CPB Discussion Paper

No 29

March 2004

An efficient method for detecting redundant feedback vertices

Berend Hasselman

CPB Netherlands Bureau for Economic Policy Analysis
Van Stolkweg 14
P.O. Box 80510
2508 GM The Hague, the Netherlands

Telephone +31 70 338 33 80
Telefax +31 70 338 33 50
Internet www.cpb.nl

ISBN 90-5833-168-7

Abstract in English

A feedback vertex set of a directed graph cuts all cycles in a directed graph. Such a set can be obtained with the Levy–Low contraction algorithm, which generates a small feedback vertex set but not always a minimum size feedback set since it sometimes needs a heuristic contraction rule in order to reduce a graph completely. This may lead to members in a feedback vertex set being redundant in the sense that they do not cut a cycle of the original directed graph. In this paper we develop two algorithms for detecting such redundant feedback vertices efficiently. The algorithms are applied to analysing large nonlinear normalised systems of equations and to the feedback sets of a series of random directed graphs used by other authors. Computational experiments show that the algorithms developed in this paper significantly improve on the brute force algorithm.

Abstract in Dutch

De feedbackknoopverzameling van een gerichte graaf knipt alle gesloten paden in de gerichte graaf door. Zo'n verzameling kan worden verkregen met behulp van het Levy-Low contractie algoritme, dat een kleine feedbackverzameling kan genereren die echter niet altijd minimaal van omvang is omdat het algoritme soms een heuristische keuze moet maken om de graaf geheel te kunnen reduceren. Dit kan ertoe leiden dat elementen van de feedbackverzameling overbodig zijn in de zin dat ze geen gesloten pad van de oorspronkelijke gerichte graaf doorknippen. In dit stuk wordt een tweetal algoritmes ontwikkeld om dergelijke overbodige feedbackknopen efficiënt op te sporen. De algoritmes worden toegepast op de analyse van grote niet-lineaire genormaliseerde stelsels vergelijkingen en op de feedbackverzamelingen van een reeks gerichte grafen die gebruikt zijn door anderen. Experimenten tonen aan dat de algoritmes een significante verbetering zijn ten opzichte van het standaard rechttoe, rechtaan algoritme.

Contents

Summary

This paper presents an efficient method for determining the redundant members of a feedback vertex set of a directed graph. A feedback vertex set of a directed graph cuts all cycles in a directed graph. Such a set can be obtained with the Levy–Low contraction algorithm, which generates a small feedback vertex set but not always a minimum size feedback set. The Levy–Low algorithm needs a heuristic contraction rule in order to completely reduce a directed graph. Unfortunately the use of the heuristic contraction may lead to members of a feedback vertex set being redundant in the sense that they do not cut a cycle of the original directed graph. It is of interest to detect such redundant feedback vertices as efficiently as possible. The brute force method for detecting redundant feedback vertices removes all feedback vertices except one from the directed graph and proceeds to test the remaining graph for acyclicity. It then repeats this process for all feedback vertices in turn.

In this paper we develop two algorithms for detecting such redundant feedback vertices efficiently, by partitioning the original graph into a graph containing the feedback vertices only and an acyclic directed graph. A byproduct of the analysis is a simple algorithm for determining the cycles of each feedback vertex.

Two applications are provided of the algorithms. The first is decomposing and analysing large nonlinear normalised systems of econometric equations. The interdependencies between variables in the equations can be represented by a directed graph. The feedback sets of the equation systems contain very few redundant elements and on average the feedback chains appear to be quite small. This leads to a more efficient calculation of the jacobian of the feedback variables.

The second application consists of some computational experiments with the algorithms on a series of random directed graphs used by Pardalos et al. (1999) and Festa et al. (2001). The algorithms for detecting redundant feedback vertices developed in this paper turn out to be significantly more efficient than the brute force algorithm.

1 Introduction[1]

This paper presents a new efficient method for determining redundant elements in a feedback vertex set of a directed graph. The property of such a set is that if all vertices in a feedback vertex set are removed from the directed graph, all loops (cycles) of the graph will be broken, producing an acyclic graph. Usually, a feedback vertex set with the smallest possible size is desirable. Since finding such a set is a NP-hard problem, the second best option is to find a small feedback vertex set. Levy and Low (1988) have developed a simple algorithm for determining a cutset of a directed graph that seems to be widely used.[2] Since their algorithm uses a heuristic choice in order to proceed whenever the standard contraction rules can no longer be applied, it is possible that the feedback vertex set generated by the algorithm may contain vertices that can be removed from the feedback set without affecting the required property of a feedback set. Detecting such redundant feedback vertices and detecting feedback cycles is (notoriously) difficult according to Festa et al. (1999). This paper shows that it is possible to efficiently determine all redundant feedback vertices in a given feedback vertex set.

First, the Levy and Low (1988) algorithm is summarised and applied to a simple directed graph to illustrate the problem of redundant feedback vertices. Next, the basic algorithm for detecting redundant feedback vertices is presented. Then, several efficient algorithms for detecting a redundant feedback vertex are developed. The algorithms are illustrated by two applications. The first application is an extension of the standard ordering algorithm used for condensing sparse normalised systems of equations and is applied to some econometric models in use at the CPB, with interesting results. In the second application, the algorithms for detecting redundant feedback vertices described in this paper are compared to the basic algorithm incorporated in Algorithm 815 (see Pardalos et al. (1999) and Festa et al. (2001). It appears that the algorithms of this paper are significantly faster than the brute force algorithm and one of the algorithms may generate smaller feedback sets in some cases.

