# Model structure analysis through graph theory: partition heuristics and feedback structure decomposition

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

# Model structure analysis through graph theory: partition heuristics and feedback structure decomposition

## Abstract

The argument of this paper is that it is possible to focus on the structural complexity of system dynamics models to design a partition strategy that maximizes the test points between the model and the real world, and a calibration sequence that permits an incremental development of model confidence. It further argues that graph theory could be used as a basis for making sense of the structural complexity of system dynamics models, and that this structure could be used as a basis for more formal analysis of dynamic complexity. After reviewing the graph representation of system structure, the paper presents the rationale and algorithms for model partitions based on data availability and structural characteristics. Special attention is given the decomposition of cycle partitions that contain all the model's feedback loops, and a unique and granular representation of feedback complexity is derived. The paper concludes by identifying future research avenues in this arena.

*(Analysis of model structure; Graph theory; Structural complexity; Partial model calibration; Independent loop set)*

## 1. Introduction

One of the main benefits of the automated calibration techniques (e.g., full-information maximum-likelihood via optimal filtering, or model reference optimization) 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 is 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, automated calibration (AC) techniques are typically used to confirm a model's dynamic hypothesis: "Can the structure match the observed behavior?" In an earlier paper (Oliva 2003b) I proposed 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 suggested a set of increasingly demanding tests to guide the assessment of the AC output in the context of hypothesis testing, but was vague on how to support heuristic ii) and concluded with an invitation for further research into formal ways of partitioning a model for calibration and testing purposes. The present paper tackles this issue by formalizing the heuristics for model partitions and a sequencing strategy for the calibration/testing process.

From an operational perspective, having a complex error function and multiple calibration handles makes the tractability of the errors and the diagnosis of mismatches 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 'adjusts' 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.

Homer (1983) first articulated the argument for partial model testing as a way to increase the confidence in the model beyond its historical range of behavior, and improve the accuracy of the explanations of unobserved behavior. 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 goal of the proposed heuristics 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 argument of this paper is that it is possible to focus on the structural complexity of system dynamics (SD) models to design a partition strategy that maximizes the test points between the model and the real world, and a calibration sequence that permits an incremental development of model confidence. It further argues that the insights and tools from graph theory could be used as a basis for making sense of the structural complexity of SD models, and that this structure could be used as a basis for more formal analysis of dynamic complexity. The remainder 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 algorithm and for model partitions based on data availability –the main strategy for calibration partitions. Section 4 presents two structural partitions: level partitions, which group variables according to their causal interdependencies, and cycle partitions, strongly connected pieces of model structure that contain all feedback loops in a model. Since feedback complexity is at the core of SD models, a more detailed strategy to decompose cycle partitions is presented in section 5. 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

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. Graph theory has been used to explore the causal linkages of system dynamics models (Burns 1977; Burns and Ulgen 1978; Burns, Ulgen, and Beights 1979; Dolado and Torrealdea 1988; Kampmann 1996). However, with the exception of Kampmann, all of these works have focused on ways to automate the formulation of a system dynamics model from a causal loop diagram, i.e., a directed graph.[1] In this paper I am reversing the approach by suggesting that we can use graph theory to aid the analysis and understanding of the structural complexity —the interactions among variables and the feedback loops that those interactions create— of SD models.

A directed graph or digraph $G$ is a pair $(V, E)$, where $V$(G) is a finite set of elements, called vertices or nodes, 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.[2] 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 (Oliva 2003c) and is available online.

Insert Figure 1

Once the model structure is captured in an adjacency matrix, it is possible to use graph theory 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" (see Figure 1D) and it is equivalent to the transitive closure of **A** ($\mathbf{R}_{ii}$=1; $\mathbf{R}_{ij}$=$\mathbf{A}_{ij}$; $\mathbf{R}_{ij}$□ $\mathbf{R}_{jk}$□ $\mathbf{R}_{ik}$). 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 (**B=A+I**) and raising the sum to a Boolean power k such that $\mathbf{B}^{k-1}\neq\mathbf{B}^{k}=\mathbf{B}^{k+1}=\mathbf{R}$ (Warfield 1989).[3]

The following sections describe algorithms to facilitate the analysis of the model structure and follow the heuristic for model calibration in the smallest possible equation set. In particular, I 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 (Oliva 2003a) and are available online.

## 3. Data-availability partition

Given an available set of data series (model variables for which historical data is available), 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.

I will 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 structure to identify 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 (italics in figure 2), or an element is already identified in the tree structure, i.e., its predecessors have already been identified (variables in parenthesis in figure 2).

Insert Figure 2

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 using 7 equations. Table 1 lists the calibration problem by substituting the variable names with model equations (see Oliva 2003b for a full description and specification of a calibration problem).

Insert Table 1

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 (□) that could be estimated from it (see Oliva 2003a for an illustration of the use of this algorithm to specify multiple calibration problems).

Insert Figure 3

## 4. Structural partitions

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 a dynamic hypothesis is normally built by gradually integrating more complex and strongly connected components of the system to simple and observable pieces of structure. 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. Levels and cycles are structural partitions independent of data availability. These partitions are based solely on the relationships among model variables, and can facilitate the navigation of the model structure and the sequencing of partial model calibrations. [4]

### Level partitions

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 (Warfield 1989). A level partition groups variables according to their dependency on other model variables. The members of the first level are those that do not have any successors outside of its predecessor set, that is, the group of variables that have no further dependencies beyond themselves—normally the outcome variables of a model. Successive levels are identified by eliminating from the reachability matrix the top level and looking again for vertices without successors outside of the predecessor set. All the vertices in a feedback loop are at the same level since it is not possible to specify causal precedence between any two vertices in a loop.[5] Figure 4 shows the pseudo-code for an algorithm to partition a digraph by level blocks. The algorithm generates an array with the list of vertices that correspond to each level in each cell. If the adjacency matrix is reordered according to level structure, the resulting matrix is a lower block triangular with each block representing a level. Figure 5A shows the adjacency matrix, bock ordered by level, of the model in figure 1.

Insert Figure 4
Insert Figure 5

### Cycle partitions

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 level. 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 are 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 by a single (maximal) feedback loop. Cycle partitions are strongly connected graphs —graphs with a directed path from each vertex to every other vertex (Balakrishnan 1997)— and are at the core of SD model structure as they contain all the feedback loops in a model. In an adjacency matrix block ordered by level, elements with interactions in the main block diagonal correspond to elements in maximal cycle sets. Figure 5A shows a cycle set in level two, and the elements of the cycle (P, D and B) are the variables involved in the model feedback loops. Figure 5B shows the adjacency matrix (ones are represented by bullets), block ordered by level, for the 77-equation model described in Oliva and Sterman (2001).[6] The model has five levels with a cycle set in level 2; the 33 variables in level 2 belong to the cycle set. Figure 6 shows the details, and variable names, of the cycle partition in figure 5B.

