Go Back

Formalization of Model Partition Heuristics through Graph Theory

Rogelio Oliva
Harvard Business School
Morgan Hall T87
Boston, MA 02163
Ph +1/617-495-5049; Fx +1/617-496-5265
roliva@hbs.edu

Abstract

This paper builds on the argument that calibration of system dynamics models can be used as a testing strategy for dynamic hypotheses. In particular, it addresses the issue of how to partition the model for calibration and testing purposes. Working with small calibration problems reduces the risk of the structure being forced into fitting the data, increases the efficiency of the estimation, and concentrates the differences between observed and simulated behavior in the piece of structure responsible for that behavior. The premise of this paper is that it is possible to focus on the structural complexity of SD models by using the tools and insights from graph theory to design a partition strategy that maximizes the test points between the model and the real-world. After reviewing the graph representation of system structure, the paper presents the rationale and algorithms and for model partitions based on data availability, and block and cycle partitions. The paper concludes by identifying future research avenues in this arena, and conjecturing on other potential applications of the tools developed for the analysis of models structure.

(Analysis of model structure; Graph theory; Hypotheses testing; Model calibration; Parameter estimation; Simulation; System Dynamics)

1. Introduction

The argument for calibration as a testing strategy for a dynamic hypothesis has been articulated elsewhere by this author (Oliva, 2000b). The argument can be summarized as follows: SD interventions can only be as good as the dynamic hypothesis (DH) that is used for policy design, thus careful consideration must go into building confidence in a DH. A DH explicitly posits a causal link between structure and behavior. Although it is impossible to verify a hypothesis, science has refined a systematic approach for increasing the confidence of stated hypothesis and ruling out alternative explanations, viz., experiments designed to falsify the hypothesis. System dynamics models are well suited for this experimental approach since they are logically sound and need to be relevant to the problem situation, i.e., they are empirically testable. Calibration explicitly attempts to link structure and behavior, thus making it a more stringent test than matching either structure or behavior alone. Automated calibration (AC) techniques are capable of generating an optimal fit to historical data from a given structure and set of parameters. However, because of the assumption of correct structure and the effort to match the historical behavior, AC techniques are typically used to confirm the dynamic hypothesis –can the structure match the observed behavior? Thus, we are faced with a paradox: *While model calibration, by requiring simultaneous adherence to observable behavior and structure, constitutes a stringent test of the dynamic hypothesis, automated calibration, the most powerful tool available for calibration, strips the process of its power to perform the test.*

In that paper I argued for three heuristics to increase the power of AC as a testing tool: i) do not override known (observable) structure, ii) tackle small calibration problems, and iii) use AC to test the hypothesis: "Does the estimated parameter match the observable structure of the system?" The paper concluded with a set of increasingly demanding tests to guide the assessment of the AC output in the context of hypothesis testing, i.e., a set of tools to formalize heuristic iii). The paper, however, is vague on how to support heuristic ii) and concludes with an invitation for further research into formal ways of partitioning a model for estimation purposes. This paper tackles this issue by formalizing the heuristics for model partitions and a sequencing strategy for the calibration/testing process. But before getting into model partitions, I would like to re-state the rationale for partial model testing (Homer, 1983) and calibrations over small model partitions.

One of the main benefits of the AC techniques is that it is possible to specify the calibration problem as a single optimization problem with an error function that contains all data available and a set of calibration handles that contain all model parameters (Lyneis and Pugh, 1996). By providing total flexibility to the model structure to adapt to the existing data, such an approach generates the best possible fit to all data available. From an operational perspective, however, having a complex error function and multiple optimization handles makes the tractability of the errors and the diagnosis of mismatches more difficult.

Since not all model parameters affect all output variables in the model, as the number of data series in the error function increases, individual parameters become less significant. Variations in individual parameter values have a small impact in a complex error function, thus resulting in wider confidence intervals for the estimated parameters, i.e., less efficient estimators. Similarly, increasing the number of parameters to be estimated from an error function reduces the degrees of freedom in the estimation problem, thus resulting in wider confidence intervals.

The most serious difficulty with a large number of calibration handles, however, is the increased difficulty to detect formulation errors. In an endeavor to match historical data, the calibration process 'fixes' the model structure to cover formulation errors. Since these corrections are distributed among the parameters being used in the calibration problem, as the number of parameters being estimated increases, the 'correction' to each parameter becomes smaller. Small deviations from 'reasonable' values and wider confidence intervals make it more difficult to detect

fundamental formulation errors, especially when a 'good fit' to historical behavior has been achieved.