[1] I wish to thank A. ten Cate, O.L.A. van 't Veer and A.A. van der Giessen for their helpful comments.

[2] see for example Festa et al. (2001), Pardalos et al. (1999) and Becker and Rustem (1993).

2 Basic cutset algorithm

Let G be a directed graph with no parallel edges defined by the vertex (node) set V and the edge (arc) set E. E is a set of ordered pairs (u,v) with $u,v \in V$. The pair (u,v) represents an edge from u to v, also denoted as $u \to v$. A *path* from vertex v to vertex u is a sequence of edges in G of the form $v = v_1 \to v_2 \cdots \to v_k = u$. A *cycle* is a *nonempty path* from v to itself. A *self-loop* is an edge $v \to v$; of course a *self-loop* is a *cycle*.

The problem is to find a set of vertices that cuts all cycles in the graph. In other words: to find a set of vertices, which after removal from the graph make the resulting graph acyclic. Such a set is called a *cutset* or a *feedback vertex set*. Often it is desirable to find a minimum size cutset. For a general graph this is an NP-complete problem (see Levy and Low (1988) for references).[3]

Levy and Low (1988) have designed an algorithm that finds small cutsets efficiently. The algorithm consists of the repeated application of certain contraction operations which preserve properties of minimum cutsets. Vertices that possess certain properties are added to the cutset. The algorithm stops when the contracted graph is empty. The contraction operations are:

IN0(v)	if vertex v has no incoming edge then remove v from the graph G and all edges leaving v;
OUT0(v)	if vertex v has no outgoing edge then remove v from the graph G and all edges entering v;
IN1(v)	if vertex v has one incoming edge, transfer all outgoing edges from v to its predecessor u ($u \neq v$) and remove v from the graph;
OUT1(v)	if vertex v has one outgoing edge, transfer all incoming edges from v to its successor u ($u \neq v$) and remove v from the graph;
LOOP(v)	if there is an edge $v \to v$ then delete v from the graph and add v to the cutset.

Levy and Low (1988) show that repeated application of these contractions until no further contraction is possible, results in a unique graph, apart from edge and vertex names. Therefore the order in which the contractions are applied does not affect the end result.

There exist directed graphs which cannot be reduced to an empty graph using the above contractions only. Therefore, an additional contraction operation is required:

REMOVE(v)	select a vertex v from the graph either randomly or heuristically, remove v from the graph and add v to the cutset. A common rule is to select the vertex with the largest indegree $\times$ outdegree.[4]

[3] A comprehensive discussion of feedback set problems is given by Festa et al. (1999). Additional reduction rules are possible, which are however quite involved (see for example Orenstein et al. (1995) and Lin and Jou (2000).). An exact algorithm for determining the minimum feedback vertex set has been developed by Smith and Walford (1975); the algorithm may require exponential time.

[4] see for example Pardalos et al. (1999), Festa et al. (2001), Becker and Rustem (1993) and Don and Gallo (1987).

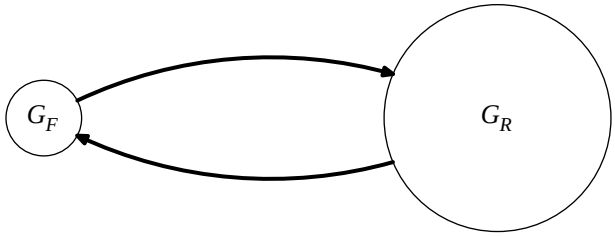
Repeated application of the above contraction operations until the contracted graph is empty results in a cutset S. Decompose this set into a set of feedback vertices chosen by the $\mathsf{LOOP}(v)$ rule, S_d, and a set chosen by the heuristic $\mathsf{REMOVE}(v)$ rule, S_h. Let $S_{\min}$ be a minimum size cutset of the graph and let $|S|$ denote the cardinality of set S. Levy and Low show that the following properties hold

1. $S_d \cup S_h$ is a cutset of the directed graph
2. $S_d \cap S_h = \emptyset$
3. $|S_d| \leq |S_{\min}| \leq |S_d| + |S_h|$

Thus if the heuristic step is never taken the feedback set is a minimum size feedback vertex set (see Levy and Low (1988), page 479).

The intended effect of this algorithm is to transform the original graph into a directed graph G_F consisting of the feedback vertices and a directed acyclic graph G_R consisting of the remaining vertices. Figure 2.1 depicts this graphically. The thick arrows from G_F to G_R and

Figure 2.1 Ordered graph



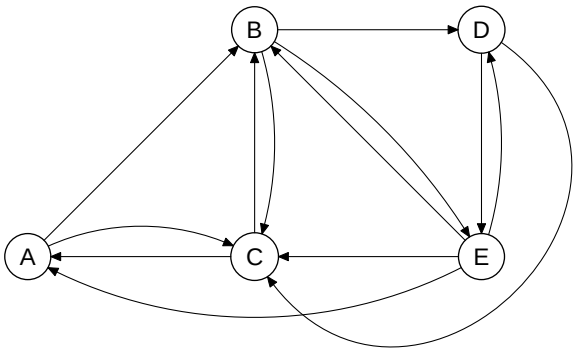
back represent all directed edges from the feedback vertices in G_F to the non feedback vertices in G_R and vice versa. This is the intended result of the contraction algorithm as described above. The heuristic choice of feedback vertices, however, throws a spanner in the works. However, one or more of the selected feedback vertices may be redundant in the sense that on removal from the cutset, the remaining cutset still breaks all cycles of the directed graph. This redundancy is the subject of the next sections.