Insert Figure 6

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 connected blocks.

Level partitions, however, are of limited usefulness when confronted with large cycle sets. The cycle set in figure 6 represents 43% of the model variables 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.

## 5. Structure of cycle partitions

The main advantage of a cycle partition is that it identifies the set of strongly connected elements that contain *all* the feedback complexity of a model structure. The interconnectedness of a cycle partition, and the large number of feedback loops that they contain (see Lemma A.1 in Kampmann 1996 for an estimate of the number of loops in a maximally connected graph), make 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) suggested the independent loop set (ILS) —a maximal set of loops whose incidence vectors are linearly independent— as a way to express the feedback complexity of a cycle partition and showed that the ILS, while not unique, is a complete 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 each cycle partition in the model.

### Identifying loops in a cycle partition

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” (see description of reachability matrix in §2). 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). A path is a walk, a sequence of connected vertices and edges, with non-repeating vertices (Balakrishnan 1997).

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

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

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.

Note that, since each element of a cycle set is 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 the shortest path between any two vertices. Linking the paths: $u \to v$ and $v \to u$ defines a circuit (a walk between a vertex and itself, i.e., a feedback loop) between vertices *u* and *v*. A circuit identified from a distance matrix is called a geodetic circuit since there is no shorter circuit in which these two vertices are involved (Warfield 1989). Figure 7 shows the pseudo-code of an algorithm to identify the elements of a geodetic circuit between two vertices by identifying first the elements of the $u \to v$ path and then tracking the $v \to u$ path.

Insert Figure 7

The length of the geodetic circuit between all vertex-pairs in the cycle partition can be obtained by adding the transpose of the lower block triangular component of the *distance* matrix to its upper triangular component (**L**=$\mathbf{D}_L$’+$\mathbf{D}_U$). The resulting matrix is upper triangular, and the maximum entry in **L** defines the longest geodetic circuit 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 8 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. By exploring loops for all the entries in **L** (every possible vertex pair in the cycle set), the algorithm is exhaustive, i.e., it tests every edge in the cycle partition. 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.

Insert Figure 8

Note that the algorithm only reports geodetic circuits and does not identify all the feedback loops in the cycle partition. Furthermore, a simple logical test (see if-line in figure 8) ensures that only cycles, circuits with non-repeating nodes (Chartrand 1977), are included in the list.[7] The geodetic cycle list generated by this algorithm is *unique* if the cycle partition has no shortest paths of equal length between any two vertices in the cycle set (proof is trivial). While the algorithm has no explicit way of selecting among alternative shortest paths of equal length (if they exist), it does guarantee that, if it exists, a shortest cycle linking every vertex-pair is included in the list. Nevertheless, the algorithm is exhaustive and the cycle list it generates contains every vertex and edge of the cycle partition, i.e., there is no information loss and it is possible to recreate the full cycle partition from the loop list.

For the 33-element cycle set in figure 6, the algorithm identified 48 geodetic cycles. While the number of geodetic cycles is still large, the algorithm is much more precise and efficient than an exhaustive loop search. For comparison, the exhaustive algorithm suggested by Kampmann (1996) required 86% more CPU time to identify 106 feedback loops in the same cycle set.

### Independent Loop Set

Kampmann (1996) proved that an ILS can be formed by accepting a feedback loop into the set if, and only if, it contains at least one edge not included in the previously accepted loops. He suggests an algorithm to identify an ILS that expands the existing loop set by considering an edge to an adjacent vertex and identifying the shortest path back to the accepted loop set (see theorem 2.1 in Kampmann 1996). While Kampmann's algorithm identifies the maximum number of independent loops for a cycle partition —for a cycle partition of E edges and V vertices the maximum number of independent loops is equal to E-V+1 (see theorem 2.1 in Kampmann 1996)— the algorithm does not control for the complexity of the resulting loops; the shortest path back to the existing loop set does not guarantee that the shortest loop for the selected edge will be identified. Furthermore, the algorithm is path dependent, new loops are added based on the existing loop set, and since the algorithm does not provide a clear starting point, nor a rule to select the next adjacent vertex, no rationale is defined for the selection of loops in the ILS.

Using Kampmann's simple construction rule (accept a loop in the ILS iff it contains at least one edge not included in the previously accepted loops) on a list of all the geodetic cycles in a cycle partition is equally effective in generating an ILS for that cycle partition. Moreover, if the loops are considered for ILS inclusion by selecting the shortest loop that makes the smallest addition of new edges to the loop set, the resulting ILS is formed by the shortest loops possible. That is, by ensuring that every loop considered for the inclusion is a geodetic cycle (the shortest loop that links the two vertices linked by the edge under consideration), and that the edge for consideration is always the edge that can be exclusively covered by the shortest cycle not in the set, the algorithm produces an ILS with the shortest loops. I call the ILS generated through this process the shortest independent loop set (SILS). Figure 9 shows an implementation of this algorithm that takes as inputs the geodetic loop list and the adjacency matrix of the cycle partition (**C**).

Insert Figure 9

Although the SILS algorithm in figure 9 is constrained to work with geodetic cycles (a subset of all the possible feedback loops in the partition), by conditioning the introduction of new cycles into the ILS to the simplest (shortest) loop that makes the minimal contribution of new edges, the algorithm ensures each loop introduced into the ILS brings relatively few new edges to the loop set, and the number of loops in the SILS is always equal to the maximum identified by Kampmann (see Appendix for proof).

The SILS decomposes the cycle partition into the maximum number of loops. It is the simplest and most granular description of the structure in a cycle partition and it permits to express feedback complexity in its simplest components: the shortest loops. While the SILS is not unique (the algorithm to identify geodetic cycles has no way of selecting if there are two or more “shortest” paths of equal length between two variables) the number of loops in an SILS *and* their length is *unique*. That is, the SILS of a cycle partition has a fixed number of loops and those loops have a predetermined length, regardless of starting points or the order in which variables are arranged in the adjacency matrix. If there are no shortest paths of equal length between two vertices in the cycle partition, the SILS for that partition is unique (see Appendix for proof). The 48 geodetic cycles identified in the previous section were reduced to an SILS with 25 loops (see Table 2).[8] Figure 10 shows the stock and flow diagram of the elements in the cycle partition in figure 6; each loop in the SILS is identified in Figure 10 by its sequential number in table 2.

Insert Table 2
Insert Figure 10

### Loop hierarchy in an SILS

While the SILS is a detailed decomposition of the feedback complexity of a cycle partition, the number of loops it contains still makes it difficult to work with for calibration purposes. Following are two criteria to organize the SILS in a way that permits the structural understanding of the relationships among the loops and informs a more structured partial testing strategy.

