Generating a compact representation of an exponentially large solution space *

Daniel J. Clancy
NASA Ames/Caelum Research Center, MS 269-3
Moffett Field, CA 94035 USA
clancy@ptolemy.arc.nasa.gov
http://www.cs.utexas.edu/users/clancy

Abstract

Many of the techniques developed within the field of artificial intelligence focus on methods of searching an exponentially large solution space for a solution that satisfies a given set of constraints. Certain problems, however, require a complete description of the solution thus restricting the application of these techniques. To tractably solve problems of this nature, the solution space must be transformed into a more compact representation that highlights relevant information. An example of this type of problem is encountered when using *qualitative simulation* to reason about the behavior of an imprecisely defined dynamical system.

In this paper, we present abstraction and problem decomposition techniques that have been used to solve many of the complexity problems encountered when performing a qualitative simulation. Abstraction is used to eliminate irrelevant distinctions within unconstrained regions of the solution space while problem decomposition is used to eliminate combinatoric branching due to the complete temporal ordering of unrelated distinctions that occur within the simulation. These techniques result in an order-of-magnitude improvement in the simulation time required for certain problems thus enabling the application of qualitative simulation techniques to larger, more realistic problems.

Introduction

Many of the techniques developed within the field of artificial intelligence have focused on developing algorithms to efficiently search an exponentially large solution space. Commonly, these techniques are concerned with identifying a single solution within this space. Thus, these techniques often use heuristics or efficient search strategies that allow the algorithm to focus its search on regions of the solution space most likely to result in a consistent solution. Certain problems, however, require a complete description of the solution space thus restricting the application of these techniques. To tractably solve problems of this nature, the solution space must be transformed into a more compact representation that highlights relevant information and eliminates distinctions that are not required to address the problem at hand.

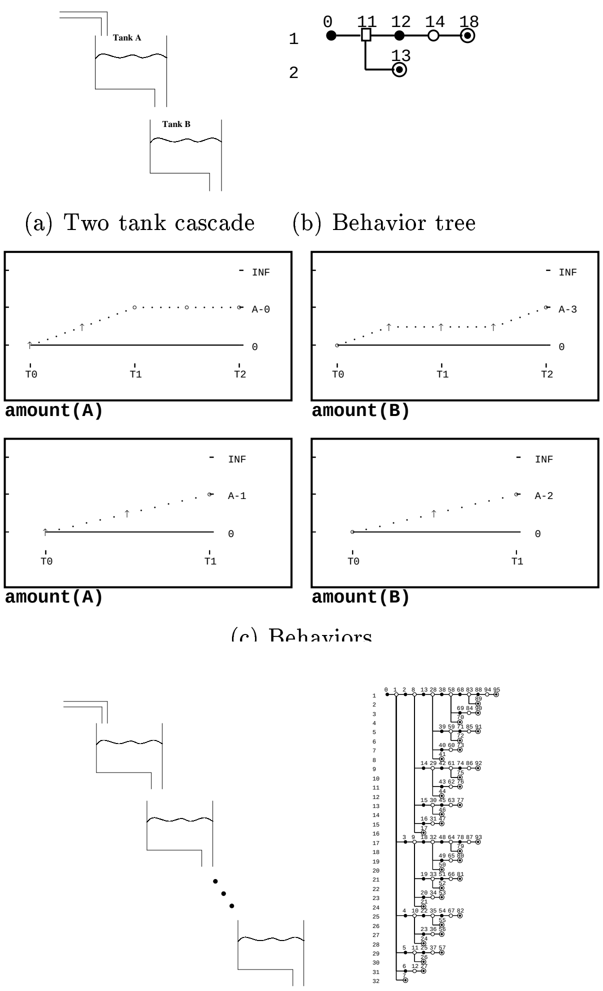
An example of this type of problem is encountered when using *qualitative simulation* to reason about the behavior of an imprecisely defined dynamical system. Qualitative simulation (Forbus 1984; Kuipers 1994) uses an abstract model describing a class of dynamical systems to generate a branching–time description of all qualitatively distinct behaviors consistent with the model. This process is equivalent to finding all solutions to a constraint satisfaction problem (CSP) and in fact can be shown to be NP-hard. As with any CSP, if two variables are completely unconstrained with respect to each other, then the set of all possible solutions will contain the cross–product of the possible values for each variable. Since qualitative models describe phenomenon within the physical world, in general the degree of interaction between any two variables tends to decrease as the size of the model grows (Simon 1969). This factor results in combinatoric branching within the behavioral description thus restricting the simulation of larger, more realistic models.

---

*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.