3 An example of redundant feedback vertices

The removal of a feedback vertex of a directed graph should break a cycle of the graph. If the heuristic step was needed in the generation of the set, there is no guarantee that the vertex so chosen actually breaks a cycle. This section first illustrates this problem with an example graph and then develops a method for detecting redundant elements in a cutset.

An example directed graph is given in figure 3.1; it was taken from figure 2 of Orenstein et al. (1995). Table 3.1 gives the in- and outdegrees of the vertices of the initial state and after removal of vertex B. The graph cannot be contracted by any of the basic contraction operations

Figure 3.1 Example of a directed graph



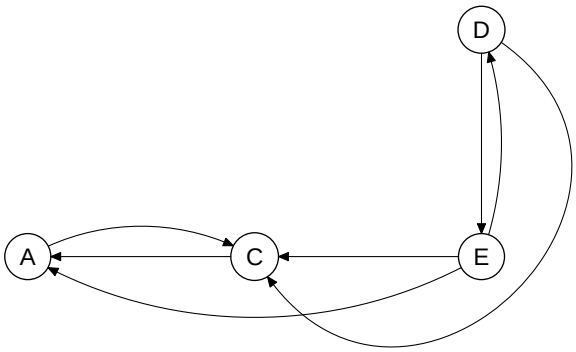
described in the previous section. There are no vertices with no or one incoming or outgoing edges. In addition there are no self loops. Therefore a feedback vertex must be chosen by the heuristic step. Vertex B will be selected since it has the largest product of indegree and outdegree (see the column labelled $|in(v)| \times |out(v)|$ in table 3.1). Thus the contraction REMOVE(B) is

Table 3.1 In- and out degrees of the example graph

	Initial state			After removal of B	
v	$\|in(v)\|$	$\|out(v)\|$	$\|in(v)\| \times \|out(v)\|$	$\|in(v)\|$	$\|out(v)\|$
A	2	2	4	2	1
B	3	3	9		
C	4	2	8	3	1
D	2	2	4	1	2
E	2	4	8	1	3

executed. The result is shown in figure 3.2. Two vertices have indegree 1 and two different vertices have outdegree 1. The graph has been changed in such a way that the basic contraction rules can be applied. Depending on the order in which the contraction operations are performed, the following four distinct cutsets are generated, namely $\{B,C,E\}$, $\{B,C,D\}$, $\{B,A,E\}$ and $\{B,A,D\}$. The additional feedback vertices are all chosen by the LOOP contraction. Since the

Figure 3.2 Example graph after removal of B

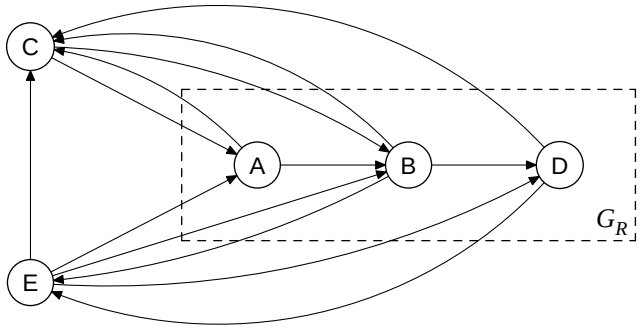


heuristic choice was made only once we know that the size of the minimal feedback vertex set must be at least 2 and will not be larger than 3.

Only the first cutset contains a redundant element B. This can be seen by removing the vertices C and E from the graph. The result is a graph consisting of A, B and D with only one path $A \rightarrow B \rightarrow D$. In the second cutset B is not redundant. If C and D are removed from the graph, the graph still has a cycle containing B and E. Similar reasoning applies to the third and fourth cutset.

The example graph has a minimum size cutset $S = \{C, E\}$. A redrawing of the graph in figure 3.3 illustrates the partitioning of the graph in G_F and G_R. The G_F part is formed by the vertices C and E and the edge (E, C). The G_R partition is formed by the vertices A, B and D with edges (A, B) and (B, D). All remaining edges provide the connection between the two partitions.

Figure 3.3 Example graph with cutset $\{C, E\}$



4 Algorithms for detecting redundant feedback vertices

4.1 The basic algorithm

A feedback vertex v in a feedback vertex set is redundant if the graph obtained by removing all vertices in the feedback vertex set from the original graph except v is acyclic (see Pardalos et al. (1999, page 404)). Pardalos et al. (1999) use this definition to derive an algorithm for detecting redundant feedback vertices. This algorithm is given in figure 4.1. It is a straightforward

Figure 4.1 Basic algorithm for removing redundant feedback vertices

```
flag = 1
while flag == 1 do
    flag = 0
    for i = 1,...,|S| do
        delete all elements from S \ {s_i} from the graph
        if graph == acyclic then
            S = S \ {s_i}
            flag = 1
            break
        endif
    endfor
endwhile
```

application of the definition of a redundant feedback vertex. To test if a feedback vertex is redundant, all feedback vertices but one are deleted from the graph. If the resulting graph is acyclic then the feedback vertex being examined is redundant. Whenever a redundant feedback vertex is detected it is removed from the feedback set and the algorithm is restarted. This process continues until no more redundant feedback vertices are detected. Depending on the implementation of the graph contraction rules and how the test for acyclicity of a graph is actually implemented, this basic algorithm may require a lot of work and thus be slow, since it requires as many passes through the complete graph as there are redundant feedback vertices.