The first analysis takes advantage of the tools already developed by creating a graph with each feedback loop in the SILS in a vertex and the relationship “included in” as edges (Warfield 1989) —loop A is said to be "included" in loop B if all the vertex of A are also in B. Note that although all the vertices of A are in B, not all the edges in A are in B. For example, the loop {*u,v*} would be included in the loop {*u,v,x*} although the first loop does not contain the edges *v* *x* and *x* *u* and the second loop does not contain the edge *v* *u*.[9] Applying the level partition algorithm (figure 4) to the resulting inclusion graph yields a loop-hierarchy, from simple (short) to complex (long), of all the loops in the SILS. Figure 11A shows the SILS’ loop-hierarchy of the cycle partition in figure 6; each loop is represented by its id number (see Table 2), and its length is reported in brackets. In the top row of the graph in figure 11A are listed the shortest loops in the partition or, more generally, those loops that do not include any other loop (level 1). The next row contains the loops that include one or more loops from the first level, and so on. Note that the "included in" relationship is transitive (A□B □ B□C □ A□C) but not symmetrical (A□B □ ~(B□A)), thus the resulting graphs are always forests: collections of acyclical graphs (trees). Furthermore, the loop hierarchy structure of an ILS is relatively flat, and the density of "inclusions" among loops is not very high.

Insert Figure 11

A first benefit of this hierarchical representation is that it explicitly shows how the each level of loops incorporates more structure and linkage between separated loops. This representation makes evident that to understand the workings of any loop it is first necessary to have an understanding of the loops from previous levels included in it. For instance, note in figure 11A how loop 25 includes the elements of loops 1, 5, and 6. This suggests that, if possible, loops 1, 5, and 6 should be independently calibrated before any attempts to estimate the parameters governing the behavior of loop 25.

The second benefit of this representation is that it helps isolate elements of the feedback complexity for calibration purposes. For instance, note how loops 2, 7, 8, 9, and 12 are isolated from the rest of the graph in figure 11A. These loops correspond to the causal tree in figure 2 and the calibration problem in Table 1. Although figure 2 was used as an illustration of data availability partition, the structural representation in figure 11A makes it clear that that particular piece of structure could be analyzed and calibrated as a reasonable partition of model structure.

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. Oliva (1996a) describes the sequence of tests performed to calibrate the cycle partition in figure 6. The sequence can be mapped directly into the graph of figure 11A to illustrate the process of gradual model testing and development (I suggest also following the test sequence in figure 10). The parameters in loops 1, 3, 4, and 6 were either directly observable or estimated from interview data (simple first level loops). Loops 2, 7, 8, 9, and 12 were calibrated together as suggested in figure 2, and a separate calibration problem was used to estimate the parameters of loops 11 and 17. With parameters from loops 1 and 17 already identified, it was possible to estimate additional parameters for loops 13 and 14. With loops 4 and 6 as anchors, it was possible to estimate the parameters around loops 18 and 21, thus solving also loops 3, and 10. The process continues until there are no independent parameters left for the reminding loops (for example, the independent edges in loops 15 and 19 are capture only a definitional relationship for total labor).

The second analysis of feedback structure is based on the eccentricity of each vertex with respect to the rest of the vertices in the cycle partition. Eccentricity of a vertex is defined in graph theory as the longest of the shortest paths between that vertex and every other vertex in the graph —this measures how far the node is from its furthest node in the graph. The vertex eccentricity can be obtained directly from the distance matrix (**D**) calculated in §4 —the maximum of each row represents the vertex eccentricity. The vertex with the smallest eccentricity is said to be in the center of the graph, and the minimum eccentricity of a graph (that of its central vertex) is the graph's radius. The average eccentricity of a loop is the average of the eccentricities of the vertices that constitute the loop, and by subtracting the eccentricity of the central node of the partition is possible to calculate how far the loop is from the center of the graph. Figure 11B shows every loop (identified by its id number) in a grid of "mean distance from the center" vs. "loop length." Overlaid in the grid is also the graph of the "included" relationship described above. Used in combination with the level graph, this second clustering of the loops in an SILS provides the guidelines for a strategy to make sense of the structural complexity of a partition by either beginning to make sense of the dynamics from the center of the partition outward, or by starting with loops in the edges of the partition and working towards the center of the partition.

## 6. Beyond structural complexity

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. Granted, SD is interested in understanding system behavior (not structure per se), yet, one of the core assumptions of the field is that "behavior arises out of system structure"(Meadows 1989). Certainly in developing policy recommendations we look for structural changes that would eliminate the undesired/pathological behavior. Having better tools to understand and simplify structural complexity will permit a more efficient policy design process. The usefulness of the frameworks for model structural analysis goes beyond model partition for calibration. A more organized understanding of the structural complexity of our systems' models yields deeper understanding of the causes of their dynamic complexity.

From a policy analysis perspective, the inclusion graph of the SILS (figure 11) helps identify loops that, although independent in the ILS definition of independent, might not be independently adjustable. Bundling loops by their vertex-similarity makes it evident that the same parameters affect the gains of a set of loops in a tree. It may be impossible to adjust the gain of, say, loop 1 in figure 11, without affecting the gains of loop 13, 14, and 25, i.e., loop 1 is not independently adjustable.

Because the loop inclusion rule proposed by Kampmann checks only if a new edge is being added to the set, and not the impact of the addition on the existing set, a combination of loops might make a previously accepted loop redundant. Consider, for example, a cycle partition with an SILS with three loops {{*u,v*},{*u,v,x*},{*u,w,v*}}.While the set is an ILS, the combination of the second and third loop makes the inclusion of the first loop redundant. The two edges in the first loop (*u*□ *v* and *v*□ *u*) are included in the other two loops and the set {{*u,v,x*},{*u,w,v*}} is a more parsimonious ILS for the cycle partition. These redundancies can be eliminated from an ILS by applying Kampmann's construction rule a second time, but considering the loops for inclusion in the inverse contribution and length order (i.e., from longest to shortest). That is, a loop of certain length is only included in the ILS if at least one of the edges was not included in longer loops. Applying this algorithm to the SILS yields the minimal shortest ILS (MSILS).[10] The MSILS is a unique derivation from a starting SILS (see Appendix for proof), and by eliminating redundancies, a MSILS provides a set of independently adjustable loops that may be best suited for loop dominance analysis and policy design. For cycle sets with more than 10 vertices, the MSILS typically contains 20% less loops than the SILS; the difference tends to disappear with smaller sets. The SILS in Table 2 reduces to an MSILS with 18 loops (loops in the MSILS are identified with a bold number in Table 2).