The problem encountered can be demonstrated using a simple model of a sequence of cascaded tanks. For a two tank cascade, the QSIM qualitative simulation algorithm (Kuipers 1994) generates a total of two behaviors providing a concise description of the system behavior (see figure 1). As we increase the number of tanks within the cascade, however, the complexity of the simulation grows exponentially since the model fails to completely constrain the order in which the tanks become quiescent (*i.e.* each tank is only related to the tanks immediately above and below it). Thus, for a nine tanks cascade, QSIM takes over 13 minutes to generate a total of 256 behaviors.[1]

The complexity problems encountered when simulating larger, more complex models results from explicitly computing all solutions to the constraint satisfaction problem defined by the model. For many of the tasks addressed by qualitative simulation, however, an explicit enumeration of all solutions is not required. Often a more compact characterization of all possible behaviors is sufficient to address the task at hand. In this paper, we present two techniques that have been used to reduce the problem of intractable branching within qualitative simulation. These techniques use abstraction and problem decomposition to modify the representation used during simulation thus providing a more compact representation of the complete solution space. The first technique, implemented in the DecSIM component–based qualitative simulation algorithm, decomposes the model into a set of loosely connected components. A complete set of solutions is computed for each component while the interactions between components are represented implicitly via links that are maintained throughout the simulation. Thus, the algorithm avoids explicitly enumerating all possible solutions to the CSP. The second technique dynamically identifies unconstrained regions of the state space during simulation and abstracts these regions into a single qualitative state in the behavioral description. Thus, it avoids explicitly enumerating solutions that differ only with respect to the behavior of a variable within this unconstrained region. Both techniques result in an exponential speedup in the time required to perform a qualitative simulation as the



(a) Two tank cascade (b) Behavior tree

(c) Behaviors

(d) Multi-tank cascade (e) Multi-tank tree

- Simulation of a qualitative model of a simple two tank cascade (a) generates a behavior tree (b) with a total of two behaviors. In the first behavior (c), tank A reaches quiescence before tank B while in the second they both reach quiescence at the same time.

- Simulation of a multi-tank cascade (d) results in a more complex behavioral description as branching occurs depending upon the order in which the tanks reach quiescence. For N tanks, a total of 2^{N-1} behaviors are generated. Thus, for a 6 tanks cascade 32 behaviors are generated (e).

Key: In a behavior tree, a filled-in circle corresponds to a time–point state, an open circle to a time–interval state and a dot inside a circle a quiescent state. In a behavior, up and down arrows correspond to increasing and decreasing respectively while a circle with a line through describes a variable that is steady.

Figure 1: Cascaded tanks Example

[1] Actually, in this example we have already applied the chatter abstraction technique described later in the paper. If chatter is not eliminated, then QSIM generates an infinite behavioral description.

size of the problem grows causing the complexity of a simulation to become a function of the problem specification as opposed to an artifact of the simulation algorithm being used. Within this paper, we primarily focus on the DecSIM algorithm; however, a brief discussion of the techniques used to identify and abstract unconstrained regions of the state space is also provided along with results demonstrating the benefits of these techniques.

While these algorithms were developed specifically for the problems encountered when performing a qualitative simulation, they will be described in more general terms highlighting their relevance to other search problems that require a representation of a large portion of the search space. To ground the discussion, we will start by providing a brief introduction to qualitative simulation along with a more detailed discussion of the problems encountered when simulating larger, more complex models. In particular, we will focus on how qualitative simulation can be viewed within a general class of constraint satisfaction problems.[2] This section is followed by a brief presentation of the specific algorithms that have been developed along with results demonstrating the benefits provided by both techniques and a discussion of how these techniques could be used within other domains.

Qualitative simulation and constraint satisfaction

In its most basic form, qualitative simulation reasons about how an imprecisely defined dynamic world evolves over time. The world is described using a finite set of discrete variables and a set of constraints defining the valid states within the world. The variable values are defined using a totally ordered sequence of landmark values that is isomorphic to the real number line. For example, the landmark values for a variable describing the temperature for a pot of water might correspond to minus infinity, the freezing point, the boiling point and positive infinity. Alternatively, the set of landmark values might also include a symbolic landmark that corresponds to some qualitatively distinct landmark that is important with respect to the other variables within the model. The magnitude of a variable can either be at a landmark or within the interval between two landmarks.

[2]Throughout our discussion, we assume that the reader is familiar with the basic concept of a CSP. Please refer to (Tsang 1993) for a more detailed discussion of constraint satisfaction.

Change occurs as the system evolves autonomously over time. To determine how the world can evolve, temporal *continuity constraints* are used to restrict the valid transitions for a variable from one state to the next. For example, if in a given state S_1 the variable X is increasing within an open interval between the landmarks 0 and $X*$ then at the ensuing time–point there are four different values for X consistent with continuity: X can continue to increase within the interval, it can reach $X*$ and continue to increase, it can become steady in the interval or it can reach $X*$ and become steady at the same time. Which of these transitions occur, depends upon the interaction between X with other variables in the model. For example, if the derivative of X is explicitly represented within the model and in S_1 it was positive and increasing, then it will not be possible for X to become steady in the ensuing state. (Since its derivative is positive and heading away from zero, continuity requires the derivative to become steady and start heading toward zero before X can become steady.) Thus, when performing simulation successor states are generated by identifying for each variable all possible successor values consistent with continuity and then computing all combinations of these values consistent with the constraints in the model

For the rest of the discussion, it may be helpful to understand the following terms and concepts:

- A variable is described using a qualitative magnitude (*qmag*) and a direction of change (*qdir*). For each variable, a *quantity space* defines the sequence of landmark values used to describe the magnitude. The direction of change can either be increasing, decreasing or steady.
- In general, the constraints correspond to abstractions of mathematical relationships. For example, the M^+ constraint relates two variables that are related by a some monotonically increasing function. As a result, the direction of change for the two variables must be equal.
- The behavior of the system is described by a tree in which each path corresponds to an alternating sequence of time-point and time-interval states. Each path within the tree corresponds to a different qualitatively distinct behavior consistent with the constraints in the model and the temporal continuity constraints. Branches occur

whenever there are multiple successors for a given state due to the imprecise description provided by the qualitative model.

