SANDIA REPORT

SAND2008-7327
Unlimited Release
Printed October 2008

Analysis of Complex Networks Using Aggressive Abstraction

Richard Colbaugh, Kristin Glass, and Gerald Willard

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94AL85000.

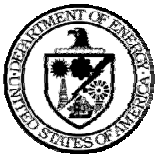
Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd.
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online



SAND2008-7327
Unlimited Release
Printed October 2008

Analysis of Complex Networks Using Aggressive Abstraction

Richard Colbaugh
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185

Kristin Glass
New Mexico Institute of Mining Technology
801 Leroy Place
Socorro, NM 87801

Gerald Willard
Department of Defense
9800 Savage Road
Ft. Meade, MD 20755-6000

Abstract

This paper presents a new methodology for analyzing complex networks in which the network of interest is first abstracted to a much simpler (but equivalent) representation, the required analysis is performed using the abstraction, and analytic conclusions are then mapped back to the original network and interpreted there. We begin by identifying a broad and important class of complex networks which admit abstractions that are simultaneously dramatically simplifying and property preserving – we call these *aggressive abstractions* -- and which can therefore be analyzed using the proposed approach. We then introduce and develop two forms of aggressive abstraction: 1.) *finite state abstraction*, in which dynamical networks with uncountable state spaces are modeled using finite state systems, and 2.) *one-dimensional abstraction*, whereby high dimensional network dynamics are captured in a meaningful way using a single scalar variable. In each case, the property preserving nature of the abstraction process is rigorously established and efficient algorithms are presented for computing the abstraction. The considerable potential of the proposed approach to complex networks analysis is illustrated through case studies involving vulnerability analysis of technological networks and predictive analysis for social processes.

Contents

Figures

Acronyms

BMC	bounded model checking
BNF	Brunovsky normal form
CSM	complex system matrix
DILO	deep information from limited observation
HDS	hybrid dynamical systems
ISS	indistinguishable starting sets
LTL	linear temporal logic
PCE	post/context entropy
PEP	positive externality processes
RYF	robust yet fragile
SDP	social democratic party
SI	social influence
SMT	social movement theory
SSI	state space subsets of interest
SDE	stochastic differential euqation
S-HDS	stochastic hybrid dynamical system
SOS	sum of squares
SCFS	supervisory control for finite state systems

1. Introduction

An enormous range of systems and phenomena of importance in nature and society can be profitably represented and analyzed as networks (or graphs), with system components being modeled as vertices in the network and relationships or dependencies among these elements being encoded as network edges. For example, many advanced technologies (e.g., electric power grids, computer networks), biological processes (e.g., metabolism, gene regulation), and social phenomena (e.g., organizations, diffusion of innovations) can be naturally modeled in this way [see, for instance, Carlson and Doyle 2002, Newman 2003, Barabasi and Oltvai 2004, and the references therein]. An interesting aspect of most real world networks is the fact that their topologies, while not perfectly uniform (like a lattice), possess considerable structural regularity; the topologies of these networks are related to, and indeed largely enable, their myriad functionalities. This observation has motivated much recent work in which researchers model a system of interest as a network and then attempt to deduce from vertex connectivity patterns something about the properties and behaviors of the underlying system. This *complex networks* perspective has already produced significant advances and appears to hold vast potential. However, in order to realize this promise we must overcome daunting challenges. For example, real world networks are typically very large, nonlinear dynamical systems, their topological structures are usually heterogeneous, subtle, and intricate, and the data describing them are often noisy and incomplete.

This paper introduces a new approach to analyzing complex networks which addresses these challenges. A key step in the proposed methodology is to abstract the network of interest in a manner which is simultaneously dramatically simplifying and property preserving; we call this process *aggressive abstraction*. The network abstraction then becomes the focus of analysis, leading to significantly enhanced tractability while ensuring application relevance. In particular, any analytic conclusion obtained for the abstraction can be related back to the original network because the network and its abstraction are equivalent by construction.

One should not expect aggressive abstractions to exist for arbitrary networks, of course, and in fact one contribution of this paper is to identify classes of networks which admit such abstraction. Here we introduce in an informal way perhaps the most important class of such networks and defer to subsequent sections a careful, mathematically rigorous discussion of these networks and their abstraction properties. Consider, then, those networks that have both a set of functions to perform and the opportunity to evolve in order to improve their functionality.