Working with small calibration problems reduces the risk of the structure being forced into fitting the data, increases the efficiency of the estimation (estimators with smaller variances), and concentrates the differences between observed and simulated behavior in the piece of structure responsible for that behavior. The goal of the proposed heuristic is to partition the model as finely as possible, thus focusing the analysis and yielding more efficient estimators of the underlying parameters governing the model behavior. Model partition requires an understanding of model structure, the role of individual parameters in determining the model's behavior, and knowledge about the sources of data available.

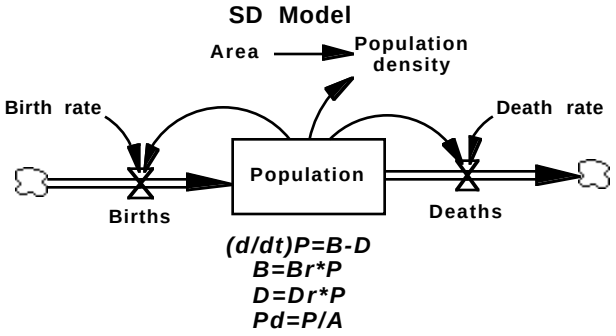
The main challenge when using calibration as testing is to identify how best to use the data available to validate the structural representation of the system. Exploring and understanding the structural complexity of a model permits better utilization of the data and a direct understanding of the role of individual parameters in determining the system's behavior. "Structural complexity" of a SD model refers to the interactions among variables and the feedback loops that those interactions create. Graph theory is concerned with the topology of interconnected sets of nodes, abstracting from their spatial location (embedding) and the exact nature of the links. By adopting the graph representation of a SD model, we can focus on its structural complexity rather than the dynamic complexity that arises from non-linear relations and accumulations. Graph theory has been successfully used to explore the structure of SD models (Burns, 1979; Kampmann, 1996).

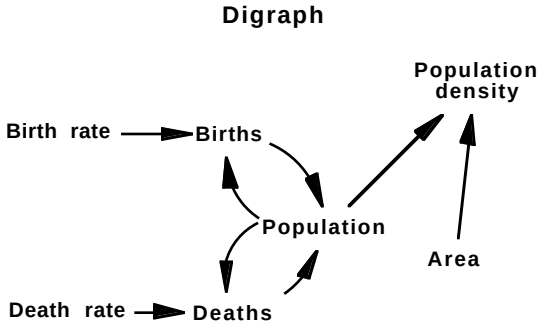
The premise of this paper is that it is possible to focus on the structural complexity of SD models by using the tools and insights from graph theory to design a partition strategy that maximizes the test points between the model and the real-world. The rest of the paper is structured as follows. The following section summarizes the graph representation of system structure and presents some of the nomenclature used in the paper. Section 3 presents the rationale and algorithms and for model partitions based on data availability, and block and cycle partitions. Cycle partitions constitute heavily interconnected pieces of model structure and are at the core of SD models –all feedback loops are captured in cycle partitions. Since cycle partitions can encompass as much as 90% of the model's variables, a more detailed strategy to decompose cycle partitions is presented in section 4. The paper concludes by identifying future research avenues in this arena, and conjecturing on other potential applications of the tools developed for the analysis of models' structure.

2. Graph representation of system structure

A directed graph or digraph G is a pair (V,E), where $V(G)$ is a finite set of elements, called vertices, and $E(G)$ is a binary relation on V —a subset of ordered pairs of elements of $V(G)$. The elements of $E(G)$ are called edges and constitute the edge set of G. The structure of a system dynamics model can be represented as a digraph, where the variables are the vertices and the edges are the relationship "is used in," i.e., there is a directed edge $(u \rightarrow v)$ if u is used as an argument in the computation of v. To facilitate computations, a digraph is often represented as an adjacency matrix.[1] The adjacency matrix representation of a digraph is a square matrix $\mathbf{A}$ of size $|V|$, where each row (and column) represents a vertex (vertices are numbered 1,2, … $|V|$). The entries in the matrix are restricted to zero and one, where $a_{uv}=1$ IFF $(u,v) \in E(G)$. The ones in row $\mathbf{A}_u$ represent the successor set ($Succ[u]$) for vertex u, while the ones in column $\mathbf{A}_u$ represent the predecessor set ($Pred[u]$) for vertex u. Figures 1a-1c illustrate the mapping of model structure into a digraph and its corresponding adjacency matrix representation. A tool to generate the adjacency matrix from a text file containing the model documentation has been developed and is available online.[2]

Figure 1. Graph representation of system structure





Adjacency matrix

	Br	*B*	*P*	*Dr*	*D*	*Pd*	*A*
Br	0	1	0	0	0	0	0
B	0	0	1	0	0	0	0
P	0	1	0	0	1	1	0
Dr	0	0	0	0	1	0	0
D	0	0	1	0	0	0	0
Pd	0	0	0	0	0	0	0
A	0	0	0	0	0	1	0

