MODIFIED DEPTH FIRST SEARCH FOR STATE-SPACE REPRESENTATION OF BLOCK DIAGRAMS.

M Bacic, R W Daniel

Oxford University Department of Engineering Science
Parks Road
Oxford OX1 3PJ

Abstract: A new graph pruning algorithm is presented that overcomes some the difficulties of depth first search when building a state-space description of an input-output model prior to control system analysis. The algorithm is suitable for manipulating state-space models directly without transformation into the Laplace domain and overcomes some the numerical problems associated with existing techniques. The approach is particularly relevant to Web-based control system design.

Keywords: Algorithms; Block diagrams; Computer-aided control system design

1. INTRODUCTION

There are a number of commercially available computer aided control system design packages, such as Matlab™. There is also interest in remote access to design suites over the Web. However, the most common approach is for the remote client to present a script file to the server to be run, the Web browser being used effectively as a smart terminal. A more sophisticated approach is to provide intelligent agents on the server, written in a browser-supported and machine-independent language such as JAVA™, and to download this applet to carry out the design. In order to do this the applet needs to have a minimal set of numerical algorithms that can carry out simple design analysis on a system model. The work presented in this paper grew out of a student project that aimed to investigate the feasibility of generating such an applet.

A key step in building an applet for control system design is the representation of a system model that has been entered as an interconnected set of simple system models (such as transfer functions in the Laplace variable s). The entry of such a model and its graphical editing is outside the scope of this paper. Once a model has been entered one is faced with transforming it into a form so that input-output transmittances can be identified and calculated. The classical approach is via some form of Gaussian elimination on the set of linear equations describing the block diagram. In this paper a new approach is developed for simple single-input single-output models that can be readily extended to multiple-input multiple-output systems.

Munro (1977) states that any set of simultaneous differential and algebraic equations describing a physical system with unidirectional signal transmission can be represented graphically by a signal-flow graph consisting of a set of nodes interconnected by directed branches. Most of the literature relevant to this problem dates from the 60s, and 70s and relates to signal flow graph reduction. Abrahams (1966a, 1966b) states that the reduction of signal flow graphs is really a graphical equivalent to the method of Gaussian elimination. In his model, transmittances of the flow graph are in the form of transfer-function relationships between variables. Munro (1977) however, states that the signal flow graph representation of equations describing a physical system is not restricted to the form of a transfer function. The transmittances can be in the form of constants with differential operator, polynomials in the Laplace s-operator (Rosenbrock, 1974) or in the form of ratios of polynomials in s.

To find the relationship between two particular points in a signal flow graph, and therefore between the "input" and the "output", one can use Mason's gain formula (Mason and Zimmerman, 1960). It is useful to quote this formula. The determinant of a flow graph is defined as

$$\Delta = 1 - \sum \text{loop gains} + \sum \text{products of loop-gain of pairs of disjoint loops} - \sum \text{products of loop-gain of triples of disjoint loops} + \cdots$$

For the k^{th} path in the signal flow-graph with non-zero open-loop path gain, define a cofactor of the flow graph with respect to this path as

$$\Delta_k = 1 - \sum_{\text{disjoint from the path}}^{\text{loop gains that are}} + \sum_{\text{of disjoint loops that are disjoint from path}}^{\text{products of loop-gain of pairs}} - \sum_{\text{of disjoint loops that are disjoint from the path}}^{\text{products of loop-gain of triples}} + \cdots$$

Hence, according to Mason (1966), the overall gain (or a transfer function) of the original flow-graph from "source" to "sink" is given by

$$H = \frac{1}{\Delta}\sum_k g_k \Delta_k \tag{1}$$

where g_k is the transmittance from the source to the sink of path k. However, to find this gain one must find all the paths from the "source" to the "sink", plus pairs, triples, etc. of the disjoint loops. Finding loops is a Hamiltonian cycle problem (Cormel, *et al.*, 1997). Hence Mason's procedure is NP-complete. This fact precludes any possibility of its general use. An approach taken by Munro (1977) introduces a connection matrix based on Abrahams' transmittance matrix with an extension to take into account MIMO systems. Given a transmittance matrix **T** of dimension n x n, where $T_{ij} = g_{ij}$ is a transmittance from node i to node j, and n the number of nodes, one introduces $l+m$ artificial nodes where l is the number of inputs of a system and m the number of outputs. The newly introduced artificial input and output nodes are connected to the corresponding real nodes inside a flow graph via unit transmittances. Let the augmented matrix be called Γ where

