Building steady-state simulators via hierarchical feedback decomposition

Nicolas Rouquette
Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive, M/S 525-3660
Pasadena, CA 91109
Nicolas.Rouquette@jpl.nasa.gov

Abstract

In recent years, compositional modeling and self-explanatory simulation techniques have simplified the process of building dynamic simulators of physical systems. Building steady-state simulators is, conceptually, a simpler task consisting in solving a set algebraic equations. This simplicity hides delicate technical issues of convergence and search-space size due to the potentially large number of unknown parameters. We present an automated technique for reducing the dimensionality of the problem by 1) automatically identifying feedback loops (a generally NP-complete problem), 2) hierarchically decomposing the set of equations in terms of feedback loops, and 3) structuring a simulator where equations are solved either serially without search or in isolation within a feedback loop. This paper describes the key algorithms and the results of their implementation on building simulators for a two-phase evaporator loop system across multiple combinations of causal and non-causal approximations.

Introduction

Recent advances in model-based reasoning have greatly simplified the task of building and using dynamic simulators of physical systems (Nayak 1993; Forbus & Falkenhainer 1995; Amador, Finkelstein, & Weld 1993). While the usefulness of dynamic simulators is well established in various fields from teaching to high-fidelity simulation, steady-state simulators are characterized by low computational requirements (i.e., that of solving a set of equations only once) which makes them attractive for a wide range of engineering analyses such as stress tolerance, sensitivity, and diagnosis (Biswas & Yu 1993). However, building steady-state simulators can be a challenging task dominated by issues related to the existence of numerical solutions, the physical interpretability of the solutions found and the convergence properties of the simulator (Manocha 1994).

Building a steady-state simulators is conceptually a simple task, that of solving N algebraic equations in $M < N$ unknown parameters with respect to $N - M$ known parameter values. This simplicity hides the computational and numerical task of efficiently and accurately searching a solution in a space as large as M dimensions. There are two extreme approaches for solving a set of algebraic equations: the brute-force approach uses an algorithm to search the numerical solution in the M-dimensional space of possible values; the clever approach seeks to identify closed-form algebraic formulae for computing the unknown parameters in terms of the known values. This paper presents an intermediate approach relying on an automated technique for reducing the dimensionality of the original search space thereby greatly simplifying the task of selecting numerical algorithms and initial solution estimates. This is achieved by 1) automatically identifying feedback loops, 2) hierarchically structuring a equation solver where groups of equations are solved either serially or independently from each other and 3) structurally merging the modeler's choice of algorithms and initial estimates with the feedback decomposition to build the steady-state simulator.

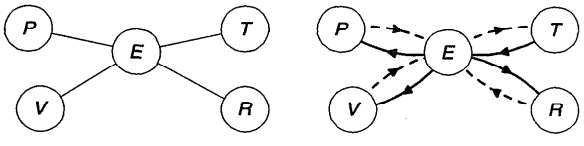
The construction of high-performance hierarchical steady-state simulators is organized as an operationalization process transforming the steady-state model into numerical simulation software. The former is a non-computable, unstructured, conceptual specification of the latter which is is a computable, structured, pragmatic description of the former. In doing so, we elicit engineering understanding of feedback loops related to closed-loop circuits (physical loops) or interdependent equations (algebraic loops). In varying modeling assumptions, we earnmark conceptual progress in tuning modeling assumptions not only to the purpose of the model and the conditions of the physical system but to the various numerical and physical aspects of the simulator (initial estimates, speed of convergence, and physical interpretability of solutions). Therefore this approach enables modelers to progress on conceptual and cognitive fronts; the alternance of which is characteristic of the cyclic nature of the modeling process from theory formation and revision, to experimentation, evaluation and interpretation as described in (Aris 1978). Like all modeling tools, efficiency is a practical concern. Consequently, we have limited the hierarchical feedback decomposition technique to a

tractable domain of models where the algorithms have polynomial-time complexity and efficient implementations.

