Finding Circular Relationships in Networks

Dr. Kenneth Fordyce
Quantitative Methods and Decision Support
Department 33VA, Mail Station 284
Engineering / Scientific Computing Center
International Business Machines Corporation
Kingston, New York 12401 USA
914-385-5944

Dr. Jan Jantzen
Technical University of Denmark
Electric Power Engineering Department
DK-2800 Lyngby, Denmark

Gerald Sullivan
Senior Engineer
Manager, Advanced Industrial Engineering
and LMS Development
Department 746, Building 965-3
International Business Machines Corporation
Burlington Semiconductor Manufacturing
Essex Junction, Vermont 05452 USA

Abstract

A critical computational requirement for many of the decision technologies in the fields of MS/OR, AI/KBS, and DSS is the development and manipulation of a network describing the relationship between "actors" involved in the application of the decision technology to a specific problem. The manipulation of such networks using Boolean arrays and functions is well known in the APL community (see bibliography). Often in such networks it is important to identify circular conditions as a preprocessing step, but the techniques to accomplish often yield incomplete information. This paper describes a simple and efficient method to find all circular conditions as a preprocessing step.

This paper is a subset of a longer paper (Fordyce, Jantzen, and Sullivan, 1990) which describes how we can fully build and manipulate a function network with Boolean arrays including focusing networks, finding circular conditions, and grouping functions based on relative independence to identify parallel computational opportunities and substantially reduces the non-procedural aspect of the problem.

Introduction

A critical computational requirement for many of the decision technologies in the fields of MS/OR (PERT/CPM, Markov chains, decision trees, Baysien analysis, MRP, simulation, ...), AI/KBS (evidential reasoning, truth maintenance systems, propositional logic, rule based inference, frames and semantic nets, ...), and DSS (worksheet or financial planning models, data / entity models, ...) is the development and manipulation of a function network describing the relationship between "variables," "objects," or "actors" involved in the application of the decision technology to a specific problem.

A critical problem in many network problems is identifying groups of variables or functions that have a circular relationship. That is: A depends on B, B depends on C, C depends on A, therefore A depends on A, etc. An example using function notation would be:

```
V1 = f(V2, V3)
V2 = g(V1, V3)
V3 = h(V1, V2)
```

An example of such a condition in manufacturing would be: The machine a lot is assigned to depends on the estimate of how far ahead or behind (delta) of schedule the lot is. The delta schedule estimate depends on the machine the lot is assigned to.

In systems of algebraic equations circular conditions are a simultaneous set of equations. An example of one that often occurs in financial planning models is:

```
PROFIT = REVENUE - (EXPENSE + BONUS)
BONUS  = .05 × PROFIT
```

In this case REVENUE and EXPENSE are input variables, and PROFIT and BONUS are calculated and circular.

This paper describes a simple method to find such relationships. All code is in APL2.

Example Problem

Throughout the rest of the paper we will use the following example to demonstrate how to find circular relationships. This example has nine relationships and three circular relationships:

```
1. AA = FN1 (AA,A)
   Read the variable AA is a function of the
   variables AA and A through the function FN1.
   Therefore AA depends on AA and A.
2. BB = FN2 (AA)
3. C = FN3 (BB,AA)
4. D = FN4 (E,AA)
5. E = FN5 (D,BB)
6. F = FN6 (E,D)
7. G = FN7 (H,F,AA)
8. H = FN8 (I)
9. I = FN8 (G,BB)
```

Generating The Base Boolean Matrices

The first items we need to generate are two Boolean matrices called *INMATIP* (*IP* is for *INPUT*) and *INMATOP* (*OP* is for output).

INMATIP records which variables are input variables for which functions. *INMATIP* has one row for each variable, and one column for each function. A cell gets a 1 if the column variable is in the "input portion" of a function, else a 0. For our example *INMATIP* is:

INMATIP MATRIX

	FN1	FN2	FN3	FN4	FN5	FN6	FN7	FN8	FN9
AA	1	1	1	1	0	0	1	0	0
A	1	0	0	0	0	0	0	0	0
BB	0	0	1	0	1	0	0	0	1
E	0	0	0	1	0	1	0	0	0
D	0	0	0	0	1	1	0	0	0
H	0	0	0	0	0	0	1	0	0
F	0	0	0	0	0	0	1	0	0
I	0	0	0	0	0	0	0	1	0
G	0	0	0	0	0	0	0	0	1
C	0	0	0	0	0	0	0	0	0