The fact that feedback complexity can be reduced to a unique granular representation of independent feedback loops leads me to posit that the SILS and MSILS should be considered a basis for our field's efforts to understand loop dominance. Explicitly exploring loop dominance in terms of all possible independent paths in a cycle partition, as opposed to the aggregated gains between state variables as suggested by Gonçalves, Lerpattarapong, and Hines (2000), should generate more detailed explanations of loop dominance. Note that Kampmann's suggested approach for of loop dominance analysis (1996) is contingent to the set of pre-selected loops. That is, Kampmann's assessment of loop dominance for a phase of the system behavior will always identify one of the pre-selected loops. Thus, performing the analysis on an SILS will always generate an explanation of dominant behavior in term of shorter loops. I posit that the loops identified by the SILS algorithm will generate the most intuitive explanation of observed behavior in terms of loop dominance. It will be interesting to test how the SILS compares to the single loop explanations generated by the pathway participation method (Mojtahedzadeh 1996; Mojtahedzadeh, Andersen, and Richardson 2003).[11]

## 7. 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. In the process of developing these strategies, a set of tools and distinctions were proposed to talk more precisely about the structural complexity. The constructs of level and cycle partitions, SILS, loop inclusion, and loop eccentricity give us another way in into making sense of models and develop insights from their structural complexity. Of course, a good system dynamicist might develop the same insights and identify the interconnectedness of loop bundles just from staring at a model diagram. My hope is that this formal process and representation will, first, simplify the cognitive burden for thinking about the structural complexity of a model, and reduce the learning curve for a novice to develop insights from it. Second, by providing a structured way to think and talk about structural complexity these formal distinctions should facilitate knowledge accumulation and transferability of insights across models.

Focusing on structural complexity is just a means to an end. System dynamicists should strive to develop confidence in their dynamic hypotheses: how the structure drives behavior. Model calibration—the process of estimating the model parameters (structure) to obtain a match between observed and simulated structures and behaviors—is a stringent test of a hypothesis linking structure and behavior, but 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 2003b). It should be noted, however, that calibration is only the first step for testing a dynamic hypothesis 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 dynamic hypothesis 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 1996b).

In terms of future developments for this research, I foresee two fronts where it 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, but having a consistent and predictable way to partition a model should be one of the first steps to develop such semi-intelligent process. Second, the tools developed to assist in the analysis of the model structure should also help the research to use loop dominance as a tool for analysis of the behavior of system dynamics models. I am particularly optimistic for the potential of the SILS and MSILS. I believe that short feedback loops provide a more intuitive explanation of the loop dominance proposed by Kampmann (1996) and the fact that these sets have unique topology for any given model suggests that this set could be the foundation for a generalizable tool to analyze loop dominance.

[1] Since a full SD formulation contains much more information than a causal loop diagram, it is not surprising that these efforts were not successful and that the algorithms never became part of the SD toolkit.

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

[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 $w_{ij}=\square_k y_{ik}z_{kj}$.

[4] Burns and Ulgen (1978) proposed “sector partition” —“quantities [vertex] and couplings [edges] that can be associated with one and only one flow” (p. 653)— was not considered because it becomes ambiguous if two flows are in a feedback loop.

[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] The model is fully documented and available at: http://www.people.hbs.edu/roliva/research/service/esq.html. The model used for this example is the same model available in the web site but without the variables in the quality sector that were not active in the final calibration of the model.

[7] A repeated node in a loop means that the two paths connecting the two extremes of the circuit cross before reaching their destination and that shorter loops have been identified for those paths. For example, if the paths $u \to v$ and $v \to u$ share the node $w$, it means that the paths $u \to w$, $w \to v$, $v \to w$, and $w \to u$ exist, and that those paths, by virtue of being shorter, have been identified by the algorithm as part of the loops $u \to w \to u$ and $v \to w \to v$.

[8] The cycle partition contains 61 edges and 33 vertices, thus the rank of the partition matrix is 29. However, four of the edges in the partition are self-reference edges (main diagonal in figure 6). The algorithm was implemented to ignore loops of a single edge, and effectively treats the partition matrix as a matrix with rank 25: 57 edges and 33 vertices.

[9] The loop notation $\{u,v,x\}$ represents an ordered sequence of vertices in the loop, i.e., the loop is constituted by the path $u \to v \to x \to u$.

[10] Variations in the applications of these algorithms generate ILS with different properties. For instance, selecting always the shortest loop for inclusion (regardless of its contribution of new edges) generates an ILS with the shortest loops but not necessarily with a full decomposition of the feedback structure (the number of loops will be less than the theoretical maximum). Applying the inverse inclusion rule (from longer to shorter) to the exhaustive list of geodetic cycles yields the most parsimonious ILS formed with cycles.

[11] I thank David Andersen and Hazhir Rahmandad who independently suggested this test.

## Appendix

*Theorem 1. The SILS algorithm identifies E-V+1 independent loops in a partition with E edges and V vertices.*

*Proof:* The SILS algorithm is a special implementation of Kampmann's algorithm. The "minimal contribution of new edges" criterion is equivalent to Kampmann's "shortest path back to the set" criterion for loop construction. Both of these criteria ensures that each added loop adds a minimum number of edges to the edge set of the ILS. Kampmann's theorem 2.1 also proves that the SILS algorithm will always find the maximum number of loops possible in a cycle partition.

*Theorem 2. The SILS algorithm generates a unique outcome from a list of geodetic cycles.*

*Proof:* Let G be the set of geodetic cycles of a cycle partition C with E edges and V vertices, identified by the algorithm in Figure 8. A geodetic cycle *g*□G does not belong in the shortest independent loop set $S^*$ if for each edge $e_i$□*g* there is a shorter loop that contains that edge. That is, *g* does not belong in $S^*$ if there is a set *S* of shorter geodetic cycles that includes all its edges.

Whenever an edge *u*□*g* is considered to expand $S^*$, regardless of the circumstances under which *u* became the edge to be considered, the algorithm will *always* select the shortest loop to cover that edge, i.e., a member of *S*, thus systematically rejecting *g* from $S^*$. Since the algorithm considers all the edges in C, and always excludes the same loops from $S^*$, by implication, it always accepts the same loops.

Note that G contains only one geodetic cycle for each vertex pair (if it exists). If G contains more than one loop of the same length over an edge *u*, all these loops but one will have at least one edge *v*, different from *u*, for which that loop is a geodetic cycle to include the edge —the only reason why subsequent loops would be added for the same edge is if the loop brought in edge *v* not covered by the loops already in G. Since the SILS algorithm only includes loops with minimal contribution to the edge set in $S^*$, if edge *u* is considered for inclusion in $S^*$ before edge *v*, only the loop for which *u* is the unique geodetic cycle will be selected (all other loops would contribute $u+v$ into the edge set in $S^*$). Finally, note that if edge *v* is considered for inclusion in $S^*$ before *u*, and a loop including both edges was accepted into $S^*$, edge *u* would never be considered to expand $S^*$. Thus even with geodetic cycles of equal length, the algorithm identifies a unique loop set from the list of geodetic cycles.