- Formally, a model, called a qualitative differential equation (QDE) in the QSIM [3]framework, is defined by the tuple $<V, Q, I, C>$ where V is a set of variables, Q a discrete set of values for each variable, I is an initial state, and C a set of state–based constraints on the variables in V

The general task addressed by qualitative simulation is best understood when it is viewed as a constraint satisfaction problem. A qualitative model maps directly to a traditional, *state-based* CSP in which the set of constraints identifies the set of valid world states. During simulation, however, this CSP is composed with a temporal CSP defined by the continuity constraints.[4] Thus, qualitative simulation can be viewed as a *temporally-extended* CSP defined by the composition of these two sets of constraints (see figure 2). Each qualitative behavior corresponds to a distinct solution to the CSP. Qualitative simulation attempts to find all consistent solutions to the temporally-extended CSP defined by the model coupled with the initial state. Please refer to (Clancy & Kuipers 1998) for a formal definition of a temporally-extended CSP.

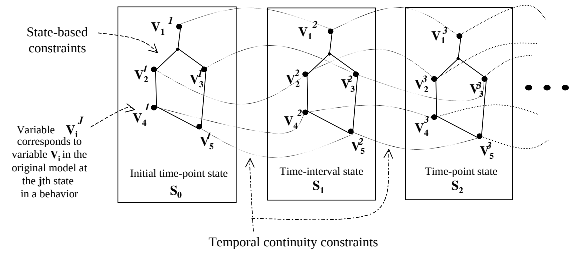


Figure 2: Qualitative simulation as a temporally-extended CSP

Problem description

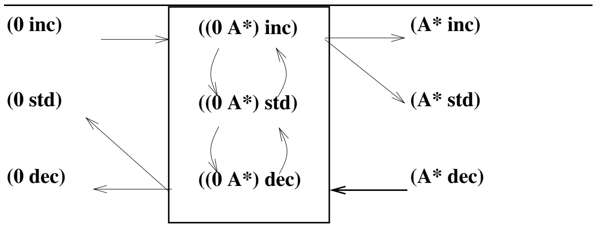
Since qualitative simulation computes all solutions to a CSP, the complexity of the simulation is determined by the size of the solution space. Large problems are tractable if the number of consistent solutions remains small. Within qualitative simulation, however, larger models tend to be more loosely constrained thus leading to a large number of solutions. To reduce the complexity of the simulation for larger models, we must either reduce the size of the solution space by further constraining the model or use abstraction, aggregation and problem reformulation techniques to provide an alternative representation.

Complexity problems encountered when performing a qualitative simulation can be classified into two categories: *combinatoric event branching* and *chatter*. An *event* occurs when a variable reaches a landmark or becomes steady. If the model fails to constrain the temporal ordering of a set of upcoming events, a branch results within the behavioral description. If a set of events are unrelated, then the simulation computes all possible temporal orderings of the events thus resulting in combinatoric event branching. For example, a simulation of two independent balls being thrown into the air may result in a total of 16 different behaviors. Each behavior specifies a different ordering between which ball reaches its apex first and when each ball returns to the ground.

While combinatoric event branching maps conceptually into other domains, the problem of chatter is a little more specific to the representation used when performing a qualitative simulation. Chatter occurs whenever the derivative of a variable is unconstrained within a region of the state space. Since the derivative is unconstrained, the variable is free to alternate between increasing, steady and decreasing. When the variable becomes steady, a branch occurs depending upon whether it starts to increase or decrease in the next state (see figure 3). Since qualitative simulation, however, is computing all consistent temporal evolutions, it is possible for the variable to remain within this unconstrained region of the state space for an arbitrary amount of time. As a result, the behavioral description is completely dominated by branches within this re-

[3]In this presentation, we focus on the representation used by the QSIM qualitative simulation algorithm. These concepts can also be applied to alternative representations (Forbus 1984; de Kleer & Brown 1985) that have been proposed within the literature.

[4]Note that the concept of temporal constraints within a simulation can be generalized to allow for a declarative and more expressive representation of temporal constraints. The TeQSIM (Brajnik & Clancy 1996) qualitative simulation algorithm does this my integrating temporal logic model checking into the simulation process allowing modelers to specify arbitrary temporal constraints using a propositional, linear-time temporal logic.



- Each "node" within the figure corresponds to a different qualitative value for a variable while the arrows represent transitions that are consistent with continuity.
- The boxed region identifies the region of the state space in which chatter occurs if the direction of change of the variable is unconstrained. Once the variable enters this region, it can continue sequencing between the different values for an arbitrary number of states.

Figure 3: Intractable chatter branching

gion of the state space. For the purposes of this discussion, the details of this problem are not that important. To understand the ideas presented, it is sufficient to view chatter as an unconstrained region of the solution space in which the distinctions being made are irrelevant within the broader problem solving context.

Proposed Solutions

Two different solutions have been developed to address these problems. Combinatoric event branching is addressed using the DecSIM component–based qualitative simulation algorithm. DecSIM decomposes the model into components and then simulates each component separately. Chatter is branching is eliminated by abstracting unconstrained regions of the state space into a single qualitative state within the behavioral description. This abstraction is performed on-the-fly as the behavioral description is being incrementally generated.