Adjacency matrix block ordered by levels

		Pd	*A*	*P*	*B*	*D*	*Br*	*Dr*
L1	*Pd*	0	0	0	0	0	0	0
	A	1	0	0	0	0	0	0
L2	*P*	1	0	0	1	1	0	0
	B	0	0	1	0	0	0	0
	D	0	0	1	0	0	0	0
L3	*Br*	0	0	0	1	0	0	0
	Dr	0	0	0	0	1	0	0

Once the model structure is captured in an adjacency matrix, it is possible to use recursive algorithms to analyze and visualize model structure. For instance, another useful representation of model structure is the reachability matrix. The reachability matrix represents a digraph whose edge set is defined by the relation "is antecedent to" and it is equivalent to the transitive closure of **A** ($\mathbf{R}_{ii}=1$; $\mathbf{R}_{ij}=\mathbf{A}_{ij}$; $\mathbf{R}_{ij} \cap \mathbf{R}_{jk} \Rightarrow \mathbf{R}_{ik}$). The reachability matrix is reflexive, i.e., its main diagonal is filled with ones, thus making each vertex a member of its own predecessor and successor sets. The reachability matrix can easily be obtained by adding the identity matrix to the adjacency matrix ($\mathbf{B}=\mathbf{A}+\mathbf{I}$) and raising the sum to some Boolean power k such that $\mathbf{B}^{k-1} \neq \mathbf{B}^{k} = \mathbf{B}^{k+1} = \mathbf{R}$ (Warfield, 1989).[3]

3. Model partitions

This section describes tools and techniques to facilitate the analysis of the model structure and follow the heuristic for model calibration in the smallest possible equation set. In particular, we will explore different algorithms to partition a model and identify structural clusters that can be independently calibrated. The algorithms described below have been coded in a popular computer environment and are available online (Oliva, 2000a).

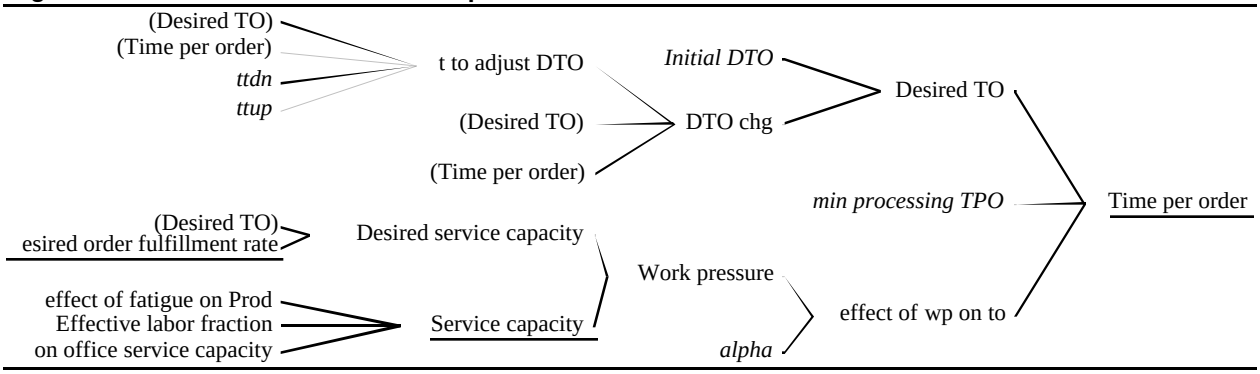
3.1 Data-availability partition

Given an available set of data series, it is possible to determine the minimum equation set that can be used for estimation purposes. In other words, it is possible to systematically identify the set of equations and parameters that are directly involved in the computation of an outcome variable from a set of known inputs. The minimum equation set will guarantee that all parameter 'handles' used in the estimation are involved in the generation of the selected model outcome, hence, being the best use of the data possible.

Let us first illustrate the process. Assume that time series data are available for a set of variables represented in a model. Considering one of these variables as an outcome variable (independent variable in the traditional regression model), it is possible to use the model to explore the causal structure behind that variable, and obtain an expanding tree of causal factors (see figure 2). The

tree branches as each input required for calculating a variable is in turn expanded into its required components. The branch ends whenever an element is either a parameter (i.e., it has no predecessors), or an element is already identified in the tree structure (i.e., its predecessors have already been identified).