This evolutionary process, which is ubiquitous in both natural and manmade networks, frequently yields abstractable networks. To see why this is so, observe first that such evolution leads to "robust, yet fragile" (RYF) networks which perform reliably in the presence of familiar, expected disturbances but can fail catastrophically in response to a small, new perturbation [Carlson and Doyle 2002]. Examples of RYF networks include electric power grids which deliver reliable, inexpensive power but are susceptible to continent-spanning cascading outages, financial markets which enable flexible, efficient transactions but also experience billion dollar crashes, and immune systems that provide responsive, adaptive protection against infection but also introduce the possibility of lethal autoimmune disease.

The evolution underlying RYF increases the robustness of those network components and structures which are most susceptible to failure but, simultaneously and inevitably, generates fragilities associated with other structures [Carlson and Doyle 2002, Laviolette et all 2008]. A canonical example is provided by the feedback control systems which are pervasive in evolved networks: feedback regulation greatly improves robustness to variations in network elements but also introduces new fragilities (e.g., to a sign change in a feedback loop). The networks that result from RYF evolution have "designs" composed of robust and fragile features, and these configurations tend to admit aggressive abstraction. The explanation is straightforward. Such networks can be accurately modeled using only simple representations for the robust features – because network behavior is only weakly dependent on the details of these features – provided the fragile features are captured with fidelity. Because so many of the features in evolved networks are robust, the overall models can be very simple and still provide a faithful representation of the original network. Of course, actually constructing aggressive abstractions for complex, real world networks can be very challenging, and much of this paper is devoted to providing provably-correct, computationally tractable methods for performing the abstraction.

While the past decade has seen great advances in our understanding of complex networks [e.g., Newman 2003], much remains to be done. In particular, analysis methods are needed which can address the scale, complexity, and dynamics of real world systems. A natural approach to understanding large-scale systems is to employ *model reduction* techniques to obtain a more tractable system representation, and of course model reduction is a well established methodology being employed in a variety of fields [e.g., Gorban et al. 2006]. However, very little work has been done to apply model reduction ideas to complex networks, particularly in ways that exploit the considerable structure of these networks. Moreover, in many complex networks applications it would be of considerable value to derive simplifying system representations which are property-preserving, so that conclusions obtained through analysis of

the simplified model also pertain to the original network; classical model reduction methods typically cannot provide both dramatic simplification and property equivalence.

Researchers in theoretical computer science have proposed useful definitions of system equivalence and developed algorithms which generate equivalent abstractions for large-scale systems [Milner 1989]. However, this work has focused on abstracting finite state systems, while the dynamics of most complex networks require infinite (typically uncountable) state representations. Recent work in the systems and controls literature has begun to address this latter challenge, proposing methods for obtaining equivalent abstractions for infinite state systems [e.g., Alur et al. 2000, Tabuada and Pappas 2006]; this work is closer in spirit to our approach and, indeed, provides inspiration for some of our development.

Finally, we mention that using scalar variables to analyze and characterize the dynamics of complex systems can also be viewed as a form of model abstraction, and this approach to complexity management has a long and rich history. For example, energy-based analysis has been used for centuries in mechanics, and methods for extending this basic idea to a broad range of systems and analysis/synthesis objectives (e.g., using Lyapunov functions) have become a mainstay of systems theory and practice [Sontag 1998]. A major obstacle to the adoption of such methods for complex networks analysis is the lack of systematic, computationally tractable methods for obtaining the appropriate scalar functions with which to conduct the analysis.

This paper presents a new framework for analyzing complex networks based on aggressive abstraction, that is, dramatically simplifying and property preserving abstraction of the network of interest. Once an aggressive abstraction is derived, all required analysis is performed using the abstraction. The analytic conclusions are then mapped directly to the original network and interpreted there; this is possible because of the property preserving nature of the abstraction process. Our first contribution is to identify broad and important classes of complex networks which are abstractable in this way and which can therefore be analyzed using the proposed approach. As indicated above, these networks are typically the result of an evolving process which favors systems that are robust to (familiar) environmental perturbations and internal flaws. While this characterization of abstractable networks explains why they are common, it does not lend itself to computational tests which can be used to determine whether a given network is abstractable. Thus we give precise, tractable algorithms for testing whether a complex, evolving network of interest can be abstracted using any of our methods.

