



**HAL**  
open science

## **LISA: a linear structured system analysis program**

Sinuhé Martinez-Martinez, Theodor Mader, Taha Boukhobza, Frédéric Hamelin

► **To cite this version:**

Sinuhé Martinez-Martinez, Theodor Mader, Taha Boukhobza, Frédéric Hamelin. LISA: a linear structured system analysis program. 3rd IFAC Symposium on System, Structure and Control, SSSC'07, Oct 2007, Foz do Iguacu, Brazil. pp.CDROM. hal-00159436

**HAL Id: hal-00159436**

**<https://hal.science/hal-00159436>**

Submitted on 3 Jul 2007

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

## **LISA: A LINEAR STRUCTURED SYSTEM ANALYSIS PROGRAM**

**S. Martinez-Martinez, T. Mader, T. Boukhobza, and  
F. Hamelin**

*Research Center in Automatic Control (CRAN - CNRS UMR  
7039), Nancy University, BP 239, 54506 Vandœuvre Cedex,  
Nancy, France, Phone: 33 383 684 464, Fax: 33 383 684 462,  
email: sinuhe.martinez@cran.uhp-nancy.fr*

**Abstract:** In this paper, the so-called software LISA is presented. LISA is a flexible and portable software, which has been developed to analyse structural properties of large scale linear and bilinear structured systems. More precisely, LISA contains programmed algorithms, which allow us to apply recent results in the analysis of structured systems.

**Keywords:** Structured systems, graph theory, observability, isolability.

### 1. INTRODUCTION

Structured systems have received much attention since the beginning of the 70's. Based on the work of (Lin 1974), where graphic conditions for the structural controllability are given, (Reinschke 1988) and (Murota 1987) propose theoretic algorithms for studying the structural properties of multi-input linear systems, like controllability, observability. The main results concerning graph theoretic approach are summarized in (Dion et al. 2003).

Recently, some algorithms have been proposed for the analysis of structured linear systems. In (Hovelaque et al. 1996), for example, the primal-dual algorithm is proposed to derive the infinite structure of a structured system. Later, the authors have implemented the algorithm to analyse the solvability of disturbance decoupling problem (Hovelaque et al. 1997).

In this context, (Blanke and Lorentzen 2006) present a Matlab toolbox called SaTool, in which are implemented some results concerning structural analysis theory like reachability, controllability and fault detectability. SaTool uses mainly the bipartite graph to represent structured systems.

In this article, a new tool for the analysis of structured linear and bilinear systems is presented. This tool is based on the representation of structured systems by directed graphs (or digraphs). In fact, it is possible to transform graphic conditions given in terms of digraphs into flow graph conditions. Implementation of flow graphs is relatively easy and mainly efficient (lower computational burden) because there already exist many optimized algorithms useful to analyse structured system properties. The first version of LISA program deals with some new results concerning generic unknown input and state observability and fault isolability of structured linear systems and observability of structured bilinear systems. In order to deal with such complex problems, basic tools have been developed, these basic tools can be used to solve many other structural proprieties as the optimization of sensor placement for observability and FDI, the so-called autonomy for the networked systems, etc.

The paper is organized as follows: in section 2, structured linear and bilinear systems are presented, as well as their graphical representation. In section 3, the LISA program is presented, different useful algorithms are summarized and explained. An estimation of their complexity orders is given. Finally, in section 4, we give the conclusion of the paper.



$\mathcal{G}(\Sigma_{\Lambda}^b) = (\mathcal{V}, \mathcal{E})$ . Vertex set is defined identically as in the linear case. Edge set is  $\mathcal{E} = \bigcup_{l=0}^m A_l\text{-edges} \cup B\text{-edges} \cup C\text{-edges} \cup D\text{-edges} \cup E_1\text{-edges} \cup E_2\text{-edges} \cup F_1\text{-edges} \cup F_2\text{-edges}$ . Note that, we indicate the number  $i$  under each  $A_i$ -edge in order to preserve the information about the belonging of the edges in the digraph representation. With this aim, we define the following vertex subset  $\mathcal{X}' = \{\mathbf{x}'_{k,i}\}$  where  $0 \leq i \leq n$  and  $1 \leq k \leq m$  and the edge sets  $A'_k\text{-edges} = \{(\mathbf{x}_j, \mathbf{x}'_{k,i}) | A_k(i, j) \neq 0\}$ .

