \Longleftrightarrow i \xrightarrow[G]{*} j$ for $i, j \in V$.

Finally, if there is no danger of ambiguity, we often omit the subscript G.

Definition 2.1 *Let $G =< V, E >$ be a graph, $n = \#V$. Then the input matrix N_G of G is the $n \times n$-matrix given by*

$$N_G(k,m) = \#\{(l,m) \in E | k \xrightarrow[G]{*} l\}$$

for $k, m \in V$.

Hence, $N(k,m)$ is the number of edges incoming to m, that are reachable from k. (We will say that an edge (l,m) is reachable from k if its starting node l is reachable from k.) Stated differently, $N(k,m)$ is the number of edges over which one can arrive at m, coming from k. The following properties hold evidently.

Lemma 2.2 *Let G be a graph, $G =< V, E >$. Then for $k, m \in V$:*

$$k \xrightarrow[G]{*} m \iff [N(k,m) \geq 1 \vee k = m].$$

Lemma 2.3 *For the input matrix N_G of any graph $G =< V, E >$, the following holds for $k, m \in V$:*

$$N_G(k,m) = \sum_{l \in V} M^*_G(k,l) M_G(l,m).$$

3 Maintaining transitive closures and transitive reductions: basic solutions

Let $G_{\text{old}} =< V, E_{\text{old}} >$ be an arbitrary directed graph with $n = \#V$ and $V = \{1, \cdots, n\}$. Let $G^*_{\text{old}} =< V, E^*_{\text{old}} >$ be the transitive closure of G_{old}. Suppose G_{old} and G^*_{old} are represented by matrices M and M^* and by the arrays of sets P and S, corresponding to $M_{\text{old}}, M^*_{\text{old}}, P_{\text{old}}$ and S_{old} respectively.

Suppose an edge (i,j) is inserted in or deleted from G_{old}, resulting in the new graph G_{new}. The problem is to update M, M^*, P and S in such a way that they correspond to $M_{\text{new}}, M^*_{\text{new}}, P_{\text{new}}$ and S_{new} respectively.

In the next few subsections we give algorithms for the insertion or deletion of a single edge. (Henceforth we will assume that an alternation of M results in an alternation of P and S at the same time.) The algorithms are presented in order to develop the basic ideas for the efficient solutions to the update problem in later sections. We assume the reader to be familiar with the Hoare-style of program specification. Pre- and postconditions will be labeled by "pre:" and "post:", respectively.

(We write $i \xrightarrow[\text{old}]{*} j$ to denote that j is reachable from i in G_{old}, and $i \xrightarrow[\text{new}]{*} j$ likewise to denote that j is reachable from i in G_{new}.)

3.1 Edge insertions

Suppose $(i,j) \notin E_{\text{old}}$ and (i,j) is inserted in G_{old}. Then we have $G_{\text{new}} =< V, E_{\text{old}} \cup \{(i,j)\} >$.

Lemma 3.1 *Let* $G'_{\text{old}} =< V, E'_{\text{old}} >$, $(i,j) \notin E'_{\text{old}}$ *and* $G'_{\text{new}} =< V, E'_{\text{old}} \cup \{(i,j)\}\}$. *Then for* $k, m \in V$.

$$k \xrightarrow[\text{new}]{*} m \iff k \xrightarrow[\text{old}]{*} m \vee (k \xrightarrow[\text{old}]{*} i \wedge j \xrightarrow[\text{old}]{*} m).$$
$$k \xrightarrow[\text{new}]{*} i \iff k \xrightarrow[\text{old}]{*} i.$$
$$j \xrightarrow[\text{new}]{*} m \iff j \xrightarrow[\text{new}]{*} m.$$

Proof Trivial. □

We now present procedure insert*, that satisfies the specification

$$\{M = M_{\text{old}} \wedge M^* = M^*_{\text{old}} \wedge M(i,j) = 0\}$$
$$\text{insert}^*(i,j)$$
$$\{M = M_{\text{new}} \wedge M^* = M^*_{\text{new}}\}$$

The procedure is given in Figure 1. The predicates P_0 and $R(i)$ occuring in the subsequent procedures will be stated afterwards. The procedure uses two auxiliarly colours (red and blue) to colour nodes. Initially all nodes are assumed to be neutral, i.e., not coloured.

We will now argue that the procedure insert* satisfies the specifications.

Procedure insert* operates in the following way. First, M is adjusted to record the inserted edge. By Lemma 3.1 it is easily seen that M^* only needs to be updated if $\neg(i \xrightarrow[\text{old}]{*} j)$, i.e. $M^*(i,j) = 0$. If this is the case, M^* is updated per row, i.e., by handling $M^*(k,\cdot)$ for each k (line 4). By Lemma 3.1 it follows that row $M^*(k,\cdot)$ only needs to updated if $k \xrightarrow[\text{old}]{*} i \wedge \neg(k \xrightarrow[\text{old}]{*} j)$, i.e., $M^*(k,i) = 1 \wedge M^*(k,j) = 0$. Procedure insert* performs this update for any given k by first colouring j red (line 6). At this moment (line 7) condition P_0 holds, where P_0 is given in Figure 2. For each k, $R(k)$ denotes the predicate.

$$\bigwedge_m [M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m].$$

P_0 indeed holds at line 7 of the procedure, since j is the only red node at this moment and there are no blue nodes.

Lemma 3.2 $\{P_0\}$ adapt*(k) $\{P_0 \wedge$ there are no red nodes$\}$.

Proof. It is readily seen that P_0 is an invariant of the do-loop of procedure adapt*. Moreover this loop terminates, since at every pass of the loop a red node l is coloured blue and $M^*(k,l) := 1$. Note that a node m can only become red if $M^*(k,m) = 0$ and that an entry of M^* is never changed from 1 to 0, hence blue nodes cannot turn red again during the call of adapt*(k). □

Lemma 3.3 $P_0 \wedge$ *there are no red nodes* $\Longrightarrow R(k)$.

Figure 1: Procedure insert*.

```
(1)  procedure insert*(i,j);{pre : M = M_old ∧ M* = M*_old; post : M = M_new ∧ M* = M*_new}
(2)      M(i,j) := 1;
(3)      if M*(i,j) = 0
(4)          ⟶ for k := 1 to n do
(5)                ⟶ if M*(k,j) = 0 ∧ M*(k,i) = 1
(6)                      ⟶ colour j red;
(7)                        {P0}
(8)                        adapt*(k)
(9)                        {P0∧ there are no red nodes }{R(k)}
(10)                       discolour all (coloured) nodes
(11)               fi {R(i) holds for 1 ≤ i ≤ k}
(12)           rof {R(i) holds for 1 ≤ i ≤ n}
(13)     fi

(15) procedure adapt*(k) {local to procedure insert*}
(16)     {P0}
(17)     do there are red nodes
(18)         ⟶ let l be a red node;
(19)           {P0 ∧ l is red}
(20)           M*(k,l) := 1; colour l blue;
(21)           for all m ∈ S(l)
(22)           ⟶ if M*(k,m) = 0⟶ colour m red fi
(23)           rof
(24)           {P0}
(25)     od
(26)     {P0∧ there are no red nodes }
```

Figure 2: Condition P_0.

P_0	For all nodes m the following conditions hold.
$P_{0,0}$:	Node m is red, blue or neutral; node j is red or blue.
$P_{0,1}$:	If m is red, then $M^*(k,m) = 0 \wedge \neg(k \xrightarrow[\text{old}]{*} m) \wedge k \xrightarrow[\text{new}]{*} m.$
$P_{0,2}$:	If m is blue, then $M^*(k,m) = 1 \wedge \neg(k \xrightarrow[\text{old}]{*} m) \wedge k \xrightarrow[\text{new}]{*} m.$
$P_{0,3}$:	If m is neutral, then $M^*(k,m) = 1 \iff k \xrightarrow[\text{old}]{*} m.$
$P_{0,4}$:	If l is a blue node, then $(l,m) \in E_{\text{new}} \iff$ (m is blue or red) $\vee\, k \xrightarrow[\text{old}]{*} m.$

Figure 3:

$P'_{0,1}$:	Node m is either blue or neutral; node j is blue.
$P'_{0,2}$:	If m is blue, then $M^*(k,m) = 1 \wedge k \xrightarrow[\text{new}]{*} m$
$P'_{0,3}$:	If m is neutral, then $M^*(k,m) = 1 \iff k \xrightarrow[\text{old}]{*} m.$
$P'_{0,4}$:	If l is a blue node, then $(l,m) \in E_{\text{new}} \implies$ (m is blue) $\vee\, k \xrightarrow[\text{old}]{*} m.$

Proof. Suppose "$P_0 \wedge$ there are no red nodes" holds. Hence we have for all nodes m assertion P_0' (cf. fig. 3).

To prove $R(k)$ we need to show that $M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m$ for all nodes m. Let m be a node.

($\Rightarrow$) Suppose $M^*(k,m) = 1$. By P_0' we have $k \xrightarrow[\text{new}]{*} m \vee k \xrightarrow[\text{old}]{*} m$. Hence $k \xrightarrow[\text{new}]{*} m$ (By property 3.1).

($\Leftarrow$) Suppose by way of contradiction that $k \xrightarrow[\text{new}]{*} m \wedge M^*(k,m) = 0$. Then it follows from P_0' that m is neutral, $\neg(k \xrightarrow[\text{old}]{*} m)$ and $m \neq k$. Lemma 3.1 yields that $j \xrightarrow[\text{old}]{*} m$. Therefore there is a path from j to m in G_{old}. Consider such a path. Since j is on this path, there is a blue node on it (by $P_{0,1}'$). Now let l be the last blue node on the path. Since m is neutral, l has a successor l' on the path that is neutral. Therefore $P_{0,4}'$ yields $k \xrightarrow[\text{old}]{*} l'$. Since $l' \xrightarrow[\text{old}]{*} m$ this implies $k \xrightarrow[\text{old}]{*} m$ and hence $M^*(k,m) = 1$. Contradiction. □

Observe that procedure adapt*(k) in fact employs a kind of search strategy on the graph, starting at j, for finding the nodes m for which $M^*(k,m)$ needs to be updated.

Theorem 3.4 *Procedure* insert*(i,j) *satisfies the following specification:*

$$\{M = M_{\text{old}} \wedge M^* = M^*_{\text{old}} \wedge M(i,j) = 0\}$$

$$\text{insert}^*(i,j)$$

$$\{M = M_{\text{new}} \wedge M^* = M^*_{\text{new}}\}$$

Example 3.5 Let G_{old} be given by Figure 4 (not including edge (1,2)). Suppose edge (1,2) is inserted using procedure insert*. The algorithm 'searches' for node pairs (k,l) for which $M^*(k,l)$ must be increased to 1. Now adapt*(1) starts the search in node 2 and first node 2 red, then node 4 and 5 and finally node 9. Note that the outgoing edges of a node m are only traversed if m is red and therefore if $M^*(l,m)$ is changed from 0 to 1.

3.2 Edge deletions: an algorithm for the acyclic case

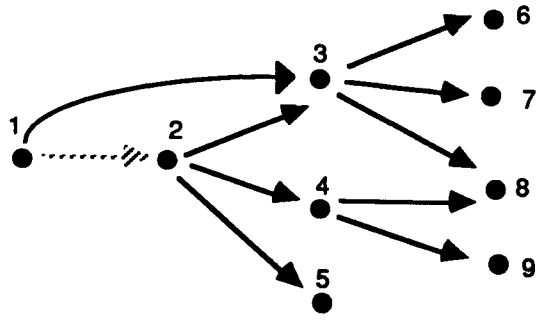
Suppose $(i,j) \in E_{\text{old}}$ and suppose (i,j) is deleted from G_{old}. Then we have $G_{\text{new}} =< V, E_{\text{old}} \backslash \{(i,j)\} >$. For convenience, we state the converse of Lemma 3.1.

Lemma 3.6 *Let* $G'_{\text{old}} =< V, E'_{\text{old}} >, (i,j) \in E'_{\text{old}}$ *and* $G'_{\text{new}} =< V, E'_{old} \backslash \{(i,j)\} >$. *Then for* $k, m \in V$*:*

$$\begin{aligned}
k \xrightarrow[\text{old}]{*} m &\iff k \xrightarrow[\text{new}]{*} m \vee (k \xrightarrow[\text{new}]{*} i \wedge j \xrightarrow[\text{new}]{*} m) \\
k \xrightarrow[\text{old}]{*} i &\iff k \xrightarrow[\text{new}]{*} i \\
j \xrightarrow[\text{old}]{*} m &\iff j \xrightarrow[\text{new}]{*} m
\end{aligned}$$

Henceforth in this section, we assume that G_{old} is acyclic.

Figure 4:



We now present procedure delete*, satisfying the specification

$$\{M = M_{\text{old}} \wedge M^* = M^*_{\text{old}} \wedge N = N_{\text{old}} \wedge M(i,j) = 1\}$$

$$\text{delete}^*(i,j)$$

$$\{M = M_{\text{new}} \wedge M^* = M^*_{\text{new}} \wedge N = N_{\text{new}}\}$$

The procedure is given in Figure 5. The predicates Q_0 and $S(k)$ are stated afterwards. The procedure uses one auxiliarly colour (red) to colour nodes. Initially all nodes are assumed to be neutral, i.e. not coloured.

We discuss the algorithm and argue in support of its correctness.

For $k \in V$, let $I(k)$:

$$\underset{m}{\forall}[M^*(k,m) = 1 \iff k \xrightarrow[\text{old}]{*} m]$$

$$\underset{m}{\forall}[N(k,m) = \#\{(l,m) \in E_{\text{old}} | k \xrightarrow[\text{old}]{*} l\}]$$

At the invocation of the procedure delete*, $\underset{k}{\forall}[I(k)]$ holds. In the algorithm, first M (and hence S and P) is adjusted to record the deletion of edge (i,j). (This does not affect $I(k)$ for $k \in V$ because M^* and N are still "old"). Like in the procedure insert* , we update $M^*(k,\cdot)$ and $N(k,\cdot)$ for each row k separately, for k from 1 to n. Because edge (i,j) is removed, this edge cannot contribute to $N(k,j)$ anymore for k with $k \xrightarrow[\text{old}]{*} i$: hence for such k, $N(k,j)$ need to be updated. On the other hand, by Lemma 3.6 it is seen that the rows $M^*(k,\cdot)$ and $N(k,\cdot)$ need to be updated for such k only. This gives rise to the guard in the if-statement of line 4 and to the statements in line 5 and line 10.

We now distinguish two cases, according to the inner if-statement in line 4-11.

- If $N(k,j) = 1 \wedge I(k)$ holds (cf. line 4), then $N(k,j)$ is decreased and j is coloured red (line 5-6). From Lemma 3.6, $k \xrightarrow[\text{new}]{*} i$ and G_{old} being acyclic, it easily follows that Q_0 holds at line 7, where Q_0 is given in Figure 6.

Figure 5: Procedures for edge deletion.

```
(1)  procedure delete*(i,j){pre : M(i,j) = 1}
(2)      M(i,j) := 0; {I(k) holds for k ∈ V}
(3)      for k := 1 to n
(4)          ⟶ if   M*(k,i) = 1⟶ {I(k) ∧ M*(k,i) = 1}
(5)                                 if N(k,j) = 1⟶ N(k,j) := N(k,j) − 1
(6)                                                colour j red;
(7)                                                {Q0}
(8)                                                adjust*(k)
(9)                                                {Q∧ there are no red nodes}{S0(k)}
(10)                                [] N(k,j) > 1⟶ N(k,j) := N(k,j) − 1 {S0(k)}
(11)             fi                 fi
(12)     rof

(15) procedure adjust*(k) {local to procedure delete*}
(16)     {Q0}
(17)     do there are red nodes
(18)         ⟶ let m0 be a red node; {Q0 ∧ m0 is red }
(19)             M*(k,m0) := 0;
(20)             discolour m0
(21)             {Q0'}
(22)             for all m' ∈ S(m0)
(23)             ⟶ N(k,m') = N(k,m') − 1;
(24)                 if N(k,m') = 0⟶ colour m' red fi
(25)             rof
(26)             {Q0''}{Q}
(27)     od
(28)     {Q0∧ there are no red nodes }
```

Figure 6: Condition Q_0.

Q_0 For all nodes m the following holds.

$Q_{0,1}$: $M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m \vee \exists_{\text{red } l} [l \xrightarrow[\text{new}]{*} m]$

$Q_{0,2}$: $N(k,m) = \#\{(l,m) \in E_{\text{new}} | M^*(k,l) = 1\}$

$Q_{0,3}$: If m is red, then
$k \xrightarrow[\text{old}]{*} m \wedge N(k,m) = 0 \wedge k \neq m$

At this point (line 8), procedure adapt*(k) is called. In Lemma 3.8 it will be shown that $\{Q_0\}$adjust*$(k)\{Q_0 \wedge$ there are no red nodes$\}$ holds. It is easily seen that we have

$$Q_0 \wedge \text{there are no red nodes} \Longrightarrow S_0(k),$$

where $S_0(k)$ is given by

$$\forall_m [M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m]$$
$$\forall_m [N(k,m) = \#\{(l,m) \in E_{\text{new}} | k \xrightarrow[\text{new}]{*} l\}].$$

Therefore $S_0(k)$ holds in line 9.

- If $N(k,j) > 1 \wedge I(k)$ holds (cf. line 10), then in G_{old} there are at least two edges that are incoming to j and reachable from k. Hence there is a node $l \neq i$ such that $(l,j) \in E_{\text{old}} \wedge k \xrightarrow[\text{old}]{*} l$. Because $l \neq i$ and G_{old} is acyclic (and hence a cycle $j \xrightarrow[\text{new}]{*} l \longrightarrow \text{new} j$ does not exist), this implies $(l,j) \in E_{\text{new}} \wedge k \xrightarrow[\text{new}]{*} l$ (by using Lemma 3.6). Therefore we have $k \xrightarrow[\text{new}]{*} m \iff k \xrightarrow[\text{old}]{*} m$ for all m. This yields that after decreasing $N(k,j)$, condition $S_0(k)$ holds.

We have now established the following result.

Theorem 3.7 *Procedure delete*$^*(i,j)$ *satisfies the following specification*

$$\{M = M_{\text{old}} \wedge M^* = M^*_{\text{old}} \wedge N = N_{\text{old}} \wedge M(i,j) = 1\}$$
$$\textit{delete}^*(i,j)$$
$$\{M = M_{\text{new}} \wedge M^* = M^*_{\text{new}} \wedge N = N_{\text{new}}\}.$$

We are left with the task to prove lemma 3.8.

Lemma 3.8

$$\{Q_0\} \text{ adjust}^*(k) \{Q_0 \wedge \text{there are no red nodes}\}$$

Proof. Suppose Q_0 holds at line 16. First we prove that Q_0 is an invariant of the do-loop of adjust*(k) in line 17-27. At line 18, Q_0 holds and m_0 is red. Therefore, Q_0' holds in line 21, where Q_0' is given below. (Note that indeed $\neg(k \xrightarrow[\text{new}]{*} m_0)$ holds at line 21, because at line 18 we have that $N(k, m_0) = 0 \wedge Q_{0,2} \wedge k \neq m_0$ holds, and because there is a path in G_{new} from k to m_0 ($k \neq m_0$) only if there is a path from k to some predecessor of m_0).

$Q'_{0,}$: For all nodes $m \neq m_0$ the following two conditions hold.

$Q'_{0,1}$: $M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m \vee \exists_{\text{red } l}[l \xrightarrow[\text{new}]{*} m] \vee m_0 \xrightarrow[\text{new}]{*} m$

$Q'_{0,2}$: $N(k,m) = \#\{(l,m) \in E_{\text{new}} | k \xrightarrow[\text{new}]{*} l \vee \exists_{\text{red } l'}[l' \xrightarrow[\text{new}]{*} l] \vee m_0 \xrightarrow[\text{new}]{*} l\}$

$Q'_{0,3}$: $Q_{0,3}$

$Q'_{0,4}$: $M^*(k,m_0) = N(k,m_0) = 0 \wedge k \xrightarrow[\text{old}]{*} m_0 \wedge \neg(k \xrightarrow[\text{new}]{*} m_0) \wedge$
$\forall_{\text{red } l}[\neg(l \xrightarrow[\text{new}]{*} m_0)]$

It is easily seen (by using that G_{new} is acyclic) that Q_0'' holds at line 26 (where Q_0'' is given below). Indeed, since $Q''_{0,1}$ directly follows from $Q'_{0,1}$ and since $Q''_{0,2}, Q''_{0,3}$ and $Q''_{0,4}$ follow from $Q'_{0,2}, Q'_{0,3}$ and $Q'_{0,4}$ by means of the statements in line 22-25 and the acyclicness of G_{old}.

Q_0'': For all nodes $m \neq m_0$ the following conditions hold.

$Q''_{0,1}$: $M^*(k,m) = 1 \iff k \xrightarrow[\text{new}]{*} m \vee \exists_{\text{red } l}[l \xrightarrow[\text{new}]{*} m] \vee \exists_{m':(m_0,m') \in E_{\text{new}}}[m' \xrightarrow[\text{new}]{*} m]$

$Q''_{0,2}$: $N(k,m) = \#\{(l,m) \in E_{\text{new}} | k \xrightarrow[\text{new}]{*} l \vee \exists_{\text{red } l'}[l' \xrightarrow[\text{new}]{*} l] \vee \exists_{m':(m_0,m') \in E_{\text{new}}}[m' \xrightarrow[\text{new}]{*} l]\}$

$Q''_{0,3}$: $Q'_{0,3} \wedge Q'_{0,4}$

$Q''_{0,4}$: If $(m_0, m) \in E_{\text{new}}$ then $N(k,m) = 0 \iff m$ is red.