Figure 2. Causal tree for Time per order



It is now possible to isolate the causal tree from the rest of the model structure by clipping the tree at the variables where data are available, i.e., the computational predecessors for that variable is not needed since it is possible to use data to substitute for those computations. In the example of figure 2, by using *Service capacity* and *Desired order fulfillment rate* as known data sources, the causal tree for *Time per order* is isolated. The resulting structure can be easily translated into a calibration problem that allows for the estimation of up to 5 parameters (italics in figure 2) using 7 equations.[4] Not all parameters need to be identified as handles in the calibration problem; if there is certainty in the value of a parameter, it is best to leave it as part of the system structure and do the search with other handles. Table 1 lists the calibration problem by substituting the variable names with model equations. Note that, since the calibration problem uses the data for the outcome variables only in the objective function, the calibration problem is well specified, even if the dependent variable is fed back into the equation set (e.g., *Time per Order* are used in the computation of *DTO chg)*. Note also that the calibration problem can be solved if feedback loops are contained in the equation set (e.g., *Desired DTO, t to adjust DTO, DTO chg*).

Table 1. Calibration problem

Minimize

$\sum$(Time per order(t) - **Time per order(t)**)^2 for {t | **Time per Order**(t)= *value*}

Over:

Initial DTO > 0; alpha < 0; ttup > 0; ttdn > 0

Subject to:

Time per order = max(Desired TO * effect of wp on to, min processing TPO)
Desired TO = INTEG (DTO chg, Initial DTO)
DTO chg = (Time per order-Desired TO)/t to adjust DTO
t to adjust DTO = IF THEN ELSE(Time per order>Desired TO , ttup , ttdn)
Desired service capacity = **Desired order fulfillment rate** * Desired TO
Work pressure = (Desired service capacity - **Service capacity**)/Desired service capacity
effect of wp on to = EXP(alpha * Work pressure)
min processing TPO = 0.6

Bold variable names represent the historical time series for the variable.

For clarity the time subindex has been eliminated from the constraint equations.

The back-tracing of outcome variables can also be used to identify data requirements for estimation purposes. If an equation set expands too far back, it is possible to identify from the tree structure a

variable that, if data were available, could reduce the equation set and yield a more precise estimation of intermediate parameters.

The generation of the minimum equation set for each data series available can be automated using a breadth-first search to explore the tree structure and clipping the search whenever an element is found for which data are available. Figure 3 shows the pseudo-code for an algorithm that takes as input the model's adjacency-matrix **A** and a data-availability vector **d** —a vector containing the list of variables for which data are available. For each data series available, the algorithm generates the calibration problem that uses the minimum set of equations possible. The calibration problem is defined by a dependent variable (y), a list of independent variables or known inputs (**x**), the set of equations required for the optimization problem, and the list of parameters (β) that could be estimated from it.

Figure 3. Pseudo-code for data-availability partition - Based on Cormen et al. (1990)

```
[y,x,β,eq] ← BFS(A,d)
for each vertex i ∈ V[A]
    if Pred_A[i]=∅
        p ← p ∪ i

m ← 0
for each vertex i ∈ d
    m ← ++
    y(m) ← i
    for each vertex j ∈ V[A]-i
        color[j] ← white
    Q ← i
    while Q ≠ ∅
        j ← head[Q]
        for each vertex k ∈ Pred_A[j]
            if color[k] = white
                color[k] ← gray
                if k ∈ d
                    x(m) ← x(m) ∪ k
                else
                    if k ∈ p
                        β(m) ← β(m) ∪ k
                    else
                        eq(m) ← eq(m) ∪ k
                        Enqueue (Q,k)
        Dequeue (Q)
        color[j] ← black
    end
end
```

```
for all vertex in model
    if no predecessors
        id as system parameters

initialize problem counter
for each vertex with available data
    increment problem counter

    for all other vertex
        mark as not discovered
    initialize queue with data vertex
    while elements in queue
        take first element from queue
        for each predecessor
            if not discovered
                mark as discovered
                if data available
                    enter as known input
                else
                    if parameter
                        enter as parameter
                    otherwise
                        enter as equation
                        enter k in queue
        drop (j) from queue
        mark as explored
    end
end
```

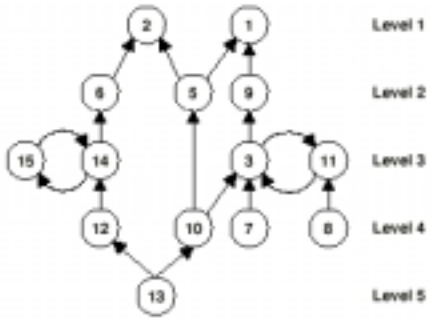
In the unlikely event that all model parameters are reachable from the existing data available, only the data-availability partition would be needed to articulate the required calibration problems. In most situations, however, it will be necessary to develop a sequence of partial model calibrations so that insights and outputs from early calibrations may be used to tune further calibrations. Confidence in DH normally built by gradually moving form simple and observable pieces of structure to more complex and strongly connected components of the system. Ideally, the same approach should be used to develop a sequence of calibration problems that yields increasing confidence in the model structure and enables the use of insights and findings from simple structures in testing complex components of the structure. In order to do that we need a map of the system structure to identify its strongly connected components and how parameters are used within the causal hierarchy of the model. Levels and cycles are structural partitions that can facilitate the navigation of the model structure and the sequencing of partial model calibrations.