The same rationale (but with inverted criteria) proves the uniqueness of a MSILS generated from an SILS.

*Theorem 2. The lengths of the loops in an SILS is unique.*

*Proof:* One list of geodetic cycles for a cycle partition will differ from another list only if alternative shortest paths between two vertices exist. That is, two list of geodetic cycles will differ only in individual loops when different paths (of same length) can be taken to link a pair through a geodetic cycle. Note, however, that alternative paths will be captured in different lists by geodetic cycles of the same length. When the SILS algorithm identifies the shortest geodetic cycle for a particular edge, it finds a cycle of a fix length, regardless of the path chosen in the geodetic cycle list.

## References

Balakrishnan VK. 1997. *Graph theory*, *Schaum's Outlines*. McGraw-Hill: New York.

Burns JR. 1977. Converting signed digraphs to Forrester schematics and converting Forrester schematics to differential equations. *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-7**(10): 695-707.

Burns JR, Ulgen OM. 1978. A sector approach to the formulation of system dynamics models. *International Journal of Systems Science* **9**(6): 649-680.

Burns JR, Ulgen OM, Beights HW. 1979. An algorithm for converting signed digraphs to Forrester's schematics. *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9**(3): 115-124.

Chartrand G. 1977. *Introduction to graph theory*. Dover Publications: New York.

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

Dolado JJ, Torrealdea FJ. 1988. Formal manipulation of Forrester diagrams by graph grammars. *IEEE Transactions on Systems, Man, and Cybernetics* **18**(6): 981-996.

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

Gonçalves P, Lerpattarapong C, Hines JH. 2000. Implementing formal model analysis. *Proceedings of the 2000 Int. System Dynamics Conference*. Bergen, Norway.

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 (unpublished manuscript). Summarized in Proceedings of 1996 Int. System Dynamics Conference (p. 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.

Meadows DH. 1989. System dynamics meets the press. *System Dynamics Review* **5**(1): 68-80.

Mojtahedzadeh MT. 1996. A path taken: Computer-assisted heuristics for understanding dynamic systems. PhD Thesis, Rockefeller College of Pubic Affairs and Policy, State University of New York at Albany, Albany, NY.

Mojtahedzadeh MT, Andersen D, Richardson GP. 2003. Using *Digest* to implement the pathway participation method for detecting influential system structure. Manuscript, Rockefeller College, University at Albany, State University of New York, Albany, NY.

Oliva R. 1996a. A dynamic theory of service delivery: Implications for managing service quality. PhD Thesis, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

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

Oliva R. 2003a. A Matlab implementation to assist Model Structure Analysis. System Dynamics Group, Massachusetts Institute of Technology, Memo, D-4864-2. Cambridge MA. Available from http://www.people.hbs.edu/roliva/research/sd/.

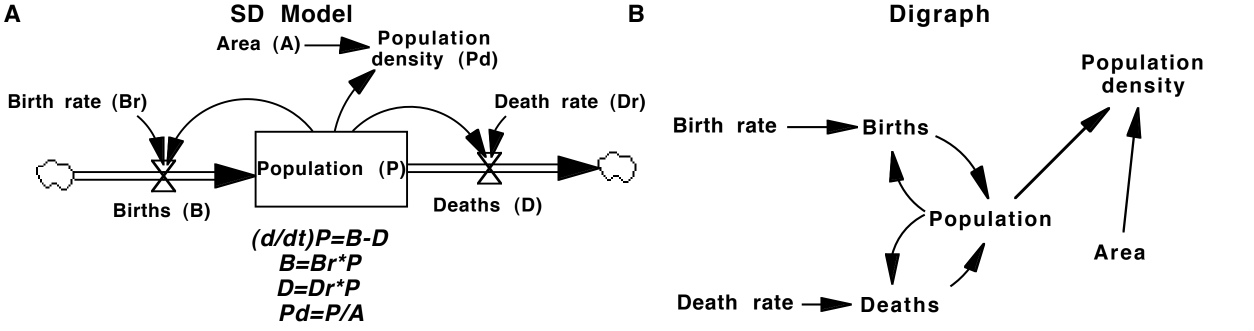
Oliva R. 2003b. Model calibration as a testing strategy for system dynamics models. *European Journal of Operational Research* **151**(3): 552-568.

Oliva R. 2003c. Vensim® model to adjacency matrix utility. Harvard Business School. Boston, MA. June 02, 2003. Available from http://www.people.hbs.edu/roliva/research/sd/.

Oliva R, Sterman JD. 2001. Cutting corners and working overtime: Quality erosion in the service industry. *Management Science* **47**(7): 894-914.

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

Figure 1. Graph representation of system structure



C 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 |

D Reachability matrix

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

Figure 2. Causal tree for Time per order

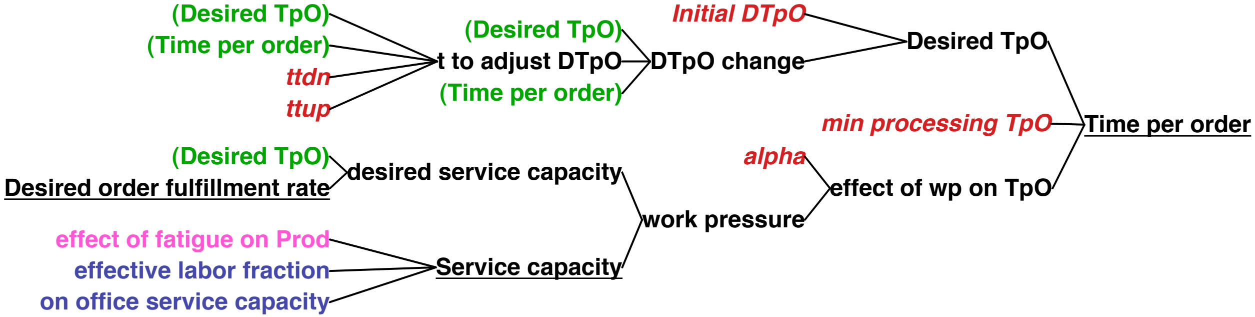


Table 1. Calibration problem

***Minimize***

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

***Over:***

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

***Subject to:***

Time per order = max(Desired TpO * effect of wp on TpO, min processing TpO)
Desired TpO = INTEG (DTpO change, Initial DTpO)
DTpO change = (Time per order-Desired TpO)/t to adjust DTpO
t to adjust DTpO = IF THEN ELSE(Time per order>Desired TpO , ttup , ttdn)
effect of wp on TpO = EXP(alpha * work pressure)
work pressure = (Desired service capacity - **Service capacity**)/Desired service capacity
desired service capacity = **Desired order fulfillment rate** * Desired TpO
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.