The basic algorithm can be modified just slightly to obtain a more efficient algorithm; see figure 4.2. Feedback vertices that already have been identified as not redundant when a redundant feedback vertex is encountered, do not need to be checked again. This can be seen as follows. Assume s_p to be the first redundant feedback vertex. If a feedback vertex s_k with $k < p$ is not redundant with vertex s_p excluded from the graph, the cycle containing s_k cannot pass through s_p (s_p does not belong to the cycle). Therefore the removal of s_p from the feedback set cannot change the redundancy state of previously investigated feedback vertices (those with $k < p$) and there is no need for a complete restart of the algorithm. It can continue with testing the feedback vertices following s_p. Even more efficiency may be achieved by starting the test for redundancy at the first feedback vertex not chosen by the LOOP contraction rule. The modified algorithm needs only one pass through the graph.

Figure 4.2 Modified basic algorithm for removing redundant feedback vertices

```
i_b = index first heuristically selected feedback vertex
if i_b == 0 then flag = 0 else flag = 1 endif
while flag == 1 do
    flag = 0
    for i = i_b, ..., |S| do
        delete all elements from S \ {s_i} from the graph
        if graph == acyclic then
            S = S \ {s_i}
            flag = 1
            i_b = i
            break
        endif
    endfor
endwhile
```

4.2 Alternative algorithms

The definition of a redundant feedback variable implies a special property of the partitioning. Let $D = \{d_1, d_2, \cdots, d_k\}$ be a subset of vertex indices in the acyclic part of the partition G_R. The index d_1 gives the first vertex with an incoming edge from the feedback vertex; $d_2 \ldots d_k$ give the remaining indices defining the path to follow in G_R. Each feedback vertex may have several sets D associated with it.

Lemma 1. *A feedback vertex v with no self-loop is redundant if no set $D = \{d_1, d_2, \cdots, d_k\}$ exists such that $v \to v_{d_1} \to v_{d_2} \to \cdots \to v_{d_k} \to v$, where $v_{d_i}, \ldots, v_{d_k}$ are vertices of the directed acyclic graph G_R.*

Proof. If all feedback vertices except v are removed from the graph and v is *not* redundant then there must be a cycle in the left over graph containing v. So for v to be not redundant there must be an edge from a vertex in the acyclic graph partition G_R directly to the feedback vertex v. If v is redundant no such edge exists and therefore no set D exists. Q.E.D.

It should be noted that if a feedback vertex is part of a cycle that only contains other feedback vertices, it will be identified as redundant by the condition of the lemma. If all other feedback vertices are excluded from the graph, the remaining feedback vertex will be an isolated vertex and therefore it is redundant.

The condition specified in Lemma 1 leads to a simple algorithm for determining whether a feedback vertex is redundant or not. First we need to determine the order in which vertices appear in the acyclic part of the graph partition G_R. This is done by removing all feedback vertices from the original graph and then repeatedly applying the IN0 contraction and recording the indices of the vertices as they are being removed. A formal description of this algorithm is

Figure 4.3 The Ordering algorithm

```
// Assume cutset algorithm has been executed
for s ∈ S do
    V = V \ {s}
    E = E \ {(u,s) ∈ E  or  (s,u) ∈ E}
endfor
initialise ordering vector d to empty
repeat
    for v ∈ V do
        if |in(v)| == 0 then
            d = d ⊕ v    // The operator ⊕ appends an element to a vector.
            V = V \ {v}
            E = E \ {(v,u) ∈ E}
        endif
    endfor
until V == ∅
```

given in figure 4.3.

The algorithm for detecting a redundant feedback vertex uses the ordering information and the information about incoming edges and is given in figure 4.4. The vector r is used to record if a vertex has been visited; the expression $r[u]$ is used to access the entry for vertex u in this vector. In order to find out if there is a cycle for an element of the cutset given the acyclic ordering, the algorithm first sets the visited flag for the current cutset element to 1. It then

Figure 4.4 Algorithm for detecting a redundant feedback vertex

```
let s ∈ S
r = 0
r[s] = 1
for j = 1,...,|V| − |S| do
    for u ∈ in(v_{d_j}) do
        if r[u] == 1 then  // if u has been visited: mark v_{d_j} visited
            r[v_{d_j}] = 1
            break
        endif
    endfor
endfor
for u ∈ in(s) do
    if r[u] == 1 then
        r[s] = 2
        break
    endif
endfor
if r[s] == 1 then
    s is redundant
endif
```

traverses the acyclic part of the graph in the order defined by d and whenever an incoming edge comes from a vertex already visited the current vertex is marked visited. After this process has been completed, it tests if incoming edges to the feedback vertex come from visited vertices. If that is the case then the visited flag for the current cutset vertex is set to 2 so that a derived visit can be distinguished from the initial visit. Finally, if the visited flag for the feedback vertex equals 1, then it has not been visited and therefore, it is redundant, since there is a no cycle with the feedback vertex as its start and end. This procedure will also treat feedback vertices with a self-loop correctly.