*Example 2.* In Figure 2 is represented the digraph associated to the following structured bilinear system:

$$A_0 = \begin{pmatrix} \lambda_1 & \lambda_2 & \lambda_3 & 0 & 0 \\ \lambda_4 & 0 & \lambda_5 & 0 & 0 \\ 0 & 0 & \lambda_6 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_7 \\ 0 & 0 & 0 & \lambda_8 & 0 \end{pmatrix}, A_1 = \begin{pmatrix} 0 & \lambda_9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \lambda_{11} & \lambda_{12} & 0 \end{pmatrix},$$

$$A_2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \lambda_{13} \\ 0 & \lambda_{14} & 0 & 0 & 0 \end{pmatrix} \text{ and } C = \begin{pmatrix} \lambda_{15} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda_{16} & 0 \end{pmatrix}.$$

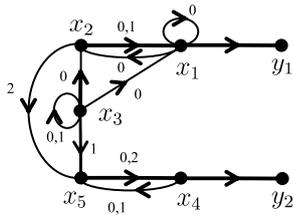


Fig. 2. Digraph for example 2

According to the graphic representation of structured linear systems by means of digraphs, we will present in the next section the different tools and functions, which are programmed in LISA.

### 3. LISA PROGRAM

LISA was developed in C++, a high level programming language, in order to reduce the computational burden. For the graphical user interface (gui), the library QT 4.0 was chosen. All these features make for LISA a flexible and portable program, which is suitable for Windows as well as Linux environment.

Roughly speaking, the program is divided into two parts: the user interface and the storages and calculation classes (*Graph*, *StateRecorder*, *MatrixExporter*). For the purpose of this paper, we will concentrate in only two classes: *Graph* and *MatrixExporter*. A very practical group of functions is programmed in *MatrixExporter* class. This class allows the user to export the graph into a state space representation, which can be read and used by Maple (version 10) and Mupad.

Most of the algorithms programmed in LISA are based on a flow graph approach. Digraph representation is

suitable for the visual interpretation of some theoretical conditions, which have to be verified by the system to ensure its observability, controllability, etc. However, it is important to note that, from a computational point of view, flow graph representation is more suitable than digraph representation. In LISA, a translation algorithm is used to convert digraphs into flow graphs and/or bipartite graphs. *FlowGraph* class contains different routines useful to these transformations. Namely, several algorithms are dedicated to find the maximum flow, to solve the minimum cost - maximum flow problem and to find essential vertices in a flow graph.

In this part, we detail some of the most important algorithms programmed in LISA. For this presentation, algorithms are divided into two main classes: basic graph properties (disjoint paths, essential vertices, etc) and structured system properties (input and state observability and fault isolability). The complexity order of every algorithm is provided in function of the total number of vertices  $N$  and the total number of edges  $M$ .

#### 3.1 User interface

In the user interface *uiMainWindow*, all the "visual effect" functions are programmed. The user must first choose if he works with a linear or bilinear system. Then, the user can create the digraph using the mouse to place vertices, edges and to choose the kind of vertices (state, output, input, fault or disturbance vertices). In the main window, several tools make possible copy, removing, and transformation of the existing vertices. The user can manipulate its graph very easily. Moreover, the digraph can be saved, printed into many formats. Finally, the structured matrices of the digraph can be extracted using Maple or Mupad formats.

#### 3.2 Algorithms for basic graph properties

- **Finding successors of a vertex subset:**  
This algorithm returns the successor (predecessor) vertices of a given vertex subset. Note that, if directed edge  $(\mathbf{v}_1, \mathbf{v}_2)$  belongs to  $\mathcal{E}$  then,  $\mathbf{v}_1$  is a predecessor of  $\mathbf{v}_2$  and  $\mathbf{v}_2$  is a successor of  $\mathbf{v}_1$ . The algorithm used to accomplish this task is directly derived from the Boost Library (Boost C++ Libraries 2005).
- **Finding predecessors of a vertex subset:**  
Finding predecessors in a graph is equivalent to determine successors in the reversed graph. According to the documentation, the complexity of the used algorithm is  $O(M)$ .
- **Finding the maximum number of disjoint paths between two vertex subsets:**  
This algorithm returns the maximal number of disjoint paths between two selected vertex subsets or

equivalently the size of the maximal linking between these vertex subsets. To cope with this problem, the digraph has to be transformed into a flow graph. This idea is inspired by the next two theorems (Bang-Jensen 2002):

