In *The 14th National Conference on Artificial Intelligence (AAAI-97)*
AAAI Press, Providence, Rhode Island, August 1997.

Model Decomposition and Simulation: A component based qualitative simulation algorithm *

Daniel J. Clancy and Benjamin Kuipers
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
clancy@cs.utexas.edu and kuipers@cs.utexas.edu

Abstract

Traditionally, qualitative simulation uses a global, state–based representation to describe the behavior of the modeled system. For larger, more complex systems this representation proves extremely inefficient since it provides a complete temporal ordering of all potential distinctions leading to a large, complex behavioral description that obscures relevant distinctions, or even fails to terminate.

The model decomposition and simulation algorithm (DecSIM) uses a divide and conquer approach to qualitative simulation. Variables within the system are partitioned into components. Each component is viewed as a separate system and is simulated using a state–based representation limited to the variables within the component. Interactions between components are reasoned about separately. DecSIM provides a promising paradigm for qualitative simulation whose complexity is driven by the complexity of the problem specification rather than the inference mechanism used.

Introduction

In *The Sciences of the Artificial*, Herb Simon (1969) observed that "systems in which each variable is linked with almost equal strength are rare." He introduces the idea of a *nearly decomposable system* as one in which the interactions among subsystems are weak but not negligible and suggests that tractable reasoning requires people to reason about the complex interactions of a sub-system independent of the interactions between subsystems.

Qualitative simulation (De Kleer and Brown, 1984; Forbus, 1984; Kuipers, 1994), however, traditionally describes a dynamical system via a single model of interacting constraints reasoning about its behavior using a global, state–based behavioral representation. For smaller, more tightly constrained models, this representation proves adequate, often resulting in a small number of behaviors and a tractable simulation. As the size of a model increases, however, the model tends to become more loosely constrained. Unrelated distinctions in distant variables combine combinatorially often resulting in intractable branching. Often, these distinctions are irrelevant to the current task and tend to obscure other relevant distinctions in the behavioral representation.

An alternative representation first proposed by Hayes (1985) and later used by Williams (1986) describes the behavior of each variable as a set of independent variable *histories*. Relevant temporal relations between variables are described separately. A history–based representation, however, does not lend itself to reasoning about the interactions between closely related variables when performing a simulation.

The Model Decomposition and Simulation (DecSIM) qualitative simulation algorithm bridges the gap between a history–based and a state–based representation. Variables within the model are partitioned into loosely coupled components and each component is simulated separately using a state–based simulation algorithm. Interactions between components are reasoned about as needed to constrain each component. Thus, the relationship between variables within the same component is state–based while the relationship between different components is history–based.

The DecSIM algorithm can be separated into the simulation algorithm and the variable partitioning algorithm. This paper focuses on the simulation algorithm and assumes that the variable partitioning is provided by the modeler. The simulation is designed to provide the same soundness guarantees and the same

*This work has taken place in the Qualitative Reasoning Group at the Artificial Intelligence Laboratory, The University of Texas at Austin. Research of the Qualitative Reasoning Group is supported in part by NSF grants IRI-9504138 and CDA 9617327, by NASA grants NAG 2-994 and NAG 9-898, and by the Texas Advanced Research Program under grant no. 003658-242.

degree of constraining power as a standard state–based qualitative simulation for arbitrary variable partitionings.

By partitioning the model into smaller chunks, DecSIM significantly reduces the overall complexity of the simulation by eliminating the temporal correlations between variables in different components. Given an appropriate partitioning of the variables, the complexity of the DecSIM algorithm becomes a function of the problem specification rather than an artifact of the simulation algorithm.