3.2 Block partition

First, it is possible to obtain the hierarchy of the causal structure by partitioning the reachability matrix into blocks of vertices at the same level in the model's causal structure. For this kind of analysis all the vertices in a feedback loop are considered to be at the same level since it is not possible to specify causal precedence between any two vertices in a loop. The members of the first block (also called level) are those that do not have any successors outside of its predecessor set[5] —normally the outcome variables of a model. Successive blocks are identified by eliminating from the reachability matrix the top level, and looking again for vertices without successors outside of the predecessor set. Figure 4 shows the pseudo-code for an algorithm to partition a digraph by level blocks and an illustration of a simple digraph mapped by level. The algorithm generates an array with the list of vertices that correspond to each level in each cell.

Figure 4. Pseudo-code for level partition - Based on Warfield (1989)

```
Lev ← Levels(R)                                  initialize level counter
k ← 1                                            while vertex left
while V[R] ≠ ∅                                       for each vertex remaining
    for each vertex u ∈ V[R]                             if vertex is at top level, i.e. all successors are in predecessor set
        do if Pred_A[u] ∩ Succ_A[u] = Succ_A[u]              introduce vertex to level
            Lev[k] ← Lev[k] ∪ u                          remove level from graph
    V[R] ← V[R] - Lev[k]                             increment level counter
    k ← k++                                      end
end
```

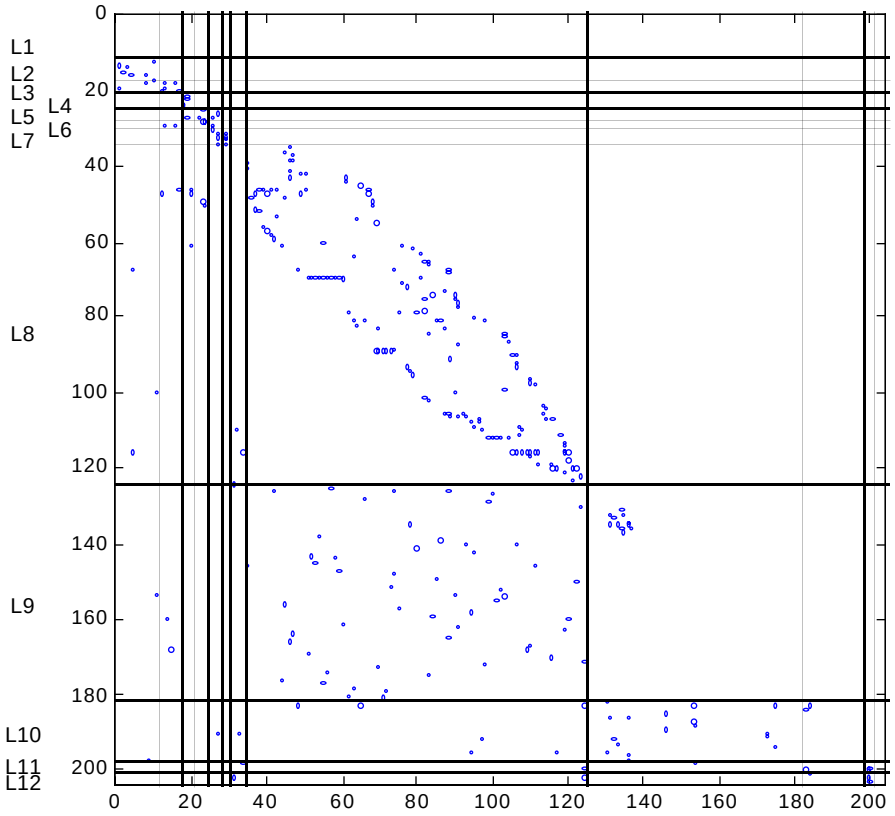


3.3 Cycle partition

Since all the elements in a feedback loop share the same predecessor and successor set in a reachability matrix, a level partition clusters the elements of interconnected feedback loops into a single block. It is possible to further partition a level into blocks of vertices that share the same set of predecessors and successors in a reachability matrix (the algorithm to do this is trivial). This further partition is called a cycle partition and the set of vertices is called a maximal cycle set (Warfield, 1989). The main characteristics of a maximal cycle set is that all its elements are reachable from every element of the set, i.e., its reachability matrix is full of ones, and that all elements can be traced into a single (maximal) feedback loop. Cycle partitions are strongly interconnected graphs and are at the core of SD model structure as they contain all the feedback loops in a model.