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

```
[y,x,□,eq] □ BFS(A,d)
for each vertex i □ V[A]                          for all vertex in model
    if $Pred_A[i]=\emptyset$                            if no predecessors
        p □ p □ i                                     id as system parameters

m □ 0                                             initialize problem counter
for each vertex i □ d                             for each vertex with available data
    m □ ++                                            increment problem counter
    y(m) □ i                                          introduce data on dependent variable
    for each vertex j □ V[A]-i                        for all other vertex
        color[j] □ white                                  mark as not discovered
    Q □ i                                             initialize queue with data vertex
    while $Q \neq \emptyset$                               while elements in queue
        j □ head[Q]                                       take first element from queue
        for each vertex k □$Pred_A[j]$                       for each predecessor
            if color[k] == white                              if not discovered
                color[k] □ gray                                   mark as discovered
                if k □ d                                          if data available
                    x(m) □ x(m) □ k                                   enter as known input
                else                                              else
                    if k □ p                                          if parameter
                        □(m) □ □(m) □ k                                   enter as parameter
                    else                                              otherwise
                        eq(m) □ eq(m) □ k                                 enter as equation
                        Enqueue (Q,k)                                     enter k in queue
        Dequeue (Q)                                       drop (j) from queue
        color[j] □ black                                  mark as explored
    end                                               end
end                                               end
```

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

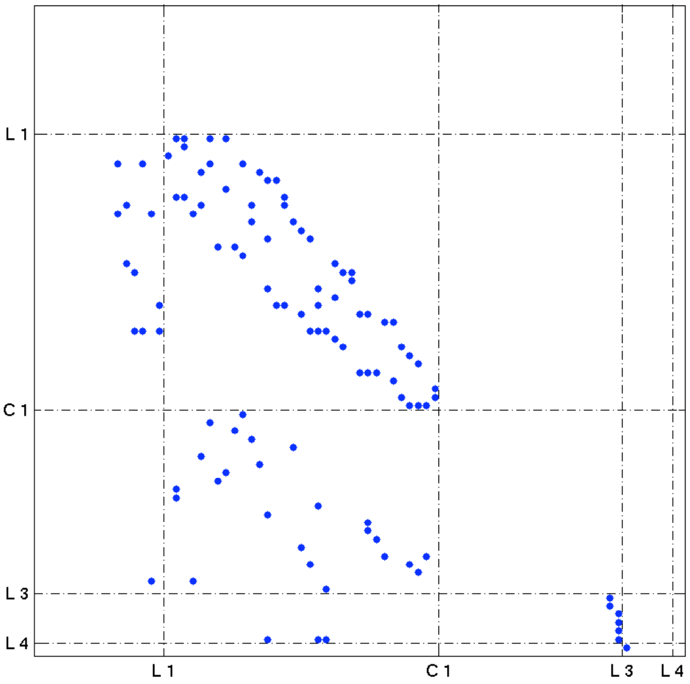
```
Lev □ Levels(R)
k □ 1                                      initialize level counter
while $V[R] \neq \emptyset$                         while vertex left
    for each vertex u □ V[R]                   for each vertex remaining
        do if $Pred_A[u]$ □ $Succ_A[u]$ == $Succ_A[u]$       if vertex is at top level, i.e. all successors are in predecessor set
            Lev[k] □ Lev[k] □ u                        introduce vertex to level
    V[R] □ V[R] – Lev[k]                       remove level from graph
    k □ ++                                     increment level counter
end                                        end
```

## Figure 5. Adjacency matrices block ordered by levels

A Model in Figure 1

| | | *Pd* | *A* | *P* | *B* | *D* | *Br* | *Dr* |
|---|---|---|---|---|---|---|---|---|
| *L*1 | *Pd* | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| | *A* | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| *L*2 | *P* | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| | *B* | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| | *D* | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| *L*3 | *Br* | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| | *Dr* | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

B Model in Oliva and Sterman (2001)†



† This sparsity pattern has the same structure as Figure 5A but variable names and zeros have been suppressed and ones have been replaced by bullets.

## Figure 6. Cycle partition of model in Oliva and Sterman (2001)

| | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Desired TpO | 1 | | 1 | 1 | | | 1 | | 1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| t to adjust DTpO | 2 | | | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| DTpO change | 3 | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Service Backlog | 4 | | | | | | 1 | | | | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| Fatigue A | 5 | | | | | 1 | | | | | | | 1 | | | | | | | | | | | | | | | | | | | | | |
| desired service capacity | 6 | | | | | | | | | | | | | 1 | 1 | | | | | | | | | | | | | | | | | | | |
| effect of wp on TpO | 7 | | | | | | | | 1 | | | | | | | | | | | | | | | | | | | | | | | | | |
| time per order | 8 | | 1 | 1 | | | | | | | | | | | | 1 | | | | | | | | | | | | | | | | | | |
| work intensity | 9 | | | | | 1 | | | | | | 1 | | | | 1 | | | | | | | | | | | | | | | | | | |
| order fulfillment | 10 | | | | 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Fatigue E | 11 | | | | | | | | | | | 1 | | | | | 1 | | | | | | | | | | | | | | | | | |
| effect of fatigue on turnover | 12 | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | | | | | | | |
| Desired Labor | 13 | | | | | | | | | | | | | 1 | | | | | 1 | | | | | | | | | | | | | | | |
| work pressure | 14 | | | | | | | 1 | | 1 | | | | | | | | | | | | | | | | | | | | | | | | |
| potential order fulfillment | 15 | | | | | | | | | | 1 | | | | | | | | | | | | | | | | | | | | | | | |
| effect of fatigue on prod | 16 | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | | | |
| turnover rate | 17 | | | | | | | | | | | | | | | | | | | | | | 1 | 1 | | | | | | | | | | |
| labor correction | 18 | | | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | |
| Perceived Labor Productivity | 19 | | | | | | | | | | | | | 1 | | | | | | 1 | | | | | | | | | | | | | | |
| on office service capacity | 20 | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | | | |
| service capacity | 21 | | | | | | | | | | | | | | 1 | 1 | | | | 1 | | | | | | | | | | | | | | |
| Experienced Personnel | 22 | | | | | | | | | | | | | | | | | 1 | | | | | | | 1 | 1 | | | | | | | | |
| desired hiring | 23 | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | 1 | | | | | |
| total labor | 24 | | | | | | | | | | | | | | | | | | 1 | 1 | 1 | | | | | | | | | | | | | |
| effective labor fraction | 25 | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | | | | | | |
| experience rate | 26 | | | | | | | | | | | | | | | | | | | | | | 1 | | | | | | | 1 | | | | |
| desired vacancies | 27 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | | |
| indicated labor order rate | 28 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | |
| Rookies | 29 | | | | | | | | | | | | | | | | | | | | | | | | 1 | 1 | 1 | | | | | | | |
| vacancies correction | 30 | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | | | | |
| labor order rate | 31 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 |
| hiring rate | 32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | | | | 1 |
| Vacancies | 33 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | 1 | 1 | 1 | |