INMATOP records which variables are output variables for which functions. *INMATOP* has one row for each variable, and one column for each function. A cell gets a 1 if the variable is in the "output portion" of a function, else a 0. For our example *INMATOP* would be:

INMATOP MATRIX

	FN1	FN2	FN3	FN4	FN5	FN6	FN7	FN8	FN9
AA	1	0	0	0	0	0	0	0	0
A	0	0	0	0	0	0	0	0	0
BB	0	1	0	0	0	0	0	0	0
E	0	0	0	0	1	0	0	0	0
D	0	0	0	1	0	0	0	0	0
H	0	0	0	0	0	0	0	1	0
F	0	0	0	0	0	1	0	0	0
I	0	0	0	0	0	0	0	0	1
G	0	0	0	0	0	0	1	0	0
C	0	0	1	0	0	0	0	0	0

Building such matrices by parsing the notation like "AA = FN1 (AA,A)" is a well-known process in APL.

Generating All Variable Links

The function *VAR_ALL_LINK* (shown on page 4) will find all dependencies between variables by manipulating *INMATIP* and *INMATOP*. A cell gets a 1 if the variable associated with the column is directly or indirectly an input to the variable associated with the row. For example *C* has indirect dependency on the variable *A* (*C* depends on *AA*, *AA* depends on *A*). Therefore the cell (*C*,*A*) has a value 1. Again this is a well-known procedure in APL. The syntax is:

```
LEVEL_ALL_VAR_LINKS←INMATIP VAR_ALL_LINK INMATOP
```

For our example this matrix is:

LEVEL_ALL_VAR_LINKS MATRIX

		INPUT VARIABLES									
		AA	A	BB	E	D	H	F	I	G	C
O	AA	1	1	0	0	0	0	0	0	0	0
U	A	0	0	0	0	0	0	0	0	0	0
T	BB	1	1	0	0	0	0	0	0	0	0
P	E	1	1	1	1	1	0	0	0	0	0
U	D	1	1	1	1	1	0	0	0	0	0
T	H	1	1	1	1	1	1	1	1	1	0
	F	1	1	1	1	1	0	0	0	0	0
V	I	1	1	1	1	1	1	1	1	1	0
A	G	1	1	1	1	1	1	1	1	1	0
R	C	1	1	1	0	0	0	0	0	0	0

Finding Circular Relationships

Step 1, If a variable is a circular variable (*CV*) then it has a dependency on itself. If this dependency situation exists, the upper left to lower right diagonal element of *LEVEL_ALL_VAR_LINKS* corresponding to that variable has a 1; else a 0. In our example this diagonal is

LEVEL_ALL_VAR_LINKS DIAGONAL

		INPUT VARIABLES									
		AA	A	BB	E	D	H	F	I	G	C
O	AA	1									
U	A		0								
T	BB			0							
P	E				1						
U	D					1					
T	H						1				
	F							0			
V	I								1		
A	G									1	
R	C										0

Therefore the circular variables are: *AA E D H I G*. The following APL2 expression will get this for you:

```
VARLIST←'AA' 'A' 'BB' 'E' 'D' 'H' 'F' 'I' 'G' 'C'
CV ← ((1 1) ⍉ LEVEL_ALL_VAR_LINKS) / VARLIST
```

Step 2, now we need a method to organize these circular variables into the groups based on the variables that "circulate" together.

We generated a reduced version of *LEVEL_ALL_VAR_LINKS* that has only the rows and columns of the circular variables. For Example 2 we get:

		INPUT VARIABLES					
		AA	E	D	H	I	G
O	AA	1	0	0	0	0	0
U	E	1	1	1	0	0	0
T	D	1	1	1	0	0	0
P	H	1	1	1	1	1	1
U	I	1	1	1	1	1	1
T	G	1	1	1	1	1	1

Variables with the same row pattern of one's and zero's "circulate" on each other. Therefore AA is a group by itself, E and D form a second group, and H, I, and G form the third group.