- ▷ **Menger's theorem** : the maximum number of edge disjoint paths in a graph is equal to the size of the minimum edge cut in that graph, and
- ▷ **max flow min cut theorem** : the size of the minimum edge cut is equal to the value of the maximum flow in a network, where all edges have unit capacity.

A direct consequence of the latter theorems is that the maximum flow is equal to the maximum number of edges disjoint paths in a graph. However, in our case, vertex disjoint paths instead of edge disjoint paths are searched. Therefore, we transform the digraph into a flow graph in order to convert the vertex disjoint problem into an edge disjoint problem. Transformation of the original digraph  $\mathcal{G}(\cdot)$  carries out into two steps: vertex splitting and residual network procedure. Both steps are done by routines in the *FlowGraph* class. Once the transformation is made, an algorithm to solve the minimum cost - maximum flow problem is implemented. This consists essentially in the Ford Fulkerson algorithm (Bang-Jensen 2002) with a shortest path search at each step. When all the paths are tested, the algorithm stops and the maximum flow is given. There can be a maximum of  $O(N)$  disjoint paths and so path searches. The complexity order of each search is  $O(N\sqrt{M})$ . Hence the overall complexity order is  $O(N^2\sqrt{M})$ .

-  **essential vertices between two vertex subsets:**

The set of essential vertices between two vertex subsets  $V_1$  (source) and  $V_2$  (sink), noted  $V_{ess}(V_1, V_2)$ , corresponds by definition to vertices present in all maximum  $V_1$ - $V_2$  linkings. The used algorithm is based on the fact that a removal of an essential vertex causes a reduction in the number of disjoint paths. To determine the essential vertices in a digraph, each vertex is removed and the maximum flow between source and sink is calculated. If the new flow is smaller than the original flow, then the removed vertex is essential. The algorithm requires  $O(N)$  flow calculations. As we have seen, the complexity order of every flow calculation is  $O(N^2\sqrt{M})$ . Then, the overall complexity order is  $O(N^3\sqrt{M})$ .

-  **Input and output minimum separators between two vertex subsets:**

This algorithm returns input and output separators. A separator is a vertex subset, which contains at least one vertex in every path of a linking. We call minimum separators between  $V_1$  and  $V_2$  any separator having smallest size. According to Menger's Theorem, the latter is equal to the number of dis-

joint paths between  $V_1$  and  $V_2$ . There exist two uniquely determined minimum separators between  $V_1$  and  $V_2$ , called input and output separators.

- the minimum input separator is the set of the end vertices of all direct  $V_1$ - $V_{ess}(V_1, V_2)$  paths,
- the minimum output separator is the set of the begin vertices of all direct  $V_{ess}(V_1, V_2)$ - $V_2$  paths.

As it is highlighted in (Boukhobza et al. 2006, Dion et al. 2003, Van der Woude 1999), these sets play an important role in the analysis of observability property and in the optimization of sensor location for observability and FDI. To find these separators, the maximal set of disjoint paths is firstly calculated. Next, the set of essential vertices is determined. Hence, to find the input (output) separator, it is sufficient to take the essential vertex closer to the source (sink) subset in each disjoint path. Essential vertices are found in  $O(N^3\sqrt{M})$ . Finding the vertices on each path, which are also essential vertices, is done in  $O(N \log_2(N))$ , hence the overall asymptotic complexity order is  $O(N^3\sqrt{M})$ .