We now prove that Q_0 follows from Q_0''. By $Q''_{0,4}, Q''_{0,2}$ and G_{new} being acyclic we have for all m with $(m_0, m) \in E_{\text{new}}$:

$$m \text{ is red} \quad \vee \quad k \xrightarrow[\text{new}]{*} m \vee \exists_{\text{red } l'}[l' \xrightarrow[\text{new}]{*} m] \quad \vee \quad \exists_{m'}[(m_0, m') \in E_{\text{new}} \wedge m' \neq m \wedge m' \xrightarrow[\text{new}]{*} m]. \tag{1}$$

Therefore, we have for all m with $(m_0, m) \in E_{\text{new}}$

$$m \text{ is red} \quad \vee \quad k \xrightarrow[\text{new}]{*} m \vee \bigvee_{\text{red } l'} [l' \xrightarrow[\text{new}]{*} m] \tag{2}$$

To see this, let A be defined by

$$A = \{m \mid (m_0, m) \in E_{\text{new}} \wedge m \text{ is not red } \wedge \neg(k \xrightarrow[\text{new}]{*} m) \wedge \forall_{\text{red } l'} [\neg(l' \xrightarrow[\text{new}]{*} m)]\}.$$

By (1) and the definition of A we have

$$\bigwedge_{m \in A} \bigvee_{m' \in A} [m' \neq m \wedge m' \xrightarrow[\text{new}]{*} m].$$

Since G_{new} is acyclic and V is finite, this implies that $A = \emptyset$. Hence (2) holds.

Finally, from (2), $Q''_{0,1}, Q''_{0,2}$ and $Q''_{0,3}$ it follows that Q_0 holds at line 26. This yields that Q_0 is an invariant for the do-loop.

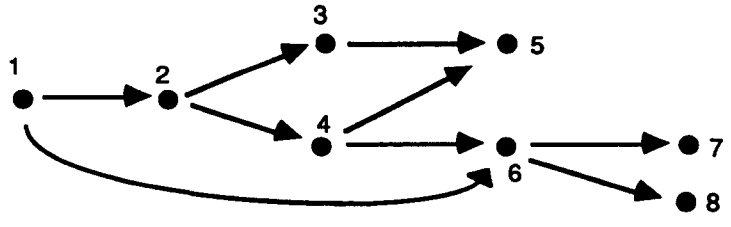
Furthermore, in each pass through the loop a red node m_0 is discoloured and $M^*(k, m_0)$ is set to 0, which implies (by $Q_{0,1}$ and since $M^*(k, m_0)$ is never set to 1) that m_0 will never be red again. Therefore the do-loop terminates. This yields that at line 28:

$$Q_0 \wedge \text{there are no red nodes left.}$$

□

Example 3.9 We illustrate procedure delete*(i, j). Let G_{old} be given by Figure 7. We have $N_{\text{old}}(1,2) = N_{\text{old}}(1,3) = N_{\text{old}}(1,4) = 1$ and $N_{\text{old}}(1,5) = N_{\text{old}}(1,6) = 2$. Suppose edge

Figure 7:

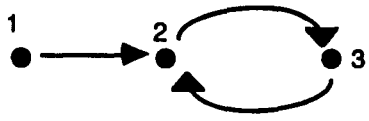


$(1,2)$ is deleted by means of procedure delete*. The algorithm 'searches' all node pairs (i, j) for which $M^*(i, j)$ must be decreased to zero, and adjusts matrix N correspondingly at the same time. Procedure delete*$(1,2)$ decreases $N(1,2)$ by one, yielding $N(1,2) = 0$. Therefore it follows that there is no alternative way from 1 to 2 and hence procedure adjust*$(1,2)$ is called. It sets $M^*(1,2)$ to 0 and decreases $N(1,3)$ and $N(1,4)$ by one, since the edges $(2,3)$ and $(2,4)$ do not contribute to $N(1,3)$ and $N(1,4)$ respectively any

longer. Since this gives that $N(1,3) = N(1,4) = 0$ (there are no alternative paths), $M^*(1,3)$ and $M^*(1,4)$ are set to zero. Therefore $N(1,5)$ and $N(1,6)$ are decreased by two and one respectively, yielding $N(1,5) = 0$ and $N(1,6) = 1$. The algorithm stops by setting $M^*(1,5)$ to zero and by leaving $M^*(1,6)$ unchanged (since $N(1,6) = 1$ finally implies that there must be an alternative path from 1 to 6). (Hence $N(1,7), N(1,8), M^*(1,7)$ and $M^*(1,8)$ remain unchanged too.)

Example 3.10 We illustrate why procedure delete* works on acyclic graphs only. Let G_{old} be given in Figure 8 and suppose edge $(1,2)$ is deleted. Note that $N_{\text{old}}(1,2) = 2$.

Figure 8:



Procedure delete*$(1,2)$ decreases $N(1,2)$ by one, yielding $N(1,2) = 1$. Therefore it stops without changing $M^*(1,2)$, etc. This is because node 2 and node 3 are strongly connected in G_{old}, i.e. $2 \xrightarrow[\text{old}]{*} 3 \xrightarrow[\text{old}]{*} 2$, and hence it seems by means of $N(1,2)$ that there still is an alternative path from 1 to 2 (via 3) after deleting $(1,2)$. This obviously is not the case.

In the ultimate algorithms that will be presented in sections 6 and 7, we will therefore use some information about the number of edges incoming to an entire strongly connected component (cf. Def. 4.5) that are reachable from another strongly connected component. (I.e., we have to eliminate the interior edges of a strongly connected component in our calculations.)

4 Transitive reduction and the component input matrix

We employ the notion of the transitive reduction of a graph as given by Aho, Garey and Ullman([1]). We only state some appropriate definitions here. For a more thorough treatment of transitive reductions we refer to [1].

We first give the definition of transitive reductions for acyclic graphs, and state some properties of it.

4.1 Transitive reductions of acyclic graphs

Definition 4.1 *Let G be an acyclic graph, $G =< V, E >$. A transitive reduction $G^- =< V, E^- >$ of G is a graph with a minimal number of edges satisfying $G^* = (G^-)^*$.*

It follows from the next lemma, that the transitive reduction of an acyclic graph is unique.

Lemma 4.2 *Let $G =< VE >$ be an acyclic graph and let $G^- =< V, E^- >$ be a transitive reduction of G. Then*

$$E^- = \{(k,m) \in E | \forall_l [k \xrightarrow[G]{*} l \wedge (l,m) \in E \Longrightarrow k = l]\}$$

Proof Let G and G^- be as given above.
($\subseteq$) Suppose $(k,m) \in E^-$. Then $k \neq m$ (since otherwise deleting (k,m) from E^- would result in a smaller graph with the same transitive closure, which violates the minimality condition). Hence, by Definition 4.1, we have $k \xrightarrow[G]{+} m$. Let l be a node such that $k \xrightarrow[G]{*} l \xrightarrow[G]{} m$ [to prove: $k = l$]. Suppose $k \neq l$. Since G is acyclic, we have $l \neq m$. Moreover, because of $G^* = (G^-)^*$ we have $k \xrightarrow[G^-]{+} l \xrightarrow[G^-]{+} m$. Since G^- is acyclic, this yields that a path from k to l cannot contain the edge (k,m). Similarly a path from l to m cannot contain (k,m). Therefore there exists a path in G^- from k to m, not using edge (k,m). Hence, deleting (k,m) from E^- would not affect the transitive closure, which contradicts the minimality condition for E^-. Therefore we have

$$\bigvee_l [k \xrightarrow[G]{*} l \wedge (l,m) \in E \Longrightarrow k = l]$$

Finally, since $k \xrightarrow[G]{+} m$, there exists an l with $k \xrightarrow[G]{*} l \xrightarrow[G]{} m$ which yields that $(k,m) \in E$.
($\supseteq$) Suppose $(k,m) \in E$ and $\bigvee_l [(k \xrightarrow[G]{*} l \wedge (l,m) \in E) \Longrightarrow k = l]$. Suppose $(k,m) \notin E^-$ [to prove: contradiction]. Since G is acyclic, we have that $k \neq m$ and hence $k \xrightarrow[G^-]{+} m$. Because of $(k,m) \notin E^-$, there exists a node l such that $k \xrightarrow[G^-]{*} l \xrightarrow[G^-]{*} m$ and $k \neq l \neq m$. Therefore we have $k \xrightarrow[G]{+} l \xrightarrow[G]{+} m$. Because G is acyclic, this implies that $k \xrightarrow[G]{+} l' \xrightarrow[G]{} m$ holds for some node $l', l' \neq k$. This contradicts our assumption. □

Corollary 4.3 *Let $G =< V, E >$ be an acyclic graph. Then there is precisely one graph that is a transitive reduction of G.*

Because of Corollary 4.3 we henceforth speak of <u>the</u> transitive closure of an acyclic graph G, denoted by G^-.

Lemma 4.4 *Let $G =< V, E >$ be an acyclic graph. Then for $G^- =< V, E^- >$ we have*

$$\begin{aligned} E^- &= \{(k,m) \in E | N(k,m) = 1\} \\ &= \{(k,m) \in V \times V | N(k,m) = M(k,m) = 1\} \end{aligned}$$

Proof By Lemma 4.2 and Definition 2.1 the assertion easily follows. □

4.2 General Graphs

We introduce some notions for graphs.

Definition 4.5 *Let G be a graph. The (strongly connected) component $C(k)$ of k is given by*

$$C(k) := \{m \in V | k \xrightarrow[G]{*} m \wedge m \xrightarrow[G]{*} k\}.$$

The leader $L(k)$ of component $C(k)$ is given by $L(k) = minC(k)$.

We generalize L in an obvious way for node pairs $(k,m) \in V^2$ by $L((k,m)) = (L(k), L(m))$. The following property is trivial.

Property 4.6 *For $k, m \in V$, the following assertions hold.*

(a) $C(k) \cap C(m) = \emptyset \bigvee C(k) = C(m)$

(b) $L(k) = L(m) \Longleftrightarrow C(k) = C(m) \Longleftrightarrow k \in C(m)$

(c) $N(k,m) = N(L(k), m)$

Definition 4.7 *Let $G =< V, E >$ be a graph. The condensed graph $G^c =< V^c, E^c >$ of G is given by*

$$\begin{aligned} V^c &= L(V) = \{L(k) | k \in V\} \\ E^c &= L(E) \backslash \{(\kappa, \kappa) | \kappa \in V^c\} \\ &= \{(\kappa, \mu) \in V^c \times V^c | \kappa \neq \mu \wedge \underset{k \in C(\kappa)}{\exists} \underset{m \in C(\mu)}{\exists} [(k,m) \in E]\} \end{aligned}$$

The weighted adjacency matrix M^c of the condensed graph G^c is the $|V^c| \times |V^c|$ matrix given by

$$\begin{aligned} M^c(\kappa, \mu) &= \#\{(k,m) \in E | k \in C(\kappa) \wedge m \in C(\mu)\} \quad \text{for } \kappa, \mu \in V^c, \kappa \neq \mu \\ M^c(\kappa, \kappa) &= 0 \qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \text{for } \kappa \in V^c \end{aligned}$$

The arrays P^c and S^c of predecessor-sets and successor-sets w.r.t G^c are the $|V|-$arrays defined by

$$P^c(\lambda) = \{\kappa \in V^c | (\kappa, \lambda) \in E^c\}$$

$$S^c(\lambda) = \{\mu \in V^c | (\lambda, \mu) \in E^c\}$$

for $\lambda \in V^c$ and $P^c(\lambda)$ and $S^c(\lambda)$ are empty otherwise.

Note that V^c is in fact the set of leaders, and that there is an edge from one leader κ to another μ in G^c if there is an edge from the component $C(\kappa)$ to the component $C(\mu)$.

For convenience we state the following property.

Property 4.8 *Let $G =< V, E >$ be a graph and let $G^c =< V^c, E^c >$ be its condensed graph. Then the following properties hold:*

(a) For $\kappa, \mu \in V^c$ we have

$$(\kappa, \mu) \in E^c \iff M^c(\kappa, \mu) > 0$$

(b) For $\kappa, \mu \in V^c$ we have

$$\bigvee_{k \in C(\kappa)} \bigvee_{m \in C(\mu)} [k \xrightarrow[G]{*} m \iff \kappa \xrightarrow[G]{*} \mu]$$

(c) For $\kappa, \mu \in V^c$ we have

$$\kappa \xrightarrow[G]{*} \mu \iff \kappa \xrightarrow[G^c]{*} \mu$$

(d) G^c is acyclic

(e) If G is acyclic, then $G^c = G$.

We now introduce the component input matrix N^c.

Definition 4.9 *Let $G =< V, E >$ be a graph and $G^c =< V^c, E^c >$ be its condensed graph. The component input matrix N^c of G is the integer $|V^c| \times |V^c|$ matrix, given by*

$$N^c(\kappa, \mu) = \#\{(l, m) \in E | l \in V \backslash C(\mu) \wedge m \in C(\mu) \wedge \kappa \xrightarrow[G]{*} l\}$$

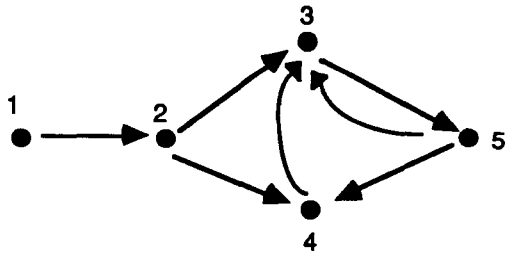
for $\kappa, \mu \in V^c$.

We can consider $N^c(\kappa, \mu)$ to be the number of edges (in G) incoming to component $C(\mu)$ from the outside, that are reachable from component $C(\kappa)$. Note that $N^c(\kappa, \kappa) = 0$ for $\kappa \in V^c$.

Consider the following example.

Example 4.10 Let G be given in Fig. 9 Then $V^c = \{1, 2, 3\}$ and $C(1) = \{1\}, C(2) = \{2\}, C(3) = \{3, 4, 5\}$. Now we have e.g. $N(1, 2) = 1$ and $N(1, 3) = 3$ while $N^c(1, 2) = 1$ and $N^c(1, 3) = 2$, since only the edges $(2, 3)$ and $(2, 4)$ contribute to $N^c(1, 3)$. Moreover, note that $N^c(3, 3) = 0$.

Figure 9:



Property 4.11 *Let G be a graph and let G^c be its condensed graph. Then the following holds for $\kappa, \mu \in V^c$.*

$$N^c(\kappa, \mu) = \sum_{\lambda \in V^c} M^*(\kappa, \lambda) M^c(\lambda, \mu)$$

Proof. By Definition 4.9, Property 4.6.a and Property 4.8.b we have

$$\begin{aligned} N^c(\kappa, \mu) &= \#\{(l, m) \in E | L(l) \neq \mu \wedge m \in C(\mu) \wedge \kappa \xrightarrow[G]{*} L(l)\} \\ &= \sum_{\lambda \in V^c} \#\{(l, m) \in E | l \in C(\lambda) \wedge m \in C(\mu) \wedge \lambda \neq \mu \wedge \kappa \xrightarrow[G]{*} \lambda\} \\ &= \sum_{\lambda \in V^c} M^*(\kappa, \lambda) M^c(\lambda, \mu) \end{aligned}$$

□

We want to express N^c in terms of N. Therefore, we first introduce a simple notion.

Definition 4.12 *Let G be a graph. The array e^c is the $|V|-$array given for $m \in V$ by*

$$e^c(m) = \#\{(l, m) \in E | L(l) = L(m)\}.$$

Hence, $e^c(m)$ is the number of edges incoming to m, that are within the component of m. In the following properties we refer to an arbitrary graph $G =< V, E >$.

Property 4.13 *For $m \in V$ we have $N(m, m) = e^c(m)$.*

Property 4.14 *For $k, m \in V$ the following holds: $k \xrightarrow[G]{*} m \Rightarrow N(k, m) \geq e^c(m)$.*

Property 4.15 *For $\kappa, \mu \in V^c$ the following holds.*

$$N^c(\kappa, \mu) = \begin{cases} \sum_{m \in C(\mu)} (N(\kappa, m) - e^c(m)) & \text{if } \kappa \xrightarrow[G]{*} \mu \\ 0 & \text{otherwise} \end{cases}$$

Proof In the proof of Claim 7.9 a more generally valid equality is proved. We therefore refer to that proof. □

Property 4.16 *For $\kappa, \mu \in V^c$ the next equivalence holds.*

$$\kappa \xrightarrow[G]{*} \mu \iff N^c(\kappa, \mu) \geq 1 \vee \kappa = \mu.$$

4.3 Transitive reductions of general graphs

For a general graph, its transitive reduction is defined through the transitive reduction of its condensed graph (representing the connection structure between components) and by means of a special graph representing the reachability structure within the components.

Definition 4.17 *Let G be a graph, $G =< V, E >$. The component-representation graph (CR−graph) G^{cr} of G is given by $G^{cr} =< V, E^{cr} >$, where E^{cr} consists of those nodepairs that are obtained in the following way:*

- *if $\{h_1, \cdots, h_k\}$ is a strongly connected component with $k \geq 2$ and $h_j < h_{j+1} (1 \leq j < k)$, then $(h_j, h_{j+1}) \in E^{cr}$ for $1 \leq j < k$ and $(h_k, h_1) \in E^{cr}$.*
- *if $\{h_1\}$ is a strongly connected component, then E^{cr} contains no edges w.r.t. h_1.*

The component-representation graph G^{cr} is the graph in which the strongly connected components of G are represented by single cycles. Notice that for $k, m \in V$ with $L(k) \neq L(m)$ we have $(k, m) \notin E^{cr}$. For convenience, we introduce the following notation.

Notation 4.18 *Let $G =< V, E >$ be a graph and let $G^{cr} =< V, E^{cr} >$ be its CR−graph. Let A be a $|V| \times |V|$ matrix with values in $\{0, 1\}$. Then we write $A \sim_{CR} G$ iff for all $(k, m) \in (V \times V) \backslash (V^c \times V^c)$ the equivalence $A(k, m) = 1 \Longleftrightarrow (k, m) \in E^{cr}$ holds.*

Since by Prop. 4.8 the condensed graph of a graph is acyclic, we are able to define the notion of a transitive reduction in the following way.

Definition 4.19 *Let G be a graph. Let $G^c =< V^c, E^c >$ and $G^{cr} =< V, E^{cr} >$ be its condensed graph and its component representation graph respectively. Let $(G^c)^- =< V^c, (E^c)^- >$ be the transitive reduction of the (acyclic!) condensed graph. Then the transitive reduction G^- of G is defined as $G^- =< V, (E^c)^- \cup E^{cr} >$. The adjacency matrix of G^- is denoted by M^-.*

It is easily verified (by Property 4.8.e) that Definition 4.19 is an extension of Definition 4.1

Example 4.20 The transitive reduction of the graph drawn in Figure 10 is drawn in Figure 11.

Figure 10:

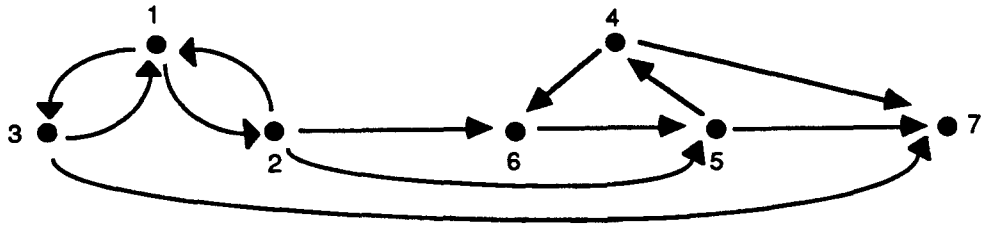
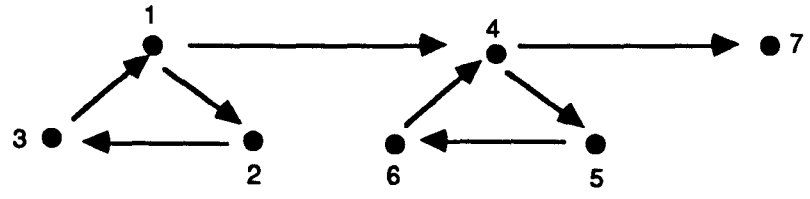


Figure 11:



Finally, we state some properties and a lemma that is an extension of Lemma 4.4. From Lemma 4.2 and Property 4.8.c and d, the next corollary follows easily.

Corollary 4.21 *Let $G =< V, E >$ be a graph. Let $(G^c)^- =< V^c, (E^c)^- >$ be the transitive reduction of its condensed graph. Then*

$$(E^c)^- = \{(\kappa, \mu) \in E^c | \forall_{\lambda \in V^c} [\kappa \xrightarrow[G]{*} \lambda \wedge (\lambda, \mu) \in E^c \Longrightarrow \lambda = \kappa]\}.$$

Lemma 4.22 *Let $G =< V, E >$ be a graph. Let $(G^c)^- =< V^c, (E^c)^- >$ be the transitive reduction of the condensed graph of G. Then the following equality holds*

$$(E^c)^- = \{(\kappa, \mu) \in V^c \times V^c | N^c(\kappa, \mu) = M^c(\kappa, \mu) > 0\}.$$

Proof Let $(\kappa, \mu) \in V^c \times V^c$. Then we have

$$\sum_{\lambda \in V^c} M^*(\kappa, \lambda) M^c(\lambda, \mu) = M^c(\kappa, \mu) > 0 \Longleftrightarrow (\kappa, \mu) \in E^c \wedge \forall_{\lambda \in V^c} [\kappa \xrightarrow[G]{*} \lambda \wedge (\lambda, \mu) \in E^c \Longrightarrow \lambda = \kappa].$$

Hence, by Corollary 4.21 and Property 4.11 the equality easily follows. □

By Definition 4.19, Notation 4.18 and Lemma 4.22 the following corollary holds.