The APL2 function *CIRCULAR* (listed on page 4) finds all circular groups. The syntax is:

```
CIRCUL_LIST_VAR←VARLIST CIRCULAR LEVEL_ALL_VAR_LINKS
```

Conclusion

In this paper we have presented a clean and fast method for finding circular relations in networks. The algorithms presented here are a direct result of APL's array data structures and Boolean functions. APL has a long history of using Boolean arrays to make difficult problems easy.

Bibliography

[1] Alfonseca, M. and Brown, J. 1987, "Parallel Solutions to Logic Problems" SEAS Spring Meeting 1987, *SEAS Proceedings*, Vol. I, p. 27-46; and reprinted in *APL-CAM Journal*, Vol. 9:4, p. 764-778, Oct. 1987.

[2] Brown, J., Pakin, S., and Polivka, R. 1988, *APL2 at a Glance*, Prentice Hall, Englewood New Jersey.

[3] Eusebi, E. 1987, "Inductive Reasoning from Relations," APL87 Conference Proceedings, *APL Quote Quad*, Vol. 17, No. 4, pp. 386-390.

[4] Fordyce, K. Morreale, M., McGrew, J. and Sullivan, G. (1991), "APL Techniques in Knowledge Based Systems," IBM, 33VA/284, Kingston, NY 12401 forthcoming in *Encyclopedia of Computer Science and Technology*, edited by Allen Kent and James William from the University of Pittsburgh.

[5] Fordyce, K. Morreale, M., and McGrew, J. (1990), "An Overview of APL2 for Knowledge Based Systems," IBM Technical Report: 21-1383, IBM, 33VA/284, Kingston, NY 12401

[6] Fordyce, K., Jantzen, J., and Sullivan, G. Sr.,and Sullivan, G. Jr. 1990, "Using Boolean Arrays to Build and Analyze Function Networks" IBM, 33VA/284, Kingston, NY 12401

[7] Fordyce, K., Jantzen, J., and Sullivan, G. Sr.,and Sullivan, G. Jr. 1989, "Representing Knowledge with Functions and Boolean Arrays " *IBM Journal of Research and Development* Vol. 33, No. 6, pp. 627-646.

[8] Fordyce, K., and Sullivan, G. 1988A, "Boolean Array Based Inference Engines," IBM, 34EA/284, Kingston, NY 12401. Proceedings of the APL and Expert Systems Conference, forthcoming in ACM publication.

[9] Fordyce, K. and Sullivan, G. 1987, "Boolean Array Structures for a Rule Based Forward Chaining Inference Engine," *APL Quote Quad*, APL87 Conference Proceedings, Vol. 17, No. 4, pp. 185-195

[10] Franksen, O., Falster, P., and Evans, F. 1979, "Qualitative Aspects of Large Scale Systems - Developing Design Rules Using APL," *Lecture Notes in Control and Information Sciences (Vol. 17)*, Monograph, Springer-Verlag.

[11] Franksen, O. 1979, "Group Representation of Finite Polyvalent Logic- A case Study Using APL Notation," in *A Link Between Science and Application of Automatic Control, Proceedings IFAC World Congress 1978*, edited by A. Niemi, Pergamon Press, New York, Vol 2., pp. 875-887. Monograph, Springer-Verlag.

[12] Franksen, O. 1984A, "Are Data Structures Geometrical Objects? Part 1: Invoking the Erlanger Program," *Syst. Anal. Model. Simul.*, Vol. 1, No. 2, pp. 113-130.

[13] Franksen, O. 1984B, "Are Data Structures Geometrical Objects? Part 2: Invariant Forms in APL and Beyond," *Syst. Anal. Model. Simul.*, Vol. 1, No. 2, pp. 131-150.

[14] Franksen, O. 1984C, "Are Data Structures Geometrical Objects? Part 3: Appendix A: Linear Differential Operators," *Syst. Anal. Model. Simul.*, Vol. 1, No. 3, pp. 249-258.

[15] Franksen, O. 1984D, "Are Data Structures Geometrical Objects? Part 4: Appendix B: Logic Invariants by Finite Truthtables," *Syst. Anal. Model. Simul.*, Vol. 1, No. 4, pp. 339-350.

[16] Franksen, O. 1985, *Mr. Babbage's Secret: The Tale of a Cypher—and APL*, Prentice Hall, New Jersey.

[17] Iverson, K. 1980, "1979 ACM Turing Award Lecture: Notation as a Tool for Thought," *Communications of the ACM*, Vol. 23, No. 8, pp. 444-465.