As an unstructured collection of equations, a model is computationally unoperational for there is no indication of which values must be computed when. The first step of operationalization consists in determining a partial ordering of computations; a process we call algebraic ordering whose emphasis on numerical computability distinguishes it from a physical notion of causation behind the process of causal ordering. We first presents a very efficient algorithm for computing algebraic orderings as a maximum flow problem through a network. This ordering produces a graph of algebraic dependencies among parameters. The key contribution of this paper is in an algorithm that decomposes parameter dependency graphs to elucidate the hierarchical feedback structure of the equation model. Knowledge of this structure can be exploited in many ways; we show here how it serves the purposes of constructing low-cost, high-performance steady-state simulators.

Algebraic ordering

In numerical analysis, it is well known that there is no general-purpose, universally good numerical equation solving algorithm (Press *et al.* 1992), there is only an ever-growing multitude of algorithms with various abilities in specific domains and for specific types of equations. For steady-state simulation, this variety poses a pragmatic problem; choosing the right algorithm for the job becomes more and more difficult as the size of the model grows. To efficiently reason about the possible ways to construct a simulation program for a given set of algebraic equations, we define the notion of an *algebraic ordering graph* that captures how each parameter of the model algebraically depends on the values of other model parameters via the algebraic model equations. Our notion of algebraic ordering bears close resemblance to that of causal ordering (Nayak 1993; Iwasaki & Simon 1993) and relevance in modeling (Levy 1993). These three notions of ordering share a common representation where an equation, E: $PV = nRT$, yields the following parameter-equation graph (left):



where edges indicate possible relations of physical causality and algebraic relevance between parameters and equations. Here, we focus on algebraic computation instead of physical causality and we explicitly distinguish two types of relations (above right). One where an equation can compute a parameter value (solid edges) and another where an equation needs other parameter values to perform such computations (dashed edges).

This allows us to distinguish several ways to numerically compute parameter values. An equation e can *directly* constrain a parameter value p if e is algebraically solvable with respect to p. An equation e *indirectly* constrains a parameter value p if e is not algebraically solvable with respect to p. For example, the equation: $y = \sqrt{x}$ can directly constrain y for a given x since the solution is unique. This equation indirectly constrains x for a given y since there two possible solutions in x.

Although, it is preferable to compute all model parameters through direct constrainment, it is not always possible to do so. The combinations of possible direct and indirect constrainment relationships lead to three categories of parameter constrainment: 1) A parameter may not have any equation directly constraining it; we say it is *under constrained* because its solution value must be guessed as there is no way to directly compute it. 2) When a parameter is directly constrained by exactly one equation, we say it is *properly constrained* because its value is unambiguously computed by solving a unique equation. 3) When multiple equations directly constrain the same parameter, we say it is *over constrained* because there is no guarantee that all such equations yield the same numerical value unless other parameters of these equations can be adjusted.

An equation can only be solved with respect to a single parameter; thus, there are only two possible constrainment categories: 1) A *properly constrained* equation e of n parameters properly constrains exactly one parameter p if p is computed numerically or analytically by solving e with respect to values for the other $n-1$ parameters. 2) An equation e of n parameters which constrains no parameter is said to be *over constrained*: there is no guarantee that the values given to the n parameters satisfy e unless some of the parameter values can be adjusted.

We have established a validity test for a set of equations and parameters which determines when a set of indirect and direct constrainment relationships is solvable (See Ch. 4 in (Rouquette 1995)). If all parameters and equations were properly constrainable, then a bipartite matching approach (e.g., (Nayak 1993; Serrano & Gossard 1987)) would suffice to establish a valid order of computations. To account for possible over and under constrainment, we defined an extended bipartite matching algorithm which ensures that each case of over constrainment is balanced by an adequate number of adjustable under-constrained parameters thereby resulting in a valid, computable ordering.