$$\Gamma_{ij} = \begin{cases} g_{ij}, \textit{for } 0 < i, j < n \\ 0, \textit{for } n < i < n+m, n < j < n+1 \\ 1 \textit{ or } 0 \textit{ otherwise} \end{cases} \tag{2}$$

Munro (1977) then states that in order to obtain effective transmittances between any arbitrary pair of nodes (i,j) follow the following simple procedure on Γ:

```
j←1
do j≠n→
    if γjj=0→
        δ=1
    [] γjj≠0→
        δ=(1-γjj)^-1
    fi
    i=j+1
    do i≠n+m→
        if γij≠0→
            k = j+1
            do k≠n+1→
                γik←γik+δγijγjk
                k←k+1
            od
        fi
        i←i+1
    od
    j←j+1
od
```

Having defined Γ according to the above one can write in compact form:

$$\Gamma = \begin{bmatrix} T(s) & V \\ U & 0 \end{bmatrix}, \quad \tilde{\Gamma} = \begin{bmatrix} \tilde{T}(s) & \tilde{V} \\ 0 & P(s) \end{bmatrix} \tag{3}$$

where $\tilde{\Gamma}$ is obtained as a result of the execution of the above algorithm. Here, $\tilde{T}(s)$ is the transpose of the transfer function matrix **G**(s), relating inputs (specified by **U**) to the outputs (specified by **V**). This method would be useful if it were not for the need to manipulate transfer functions. This clearly requires symbolic manipulation and problems of numerical conditioning arise. One could, on the other hand, argue that the same procedure could be applied using state-space models instead of transfer functions. However, it is important to note that one frequently needs to evaluate the expression $(1-\gamma_{jj})^{-1}$. It is not impossible to imagine the situation when **D** corresponding to the state-space description of γ_{jj} equals 1. If such a situation arises (and it almost certainly will at some point during the execution) then the evaluation of the expression will be impossible in the state-space form.

In conclusion, to the best of the authors' knowledge, model reduction using state-space representation of individual components in a numerically efficient manner is an open problem. It is this problem that is addressed in this paper.

Paper Structure

Following this introduction, Section 2 defines the problem to be solved. Section 3 describes classical graph pruning and then introduces a modified depth first search algorithm. Its properties are described as well as why it addresses the problem of complexity with respect to successive loop closure. Section 4 describes how loops can be represented and identified in the structures developed in Section 3 and forward plus feedback loop-closure algorithms are presented. Section 5 is a brief conclusion that assesses the performance of the system

2. PROBLEM DEFINITION

One starts from a system signal flow graph. Solving the problem for the signal flow graph automatically solves the problem on block diagrams.

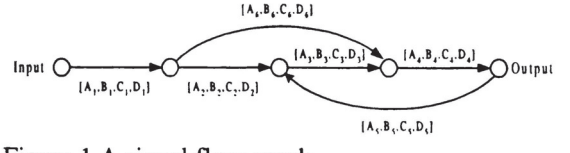

Figure 1 A signal flow graph

A signal flow graph is a network of nodes interconnected by directed branches, representing a

set of algebraic and differential equations (Abrahams, 1966a, 1966b). The nodes in a flow graph represent the variables whereas the joining branches represent coefficients relating these variables one to another. Figure 1 shows a simple signal flow graph where the edges are identified to a single-input single-output state-space quadruples [A, B, C, D]. This simple flow graph has a direct path between the input and output as well as a forward loop (sub-system 6) and a feedback loop (subsystem 5). It is a simple matter to write down the state-space equations for a system with no loops, but both forward and feedback loops generate algebraic dependencies. These dependencies need to be solved in order to arrive at the input-output state-space description. If there are a large number of components in the model, this computation becomes highly complex. The classical approach is to solve a set of linear equations where the variables are transfer functions, i.e. the system model is not manipulated in state-space form. The reason is that there is no explicit solution for state-space edges.