Qualitative Simulation and DecSIM

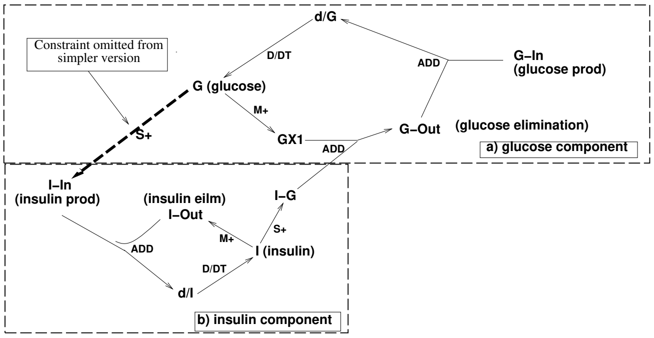
DecSIM uses the QSIM qualitative simulation algorithm (Kuipers, 1994) as its core inference mechanism when simulating an individual component. QSIM uses a qualitative differential equation (QDE) to specify constraints modeling the structural relationships between the variables within a dynamical system. The behavior of the system is described using a tree of alternating time–point and time–interval states. Variables are described using a qualitative magnitude and a direction of change. The qualitative magnitude is defined upon a quantity space of totally ordered landmarks identifying the relevant distinctions for that variable. An *event* occurs when a variable reaches a landmark or becomes steady. Branches occur within the behavioral description whenever multiple events can occur following a time–interval state. This is referred to as *event branching*. Event branching is exponential in the number of unrelated events. As a model grows, the number of loosely or unrelated events tends to grow, thus resulting in an intractable simulation.

Model decomposition uses a divide and conquer approach to control the problem of event branching. The variables within the model are partitioned into components so that closely related variables are contained within the same partition. A sub-model is created for each partition describing the relationships between the variables within the partition. This sub–model, or *sub–QDE,* is used to simulate the component. Two types of variables are contained within each sub-QDE.

Within-partition variables are the variables belonging to the partition. These are the variables of interest for this sub-QDE.

Boundary variables are non-partition variables whose behavior affects the overall behavior of the component. These are variables that are related to within–partition variable via a constraint within the original QDE.

Components containing a boundary variable are called *boundary components*. A sub-QDE is comprised of the constraints within the main QDE which contain only within-partition and boundary variables. Figure 1 describes the partitioning of variables and creation of sub-QDEs for two different versions of a model of the human glucose–insulin regulatory (GIR) system developed by Ironi and Stefanelli (1993).



The models of the human Glucose-Insulin Regulatory System (GIR) contain two connected feedback loops corresponding to the glucose (a) and insulin (b) regulatory systems.

- The nodes in the graph are variables and the links are constraints. The arrows indicate the direction of causality derived via a causal analysis.
- Two versions are used within the discussion. In the simpler version, the constraint between `G` and `I-In` represented by the bold, dotted line is omitted and `I-In` is assumed to be constant.

The model is partitioned into two components. The boxes delineate the within–partition variables for each component.

- In the simpler version of the model, the insulin component is causally upstream and is simulated prior to and independent of the glucose component.
- In the more complex version, `G` is a boundary variable of the insulin component, and the two components are simulated concurrently.

Figure 1: Two Models of the Glucose/Insulin Regulatory System

A causal analysis (Iwasaki, 1988; Nayak, 1992) is used to identify the relationships between components. Causally upstream components are simulated prior to the simulation of the downstream components. A variable is only considered a boundary variable with respect to a component C, if it is causally upstream or acausally related to a variable within C. For the example in figure 1, `I-G` is a boundary variable with respect to the glucose component, but `GX1` and `G-Out` are not boundary variables in the insulin component sub-QDE. Boundary variables are treated as exogenous variables whose behavior is determined by the simulation of the boundary component.