4.3 Removing redundant feedback vertices

There are also two ways of using the algorithms for detecting a redundant feedback vertex in an algorithm for removing all redundant feedback vertices from a cutset of a graph.

The first one, labelled as Algorithm A and given in figure 4.5, is very similar to the modified basic algorithm given in the previous section. After detecting a redundant feedback vertex it is

Figure 4.5 Algorithm A for removing redundant feedback vertices

```
i_b = index first heuristically selected feedback vertex
if i_b == 0 then flag = 0 else flag = 1 endif
while flag == 1 do
    flag = 0
    Execute ordering algorithm
    for i = i_b,...,|S| do
        if s_i is redundant then
            S = S \ {s_i}
            flag = 1
            i_b = i
            break
        endif
    endfor
endwhile
```

immediately removed from the feedback vertex set. Then the ordering algorithm is executed again with the new feedback vertex set and the feedback vertex set is again checked for redundant feedback vertices. This process is repeated until no more redundant feedback vertices are detected.

The second algorithm, Algorithm B and given in figure 4.6, first detects all redundant feedback vertices without removing them from the current feedback vertex set. Then all redundant elements are removed from the feedback vertex set. It is possible that too many elements were removed from the feedback vertex set so all elements from the modified feedback vertex set are removed from the (original) graph and the cutset algorithm is executed again on this (reduced) graph. The process of ordering the graph and detecting redundant feedback

vertices is repeated until no more redundant feedback vertices are detected. In practice at most two passes appear to be required.

Figure 4.6 Algorithm B for removing redundant feedback vertices

```
i_b = index first heuristically selected feedback vertex
if i_b == 0 then flag = 0 else flag = 1 endif
while flag == 1 do
    H = ∅
    Execute ordering algorithm
    for i = i_b, ..., |S| do
        if s_i is redundant then
            H = H ∪ {s_i}
        endif
    endfor
    if H ≠ ∅ then
        S = S \ H
        remove all elements S from graph
        rerun cutset algorithm
        i_b = index first heuristically selected feedback vertex
        if i_b == 0 then flag = 0 endif
    else
        flag = 0
    endif
endwhile
```

4.4 Feedback chains

The way the ordering of the acyclic part of the graph is used for detecting redundant feedback vertices suggests a method for determining the set of vertices that are in all the cycles of each feedback vertex. We call such a set a feedback chain. The algorithm for determining a feedback chain is given in figure 4.7. For each feedback vertex the algorithm gives a vector of vertex indices identifying the cycle(s) starting and ending with the feedback vertex and *not* including other feedback vertices. An application for this algorithm will be given in the next section.

Figure 4.7 Algorithm for determining feedback chains

```
for i = 1,...,|S| do
    r = 0
    initialise c_i to empty
    r[s_i] = 1
    for j = 1,...,|V| − |S| do
        for u ∈ in(v_{d_j}) do
            if r[u] ≥ 1 then
                r[v_{d_j}] = 1
                c_i = c_i ⊕ d_j
                break
            endif
        endfor
    endfor
endfor
```

5 Applications

5.1 Preliminaries

The algorithms of the preceding section have been implemented in a computer program written in the Fortran 95 language. This section presents two applications of the algorithms:

1. ordering a normalised system of nonlinear equations. We analyse several econometric models in use at the CPB;
2. determining the feedback set and locating redundant feedback vertices of a series of random directed graphs used by Pardalos et al. (1999) and Festa et al. (2001). The results are compared to those obtained with Algorithm 815 (Festa et al. (2001)).

The Fortran 95 implementation represents a directed graph with an adjacency matrix: an edge (v_i, v_j) is a non zero element a_{ji} in the matrix $A = (a_{ij})$. The directed graphs used in the two applications have very sparse adjacency matrices, so we use a sparse coordinate storage format, where only the row and column index are recorded, in the implementation. The contraction operations are easily translated into operations on rows and columns of the adjacency matrix.

The implementation of the REMOVE(v) contraction has been borrowed from Pardalos et al. (1999) and Festa et al. (2001). Whenever a heuristic choice is required, a list of candidate vertices for removal R is drawn up as follows.

$$R = \{v \in V \,|\, g(v) \geq \alpha G_h + (1-\alpha) G_l \,,\, 0 \leq \alpha \leq 1\}$$

where

$$\begin{aligned} g(v) &= |in(v)| \times |out(v)| \\ G_h &= \max_{v \in V} g(v) \\ G_l &= \min_{v \in V} g(v) \end{aligned}$$

The scalar parameter α is set in advance either as a fixed number or chosen at random by the program. Two methods for selecting a vertex from the list R have been implemented in the program: either the *first* vertex in R or a randomly selected vertex from R.

5.2 Econometric models

Assume a system of equations specified in normalised form

$$x = F(x) \tag{5.1}$$

where $x \in \mathbb{R}^n$ and $F : \mathbb{R}^n \to \mathbb{R}^n$.[5] The elements of the vector x are the endogenous variables of the model. There is a one-to-one correspondence between variables and equations in the sense

[5] $F(x) = \big(f_1(x), \ldots, f_i(x), \ldots, f_n(x)\big)$.

that variable x_i can be said to be determined by equation *i* and function $f_i(x)$. Variable x_i is said to depend directly on variable x_j if x_j occurs in the function f_i.[6]

The dependencies in the system of equations can be represented by a directed graph *G*. Each variable x_i becomes a vertex and the dependency of x_i on x_j is represented by an edge from x_j to x_i.[7] If x_i depends directly on x_j in more than one way there will be only one edge and therefore there are no parallel edges. The directed graph of a simultaneous system of equations will contain at least one cycle.