The following simple lemma is stated in order to motivate the later work:

Lemma 1
The set of signal flow graphs that are simple trees are directly representable as state space quadruples.

Proof.
By direct computation. A simple tree does not contain loops. As a consequence, there exists a unique path from node i to node j in the tree. Hence, to obtain g_{ij}, (a single-input, multiple-output model of the transmittance between nodes i and j) one simply cascades transmittances of adjacent nodes on the path from i to j. The algebra for this is straightforward and may be found in Maciejowski (1989).

Definition
Loop closure. Given an open-loop state space model **A,B,C,D** of a simple spanning tree from the signal flow graph, a loop closure between nodes u and v with a feedback/feedforward gain yields an overall closed loop state space model **A', B', C', D'** that accounts for the introduction of the specific feedback/feedforward gain between nodes u and v.

Corollary to Lemma 1
If a system model can be broken into a simple tree then one can solve the problem of generating a state-space quadruple relating input to output by successive loop closure.

Proof.
Successively closing loops eventually includes all the edges of the original signal flow graph and the simple tree becomes equivalent to the signal flow graph. The invariant of the loop closure is the state-space model corresponding to the current graph. Hence, once one has built the signal flow graph from the simple tree, the closed-loop state-space model of the system follows directly.

The main problem addressed in this paper is breaking a signal flow graph into a tree in an efficient manner with respect to loop closure.

Problem Statement
'How can one convert a signal flow graph into a simple tree and generate a structure that is efficient with respect to successive loop closure in state space?'

3. GRAPH PRUNING ALGORITHMS.

Converting a graph into a spanning tree is equivalent to graph pruning. This may be carried out by performing a depth-first search from input to output, deleting edges that form loops. The classical depth-first search algorithm is formulated in a manner that guarantees termination with a spanning tree in a finite number of steps.

Definition: Depth First Search (DFS)
Given a graph $G = \langle V, E \rangle$, (V is the set of vertices, E the set of edges) the algorithm obtains a spanning tree $T = \langle V, E' \rangle$ where $E' \subseteq E$ and $cardE' = cardV - 1$. Here the algorithm is stated in pseudo code.

```
procedure search(G)
    for each v∈ V→
        visited[v]←false
    endfor
    for each v∈V→
        if visited[v]=false→
            dfs(v)
        fi
    endfor
endproc
procedure dfs(v:V)
    visited[v]←true
    for each w adjacent to v→
        if visited[w]≠true→
            dfs(w)
        fi
    endfor
endproc
```

where *visited[i]* keeps track of whether node i has already been visited. The computational complexity for the adjacency list representation is $O(\max(cardE, cardV))$ (see Brassard and Brately, 1988 for proof). It is important to note that one never visits a node twice. In the procedure **dfs** when a node w has already been visited one simply skips the node. Suppose that the state-space representation of the spanning tree has been obtained by the above procedure. From Corollary 1, the state-space description of the original signal flow follows by successive loop closure. However, closing the loops

requires the augmentation of **A,B** since the feedback / feedforward gain will be in general a transmittance described by a state-space model. The augmentation of **A** would be particularly expensive as it involves copying matrices, which is $O(n^2)$. Note that this is the consequence of excluding some of the original edges of the signal flow graph in construction of the spanning tree by the procedure **dfs**.

The only way to avoid augmentation of **A** is by having a tree structure which contains all the edges of the original signal flow graph. This leads to a modified version of depth-first search.