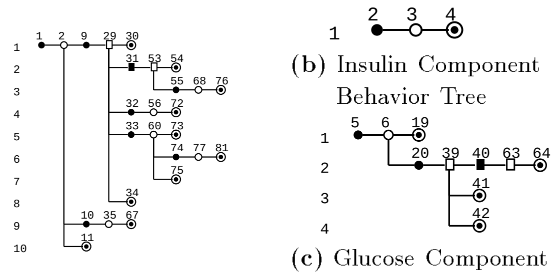
If a feedback loop exists between a set of components (as in the more complex version of the GIR model) or if two components are acausally related (i.e the causal analysis is unable to identify an ordering), then a *concurrent simulation* is used to derive the behavior of these components. When performing a concurrent simulation, the algorithm iterates over the components extending each component a single time–step. When reasoning about the behavior of a boundary variable, the simulation assumes that the behavioral description of the upstream component is complete even though it is still being derived. When a state is marked inconsistent due to a boundary variable (i.e. the value of the boundary variable does not correspond to a state in the component containing the variable), a *dependency* link is recorded. If the simulation of the boundary component is extended in a manner that satisfies the dependency, the state that was previously marked inconsistent is reinserted into the behavior tree and successors of the state are computed.

A Model of the Glucose Insulin Regulatory System

The GIR model contains two separate feedback loops. One controls the amount of glucose (G) within the body and the other controls the amount of insulin (I). Two versions of the model are discussed. In the more complex version, the amount of glucose (`G`) controls the rate of insulin production (`I-In`) while the amount of insulin (`I`) affects the elimination of glucose (`G-Out`) via an intermediate variable (`I-G`). Thus, a negative feedback loop exists between these two components. The simpler version models a pathological condition called glicemia where the rate of insulin production is independent of the amount of glucose within the body.

For the simpler version of the model, a causal ordering exists between the two components. Thus, the insulin component can be simulated to completion prior to beginning the simulation of the glucose component. Figure 2 describes the results from this simulation.

The more complex version of the model results in an infinite behavioral description due to the oscillatory nature of the feedback controlled system. Intractable branching makes it difficult to extend the simulation sufficiently to identify the asymptotic behavior of the system and evaluate the results via a standard QSIM simulation. With a state–limit of 10,000, QSIM generates a total of 3340 behaviors, none sufficiently extended to identify the asymptotic behavior of the system. A DecSIM simulation with a component state limit of 300 generates a behavior tree with 74 behaviors for the insulin component and 84 behaviors for the glucose component describing a broader range of the state space. In the DecSIM simulation, all of the behaviors are sufficiently extended to identify the decreasing oscillatory nature of the behavior. The DecSIM simulation also resulted in a factor of four speed up in the simulation time.



(a) Original Behavior Tree

(b) Insulin Component Behavior Tree

(c) Glucose Component Behavior Tree

- A standard QSIM simulation results in a total of 10 behaviors (a) and 26 states.
- A DecSIM simulation generates a single behavior for the insulin component (b) that is used to guide the behavior of the intermediate variable `I-G` when simulating the glucose component. Simulation of the glucose component results in four behaviors (c).

Figure 2: Simpler Model Simulation Results

The DecSIM Simulation Algorithm

The DecSIM simulation algorithm accepts a QDE and a partitioning of the variables within the QDE as input and generates a behavioral description for each partition. The algorithm can be divided into two main components: decomposing the model into sub-QDE's, and performing the simulation.

Once a partitioning is provided by the modeler, decomposing the model involves creating a sub-QDE for each partition and identifying the within–partition variables, boundary variables and constraints that belong within each sub-QDE as described previously.

A separate QSIM simulation is performed for each sub-model. Three main extensions are required to perform a component–based simulation: 1) **Boundary variable distinctions** must be eliminated via abstraction to avoid branches within a component tree due to non–component variables; 2) the **boundary variable behavior guide** is used to restrict the behavior of boundary variables when simulating a boundary component; and 3) dependency links must be maintained and checked during a **concurrent simulation**.

Eliminating Boundary Variable Distinctions

A standard QSIM simulation provides a total ordering of events for all variables contained within the model. In a partitioned simulation, each sub-model contains both within-partition and boundary variables. Since each sub-model is only required to provide a behavioral description for the within-partition variables, abstraction is used to eliminate branches caused by distinctions in the values of the boundary variables. This eliminates the temporal correlations between within-partition variables and boundary variables during the simulation of a sub-model.