If the adjacency matrix is reordered according to level structure, the resulting matrix is a lower block triangular with each block representing a level. Elements in the main block diagonal correspond to elements in maximal cycle sets. Figure 1d illustrates the block ordering of an adjacency matrix with three levels, and a cycle set in level 2. Figure 5 shows the adjacency matrix (ones are represented by bullets), block ordered by level, for a typical SD model of 200 equations. The model has 12 levels with cycle sets in levels 5, 8, 9, 10 and 11.

Figure 5. Level partitions of adjacency matrix



Although, the process of model calibration is determined by what kind of data is available, having the level partition gives a path for a desirable sequence to estimate parameters or test the model. For example, once a 'high-level' piece of structure has been tested and calibrated, it would be possible to use the output of intermediate equations —originally unobserved— as data for estimation/testing structure deeper into the model. The sequential process reduces the size of the calibration problems in the lower levels of the structure. I have found it useful to start at either end of the level structure and work towards the inner parts of the model structure —normally the strongly interconnected blocks.

Level partitions, however, are of limited usefulness when confronted with large cycle sets —blocks of densely interconnected variables. The cycle set in the eighth level of figure 5 represents 44% of the model variables (89/203) and knowing that all those variables are at the same depth of the causal structure is not very helpful in determining a calibration sequence. Unfortunately, such cycle sets are not atypical of SD models. The next subsection presents a strategy for tackling the structural complexity of a cycle set and providing some insights into how to sequence the calibration process.

4. Hierarchy of cycle sets

The main advantage of a cycle partition is that it identifies the set of strongly interconnected elements that contain all the feedback complexity of a model structure. The interconnectedness of a cycle partition, however, makes it difficult to segment it and make separate estimations of parameters. Furthermore, the number of possible feedback loops in a cycle set is very large.

Kampmann (1996) defined an independent loop set (ILS) as a maximal set of loops whose incidence vectors are linearly independent, and showed that the ILS, while not unique, is a complete and non-redundant description of the feedback complexity of a graph. In this section, I propose a strategy to identify an ILS based on the shortest loops possible, and an algorithm to organize the ILS in a way that permits the structural understanding of the relationships among the loops and informs a more structured partial testing strategy. The process is illustrated here with a cycle partition; a full analysis of a model's structure would require the process to be repeated for all cycle partitions in the model.

A well known result from graph theory states that raising a reflexive adjacency matrix —an adjacency matrix with its main diagonal filled with ones— to the ith power, yields the matrix of a digraph with the relationship "reachable with i steps". Using this result, it is possible to derive a *distance* matrix (**D**) that shows in each cell the length of the shortest path between two vertices (Warfield, 1989).

$$\mathbf{B} = \mathbf{C} + \mathbf{I}$$

$$\mathbf{D} = \mathbf{C} + \sum_{i=2}^{|\mathbf{C}|-1} i\left(\mathbf{B}^i - \mathbf{B}^{i-1}\right)$$

Where **C** is a cycle partition of an adjacency matrix and all operators are in ordinary (non-Boolean) matrix algebra, with the exception of the powers of B, which result from Boolean products.

Notice that, since all the elements of a cycle partition are reachable from every other element of the cycle set, all entries of **D** are non-zero. By recursive inspection of the distance matrix, it is possible to identify the particular sequence of vertices that form a path between any two vertices. A feedback loop (cycle) between vertices *u* and *v* is defined by two paths: $u \rightarrow v$ and $v \rightarrow u$. Figure 6 shows the pseudo-code of an algorithm to identify the elements of a geodetic cycle between two vertices by identifying first the elements of the $u \rightarrow v$ path and then tracking the $v \rightarrow u$ path.

A cycle identified from a distance matrix is called a geodetic cycle since there is no shorter cycle in which these two vertices are involved. The length of the geodetic cycles can be obtained by adding the transpose of the lower block triangular component of the *distance* matrix to its upper triangular component ($\mathbf{L}=\mathbf{D}_L'+\mathbf{D}_U$). The resulting matrix is upper triangular, and the maximum entry in **L** defines the longest geodetic cycle in the set.