Corollary 4.23 *Let $G^- =< V, E >$ be a graph. Then M^- satisfies the following conditions.*

1. $M^- \sim_{CR} G$
2. $M^-(\kappa, \mu) = 1 \Longleftrightarrow \sum_{\lambda \in V^c} M^*(\kappa, \lambda) \cdot M^c(\lambda, \mu) = M^c(\kappa, \mu) > 0$ *for all* $(\kappa, \mu) \in V^c \times V^c$.

Moreover, every $n \times n$ matrix A with entries from $\{0, 1\}$ that satisfies these two conditions, is equal to M^-.

5 The General Problem

Let $G_{\text{old}} =< V, E_{\text{old}} >$ be a directed graph, and let $n = \#V$ and $V = \{1 \cdots n\}$. Let $G^*_{\text{old}} =< V, E^*_{\text{old}} >$ and $G^-_{\text{old}} =< V, E^-_{\text{old}} >$ be the transitive closure and the transitive reduction of G_{old} respectively.

Suppose $G_{\text{old}}, G^*_{\text{old}}$ and G^-_{old} are represented by the matrices M, M^* and M^-, corresponding to $M_{\text{old}}, M^*_{\text{old}}$ and M^-_{old}. Let P and S be arrays of sets corresponding to P_{old} and S_{old}. Moreover, for each $k \in V$, an additional subset of $S(k)$ is related to k, being the set of all successors of k that are in the same component, i.e. $\{m \in S_{old}(k) | L_{old}(k) = L_{old}(m)\}$. We will not make the manipulations with this set explicit. We also assume the presence of the following information.

First we have matrix N corresponding to N_{old}. For representing the condensed graph G^c_{old} we have sets V^c and arrays L, C, P^c, S^c and e^c corresponding to $V^c_{\text{old}}, L_{old}, C_{old}, P^c_{\text{old}}, S^c_{\text{old}}$ and e^c_{old} respectively, where $C(\lambda)$ is an ordered list defined for $\lambda \in V^c$ only and where $P^c(\lambda)$ and $S^c(\lambda)$ are sets (for $\lambda \in V$) (cf. Definition 4.5 and 4.7). Moreover we have matrix M^c corresponding to M^c_{old} (i.e. M^c is a $n \times n-$ matrix such that for $\kappa, \mu \in V^c, M^c(\kappa, \mu) = M^c_{\text{old}}(\kappa, \mu)$ and $M^c(\kappa, \mu) = 0$ otherwise).

We assume that sets are implemented by lists and $n-$ arrays with the appropriate cross pointers (where needed), unless stated otherwise. For convenience, we need some auxiliary sets (being empty outside the procedure) and an $n-$array NC with associated list, recording the entries of NC that are not equal to -1 (with $NC = \underline{-1}$ outside the procedures insert and delete, i.e. $NC(k) = -1$ for $k \in V$).

Moreover, we assume that an alteration of M results in a corresponding alteration in P and S, and similarly for M^c, P^c and S^c , and for e^c and S (w.r.t. successors in the same component.) We therefore do not explicitly state alteration and conditions w.r.t. P, S, P^c and S^c. (However, we do make an exception in the procedures joincomponents and resetcomponents).

It is easily seen that the space complexity of the above quantities is $O(n^2)$.

Suppose an edge (i, j) is inserted to or deleted from G_{old}, resulting in the new graph G_{new}. The problem is to update all the above notions such that these will correspond to the new graph G_{new}. In Section 6 procedure insert is presented for updates in case of edge insertions, and in Section 7 procedure delete is presented for updates in case of edge deletions.

6 Edge insertions

Suppose $(i, j) \notin E_{\text{old}}$ and (i, j) is inserted in G_{old}. Then we have $G_{\text{new}} =< V, E_{\text{old}} \cup \{(i, j)\} >$. Below we present procedure insert(i, j) for performing the task described in Section 5 for the above insertion. Comments and a correctness proof are given in subsection 6.2, while the complexity is computed in subsection 6.3.

6.1 Procedure insert (i,j)

Procedures insert (i,j), adapt(κ) and joincomponents(j) are given in figures 12, 13 and 14. The operation of procedure insert(i,j) on the acyclic graph G^c_{old} can be compared to the operation of procedure insert*on an acyclic graph; i.e. it handles components (component leaders) like insert*handles nodes. In addition, it handles nodes of a component together (cf. Property 4.8.b or Property 4.6). However, insert(i,j) is more complicated because of the possibility that components are joined and hence the (heavy) changing of the condensed graph.

6.2 Correctness proof

We now argue that procedure insert(i,j) performs the task described in Section 5 for inserting the (new) edge (i,j).

Suppose we have a state as described in Section 5. Then procedure insert operates in the following way. First M is adjusted, and $Li := L_{old}(j)$ and $Lj := L_{old}(j)$. If $L_{old}(i) = L_{old}(j)$, then i and j belong to the same component. Therefore, inserting (i,j) does not cause any changes in the condensed graph. Hence, $G^c_{\text{new}} = G^c_{\text{old}}$. Therefore M^-, M^c, L and C do not need to be changed. Moreover we have by $i \xrightarrow[\text{old}]{*} j$ and by Lemma 3.1 that $G^*_{\text{new}} = G^*_{\text{old}}$. Therefore, M^* does not need to be changed either. In this case, line 3-6 is executed. Hence e^c is adjusted in line 3 and N is adjusted in line 5 according to its definition. Therefore, all entities refer to G_{new} at line 6, and hence RR holds, where RR is given below.

$$RR: V^c = V^c_{\text{new}} \wedge C = C_{new} \wedge L = L_{new} \wedge e^c = e^c_{\text{new}} \wedge M^c = M^c_{\text{new}} \wedge M^* = M^*_{\text{new}} \wedge N = N_{\text{new}} \wedge M^- = M^-_{\text{new}} \wedge M = M_{\text{new}}.$$

If $L_{old}(i) \neq L_{old}(j)$, then line 8-28 is executed. In this case a <u>new</u> component may arise. This is actually the case if $j \xrightarrow[\text{old}]{*} i$, since then $L_{old}(i) \neq L_{old}(j)$ implies that $\neg(i \xrightarrow[\text{old}]{*} j)$ and since insertion of (i,j) then yields $i \xrightarrow[\text{new}]{} j \xrightarrow[\text{new}]{*} i$ (by Lemma 3.1). This is recorded at variable *newcycle* in line 8. Since $L_{old}(i) \neq L_{old}(j)$ and an edge from i to j is inserted, the number of edges from $C_{old}(i)$ to $C_{new}(j)$ must be increased by one. Since, as we saw above, a new component may arise and hence we may have $V^c_{\text{old}} \neq V^c_{\text{new}}$, we introduce the intermediate generalized adjacency matrix M^c_{int} by

$$\begin{aligned} M^c_{\text{int}}(k,m) &= M^c_{\text{old}}(k,m) \text{ for } (k,m) \neq (Li, Lj) \\ M^c_{\text{int}}(Li, Lj) &= M^c_{\text{old}}(Li, Lj) + 1. \end{aligned}$$

Hence we have $M^c_{\text{int}}(\kappa,\mu) = \#\{(k,m) \in E_{\text{new}} | k \in C_{old}(\kappa) \wedge m \in C_{old}(\mu)\}$ for $\kappa, \mu \in V^c_{\text{old}}, \kappa \neq \mu$.

Now, $I(\kappa)$ holds at line 10 for all $\kappa \in V^c_{\text{old}}$, where $I(\kappa)$ is given below.

$I(\kappa)$ is the conjunction of the following clauses.

$$I_0(\kappa) \ : \ \kappa \in V^c_{\text{old}} \wedge V^c = V^c_{\text{old}} \wedge C = C_{old} \wedge L = L_{old} \wedge e^c = e^c_{\text{old}} \wedge M^- \sim_{CR} G_{\text{old}} \wedge M^c =$$

Figure 12:

```
1  Procedure insert (i, j);
2  M(i,j) := 1; Li := L(i); Lj := L(j);
3  if Li = Lj ⟶ adjust e^c;
4                  for k := 1 to n
5                      ⟶ if M*(k,i) = 1 ⟶ N(k,j) := N(k,j) + 1 fi
6                  rof{RR}
7  [] Li ≠ Lj
8     ⟶           newcycle:=[M*(j,i) = 1;]
9                  M^c(Li, Lj) := M^c(Li, Lj) + 1;
10                 {(∀κ : κ ∈ V^c : I(κ))}
11                 for all κ ∈ V^c
12                 ⟶ {I(κ)}
13                     if M*(κ,i) = M*(κ,j) = 1
14                         ⟶ for all k ∈ C(κ) ⟶ N(k,j) := N(k,j) + 1 rof;
15                             if κ ≠ Li ∧ κ ≠ Lj ⟶ M⁻(κ, Lj) := 0 fi
16                             {R(κ)}
17                     [] M*(κ,i) = 1 ∧ M*(κ,j) = 0
18                         ⟶ {I(κ) ∧ J(κ)}
19                             for all k ∈ C(κ) ⟶ N(k,j) := N(k,j) + 1 rof;
20                             if κ = Li ⟶ M⁻(Li, Lj) := 1 fi;
21                             colour Lj red;
22                             {P1} adapt (κ){P1 there are no red leaders };
23                             {R(κ)} discolour all coloured nodes;
24                     [] M*(κ,i) = 0
25                         ⟶ skip {R(κ)}
26                     fi {R(κ) is established}
27                 rof; {(∀κ : κ ∈ V^c : R(κ))}
28                 if newcycle ⟶ joincomponents(j){RR} fi
29 fi {RR}
```

Figure 13:

```
1   procedure adapt(κ);{local to procedure insert}
2       {P1}
3       do there are no red leaders
4       ⟶ let λo be a red leader;
5           {P1 ∧ λo is a red leader }
6           colour λo blue;
7           for all (k,l) ∈ C(κ) × C(λo)⟶M*(k,l) := 1 rof;
8           for all l ∈ C(λo)
9           ⟶ for all m ∈ S(l)
10              ⟶ for all k ∈ C(κ)⟶N(k,m) := N(k,m) + 1
11          rof rof rof;
12          for all μo ∈ S^c(λo)
13          ⟶ if M*(κ,μo) = 1⟶M⁻(κ,μo) := 0
14              [] M*(κ,μo) = 0⟶ colour μo red
15              fi
16          rof; {P1}
17      od;
18      {P1∧ there are no red leaders}
```

Figure 14:

```
1   procedure joincomponents (j);
2       *Λ := {l ∈ V | M*(l,j) = M*(j,l) = 1};
3       *λ_o := min Λ;
4       * set M⁻(κ,μ) := 0 for κ,μ ∈ V^c with κ ∈ Λ ⋁ μ ∈ Λ by enumerating all edges;
5       set M^c(κ,μ) := 0, P^c(κ) := S^c(κ) := ∅, e^c(k) := 0 for κ,μ ∈ V^c, k ∈ V,
        by enumerating all edges.
6       * for all κ ∈ V^c ∩ Λ
7           ⟶ set M⁻(k_1,k_2) := 0 for k_1,k_2 ∈ C(κ) by traversing
8               the ordered list C(κ)
9           rof;
10      * for all l ∈ Λ⟶L(l) := λ_o rof
11      *C(λ_o) := Λ
12      * adjust M^c, P^c, S^c, e^c and S (cf. section 5) by enumerating all edges;
13      * adjust V^c : V^c = (V^c \ Λ) ∪ {λ_o};
14      * adjust M⁻(w.r.t G^cr_new) by traversing the ordered list
15          C(λ_o)
16      * adjust M⁻(λ_o,·) for μ ∈ V^c_new as follows:
17          apply a breadth first search w.r.t. G^c_new (by means of
18          E^c) starting at λ_o to detect
19          those μ ∈ V^c_new, that are reachable from
20          λ_o in G^c_new by the direct edge (λ_o,μ) ∈ E^c_new only;
21          for such leaders, set M⁻(λ_o,μ) := 1;
22      * adjust M⁻(·,λ_0) for μ ∈ V^c_new as follows:
23          apply a backward breadth first search w.r.t. G^c_new
24          (by means of E^c) starting at λ_o
25          to detect those μ ∈ V^c from which λ_o
26          is reachable by the direct edge (μ,λ_o) only;
27          for such leaders μ, set M⁻(μ,λ_o) := 1
```

$$M^c_{\text{int}} \wedge M = M_{\text{new}}$$

$I_1(\kappa)$: $\underset{k \in C_{old}(\kappa)}{\forall} \underset{m}{\forall} [M^*(k,m) = 1 \iff k \xrightarrow[\text{old}]{*} m]$

$$\underset{k \in C_{old}(\kappa)}{\forall} \underset{m}{\forall} [N(k,m) = N_{\text{old}}(k,m)]$$

$I_2(\kappa)$: for all $\mu \in V^c_{\text{old}}$ the following holds

$$M^-(\kappa,\mu) = 1 \iff \sum_{\lambda \in V^c_{\text{old}}} M^*(\kappa,\lambda) M^c_{\text{old}}(\lambda,\mu) = M^c_{\text{old}}(\kappa,\mu) > 0$$

Now M^*, M^- and N are adjusted in the for-loop given in line 11-27: this is performed per row κ for each $\kappa \in V^c_{\text{old}}$ separately. Meanwhile, V^c, C, L and e^c refer to the old graph. We distinguish three cases, according to the if-statement in line 13-26.

Case 1 $\kappa \xrightarrow[\text{old}]{*} i \wedge \kappa \xrightarrow[\text{old}]{*} j$.

By Lemma 3.1 it is easily seen that $M^*_{\text{old}}(\kappa,\cdot) = M^*_{\text{new}}(\kappa,\cdot)$, which yields that $M^*(\kappa,\cdot)$ does not need to be updated. Because of the insertion of the (new) edge (i,j), and because $\kappa \xrightarrow[\text{old}]{*} i$ holds, $N(k,j)$ needs to be increased by one for all $k \in C_{old}(\kappa)$ (cf. Definition 2.1 and Property 4.8.b).

Moreover, because of the adaptations of M^c and $N(\kappa,\cdot)$ we must have $M^-(\kappa, Lj) = 0$ if $L(i) \neq \kappa \neq L(j)$. (cf. Corr. 4.23). Since in this case line 14-16 is executed, and hence all the above updates are performed, it is seen that at line 16 condition $R(\kappa)$ holds, where $R(\kappa)$ is given below.

$R(\kappa)$ is the conjunction of the following clauses.

$R_0(\kappa)$: $\kappa \in V^c_{\text{old}} \wedge V^c = V^c_{\text{old}} \wedge C = C_{old} \wedge L = L_{old} \wedge e^c = e^c_{\text{old}}$

$$\wedge M^- \sim_{CR} G_{\text{old}} \wedge M^c = M^c_{\text{int}} \wedge M = M_{\text{new}}$$

$R_1(\kappa)$: $\underset{k \in C_{old}(\kappa)}{\forall} \underset{m}{\forall} [M^*(k,m) = M^*_{\text{new}}(k,m)]$

$R_2(\kappa)$: $\underset{k \in C_{old}(\kappa)}{\forall} \underset{m}{\forall} [N(k,m) = N_{\text{new}}(k,m)]$

$R_3(\kappa)$: $\underset{\mu \in V^c}{\forall} [M^-(\kappa,\mu) = 1 \iff \sum_{\lambda \in V^c} M^*(\kappa,\lambda) M^c_{\text{int}}(\lambda,\mu) = M^c_{\text{int}}(\kappa,\mu) > 0]$

Case 2 $\neg(\kappa \xrightarrow[\text{old}]{*} i)$.

Then by Lemma 3.1 we have that $M^*_{\text{old}}(\kappa,\cdot) = M^*_{\text{new}}(\kappa,\cdot)$. Moreover, $N_{\text{old}}(\kappa,\cdot) = N_{\text{new}}(\kappa,\cdot)$ holds. Therefore, condition $R(\kappa)$ holds. (In particular, $R_3(\kappa)$ holds, because, among oth-

ers, $M^c_{\text{old}}(\kappa,\cdot) = M^c_{\text{int}}(\kappa,\cdot)$ holds as a consequence of $\neg(\kappa \xrightarrow[\text{old}]{*} i)$.) Since in this case line 24-25 is executed, $R(\kappa)$ holds at line 25.

Case 3 $\kappa \xrightarrow[\text{old}]{*} i \wedge \neg(\kappa \xrightarrow[\text{old}]{*} j)$.
In this case, line 18-23 is executed. Now, at line 18 $I(\kappa) \wedge J(\kappa)$ holds, where $J(\kappa)$ is given by

$$J(\kappa) : \neg(\kappa \xrightarrow[\text{old}]{*} Lj) \wedge \kappa \xrightarrow[\text{old}]{*} Li \wedge M^-(\kappa, Lj) = 0.$$

(Since $M^-(\kappa, Lj) = 0$ follows from $\neg(\kappa \xrightarrow[\text{old}]{*} Lj) = 0$).

Therefore, P_1 holds at line 22, where P_1 is stated in Figure 15 . (In P_1, a leader is called to be neutral, if it is neither red nor blue). Remark that $P_{1,2}$ holds for $\mu = Lj$, because $J(\kappa)$ yields that $\neg(\kappa \xrightarrow[\text{old}]{*} Lj)$ and $\kappa \xrightarrow[\text{old}]{*} Li \xrightarrow[\text{old}]{*} i \xrightarrow[\text{new}]{} j \xrightarrow[\text{old}]{*} Lj$. Furthermore $P_{1,6}$ holds, because of the definition of M^c_{int} and because $M^-(\kappa, Lj) = 0$ holds at line 18. Finally $P_{1,7}$ holds by means of Property 4.8.b. Lemma 6.3 will state that the following specification holds.

$$\{P_1\}\text{adapt}(\kappa)\{P_1 \wedge \text{there are no red leaders}\}.$$

Hence, at line 23, ($P_1 \wedge$ there are no red leaders) holds. Now consider conditions $P_{1,2}, P_{1,3}, P_{1,4}, P_{1,5}$ and $P_{1,1}$ respectively. Then by Property 4.8.b these conditions correspond to $P_{0,1}, P_{0,2}, P_{0,3}, P_{0,4}$ and $P_{0,0}$ respectively (cf. Section 3, Figure 2) if the condensed graph G^c is substituted in P_0 and if Property 4.8.c is applied. Therefore Lemma 3.3 yields

$$P_1 \wedge \text{there are no red leaders} \Rightarrow \bigwedge_{\mu \in V^c} [M^*(\kappa,\mu) = 1 \iff \kappa \xrightarrow[\text{new}]{*} \mu].$$

By using $P_{1,7}$ this establishes $R_1(\kappa)$. This yields together with $P_{1,8}$ that $R_2(\kappa)$ holds. Finally, $R_0(\kappa)$ and $R_3(\kappa)$ follow from $P_{1,1}$ and $P_{1,6}$. Therefore $R(\kappa)$ holds at line 23.

Combining the three above cases yields that $R(\kappa)$ holds at line 26. Moreover, it is seen that the validness of $I(\kappa')$ or $R(\kappa')$ for $\kappa' \neq \kappa$ does not change during a pass of this loop (w.r.t. κ)(cf. procedure adapt). This yields that we have at line 27, that

$$\bigwedge_{\kappa} [\kappa \in V^c : R(\kappa)]$$

holds. At line 28 two cases are distinguished. If no new cycle has arisen because of the insertion of (i,j), then all components remain the same. Therefore we find $V^c_{\text{old}} = V^c_{\text{new}}, C_{old} = C_{new}, L_{old} = L_{new}, e^c_{\text{old}} = e^c_{\text{new}}$ and $M^c_{\text{int}} = M^c_{\text{new}}$.

Therefore in $R_0(\kappa)$ all subscripts 'old' and 'int' can be replaced by 'new', and it follows that M^*, M^- and N are adjusted property, and that RR holds.

If a new cycle has arisen because of inserting (i,j), then newcycle = true, and procedure joincomponents(j) is called to adapt V^c, C, L, M^c and e^c, and to adapt M^- as well (since C changes). We consider this procedure now: it is given in Figure 14.

In line 2 and 3, $\Lambda = \{l \in V | j \xrightarrow[\text{new}]{*} l \wedge l \xrightarrow[\text{new}]{*} j\}$ and $\lambda_0 = \min \Lambda$ is established.

Property 6.1 *1.* $V^c_{\text{new}} = (V^c_{\text{old}} \backslash \Lambda) \cup \{\lambda_0\}$

2. $C_{new}(\lambda_0) = \Lambda = \bigcup_{\kappa \in V^c_{\text{old}} \cap \Lambda} C_{old}(\kappa)$

Figure 15:

P_1 is the conjunction of the following clauses.