The abstraction process is performed during the simulation after the successors of a state are computed. Equivalent successor states with respect to the within-partition variables are combined to form a single abstract state. There are two main steps in the creation and simulation of abstract states: 1) creating an abstract state from a set of detailed (i.e. non-abstracted) successor states; and 2) computing the successors of an abstract state.

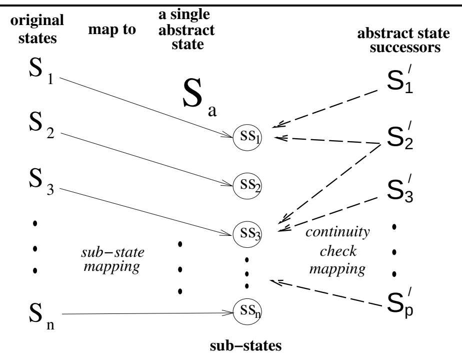
Creating an Abstract State An abstract state contains a unique value for each within-partition variable. Qualitative value information for the boundary variables is maintained in the form of a disjunctive list of *sub-states*. Each sub-state contains a complete set of values for the boundary variables. A sub-state is created for each detailed state used to create an abstract state. The information in the sub-states is used to ensure that no constraining power is lost when the successors of an abstract state are created.

Creating Successors of an Abstract State The abstraction process eliminates distinctions in the boundary variables which are normally used by QSIM when computing a state's successors. The algorithm used to create the successors of a state during a simulation has been modified to handle an abstract state. These modifications retain the QSIM soundness guarantee as well as the constraining power of a standard QSIM simulation. The successors of an abstract state are computed as follows:

Step 1: QSIM uses continuity to determine the possible successor values for each variable within the model. For within-partition variables, the unique qualitative value provided by the abstract state is used to compute the set of valid successor values. For boundary variables, the union of the possible successor values for each distinct boundary variable value within a sub–state is used.

Step 2: Use the standard QSIM algorithm to create and filter potential successor states from this set of possible values.

Step 3: For each successor state, perform a continuity test to ensure that the boundary variable values can be reached from at least one sub-state maintained by the abstract state.



A state is generated as an abstract state successor if and only if it would have been generated as a successor of an original, unabstracted state. (Left to right arrows in figure emphasize procedural sequences.)

- A one-to-one correspondence (the sub-state mapping) exists between the sub-states attached to an abstract state and the original states used to create the abstract state.
- No successors are *omitted* because the possible successor values for each boundary variable is the union of the possible successor values for each sub–state (step 1).
- A continuity check (step 3) ensures that each successor of an abstract state can be reached from at least one of the sub-states. No additional successors are *generated* because each abstract state successor can be mapped back to the original states which it would have succeeded if the abstraction were not performed.

Figure 3: State abstraction does not change the set of successors generated.

Step 1 in this algorithm ensures that the QSIM soundness guarantee is retained since the union of the possible successor values for the boundary variables is used. Step 3 ensures all of the constraining power of the standard QSIM algorithm is retained.[1] Figure 3 shows how each successor of an abstract state would have followed from at least one of the original non-abstracted states.

Boundary Variable Behavior Guide

During the simulation of a component, boundary variables are viewed as exogenous variables whose behavior is completely determined by the components to which they belong. The constraints which restrict the behavior of these variables are contained within these upstream sub-models. The constraints which exist between the boundary variables and the within-partition

[1] Since the union of the possible successor values for the boundary variables is used (in step 1), it is possible that these successor values may combine in ways that would not have been possible with the non-abstracted states. This test ensures that a possible successor value that is valid for one of the sub-states does not combine with a value from a different sub-state for which it is not a valid successor.

variables in the current sub-model serve to restrict the behavior of the within-partition variables. By restricting the behavior of the boundary variables to match the description provided by the upstream sub-models, no constraining power is lost with regard to the within-partition variables.

The boundary variables of a sub-model are grouped into sets according to the components to which they belong. For each boundary component, a *guide tree* is maintained describing the behavior of the boundary variables contained within the component.