The second main contribution of the paper is to introduce and develop two forms of aggressive abstraction: 1.) *finite state abstraction*, in which dynamical networks with

uncountable state spaces are modeled using finite state dynamical systems, and 2.) *one-dimensional abstraction*, whereby high dimensional network dynamics are meaningfully captured using scalar functions. In each case, the property preserving nature of the abstraction process is rigorously established, efficient algorithms for computing the abstraction are presented, and the main concepts are illustrated through simple examples.

Finally, the considerable potential of the proposed approach to complex networks analysis is demonstrated through real world case studies. In our first case study, we develop a powerful new approach for vulnerability analysis of large, complex networks by leveraging the scalability and property preserving character of the finite state abstraction process; the efficacy of this vulnerability assessment framework is illustrated through analysis of fairly large-scale electric power grids. The second case study focuses on predictive analysis for complex network dynamics, with a focus on social processes on networks. This study develops a formal approach to predictability and prediction analysis that is enabled by one-dimensional abstraction techniques, and the utility of the methodology is illustrated through analysis of social processes for which standard approaches to prediction have been ineffective.

2. Complex Networks

This section begins with an informal but fairly detailed description of the classes of complex networks which are likely to admit aggressive abstraction and then introduces the mathematical framework we will use to model and analyze these networks.

2.1 Evolving networks

Complex, evolving systems are ubiquitous in nature and society, and recent research has clearly demonstrated the considerable utility of modeling and analyzing these systems as networks [e.g., Newman 2003]. In particular, focusing on networks which evolve to provide reliable performance in the presence of uncertainty and disturbances has revealed that 1.) most real world networks of practical interest belong to this class of systems, and 2.) this evolution yields networks with sufficient structure to allow informative, tractable analysis. An important example of the latter is RYF networks, which evolve robustness to familiar perturbations but become increasingly fragile to unexpected disturbances [Carlson and Doyle 2002, Laviolette et al. 2008]. Interestingly, the evolution which generates RYF also produces networks which allow *deep information from limited observations* (DILO) [Colbaugh and Glass 2003, Laviolette et al. 2008]. An example of DILO is the extent to which significant insight into the operation of complex networks can be obtained through topological analysis, with little or no consideration of the

characteristics of the network's vertices [Colbaugh and Glass 2003, Newman 2003]. The possibility to construct aggressive abstractions for complex networks, demonstrated rigorously in later sections of this paper, can be viewed as another realization of DILO.

One of the goals of this paper is to show that complex, evolving networks admit aggressive abstraction; for instance, we will show that networks with uncountable state spaces can be represented as finite state systems. As indicated above, this surprising fact becomes more understandable when viewed through the lens of RYF/DILO. RYF networks evolve to have configurations with mainly robust features and a few fragile ones. Because only the fragile features must be modeled in detail, the overall system representation can be simple and still provide a faithful model of the network.

One of the most important structures that evolves in RYF/DILO networks is the feedback control loop. Feedback regulation "robustifies" network features which are susceptible to failure, and this leads to network configurations which are complex but also highly abstractable. Indeed, even a cursory inspection of advanced technological and biological networks reveals that most of their complexity is associated with feedback control mechanisms. For example, electric power grids and the Internet contain vast numbers of feedback controllers which enable reliable and inexpensive delivery of electricity and packets, respectively, and the genomes of even simple bacteria consist mostly of genes which code for sensors, actuators, and the regulatory networks that control them. As a consequence, these networks exhibit remarkable robustness to wide variations in both environmental conditions and internal component behavior and thus require only simple models for these robust features.

The above discussion suggests that it is not completely unreasonable to expect advanced technologies and biological systems to admit aggressive abstraction. Note that the arguments are intuitively appealing in part because generally accepted models are available for these systems; among other things, these models make explicit the feedback regulation in the networks and the way this feedback robustifies system features. These "noncontroversial" models for technological and biological systems are also leveraged in subsequent sections to enable rigorous proofs to be given for the existence of aggressive abstractions for these networks. However, these arguments may be less persuasive in the case of social systems precisely because of the *lack* of accepted models for phenomena in this domain. We therefore present in what follows an alternative argument for the plausibility of the existence of aggressive abstractions which is more suited to social processes.