Extended bipartite matching

Algorithm 1 constructs a network flow graph F to match parameters and equations (Step 1 and 2). Step

3 creates paths between s and t for each exogenous parameter. The key difference with Nayak's algorithm is in the construction of paths corresponding to the possible constrainment relationships among equations and parameters. By default, equation e_j could be solved iteratively to find the value of one of its parameters $p_i \in P(e_j)$ ($p_i \rightarrow e_j^{\text{indirect}} \rightarrow e_j$). For a given e_j, at most one $p_i \in P(e_j)$ can be computed in this manner. If an equation e_j can properly constrain a parameter p_i, then there is a path: $p_i \rightarrow e_j^{\text{direct}} \rightarrow e_j$ in F (Step 4). Since all paths have unit capacity, the paths $p_i \rightarrow e_j^{\text{direct}} \rightarrow e_j$ and $p_i \rightarrow e_j^{\text{indirect}} \rightarrow e_j$ for all p_i's and e_j's are mutually exclusive. This property confers to a maximum flow the meaning of a bipartite matching between the set of equations and parameters (step 5) as is also used in Nayak's causal ordering algorithm. Further, the costs associated to paths allow a maximum flow, minimum cost algorithim to optimize the use of direct computations as much as possible. Finally, the results of the matching are used to define the edges of the algebraic ordering graph (step 7). For models where the equations are causal, it follows that an algebraic ordering is identical to a causal ordering when every non-exogenous parameter is directly computed.

As an example, we consider the following hypothetical set of algebraic equations:

e_1	$f_1(P_1, P_2, P_3, P_8) = 0$	ex_5	exogenous(P_5)
e_2	$f_2(P_2, P_7) = 0$	ex_6	exogenous(P_6)
e_3	$f_3(P_3, P_4) = 0$	e_7	$f_7(P_4, P_6, P_7) = 0$
e_4	$f_4(P_4, P_5) = 0$	e_8	$f_8(P_5, P_8) = 0$

Suppose that e_2 is solvable in P_7 but not P_2 and that e_4 is solvable in P_5 but not P_4. The extended bipartite matching graph for this example is shown in Fig. 1.

Intuitively, Alg. 1 combines the idea of using a perfect matching as a validity criteria and the flexibility of both direct and indirect computations. Edges of the form (e, p) represent direct computations where the value of p is computed by e as a function of some arguments. Edges of the form (p, e) represent indirect computations where the value of p is constrained by e: the solution value of p is computed by search.

If the algebraic ordering graph were acyclic, a topological ordering would define an adequate order of computations. With cycles, the key to globally order computations is to relate the topological structure of the graph to feedback loops.

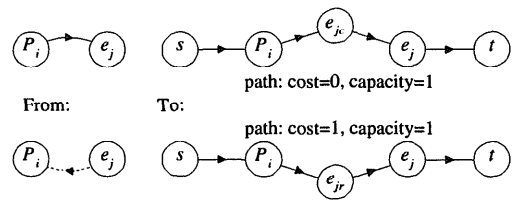
Feedback

Feedback is a property of the topological interdependencies among parameters.

Parameter Dependency Graph

Definition 1 (Algebraic dependency) *A parameter p' depends on p, noted by $p \rightsquigarrow p'$, iff there exists an equation e such that $p \in P(e)$ and $p' \in P(e)$*

Input: A parameter-equation graph $G = (V, A)$
Output: A predicate: $EBM(e, p)$ for $e \in E$ and $p \in P(e)$.
1) Create a network flow graph $F = (V_f, A_f)$.
2) $V_f = V \bigcup \{e^{\text{direct}}, e^{\text{indirect}} \mid e \in E\} \bigcup \{s, t\}$
(s and t are respectively the source and sink vertices)
3) $A_f = P_f \bigcup E_f$ where:
$P_f = \{(s, p) \mid p \in P\} \bigcup$
$\{(p, ex_p), (ex_p, t) \mid p \in P \wedge \text{exogenous}(p)\}$
$E_f = \{(e^{\text{direct}}, e), (e^{\text{indirect}}, e), (e, t) \mid e \in E\}$
Each path $s \rightarrow p \rightarrow ex_p \rightarrow t$ (p exogenous) has unit flow capacity and zero cost.
4) Edges between p_j and $e_{ic} = e_i^{\text{direct}}$ and $e_{ir} = e_i^{\text{indirect}}$ are defined as follows:



5) Apply a min. cost, max. flow algorithm on F
Nonzero transhipment nodes are:
- the source, s, with $b(s) = |P|$
- the sink, t, with $b(t) = -|P|$.
6) If $f(s, t) < |P|$ then return $\emptyset$
7) Define $EBM()$ from the maximum flow topology:
7a) $EBM(P_i, e_j, \text{indirect})$ holds iff $f(P_i, e_j^{\text{indirect}}) = 1$
7b) $EBM(P_i, e_j, \text{direct})$ holds iff $f(P_i, e_j^{\text{direct}}) = 1$
8) Return $EBM()$

Algorithm 1: Extended bipartite matching for constructing an algebraic ordering.