A guide tree is an abstraction of a behavior tree focusing on the distinctions of a subset of the variables within the original tree. A branch is introduced in the guide tree only for distinctions in these variables. A one-to-many mapping exists between states in the guide tree and states is the original tree that it abstracts.

The guide trees are used to determine the valid successor values for the boundary variables during the simulation of a downstream component. Each sub–state describing the behavior of the boundary variables is mapped to a state in the guide tree. A state S_g within a guide tree *matches* a sub–state S in the current simulation if and only if:

- S_g is equivalent to S with respect to the boundary variables, and
- the predecessor of S matches either S_g or the predecessor of S_g, and
- if S is a time–interval state then S_g must also be a time–interval state.

A sub-state is marked inconsistent if it cannot be matched against a state within each of the boundary variable guide trees. An abstract state is considered inconsistent if all of its sub-states are marked inconsistent.

This technique for restricting the behavior of a set of variables based upon a predefined behavioral description has been generalized to allow a modeler to control the behavior of any exogenous variable within a QSIM simulation.

Concurrent simulation

A concurrent simulation is used when a feedback loop exists between a set of components or when two components are acausally related. The concurrent simulation alternates between extending the simulation of each of the related components. When reasoning about the behavior of a boundary variable, the concurrent simulation assumes that the behavior tree of the component describing the boundary variable is complete. A *dependency* is recorded when a state is marked inconsistent due to the boundary variable behavior guide. A dependency is a triple of the form *(⟨sub–state⟩ ⟨condition⟩ ⟨guide–state⟩)* and can be interpreted as meaning

> ⟨sub-state⟩ *has been marked inconsistent because a successor of* ⟨guide-state⟩ *satisfying* ⟨condition⟩ *does not exist*

where *⟨condition⟩* contains a set of qualitative values for the boundary variables.

As the behavior of the related component is extended, the dependencies are checked to determine if any of the conditions are satisfied by the additional behavioral information. If a dependency is satisfied, then the sub–state which had previously been marked inconsistent is reinserted into the behavior tree and the successors of its parent state are updated.

The dependencies are checked after each sub-model has been extended one time step. Dependencies are cross checked against each other to ensure that a deadlock condition does not occur (i.e. two states in different components are marked inconsistent because they are each waiting for the other to occur.)

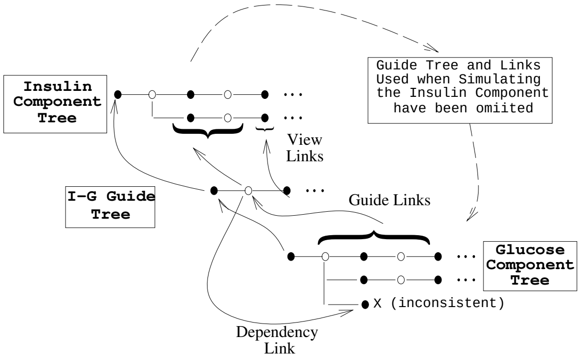
The number of links maintained by the DecSIM algorithm is linear in the number of states within the component trees. Figure 4 provides a summary of these links and trees with respect to the simulation of the glucose component in the GIR model. Using the guide tree simplifies the complexity of these links by avoiding the need to maintain multiple links from each state in the downstream component to multiple states in the upstream component.

Discussion

To gain an understanding of the complexity the DecSIM algorithm, it is best to view the task of qualitative simulation as a constraint satisfaction problem (CSP). Each behavior generated is a solution to the CSP. QSIM attempts to enumerate all possible solutions. Both the number of solutions and the length of each solution are potentially infinite due to the introduction of landmarks. Thus, the complexity of a simulation is best characterized by the branching factor within the tree.

Following a time–interval state, QSIM generates successors based upon different combinations of events that can occur. If the ordering of a set of n events is unconstrained by the model, QSIM generates 2^n successor states. Of course the model usually significantly constrains the set of events that can follow any given state.