[18] Jantzen, J. 1989, "Inference Planning Using Digraphs and Boolean Arrays," Technical University of Denmark, Electric Poer Engineering Department, DK-2800, Lyngby, DENMARK *APL Quote Quad*, APL89 Conference Proceedings, Vol. 19, No. 4. pp. 200-204.

[19] Moller, G. 1986, "A Logic Programming Tool for Qualitative System Design," *APL Quote Quad*, APL86 Conference Proceedings, Vol. 16, No. 4, pp. 266-71.

[20] Tarjan, R. 1974, "Testing Flow Graph Reducibility," *Journal of Computer and Systems Sciences*, Vol. 9, No. 3, December 1974.

[21] Thomson, N. 1989A, *APL Programs for the Mathematical Classroom* Springer Verlag, New York, ISBN 0-387-97002-9.

[22] Thomson, N. 1989B, "APL2 and Basic Operation Research Algorithms," *Proceedings of Share European Association (SEAS) Fall 1989 Meeting*, Vol. 2, pp. 1689-1703 IBM UK Labs, UK Development and Manufacturing Process Centre, Mail Point 188, Hursley House, Hursley Park, Winchester, Hampshire S021 21N, England ■

Functions for Finding Circular Relationships in Networks

```
     ∇
[0]    VARLINKS←INMATIP VAR_ALL_LINKS INMATOP;JK
[1]   ⍝ This finds all linkages between variables
[2]   ⍝ Rows are output variables
[3]   ⍝ Columns are input variables
[4]    VARLINKS←INMATOP∨.∧⍉INMATIP
[5]   L10:
[6]    JK←VARLINKS
[7]    VARLINKS←VARLINKS∨(VARLINKS∨.∧VARLINKS)
[8]    →(~JK≡VARLINKS)/L10
     ∇

     ∇
[0]    CIR_GROUP←LIST CIRCULAR ALL_LINK;CIR_ID;CIR_LIST;ORDER
[1]   ⍝
[2]   ⍝ Finding the circular or SIMO variables or functions
[3]   ⍝
[4]   ⍝ LIST is fnlist or varlist
[5]   ⍝
[6]   ⍝ALL_LINK is LEVEL_ALL_VAR_LINK or LEVEL_FN_VAR_LINK
[7]   ⍝
[8]    CIR_ID←(1 1⍉ALL_LINK)/⍳1↑⍴ALL_LINK
[9]    CIR_LIST←LIST[CIR_ID]
[10]  ⍝
[11]  ⍝ Grouping the circular or SIMO variables
[12]   ALL_LINK←ALL_LINK[CIR_ID;CIR_ID]
[13]   ALL_LINK←2⊥⍉ALL_LINK
[14]   ORDER←⍋ALL_LINK
[15]   ALL_LINK←ALL_LINK[ORDER]
[16]   CIR_LIST←CIR_LIST[ORDER]
[17]  ⍝
[18]   CIR_GROUP←ALL_LINK⊂CIR_LIST
     ∇

     ∇
[0]    Z←X ODA Y;JK;JK1;N;I
[1]   ⍝ This function is an alternative to X ∨.∧ Y
[2]    Z←((1↑⍴X),(¯1↑⍴Y))⍴0
[3]   ⍝ Initialize the outcome matrix
[4]   ⍝ This is initialized to all zeroes (0).
[6]   ⍝ It has the same number of rows as X,
[7]   ⍝ and the same number of columns as Y
[8]    JK←⍳¯1↑⍴X
[9]   ⍝ This is a list of the columns in X
[10]  ⍝ If X has six columns then this JK is 1 2 3 4 5 6
[11]   N←1↑⍴Z
[12]  ⍝ Number of rows in Z
[13]   I←1
[14]  ⍝ I is the cycle counter
[15]  L10:
[16]  ⍝ Start of loop which produces Z
[17]   JK1←X[I;]/JK
[18]   →(0=1↑⍴JK1)/L20
[19]   JK1←Y[,JK1;]
[20]   JK1←∨/[1]JK1
[21]   Z[I;]←JK1
[22]  L20:
[23]   →(N≥I←I+1)/L10
[24]  ⍝ Check if completed each row of X, if not branch to L10
[25]   →0
     ∇
```