In contrast to the reasoning used in the discussion of technological and biological networks, which is based on network robustness, with social processes we argue that there are strong

incentives for groups of social agents working on complex problems to adopt individually simple strategies and to coordinate their actions. The simplicity of agent behavior, relative to the complexity of the problem and the overall social process, then leads to the plausibility that the overall system is abstractable. Note that in order to make these notions precise in the presence of ambiguities and uncertainties associated with social processes, we will work at a fairly high level of abstraction.

Consider a collection of social agents working on a complex problem. Suppose the agents can observe their environment and, for each possible environmental "state", must specify an action to be performed. For example, the agents may form the team responsible for planning a city's "portfolio" of responses to natural disasters. Let the agents' plan be specified in terms of a *strategy* s: $I \rightarrow A$ which maps the current *information state* $i \in I$ characterizing the environment to an *action* $a \in A$. A particular strategy s defines a response or action for each possible information state, and the challenge facing the group is to choose a strategy which provides good performance over a broad spectrum of environmental states.

If the sensors used by agents to observe their "world" produce quantized measurements and the candidate strategies which map these sensor readings to actions are implemented on digital computers, then the sets of information states I and strategies S are discrete and finite. In this case the above set-up can be encoded as a *complex systems matrix* (CSM), in which rows and columns correspond to information states and strategies, respectively, and matrix entries denote actions; thus a_{ij} is the action specified by strategy j when the world is in state i (see Figure 1).

Complex System Matrix		**strategies**		
		s_1	s_2	. . .
info states	i_1	a_{11}	a_{12}	. . .
	i_2	a_{21}	a_{22}	. . .
	⋮	⋮		⋮

Figure 1: Complex System Matrix, with rows and columns corresponding to information states and strategies, respectively, and matrix entries denoting actions associated with the given information state-strategy pair.

Consider now the simple setting of binary actions $a_{ij} \in \{0,1\}$ and the problem of choosing from the set S of all possible strategies one which provides good performance for all (or an important range of) information states. Even in this simple binary setting, selecting a strategy through exhaustive exploration of the strategy space is intractable for any real world problem. Indeed, real world social groups possess numerous "channels" through which to observe the world, and because the set I of information states grows exponentially in the number of these channels we can suppose that $|I|$ is large. It is easy to see that the number of possible strategies is exponential in $|I|$: $|S| = 2^{|I|}$. For instance, if the rows and columns of the CSM are arranged to produce a binary "counting matrix", with leftmost column $[0\ 0\ \ldots\ 0]^T$ and rightmost column $[1\ 1\ \ldots\ 1]^T$, then this encodes all strategies without duplication and has $2^{|I|}$ columns. If we make the standard assumption that computational tractability requires algorithms which scale polynomially in the size of the input set, then only a fraction $P(|I|)/2^{|I|}$ of the strategy set S can be explored, where $P(|I|)$ is a polynomial in $|I|$. It follows that only a small portion of the strategy set S can be examined by a given agent.

We now show that an effective way for social groups to implement good strategies despite the complexity indicated above is for each agent to adopt a simple strategy and for the group to appropriately coordinate the individual strategies. More specifically, we seek an alternative to the computationally intractable situation in which one "super agent" would have to learn all $|S| = 2^{|I|}$ strategies and then choose a good one. We have

Theorem 1: Each of the following methods enables tractable construction of any strategy which appears in the CSM:

1) Have each of $|I|$ agents choose an elemental strategy e_i (i.e., a column with a 'one' in the ith row and zeros everywhere else) and then form the group strategy using a linear combination of these elemental strategies.

2) Assemble $n = f|I|$ agents (for some small fraction f), partition the information state set I into n disjoint subsets I_k, and have each agent formulate a good strategy for one of these subsets; then the group monitors the world state, identifies which subset I_k is relevant, and adopts the appropriate agent's (good) strategy.