Modified Depth First Search (MDFS)

Note that in the procedure **dfs**, when a node w has already been visited, a node is simply skipped. This is the point where MDFS differs from standard DFS. When examining nodes adjacent to a node v, breed those nodes that have already been visited. In this way the tree structure is maintained, yet all the edges are included from the original graph. The result of MDFS will be a tree with exactly $cardE$ nodes and $cardE-1$ edges. Suppose node w in the inner loop has already been visited. Create a node w' and add it to the adjacency list of v. At the same time remove w from the adjacency list of v[1]. Subsequently, add w to the adjacency list of w'. The value of edge (w',w) will of course be **D = 1**, indicating a unit transfer function with no dynamics. Note, that this does not change the relationship between the nodes (v,w) since (w',w) represents unit gain and (v,w') is given by the same **A,B,C,D** as (v,w). More formally, one writes:

$$E' = E \setminus (v,w) \cup \{(v,w') \cup (w',w)\}$$
$$V' = V \cup w' \qquad (4)$$

Now modify the **dfs** procedure to account for these modifications. Also assume that the given graph is connected, and that one starts from the designated input node, labelled *source*.

```
procedure search(G)
    for each v∈ V→
        visited[v]←false
        depth[v] ←0
    endfor
    V'=∅
    E'=∅
    dfs(source)
endproc
procedure dfs(v:V)
    visited[v]←true
    V'=V'∪{v}
    for each w adjacent to v→
        if visited[w]≠true→
            dfs(w)
            E'=E'∪(v,w)
            V'=V'∪{w}
            path[w]=v
            depth[w]=depth[v]+1
        [] visited[w]=true→
            V'=V'∪{w'}
            E'=E'∪{(v,w'),(w',w)}
            path[w']=v
            depth[w']=depth[v]+1
        fi
    endfor
endproc
```

Here *path[i]* provides information about how the node i is reached, and *depth[i]* records how many nodes were passed from the *source* before reaching node i. Noting that the above procedure includes all the edges of the original signal flow graph, it is easy to see that the complexity of the procedure for the adjacency list representation of the graph is $O(cardE)$. Also note that the spanning tree of the new signal flow graph can be easily recovered using *path[i]*.

4. OPEN-LOOP STATE-SPACE MODEL OF THE AUGMENTED GRAPH

MDFS creates a new signal flow graph $G' = \langle V', E' \rangle$ whose spanning tree can be recovered by using *path[i]*. Furthermore, the spanning tree of G' contains all the edges of G. The state-space model of the spanning tree is the open-loop state-space model of G. Denoting the output node as *source*, proceed as follows:

i. Build an open loop state space model from *source* to *sink*, keeping track of **C[u]**, **D[u]** for each node u visited on the path *source→sink*. This simply involves cascading state-space representations associated with the edges on the given path. Mark all the nodes on the path as being *visited*. At the end of this part, **A, B, C**[sink], **D**[sink] represents the open-loop state-space model of the system.

ii. Find the node u, such that *depth[u]* is the maximum of all the depths of as yet unvisited nodes. Trace the path *source→u*, and for all nodes v not visited before calculating **C**[v], **D**[v]. Furthermore, for all edges not visited, augment **A, B** accordingly. Also note that the open-loop model of *source→sink* as well as for any other $u \rightarrow v$ remains unchanged.

iii. Repeat (ii) until all the nodes in G' have been visited

The above procedure obtains {**A, B**} and set of {**C**[u], **D**[u]} that enables one to read off the open loop relationship between the *source* and any other node including *sink*. The above procedure can be expressed in more formal form as follows

[1] Note that deletion actually means replacing edge (v,w) with (v,w').

```
Stack S;
    u←sink
    do path[u]≠source→
        S.push(u)
        u = path[u]
    od
    u←source
    s←StateSpace(empty);
    do S≠empty→
        v←S.pop(u)
        visited[v]←true
        s.concat((u,v).statespace)
        C[v]←s.GetC()
        D[v]←s.GetD()
        v←u
    od
    do Set(visited)∉∅ →
        m←0
        v←0
        do for all v∈V ∧ visited[v]≠true→
            if depth[v]>m→
                m = depth[v]
                u=v
            fi
        od
        branch←StateSpace(empty)
        do visited[path[u]]≠true→
            S.push(u)
            u = path[u]
        od
        u←path[u]
        if u=source→
            v←S.pop(u)
            visited[v] ←true
            C[v] ←(u,v).statespace.GetC()
            D[v] ←(u,v).statespace.GetD()
            branch.concat((u,v).statespace)
            u←v
        fi
        do S≠empty→
            v←S.pop(u)
            visited[v]←true
            edge←(u,v).statespace
            branch.concat(edge)
            C[v]←[edge.GetD()C[u] edge.GetC()]
            D[v]←edge.GetD()D[u]
            v←u
        od
        newA←[ s.GetA()                0
               branch.GetB()C[u]  branch.GetA() ]
        newB←[ s.GetB()
               branch.GetB()s.GetD() ]
        s←StateSpace(newA,newB,s.GetC(),s.GetD())
    od
```