(i.e., p and p' are parameters of e) and $EBM(p', e, \text{direct})$ holds.

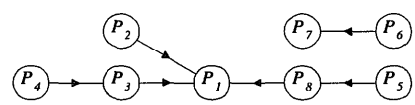
The dependency digraph $Gd = (Pd, Ed)$ corresponding to an algebraic ordering digraph $G' = (V = P \bigcup E, A')$ is defined as follows:

- $Pd = \{p \mid p \in P \wedge \neg \text{exogenous}(p)\}$
- $Ad = \{(p, p') \mid p, p' \in Pd \wedge p \rightsquigarrow p'\}$

For notation convenience, we say that $p \rightsquigarrow^* p'$ when there exists a sequence of parameters, $p = p_1, \ldots, p_n = p'$ such that

$$p = p_1 \rightsquigarrow p_2 \cdots p_{n-1} \rightsquigarrow p_n.$$

For the 7-equation example, we have the following parameter-dependency graph:



where parameters P_4 and P_2 are under-constrained while P_1 is over constrained. The validity of this

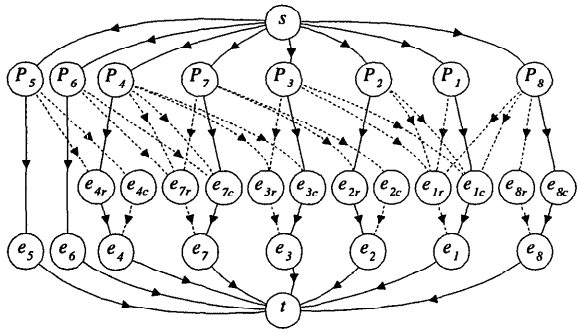
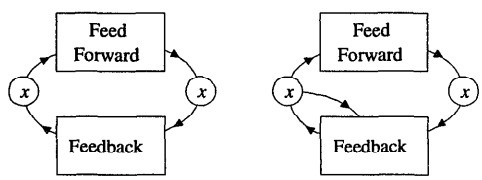


Figure 1: Extended bipartite matching for the 7-equation example. The minimum cost, maximum flow solution is drawn with solid edges.

ordering stems from the proper balancing between over and under-constrained parameters. Indeed, there are 3 ways to compute a value of P_1, two of which ($P_4 \rightarrow P_3 \rightarrow P_1$, $P2 \rightarrow P_1$) can be relaxed to match the value derived from the exogenous parameter P_5 ($P_5 \rightarrow P_8 \rightarrow P_1$).

Hierarchical Feedback Decomposition

Intuitively, feedback occurs when there exists at least two parameters p and p' in the dependency graph Gd such that $p \rightsquigarrow^* p'$ and $p' \rightsquigarrow^* p$ hold in Gd. Feedback is described by various terms in various scientific disciplines and engineering fields. Terms such as closed-loop circuit (as opposed to an open-loop circuit), circular dependencies, closed-loop control, circular state dependencies, and state or control feedback are commonly used. Here, we follow some basic ideas of system theory (Padulo & Arbib 1974) and concepts of connectedness of graph theory (Even 1979) to distinguish two types of feedback structures, namely state (left) and control (right) as shown below:



In a state feedback loop, input parameters, x, affect the output parameters, y, through a feedforward transformation. In a dependency graph, we will have: $x \rightsquigarrow^* y$. The feedback transformation in turns makes the inputs x dependent on the outputs y, or: $y \rightsquigarrow^* x$. A control feedback loop is similar to a state feedback loop except that the feedback transformation (usually the controller) uses both inputs x and outputs y inputs, i.e., $x, y \rightsquigarrow^* x$.