Figure 7. Pseudo-code for tracking loop elements in Distance matrix

```
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 ← ++                                  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 ← ++;                                 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 ← ++                                  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
```

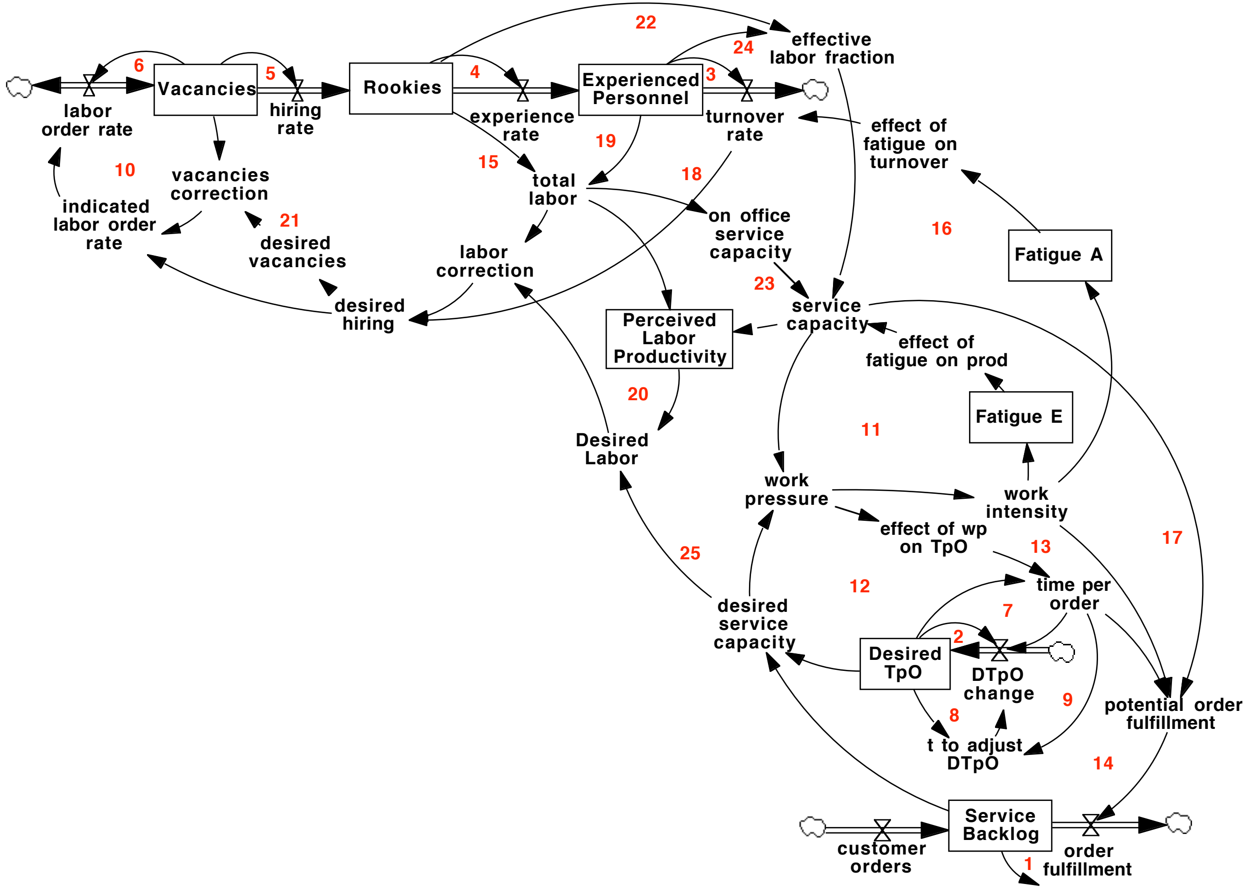
Figure 8. Pseudo-code for identifying geodetic cycles in cycle partition

```
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
        loop ← loop_track(j(1),j(2),D)          call loop_track routine
        pairs ← pairs - (pairs ∩ loop)          remove other pairs already identified in loop
        if loop == unique[loop]                 if elements of loop are unique – no repeating nodes
            lps(k) ← loop                           accept loop in list
            k ← k++                                 increment loop counter
        end                                     end
    end                                     end
end                                     end
```

Figure 9. Pseudo-code for identifying SILS from a length-sorted geodetic cycle list

```
sils □  SILS(lps,C)
B □  zeros(C)                         initialize empty adjacency matrix (same size as C)
k □  1                                initialize loop counter
S □  □ lps                            initialize S set with all loops
for each vertex i □ C                 for each vertex in partition C
    C(i,i)=0                              clear self-referring edges
                                      Id edges in each loop i and store them in E(i)
for each loop i □ lps                 for each loops in lps
    len □  length(lps(i))                 find loop length
    for j=1:len-1                         for each vertex in loop (sequentially)
        E(i,lps(i,j),lps(i,j+1))=1            fill in E(i) edge between current vertex and next
    E(i,lps(i,len),lps(i,1))=1            fill E(i) edge between last edge and first
end                                   end

while S ≠ ∅                           while loops left in S
    for j □ S                             for each loop in S
        c(j) □  count((E(j,:,:) – B)>0)       count new edges loop would introduce
        if c(j) == 0                          if loop does not contribute new edges
            S □  S - S(j)                         remove from S
    end                                   end
    m □  i □ min(nonzero(c(i)))           id pointers to loops that make minimum contribution of edges
    n □  m(1)                             truncate to first loop (the shortest)
    if n ≠ ∅                              if a loop was found
        B □  B + E(n,:,:)                 add loop edges into B
        sils(k) □  S(n)                       add loop into output list
        k □  ++                               increment loop counter
        S □  S - S(n)                         remove loop from S
        c(n) = 0                              clear loop contribution
    end                                   end
end                                   end
```