We assume that the system of equations is sparse, in the sense that only a small number of variables enter into each equation. We want to compress the system of equations into a (much) smaller system by (virtual) substitution of variables. This can be translated into the problem of finding a subset of vertices of the graph representation such that on removal of these vertices the resulting graph no longer contains directed cycles. For our purpose a small cutset is preferable since this implies a small compressed system of equations.

Detailed descriptions and analyses of the algorithm using directed graphs to determine a feedback set for equation systems can be found in Becker and Rustem (1993) and using an incidence matrix in Don and Gallo (1987).

The basic idea of the ordering procedure is to split the system of equations into three parts as follows

1. *prologue* equations, consisting of all equations which depend only on non-endogenous variables and/or on endogenous prologue variables. The process of determining prologue equations can be applied repeatedly until no more prologue equations can be found. This is equivalent to finding a recursive ordering of a subset of equations.
2. *epilogue* equations, consisting of all equations whose left hand side variable does not occur in other equations. The process of determining epilogue equations can be applied repeatedly until no more epilogue equations can be found.
3. *simultaneous* or *heart* equations, which are interdependent: they are not part of the prologue or epilogue group of equations.

This splitting procedure is a straightforward application of the graph contraction operation described in section 2. The prologue set of equations is determined by repeated application of the IN0 contraction and the epilogue by repeated application of the OUT0 contraction. The remaining graph represents the simultaneous part of the model. A feedback set is determined in the standard way and then the non feedback equations are topologically sorted.[8] Finally, the algorithms for detecting redundant feedback vertices and generating the feedback chains can be

[6] A variable may depend on itself.

[7] If a variable depends on itself there will be a self-loop.

[8] The process is also known as the ordering algorithm (see figure 4.3).

applied to the result. Application of the feedback chain algorithm will give, for each feedback variable, a vector of indices identifying the order in which equations must be visited for calculating the appropriate column of the feedback jacobian (the matrix with elements $J_{ij} = \partial x_i / \partial x_j$). For sparse systems of equations one may expect the length of the feedback chains to be much less than the number of equations in the simultaneous block of the complete system.

This procedure has been applied to the following econometric models:

- SAFE, a quarterly model for short term analysis described in CPB (2002);
- a crudely normalised version of MULTIMOD Mark III, a world model built at the IMF (Laxton et al. (1998));
- two versions of a sectoral model for the Dutch economy: the very large model athena(1) and a smaller version of several years ago: athena(0). A description of the latter version is given in Vromans (1998). Regrettably there is no description of the larger version;
- a large general equilibrium model for the world economy: the version used for analysis, World Scan, and the version for calibrating the model to data, World Scan(inv). The model has been described in Timmer (1998) and CPB (1999);
- a simplified labour market model (Mimkl1) extracted from MIMIC, a general equilibrium model for the Netherlands (see Graafland and de Mooij (1998)).

Algorithm B for removing the redundant feedback vertices was used (see figure 4.6). For the heuristic choice of a feedback vertex the first vertex in the list R is used.

Results for the ordering of the models are given in table 5.1 and some statistics for the feedback chains are given in table 5.2. From table 5.1 we see that the feedback sets of these

Table 5.1 Model ordering results

Model	Neq[a]	Sblock[b]	Nfb[c]	Red fb[d]	Heuristic[e]
SAFE	2047	773	28	0	6
Multimod	665	499	109	2	14
Athena(1)	83101	25786	424	0	1
Athena(0)	7662	5924	94	7	44
WorldScan	18552	10084	410	0	0
WorldScan(inv)	29824	15195	145	0	0
Mimkl1	67	60	15	1	1

[a] number of equations
[b] number of simultaneous equations
[c] number of feedback variables (after removing redundant vertices)
[d] number of redundant feedback variables detected after first ordering pass
[e] number of heuristically chosen feedback variables

models do not contain many redundant feedback vertices. Also, the feedback sets of the larger

models contain very little to no heuristically chosen vertices. For these models the feedback sets are (almost) minimum size feedback sets. From table 5.2 it may be concluded that in all models

Table 5.2 Model feedback chains

Model	Nfb[a]	Fbcmin[b]	Fbcmax[c]	Fbcavg[d]	Avgpass[e]
SAFE	28	0	575	264.6	11
Multimod	109	3	274	67.9	39
Athena(1)	424	2	16717	3034.6	57
Athena(0)	94	60	3225	1459.6	25
WorldScan	410	1	7155	2279.8	110
WorldScan(inv)	145	20	2845	850.8	10
Mimkl1	15	2	30	16.2	8

[a]number of feedback variables (after removing redundant vertices)
[b]shortest feedback chain (excludes feedback equations)
[c]longest feedback chain (excludes feedback equations)
[d]average length of all feedback chains (excludes feedback equations)
[e]average number of passes of simultaneous block needed to calculate a jacobian of the feedback variables (includes feedback equations)
(Fbcavg + nfb) × nfb / Sblock (rounded up)

there are some quite short feedback chains and that even the longest feedback chain does not cover the complete simultaneous block. The column labelled Avgpass gives the average number of passes of the simultaneous block needed to evaluate the full jacobian of the feedback variables. The brute force method for calculating the jacobian requires as many passes as there are feedback variables. Especially for the larger models it is clear that a significant reduction in the number of passes can be achieved by using the feedback chain information. If many jacobian recalculations are necessary, this can lead to a considerable reduction in computer time needed for simulating the model.