Taking advantage of the *length* matrix (**L**) to order the search process, it is possible to systematically explore the feedback loops in a cycle partition. Figure 7 shows the pseudo-code of an algorithm that makes use of the loop_track routine and generates an array with the vertices involved in a feedback loop in each cell. Since the tracking of loops is driven by loop-length —first all loops of length two, then three, etc.— loops in the output array are ordered by length. Notice that the algorithm only reports geodetic cycles —i.e., the shortest cycle linking any two elements— and does not identify all the feedback loops possible in a cycle set. Nevertheless, the number of geodetic cycles is still large. For the cycle set of 89 elements depicted in Level 8 of Figure 5, the algorithm identified 997 unique geodetic cycles. While the number is still large, the algorithm is much more efficient than an exhaustive search of loops, and still guarantees that all edges in the cycle set are considered.[6]

Kampmann (1996) proved that an ILS can be formed by only accepting a feedback loop into the set if it contains at least one edge that was not included in the previously accepted loops. It can be proven that using this simple construction rule on a list that includes all the geodetic cycles in a cycle partition generates the full ILS. Furthermore, if the loops considered are in order of their length, the resulting ILS is formed of the shortest loops possible. While not unique, an ILS constructed from the shortest loops possible does represent the simplest, and most granular, representation of the structure in a cycle set.[7] For example, the 997 unique geodetic cycles

identified for the cycle partition in Figure 5 were reduced trough this process to an ILS with 66 loops.

Figure 6. Pseudo-code for tracking loop elements - Based on Warfield (1989)

```
x ← Loop_track(u,v,D)
k ← 1                                   initialize element counter
x(k) ← u                                add first element to loop
                                        the u→v path
for i=1:D_uv-1                          for every distance 1 up to D_uv-1
    rh ← Pred_D[v] == D_uv-i            id predecessors of v that are D_uv-i steps away
    lh ← Succ_D[x(k)] == 1              id successors of last element in loop
    k ← k++                             increment element counter
    for each vertex w ∈ rh              for each predecessor of v
        if w ∈ lh                           if element of the successor group
            x(k) ← w                            add element to loop
            break                               break from for
        end                                 end
end                                     end
                                        the v→u path
k ← k++;                                increment element counter
x(k) ← v                                add mid-way element to loop
for i=1:D_vu-1                          for every distance 1 up to D_vu-1
    rh ← Pred_D[u] == D_vu-i            id predecessors of u that are D_vu-i steps away
    lh ← Succ_D[x(k)] == 1              id successors of last element in loop
    k ← k++                             increment element counter
    for each vertex w ∈ lh              for each successor of last element in loop
        if w ∈ rh                           if element of the predecessor group
            x(k) ← w                            add element to loop
            break                               break from for
        end                                 end
end                                     end
```

Figure 7. Pseudo-code for identifying geodetic cycles - Based on Warfield (1989)

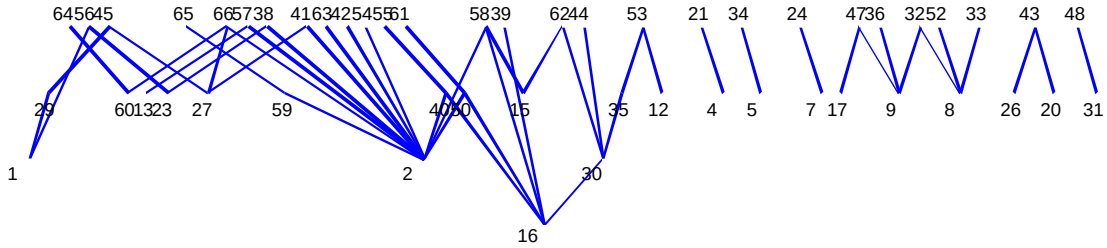
```
lps ← Loops (C)
k ← 1                                   initialize loop counter

B=C+I                                   calculate distance matrix (see text)
D=C
for i=2:|C|-1
    D=D+i*(B^i-B^(i-1))

L=D_L'+D_U                              calculate length of loops
for i=2:max(L)                          for all loop lengths
    pairs ← L_uv==i                     id all vertex pairs linked by loop of length i
    for each pair j ∈ pairs             for all active pairs linked by loop of length i
        lps(k) ← loop_track(j(1),j(2),D)    call loop_track routine
        pairs ← pairs - (pairs ∩ lps(k))    remove other pairs already id in loop
        k ← k++                         increment loop counter
    end                                 end
end                                     end
```

It is possible to create a graph with each feedback loop from the ILS in a vertex and the relationship "vertices are included in" as edges (Warfield, 1989). Applying the level partition algorithm described in the previous section to the resulting graph yields a loop hierarchy from simple (short) to complex (long). Figure 8 shows the hierarchy of loops in the cycle set of Level 8 in Figure 5.

Figure 8. Hierarchy of independent geodetic loops in a cycle partition†



† Loops not listed are level 1 loops with no other loop contained in them.

Notice that because the set of loops is an ILS, there are no cycle partitions in the resulting loop hierarchy structure. Furthermore, the loop hierarchy structure is relatively flat, and the density of 'inclusions' among loops (loops whose vertices are contained in other loops) is not very high. The resulting hierarchy of feedback loops can be used to develop an incremental testing sequence based on building confidence in simple structures first —inner loops in a model— and moving into more complex and interdependent structures.