5. LOOP CLOSURE ALGORITHM

In order to successively close the loops around the spanning tree built using the procedure in Section 4, one must identify the type of loop being closed. Consider two vertices $i,j \in V$. Let *depth[w]* represent the distance of the node $w \in V$ from the input node, denoted as *source*. One might think that there are three different cases to consider, namely

1. depth[i]<depth[j]
2. depth[i]>depth[j]
3. depth[i]=depth[j]

basing decisions about the type of loop connection (parallel or feedback). However Figure 2 clearly indicates that this approach is incorrect.

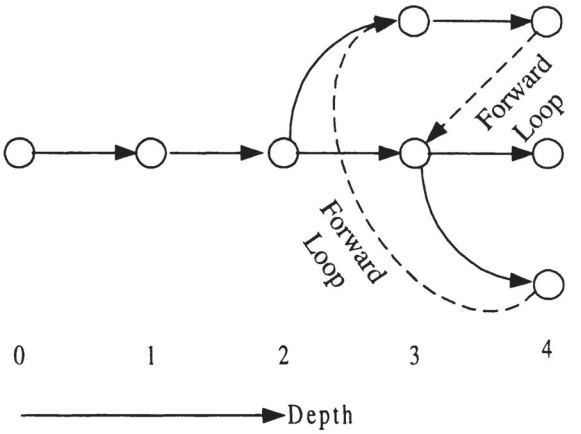

Figure 2 Illustration of potential errors in identifying forward loops.

To distinguish between parallel and feedback connections, one has to consider the basic nature of these connections. A parallel connection is one where the output in no way affects the input whereas the feedback connection is one where the output does affect the input (there is a causal relationship). Consider, as before, two vertices $i,j \in V$. Let $\mathbf{L}_{ij}$ be a Boolean connection matrix for the current, partially closed signal flow graph[2], defined as follows

$$L_{ij} = \begin{cases} \text{true, for } i \rightarrow j \\ \text{false, otherwise} \end{cases} \qquad (5)$$

where $u \rightarrow v$ denotes an operator on u,v which returns *true* if there exists an arbitrary path (it does not need to be direct path) from u to v, and *false* otherwise. So to inspect whether closing the path from j to i forms a feedback connection, one only needs to check whether $j \rightarrow i$. This can be easily found by inspecting the matrix $\mathbf{L}_{ij}$. Obtaining the matrix $\mathbf{L}$ is a simple *transitive closure* problem. Hence, one can directly apply the *Floyd-Warshall* algorithm on the spanning tree of the signal flow graph to obtain $\mathbf{L}$ (see Corman et al, 1997, Brassard and Brately, 1988). It is,

[2] A partially closed signal flow graph means a spanning tree with some of the loops closed

however, very important to emphasise that once a loop is closed, new indirect paths are created and therefore **L** needs updating.

Having labelled the loop closure, one can close the loop by applying corresponding formulae for forward and feedback loops.

Forward loop closure
The formulae for two systems connected in parallel (forward loop case) are (Maciejowski, 1989):

$$A=\begin{bmatrix} A_1 & 0 \\ 0 & A_2 \end{bmatrix},\quad B=\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}$$
$$C=\begin{bmatrix} C_1 & C_2 \end{bmatrix},\quad D=D_1+D_2 \qquad (6)$$

However, in this paper, all the dynamics are included in **A** so the second system is simply a unity gain. Furthermore, in this case, the forward loop does not need to be from the input to the output. It is rather a forward loop from node *i* to node *j*. Hence, assuming C_i , C_j have been appropriately augmented with zeros so that their length is *dimA*, one can write

$$\begin{aligned} C_j' &= C_i + C_j \\ D_j' &= D_i + D_j \end{aligned} \qquad (7)$$