DecSIM simulation algorithm

DecSIM uses a divide and conquer approach to eliminate distinctions due to the complete temporal ordering of unrelated events. Given a model M with a set of variables V and an initial state I, DecSIM uses a partitioning $\{p_1, p_2, \ldots p_n\}$ of the variables in the model to generate a set of *components.*[5]

A separate behavioral description (*i.e.* set of solutions) is explicitly generated for each component. For each partition p_i, DecSIM generates a component C_i containing two types of variables:

within-partition variables are the variables specified in p_i, and

boundary variables are variables contained in other partitions that are related to variables in p_i via a constraint within the model.[6]

Each component C_i defines an independent qualitative model in which the variables correspond to the union of the within–partition and boundary variables for the component and the constraints correspond to the set of constraints within the original model relating the variables contained within the component. The interaction between components is represented by the shared boundary variables. Please refer to figure 4 for an example demonstrating the various concepts presented in this section.

Local and global consistency

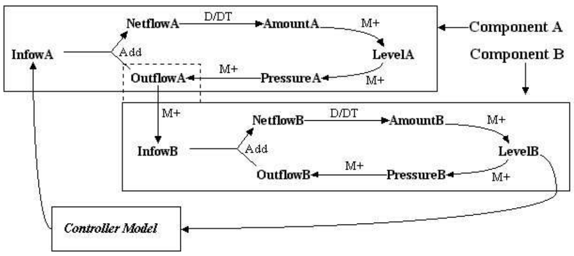
The core QSIM algorithm is used to derive a separate behavioral description, called a *component tree*, for each component. The terms *component behavior* and *component state* are used to refer to a behavior and a state within a component tree respectively.

Definition 1 (Local consistency) *A component behavior for component C_i is* locally consistent *if and only if it is consistent with the qualitative model defined by the component.*

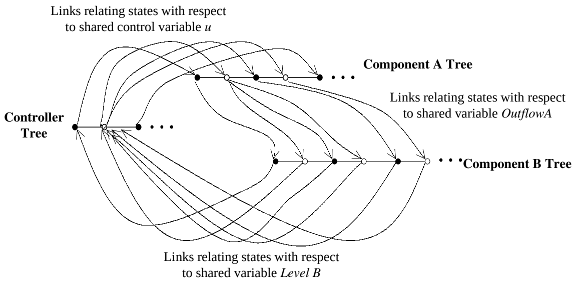
QSIM ensures that each component behavior is locally consistent. In addition, however, each component behavior must be consistent with respect to the rest of the model (*i.e. globally consistent*). The constraints between components are represented by shared boundary variables. These constraints define a separate, global CSP in which the "variables"

[5]Currently, the partitioning of the variables is provided as an input to the DecSIM algorithm. Various techniques for automating this process have been considered; however, up to this point, we have focused on the simulation algorithm since partitioning the variables is a fairly straight–forward extension of the model–building process.

[6]Causal information is used when available to restrict the set of boundary variables within a component to those that have a *direct causal influence* on the within–partition variables (see (Clancy & Kuipers 1998)).



(a) Controlled two tank cascade



(b) Links relating component states

- DecSIM translates a model of a controlled two tank cascade (a) into a causal hypergraph. Variables correspond to nodes and constraints to hyper-edges. The partitioning provided separates the variables into three partitions: tanks A, tank B and the controller.
- DecSIM generates 3 components. `OutflowA` is included as a boundary variable in component B, `LevelB` in the controller component and `u` in component A. (Causality is used to infer directionality in these relations.)
- During simulation, a many–to–many mapping (b) is maintained between the states in related components. This mapping is used to define constraints within a global CSP. In this example, a state is globally consistent if a cycle exists within the mapping. Note that the last state in the component A tree is not globally consistent at this point in the simulation.

Figure 4: DecSIM simulation of a controlled two tank cascade

correspond to components and the "variable values" to component behaviors. A component behavior is *globally consistent* if and only if there exists a solution to this global CSP. A solution to the global CSP corresponds to a set of component behaviors, one from each component, that can be combined to form at least one consistent behavior describing all of the variables in the original model. Behaviors must be combined in a manner that satisfies both the state–based and the temporal continuity constraints.

To determine if a pair of component behaviors are *compatible* (*i.e.* can be combined to form a single behavior describing all of the variables in the two components), a many–to–many mapping is maintained between states in components that share a variable. This mapping is incrementally generated as each component behavior is extended. If components A and B are related via shared variables, then state S_A maps to component state S_B if and only if S_A and S_B are equivalent with respect to shared variables, and either

- S_A and S_B are initial states,
- the predecessor of S_A maps to the predecessor of S_B,
- the predecessor of S_A maps to S_B, or
- S_A maps to the predecessor of S_B.

The mapping that is maintained between states allows the global CSP in which variable values correspond to component *behaviors* to be translated into a CSP, called the *component graph CSP*, in which the variable values correspond to component *states*. The constraints within the component graph CSP are defined by the mapping that is maintained between states in related components. A constraint is satisfied if and only if the two states are mapped together. This translation allows global consistency to be computed via a local computation with respect to the temporal progression of a behavior and is essential if DecSIM is to efficiently reason about the interaction between components. Thus, global consistency becomes a property of a qualitative state as opposed to a qualitative behavior.