Proof (sketch)**:** Assume sufficient agents exist to implement the group strategy in question. In Strategy 1), any strategy in the CSM can be implemented as a linear combination of the e_i because the elemental columns form a basis for the column space of the CSM. Moreover, only $|I|$ parameters must be "learned" to implement this scheme for a given problem. In Strategy 2),

it is clear that by construction any strategy in the CSM is implementable in this way. Tractability follows from the fact that each agent must search only $|S_k| = 2^{(1/f)}$ strategies. ■

This result suggests that an effective way to deal with the complexity of real world social system problems is for individual agents to learn simple strategies, for instance those corresponding to a limited domain of expertise (Strategy 2)), and for society to construct (or evolve) "social institutions" which enable these individual strategies to be coordinated. Although the above analysis is simple and abstract, the result is suggestive of many readily observed situations in which simple strategies (e.g., "rules of thumb") are coordinated via social institutions (e.g., markets, cultural norms) to produce good collective outcomes.

2.2 Network models

We now present a useful framework within which to model and analyze complex, evolving networks. The key observation is that evolution to provide robust performance typically leads to networks which have a hybrid structure exhibiting both continuous and discrete dynamics. More precisely, these networks are *hybrid dynamical systems* (HDS) composed of subsystems whose dynamics evolves on continuous state spaces (e.g., manifolds) interacting with switching systems that possess discrete state sets. The HDS architecture represents an effective way to increase robustness and performance in complex systems – indeed, certain robust performance goals can be met *only* through such switching [e.g., Bemporad et al. 2007] – so it is not surprising that complex networks evolve HDS structures.

We have shown that HDS models are a natural, expressive, and tractable way to represent broad and important classes of complex networks [Colbaugh et al. 2007]. Consider, for example, advanced technologies ranging from compact and ubiquitous "embedded" systems made possible by modern electronics to continent-spanning electric power grids and global computer networks. These networked systems all consist of control software and hardware, which are discrete dynamical systems, interacting with the physical world, which is naturally represented in terms of differential equations evolving on continuous state spaces. Biological networks also possess a hybrid structure; examples from cell biology include the processes of gene regulation, cell growth and division, and cell differentiation. Complex social systems also tend to evolve hybrid structures. For instance, the interaction of Central Banks with financial markets is naturally represented as an HDS, as is the propagation of epidemics through the overlapping social contexts which make up a society.

The basic structure of an HDS is depicted in Figure 2. The schematic on the left side of the figure shows the feedback relationship between the discrete and continuous subsystems which

form the HDS. More quantitatively, the discrete system dynamics depends on the continuous system state, often because discrete state transitions are "triggered" by certain continuous state behaviors, and the particular continuous system which is "active" at a given time depends on the discrete system. The cartoon on the right of Figure 2 illustrates a common way this interaction takes place. The continuous system state space is partitioned into subsets, each of which has associated to it a different vector field. The continuous system dynamics triggers a discrete state transition when the continuous state trajectory passes from one subset to another, and this discrete state change can in turn cause switching between the vector fields which define the continuous dynamics.

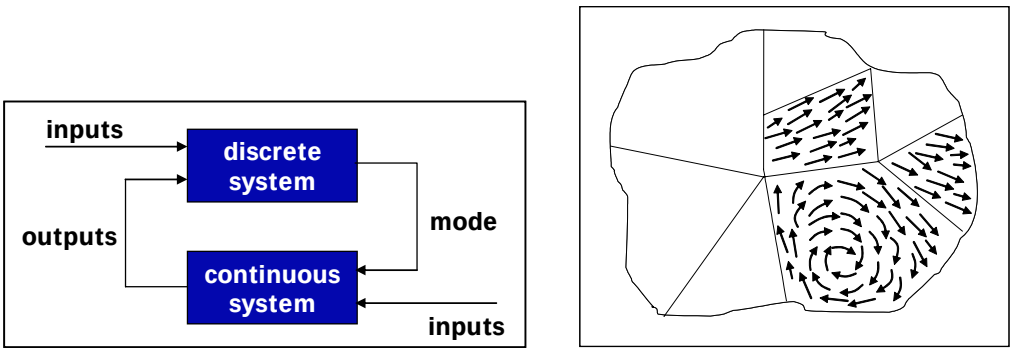


Figure 2: Schematic of an HDS (left) and HDS continuous system state space (right). The HDS diagram illustrates the feedback structure relating the discrete and continuous dynamics, while the state space cartoon provides a simple depiction of the way different vector fields can be active on different subsets of the continuous state space.