Operationalizing Feedback

Except for degenerate cases, a feedback loop must be solved iteratively for it corresponds to a system of $N \geq 2$ equations in N unknown parameters. Optimizing the solution quality and its computational cost requires making a number of choices for each feedback loop in terms of numerical algorithms, initial solution estimates, and convergence criteria. Addressing these issues globally can be very difficult. With a decomposition of the model in terms of a hierarchy of feedback loops, we can address these issues in two phases: one for the model subset corresponding to a given feedback loop and another for the structure of the model encompassing this loop. Typically, the former focuses on finding a solution for the feedback loop while the latter addresses convergence issues at a global level.

Unfortunately, identifying feedback in an arbitrary graph is an NP-complete problem. Fortunately, lumped-parameter algebraic models of physical systems are typically sparse (due to lumping) and have a low degree of connectivity (because most physical components have limited interactions with neighbor components). Combined with the fact that most man-made devices are often engineered with closed-loop control designs, it is quite common for the corresponding dependency graphs of such models to be decomposable in terms of feedback loops.

Breaking feedback loops apart

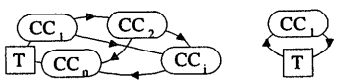
The algebraic dependency relation defined for Gd induces an equivalence relation. By definition, two parameters p_1 and p_2 are in the same equivalence class iff $p_1 \rightsquigarrow^* p_2$ and $p_2 \rightsquigarrow^* p_1$. These relations are characteristic of state and control feedback loops. Thus, feedback loops are strongly-connected subgraphs of Gd; the converse is not true[1]. Thus, we now define a structural criterion for recognizing feedback loops. In (Even 1979), a set of edges, T, is an (a, b) *edge separator* iff every directed path from a to b passes through at least one edge of T. Then for a strongly-connected component C, consider the smallest T such that, $C - T$ is either unconnected or broken into two or more strongly-connected subcomponents. For a given pair, a, b, we call such a subset T a *one-step optimal edge separator.*[2]

With the one-step optimal edge separator, we can solve a restricted version of the feedback vertex problem in polynomial time. Algorithm 2 shows how to remove optimal edge separators to analyze the topological structure of a graph G. The algorithm stops if G is not separable with the optimal edge separator (step 1). Consider $G' = (V, A - T)$. By definition of an optimal edge separator, T will break

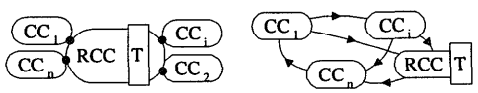
[1] A fully-connected graph is not a feedback loop.

[2] One-step because we make a single analysis of how removing T affects the strong connectivity of the given subgraph. See (Rouquette 1995, Ch. 6) for a polynomial-time algorithm.

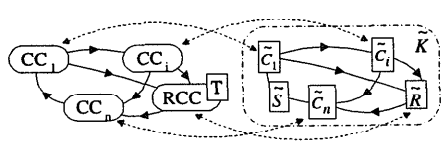
apart the strongly-connectedness of G. If G' is no longer strongly-connected, G is in fact a simple feedback loop (step 3). If G' is still strongly-connected, we need to analyze the remaining structure of G'. First, we remove the strongly-connected components already found (step 4).[3] Let G'' be the remaining subgraph (step 5). We consider two cases according to the connectivity of G''. If G'' is not strongly connected, G has a 2-level hierarchical structure (if there are 2 or more sub-components) or a 2-level nested structure (if there is only one sub-component) (step 7).



If G'' is strongly connected, it must have a single component, RCC[4]. Topologically, either RCC only shares vertices with the other sub-components already found (i.e.,CCS) (step 8) or it is distinct from them.



In the latter case, we need to abstract the sub-component already found $CC_1, \ldots, CC_n, RCC$ into equivalence class vertices $\widetilde{C_1}, \ldots, \widetilde{C_n}, \widetilde{R}$ so that we can further analyze the remaining structure of G'' (step 9). Since G was strongly-connected, $\widetilde{G}$ is also strongly-connected (step 10). The algorithm stops if this abstract component, $\widetilde{K}$, is not decomposable (step 12). Otherwise, we map to the base level graph G the optimal edge separator $\widetilde{S}$ that breaks apart $\widetilde{K}$ (step 13).



Note that the DAD algorithm is recursive for we also need to analyze the structure of the strongly-connected sub-components of G found (steps 7,8,14). The recursion stops if a strongly-connected component has either 1) the structure of a state or control feedback loop (step 3) or 2) a more complex structure than that of a feedback loop (steps 1,12).