5.3 Random graphs

Pardalos et al. (1999) and Festa et al. (2001) describe and implement an alternative algorithm for determining redundant feedback vertices. Festa et al. (2001) present Fortran code for determining feedback vertex sets including a method for determining redundant feedback vertices. The algorithm, available as Algorithm 815 in the TOMS collection, utilises the graph contraction rules described in section 2. The algorithm for identifying redundant feedback vertices incorporated in Algorithm 815 is the basic algorithm given in figure 4.1.[9]

In both of the mentioned papers the authors test their procedure on a set of randomly generated directed graphs; it is these graphs that will be used in this section.

The first test performed is of Algorithm A and B with the scalar selection parameter α set to 1 and the first vertex in R used whenever a heuristic choice has to be made. A summary of the

[9] The data for the graphs and the source code can be downloaded from the ACM website.

results is given in table 5.3.[10] The table shows the cutset sizes for the random graphs used in Festa et al. (2001). The last column gives the differences in feedback set sizes for the two algorithms. Algorithm B generates slightly smaller cutsets in nine cases and never generates larger cutsets than the algorithm A.

Some statistics for the feedback sets obtained with Algorithm A and B are given in table 5.4. Most of the feedback vertices in the feedback set are selected by the heuristic rule. For the larger graphs the number of redundant feedback vertices detected and removed, if any, varies between 1.5% and 7% of the size of the feedback set.[11] We now turn to a comparison of the efficiency of the various algorithms for detecting redundant feedback vertices, as measured by the amount of Cpu seconds used by the algorithms.[12] Table 5.5 gives the CPU seconds used by each of the algorithms for the larger graphs. From the table it can be seen that algorithm B is slightly faster than Algorithm A for graphs with a higher ratio of the number of edges to the number of vertices. The two last columns in the table give the amount of cpu seconds used for the basic algorithm of figure 4.1 and the modified version of the basic algorithm given in figure 4.2. It is clear from the table that the basic algorithm is very slow and becomes progressively more inefficient as the edge to vertex ratio for the graphs and the number of redundant feedback vertices increases. The modification of the basic algorithm is an improvement of the basic algorithm as can be seen from the last column of table 5.5.

These results show that Algorithm A and B are a significant improvement of the basic algorithm.

[10] During the tests I found that the implementation of the redundancy checking algorithm in the subroutine `local` contains a flaw which leads the published version of Algorithm 815 to miss a lot of redundant feedback vertices. In the code of subroutine `local` the argument `cset(j)` in the call to subroutine `delv` should be replaced by `cset0(j)`. A corrected version of Algorithm 815 gives results identical to those for Algorithm A in table 5.3.

[11] This is more than the 1–3 reported in (Pardalos et al., 1999, page 405). See footnote 10.

[12] On a PC with Pentium 4 800Mhz, Windows 2000 Professional, Lahey-Fujitsu Fortran compiler 5.7a.

Table 5.3 Comparison of algorithm A and B

		Size of feedback sets		
Vertices	Edges	Algorithm A	Algorithm B	B − A
50	100	3	3	0
50	150	9	9	0
50	200	14	14	0
50	250	19	19	0
50	300	20	20	0
50	500	29	29	0
50	600	34	34	0
50	700	33	33	0
50	800	37	37	0
50	900	37	37	0
100	200	9	9	0
100	300	17	17	0
100	400	25	25	0
100	500	34	34	0
100	600	43	43	0
100	1000	55	55	0
100	1100	58	58	0
100	1200	64	64	0
100	1300	64	64	0
100	1400	66	66	0
500	1000	34	34	0
500	1500	73	73	0
500	2000	112	112	0
500	2500	152	152	0
500	3000	180	178	−2
500	5000	258	258	0
500	5500	275	275	0
500	6000	286	285	−1
500	6500	304	303	−1
500	7000	313	311	−2
1000	3000	154	154	0
1000	3500	184	184	0
1000	4000	218	218	0
1000	4500	261	261	0
1000	5000	293	288	−5
1000	10000	518	514	−4
1000	15000	635	632	−3
1000	20000	700	692	−8
1000	25000	748	745	−3
1000	30000	784	784	0