We now present quantitative definitions for the HDS models we will use in our subsequent development.

Definition 2.1: A *continuous-time HDS* is a control system

$$\Sigma_{HDSct} \qquad \begin{aligned} q+ &= h(q,k), \\ dx/dt &= f_q(x,u), \\ k &= p(x), \end{aligned}$$

where $q \in Q$ (with $|Q|$ finite) and $x \in \Re^n$ are the states of the discrete and continuous systems that make up the HDS, $u \in \Re^m$ is the control input, h defines the discrete system dynamics, the $\{f_q\}$ are a family of vector fields characterizing the continuous system dynamics, and p defines a partition of the continuous state space into subsets with labels $k \in \{1, \ldots, K\}$.

Σ_{HDSct} is thus a feedback interconnection of a discrete system, the dynamics of which evolves according to h and depends on the continuous state through subset label k, and a continuous system, with dynamics defined by the vector field f_q which is presently "active" (see Figure 2). Note that in subsequent development we will frequently consider HDS with continuous state spaces which are *bounded* subsets $X \subseteq \Re^n$, as this is usually the case in practical applications.

We will also find it useful to work with discrete-time HDS:

Definition 2.2: A *discrete-time HDS* is the control system

$$\Sigma_{HDSdt} \qquad \begin{aligned} q+ &= h(q,k), \\ x+ &= f_q(x,u), \\ k &= p(x), \end{aligned}$$

where the notation is identical to that used in Definition 2.1.

We sometimes refer to an HDS using the symbol Σ_{HDS} if the nature of the continuous system (continuous- or discrete-time) is either unimportant or clear from the context.

Stochastic versions of HDS can also be specified. For instance, we have

Definition 2.3: A *stochastic HDS* (S-HDS) is a feedback interconnection of a continuous state-dependent Markov chain {Q, P(p(x))} and a collection of stochastic differential equations indexed by the Markov chain state $q \in Q$:

$$\Sigma_{S\text{-}HDS} \qquad \begin{aligned} &\{Q, P(k)\}, \\ dx &= f_q(x)dt + G_q(x)dw, \\ k &= p(x), \end{aligned}$$

where P(k) is the (k-dependent) Markov chain transition probability matrix, w is a standard $\Re^m$-valued Weiner process, and the matrices G_q define the way this stochastic "disturbance" impacts the continuous system dynamics.

An extensive discussion of HDS and S-HDS theory and applications is beyond the scope of this paper. Such material may be found in [e.g., Bemporad et al. 2007] and the interested reader is directed to that reference for additional details.

We close this brief introduction to hybrid systems with two simple examples.

Example 2.1: Electric power grid

Consider the simple two bus power grid depicted in Figure 3 [Wedeward 2006]. This system can be naturally represented as an HDS, with the continuous system modeling the generator

and load dynamics as well as the power flow constraints and the discrete system capturing the switching associated with the grid's "protection logic" (e.g., line tripping, load shedding).

To provide a simple, concrete illustration of this fact, we assume that the generator and load dynamics can be described using standard models and that the sole protection logic implemented with the grid is line tripping (without recovery). In this case, we can model the generator and load dynamics as

$$\begin{aligned} T'_{do}\, dE'_q/dt &= -\frac{X_d}{X'_d}E'_d + \frac{X_d - X'_d}{X'_d}V_1\cos(\delta-\theta_1) + E_{fd} \\ d\delta/dt &= 2\pi 60(\omega-\omega_s) \\ M\, d\omega/dt &= P_M - P_G - D(\omega-\omega_s) \\ T_L\, dP_L/dt &= -P_L + P_0(t)\frac{|V_2|^{\alpha}}{V_{ref}} \\ T_L\, dQ_L/dt &= -Q_L + Q_0(t)\frac{|V_2|^{\beta}}{V_{ref}} \end{aligned} \qquad (2.1)$$

where E'_q, δ, ω, P_L, and Q_L are state variables, V_1, V_2, and θ_1 are network (algebraic) variables, E_{fd} and P_M are (generator) control inputs, and all other terms are (constant) parameters.