Hierarchical Feedback Example

[3]We use the notation $A(X)$ to mean 'the set of edges of the subgraph X'.

[4]This follows from having removed all components found earlier (step 5) and from the nature of an optimal edge separator.

Input: $G = (V, A)$, a strongly-connected digraph
Output: The hierarchical feedback tree (HFT) decomposition of G

1) Let T be a one-step optimal edge separator of G; HFT=**ComplexFeedback**(G) if $T = \emptyset$.
2) $G' = (V, A - T)$ (remove T from G).
3) HFT=**Feedback**(G,T) if G' is not strongly connected.
4) Let $CCS = \{CC_1, \ldots, CC_n\}$ be the remaining strongly-connected components of G'
5) $G'' = \left(V, A - \left(\bigcup_{cc \in CCS} A(cc)\right)\right)$ (remove CCS from G)
6) If G'' is strongly-connected, go to step 8.
7) HFT=$\begin{cases} \textbf{Agg}(G, T, \bigcup_{cc \in CCS} DAD(cc)) & \text{if } n > 1 \\ \textbf{Nested}(G, T, DAD(CC_1)) & \text{if } n = 1 \end{cases}$
8) G'' has a single strongly-connected component RCC. HFT= $\textbf{Agg}(G, T, DAD(RCC) \bigcup \bigcup_{cc \in CCS} DAD(cc)))$ if $A(G'') - A(RCC) = \emptyset$.
9) Let $\widetilde{C}_i$ be an abstract vertex representing CC_i. Let $\widetilde{R}$ be an abstract vertex representing RCC. Let $\widetilde{G} = (\widetilde{V}, \widetilde{A})$ the abstract graph of G $\widetilde{V} = \{\widetilde{C}_1, \ldots, \widetilde{C}_n, \widetilde{R}\}$. $\widetilde{A}$ is defined according to paths among $CCS \bigcup \{RCC\}$.
10) Let $\widetilde{K}$ be the strongly-connected component of $\widetilde{G}$. Let $\widetilde{S}$ be the optimal edge separator of $\widetilde{K}$
12) HFT=**ComplexFeedback**(G) if $\widetilde{S} = \emptyset$.
13) Let $S \subset A$ be the base edges corresponding to $\widetilde{S}$.
14) HFT= $\textbf{Aggr}(G, S, \bigcup_{cc \in CCS} DAD(cc) \bigcup DAD(RCC))$

Algorithm 2: DAD: Decomposition and Aggregation of Dependencies

As an illustration example, we show in Fig 2 a schematic diagram of the evaporator loop of a two-phase, External-Active Thermal Control System (EATCS) designed at McDonnell Douglas. Liquid ammonia captures heat by evaporation from hot sources and releases it by condensation to cold sinks. The venturis maintain a sufficiently large liquid ammonia flow to prevent complete vaporization and superheating at the evaporators. The RFMD pump transfers heat between the two-phase evaporator return and the condenser loop (not shown). A model of the EATCS presented in (Rouquette 1995) yields a parameter-dependency graph of 55 parameters, 18 exogenous and

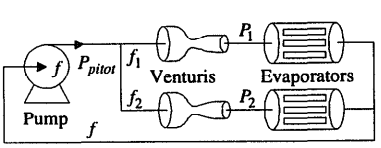


Figure 2: The evaporator loop of the External-Active Thermal Control System.

37 unknowns paired to 37 equations. A brute-force simulation approach consists in solving the 37 equations for the 37 unknown parameters at the cost of finding 37 initial value estimators for each unknown. The DAD algorithm finds a 2-level feedback decomposition shown in Fig. 3.