DecSIM's decomposition of the model into components is a partitioning of the constraint satisfaction



Three trees are used when guiding the simulation of the glucose component.

- The **insulin component tree** describes all of the variables within the insulin component. The **I-G Guide Tree** is an abstraction of the insulin component tree describing only the behavior of the boundary variable `I-G`. The **glucose component tree** describes the behavior of the variables in the glucose component.

Three types of links are maintained between these trees.

- **View links** map states in the guide tree to their corresponding states in the insulin component tree. **Guide links** map sub–states in the downstream component tree to the corresponding state in the boundary component guide tree. Sub–states are omitted to simplify the diagram. **Dependency links** are recorded when a sub–state is marked inconsistent due to the boundary variable behavior guide. A dependency link points from a guide tree to a component tree and it labeled with the condition that must be satisfied by a successor of the source state if the destination state is to be marked consistent.

Figure 4: Trees and Links Used

problem into smaller more tractable problems using an implicit representation of the solutions rather than an explicit enumeration. DecSIM provides two benefits with respect to the computational complexity of the simulation. First, the total number of events that can occur within any one simulation is decreased by subdividing the model. This feature alone can provide an exponential reduction in the complexity of a simulation. More importantly, however, since the branching factor is dominated by unconstrained events, separating loosely constrained variables into separate components eliminates the primary source of branching. Thus, the simulation of each component may become highly constrained resulting in a small number of solutions for each component.

The benefits provided by the DecSIM algorithm depend highly upon the topology of the constraint network and the degree to which it lends itself to decomposition. Furthermore, the partitioning selected clearly affects the complexity of the simulation. While DecSIM cannot guarantee a tractable simulation for any model, it does provide a simulation algorithm whose complexity is a function of the problem specification rather than an artifact of the simulation algorithm. The advantages of a component simulation become more pronounced as components becomes more tightly constrained and the interaction between components decreases.

Future Work

DecSIM has been tested on a variety of restricted examples producing very promising results as presented in the section describing the GIR example. We are in the process of assembling a corpus of models to provide a thorough empirical evaluation of the the DecSIM algorithm. Due to the limitation of the existing simulation techniques, however, many of the available models tend to be smaller and more tightly constrained. DecSIM opens up the feasibility of simulating models that previously could not be addressed via qualitative simulation. Thus, in addition to obtaining models from researchers within the field of qualitative reasoning, we are also in the process of developing new models to demonstrate the benefits of a component based simulation. The new models are being developed from compartmental models within the fields of biology, medicine and ecology (Jacquez, 1985).

Research is also still required to identify the characteristics of an optimal partitioning and to develop an algorithm to automate model decomposition. Currently, DecSIM requires the modeler to identify a partitioning of the variables. Providing such a partitioning can simply be viewed as part of the model building task. Automating this process, however, will simplify the model building process and be beneficial when used in conjunction with automated model building systems (Rickel and Porter 1997). The problem of partitioning a graph into closely related components has already been extensively studied within fields such as graph theory (Even, 1979) and constraint satisfaction (Tsang, 1993). Developing a partitioning algorithm for DecSIM primarily requires a characterization of the task so that existing research within these fields can be applied.

Related Work

Williams (1986) developed a history–based qualitative simulation algorithm called the Temporal Constraint Propagator (TCP). TCP introduces a compelling paradigm that has been frequently cited within the qualitative reasoning literature. Developing a model using TCP, however, requires a significant amount of work by the modeler and this paradigm has not been extended since publication of the initial results. In addition, TCP uses a history–based representation for the entire model while DecSIM exploits the benefits of both history–based and a state–based representations. DeCoste (1994) also investigated the problem of irrelevant distinctions within qualitative simulation. DeCoste uses a goal–directed simulation technique that only enumerates distinctions that are relevant to the evaluation of the specified goal. Often, however, a modeler is interested in a general description of the potential behaviors of the system. In this case, a specific goal is not available to assist in restricting the simulation.