$P_{1,1}$: $\kappa \in V^c_{\text{old}} \wedge L_{old}(j)$ is red or blue $\wedge$

$V^c = V^c_{\text{old}} \wedge C = C_{old} \wedge L = L_{old} \wedge e^c = e^c_{\text{old}}$

$\wedge M^- \sim_{CR} G_{\text{old}} \wedge M^c = M^c_{\text{int}} \wedge M = M_{\text{new}}$

For all leaders $\mu \in V^c_{\text{old}}$ the following clauses hold.

$P_{1,2}$: If μ is red, then

$M^*(\kappa, \mu) = 0 \wedge \neg(\kappa \xrightarrow[\text{old}]{*} \mu) \wedge \kappa \xrightarrow[\text{new}]{*} \mu$

$P_{1,3}$: If μ is blue, then

$M^*(\kappa, \mu) = 1 \wedge \neg(\kappa \xrightarrow[\text{old}]{*} \mu) \wedge \kappa \xrightarrow[\text{new}]{*} \mu$

$P_{1,4}$: If μ is neutral then

$M^*(\kappa, \mu) = 1 \iff \kappa \xrightarrow[\text{old}]{*} \mu$

$P_{1,5}$: If $\lambda \in V^c_{\text{old}}$ is a blue leader, then

$M^c_{\text{int}}(\lambda, \mu) > 0 \Rightarrow (\mu \text{ is blue or red}) \vee \kappa \xrightarrow[\text{old}]{*} \mu$

$P_{1,6}$: $M^-(\kappa, \mu) = 1 \iff \sum_{\lambda \in V^c_{\text{old}}} M^*(\kappa, \lambda) M^c_{\text{int}}(\lambda, \mu) = M^c_{\text{int}}(\kappa, \mu) > 0$

$P_{1,7}$: For all $k \in C_{old}(\kappa)$ and $m \in C_{old}(\mu)$

$M^*(k, m) = M^*(\kappa, \mu)$

$P_{1,8}$: For $k \in C_{old}(\kappa)$ and $m \in V$

$N(k, m) = \#\{(l, m) \in E_{\text{new}} | M^*(k, l) = 1\}$

---

3. $C_{new}(\kappa) = C_{old}(\kappa)$ *for* $\kappa \in V^c_{\text{old}} \backslash \Lambda$

Proof
By Lemma 3.1 we have for $\kappa \in V^c_{\text{old}}$ that

$$C_{old}(\kappa) \subseteq \{m \in V | \kappa \xrightarrow[\text{new}]{*} m \xrightarrow[\text{new}]{*} \kappa\}.$$

We prove that $C_{old}(\kappa) = \{m \in V | \kappa \xrightarrow[\text{new}]{*} m \xrightarrow[\text{new}]{*} \kappa\}$ for $\kappa \in V^c_{\text{old}} \backslash \Lambda$. Suppose $C_{old}(\kappa) \neq \{m \in V | \kappa \xrightarrow[\text{new}]{*} m \xrightarrow[\text{new}]{*} \kappa\}$ for some $\kappa \in V^c_{\text{old}}$. Then there is an m_0 such that

$$\kappa \xrightarrow[\text{new}]{*} m_0 \wedge m_0 \xrightarrow[\text{new}]{*} \kappa \wedge (\neg(\kappa \xrightarrow[\text{old}]{*} m_0) \vee \neg(m_0 \xrightarrow[\text{old}]{*} \kappa)).$$

Hence, Lemma 3.1 yields

$$\kappa \xrightarrow[\text{old}]{*} i \xrightarrow[\text{new}]{} j \xrightarrow[\text{old}]{*} m_0 \vee m_0 \xrightarrow[\text{old}]{*} i \xrightarrow[\text{new}]{} j \xrightarrow[\text{old}]{*} \kappa.$$

Since $\kappa \xrightarrow[\text{new}]{*} m_0 \wedge m_0 \xrightarrow[\text{new}]{*} \kappa$, this establishes that $\kappa \in \Lambda$. Hence, $C_{old}(\kappa) = \{m \in V | \kappa \xrightarrow[\text{new}]{*} m \xrightarrow[\text{new}]{*} \kappa\}$ for $\kappa \in V^c_{\text{old}} \backslash \Lambda$. By the definition Λ and λ_0, and by Definition 4.5 and Definition 4.7, the remainder of assertion follows. □

In line 4-9 all nonzero entries in M^c and those entries of M^- concerning some node in Λ are reset to zero, as is seen as follows. For $\kappa, \mu \in V^c_{\text{old}}, \kappa \neq \mu$ we have by $R_3(\kappa)$ that $M^-(\kappa, \mu) > 0$ (and $M^c(\kappa, \mu) > 0$) holds, only if there is at least one edge in E_{new} from $C_{old}(\kappa)$ to $C_{old}(\mu)$. Therefore, enumerating all edges (like in line 4-5) yields the claimed result w.r.t. V^c_{old}. The other nonzero entries of M^- concerning nodes of Λ are related to G^{cr}_{old} (since $M^- \sim_{CR} G_{\text{old}}$). Therefore these entries can be set to zero by using the structure of G^{cr}_{old} (cf. Definition 4.17). In line 10-13, L, C, M^c, e^c and V^c are adjusted. It is easily seen that afterwards $V^c = V^c_{\text{new}}, L = L_{new}, C = C_{new}, e^c = e^c_{\text{new}}$ and $M^c = M^c_{\text{new}}$ is established. In line 14-15, M^-–values are updated w.r.t. G^{cr}_{new} as far as $C_{new}(\lambda_0)$ is concerned. Since in procedure joincomponents, M^- is changed w.r.t. Λ only, we have by Prop. 6.1 that afterwards $M^- \sim_{CR} G_{\text{new}}$ holds.

By Corollary 4.21 it follows that in line 16-27 $M^-(\lambda_0, \cdot)$ and $M^-(\cdot, \lambda_0)$ are adjusted such that afterwards $M^-(\lambda_0, \mu) = M^-_{\text{new}}(\lambda_0, \mu)$ and $M^-(\mu, \lambda_0) = M^-_{\text{new}}(\mu, \lambda_0)$ hold for $\mu \in V^c_{\text{new}}$.

The above considerations yield that after execution of joincomponents(j) in line 28 of procedure insert(i, j), the following holds.
Firstly we have $V^c = V^c_{\text{new}}, L = L_{new}, C = C_{new}, e^c = e^c_{\text{new}}$ and $M^c = M^c_{\text{new}}$. Moreover, since M^* and N have not been changed by joincomponents(j), it follows from $\bigwedge_{\kappa \in V^c_{\text{old}}} [R(\kappa)]$ that $M^* = M^*_{\text{new}}$ and $N = N_{\text{new}}$.

Furthermore, it is stated above, that $M^-(\lambda_0, \cdot) = M^-_{\text{new}}(\lambda_0, \cdot)$ and $M^-(\cdot, \lambda_0) = M^-_{\text{new}}(\cdot, \lambda_0)$ holds for the new leader λ_0 and that $M^- \sim_{CR} G_{\text{new}}$. We now prove that $M^-(\kappa, \mu) = M^-_{\text{new}}(\kappa, \mu)$ for $\kappa, \mu \in V^c_{\text{new}} \backslash \{\lambda_0\}$. By Prop. 6.1 we find that $(V^c_{\text{old}} \backslash \Lambda) \cup \{\lambda_0\} = V^c_{\text{new}}$ and hence that $V^c_{\text{new}} \backslash \{\lambda_0\} = V^c_{\text{old}} \backslash \Lambda$. Now note that for $\kappa, \mu \in V^c_{\text{new}} \backslash \{\lambda_0\}$, procedure joincomponents has not changed $M^-(\kappa, \mu)$. Furthermore we have by Prop. 6.1 and

$V^c_{\text{new}} \backslash \{\lambda_0\} = V^c_{\text{old}} \backslash \Lambda$ that $M^c_{\text{int}}(\kappa,\mu) = M^c_{\text{new}}(\kappa,\mu)$ for such κ and μ. Therefore we have for such κ and μ (by using $R(\kappa)$ and $M^* = M^*_{\text{new}}$).

$$
\begin{aligned}
& M^-(\kappa,\mu) = 1 \\
& \iff \sum_{\lambda \in V^c_{\text{old}}} M^*(\kappa,\lambda) M^c_{\text{int}}(\lambda,\mu) = M^c_{\text{int}}(\kappa,\mu) > 0 \\
& \iff \sum_{\lambda \in V^c_{\text{old}} \backslash \Lambda} M^*(\kappa,\lambda) M^c_{\text{int}}(\lambda,\mu) + \sum_{\lambda \in \Lambda \cap V^c_{\text{old}}} M^*(\kappa,\lambda) M^c_{\text{int}}(\lambda,\mu) = M^c_{\text{int}}(\kappa,\mu) > 0 \\
& \iff \sum_{\lambda \in V^c_{\text{new}} \backslash \{\lambda_0\}} M^*(\kappa,\lambda) M^c_{\text{new}}(\lambda,\mu) + M^*(\kappa,\lambda_0) M^c_{\text{new}}(\lambda_0,\mu) = M^c_{\text{new}}(\kappa,\mu) > 0 \\
& \iff \sum_{\lambda \in V^c_{\text{new}}} M^*_{\text{new}}(\kappa,\lambda) M^c_{\text{new}}(\lambda,\mu) = M^c_{\text{new}}(\kappa,\mu) > 0
\end{aligned}
$$

Indeed, since it is easily verified that $M^c_{\text{new}}(\lambda_0,\mu) = \sum_{\lambda \in \Lambda \cap V^c_{\text{old}}} M^c_{\text{int}}(\lambda,\mu)$ holds and since the definition of Λ yields that $M^*_{\text{new}}(\kappa,\lambda_0) = M^*_{\text{new}}(\kappa,\lambda)$ for $\lambda \in V^c_{\text{old}} \cap \Lambda$.

By combining the previous results concerning M^- and by using Corollary 4.23, we find $M^- = M^-_{\text{new}}$.

Conclusion: condition RR holds at line 28 of procedure insert(i,j).

By the above three case analyses it follows that RR holds at line 29. This proves the following theorem.

Theorem 6.2 *Procedure insert(i,j) updates all information summarized in Section 5 correctly for inserting the new edge (i,j) in the graph G_{old}.*

We are left to prove the following lemma.

Lemma 6.3 *The following specification holds:*

$$\{P_1\}\, adapt(\kappa)\, \{P_1 \wedge \textit{there are no red leaders}\}.$$

Proof

We first show that P_1 is an invariant for the do-loop of line 3-17. In line 4-6, a red leader λ_0 is chosen and λ_0 is coloured blue. Notice that $\lambda_0 \neq \kappa$ because of $P_{1,2}$ and $\kappa \xrightarrow[\text{old}]{*} \kappa$. Now, the invariant P_1 is "repaired" in line 7-16 as follows. In fact, only $P_{1,3}$ and possibly $P_{1,5}$ are violated at this moment, because of the change of λ_0 from red to blue. In line 7, $P_{1,3}$ is repaired, which possibly causes the violation of $P_{1,6}$ and $P_{1,8}$. In line 8-11, $P_{1,8}$ is re-established. Therefore, only $P_{1,5}$ and $P_{1,6}$ still need to be repaired. Apparently, conditions $P_{1,5}$ and $P_{1,6}$ can only have been violated for $\mu \in V^c$ with $M^c_{\text{int}}(\lambda_0,\mu) > 0$, since the only changes made thus far, relevant for $P_{1,5}$ and $P_{1,6}$, are the change of colour of λ_0 and the change of $M^*(\kappa,\lambda_0)$ from 0 to 1. Therefore $P_{1,5}$ and $P_{1,6}$ are re-established in line 12-16 by considering every $\mu_0 \in S^c_{\text{int}}(\lambda_0)$ and by distinguishing two cases.

(a) $\mu_0 \in S^c_{\text{int}}(\lambda_0) \wedge M^*(\kappa,\mu_0) = 1$. By $M^*(\kappa,\mu_0) = 1$ and by $P_{1,4}$ it follows that $P_{1,5}$ holds for $\lambda = \lambda_0$ and $\mu = \mu_0$. By $M^*(\kappa,\lambda_0) = 1, M^c_{\text{int}}(\lambda_0,\mu_0) > 0, \lambda_0 \neq \kappa$ and $M^*(\kappa,\kappa) = 1$, we have $\sum_{\lambda \in V^c} M^*(\kappa,\lambda) M^c_{\text{int}}(\lambda,\mu_0) \geq M^c_{\text{int}}(\lambda_0,\mu_0) + M^c(\kappa,\mu_0) > M^c(\kappa,\mu_0)$. Therefore, after setting $M^-(\kappa,\mu_0) := 0$, $P_{1,6}$ holds for $\mu = \mu_0$.

(b) $\mu_0 \in S^c_{\text{int}}(\lambda_0) \wedge M^*(\kappa, \mu_0) = 0$. Because of $P_{1,2}$ and $P_{1,4}$ we have $\neg(\kappa \xrightarrow[\text{old}]{*} \mu_0)$. Since $\kappa \xrightarrow[\text{new}]{*} \lambda_0$ and $M^c_{\text{int}}(\lambda_0, \mu_0) > 0$ we have $\kappa \xrightarrow[\text{new}]{*} \mu_0$. Therefore, colouring μ_0 red (line 14) causes $P_{1,5}$ to hold for $\lambda = \lambda_0, \mu = \mu_0$ while other valid conditions remain valid (particularly $P_{1,3}$). We show that $P_{1,6}$ holds for $\mu = \mu_0$. Since $\neg(\kappa \xrightarrow[\text{old}]{*} \mu_0)$ holds, we find $M^c_{\text{old}}(\kappa, \mu_0) = 0$. We again distinguish two cases.

- $M^c_{\text{old}}(\kappa, \mu_0) = M^c_{\text{int}}(\kappa, \mu_0) = 0$. Since $P_{1,6}$ holds at line 4 we have $M^-(\kappa, \mu_0) = 0$. Therefore $P_{1,6}$ holds at line 13 too.
- $M^c_{\text{old}}(\kappa, \mu_0) = 0 \wedge M^c_{\text{int}}(\kappa, \mu_0) = 1$. We show that this case cannot occur by deriving a contradiction. We must have $\kappa = L_{old}(i)$ and $\mu_0 = L_{old}(j)$. Since $\kappa \xrightarrow[\text{new}]{*} \lambda_0 \wedge \neg(\kappa \xrightarrow[\text{old}]{*} \lambda_0)$ holds, Prop. 3.1 yields that $j \xrightarrow[\text{old}]{*} \lambda_0$ and hence $\mu_0 \xrightarrow[\text{old}]{*} \lambda_0$. By $\mu_0 \in S^c_{\text{int}}(\lambda_0)$ and $\lambda_0 \neq \kappa = L_{old}(i)$ we find $\mu_0 \in S_{old}(\lambda_0)$ and hence $\mu_0 \xrightarrow[\text{old}]{*} \lambda_0 \xrightarrow[\text{old}]{*} \mu_0$. However, $\lambda_0 \neq \mu_0$ (since $M^*(\kappa, \lambda_0) = 1$ and $M^*(\kappa, \mu_0) = 0$), $\lambda_0, \mu_0 \in E^c_{\text{old}}$ and G^c_{old} is acyclic. Contradiction.

By the above cases analysis it follows that $P_{1,6}$ holds for $\mu = \mu_0$.

From a) and b) it follows that P_1 holds at line 16. Therefore, P_1 is an invariant of the do-loop in line 3-17. Finally, the loop terminates, since at each pass of the loop a red leader is coloured blue and no blue leader is ever coloured red. Therefore the following holds at line 18:

$$P_1 \wedge \text{there are no red leaders.}$$

□

6.3 Complexity

We consider the time complexity of a number q of consecutive insertions of edges in a graph $G_{\text{old}} =< V, E_{\text{old}} >$, resulting in a graph $G_{\text{new}} =< V, E_{\text{new}} >$. (Hence procedure insert is performed q times). Let $n = \#V, e_{\text{old}} = \#E_{\text{old}}$ and $e_{\text{new}} = \#E_{\text{new}}$. We refer to the result graph after t inserions by $G_t =< V, E_t >$. (Hence, $G_0 = G_{\text{old}}$ and $G_q = G_{\text{new}}$).

We compute the cost of q insertions by considering the total net costs for q insertions for each of the three procedures separately (i.e., for insert, adapt and resetcomponents). That is, each invocation of procedure adapt or joincomponents is charged to procedure insert for an amount of $O(1)$ only. All costs exceeding an amount of $O(1)$ for the execution of procedure adapt or joincomponents are charged to these procedures themselves.

Finally, we compute the cost starting from the assumption that couloured nodes are recorded in sets that represent the colours (cf. Section 5 for the implementation of sets).

6.3.1 Analysis of procedure insert

In line 23 all coloured nodes are uncoloured. We charge the cost of discolouring a coloured node to the cost of colouring a node. Therefore the cost of colouring a node is increased by $O(1)$.

Obviously, in this way each execution of procedure insert is charged for $\mathcal{O}(n)$ time. Procedure insert is called $q = e_{\text{new}} - e_{\text{old}}$ times. Therefore the total net costs for q insertions charged to procedure insert is $O(n \cdot q) = O(n \cdot e_{\text{new}})$.

6.3.2 Analysis of procedure adapt

Consider the processing of a red leader λ_0 during one pass of the do-loop (line 3-17) in procedure adapt. Such a pass contains the execution of some simple statements (line 3,4 and 6) of $O(1)$ time. (The coloured nodes can be recorded in sets).

Moreover it contains the execution of 3 for-loops (line 7 and 8-16). Because $\kappa \in C(\kappa)$ and $\lambda_0 \in C(\lambda_0)$, the costs of the simple statements do not exceed the costs of the first for-loop (line 7) in order.

By the definitions of $S^c(\lambda_0), C(\lambda_0)$ and $S(l)$ for $l \in C(\lambda_0)$ it follows that the cost of the third for-loop (line 12-16) does not exceed the costs of the second for loop (line 8-11). Therefore it suffices to consider the complexity of the first and second for-loop only. Suppose red leader λ_0 is processed in a call adapt(κ) during the t^{th} insertion of an edge ($1 \leq t \leq q$). Because of $P_{1,2}$ and $P_{1,7}$ we have in line 5

$$\bigvee_{k \in C_{t-1}(\kappa)} \bigvee_{m \in C_{t-1}(\mu)} M^*(k,m) = 0.$$

Therefore, in each pass of the for-loop in line 7 a M^*−value is increased from 0 to 1.

Since no procedure decreases a M^*−value from 1 to 0, a M^*−value can be increased in the for-loop of line 7 only once during all executions of procedure adapt. Therefore this for-loop costs O(number of increases of M^*) $= O(e_q^* - e_0^*)$ for all executions of procedure adapt together. (Where $e_q^* = \#E_q^*$ and $e_0^* = \#E_0^*$).

Now consider the second for-loop during the processing of λ_0 in adapt(κ). Since $\lambda_0 \in C_{t-1}(\lambda_0), \kappa \in C_{t-1}(\kappa)$ and $(\#S(l) = 0 \Longrightarrow \#C_{t-1}(\lambda_0) = 1)$ for $l \in C_{t-1}(\lambda_0)$, we find that apart from an amount of $O(1)$ cost (that can be charged to the first for-loop), the costs of this for-loop is bounded by the number of executions of the statement $N(k,m) := N(k,m) + 1$ in line 10. Since we have that

$$\sum_{k,m} N_0(k,m) \geq 0$$

and

$$\sum_{k,m} N_q(k,m) \leq n \cdot e_q$$

(by the definition of N), and since N−values are never decreased in the procedures, it follows that the total costs of the second for-loop is $\mathcal{O}(n \cdot e_q)$ for all executions of procedure adapt together . Therefore we conclude that the total net costs for q insertions charged to procedure adapt are $O(e_q^* - e_o^*) + O(n \cdot e_q) = O(n \cdot e_q) = O(n \cdot e_{\text{new}})$.

6.3.3 Analysis of procedure joincomponents

We derive a bound for the costs of a single execution of procedure joincomponents, say during the t^{th} insertion.

The statements in line 2, 3, 6-9,10, 11, 13 and 14-15 can be performed in $O(n)$ time. (Indeed, line 6-9 can be performed in $O(n)$ time, since the ordered lists $C(\kappa_1)$ and $C(\kappa_2)$ are disjoint for $\kappa_1 \neq \kappa_2$. Moreover, Λ can be implemented by a set (by means of an ordered list and an index-array) in $O(n)$ time.) The statements in line 4-5 and 12 can be performed in $O(n + e_t)$ time. Finally, the statements in line 16-21 and 22-27 can be performed in $O(e_t)$ since $\#E_t^c \leq \#E_t$ and since the breadth first searches start in λ only. Hence, one execution of procedure joincomponents takes $O(n + e_{\text{new}})$ time.

Property 6.4 *Let n_{new}^{cyc} be the number of nodes that are contained in the non-trivial components of G_{new} (i.e. a component with at least 2 nodes). Then procedure joincomponents can be called at most $n_{new}^{cyc} - 1$ times during the insertions from G_{old} to G_{new}.*

Proof
Graph G_{new} contains at least $n - n_{new}^{cyc} + 1$ (possibly trivial) components (i.e., containing at least one node), since the n_{new}^{cyc} nodes contained in the non-trivial components form at least one component, and since the other $n - n_{new}^{cyc}$ nodes form $n - n_{new}^{cyc}$ components. Every time that procedure joincomponents is called starting from G_{old}, the number of (possibly trivial) components decreases. (Since at least two 'old' components are joined to one new component.) Therefore procedure joincomponents can be called at most $n_{new}^{cyc} - 1$ times. □

Hence, the total net costs for q insertions charged to procedure joincomponents is $O(n_{new}^{cyc} \cdot (e_{\text{new}} + n)) = O(n \cdot e_{\text{new}})$ (since $n_{new}^{cyc} \leq e_{\text{new}}$).