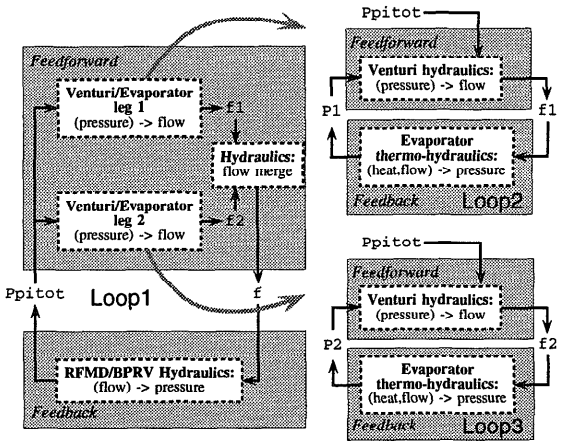
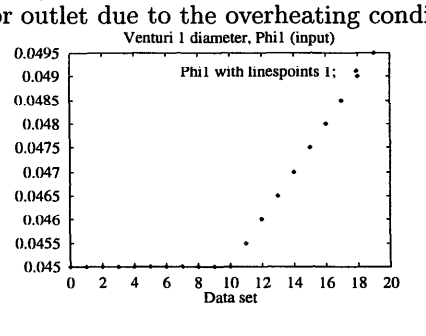


Figure 3: Physical and nested algebraic feedback loops.

With this feedback hierarchy, we now turn to producing a steady-state simulation, i.e., an equation solver for all the model equations. The modeler needs to choose for each feedback loop which subset of parameters will characterize its state[5] with the constraint that state parameters must be a graph cutset of the feedback loop[6]. Other issues influence the choice of a feedback cutset as a state vector: numerical convergence, stability and speed. For example, in Fig. 3, the modeler chose the pressures at each loop, namely, P_1, P_2, and P_{pitot}, because the pressures have the widest range of behavior across possible states. Flow rates would be a poor choice because they are mostly constant during nominal circumstances.

To produce the final hierarchical equation solver, the modeler must provide for each feedback loop the following information: 1) a state vector of parameters; 2) an initial function to compute the initial state vector values (this function can only use the exogenous parameters relative to the feedback loop.) and 3) a numerical algorithm to find the final feedback parameter values from any state vector estimate[7] Without decomposition, the equation solver has all unknown parameters to handle simultaneously. The gradient-descent approach (?) is a method to guess where the solution may be and focus the search towards there. This essentially amounts to determining, at problem-solving time, how to prioritize the unknown parameters to work on. In contrast, the hierarchical decomposition determines these priorities once and for all at compile time. With hierarchical decomposition, higher-level feedback loops effectively act as constraints on the possible values lower-level feedback parameters can take. This process is similar to the gradient-descent techniques used in numerical algorithms. The key difference is that a gradient-descent algorithm is continuously guessing the direction where the solution is. With hierarchical-feedback decomposition, there is no guesswork about the whereabouts of the solution; the hierarchical equation solver is built to find it in a very organized manner specified at compile-time instead of run-time.

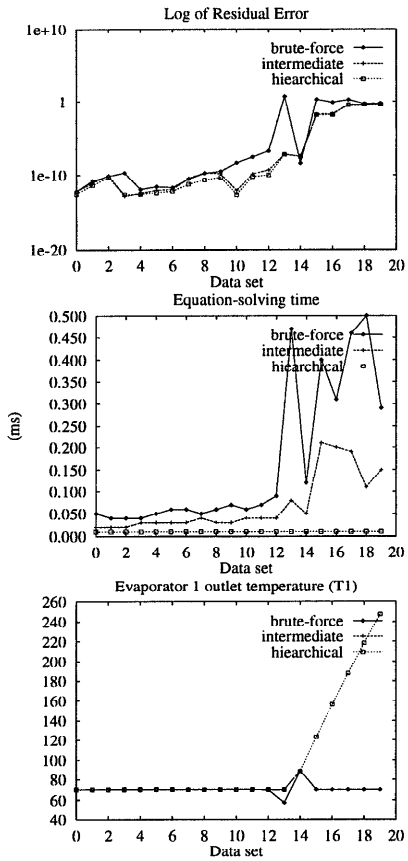
For the EATCS, we start at Loop1. From P_{pitot} we compute new estimates of the lower-level feedback loop states (P_1, P_2). Then, Loop2 and Loop3 refine P_1 and P_2 to satisfy the algebraic feedback equations. With this decomposition, search costs are effectively divided among multiple feedback loops. Furthermore, we are spared from continuously evaluating the next-best direction to go as is done with gradient descent. The charts below show experimental results demonstrating that despite the lack of flexibility in determining the next-best parameters to adjust, the decomposition approach finds solutions of equal quality at much lower computational cost, even for difficult solutions. For each chart, we considered a series of 21 exogenous conditions defined by pump speed, venturi diameters (to analyze clogging conditions) and evaporator load. As long as the model converges to a nominal state for the EATCS, all three solvers are practically equivalent (data sets 0 through 12). Data sets 13 and above correspond to overload conditions where the heat applied is greater than what the evaporator loop can circulate. In such cases, the initial estimates are quite far from the actual abnormal solutions which implies more search. In fact, the brute-force and intermediate solvers spend several orders of magnitude more time searching only to find wrong solutions. Only the hierarchical solver managed to predict the temperature increase at the evaporator outlet due to the overheating condition.