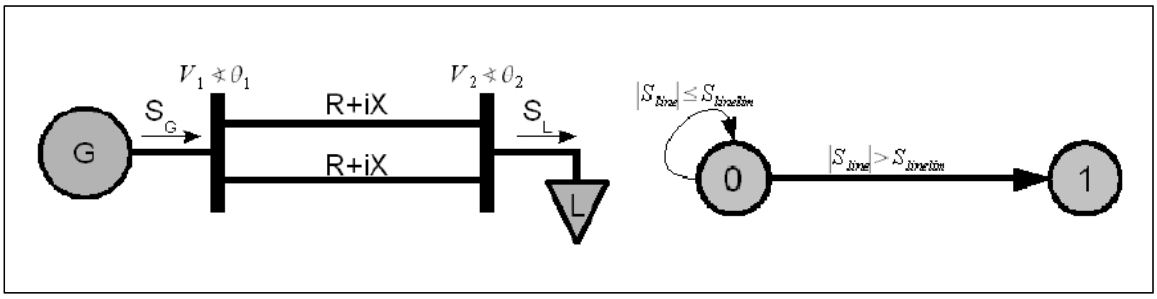


Figure 3: Two bus power system with generator, two transmission lines, and load (left) and simple state diagram illustrating discrete dynamics associated with line tripping protection logic (right).

The power flow constraints can be written

$$\begin{aligned} 0 &= \text{imag}(S_{bus}(1)) = (1-0.5q_{lt})(BV_1^2 - V_1V_2(G\sin(\theta_1-\theta_2) + B\cos(\theta_1-\theta_2))) - Q_G \\ 0 &= \text{imag}(S_{bus}(2)) = (1-0.5q_{lt})(BV_2^2 + V_1V_2(G\sin(\theta_1-\theta_2) + B\cos(\theta_2-\theta_1))) + (1-0.05q_{ls})Q_L \\ 0 &= \text{real}(S_{bus}(1)) = (1-0.5q_{lt})(GV_1^2 - V_1V_2(G\cos(\theta_1-\theta_2) - B\sin(\theta_1-\theta_2))) - P_G \\ 0 &= \text{real}(S_{bus}(2)) = (1-0.5q_{lt})(GV_2^2 - V_1V_2(G\cos(\theta_1-\theta_2) + B\sin(\theta_2-\theta_1))) + (1-0.05q_{ls})P_L \end{aligned} \qquad (2.2)$$

where P_G and Q_G are given by

$$P_G = \frac{1}{X'_d} E'_q V_1 \sin(\delta - \theta_1) + \frac{X'_d - X_q}{2X'_d X_q} V_1^2 \sin(2(\delta - \theta_1))$$

$$Q_G = \frac{1}{X'_d} E'_q V_1 \cos(\delta - \theta_1) - \frac{X'_d - X_q}{2X'_d X_q} V_1^2 + \frac{X'_d - X_q}{2X'_d X_q} V_1^2 \cos(2(\delta - \theta_1))$$

and S_{bus} is complex power, θ_2 is the final network variable, q_{lt} is a discrete system state variable described below, and all terms not yet defined are parameters.

The HDS continuous system is specified by equations (2.1)-(2.2). More precisely, the four algebraic equations (2.2) define the network variables V_1, θ_1, V_2, and θ_2 in terms of the state variables E'_q, δ, ω, P_L, and Q_L, and the latter in turn evolve according to the differential equations (2.1). The HDS discrete system defines the evolution of the discrete system state $q_{lt} \in \{0,1\}$, which corresponds to the number of lines tripped out of service. This discrete dynamics is specified diagrammatically in Figure 3, with S_{line} given by

$$\left|S_{line}\right| = \left|V_1 e^{i\theta_1}((V_1 e^{i\theta_1} - V_2 e^{i\theta_2})/Z)^*\right|$$

where Z is the line impedance.