6.3.4 Complexity for q insertions

By combining the results of subsections 6.3.1, 6.3.2 and 6.3.3, we have proved the following theorem.

Theorem 6.5 *For a number of q consecutive edge-insertions procedure insert takes $O(n \cdot e_{\text{new}})$ time, where e_{new} is the number of edges in the result graph.*

Remark that all entities can be intialised for G_{old} by just starting from the empty graph $< V, \emptyset >$ and building G_{old} by edge insertions. This takes $O(n \cdot e_{\text{old}})$ time starting from entities with initial values zero. (Moreover, this initialization cost can be incorporated in the complexity considerations of procedure insert by imagining that $G_{\text{old}} =< V, \emptyset >$.) Since the space complexity of the information is $O(n^2)$, we have proved the following theorem.

Theorem 6.6 *The initialization of the information summarized in Section 5 takes $O(n \cdot e_{\text{old}} + n^2)$ time, where e_{old} is the number of edges in the initial graph.*

7 Edge deletions

Consider the problem description of Section 5. Suppose $(i,j) \in E_{\text{old}}$ and (i,j) is deleted from G_{old}. Then we have $G_{\text{new}} =< V, E_{\text{old}} \backslash \{(i,j)\} >$. Below we present procedure delete(i,j) for performing the task described in Section 5 for the above deletion. Comments

and a correctness proof are given in subsection 7.2, while complexity is computed in subsection 7.3.

7.1 Procedure delete(i,j)

The procedure delete and its subprocedures are given in Figure 16-25. In these procedures the colours (white, grey, black and purple) of nodes on the one side and the marking of nodes on the other side, must be distinguished.

The symbols $\underline{\wedge}$ and $\underline{\vee}$ denote the "conditional and"(cand) and the "conditional or"(cor) operator respectively.

7.2 Comment and correctness proof

The operation of procedure delete(i,j) on the acyclic graph $G^c_{\mathbf{new}}$ can be compared to the operation of procedure delete* on an acyclic graph; i.e., it handles components (componentleaders) like delete* handles nodes. However, delete (i,j) is more complicated because of the possibility that components fall apart (cf. line 4 and line 13) and because of the need of limiting the number of calculations in procedure adjust (in particular w.r.t. N^c-values). The former case is tackled by using a set NEW, that records the leaders of the newly formed components. The transitive reduction w.r.t. these leaders is adjusted only at the end of the procedure. The latter case is tackled by computing N^c-values only when they are needed and by applying a kind of backward search strategy in procedure delete, using white, grey, black and purple leaders. Generally speaking, the black leaders are leaders that have been processed already, whereas grey leaders must still be processed, white leaders might be processed and purple leaders need not to be processed at all. This enables the use of an array $h(k : 1 \leq k \leq n)$, where $h(\kappa)$ is a black successor leader of $\kappa \in V^c(\kappa \neq L(i))$, by means of which the number of calculations of N^c-values can be reduced. However, we only record computed N^c-values temporary, since recording computed N^c-values does not decrease the order of time complexity while it causes more convenience in our considerations.

We now prove that procedure delete(i,j) performs the task described in Section 5 for deleting the (old) edge (i,j)

We first prove that RR holds in line 9 and that TT holds in line 15, where TT is given in Figure 26. Note that RR denotes the condition we want to establish with procedure delete (i,j). (RR is stated in subsection 6.2)

Figure 16:

```
(1)  Procedure delete(i, j);
(2)      M(i,j) := 0; Li := L(i); Lj := L(j);
(3)      for all k ∈ V ⟶ N(k,j) := N(k,j) − M*(k,i) rof
(4)      if Li = Lj ⟶ continue:=componentsbreak
(5)      [] Li ≠ Lj ⟶ continue:= (M^c(Li, Lj) = 1)
(6)      fi;
(7)      if ¬ continue
(8)          ⟶ if Li ≠ Lj ⟶ M^c(Li, Lj) := M^c(Li, Lj) − 1 [] Li = Lj ⟶ e^c(j) := e^c(j) − 1 fi
(9)                {RR}
(10)     [] continue
(11)         ⟶ if Li ≠ Lj ⟶ M^c(Li, Lj) := M^−(Li, Lj) := 0; NEW:=∅;
(12)                colournodes {TT}
(13)           [] Li = Lj ⟶ resetcomponents {TT}
(14)           fi;
(15)           {TT}
(16)           do there are grey leaders
(17)               ⟶ let κ0 be a grey leader; mark Lj; neutralNC;
(18)                     {Q1 ∧ TT(κ0)}
(19)                     adjust(κ0);
(20)                     {Q1 ∧ TT(κ0) ∧ there are no marked leaders }
(21)                     if M*(κ0, Lj) = 0 ⟶ for all φ ∈ P^c(κ0) with φ white
(22)                                          ⟶ colour φ grey; h(μ) := κ0
(23)                                       rof
(24)                     fi;
(25)                     colour κ0 black {TT}
(26)           od;
(27)           {TT∧ there are no grey leaders }
(28)           if NEW ≠ ∅ ⟶ adjusttransitivereduction {RR}
(29)           [] NEW = ∅ ⟶ skip {RR}
(30)           fi; {RR}
(31)           discolour all leaders {RR}
(32)     fi
(33)     {RR}
```

Figure 17:

(1) **Procedure** adjust (κ_0);
(2) $\{Q_1 \wedge TT(\kappa_0)\}$
(3) **do** there are marked leaders
(4) $\longrightarrow$ let ν_0 be a marked leader;
(5) dismark ν_0; $\{Q_1'\}$
(6) **if** $\kappa_0 \neq Li \triangleq M^*(h(\kappa_0), \nu_0) = 1 \longrightarrow$ skip $\{Q_1 \wedge TT(\kappa_0)\}$
(7) $[]$ $\kappa_0 = Li \veebar M^*(h(\kappa_0), \nu_0) = 0$
(8) $\longrightarrow$ **if** value$NC(\kappa_0, \nu_0) > 0$
(9) $\longrightarrow \{Q_1' \wedge NC(\nu_0) > 0\}$
(10) **if** value$NC(\kappa_0, \nu_0) = M^c(\kappa_0, \nu_0) \wedge \kappa_0 \notin$ NEW $\wedge \mu \notin$ NEW $\longrightarrow M^-(\kappa_0, \mu) := 1$
(11) $[]$ value$NC(\kappa_0, \nu_0) > M^c(\kappa_0, \nu_0) \vee \kappa_0 \in$ NEW $\vee \mu \in$ NEW $\longrightarrow$ skip
(12) **fi**$\{Q_1 \wedge TT(\kappa_0)\}$
(13) $[]$ value $NC(\kappa_0, \nu_0) = 0$
(14) $\longrightarrow \{Q_1' \wedge NC(\nu_0) = 0\}$
(15) disconnect (κ_0, ν_0);
(16) **for all** $n_0 \in C(\nu_0)$
(17) $\longrightarrow$ **for all** $n' \in S(n_0)$
(18) $\longrightarrow$ **for all** $k \in C(\kappa_0)$
(19) $\longrightarrow N(k, n') := N(k, n') - 1$
(20) **rof rof rof**;
(21) **for all** $\mu_0 \in S^c(\nu_0) \backslash \{\kappa_0\} \longrightarrow$ mark μ_0 **rof**;
(22) **for all** $\mu_0 \in S^c(\nu_0) \longrightarrow$ decrease$NC(\nu_0, \mu_0)$ **rof**
(23) $\{Q_1 \wedge TT(\kappa_0)\}$
(24) **fi fi**;
(25) $\{Q_1 \wedge TT(\kappa_0)\}$
(26) **od**;
(27) $\{Q_1 \wedge TT(\kappa_0) \wedge$ there are no marked leaders $\}$

Figure 18:

function componentsbreak;
{ detects whether the deletion of edge (i, j) with $L(i) = L(j)$
causes a burst in the component of i and j }.
* if $i \neq j$ then apply a depth first search within component $C(L(i))$, starting at i,
to detect whether this component falls apart in new components;
if it does, componentsbreak := true, otherwise false;
* if $i = j$ then componentsbreak := false.

Figure 19:

```
Procedure colournodes;
    for all κ ∈ V^c ⟶ if M*(κ, Li) = 0 ⟶ colour κ purple
                     [] M*(κ, Li) = 1 ⟶ colour κ white
                     fi
    rof;
    colour Li grey
```

Figure 20:

```
(1)  Procedure resetcomponents; {L(i) = L(j); (13) holds}
(2)      *   set M⁻(k1, k2) := 0  for k1, k2 ∈ C(L(i)) by traversing
(3)          the ordered list C(L(i));
(4)          set M⁻(L(i), μ) := M⁻(μ, L(i)) := 0 for μ ∈ V^c
(5)      *   set M^c(κ, μ) := 0, P^c(κ) := S^c(κ) := ∅ (by removing
(6)          the elements) and e^c(k) := 0 for κ, μ ∈ V^c, k ∈ V,
(7)          by enumerating all edges in E_new
(8)      *   apply a depth first search inC(L(i)) to determine
(9)          the new components arisen from the breakage of C(L(i));
(10)         adjust C, L and V^c according to these new
(11)         components { then C = C_new, L = L_new and V^c = V^c_new}
(12)         { (cf. [4]) }
(13)     *   NEW:= {λ ∈ V^c_new | M*(i, λ) = M*(λ, i) = 1}; Li := L(i); Lj := L(j);
(14)         { NEW is the set of leaders of the newly formed components }
(15)     *   set M^c, e^c, P^c and S^c according to M^c_new, e^c_new
(16)         P^c_new and S^c_new by enumerating all edges in
(17)         E_new
(18)     *   adjust M⁻ w.r.t. G^cr_new by traversing the ordered
(19)         lists C(λ) for λ ∈ NEW
(20)     *   colournodes
(21)     *   colour Lj black
(22)     *   {TT}
```

Figure 21:

```
(1)  Procedure adjusttransitivereduction
(2)      *      for all κ ∈ NEW
(3)             ⟶ apply a breadth first search in G^c_new starting at κ,
(4)                 to detect those μ ∈ V^c_new that are reachable
(5)                 from κ by the direct condensed edge (κ,μ) ∈ E^c_new
(6)                 only;
(7)                 for such μ set M⁻(κ,μ) := 1; for other μ ∈ V^c_new,
(8)                 M⁻(κ,μ) = 0
(9)             rof
(10)     *      for all μ ∈ NEW
(11)            ⟶ apply a backwards breadth first search in G^c_new
(12)                starting at μ, to detect those κ ∈ V^c_new
(13)                from which μ is reachable by the direct condensed edge
(14)                (κ,μ) ∈ E^c_new only;
(15)                for such κ, set M⁻(κ,μ) := 1; for other κ ∈ V^c_new
(16)                M⁻(κ,μ) = 0
(17)            rof
```

Figure 22:

```
Procedure disconnect(κ,ν);
    for all k ∈ C(κ)
        ⟶ for all n ∈ C(ν)⟶M*(k,n) := 0 rof
    rof
```

Figure 23:

```
Function valueNC(κ,ν);
    if NC(ν) = −1  ⟶  NC(ν) := Σ_{n∈C(ν)}(N(κ,n) − e^c(n))
    [] NC(ν) ≥ 0   ⟶  skip
    fi;
    valueNC(κ,ν) := NC(ν)
```

Figure 24:

Procedure decrease$NC(\nu, \mu)$;
if $NC(\mu) = -1 \longrightarrow$ skip
$[]\ NC(\mu) \geq 0 \longrightarrow NC(\mu) := NC(\mu) - M^c(\nu, \mu)$
fi

Lemma 7.1 *Condition RR holds in line 9 and condition TT holds in line 15.*

Proof
Suppose all information is related to G_{old} at line 1 (cf. Section 5).We consider procedure delete up to and including line 15. For the input G_{old}, two cases need to be distinguished. We shall show that in both cases either line 9 is reached and RR holds, or line 15 is reached and TT holds.

Case a:
Edge (i, j) is not interior to any component. Then we have

$$L_{old}(i) \neq L_{old}(j),\ M^c_{\text{old}}(L_{old}(i), L_{old}(j)) > 0 \text{ and } V^c_{\text{new}} = V^c_{\text{old}},$$
$$C_{new} = C_{old}, L_{new} = L_{old} \text{ and } e^c_{\text{new}} = e^c_{\text{old}}. \tag{3}$$

In the following (of this case) we write L instead of L_{old} or L_{new}. We also have

$$M^c_{\text{new}}(k, m) = M^c_{\text{old}}(k, m) \text{ for } (k, m) \neq (L(i), L(j))$$
$$M^c_{\text{new}}(L(i), L(j)) = M^c_{\text{old}}(L(i), L(j)) - 1. \tag{4}$$

We again distinguish two cases.

- $M^c_{\text{old}}(L(i), L(j)) > 1$. Then $M^c_{\text{new}}(L(i), L(j)) > 0$ and hence $i \xrightarrow[\text{new}]{*} j$. By Prop. 3.6 this gives $M^*_{\text{old}} = M^*_{\text{new}}$. Consider Cor. 4.23 for M^-_{old}. By using (4), $M^c_{\text{new}}(L(i), L(j)) > 0, M^*_{\text{old}} = M^*_{\text{new}}$ and $C_{old} = C_{new}$ we find (by using Cor. 4.23 again for M^-_{new}) that $M^-_{\text{old}} = M^-_{\text{new}}$. Therefore, only M, M^c and N need to updated (cf. Lemma 2.3) in this case. It is easily seen that in this case the update is properly performed by the algorithm, by executing line 2,3 and 5-9. Hence, in this case line 9 is reached and RR holds.
- $M^c_{\text{old}}(L(i), L(j)) = 1$. Then $M^c_{\text{new}}(L(i), L(j)) = 0$, and apart from M, M^c and N, M^* and M^- may need to be updated. In this case the algorithm performs line 2,3,5 and 10-12. Now, condition TT holds in line 12.

Figure 25:

Procedure neutral NC;
for all $k \in V$ **with** $NC(k) \neq -1 \longrightarrow NC(k) := -1$ **rof**;

Figure 26:

TT is the conjunction of the following clauses.

T_1 : $V^c = V^c_{\text{new}} \wedge L = L_{new} \wedge C = C_{new} \wedge M^c = M^c_{\text{new}} \wedge e^c = e^c_{\text{new}}$
$\wedge M = M_{\text{new}} \wedge M^- \sim_{CR} G_{\text{new}}$

T_2 : $Li = L_{new}(i) \neq L_{new}(j) = L(j) \wedge Li$ is black or grey
$\wedge (Lj \xrightarrow[\text{new}]{*} Li \Longrightarrow Lj$ is black$) \wedge$ NEW $= \{\lambda \in V^c_{\text{new}} | Lj \xrightarrow[\text{new}]{*} \lambda \xrightarrow[\text{new}]{*} Li\}$

For all leaders $\kappa, \mu \in V^c_{\text{new}}$ the following clauses hold.

T_3 : κ is black, grey, white or purple. Moreover
$\kappa \xrightarrow[\text{new}]{*} Li \Longleftrightarrow \kappa$ is black, grey or white

T_4 : If κ is black or purple,then
$M^*(\kappa, \mu) = 1 \Longleftrightarrow \kappa \xrightarrow[\text{new}]{*} \mu$

T_5 : If κ is grey or white, then
$M^*(\kappa, \mu) = 1 \Longleftrightarrow \kappa \xrightarrow[\text{new}]{*} \mu \vee Lj \xrightarrow[\text{new}]{*} \mu$

T_6 : If κ is black, then
$\mu \in P^c_{\text{new}}(\kappa) \wedge M^*(\kappa, Lj) = 0 \Longrightarrow \mu$ is black or grey

T_7 : If κ is grey and $\kappa \neq Li$, then
$h(\kappa) \in S^c_{\text{new}}(\kappa) \wedge h(\kappa)$ is black

T_8 : For all $k \in C(\kappa)$ and $m \in C(\mu)$
$M^*(k, m) = M^*(\kappa, \mu)$

T_9 : For all $k \in C(\kappa)$ and $m \in C(\mu)$
$N(k, m) = \{(l, m) \in E_{\text{new}} | M^*(k, l) = 1\}$

T_{10} : If κ is black or purple, then
$M^-(\kappa, \mu) = 1 \Longleftrightarrow \sum_{\lambda \in V^c} M^*(\kappa, \lambda) M^c_{\text{new}}(\lambda, \mu) = M^c_{\text{new}}(\kappa, \mu) > 0$
$\wedge \kappa \notin$ NEW $\wedge \mu \notin$ NEW

T_{11} : If κ is grey or white, then
$M^-(\kappa, \mu) = 1 \Longleftrightarrow \sum_{\lambda \in V^c} M^*(\kappa, \lambda) M^c_{\text{new}}(\lambda, \mu) = M^c_{\text{new}}(\kappa, \mu) > 0$
$\wedge \kappa \notin$ NEW $\wedge \mu \notin$ NEW $\wedge \mu \neq Lj$.

Consider line 12. We show that TT holds. Because of (3), line 11 and line 2, T_1 and the first conjunct of T_2 hold. (Notice that $M^- \sim_{CR} G_{\text{new}}$ holds because of $M^- = M^-_{\text{old}}, C_{old} = C_{new}$ and Cor. 4.23.) By Lemma 3.6, $i \in C(L(i))$ and $j \in C(L(j))$, we find for $k, m \in V$

$$k \xrightarrow[\text{old}]{*} m \iff k \xrightarrow[\text{new}]{*} m \vee (k \xrightarrow[\text{new}]{*} L(i) \wedge L(j) \xrightarrow[\text{new}]{*} m). \tag{5}$$

Hence, the execution of procedure colournodes at line 12 causes T_3 and the second conjunct of T_2 to be true, where procedure colournodes is given in Figure 19. (Remark that $M^* = M^*_{\text{old}}$ at line 1 and that M^* is not changed in line 2-12.) Since $L(i) \neq L(j)$ and $(i,j) \in E_{\text{old}}$ we have $\neg(L(j) \xrightarrow[\text{old}]{*} L(i))$ and hence (by (5)) $\neg(L(j) \xrightarrow[\text{new}]{*} L(i))$. Therefore we find by means of line 11 (yielding NEW $= \varnothing$), that T_2 holds entirely. Conditions T_4 and T_5 follow by means of (5), T_3 and $M^* = M^*_{\text{old}}$. Conditions T_6, T_7 and T_8 hold trivially, whereas T_9 holds because of line 3 and Lemma 2.3. Finally, we have for $\kappa, \mu \in V^c_{\text{old}}$ (by Cor. 4.23 and $M^* = M^*_{\text{old}}$.)

$$M^-_{\text{old}}(\kappa, \mu) = 1 \iff \sum_{\lambda \in V^c_{\text{old}}} M^*(\kappa, \lambda) M^c_{\text{old}}(\lambda, \mu) = M^c_{\text{old}}(\kappa, \mu) > 0. \tag{6}$$

For purple leaders κ, T_3 and T_4 yield $M^*(\kappa, L(i)) = 0$. Therefore (6) and (4) give that T_{10} holds, since $M^- = M^-_{\text{old}}$ holds at line 1, Li is not purple (w.r.t. line 11) and there are no black nodes. For white and grey leaders κ, T_3 and T_5 yield $M^*(\kappa, L(i)) = 1$. By means of (6), (4) and line 11 (yielding $M^-(L(i), L(j)) = 0$ and NEW $= \varnothing$) it is seen that T_{11} holds. Hence we may conclude that TT holds at line 12. Therefore, in this case line 15 is reached and TT holds.

Case b

Edge (i,j) is interior to component $C_{\text{old}}(L_{\text{old}}(i))$. Then $L_{\text{old}}(i) = L_{\text{old}}(j)$. Deletion of edge (i,j) may cause that $\neg(i \xrightarrow[\text{new}]{*} j)$ and hence it may cause that component $C_{\text{old}}(L_{\text{old}}(i))$ falls apart. (I.e. $C_{\text{new}}(L_{\text{new}}(i)) \neq C_{\text{old}}(L_{\text{old}}(i))$). In line 4 procedure componentsbreak (cf. Figure 18) computes whether this is the case or not, and returns value true or false respectively. (Indeed, since there is a path from i to j iff there is a path from i to j within the component.) We again distinguish two cases.

- The deletion of (i,j) does not break the component. Then we have $G^c_{\text{new}} = G^c_{\text{old}}, C_{new} = C_{old}, M^c_{\text{new}} = M^c_{\text{old}}$ and $i \xrightarrow[\text{new}]{*} j$. By Lemma 3.6 we therefore find that $M^*_{\text{new}} = M^*_{\text{old}}$ holds. By $C_{old} = C_{new}$ and $G^c_{\text{old}} = G^c_{\text{new}}$, Def. 4.16 yields that $M^-_{\text{new}} = M^-_{\text{old}}$ holds. Therefore, only M, N and e^c need to be updated. The algorithm performs this task in line 2, 3, 4 and 8 and finally terminates in line 9. (Since, in this case $Li = Lj$ and componentsbreak = false.) Hence, in this case line 9 is reached and RR holds.