Definition 2 (Globally consistent) *A component state is* globally consistent *if and only if the state is both locally consistent and participates in a solution to the component graph CSP.*

A partially computed component behavior is globally consistent if and only if the final state in the behavior is globally consistent.

Performing the simulation

When performing a simulation, DecSIM iterates through the components incrementally simulating the leaf states in each component tree. DecSIM uses QSIM to generate the successors of each component state, thus ensuring that each state is locally consistent. In addition, however, DecSIM must ensure that each state is also globally consistent.

Determining whether a component state is globally consistent for a fully simulated set of component behaviors is a straight–forward constraint satisfaction problem given the characterization that has been provided. DecSIM, however, incrementally generates each component behavior. Thus, it is possible that as component behaviors are extended, a solution to the component graph is generated containing a state already within a component tree that had previously not been globally consistent. In other words, a state S not being globally consistent at a given point during the simulation does not imply that S is globally *inconsistent*. DecSIM addresses this issue using two techniques:

1. successors of a component state S are only computed if S is determined to be globally consistent, and
2. the identification of a state as globally consistent must be propagated through the component graph CSP by identifying all new solutions to the component graph that contain at least one state whose status with respect to global consistency had previously been undetermined.

The first condition ensures that globally inconsistent states are not simulated while the second condition ensures that all globally consistent solutions are still computed given the first restriction.

The algorithm used to test a state for global consistency exploits causality and structure within the component graph CSP to reduce the complexity of finding a solution. First, a tree-clustering algorithm (Dechter & Pearl 1988; 1989) is used to transform the component graph CSP into an acyclic *cluster graph*. Since the structure of the component graph CSP is constant throughout a simulation, this transformation significantly reduces the time required to find a solution to the component graph by allowing DecSIM to use constraint propagation between clusters as opposed to computing a complete solution to the CSP. Second, causality is used to further reduce the complexity of this process by asserting the independence of causally upstream components from those components that are strictly downstream with respect to the causal ordering.

Two primary benefits are provided by DecSIM with respect to a standard QSIM simulation. First, by transforming the component graph into an acyclic cluster graph, the worst case complexity required to find a solution to the component graph CSP is exponential in the size of the largest cluster as opposed to overall size of the component graph. Second, DecSIM is only required to find a single solution to the component graph for each component behavior as opposed to computing all possible solutions as is required by a standard QSIM simulation. This significantly reduces the complexity of a simulation for models that lend themselves to decomposition.

The increased efficiency of the DecSIM algorithm, however, comes at a cost. While DecSIM is sound with respect to the CSP defined by the original model, it is potentially incomplete (*i.e.* DecSIM may generate a component behavior that does not correspond to a behavior for the non–decomposed model). The source of the incompleteness is the use of component *states* as opposed to complete *behaviors* within the global CSP. The problem encountered is analogous to the distinction between constraint propagation and constraint satisfaction with respect to the temporal continuity constraints. Before allowing a state S_A to participate in a solution to the component graph CSP, DecSIM requires that its predecessor participate in a solution. However, it does not check to ensure that the solution containing S_A satisfies the continuity constraints with respect to a solution containing the predecessor of S_A. To do this, DecSIM would be required to maintain a record of solutions to the component graph CSP to ensure that a proposed solution is continuous with a solution identified for the preceding time-step.

In practice, the incompleteness of the algorithm has not been a problem for a number of reasons. First, the conditions under which the incompleteness of the algorithm is encountered is quite restricted and only occurs when three or more components are closely related. In fact, we have yet to encounter this problem in any of the models that have been tested. In addition, we have developed an algorithm that can be used to identify when this problem occurs which can be run following completion of a simulation. Finally, qualitative simulation already encounters a problem with behaviors being generated that do not correspond to a real-valued trajectory of a dynamical system described by the model. Thus, techniques using qualitative simulation already must account for possible spurious behaviors.

General relevance

The preceding discussion focuses on how the DecSIM algorithm with respect to qualitative simulation. In general, however, a similar approach can be used whenever it is necessary to compute all possible evolutions of the world over time. This problem occurs whenever the state of the world is incompletely known. For example, within the planning domain, uncertainty within the state of the world and/or non-deterministic actions requires the inference procedure to reason about all possible evolutions that are consistent with a sequence of proposed actions. This is distinct from the traditional problem addressed in planning in which only a single solution is considered. In this situation, it is necessary to provide a more compact representation of the solution space. More generally, the ideas presented here are applicable when reasoning about all possible temporal evolutions of a world that can change non-deterministically.

Eliminating chatter

For tasks that require the identification of a single solution to a CSP, unconstrained regions of the state space simplify the process of identifying a solution. When all possible solutions are required, however, an unconstrained region results in an exponential number of solutions that are distinguished only with respect to the values for the variables within this region. Chatter occurs when the derivative of a variable is constrained only by continuity within an open interval. Since the behavior is unconstrained, the variable is free to change between increasing, decreasing and steady an arbitrary number of times within the interval. This results in a potentially infinite behavioral description in which the chattering behaviors completely obscure other behaviors within the description without providing additional relevant information. The problem of chatter is quite distinct from event branching due to the nature of the distinctions that are being made. When an event branch occurs, the behavior of a variable is only unconstrained with respect to the temporal occurrence of a set of events. Thus, the variable is not able to move freely as it does when chatter occurs.