-  **maximal matching between two vertex subsets:**

The algorithm determines the size of the maximal matching between two selected vertex subsets (source and sink). A matching is a set of disjoint edges in a bipartite graph. For this algorithm, the equivalence between the maximum flow and the maximal matching in a bipartite graph is considered (Murota 1987). To this aim, an algorithm dedicated to the construction of a bipartite flow graph from a directed graph is used. Once the bipartite flow graph is built, the maximum flow algorithm is run over this bipartite flow graph. The conversion of digraph into bipartite flow graph has a complexity equal to  $O(M + N)$  for linear case and  $O(M + MN)$  for bilinear case. The complexity order of the computation of the maximum flow is  $O(N^2\sqrt{M})$ , the overall complexity order is  $O(N^2\sqrt{M})$ .

In the program's main window, to select the vertex subsets, which are arguments of the different presented functions, the user must use the command  depending on whether the vertex subset is an input (in green) or an output (in red) to the algorithm. According to the used algorithm, the selection of one or two vertex subsets is necessary.

### 3.3 Algorithms for properties of structured system

Generic properties of structured systems related to control problems are considered in this section. Based on the works of (Boukhobza et al. 2006), we recall, in a first time, the graphic conditions for the generic input and state observability of structured linear systems. Next, according to the works of (Boukhobza and Hamelin 2006a), observability of structured bilinear

systems is treated. Finally, FPRG (Fundamental Problem of Residual Generation) of structured linear systems is introduced. We will briefly discuss algorithms for checking these properties.

**3.3.1. Generic input and state observability of structured linear systems** In this paragraph, the conditions for the input and state observability of structured linear systems are given and they are translated to an algorithm-based language.

Input and state observability of structured linear system  $\Sigma_{\Lambda}^l$  requires the definition of the following subsets:

- $\mathcal{X}_1$  is defined as all vertices  $\mathbf{x}_i$  of  $\mathcal{X}$  such that the number of disjoint paths from  $\{\mathbf{x}_i\} \cup \mathcal{W} \cup \mathcal{F}$  to  $\mathcal{Y}$  is greater than the maximal number of disjoint edges from  $\mathcal{W} \cup \mathcal{F}$  to  $\mathcal{Y}$ . According to this,  $\mathcal{X}_1$  is determined with function **disjoint paths** presented in section 3.2.
- $\mathcal{X}_s$  contains all vertices in  $\mathcal{X}$  of all output separators between  $\mathcal{F} \cup \mathcal{W}$  and  $\mathcal{Y}$  calculated with function **separators** and  $\mathcal{X}_0$  is the vertex subset defined by  $\mathcal{X} \setminus (\mathcal{X}_1 \cup \mathcal{X}_s)$ .
- $\Omega_0$  contains all vertices  $\mathbf{v}_i$  of  $\mathcal{F} \cup \mathcal{W}$  such that the maximal matching from  $\mathbf{v}_i$  to  $\mathcal{X} \setminus (\mathcal{X}_s \cup \mathcal{X}_0)$  equals zero. This subset is calculated by means of the **maximal matching** function of 3.2.
- $\mathcal{Y}_0$  consists in all essential vertices between  $\mathcal{W} \cup \mathcal{F} \cup \mathcal{Y}$  and  $\mathcal{Y}$ .  $\mathcal{Y}_0$  is then determined with **essential vertices** function of 3.2.

The different vertex subsets are computed using respectively the commands  $\mathbf{X}_0$ ,  $\mathbf{X}_1$ ,  $\mathbf{X}_s$ ,  $\mathbf{\Omega}_0$ , and  $\mathbf{Y}_0$ . In addition, LISA derive the vertex subsets  $\Omega_1$  and  $\mathcal{Y}_1$  with commands  $\mathbf{\Omega}_1$  and  $\mathbf{Y}_1$  respectively.  $\Omega_1$  is defined as all vertices in  $\mathcal{W} \cup \mathcal{F} \setminus \Omega_0$  and  $\mathcal{Y}_1$  as all vertices in  $\mathcal{Y} \setminus \mathcal{Y}_0$ .

According to these definitions, input and state observability of structured linear systems can be verified if next three conditions are satisfied (Boukhobza et al. 2006):

**Cond1:** each vertex in  $\mathcal{X} \cup \mathcal{F} \cup \mathcal{W}$  is predecessor of vertex set  $\mathcal{Y}$ .