- The deletion of (i,j) breaks component $C_{old}(L_{old}(i))$ into pieces. Then we must have $\neg(i \xrightarrow[\text{new}]{*} j) \wedge (j \xrightarrow[\text{new}]{*} i)$. Hence we can write

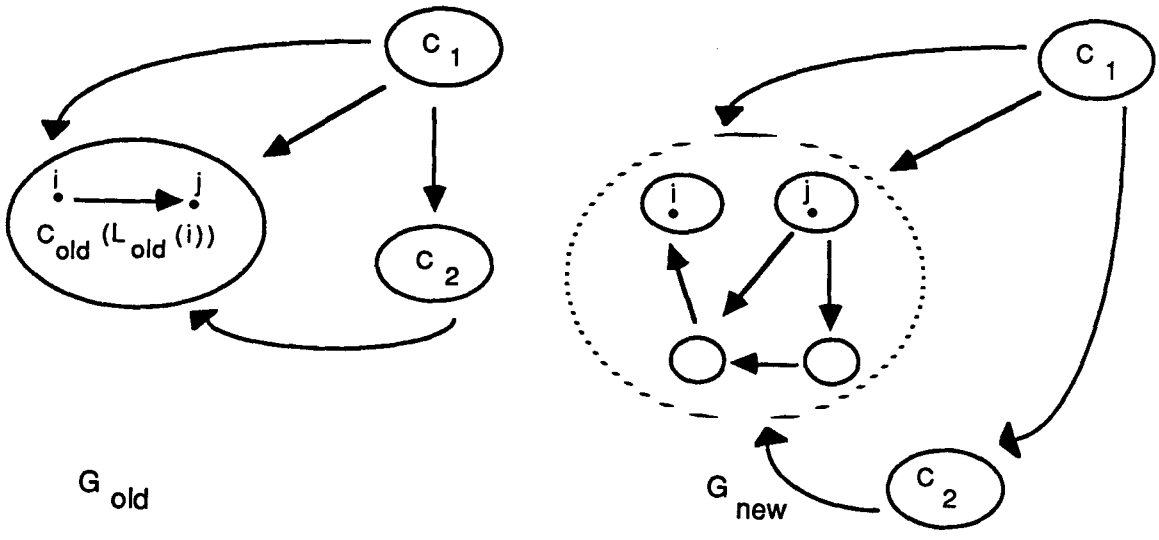
$$L_{new}(i) \neq L_{new}(j) \wedge L_{new}(j) \xrightarrow[\text{new}]{*} L_{new}(i). \tag{7}$$

In this case, the algorithm performs line 2, 3, 4 (yielding continue = true) and 13 (executing procedure resetcomponents). In Claim 7.2 it is stated that TT holds in line 13 as a postcondition of the call of procedure resetcomponents. (Procedure resetcomponents is given in Figure 20.) Therefore we may conclude that in this case line 15 is reached and TT holds.

Claim 7.2 *In the above case, TT holds as a postcondition of the call of procedure resetcomponents in line 13.*

Proof
The situation in this case is illustrated by Fig. 27. We first derive some properties

Figure 27:



for this situation. Define

$$\text{NEW} := \{\lambda \in V^c_{\text{new}} | i \xrightarrow[\text{old}]{*} \lambda \xrightarrow[\text{old}]{*} i\}. \tag{8}$$

Then the following property holds.

Property 7.3 *1.* $V^c_{\text{new}} = V^c_{\text{old}} \cup NEW$; $V^c_{\text{old}} \cap NEW = \{L_{\text{old}}(i)\}$

2. $\bigcup_{\kappa \in \mathrm{NEW}} C_{\mathrm{new}}(\kappa) = C_{\mathrm{old}}(L_{\mathrm{old}}(i)) \supseteq NEW$
3. $C_{\mathrm{new}}(\kappa) = C_{\mathrm{old}}(\kappa)$ *for* $\kappa \in V^c_{\mathrm{new}} \setminus NEW$.

Proof This is actually the reversed version of Prop. 6.1, where $\Lambda = \bigcup_{\kappa \in \mathrm{NEW}} C_{\mathrm{new}}(\kappa)$ and (hence) $\mathrm{NEW} = V^c_{\mathrm{new}} \cap \Lambda$. □

Property 7.4 *For* $k, m \in V$ *such that* $k \notin C_{\mathrm{old}}(L_{\mathrm{old}}(i))$ *and* $m \notin C_{\mathrm{old}}(L_{\mathrm{old}}(i))$, *the following equivalence holds.*

$$(k,m) \in E^{\mathrm{CR}}_{\mathrm{old}} \iff (k,m) \in E^{\mathrm{CR}}_{\mathrm{new}}.$$

Proof This follows by means of Prop. 7.3.3, 7.3.2 and Def. 4.17. □

By Lemma 3.6, by $i \in C_{\mathrm{new}}(L_{\mathrm{new}}(i))$ and by $j \in C_{\mathrm{new}}(L_{\mathrm{new}}(j))$ we have for $k, m \in V$:

$$k \xrightarrow[\mathrm{old}]{*} m \iff k \xrightarrow[\mathrm{new}]{*} m \vee (k \xrightarrow[\mathrm{new}]{*} L_{\mathrm{new}}(i) \wedge L_{\mathrm{new}}(j) \xrightarrow[\mathrm{new}]{*} m) \tag{9}$$

Hence it follows by (7) that we have for $\lambda \in V^c_{\mathrm{new}}$:

$$\lambda \xrightarrow[\mathrm{old}]{*} L_{\mathrm{new}}(i) \iff \lambda \xrightarrow[\mathrm{new}]{*} L_{\mathrm{new}}(i). \tag{10}$$

and similarly

$$L_{\mathrm{new}}(j) \xrightarrow[\mathrm{old}]{*} \lambda \iff L_{\mathrm{new}}(j) \xrightarrow[\mathrm{new}]{*} \lambda. \tag{11}$$

Since $i, L_{\mathrm{new}}(i), L_{\mathrm{new}}(j) \in C_{\mathrm{old}}(L_{\mathrm{old}}(i))$ we have $i \xrightarrow[\mathrm{old}]{*} L_{\mathrm{new}}(i) \xrightarrow[\mathrm{old}]{*} L_{\mathrm{new}}(j) \xrightarrow[\mathrm{old}]{*} i$.

By combining this with the definition of NEW we find

$$NEW = \{\lambda \in V^c_{\mathrm{new}} | L_{\mathrm{new}}(j) \xrightarrow[\mathrm{old}]{*} \lambda \xrightarrow[\mathrm{old}]{*} L_{\mathrm{new}}(i)\}.$$

and by (10) and (11) we can write

$$NEW = \{\lambda \in V^c_{\mathrm{new}} | L_{\mathrm{new}}(j) \xrightarrow[\mathrm{new}]{*} \lambda \xrightarrow[\mathrm{new}]{*} L_{\mathrm{new}}(i)\}. \tag{12}$$

Finally we state a property for M^- w.r.t. $\kappa, \mu \in V^c_{\mathrm{new}} \setminus$ NEW.

Property 7.5 *For* $\lambda, \mu \in V^c_{\mathrm{new}} \setminus NEW$ *the following holds*

$$M^-_{\mathrm{old}}(\kappa,\mu) = 1 \iff \sum_{\lambda \in V^c_{\mathrm{new}}} M^*_{\mathrm{old}}(\kappa,\lambda) M^c_{\mathrm{new}}(\lambda,\mu) = M^c_{\mathrm{new}}(\kappa,\mu) > 0.$$

Proof By Prop. 7.3 it follows that $M^c_{\mathrm{new}}(\kappa,\mu) = M^c_{\mathrm{old}}(\kappa,\mu)$ for $\kappa, \mu \in V^c_{\mathrm{new}} \setminus$ NEW. Therefore we have for $\kappa, \mu \in V^c_{\mathrm{new}} \setminus$ NEW (by using Prop. 7.3, Cor. 4.23 and $\mu \notin$ NEW)

$$\begin{aligned}
M^-_{\mathrm{old}}(\kappa,\mu) = 1 \iff & \sum_{\lambda \in V^c_{\mathrm{old}} \setminus \{L_{\mathrm{old}}(i)\}} M^*_{\mathrm{old}}(\kappa,\lambda) M^c_{\mathrm{old}}(\lambda,\mu) + M^*_{\mathrm{old}}(\kappa, L_{\mathrm{old}}(i)) M^c_{\mathrm{old}}(L_{\mathrm{old}}(i),\mu) \\
& = M^c_{\mathrm{old}}(\kappa,\mu) > 0 \\
\iff & \sum_{\lambda \in V^c_{\mathrm{new}} \setminus \mathrm{NEW}} M^*_{\mathrm{old}}(\kappa,\lambda) M^c_{\mathrm{new}}(\lambda,\mu) + \sum_{\lambda \in \mathrm{NEW}} M^*_{\mathrm{old}}(\kappa,\lambda) M^c_{\mathrm{new}}(\lambda,\mu) \\
& = M^c_{\mathrm{new}}(\kappa,\mu) > 0 \\
\iff & \sum_{\lambda \in V^c_{\mathrm{new}}} M^*_{\mathrm{old}}(\kappa,\lambda) M^c_{\mathrm{new}}(\lambda,\mu) = M^c_{\mathrm{new}}(\kappa,\mu) > 0.
\end{aligned}$$

Indeed, since it is easily seen that

$$M^c_{\text{old}}(L_{\text{old}}(i),\mu) = \sum_{\lambda \in \text{NEW}} M^c_{\text{new}}(\lambda,\mu)$$

and

$$M^*_{\text{old}}(\kappa, L_{\text{old}}(i)) = M^*_{\text{old}}(\kappa,\lambda) \quad (\lambda \in \text{NEW})$$

hold (cf. Prop. 7.3 and (8) respectively). □

We now show that TT holds at the end of line 13, by considering the execution of line 2 and 3 and the execution of procedure resetcomponents at line 13.

By line 2 and 3 it follows that at the beginning of line 13, the following holds

$$\begin{aligned} V^c = V^c_{\text{old}} \quad &\wedge \quad L = L_{old} \wedge C = C_{\text{old}} \wedge M^c = M^c_{\text{old}} \wedge e^c = e^c_{\text{old}} \\ &\wedge \quad M^- = M^-_{\text{old}} \wedge M^* = M^*_{\text{old}} \wedge M = M_{\text{new}} \\ &\quad \bigwedge_k \bigwedge_m [N(k,m) = \#\{(l,m) \in E_{\text{new}} | M^*(k,l) = 1\}]. \end{aligned} \tag{13}$$

Now consider the execution of procedure resetcomponents at line 13. (Procedure resetcomponents is given in Figure 20.) Then (13) holds at line 1 of this procedure. We prove that TT holds at line 22. By line 8-14 the following is established for line 15-22:

$$\begin{aligned} Li &= L_{\text{new}}(i) \neq L_{\text{new}}(j) = Lj \wedge \\ V^c &= V^c_{\text{new}} \wedge C = C_{\text{new}} \wedge L = L_{\text{new}} \wedge \\ \text{NEW} &= \{\lambda \in V^c_{\text{new}} | Lj \xrightarrow[\text{new}]{*} \lambda \xrightarrow[\text{new}]{*} Li\}. \end{aligned} \tag{14}$$

Indeed, since (7), (8), $M^* = M^*_{\text{old}}$ and (12) yield the required information. (Note that the set NEW used in procedure resetcomponents does correspond to the set NEW defined above indeed.) In line 2-4, all M^-–values concerning a node in $C_{\text{old}}(L_{\text{old}}(i))$ are set to zero. (Remark that the structure of G^-_{old} is used. Cf. Def. 4.19 and Def. 4.17. Therefore

$$\begin{aligned} M^-(l,m) &= 0 \quad \text{if } l \in C_{\text{old}}(L_{\text{old}}(i)) \vee m \in C_{\text{old}}(L_{\text{old}}(i)) \\ M^-(l,m) &= M^-_{\text{old}}(l,m) \quad \text{if } l \notin C_{\text{old}}(L_{\text{old}}(i)) \wedge m \notin C_{\text{old}}(L_{\text{old}}(i)) \end{aligned} \tag{15}$$

holds at line 5-17. By Prop. 7.3.2 we have $\bigcup_{k \in \text{NEW}} C_{\text{new}}(\kappa) = C_{\text{old}}(L_{\text{old}}(i))$. Therefore it is seen by (15), Prop. 7.4 and Def. 4.17 that the statements in line 18-19 establish

$$M^- \sim_{CR} \; G_{\text{new}} \tag{16}$$

and for $\kappa, \mu \in V^c_{\text{new}}$

$$\begin{aligned} M^-(\kappa,\mu) &= \quad 0 \quad \text{if } \kappa \in \text{NEW} \vee \mu \in \text{NEW} \\ M^-(\kappa,\mu) &= M^-_{\text{old}}(\kappa,\mu) \quad \text{otherwise} \end{aligned} \tag{17}$$

at line 20-22. Moreover, by line 5-7 and line 15-17

$$M^c = M^c_{\text{new}} \wedge e^c = e^c_{\text{new}} \tag{18}$$

is established at line 18-22.

Finally, it follows by (13) that

$$M^* = M^*_{\text{old}} \wedge M = M_{\text{new}} \wedge$$
$$\forall_k \forall_m [N(k,m) = \#\{(l,m) \in E_{\text{new}} | M^*(k,l) = 1\}] \qquad (19)$$

holds at line 22, since M^*, M and N are not effected by line 1-22.

By (14), (16), (18) and (19) it follows that T_1 holds at line 22. By (14) and line 20-21 we find that T_2 holds in line 22. Line 20-21, $M^* = M^*_{\text{old}}$ (cf. (19)), (10) and (7) yield T_3. Since by line 21 we have that Lj is the only black leader, (11) and $M^* = M^*_{\text{old}}$ yield T_4 for black leaders. Conditions T_3, (9), $M^* = M^*_{\text{old}}$ and line 20-21 therefore yield T_4 and T_5 at line 22. Furthermore, T_6, T_7 and T_8 hold trivially, whereas T_9 follows by (19).

Finally, note that $Lj \in$ NEW. Therefore we have for $\mu \in V^c_{\text{new}}$

$$\mu \notin NEW \wedge \mu \neq Lj \Longleftrightarrow \mu \notin NEW$$

and hence, (17), Prop. 7.5 and $M^* = M^*_{\text{old}}$ yield conditions T_{10} and T_{11}.

Conclusion: TT holds at line 22 of procedure resetcycles, called in line 13 of procedure delete.

Therefore we have shown that in this case, TT holds at the end of line 13 of procedure delete. □

Lemma 7.6 *At line 27, the following condition holds: $TT\wedge$ there are no grey leaders.*

Proof By Lemma 7.1 we have that TT holds at line 15. We prove, that TT is an invariant of the do-loop in line 16-26.

Suppose we have at line 16: "$TT\wedge$ there are grey leaders." It is seen that there are no marked leaders at line 16. (Since leaders are only marked in line 17 and line 19 and since it will be proved that there are no marked leaders at line 20. In fact, this could have been part of the invariant of the do-loop.) In line 17, some grey leader κ_0 is taken and Lj is marked, while NC is set to $\underline{-1}$ by procedure call neutral NC (cf. Figure 25). It follows, that $Q_1 \wedge TT(\kappa_0)$ holds at line 18, where $TT(\kappa_0)$ and Q_1 are given below. (Note that $TT(\kappa_0)$ equals TT except for $\kappa = \kappa_0$, i.e. except for $\kappa = \kappa_0$ w.r.t. T_3 up to and including T_{11}). Indeed, since $\kappa_0 \neq Lj$ follows from κ_0 being grey and Lj being black or purple (cf. T_2 and T_3). Hence, $Q_{1,1}$ follows by using T_3. Moreover, conditions $Q_{1,2}, Q_{1,3}, Q_{1,4}, Q_{1,6}$ and $Q_{1,7}$ can be obtained by means of T_7, T_5, T_{11}, T_8 and T_9 respectively.

In Lemma 7.8 it is stated that the following specification holds:

$$\{Q_1 \wedge TT(\kappa_0)\} \text{ adjust } (\kappa_0) \{Q_1 \wedge TT(\kappa_0) \wedge \text{there are no marked leaders}\}$$

Therefore

$$Q_1 \wedge TT(\kappa_0) \wedge \text{ there are no marked leaders} \qquad (20)$$

holds in line 20. In line 21-24, all white predecessor leaders of κ_0 are coloured grey and their $h-$values are adapted, if $M^*(\kappa_0, Lj) = 0$. In line 25, κ_0 is coloured black. Since (20)

Figure 28:

Q_1 is the conjunction of the following clauses.

$Q_{1,1}$ $\quad \kappa_0 \neq Lj \wedge \kappa_0 \xrightarrow[\text{new}]{*} Li \wedge \underset{\mu \in V^c_{\text{new}}}{\forall} [\mu \text{ marked} \Longrightarrow Lj \xrightarrow[\text{new}]{*} \mu \wedge \mu \neq \kappa_0]$
$\wedge \kappa_0$ is grey

$Q_{1,2}$ $\quad$ If $\kappa_0 \neq Li$, then $h(\kappa_0) \in S^c_{\text{new}}(\kappa_0)$ and $h(\kappa_0)$ is black

The following clauses hold for all leaders $\mu \in V^c_{\text{new}}$.

$Q_{1,3}$ $\quad M^*(\kappa_0, \mu) = 1 \Longleftrightarrow \kappa_0 \xrightarrow[\text{new}]{*} \mu \vee \underset{\text{marked}\lambda}{\exists} [\lambda \xrightarrow[\text{new}]{*} \mu]$

$Q_{1,4}$ $\quad M^-(\kappa_0, \mu) = 1 \Longleftrightarrow \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu) = M^c_{\text{new}}(\kappa_0, \mu) > 0 \wedge$
μ is not marked $\wedge \kappa_0, \mu \notin$ NEW

$Q_{1,5}$ $\quad NC(\mu) = -1 \vee NC(\mu) = \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu)$

$Q_{1,6}$ $\quad$ For all $k_0 \in C_{new}(\kappa_0)$ and $m \in C_{\text{new}}(\mu)$
$M^*(k_0, m) = M^*(\kappa_0, \mu)$

$Q_{1,7}$ $\quad$ For all $k_0 \in C_{\text{new}}(\kappa_0)$ and $m \in C_{\text{new}}(\mu)$
$N(k_0, m) = \#\{(l, m) \in E_{\text{new}} | M^*(k, l) = 1\}$

$TT(\kappa_0)$ is the conjunction of the following clauses:

- $T_1 \wedge T_2$ holds
- for all leaders $\kappa, \mu \in V^c_{\text{new}}$ with $\kappa \neq \kappa_0$ the clauses T_3 up to and including T_{11} hold.

---

holds in line 20 and because of line 21-25, it follows that TT holds in line 25, which can be seen in the following way. Firstly, the colour changes from white to grey in line 22 does not affect the validness of $TT(\kappa_0)$ for $\kappa \neq \kappa_0$. It is readily verified that $TT(\kappa_0)$ holds in line 25. Because κ_0 is coloured black in line 25, T_3, T_5, T_6, T_7 and T_{11} hold for $\kappa = \kappa_0$ too (note that $[\varphi \in P^c_{\text{new}}(\kappa_0) \Longrightarrow \varphi$ is not purple] holds, because $Q_{1,1}$ and $\varphi \in P^c_{\text{new}}(\kappa_0)$ imply $\varphi \xrightarrow[\text{new}]{*} Li$). Moreover, T_4, T_8, T_9 and T_{10} hold for $\kappa = \kappa_0$ because "$Q_1 \wedge$ there are no marked leaders" holds at line 20: these conditions are obtained by means of $Q_{1,3}, Q_{1,6}, Q_{1,7}$ and $Q_{1,4}$ respectively. Therefore, TT holds in line 25. This yields that TT is an invariant of the do-loop in line 16-26. Since each time line 17-25 is executed, a grey leader is coloured black, and since a black leader never changes its colour again, the do-loop terminates. Therefore we conclude that the condition " $TT \wedge$ there are no grey leaders" holds at line 27. □

Lemma 7.7 *Condition RR holds at line 30-31.*

Proof By Lemma 7.6 we have that

$$TT \wedge \text{there are no grey leaders} \tag{21}$$

holds at line 27. We first prove that (21) implies (22):

$$\begin{aligned} T_1 \quad & \wedge \quad M^* = M^*_{\text{new}} \wedge N = N_{\text{new}} \\ & \wedge \quad \bigwedge_{\kappa \in V^c_{\text{new}} \setminus \text{NEW}} \bigwedge_{\mu \in V^c_{\text{new}} \setminus \text{NEW}} [M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu)] \\ & \wedge \quad \bigwedge_{\kappa \in \text{NEW}} \bigwedge_{\mu \in V^c_{\text{new}}} [M^-(\kappa,\mu) = M^-(\mu,\kappa) = 0] \end{aligned} \tag{22}$$

Suppose (21) holds. By T_4 it follows that $M^*(\kappa,\mu) = M^*_{\text{new}}(\kappa,\mu)$ holds for $\kappa, \mu \in V^c_{\text{new}}$ with κ being black or purple. Now suppose κ is white. Then T_3 gives $\kappa \xrightarrow[\text{new}]{*} Li$. Consider a path from κ to Li in G^c_{new}, and let $\lambda \in V^c_{\text{new}}$ be the first black leader on that path. (Such a λ exists since Li is black (cf. T_2).) Then $\lambda \neq \kappa$, since κ is white. Since the predecessor leader of λ on that path is not black, it follows by T_6 that $M^*(\lambda, Lj) = 1$ and hence by T_4 that $\lambda \xrightarrow[\text{new}]{*} Lj$. Therefore we have $\kappa \xrightarrow[\text{new}]{*} Lj$. Combination with T_5 yields that $M^*(\kappa,\mu) = M^*_{\text{new}}(\kappa,\mu)$ for $\mu \in V^c_{\text{new}}$. Hence $M^*(\kappa,\mu) = M^*_{\text{new}}(\kappa,\mu)$ holds for all $\kappa, \mu \in V^c_{\text{new}}$. By T_8 this yields $M^* = M^*_{\text{new}}$. Moreover, we find by T_9 and Def. 2.1 that $N = N_{\text{new}}$ holds.