Coiera (1992) developed a system that can superimpose qualitative predictions from two causally unrelated processes on a single downstream variable. He does not address how these techniques can be applied to more complicated causal interactions or handle phenomenon such as feedback loops.

Clancy and Kuipers (1993) present abstraction techniques that eliminate irrelevant distinctions via post–processing abstraction techniques. While these techniques simplify the process of analyzing the results of a simulation by highlighting relevant distinctions, they do not address the problem of simulation complexity since they are applied following completion of the simulation.

Conclusions

Intractable branching due to irrelevant distinctions is one of the major factors hindering the application of qualitative reasoning techniques to large, real–world problems. Many of these distinctions result from inherent limitations of a global, state–based representation. DecSIM eliminates the need to explicitly enumerate all possible solutions and instead provides a more compact representation that exploits the existing structure within a model. Given an appropriate partitioning of the model, a DecSIM simulation approaches the inherent complexity of the problem specification as opposed to being an artifact of the simulation algorithm dominated by the complexity of the output representation.

The DecSIM algorithm complements recent research that eliminates another well–known source of irrelevant distinctions within a qualitative simulation – chatter branching (Clancy and Kuipers, 1997). Chatter occurs when a variable's direction of change is constrained only by continuity within a region of the state space. Chatter results in intractable branching and a potentially infinite simulation. In combination, these two pieces of research significantly broaden the range of models that can be tractably simulated via qualitative simulation thus supporting the application of qualitative reasoning techniques to such tasks as monitoring, diagnosis and design.

References

D. J. Clancy and B. J. Kuipers, 1993. Behavior Abstraction for Tractable Simulation. In *QR-93*, pp 57-64, 1993.

D. J. Clancy and B. J. Kuipers, 1997. Static and dynamic abstraction solves the problem of chatter in qualitative simulation. In *Proceedings from the Fourteenth National Conference on Artificial Intelligence, AAAI-97*, 1997.

E. W. Coiera, 1992. Qualitative Superposition. In *Artificial Intelligence*, 56:171-196, 1992.

D. DeCoste, 1994. Goal–directed qualitative reasoning with partial states. Technical Report 57, The Institute for the Learning Sciences, Northwestern University, August 1994.

J. De Kleer and J. S. Brown, 1984. A Qualitative Physics Based on Confluences. In *Artificial Intelligence* 24:7-83, 1984.

S. Even 1979. *Graph Algorithms.* Computer Sciences Press, Rockville, MD, 1979.

K. D. Forbus 1984. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

P. J. Hayes, 1985. The Second Naive Physics Manifesto. In J. Hobbs and B. Moore (Ed), *Formal Theories of the Commonsense World.* Ablex Publishing Corp, 1985.

L. Ironi and M. Stefanelli, 1993. A Framework for Building Qualitative Models of Compartmental Systems. Technical Report N. 897, Instituto Di Analisi Numerica, Pavia, Italy, 1993.

Y. Iwasaki, 1988. Causal Ordering in a Mixed Structure. In *AAAI-88*, pp 313-318, 1988.

J. Jacquez, 1985. *Compartmental Analysis on Biology and Medicine.* Ann Arbor, MI. The University of Michigan Press, 1985.

B. J. Kuipers, 1994. *Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge.* Cambridge, MA: MIT Press, 1994.

P. P. Nayak, 1992. Automated Modeling of Physical Systems. Ph.D. Thesis, Department of Computer Science, Stanford University, Stanford, CA, 1992. Stanford Technical Report No. STAN-CS-92-1443.

J. Rickel and B. Porter, 1997. Automated Modeling of Complex Systems to Answering Prediction Questions. To appear in *Artificial Intelligence*, 1997.

H. A. Simon, 1969. *The Sciences of the Artificial.* Cambridge, MA: MIT Press, 1969.

E. Tsang, 1993. *Foundations of Constraint Satisfaction.* Academic Press Inc, San Diego, CA, 1993.

B. C. Williams, 1986. Doing time: Putting qualitative reasoning on firmer ground. In *AAAI-86*, pp 105-113, 1986.