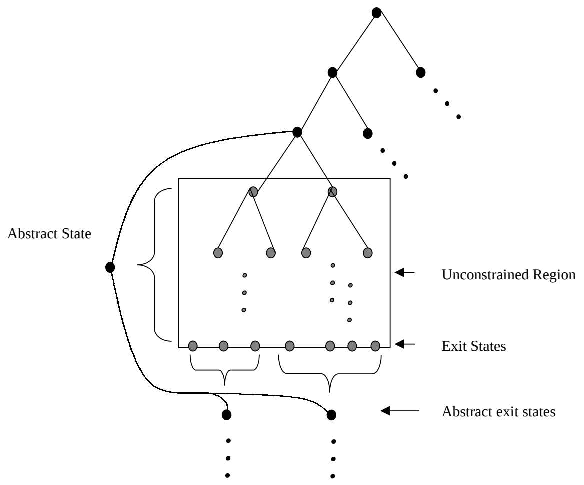
Chatter occurs due to the abstract representation used when performing a qualitative simulation and can be eliminated in many cases by ignoring the direction of change for the chattering variables throughout the simulation (Fouché & Kuipers 1990). At times, however, information about the derivative of a variable is required within certain regions of the state space and unconstrained within others. Furthermore, simply ignoring the direction of change does not eliminate chatter that occurs in a variable whose derivative is explicitly represented within the model. To provide a general purpose solution to the problem of chatter, we must selectively ignore distinctions in a variables direction of change only when it is unconstrained.

Two different abstraction techniques have been developed to eliminate this problem when performing a qualitative simulation: *chatter box abstraction* and *dynamic chatter abstraction*. Both techniques perform the following steps (see figure 5):

1. Detect when an unconstrained region is entered.
2. Identify the boundaries of the unconstrained region.
3. Generate a single abstract state providing an abstract qualitative value for the unconstrained variables within this region.
4. For the abstract state generated in step 3, generate successor states corresponding to the points at which the system can exit the unconstrained region.

The techniques differ in the manner in which the boundaries of the unconstrained region are identified and the techniques used to generate the abstract state and its successors.

Chatter box abstraction identifies the boundaries of the unconstrained region by identifying an upper bound on this region and then using a recursive call to the simulation algorithm to explore the region via simulation. The results of the recursive call are used to generate the abstract state and generate its successors (*i.e.* steps 3



Both techniques eliminate distinctions within unconstrained regions of the solution space by abstracting these regions into a single state within the search tree. (The figure above is a generalized representation of the search space explored during simulation.)

- By identifying the boundaries of the unconstrained region, both algorithms are able to generate an abstract state describing the region (a) in addition to identifying the states that exit the region.
- The states that exit the region must be combined to eliminate the distinctions that are made within the abstracted region.

Figure 5: Abstracting unconstrained regions of the state space

and 4). Thus, it leverages the inference capabilities already provided by the simulation algorithm and can be shown to be both sound and complete with respect to the inferences made by the unmodified simulation algorithm. The problem with this technique is that the complexity of the recursive call to the simulation algorithm is exponential in the number of unconstrained variables within the region. Thus, as the size of a model grows, the technique becomes prohibitively expensive.

Dynamic chatter abstraction identifies the unconstrained region by a dynamic analysis of the constraints within the model and the current state. Thus, as opposed to using brute force search (as chatter box abstraction does), dynamic chatter abstraction reformulates the problem into a CSP that simply attempts to answer the question: "Is variable x unconstrained within the current region of the state space." Thus, dynamic chatter abstraction provides a more focused search of the state space as opposed to enumerating all possible paths through the unconstrained region as chatter box abstraction does.

The chatter box abstraction algorithm is fairly straight–forward. Dynamic chatter abstraction algorithm, on the other hand, must address a number of detailed issues that makes it much more complicated. In particular, it must reason not only about the values within the current state, but it also must reason about how these values might change over time as variables begin to chatter. For example, suppose that the derivative of a variable x is unconstrained within the current qualitative state. Thus, its direction of change is free to chatter. Once the variable begins to change in a new direction, however, a variable y that was previously constrained may now become unconstrained and thus begin to chatter. Both variables must be identified as chattering.

Understanding the nuances of the dynamic chatter algorithm requires a more detailed understanding of qualitative simulation and the manner in which chatter is manifested within a simulation. In general, the dynamic chatter abstraction algorithm can be viewed as a special purpose inference engine that has been designed to answer the focused question: "Does a given variable chatter following the current qualitative state before a non–chattering variable changes?" The algorithm encodes knowledge not only about how the constraints that restrict a given variable at a single point in time, but it also encodes information about how these values can change across time without explicitly computing all of the behaviors.