Now, let $\kappa, \mu \in V^c_{\text{new}}$ be such that $\kappa, \mu \notin$ NEW. By T_{10} and Cor. 4.23 it follows that $M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu)$ if κ is black or purple. Condition T_{11} and Cor. 4.23 yield that $M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu)$ holds if κ is white and $\mu \neq Lj$ too. Now suppose κ is white and $\mu = Lj$. By T_{11} we have $M^-(\kappa,\mu) = 0$. Like before there exists a black leader $\lambda \in V^c_{\text{new}}$ that satisfies $\kappa \xrightarrow[\text{new}]{*} \lambda \xrightarrow[\text{new}]{*} Lj$ and $\kappa \neq \lambda$. Since $Lj = \mu$ and $\mu \notin$ NEW we have $\neg(Lj \xrightarrow[\text{new}]{*} Li)$ and hence Lj is purple (cf. T_3). Therefore $\lambda \neq \mu$. Since $\kappa \neq \lambda \neq \mu, \kappa \xrightarrow[\text{new}]{*} \lambda \xrightarrow[\text{new}]{*} \mu$ and G^c_{new} is acyclic, there exists a leader $\lambda_0 \in V^c_{\text{new}}$ that satisfies $\kappa \xrightarrow[\text{new}]{*} \lambda_0 \wedge (\lambda_0, \mu) \in E^c_{\text{new}} \wedge \kappa \neq \lambda_0$. (Indeed, since the predecessor leader of μ on a path from κ to μ via λ must satisfy $\kappa \neq \lambda_0$ because of the acyclicness of G^c_{new}). By Cor. 4.21 this yields that $(\kappa,\mu) \notin (E^c_{\text{new}})^-$ and hence that $M^-_{\text{new}}(\kappa,\mu) = 0$. Since $M^-(\kappa,\mu) = 0$ holds, this yields $M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu)$. Conclusion: $M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu)$ for $\kappa, \mu \in V^c_{\text{new}} \setminus$ NEW.

Finally, T_{10} and T_{11} imply that $M^-(\kappa,\mu) = 0$ for $\kappa,\mu \in V^c_{\text{new}}$ with $\kappa \in$ NEW $\mu \in$ NEW. Hence, (22) holds at line 27.

By the conditional call of procedure adjusttransitivereduction in line 28 (where the procedure is given in Figure 21) and by Cor. 4.21 it follows that at line 30

$$M^-(\kappa,\mu) = M^-_{\text{new}}(\kappa,\mu) \tag{23}$$

holds for all $\kappa,\mu \in V^c_{\text{new}}$ satisfying $\kappa \in$ NEW $\vee \mu \in$ NEW. Since line 28-29 only effects $M^-(\kappa,\mu)-$ values for such κ and μ and since (22) holds in line 27, this yields that RR holds in line 30. □

Lemma 7.8 *The following specification holds.*

$$\{Q_1 \wedge TT(\kappa_0)\}\mathit{adjust}(\kappa_0)\ \{Q_1 \wedge TT(\kappa_0) \wedge \textit{there are no marked leaders}\}$$

Proof Consider procedure adjust(κ_0) with precondition $Q_1 \wedge TT(\kappa_0)$ (cf. Figure 17). We prove that $Q_1 \wedge TT(\kappa_0)$ is an invariant of the do-loop in line 3-26. Since it is readily seen that $TT(\kappa_0)$ holds at each line of the procedure, we only concern about Q_1. Suppose Q_1 holds in line 4 and ν_0 is a marked leader. Then dismarking ν_0 establishes Q'_1 at line 5, where Q'_1 is given below.

Q'_1 consists of the clauses $TT(\kappa_0), \kappa_0 \neq \nu_0, Lj \xrightarrow[\text{new}]{*} \nu_0$ and $Q'_{1,i}$ for $i = 1,\cdots,7$, where

$$\begin{array}{lcll} Q'_{1,i} & = & Q_{1,i} \text{ for } i \neq 3,4 & \\ Q'_{1,3} & : & M^*(\kappa_0,\mu) = 1 & \iff \kappa_0 \xrightarrow[\text{new}]{*} \mu \vee \exists_{\text{marked}\lambda}[\lambda \xrightarrow[\text{new}]{*} \mu] \vee \nu_0 \xrightarrow[\text{new}]{*} \mu \\ Q'_{1,4} & : & M^-(\kappa_0,\mu) = 1 & \iff \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0,\lambda) M^c_{\text{new}}(\lambda,\mu) = M^c_{\text{new}}(\kappa_0,\mu) > 0 \\ & & & \wedge\ \mu \text{ is not marked} \wedge \mu \neq \nu_0 \wedge \kappa_0,\mu \notin \text{NEW} \end{array}$$

for $\mu \in V^c_{\text{new}}$.
We show that $Q_1 \wedge TT(\kappa_0)$ holds in line 25.

We distinguish two cases for the if-statement in line 6-24.

(a) Suppose $\kappa_0 \neq L_{new}(i)$ and $M^*(h(\kappa_0),\nu_0) = 1$. Then $Q'_1 \Longrightarrow Q_1$ as will be shown below. We only have to prove that $Q'_1 \Longrightarrow Q_{1,3} \wedge Q_{1,4}$ for $\mu \in V^c_{\text{new}}$. Since $M^*(h(\kappa_0),\nu_0) = 1$, $h(\kappa_0)$ is a black leader and $h(\kappa_0) \neq \kappa_0$ (by $Q'_{1,2} = Q_{1,2}$), T_4 (w.r.t. h(κ_0)) yields $h(\kappa_0) \xrightarrow[\text{new}]{*} \nu_0$. By $h(\kappa_0) \in S^c_{\text{new}}(\kappa_0)$ this implies $\kappa_0 \xrightarrow[\text{new}]{*} \nu_0$. Therefore for all $\mu \in V^c_{\text{new}}$ we have $\nu_0 \xrightarrow[\text{new}]{*} \mu \Longrightarrow \kappa_0 \xrightarrow[\text{new}]{*} \mu$, and hence $Q'_{1,3} \Longrightarrow Q_{1,3}$.

By $Q'_{1,4}$ it follows that $Q_{1,4}$ holds for $\mu \in V^c_{\text{new}}$ if $\mu \neq \nu_0$ or $\mu \in$ NEW. Therefore we only need to prove $Q_{1,4}$ for the case that $\mu = \nu_0 \notin$ NEW holds. By $Q'_{1,4}$ we have $M^-(\kappa_0,\nu_0) = 0$. From the above consideration, we have $\kappa_0 \xrightarrow[\text{new}]{*} h(\kappa_0) \xrightarrow[\text{new}]{*} \nu_0$ with $\kappa_0 \neq h(\kappa_0)$, $h(\kappa_0) \in V^c_{\text{new}}$ and $h(\kappa_0)$ is black. We prove that $h(\kappa_0) \neq \nu_0$. Suppose $h(\kappa_0) = \nu_0$. Then ν_0 is black and hence (by T_3 and $\kappa_0 \neq h(\kappa_0) = \nu_0$) $\nu_0 \xrightarrow[\text{new}]{*} Li$. Moreover, by Q'_1 we find $Lj \xrightarrow[\text{new}]{*} \nu_0$. Hence $Lj \xrightarrow[\text{new}]{*} \nu_0 \xrightarrow[\text{new}]{*} Li$, which contradicts $\nu_0 \notin$ NEW. Hence $\kappa_0 \neq \nu_0$.
Therefore we have $\kappa_0 \xrightarrow[\text{new}]{*} h(\kappa_0) \xrightarrow[\text{new}]{*} \nu_0$, $h(\kappa_0) \in V^c_{\text{new}}$ and $\kappa_0 \neq h(\kappa_0) \neq \nu_0$. Since G^c_{new} is acyclic, this implies that there exists a $\lambda_0 \in V^c_{\text{new}}$ such that $\kappa_0 \xrightarrow[\text{new}]{*} \lambda_0 \wedge$

$(\lambda_0, \nu_0) \in E^c_{\text{new}} \wedge \lambda_0 \neq \kappa_0$. By $Q'_{1,3}$ we find $M^*(\kappa_0, \lambda_0) = 1$. Hence we have $\sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \nu_0) \geq M^*(\kappa_0, \lambda_0) M^c_{\text{new}}(\lambda_0, \nu_0) + M^*(\kappa_0, \kappa_0) M^c_{\text{new}}(\kappa_0, \nu_0) > M^c_{\text{new}}(\kappa_0, \nu_0)$. Since $M^-(\kappa_0, \nu_0) = 0$, it follows that $Q_{1,4}$ holds for this ν_0 too.

Conclusion: in this case, $Q_1 \wedge TT(\kappa_0)$ holds at line 6 and hence at line 25.

(b) Suppose $\kappa_0 = Li \vee M^*(h(\kappa_0), \nu_0) = 0$. In this case, line 7-23 is performed, starting by calling function value$NC(\kappa_0, \nu_0)$ (which is given in Figure 23).

Claim 7.9 *The function calls at line 8, 10, 11 and 13 yield*

$$valueNC(\kappa_0, \nu_0) = NC(\nu_0) = \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c(\lambda, \nu_0).$$

Proof

Recall that Q'_1 holds at the moment of calling value $NC(\kappa_0, \nu_0)$. From function value$NC(\kappa_0, \nu_0)$ and $Q'_{1,5} (= Q_{1,5})$ it follows that

$$valueNC(\kappa_0, \nu_0) = \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \nu_0)$$

$$\bigvee valueNC(\kappa_0, \nu_0) = \sum_{n \in C_{\text{new}}(\nu_0)} (N(\kappa_0, n) - e^c_{\text{new}}(n)).$$

We prove that the right-hand values are equal. By $Q'_{1,6} (= Q_{1,6})$ and $Q'_{1,7} (= Q_{1,7})$ we have

$$\begin{aligned}
& \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \nu_0) \qquad (24) \\
= & \sum_{\lambda \in V^c_{\text{new}}, \lambda \neq \nu_0} M^*(\kappa_0, \lambda) \cdot \#\{(l, n) \in E_{\text{new}} | l \in C_{\text{new}}(\lambda) \wedge n \in C_{new}(\nu_0)\} \\
= & \#\{(l, n) \in E_{\text{new}} | l \in V_{\text{new}} \backslash C_{\text{new}}(\nu_0) \wedge n \in C_{\text{new}}(\nu_0) \wedge M^*(\kappa_0, l) = 1\} \\
= & \sum_{n \in C_{\text{new}}(\nu_0)} (\#\{(l, n) \in E_{\text{new}} | M^*(\kappa_0, l) = 1 \wedge l \in V^c\} - \\
& \quad \#\{(l, n) \in E_{\text{new}} | M^*(\kappa_0, l) = 1 \wedge l \in C_{\text{new}}(\nu_0)\}) \\
= & \sum_{n \in C_{\text{new}}(\nu_0)} (N(\kappa_0, n) - \#\{(l, n) \in E_{\text{new}} | M^*(\kappa_0, l) = 1 \wedge l \in C_{\text{new}}(\nu_0)\}) \ .
\end{aligned}$$

By $Q'_{1,3}$ we have $M^*(\kappa_0, \nu_0) = 1$. Therefore $Q'_{1,6} (= Q_{1,6})$ gives $M^*(\kappa_0, l) = 1$ for $l \in C_{\text{new}}(\nu_0)$. Combination with Def. 4.12 gives for $n \in C_{\text{new}}(\nu_0)$.

$$\begin{aligned}
& e^c_{\text{new}}(n) = \#\{(l, n) \in E_{\text{new}} | L_{\text{new}}(l) = L_{\text{new}}(n)\} \\
= & \#\{(l, n) \in E_{\text{new}} | M^*(\kappa_0, l) = 1 \wedge l \in C_{\text{new}}(\nu_0)\}. \qquad (25)
\end{aligned}$$

Finally, by combining (24) and (25) we find

$$\sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \nu_0) = \sum_{n \in C_{\text{new}}(\nu_0)} (N(\kappa_0, n) - e^c_{\text{new}}(n)).$$

This concludes the proof. □

We distinguish two cases according to the guards of the if-statement in line 8-24.

Case (i)

$$\text{value}NC(\kappa_0,\nu_0) = \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0,\lambda) M^c_{\text{new}}(\lambda,\nu_0) > 0 \tag{26}$$

In this case, line 8-12 is executed and by Claim 7.9 $Q'_1 \wedge NC(\nu 0) > 0$ holds at line 9. We prove that Q_1 holds in line 12. Since line 10-11 can only effect $Q'_{1,4}$, we only have to show that $Q_{1,3}$ and $Q_{1,4}$ hold at line 12. First we show that $Q_{1,3}$ holds at line 9. Because of (26), there is a $\lambda_0 \in V^c_{\text{new}}$ with $M^*(\kappa_0,\lambda_0) = 1 \wedge (\lambda_0,\nu_0) \in E^c_{\text{new}}$. By $Q'_{1,3}$ we find $\kappa_0 \xrightarrow[\text{new}]{*} \lambda_0 \vee \exists_{\text{marked } \lambda}[\lambda \xrightarrow[\text{new}]{*} \lambda_0] \vee \nu_0 \xrightarrow[\text{new}]{*} \lambda_0$. Since $(\lambda_0,\nu_0) \in E^c_{\text{new}}$ and G^c_{new} is acyclic (and hence $\lambda_0 \neq \nu_0$), this implies

$$\kappa_0 \xrightarrow[\text{new}]{*} \nu_0 \vee \underset{\text{marked } \lambda}{\exists} [\lambda \xrightarrow[\text{new}]{*} \nu_0].$$

Therefore, $Q'_{1,3}$ yields for all $\mu \in V^c_{\text{new}}$:

$$M^*(\kappa_0,\mu) = 1 \Longleftrightarrow \kappa_0 \xrightarrow[\text{new}]{*} \mu \vee \underset{\text{marked } \lambda}{\exists} [\lambda \xrightarrow[\text{new}]{*} \mu].$$

Hence $Q_{1,3}$ holds at line 9. Since line 10-11 does not effect $Q_{1,3}$, it holds at line 12 too. Finally, because $Q'_{1,4}$ holds at line 9 and because of the if-statement in line 10-12 it easily follows that $Q_{1,4}$ holds at line 12.

Conclusion: Q_1 holds at line 12 and line 25 in this case.

Case (ii)

$$\text{value}NC(\kappa_0,\nu_0) = \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0,\lambda) M^c_{\text{new}}(\lambda,\nu_0) = 0 \tag{27}$$

In this case line 14-23 is executed and by Claim 7.9 it follows that $Q'_1 \wedge NC(\nu_0) = 0$ holds in line 14. It is easily seen that $Q_{1,2}(= Q'_{1,2})$ still holds in line 23. The only line that involves $Q'_{1,1}(= Q_{1,1})$ is line 21. Therefore it follows that $Q_{1,1}$ holds at line 23. The only statement that changes M^* is procedure disconnect in line 15 (cf. Fig. 22). It is seen that it preserves $Q'_{1,6}(= Q_{1,6})$. Therefore $Q_{1,6}$ holds at line 23. Because of the change of M^* in line 15, N is adjusted in line 16-20 while NC is adjusted in line 22 by procedure call decrease$NC(\nu_0,\mu_0)$ (cf. Figure 24). Since $Q_{1,5}(= Q'_{15})$ and$Q_{1,7}(= Q'_{1,7})$ hold at line 14, it herefore follows by line 15, line 16-20, and line 22 that $Q_{1,5}$ and $Q_{1,7}$ hold at line 23 too. Hence, condition $Q_{1,3}$ and $Q_{1,4}$ remain to be inspected. Consider $Q'_1 \wedge NC(\nu_0) = 0$ holding at line 14. By $Q'_{1,3}$, condition (27) yields for $\lambda_0 \in V^c_{\text{new}}$

$$\kappa_0 \xrightarrow[\text{new}]{*} \lambda_0 \vee \underset{\text{marked}\lambda}{\exists} [\lambda \xrightarrow[\text{new}]{*} \lambda_0] \vee \nu_0 \xrightarrow[\text{new}]{*} \lambda_0 \Longrightarrow (\lambda_0,\nu_0) \notin E^c_{\text{new}}.$$

Hence we have (by $\kappa_0 \neq \nu_0$ and ν_0 is not marked)

$$\neg(\kappa_0 \xrightarrow[\text{new}]{*} \nu_0) \wedge \neg \underset{\text{marked}\lambda}{\exists} [\lambda \xrightarrow[\text{new}]{*} \nu_0]. \tag{28}$$

Furthermore, we have for $\mu \in V^c_{\text{new}}$ with $\mu \neq \nu_0$:

$$\nu_0 \xrightarrow[\text{new}]{*} \mu \Longrightarrow \underset{\mu_0 \in S^c_{\text{new}}(\nu_0)}{\exists} [\mu_0 \xrightarrow[\text{new}]{*} \mu]. \tag{29}$$

Since (28) holds, it follows by the call of procedure disconnect(κ_0, ν_0) in line 15 (yielding $M^*(\kappa_0, \nu_0) = 0$) that $Q_{1,3}$ holds at line 16-20 for $\mu = \nu_0$. Because G^c_{new} is acyclic, we can write

$$\mu_0 \in S^c_{\text{new}}(\nu_0) \Longrightarrow \neg(\mu_0 \xrightarrow[\text{new}]{*} \nu_0).$$

Therefore $Q_{1,3}$ holds for $\mu = \nu_0$ at line 21-23 too. Since $Q'_{1,3}$ holds at line 14 and since line 15-20 does not effect $Q'_{1,3}$ for $\mu \neq \nu_0 (\mu \in V^c_{\text{new}})$ it follows by (29) and by line 21 that $Q_{1,3}$ holds at line 22-23 for $\mu \neq \nu_0$. Hence, $Q_{1,3}$ holds entirely at line 23.

Finally we show that $Q_{1,4}$ holds at line 23. Consider line 14 again. Since Q'_1 holds, we obtain by using $Q'_{1,4}, \kappa_0 \neq \nu_0, M^*(\kappa_0, \nu_0) = 1$ and $M^c_{\text{new}}(\kappa_0, \kappa_0) = 0$ respectively, that the following equivalences hold for $\mu \in V^c_{\text{new}}$ (at line 14):

$$\begin{aligned}
M^-(\kappa_0, \mu) = 1 \iff & \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu) = M^c_{\text{new}}(\kappa_0, \mu) > 0 \qquad (30)\\
& \wedge\, \mu \text{ is not marked } \wedge\, \mu \neq \nu_0 \wedge \kappa_0, \mu \notin \text{NEW}\\
\iff & \sum_{\lambda \in V^c_{\text{new}}, \lambda \neq \kappa_0, \lambda \neq \nu_0} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu) + M^*(\kappa_0, \nu_0) M^c_{\text{new}}(\nu_0, \mu)\\
& + M^c_{\text{new}}(\kappa_0, \mu) = M^c_{\text{new}}(\kappa_0, \mu) > 0\\
& \wedge\, \mu \text{ is not marked} \wedge\, \mu \neq \nu_0 \wedge \kappa_0, \mu \notin \text{NEW}\\
\iff & \sum_{\lambda \in V^c_{\text{new}}, \lambda \neq \kappa_0, \lambda \neq \nu_0} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu) + M^c_{\text{new}}(\kappa_0, \mu)\\
& = M^c_{\text{new}}(\kappa_0, \mu) > 0\\
& \wedge\, \mu \text{ is not marked} \wedge\, \mu \neq \nu_0 \wedge \mu \notin S^c_{\text{new}}(\nu_0) \wedge \kappa_0, \mu \notin \text{NEW}\\
\iff & \sum_{\lambda \in V^c_{\text{new}}, \lambda \neq \nu_0} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \mu) = M^c_{\text{new}}(\kappa_0, \mu) > 0\\
& \wedge\, \mu \text{ is not marked } \wedge\, \mu \neq \nu_0 \wedge \mu \notin S^c_{\text{new}}(\nu_0) \backslash \{\kappa_0\} \wedge \kappa_0, \mu \notin \text{NEW}
\end{aligned}$$

Since (30) holds at line 14, since $M^*(\kappa_0, \lambda)$ is not changed for $\lambda \neq \nu_0$ in line 15-22 and by means of line 21, we have that $Q_{1,4}$ holds at line 23 for $\mu \neq \nu_0$. Since $Q'_{1,4}$ and (27) hold at line 14, we have that

$$M^-(\kappa_0, \nu_0) = 0 \wedge \sum_{\lambda \in V^c_{\text{new}}} M^*(\kappa_0, \lambda) M^c_{\text{new}}(\lambda, \nu_0) = 0 \tag{31}$$

holds at line 14 too. Because $M^*(\kappa_0, \lambda)$ is not changed for $\lambda \neq \nu_0 (\lambda \in V^c_{\text{new}})$ in line 15-22 and because $M^c_{\text{new}}(\nu_0, \nu_0) = 0$, it follows that (31) still holds at line 23. Therefore, $Q_{1,4}$ holds at line 23 for $\mu = \nu_0$ too.