5. Closing remarks

This paper has identified strategies to partition model structure and to sequence the calibration problems in order to develop confidence in the model structure while maximizing the information and insights that can be extracted from the data available. Of course, each calibration problem should be assessed to test whether the model structure is capable of replicating the historical behavior AND if the estimated model structure is consistent with what is known about the system (Oliva, 2000b). Furthermore, it should be noted that calibration is only the first step for testing a DH and should really be viewed as part of the model building process. Forrester stated that "confidence in a model arises form a twofold test —the defense of the components and the acceptability of over-all system behavior" (1961, p. 133). The proposed testing strategy aims only to increase the defensiveness of the model components. Full testing of a DH also requires tests at the system level, specifically, the historical fit of the model and the significance of the behavioral components of the hypothesis (Oliva, 1996).

In terms of future developments for this research, I foresee two fronts where this could be utilized. First, the articulation of partition heuristics and the formalization of the proposed heuristics are the first steps towards a "semi-intelligent process to use modeler's expertise with the abilities of automated calibration" that Lyneis and Pugh (1996) argued for. Much work still needs to be done in this area. Second, the tools developed to assist in the analysis of the model structure should also help on the research to use loop dominance as the main tool for analysis of the behavior of system dynamics model. I am particularly optimistic of the potential of the reduced ILS formed with the shortest geodetic cycles. I believe that short feedback loops provide a more intuitive explanation of the loop dominance proposed by Kampmann (1996). The conjecture that the reduced ILS is unique for any given model needs to be proven formally. If that is the case, such ILS should be the foundation for any tool-set to analyze loop dominance.

[1] Although system dynamics models normally generate sparse graphs that are represented more efficiently through an adjacency-list (Cormen *et al.*, 1990), the adjacency-matrix representation will be used throughout this presentation because of the simplifications it allows in the codification of algorithms.

[2] http://www.people.hbs.edu/roliva/research/sd/.

[3] Boolean matrix algebra is defined based on the Boolean addition (0+0=0; 0+1=1; 1+1=1). The Boolean product of two binary matrices W=YZ is defined only if the number of columns of Y corresponds to the number of rows in Z. Each entry on W is found by performing the Boolean sum $w_{ij}=\Sigma_k y_{ik} z_{kj}$.

[4] See Oliva (2000b) for a full description and specification of a calibration problem.

[5] If a vertex *v* is an element of *Succ[u]* and *Pred[u]* in a reachability matrix, it means that *v* and *u* are in a feedback loop (i.e., *v* is reachable form *u* and *u* is reachable from *v*) and belong to the same level.

[6] For comparison, Kampmann's (1996) exhaustive algorithm to identify feedback loops in a cycle set had identified over 12,000 feedback loops in the same cycle set before it was interrupted by an out-of-memory error.

[7] There is a chance that the ILS formed from the shortest possible loops might be unique.

6. References

Burns J. 1979. An algorithm for converting signed digraphs to Forrester's Schematics. *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9**(3): 115-124.

Cormen TH, Leiserson CE, Rivest RL. 1990. *Introduction to algorithms*. MIT Press: Cambridge MA.

Forrester JW. 1961. *Industrial dynamics*. MIT Press: Cambridge MA.

Homer JB. 1983. Partial-model testing as a validation tool for system dynamics. *Proceedings of the 1983 Int. System Dynamics Conference*, pp.919-932. Chestnut Hill MA.

Kampmann CE. 1996. Feedback loop gains and system behavior. *Proceedings of the 1996 Int. System Dynamics Conference*, pp.260-263. Cambridge MA.

Lyneis JM, Pugh AL. 1996. Automated vs. 'hand' calibration of system dynamics models: An experiment with a simple project model. *Proceedings of the 1996 Int. System Dynamics Conference*, pp.317-320. Cambridge MA.

Oliva R. 1996. Empirical validation of a dynamic hypothesis. *Proceedings of the 1996 Int. System Dynamics Conference*, pp.405-408. Cambridge MA.

Oliva R. 2000a. A Matlab implementation to assist Model Structure Analysis. System Dynamics Group, Massachusetts Institute of Technology, Memo D-4864. Cambridge MA. Available from http://www.people.hbs.edu/roliva/research/sd/.

Oliva R. 2000b. Model Calibration as a Testing Strategy for Dynamic Hypotheses. Harvard Business School, Working Paper #01-047. Boston, MA. Available from http://www.people.hbs.edu/roliva/research/sd/.

Warfield JN. 1989. *Societal systems: Planning, policy and complexity*. Intersystems Publications: Salinas CA.