**Cond2:** all vertices in  $\mathcal{X}_0$  and  $\Omega_0$  are essential for the linking between  $\Omega_0$  and  $\mathcal{X}_s \cup \mathcal{Y}_0$ .

**Cond3:** the maximal matching in the bipartite graph  $\mathcal{X} \cup \mathcal{F} \cup \mathcal{W} \rightarrow \mathcal{X} \cup \mathcal{Y}$  is equal to  $n + q + r$ .

Command  **Observable** checks conditions **Cond1**, **Cond2** and **Cond3** as follows:

To verify condition **Cond1**, algorithm **predecessors** is used and then all vertices of  $\mathcal{X}$ ,  $\mathcal{F}$  and  $\mathcal{W}$  are searched in the result list of predecessors of  $\mathcal{Y}_0$ .

According to the complexity orders associated to the basic graph functions, it results that the complexity order on calculating **Cond1** is  $O(M + N \log_2(N))$ .

The two main steps for the verification of **Cond2** are recapitulated as follows:

- (1) Essential vertices between  $\Omega_0$  and  $\mathcal{X}_s \cup \mathcal{Y}_0$  are calculated,
- (2) If all vertices in  $\mathcal{X}_s$  and  $\Omega_0$  are in the previously calculated set, then **Cond2** is satisfied.

According to the complexity orders associated to the basic graph functions, it results that the complexity order on calculating **Cond2** is  $O(N^3 \sqrt{M})$ .

To check condition **Cond3**, it is sufficient to compute the maximal matching between  $\mathcal{X} \cup \mathcal{F} \cup \mathcal{W}$  and  $\mathcal{X} \cup \mathcal{Y}$ . That is, the complexity order of **Cond3** is  $O(N^2 \sqrt{M})$ .

**3.3.2. Observability for structured bilinear systems** Graphic conditions for the observability of structured bilinear systems can be found in (Boukhobza and Hamelin 2006a). We can summarize these conditions as follows:

**CondA:** Each vertex in  $\mathcal{X}$  is a predecessor of vertex set  $\mathcal{Y}$ ,

**CondB:** The maximal matching in the bipartite graph  $\mathcal{X} \rightarrow \mathcal{X}' \cup \mathcal{Y}$  is equal to  $n$ .

For condition **CondA**, function **predecessors** implemented in section 3.2 can be used.

To check condition **CondB**, it is sufficient to compute the maximal matching between  $\mathcal{X}$  and  $\mathcal{X}' \cup \mathcal{Y}$ . Thus, the complexity order of **CondB** is  $O(NM^{3/2})$ .

**3.3.3. Fault Isolability of structured linear systems**

An important property in diagnosis context is the fault isolability. LISA allows us to verify this property for the multiple fault case as well as for the single fault case. Conditions for the FPRG solvability are treated in detail in (Commault et al. 1999, Commault et al. 2002).

Only one condition has to be verified for ensuring fault isolability in a multi-fault context:

**Cond.I** : a subset of faults  $\mathcal{F}_0$ , belonging to  $\mathcal{F}$  set, is isolable if the number of disjoint paths between  $\mathcal{F} \cup \mathcal{W}$  and  $\mathcal{Y}$  is equal to the number of disjoint paths between  $\mathcal{F} \setminus (\mathcal{F}_0 \cup \mathcal{W})$  and  $\mathcal{Y}$  and the cardinality of  $\mathcal{F}_0$ .

In the single fault context, the following condition must be satisfied to ensure the fault isolability:

**Cond.II** : a subset of faults  $\mathcal{F}_0$  belonging to  $\mathcal{F}$  set, is isolable if,  $\forall f_i, f_j$  included in  $\mathcal{F}_0$ , the number of disjoint paths between  $\{f_i, f_j\} \cup \mathcal{W} \cup (\mathcal{F} \setminus \mathcal{F}_0)$  and  $\mathcal{Y}$  is greater than the number of disjoint paths between  $\{f_j\} \cup \mathcal{W} \cup (\mathcal{F} \setminus \mathcal{F}_0)$  and  $\mathcal{Y}$ .

Command  **Isolability** checks condition **Cond.I** and **Cond.II** using the function **disjoint paths** presented in section 3.2. Thus, here also, the overall complexity order is  $O(N^2 \sqrt{M})$ .