Conclusion: Q_1 holds at line 23 (and hence at line 25 too). Hence, in case b Q_1 holds at line 25.

By the above case-analysis we have that $Q_1 \wedge TT(\kappa_0)$ holds at line 25. Hence, $Q_1 \wedge TT(\kappa_0)$ is an invariant of the do-loop in line 3-26. Moreover, the loop-terminates. This is seen as follows. In line 21, unmarked leaders may be marked. However, in this case line 15 has been executed just before the marking process. Since in line 15 some $M^*(\kappa_0, \lambda_0)$- values are changed from 1 to 0 and since $M^*(\kappa_0, \lambda_0)$-values are never changed from 0 to 1, this marking process can be executed only finitely many times. Since each pass of the loop a marked leader is dismarked, this yields that the loop will terminate.

Conclusion: "$Q_1 \wedge TT(\kappa_0) \wedge$ there are no marked leaders " holds at line 27. Therefore we have proved that the given specification holds. □

7.3 Complexity

We consider the time complexity of a number q of consecutive edge deletions from a graph $G_{\text{old}} =< V, E_{\text{old}} >$, resulting in the graph $G_{\text{new}} =< V, E_{\text{new}} >$. Again we take $n = \#V, e_{\text{old}} = \#E_{\text{old}}$ and $e_{\text{new}} = \#E_{\text{new}}$ and we refer to the result graph after t deletions by $G_t =< V, E_t >$.

Like in subsection 6.3 we compute the cost of q deletions by considering the total net costs for q deletions for each of the procedures delete (including procedure colournodes), adjust, componentsbreak, resetcomponents (including procedure colournodes), adjusttransitivereduction, disconnect, valueNC and neutralNC. That is, if one of these procedures is called within an other procedure, the cost changed to the latter is $O(1)$ and the cost charged to the former is the amount "exceeding" $O(1)$.

7.3.1 Analysis of procedure delete

We split procedure delete into two parts.

(i) Line 1-15 and line 27-33. Obviously, each execution of procedure delete takes $O(n)$ time for this part. Therefore the total net costs for q deletions changed to this parts is $O(n \cdot q) = O(n \cdot e_{\text{old}})$.

(ii) Line 16-26 (the do-loop). We again distinguish wish two parts: part A and part B, where part A is all except the for-statement (line 16-20 and line 24-26) and part B contains the for-statement only (line 21-23).

(a) For each pass of the for-loop, the execution of part A takes $O(1)$ time. Moreover, during that pass a grey leader is "processed" and coloured black. Therefore we charge this $O(1)$ time to the cost of colouring a leader grey. This yields that for part (i) the cost is augmented by $O(1)$ time for each execution of procedure delete. (Obviously this does not affect the cost of part (i)). Moreover, each pass of the for-loop in line 21-23 is charged with an extra $O(1)$ time. This will be included in the cost of part B.

(b) Consider the t-th deletion of an edge: edge (i, j). First note that for any $\kappa_0 \in V_t^c$ the for-statement (part B) can be executed at most once during the execution of procedure delete(i, j), since it is only executed for $\kappa_0 \in V_t^c$ if κ_0 is grey, while

afterwards κ_0 is coloured black and never becomes grey again.

In the for-statement, all predecessor leaders φ of some $\kappa_0 \in V_t^c$ are processed in $O(1)$ time each. If φ is such a predecessor leader of κ_0, then there is an edge $(f,k) \in E_t$ such that $L_t(f) = \varphi$ and $L_t(k) = \kappa_0$. Therefore we charge the above $O(1)$ cost for processing φ as a predecessor leader of κ_0 to the edge (f,k). Hence, the cost of the execution of the for-statement for some $\kappa_0 \in V_t^c$ is

$$O(\#\{(f,k) \in E_t | f \in V \wedge k \in C_t(\kappa_0)\}). \tag{32}$$

On the other hand, by line 21 it is seen that the for-statement is executed for $\kappa_0 \in V_t^c$ only if $M^*(\kappa_0, Lj) = 0$. By $Q_{1,3}, Q_{1,6}$ and $Q_{1,1}$ we find for such a κ_0, that $M_t^*(\kappa_0, j) = 0 \wedge M_t^*(\kappa_0, i) = 1$. Since (i,j) is the edge that has just been deleted, this yields for such a κ_0 that $M_t^*(\kappa_0, j) = 0 \wedge M_{t-1}^*(\kappa_0, j) = 1$. Hence we have for such a κ_0 that

$$M^*{}_t(k,j) = 0 \wedge M^*{}_{t-1}(k,j) = 1 \text{ for } k \in C_t(\kappa_0) \tag{33}$$

Combining (32) and (33) yields that the total cost of the for-statement for this t-th deletion is

$$\begin{aligned} & O(\#\{(f,k) \in E_t | M^*{}_t(k,j) = 0 \wedge M^*{}_{t-1}(k,j) = 1\}) \\ = \; & O(\#\{(f,k,l) | (f,k) \in E_t \wedge l \in V \wedge M^*{}_t(k,l) = 0 \\ & \wedge M^*{}_{t-1}(k,l) = 1\}). \end{aligned} \tag{34}$$

Since $M_t(k,l) \leq M_{t-1}(k,l)$ and $M^*{}_t(k,l) \leq M^*{}_{t-1}(k-l)$ for $k,l \in V, t \geq 1$, and since $E_t \subseteq E_0$ for $t \geq 1$, (34) yields that the total cost of the for-statement for q deletions is

$$O(\#\{(f,k,l) | (f,k) \in E_0 \wedge l \in V \wedge M^*{}_q(k,l) = 0 \wedge M^*{}_0(k,l) = 1\}) =$$

$$O(e_{\text{old}} \cdot n).$$

Conclusion: by (i) and (ii) it follows that the total net costs for q deletions, charged to procedure delete is $O(n \cdot e_{\text{old}})$.

7.3.2 Analysis of procedure adjust

Consider procedure adjust. Except for the for-loops in line 16-22, a pass of the do-loop costs $O(1)$ time. At each pass of the do-loop, a marked leader is processed and dismarked. We therefore charge this $O(1)$ time to the cost of marking a leader. Therefore we only have to compute the cost of the three for-statements in line 16-22, where in line 21 each marking needs to be charged with an extra $O(1)$ time. Since for each $(\nu_0, \mu_0) \in E_{\text{new}}^c$ there exists an edge $(n_0, n') \in E_{\text{new}}$ with $n_0 \in C_{new}(\nu_0)$ and $n' \in C_{new}(\mu_0)$ (and hence $n' \in S_{new}(n_0)$) the costs of executing the last two for-loops (line 21 and 22) can be charged to the first for-loop by augmenting the cost of the statement $N(k,n') := N(k,n') - 1$ with $O(1)$ time. Therefore we have to compute the cost of the for-statement in line 16-20 only.

Consider the for-loop in line 16-20. At each step during the execution of this for-loop, the value of $N(k,n')$ is decreased for some $k,n' \in V$. Moreover, we have $N(k,n') \geq 0$ for $k,n' \in V$ and $\sum_{k \in V} \sum_{m \in V} N(k,m) \leq n \cdot e_{\text{old}}$. Since the value of $N(k,n')$ is never increased in any of the considered procedures, the above considerations yield that the cost of this for-loop is $O(n \cdot e_{\text{old}})$. Therefore we conclude that the total net costs for q deletions charged to procedure adjust is $O(n \cdot e_{\text{old}})$.

7.3.3 Analysis of procedure componentsbreak

The depth-first search in procedure componentsbreak can be performed in $\mathcal{O}(e_{\text{old}}^{\text{maxcyc}})$ time, where $e_{\text{old}}^{\text{maxcyc}} = max_{\kappa \in V_{old}^c} \#\{(k_1,k_2) \in E_{\text{old}} | k_1,k_2 \in C_{old}(\kappa)\}$ i.e., the maximal number of interior edges of any strongly connected component. To perform the depth first search in $O(e_{\text{old}}^{\text{maxcyc}})$ indeed, it is necessary to use the set of successors of a node that are within the same component (cf. Section 5).

Procedure componentsbreak is called within procedure delete(i,j) only for edges (i,j) that are interior to any component. Hence, it can be called at most $e_{\text{old}}^{\text{totcyc}}$ times, where $e_{\text{old}}^{\text{totcyc}} = \#\{(k_1,k_2) \in E_{\text{old}} | L_{old}(k_1) = L_{old}(k_2)\}$, i.e., the total number of edges that are interior to any strongly connected component.

Conclusion: the total net cost for q deletions charged to procedure componentsbreak is $O(e_{\text{old}}^{\text{totcyc}} \cdot e_{\text{old}}^{\text{maxcyc}})$.

7.3.4 Analysis of procedure resetcomponents

We consider the cost of an execution of procedure resetcomponents.

The statements in line 2-4, 18-19 and 20-21 can be performed in $O(n)$ time. (For, the ordered lists $C(\lambda)$ in line 19 are disjoint). The statements in line 5-7, 8-14, and 15-17 can be performed in $O(n + e_{\text{old}})$ time. Indeed, since in line 8-14 the strongly connected components are determined in $O(n+e_{\text{old}})$ time and afterwards V^c and L can be computed in $O(n)$ time. Then C can be adjusted by traversing array L once in $O(n)$ time. Hence, the cost of an execution of procedure resetcomponents costs $O(n + e_{\text{old}})$ time. We need the following property.

Property 7.10 *Let n_{old}^{cyc} be the number of nodes that are contained in any non-trivial component of G_{old} (i.e. a component with at least 2 nodes). Then procedure resetcomponents can be called at most $n_{old}^{cyc} - 1$ times and at most $2n_{old}^{cyc} - 1$ new components may arise during the q edge deletions.*

Proof Every time that procedure resetcomponents is called, the number of (possibly trivial) components increases. Since G_{old} contains at least $n - n_{old}^{cyc} + 1$ components (possibly trivial), procedure resetcomponents can be performed at most $n_{old}^{cyc} - 1$ times. Moreover, only the n_{old}^{cyc} nodes contained in any non-trivial component of G_{old} may be concerned in new arising components. These n_{old}^{cyc} nodes can give rise to at most n_{old}^{cyc} new components in G_{new} (compared to G_{old}). Since the components in G_{i+1} yield a finer partition of V than the components of G_i (for $0 \leq i < q$), this yields that we can encounter at most $2 \cdot n_{old}^{cyc} - 1$ newly arisen components in all graphs $G_i (0 \leq i \leq q)$. □

By Property 7.10 and by $n_{old}^{cyc} \leq min\{n, e_{\text{old}}\}$, the above considerations yield that the total net cost for q deletions charged to procedure resetcomponents is $O(n_{old}^{cyc}(n+e_{\text{old}})) = O(n \cdot e_{\text{old}})$.

7.3.5 Analysis of procedure adjusttransitivereduction

Consider the cost of an execution of procedure adjusttransitivereduction .(Note that NEW $\neq \emptyset$ holds, since this procedure is called at line 28 of procedure delete only.) It is easily seen that executing this procedure costs $O(n+e_{\text{old}})$ time for each $\kappa \in$ NEW. Since NEW contains the leaders of the newly arisen components only, we can write that the execution costs $O(n+e_{\text{old}})$ time per newly arisen component. By using Prop. 7.10 we find that the total net cost for q deletions charged to procedure adjusttransitivereduction is $O(n_{old}^{cyc} \cdot (n+e_{\text{old}})) = O(n \cdot e_{\text{old}})$.

7.3.6 Analysis of procedure disconnect

Procedure disconnect is only called within procedure adjust viz., at line 15. Since $Q'_{1,3}$ and $Q'_{1,6}(= Q_{1,6})$ hold at line 15, it follows that in procedure disconnect M^*-values are decreased from 1 to 0. Since M^*-values are never increased in any procedure, this yields that procedure disconnect costs $O(e^*_{\text{old}} - e^*_{\text{new}}) = O(n \cdot e_{\text{old}})$ time, where $e^*_{\text{old}} = \#E^*_{\text{old}}$ and $e^*_{\text{new}} = \#E^*_{\text{new}}$.

7.3.7 Analysis of procedure valueNC

Procedure valueNC is called within procedure adjust only (viz. line 8, 10, 11 and 13). Consider the t-th deletion of an edge, say (i,j). Then a call "valueNC(κ,ν)" costs $O(1)$ time if $NC(\kappa,\nu) \geq 0$ and it costs $O(\#C_t(\nu))$ time if $NC(\kappa,\nu) = -1$. By our definition of net cost, we charge the $O(1)$ cost to procedure adjust and not to procedure value NC. Therefore we may say that the net cost of one or more calls "value$NC(\kappa_0,\nu)$" for fixed $\kappa_0, \nu \in V_t^c$ during this t-th deletion, is $O(\#C_t(\nu))$ time. Since $C_t(\nu_1) \cap C_t(\nu_2) = \emptyset$ for $\nu_1 \neq \nu_2, \nu_1, \nu_2 \in V_t^c$, we therefore have: (++) the net cost of any number of calls of procedure value$NC(\kappa_0,\nu)$ for fixed $\kappa_0 \in V_t^c$ and for $\nu = \nu_1, \nu_2, \cdots \nu_p \in V_t^c$ with $\nu_r \neq \nu_s (1 \leq r < s \leq p)$ is $O(\sum_{1 \leq r \leq p} \#C_t(\nu_r))$.

We distinguish two cases (according to the second guard of the if-statement in line 6-24 of procedure adjust).

(a) $\kappa_0 = L_t(i)$. By (++) we have that the net cost for calling procedure value$NC(\kappa_0,\cdot)$ a number of times during this t-th deletion is $O(n)$ time.

(b) $\kappa_0 \neq L_t(i)$. In this case, the following claim holds.

Claim 7.11 *If in this case valueNC(κ_0,ν) is called for some $v \in V_t^c$, then there is an edge $(k,l) \in E_t$ such that $k \in C_t(\kappa_0), l \in C_t(h(\kappa_0))$ and*

$$\bigwedge_{m \in C_t(\nu)} [l \xrightarrow[t-1]{*} m \wedge \neg(l \xrightarrow[t]{*} m)].$$

Proof Note that value$NC(\kappa_0, \nu)$ is called in line 8, 10, 11 and 13 of procedure adjust and that Q_1' holds at the time of calling. Since $\kappa_0 \neq L_t(i), Q'_{1,2}(= Q_{1,2})$ yields that $h(\kappa_0) \in S_t^c(\kappa)$ and $h(\kappa_0)$ is a black leader. Since $\kappa_0 \neq L_t(i)$, line 7 of procedure adjust yields that $M^*(h(\kappa_0), \nu) = 0$. Hence we find by T_4 that $\neg(h(\kappa_0) \xrightarrow[t]{*} \nu)$. By T_3 we have $h(\kappa_0) \xrightarrow[t]{*} L_t(i)$ and hence $h(\kappa_0) \xrightarrow[t]{*} i$. By Q_1' we also have $L_t(j) \xrightarrow[t]{*} \nu$ and hence $j \xrightarrow[t]{*} \nu$. Since (i,j) is the t-th edge that is deleted, it follows by $h(\kappa_0) \xrightarrow[t]{*} i, j \xrightarrow[t]{*} \nu$ and Prop. 3.6 that $h(\kappa_0) \xrightarrow[t-1]{*} \nu$. Hence $h(\kappa_0) \xrightarrow[t-1]{*} \nu \wedge \neg(h(\kappa_0) \xrightarrow[t]{*} \nu)$ holds. Because $h(\kappa_0) \in S_t^c(\kappa_0)$, there is an edge $(k,l) \in E_t$ satisfying $k \in C_t(\kappa_0)$ and $l \in C_t(h(\kappa_0))$. This completes the proof. □

By (++) and by the above claim we can conclude: the net cost of any number of calls of procedure value$NC(\kappa_0, \nu)$ for fixed $\kappa_0 \in V_t^c$ and for a number of ν's with $\nu \in V_t^c$ is

$$O(\#\{(k,l,m) | l \xrightarrow[t-1]{*} m \wedge \neg(l \xrightarrow[t]{*} m)\}) \tag{35}$$

for some fixed $k \in C_t(\kappa_0), l \in C_t(h(\kappa_0))$ satisfying $(k,l) \in E_t$. Now consider all calls of value$NC(\kappa, \nu)$ during the t-th deletion, i.e. for a number of κ's with $\kappa \in V_t^c$ and a number of ν's with $\nu \in V_t^c$. Since $C_t(\kappa_1) \cap C_t(\kappa_2) = \emptyset$ if $\kappa_1 \neq \kappa_2, \kappa_1, \kappa_2 \in V_t^c$,(35) yields that the total net cost of any number of calls of procedure valueNC during the t-th deletion is bounded by

$$O(\#\{(k,l,m) | (k,l) \in E_t \wedge l \xrightarrow[t-1]{*} m \wedge \neg(l \xrightarrow[t]{*} m)\}).$$

We now compute the cost for all q deletions. By the above case analysis and by an argument similar to that in 7.3.1 we may write that the total net cost is bounded by

$$O(n \cdot e_{\text{old}}) + O(\#\{(k,l,m) | (k,l) \in E_0 \wedge l \xrightarrow[0]{*} m \wedge \neg(l \xrightarrow[q]{*} m)\}).$$

Hence, the total net cost for q deletions, charged to procedure valueNC is $O(n \cdot e_{\text{old}})$.

7.3.8 Analysis of procedure neutralNC

Consider the t-th deletion. Procedure neutralNC is only called within procedure delete, viz. at line 17 of this procedure. The first call of procedure costs $O(n)$ time. However, for each subsequent call only those $\kappa \in V$ for which $NC(\kappa) \neq -1$ holds are processed (cf. Section 5), viz. at cost of $O(1)$ time for each such κ. Therefore we consider this $O(1)$ time to be cost of altering a NC-value that is -1. (Obviously, this does not effect the cost analysis of the other procedures.) Hence the total costs charged to procedure neutral NC during this t-th deletion is $O(n)$.

Conclusion: the total net cost for q deletions, charged to procedure neutralNC is $O(n \cdot e_{\text{old}})$.

7.3.9 Complexity of q deletions

By combining the results of 7.3.1 up to and including 7.3.8 we have proved the following theorem.

Theorem 7.12 *For a number of q edge deletions, procedure delete takes $O(n \cdot e_{\text{old}} + e_{\text{old}}^{\text{totcyc}} \cdot e_{\text{old}}^{\text{maxcyc}})$ time, where e_{old} is the number of edges in G_{old}, $e_{\text{old}}^{\text{totcyc}}$ is the total number of edges in G_{old} that are interior to any component of G_{old}, $e_{\text{old}}^{\text{maxcyc}}$ is the maximal number of interior edges of any component of G_{old}.*

Moreover, it is easily seen that the following holds.

Lemma 7.13 *If the values of breakcomponents can be computed in $O(F(G_{\text{old}}))$ time for q edge deletions, then procedure delete takes $O(n \cdot e_{\text{old}} + F(G_{\text{old}}))$ time for q edge deletions.*

8 Conclusion and remarks

By the results in Sections 5, 6 and 7 the following theorem is obtained.

Theorem 8.1 *The transitive closure and the transitive reduction of a graph can be maintained under edge insertion and edge deletions, together with some auxiliary information. A number of q consecutive edge insertions takes $O(n \cdot e_{\text{new}})$ time, where e_{new} is the number of edges in the resulting graph, whereas a number of q consecutive edge deletions can be performed in $O(n \cdot e_{\text{old}} + e_{\text{old}}^{\text{totcyc}} e_{\text{old}}^{\text{maxcyc}})$ time, where e_{old} is the number of edges in the original graph, $e_{\text{old}}^{\text{totcyc}}$ is the number of edges interior to the original components and each original component contains at most $e_{\text{old}}^{\text{maxcyc}}$ interior edges. Moreover, all information that must be maintained takes $O(n^2)$ space and can be initialized in $O(n \cdot e_{\text{old}} + n^2)$ time for a graph with e_{old} edges.*

As stated in subsection 7.2, N^c–values are only computed (and maintained) when they are needed. However, if the matrix N^c must be available entirely, this can be achieved by means of some modifications of the procedures presented in Sections 6 and 7, yielding an extra cost term of $O(n \cdot e_{\text{old}}^*)$ in the time complexity of procedure delete. (Note that at each componentsbreak the matrix N^c must be recomputed completely w.r.t. the newly arisen components.) The arrays P^c and S^c are not really necessary for the algorithms, since P^c and S^c can be obtained by using P, S and C without increasing the total time complexity. Similarly e^c can be omitted as global information: by Prop. 4.13 it can be computed in procedure delete by means of N or by means or procedure resetcomponents (depending on whether a componentsbreak arises or not), without increasing the time complexity. Finally we remark that the algorithms presented above can be transformed into algorithms traversing on reduction edges only, yielding some better time bounds.

References

[1] A.V. Aho, M.R. Garey and J.D. Ullman, The transitive reduction of a directed graph, SIAM Journal of Computing, vol.1, no.2 (June 1972) 131-137.

[2] T. Ibaraki and N. Katoh, On-line computation of transitive closures of graphs, Information Processing Letters 16 (1983) 95-97.

[3] H. Rohnert, A dynamization of the all pairs least cost path problem, in: K. Mehlhorn (ed.), STACS 85, Lecture Notes in Computer Science, Vol 182, Springer Verlag, Heidelberg, 1985, pp. 279-286.

[4] R.E.Tarjan, Depth-first search and linear graph algorithms, SIAM Journal of Computing, vol.1, no.2 (June 1972) 146-160.