[5]The state parameter common to control and state feedback structures is a good candidate.

[6]i.e.,if state parameters are removed, the connectivity of the loop is broken.

[7]See (Rouquette 1995) for algorithms to generate C hierarchical equation solvers based on the above information.



Conclusion

To describe the possible ways for solving a set of parameters from a set of algebraic equations, we presented the notion of algebraic ordering which is equivalent to causal ordering if all equations are believed to be causal. From an algebraic ordering, we constructed a parameter-dependency graph and described a decomposition algorithm based on analyzing the topological structure of the dependencies in terms of its strongly-connected components. By carefully choosing how to break apart such components, we showed how to construct the hierarchical decomposition of the dependency graph in terms of state and control feedback structures. Once the modeler chooses state feedback parameters and initial estimators for them, the set of all equations is solved bottom-up by applying a chosen equation solver according to the hierarchical feedback decomposition found. Compared to knowledge-free gradient-descent approaches to equation solving, our knowledge-intensive approach seeks to elucidate knowledge about feedback from the model itself to help the modeler provide as much relevant equation-solving knowledge as possible in terms of initial solution estimates and convergence metrics. Experimentally, this produced faster, better, and cheaper simulation programs trading off an expansive and broad search space (brute force approach) for a narrow, structured search space (fewer independent parameters) thereby achieving greater computational efficiency without loss of accuracy.

References

Amador, F.; Finkelstein, A.; and Weld, D. 1993. Real-time self-explanatory simulation. In *Proceedings of the Eleventh National Conference on Artificial Intelligence.* The AAAI Press.

Aris, R. 1978. *Mathematical Modeling Techniques*, volume 24 of *Research notes in mathematics.* Pitman.

Biswas, G., and Yu, X. 1993. A formal modeling scheme for continuous systems: Focus on diagnosis. In *International Joint Conference on Artificial Intelligence*, 1474–1479.

Even, S. 1979. *Graph Algorithms.* Computer Science Press.

Forbus, K., and Falkenhainer, B. 1995. Scaling up self-explanatory simulators: Polynomial-time compilation. In *International Joint Conference on Artificial Intelligence*, 1798–1805.

Iwasaki, Y., and Simon, H. A. 1993. Retrospective on causality in device behavior. *Artificial Intelligence Journal* 141–146.

Levy, A. 1993. *Irrelevance Reasoning in Knowledge Based Systems.* Ph.D. Dissertation, Department of Computer Science, Stanford University.

Manocha, D. 1994. Algorithms for computing selected solutions of polynomial equations. *J. Symbolic Computation* 11:1–20.

Nayak, P. 1993. *Automated Modeling of Physical Systems.* Ph.D. Dissertation, Department of Computer Science.

Padulo, L., and Arbib, M. A. 1974. *System Theory.* W. B. Saunders Company.

Press, H.; Teukolsky, S.; W.Vetterling; and Flannery, B. 1992. *Numerical Recipes in C.* Cambridge University Press.

Rouquette, N. 1995. *Operationalizing Engineering Models of Steady-State Equations.* Ph.D. Dissertation, Dept. of Computer Science, Univ. of S. California.

Serrano, D., and Gossard, D. 1987. Constraint management in conceptual design. In Sriram, D., and Adey, R., eds., *Knowledge Based Expert SYstems in Engineering: Planning and Design.* Computational Mechanics Publications. 211–224.