Comparison of these techniques

Both chatter box abstraction and dynamic chatter abstraction provide unique benefits. Chatter box abstraction provides a simple algorithm that leverages the inference capabilities of the simulation algorithm thus enabling the seamless integration of extensions to the simulation algorithm with the abstraction technique. Dynamic chatter abstraction, on the other hand, provides a much more efficient algorithm that scales to larger problems. As the simulation algorithm is extended, however, addi-

# of Tanks	Cascade		Chained		Loop	
	Qsim	DecS	Qsim	DecS	Qsim	DecS
2	0.20	0.815	3.07	6.79	0.757	5.58
3	0.62	1.6	10.9	19.90	16.14	8.14
4	2.2	3.12	37.5	25.98	89.41	12.6
5	7.09	5.49	139	36.71	493.8	23.2
6	21.9	6.32	676	62.40	2758	48.7
7	71.5	8.39	1633	70	14474	116
8	236	11.6	8101	77	nc	442
9	806	11.7	nt	nt	nt	nt
10	nc	14.0	nt	nt	nt	nt

nt = Not Tested, nc = Resource limitation prevented completion

Table 1: DecSIM Simulation Time Results

tional extensions are required to the dynamic chatter abstraction algorithm.

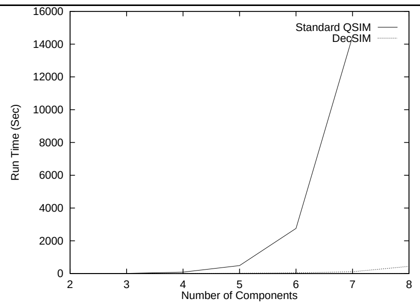
Empirical Evaluation

Empirical evaluation has been used measure the benefits provided by both DecSIM and chatter abstraction using a set of "extendible" models. An extendible model is a model composed of a sequence of identical components thus enabling the incremental extension of the model to facilitate an evaluation of the asymptotic behavior of the algorithm. All of the results reported here are for variations on the cascaded tank example described in figure 1.

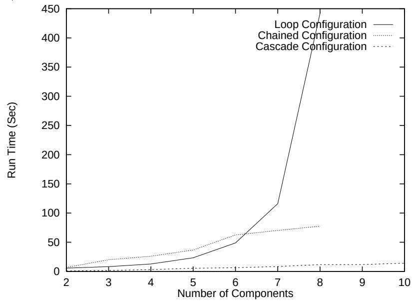
DecSIM

The DecSIM algorithm was evaluated by comparing a DecSIM simulation against a standard QSIM simulation. Three different variations on the cascaded tanks example have been used: a simple *cascade* topology, a *loop* topology in which the top tank is controlled by the bottom tank, and a *chain* topology in which the outflow for tank n is controlled by the level for tank $n + 1$. Each version provides a different topology with respect to the relationship between the components.

For all three models, DecSIM performed exponentially better than QSIM. Table 1 shows the simulation time results as the number of tanks are varied while figure 6(a) plots the results for the loop configuration comparing QSIM to DecSIM. The benefits provided by DecSIM are even more pronounced for the other two topologies. Figure 6(b) provides a comparison of the results from the DecSIM simulation for all three topologies. Note the dependence of the computational complexity on the topology of the model. In the loop configuration,



(a) DecSIM vs QSIM: Loop configuration



(b) DecSIM on different versions of the N-tank cascade

Figure 6: DecSIM results

the component graph CSP is composed of a single, large cycle. Thus, it is more likely to encounter backtracking when determining if a component state is globally consistent. Thus, the complexity of the simulation becomes exponential in the number of tanks. DecSIM, however, still performs significantly better than QSIM. Conversely, for the simple N-tank cascade the complexity is linear in the number of tanks.

Chatter abstraction

When chatter occurs within a qualitative simulation, it totally obscures other relevant distinctions within the behavioral description and can result in an infinite behavioral description. Thus, chatter must be eliminated if the results of the simulation are to be used in an effective manner. While various techniques have been developed to eliminate chatter in the past (Fouché & Kuipers 1990; Kuipers *et al.* 1991), none of them provide a general purpose solution that eliminates all instances of chatter without over–abstracting. Both of the techniques presented here provide such solutions. Please refer to (Clancy 1997) or (Clancy & Kuipers 1997) for a more detailed discussion and comparison of existing techniques with respect to the algorithms presented here along with a presentation of theoretical results demonstrating that both chatter

# of Tanks	Execution Time (sec)		
	Chatter Box	**Ignore Qdirs**	**Dynamic Chatter**
2	0.369	0.107	0.067
3	1.232	0.252	0.543
4	6.586	0.805	0.523
5	39.67	2.609	0.98
6	287.6	8.768	1.552
7		29.52	2.377
8		102.9	3.443
9		363.5	4.966
10		1284	6.905

- Tests were run on a Sparc 10 using Lucid Common Lisp. Execution time measures the processing time for a single state. Blank entries correspond to simulations that could not be completed due to resource limitations.

Table 3: Comparison of chatter abstraction techniques for the N–tank cascade

abstraction algorithms provide general solutions to the problem of chatter.

To demonstrate the benefits of dynamic chatter abstraction versus chatter box abstraction, both techniques were tested on a variety of models that exhibit chatter. In all cases, dynamic chatter abstraction performed significantly better. Table 2 reports the results for a number of these models. To evaluate the asymptotic behavior, both algorithms along with the ignore-qdirs chatter elimination technique (Fouché & Kuipers 1990) were compared on the simple multi-tank cascade example used to evaluate DecSIM.

In this model, the $netflow$ variables for each tank exhibit chatter. The results from these tests are listed in table 3. A plot of these results, demonstrates that the that execution time increases exponentially for both chatter–box abstraction and ignore qdirs. For chatter–box abstraction, we were unable to complete the simulation for models with six or more tanks because of resource limitations.