Having calculated the new C_j, D_j, all C_k, D_k that are dependent on C_j, D_j need updating. One might think that the above is sufficient to obtain the closed loop state-space model. This is true provided *j* were the output node. However, if *j* is connected to a node *k* via a transfer function *T(s)*, with corresponding state space description {**A'**, **B'**, **C'**, **D'**}, then the above does not suffice. The injected signal at point *j* is $\mathbf{C}_i x+\mathbf{D}_i u$. Hence the injection of a signal from *i* to *j* (the input of *T(s)*) causes a change in the appropriate states by an amount $\mathbf{B'C}_i$. Similarly, both **B** of the closed loop system and $\mathbf{C}_k,\mathbf{D}_k$ change appropriately according to the formulae for a cascade realisation. A change in $\mathbf{C}_k,\mathbf{D}_k$ has occurred and its effect has to be propagated further, until all the nodes connected to *j* are updated.

Feedback loop closure
The formulae for two systems in feedback connection are given by (Maciejowski, 1989):

$$A=\begin{bmatrix} A_1-B_1X^{-1}D_2C_1 & -B_1X^{-1}C_2 \\ B_2\left[I-D_1X^{-1}D_2\right]C_1 & A_2-B_2D_1X^{-1}C_2 \end{bmatrix},$$
$$B=\begin{bmatrix} B_1X^{-1} \\ B_2D_1X^{-1} \end{bmatrix}$$
$$C=\begin{bmatrix} (I-D_1X^{-1}D_2)C_1 & -D_1X^{-1}C_2 \end{bmatrix}$$
$$D=D_1X^{-1} \qquad (8)$$

where $X=1+D_2D_1$. Again note that the dynamics have already been built in the system, and that one does not necessarily close the feedback between the input and the output, thus

$$\begin{aligned} A' &= A-B_jX^{-1}C_j+B_jX^{-1}C_i \\ B' &= B+(B_jX^{-1}D_i-B_j) \\ B_j' &= B_jX^{-1}D_i \\ C_j' &= (1-D_jX^{-1})C_j+D_jC_iX^{-1} \\ D_j' &= D_iD_jX^{-1} \end{aligned} \qquad (9)$$

Having found the new $\mathbf{C}_j'$, $\mathbf{D}_j'$ we proceed with updating all the $\mathbf{C}_k,\mathbf{D}_k$, where *k* is connected to *j*.

CONCLUSION

Formal definitions of algorithms that compute the state-space representation of an input-output model given a general state-space graph have been presented. The algorithms are provably correct and are based on well-established graph theory. A new depth-first search algorithm has been presented that minimises and simplifies algebraic calculations during loop closure. Such a consideration is essential for a simple Web based control system design tool when bandwidth, space and copyright constraints precludes the use of sophisticated mathematical packages. The algorithms presented provide part of the mathematical engine underlying a simple CAD package about to go live on the Oxford University undergraduate web server.

REFERENCES

Munro, N. (1977) Composite System Studies Using the connection matrix, *International Journal of Control,* Vol.**26**, pp.831-839.

Abrahams, J.R. (1966a) Signal Flow Graphs I, *Control*, pp.31-33.

Abrahams, J.R. (1966b) Signal Flow Graphs II", *Control*, pp.88-90.

Maciejowski, J.M. (1989) *Multivariable Feedback Design*, Addison-Wesley.

Mason, S.J. and H.J. Zimermann (1960) *Electronic Circuits, Signals and Systems,* Wiley.

Rosenbrock, H.H. (1974) *Computer-Aided Control System Design*, Academic Press.

Kailath, T. (1980) *Linear Systems*, Prentice-Hall.

Cormen, T.H., C.E. Leiserson and R.L. Rivest (1997) *Introduction to Algorithms*,MIT Press.

Brassard, G. and P.Brately, (1988) *Algorithmics-Theory and Practice,* Prentice-Hall.