Figure 10. Stock and flow diagram of elements in cycle partition in Figure 6



## Table 2. SILS of cycle partition in Figure 6

**1** Backlog 1st order controller
Service Backlog
order fulfillment

**2** TpO adjustment (reference)
Desired TpO
DTpO change

3 Turnover rate
turnover rate
Experienced Personnel

**4** Experience rate
Rookies
experience rate

**5** Hiring rate
hiring rate
Vacancies

**6** Vacancies 1st order controller
Vacancies
labor order rate

7 TpO adjustment (update)
time per order
DTpO change
Desired TpO

**8** TpO constant adjustment (reference)
Desired TpO
t to adjust DTpO
DTpO change

**9** TpO constant adjustment (update)
time per order
t to adjust DTpO
DTpO change
Desired TpO

**10** Vacancy correction
Vacancies
vacancies correction
indicated labor order rate
labor order rate

11 Erosion of service capacity (fatigue)
service capacity
work pressure
work intensity
Fatigue E
effect of fatigue on prod

**12** Erosion of Time per Order
desired service capacity
work pressure
effect of wp on TpO
time per order
DTpO change
Desired TpO

**13** Work hard
Service Backlog
desired service capacity
work pressure
work intensity
potential order fulfillment
order fulfillment

**14** Cut corners
Service Backlog
desired service capacity
work pressure
effect of wp on TpO
time per order
potential order fulfillment
order fulfillment

15 Rookie adjustment
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
total labor
labor correction

**16** Erosion of service capacity (turnover)
service capacity
work pressure
work intensity
Fatigue A
effect of fatigue on turnover
turnover rate
Experienced Personnel
effective labor fraction

**17** Falling behind
Service Backlog
desired service capacity
work pressure
work intensity
Fatigue E
effect of fatigue on prod
service capacity
potential order fulfillment
order fulfillment

18 Turnover replacement
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
experience rate
Experienced Personnel
turnover rate

**19** Experienced adjustment
total labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
experience rate
Experienced Personnel

**20** Capacity adjustment decision (productivity)
Perceived Labor Productivity
Desired Labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
total labor

**21** Vacancy adjustment decision
turnover rate
desired hiring
desired vacancies
vacancies correction
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
experience rate
Experienced Personnel

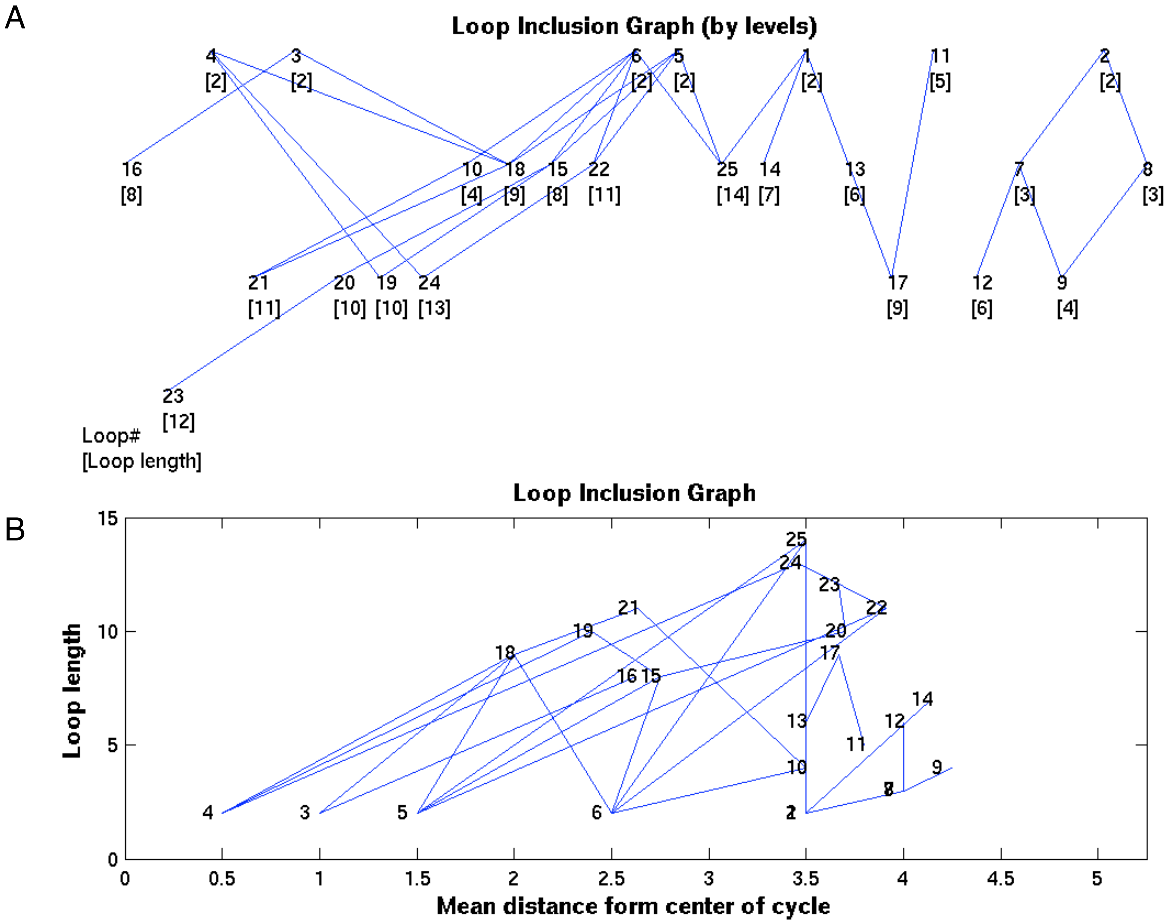
22 Perception of Rookie productivity
service capacity
Perceived Labor Productivity
Desired Labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
effective labor fraction

**23** Perception of absenteeism's effect on productivity
on office service capacity
service capacity
Perceived Labor Productivity
Desired Labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
total labor

24 Perception of Experienced productivity
service capacity
Perceived Labor Productivity
Desired Labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
experience rate
Experienced Personnel
effective labor fraction

**25** Capacity adjustment decision (backlog)
Service Backlog
desired service capacity
Desired Labor
labor correction
desired hiring
indicated labor order rate
labor order rate
Vacancies
hiring rate
Rookies
effective labor fraction
service capacity
potential order fulfillment
order fulfillment

Figure 11. Loop hierarchy of SILS in Table 2


A
Loop Inclusion Graph (by levels)
4 [2]
3 [2]
6 [2]
5 [2]
1 [2]
11 [5]
2 [2]
16 [8]
10 [4]
18 [9]
15 [8]
22 [11]
25 [14]
14 [7]
13 [6]
7 [3]
8 [3]
21 [11]
20 [10]
19 [10]
24 [13]
17 [9]
12 [6]
9 [4]
23 [12]
Loop#
[Loop length]
B
Loop Inclusion Graph
Loop length
Mean distance form center of cycle
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5
10
15
25
24
23
22
21
20
19
18
17
16 15
14
13
12
11
10
9
8
7
6
5
4
3
2
1