Table 5.4 Statistics for feedback sets generated with algorithm A and B

		Algorithm A			Algorithm B		
Vertices	Edges	Feedback set	Heuristic	Redundants	Feedback set	Heuristic	Redundants
50	100	3	1	0	3	1	0
50	150	9	4	0	9	4	0
50	200	14	10	0	14	10	0
50	250	19	15	0	19	15	0
50	300	20	15	0	20	15	0
50	500	29	25	1	29	25	1
50	600	34	29	1	34	29	1
50	700	33	29	1	33	29	1
50	800	37	35	0	37	35	0
50	900	37	33	0	37	33	0
100	200	9	2	0	9	2	0
100	300	17	12	1	17	12	1
100	400	25	13	0	25	13	0
100	500	34	25	0	34	25	0
100	600	43	36	1	43	36	1
100	1000	55	49	3	55	49	3
100	1100	58	50	4	58	49	4
100	1200	64	58	0	64	58	0
100	1300	64	55	1	64	55	1
100	1400	66	62	0	66	62	0
500	1000	34	22	1	34	22	1
500	1500	73	65	4	73	65	4
500	2000	112	105	6	112	105	6
500	2500	152	144	8	152	144	8
500	3000	180	168	9	178	162	11
500	5000	258	247	11	258	244	11
500	5500	275	265	14	275	260	14
500	6000	286	273	12	285	268	13
500	6500	304	297	11	303	292	12
500	7000	313	300	11	311	295	13
1000	3000	154	140	4	154	140	4
1000	3500	184	168	3	184	167	3
1000	4000	218	206	15	218	204	15
1000	4500	261	249	9	261	246	9
1000	5000	293	278	14	288	269	19
1000	10000	518	503	23	514	494	27
1000	15000	635	627	14	632	620	17
1000	20000	700	689	25	692	677	33
1000	25000	748	737	16	745	727	19
1000	30000	784	771	20	784	766	20

Table 5.5 CPU seconds for four algorithms

		Cpu seconds			
Vertices	Edges	Algorithm A	Algorithm B	Basic algorithm	
				Standard	Modified
500	2000	0.05	0.03	0.39	0.11
500	2500	0.05	0.05	0.56	0.16
500	3000	0.06	0.06	0.67	0.16
500	5000	0.09	0.09	1.31	0.27
500	5500	0.09	0.09	2.03	0.30
500	6000	0.09	0.09	1.69	0.33
500	6500	0.11	0.11	1.81	0.34
500	7000	0.11	0.11	2.31	0.42
1000	3000	0.09	0.08	0.67	0.23
1000	3500	0.11	0.13	0.38	0.30
1000	4000	0.17	0.17	3.09	0.42
1000	4500	0.17	0.19	2.47	0.52
1000	5000	0.20	0.22	3.77	0.59
1000	10000	0.39	0.34	20.55	1.78
1000	15000	0.47	0.42	24.81	3.31
1000	20000	0.70	0.52	63.27	5.00
1000	25000	0.83	0.59	68.63	7.08
1000	30000	1.03	0.78	105.31	9.52

6 Conclusion

The new algorithms for detecting and removing redundant feedback vertices from the cutset of a directed graph presented in this paper appear to offer significant improvements compared to the algorithm of Festa et al. (2001). The former are much faster and in some cases also generate slightly smaller cutsets.

This paper also gave a method for determining the feedback chains of a directed graph. The information so obtained can be used for efficiently calculating the jacobian matrix of the feedback variables of large sparse normalised systems of equations.

References

Becker, R. and B. Rustem, 1993, Algorithms for solving nonlinear dynamic decision models, *Annals of Operations Research*, vol. 44, pp. 117–142.

CPB, 1999, *WorldScan; the core version*, CPB Special Publications 20, CPB.

CPB, 2002, *SAFE Een kwartaalmodel van de Nederlandse economie voor korte-termijn analyse*, CPB Document 27, CPB, ISBN 90-5833-120-2.

Don, F.J.H. and G.M. Gallo, 1987, Solving large sparse systems of equations in econometric models, *Journal of Forecasting*, vol. 6, pp. 167–180.

Festa, P., P.M. Pardalos and M.G.C. Resende, 1999, Feedback set problems, in D.Z. Du and P.M. Pardalos, eds., *Handbook of Combinatorial Optimization*, vol. A, pp. 209–258, Kluwer Academic Publishers.

Festa, P., P.M. Pardalos and M.G.C. Resende, 2001, Algorithm 815: FORTRAN subroutines for computing approximate solutions of feedback set problems using GRASP, *ACM Transactions on Mathematical Software*, vol. 27, no. 4, pp. 456–464.

Graafland, J. and R. de Mooij, 1998, MIMIC: An applied general equilibrium model for the Netherlands, *CPB Report*, no. 1998/3, pp. 27–31.

Laxton, D., P. Isard, H. Faruqee, E. Prasad and B. Turtelboom, 1998, Multimod mark III, the core dynamics and steady state models, IMF Occasional Papers 164.

Levy, H. and D.W. Low, 1988, A contraction algorithm for finding small cycle cutsets, *Journal of Algorithms*, vol. 9, pp. 470–493.

Lin, H.M. and J.Y. Jou, 2000, On computing the minimum feedback vertex set of a directed graph by contraction operations, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 19, no. 3, pp. 295–307.

Orenstein, T., Z. Kohavi and I. Pomeranz, 1995, An optimal algorithm for cycle breaking in directed graphs, *Journal of Electronic Testing; Theory and Applications*, vol. 7, pp. 71–81.

Pardalos, P.M., T. Qian and M.G.C. Resende, 1999, A greedy randomized adaptive search

procedure for the feedback vertex set problem, *Journal of Combinatorial Optimization*, vol. 2, pp. 399–412.

Smith, G.W., Jr. and R.B. Walford, 1975, The identification of a minimal feedback vertex set of a directed graph, *IEEE Transactions on Circuits and Systems*, vol. 22, no. 1, pp. 9–15.

Timmer, H., 1998, WorldScan: A world model for long-term scenario analysis, *CPB Report*, no. 1998/3, pp. 37–41.

Vromans, M., 1998, ATHENA: the multi-sector model, *CPB Report*, no. 1998/3, pp. 32–36.