These results demonstrate the need for the automatic elimination of chatter when performing a qualitative simulation. In addition, they demonstrate the benefits provided by a special purpose inference engine designed to identify and abstract the unconstrained regions of the state space.

Conclusions

In this paper we have demonstrated how abstraction and problem reformulation techniques can be used to significantly reduce the complexity encountered when reasoning about the behavior of an imprecisely defined dynamical system using qualitative simulation. The primary contribution of this work is the application of these techniques within qualitative simulation thus facilitating the simulation of larger, more realistic problems. In addition, however, the basic concepts developed here can potentially be applied within other areas.

The DecSIM algorithm reasons about the detailed interaction between closely related variables while eliminating irrelevant distinctions resulting from the interaction of loosely related variables. This approach is similar to the manner in which people commonly reason about complex systems. Similar problems are potentially encountered in fields such as planning or non-monotonic reasoning when reasoning about discrete change within the world. In (Clancy & Kuipers 1998), we provide a more detailed description of the concept of a temporally-extended constraint satisfaction problem thus highlighting many of the relationships between planning and qualitative simulation and demonstrating how the ideas contained within the DecSIM algorithm may be applicable when generating robust plans within an autonomously changing world.

While the specific algorithms used to eliminate chatter are clearly specialized to a problem that is unique to qualitative simulation, the basic approach highlights a general strategy that can be used within other domains. In addition, the results demonstrate the benefits provided by special purpose inference algorithms when attempting to identify and abstract unconstrained regions of the solution space.

References

Brajnik, G., and Clancy, D. J. 1996. Temporal constraints on trajectories in qualitative simulation. In Clancey, B., and Weld, D., eds., *Proc. of the Thirteenth National Conference on Artificial Intelligence*. AAAI Press. To appear.

Clancy, D. J., and Kuipers, B. J. 1997. Static and dynamic abstraction solves the problem of chatter in qualitative simulation. In Kuipers, B. J., and Webber, B., eds., *Proc. of the Fourteenth National Conference on Artificial Intelligence*. AAAI Press.

Clancy, D. J., and Kuipers, B. J. 1998. Qualitative simulation as a composition of state-based and

Model	Numb of Vars	Numb of Chattering Vars	Number of Behaviors				Simulation time (sec)	
			No chatter abstraction		Abstract chatter			
			Env	**Beh tree** (no lms)	**Beh tree** (w/ lms)		**Dyn Chat**	**Chat Box**
W–Tube	16	1	3	> 1845	1		0.9	3.4
Glucose-insulin Interaction	11	2	155	> 2807	41		14	52
Van der Pol Equation	10	4	43	> 1788	12[a]		2.3	15
Controlled Hot/Cold tank[a]	14	5	24	> 601[b]	14		5.0	48.2
Turgor Stomates	19	7	509	> 2598	1		2.4	20.5
Cooling Plant[a]	15	10	659	> 4095	1		7.7	418
Heart Model	42	28	689	> 2784	200		100	[c]

- Each model was tested both with and without chatter abstraction. Both an envisionment (a simulation in which a graph is generated as opposed to a tree) and a behavior tree simulation (without landmark introduction) were performed without chatter abstraction using a state–limit of 5000. The number of behaviors are listed above. Note that while an envisionment often yields fewer behaviors, the total number of qualitatively distinct behaviors is the same, as there are an infinite number of paths within the envisionment graph.
- Both types of chatter abstraction generated the same number of behaviors; dynamic chatter abstraction, however, was significantly faster than chatter–box abstraction. In general, the benefits provided by dynamic chatter abstraction become more pronounced as the number of chattering variables increases.

[a] Oscillatory behavior results in infinite behavior tree. Simulation terminated once the structure within the tree could be determined.

[b] Could not be simulated to completion due to resource limitations.

Table 2: Evaluation of Dynamic Chatter and Chatter–Box Abstraction

temporal-based csps. In *Proc. of the Fifteenth National Conference on Artificial Intelligence.* AAAI Press.

Clancy, D. J. 1997. Solving complexity and ambiguity problems in qualitative simulation. Technical Report AI-TR97-264, Artificial Intelligence Laboratory, The University of Texas at Austin.

de Kleer, J., and Brown, J. S. 1985. A qualitative physics based on confluences. In Hobbs, J. R., and Moore, R. C., eds., *Formal Theories of the Commonsense World.* Norwood, New Jersey: Ablex. chapter 4, 109–183.

Dechter, R., and Pearl, J. 1988. Tree-clustering schemes for constraint processing. In *Proceedings of the Seventh National Conference on Artificial Intelligence.* Los Altos, CA.: Morgan Kaufman.

Dechter, R., and Pearl, J. 1989. Tree–clustering for constraint networks. *Artificial Intelligence* 38:353–366.

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

Fouché, P., and Kuipers, B. 1990. An assessment of currrent qualitative simulation techniques. In *Proc. of Fourth International Worskhop on Qualitative Reasoning about Physical Systems*, 195–205.

Kuipers, B.; Chiu, C.; Molle, D. D.; and Throop, D. 1991. Higher–order derivative constraint in qualitative simulation. *Artificial Intelligence* 51:343–379.

Kuipers, B. 1994. *Qualitative Reasoning: modeling and simulation with incomplete knowledge.* Cambridge, Massachusetts: MIT Press.

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

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