Some other options are available as the computation of isolable fault components using command 

#### 4. CONCLUSION

LISA is a program used to display and manipulate linear/bilinear control systems. The systems are displayed in terms of graphs, which can be manipulated by the user. Important concepts, such as observability, fault isolability, etc. have been translated into graph theoretic terms, and were implemented with different graph algorithms.

Since many problems that needed to be solved here are rather expensive in terms of time complexity (i.e. finding disjoint paths), it was crucial to use a programming language with little overhead, close to the machine code level, C++ was chosen to benefit from a much great flexibility (e.g. classes, templates), with only little sacrifice of performance. For the graphical user interface (gui), the library QT 4.0 was chosen, since it is very flexible, portable and easy to use for creating graphical user interfaces.

An exhaustive description of the algorithms developed in LISA was done in this report. Some other properties can be studied using these graphic algorithms (Dion et al. 2003, Van der Woude 1999, Van der Woude and Murota 1995). Hence, LISA program is intended to expand its capabilities to analyze other structural properties as controllability, disturbance rejection, etc. Moreover, some specific properties related to Network Controlled Systems are currently in development. Algorithms to check the autonomy of distributed FDI nodes will be soon added as well as algorithms for sensor placement optimization.

#### REFERENCES

Bang-Jensen, J. and G. Gutin (2002). *DIGRAPHS Theory, Algorithms and Application*, Springer-Verlag, London, England.

Blanke, M. and T. Lorentzen (2006). 'SaTool - A software Tool for Structural Analysis of Complex Automation Systems', In: *6th IFAC SAFEPROCESS' 2006*, Beijing, P. R. China.

Boost C++ Libraries (2005): Version 1.0 'http://www.boost.org'.

Boukhobza T. and F. Hamelin (2007a). 'Observability analysis for structured bilinear systems: a graph theoretic approach', *Automatica*, provisionally accepted.

Boukhobza, T., F. Hamelin, and S. Martinez-Martinez (2006). 'State and input observability analysis for structured linear systems: a graph theoretic approach', *Automatica*, to appear in 2007.

Commault, C., J.M. Dion, O. Sename, and J.C. Avila Vilchis (1999). 'Fault detection and isolation: a graph approach', *Proc. of the 5th European Control Conference*, Karlsruhe, Germany.

Commault, C., J.M. Dion, O. Sename, and R. Motyeian (2002). 'Observed-based fault detection and isolation for structured systems', *IEEE Transactions on Automatic Control*, **AC-47**, 2074–2079.

Dion, J.M., C. Commault and J. Van der Woude (2003). 'Generic properties and control of linear structured systems: A survey', *Automatica*, vol.39 no.7, 1125–1144.

Hovelaque, V., C. Commault and J.M. Dion (1996). 'Analysis of linear structured systems using a primal-dual algorithm', *Systems and Control Letters*, **27**, 73-85.

Hovelaque, V., N. Djidi, C. Commault and J.M. Dion (1997). 'Decoupling Problem for Structured Systems Via a Primal Dual Algorithm', *Proc. of the 4th European Control Conference*, Brussel, Belgium.

Lin, C. T. (1974). 'Structural controllability', *IEEE Transactions on Automatic Control*, **AC-19**, no. 3, 201–208.

Murota, K. (1987). *Systems Analysis by Graphs and Matroids, Structural Stability and Controllability*, Springer-Verlag, New York, U.S.A.

Reinschke, K. J. (1988). *Multivariable Control. A Graph-Theoretic Approach.*, Springer-Verlag, New York, U.S.A.

Shields, R. W. and J. B. Pearson (1976). 'Structural controllability of multi-input linear systems', *IEEE Transactions on Automatic Control*, **AC-21**, no.2, 203–212.

Van der Woude, J. W. (1999). 'The generic number of invariant zeros of a structured linear system', *SIAM J. Control Optim.*, vol.38, no.1, 1–21.

Van der Woude, J. W. and K. Murota (1995). 'Disturbance decoupling with pole placement for structured systems: A graph-theoretic approach', *SIAM Journal on Matrix Analysis and Applications*, vol.16, no.3, 922–942.