Example 2.2: Switching diffusion process

Consider next a class of switching diffusion processes. Such models can be used to study a wide range of important phenomena [Prajna et al. 2007] and, indeed, we will use the basic structure introduced here to model real world systems later in this paper. Switching diffusion processes are a special case of the stochastic hybrid system model given in Definition 2.3. As a simple example, consider an S-HDS in which the continuous system is a stochastic differential equation with multiplicative noise:

$$dx = A_q x\, dt + \sigma(x)\, dw$$

where $x \in \Re^2$ is the continuous system state, $q \in \{1,2\}$ is the discrete system state, w is a scalar Wiener process, $\sigma = [0 \quad 0.5x_2]^T$, and the matrices A_1, A_2 are given by

$$A_1 = \begin{bmatrix} -5 & -4 \\ -1 & -2 \end{bmatrix}, \quad A_2 = \begin{bmatrix} -2 & -4 \\ 20 & -2 \end{bmatrix}$$

The discrete system is a continuous-time Markov chain with state set Q = {1,2} and continuous state-dependent transition probabilities. These transition probabilities are defined in terms of transition *rates* $\lambda_{qq'}(x)$, where $\lambda_{qq'}(x) \geq 0$ if $q \neq q'$ and $\Sigma_{q'} \lambda_{qq'}(x) = 0 \ \forall q$. The rates $\lambda_{qq'}(x)$ are related to the state transition probabilities as follows:

$$P\{q(t+\Delta) = q' \mid q(t) = q\} = \begin{cases} \lambda_{qq'}(x(t))\Delta + o(\Delta) & \text{if } q \neq q' \\ 1 + \lambda_{qq}(x(t))\Delta + o(\Delta) & \text{if } q = q' \end{cases}$$

where $\Delta > 0$ [Bujorianu and Lygeros 2004].

Figure 4 illustrates the sort of behavior which can be generated by this simple example of switching diffusion dynamics. The system trajectory shown (solid curve) corresponds to initial conditions $x(0) = [0 \quad 3]^T$, $q(0) = 1$ and discrete state transition rate $\lambda = 10$. It is interesting to observe the way the two deterministic "components" of the process, $dx/dt = A_1x$ (dashed) and $dx/dt = A_2x$ (dash-dotted), combine to form the S-HDS trajectory.

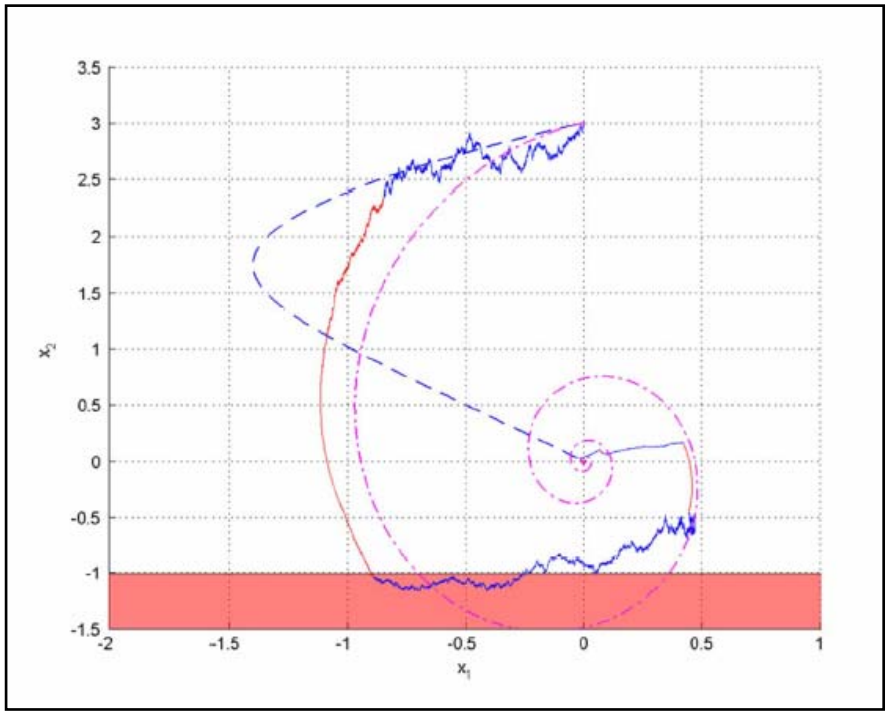


Figure 4: Sample trajectory of the switching diffusion process (solid) described in Example 2.2 along with trajectories for the two deterministic components of the process, $dx/dt = A_1x$ (dashed) and $dx/dt = A_2x$ (dash-dotted).

3. Finite State Abstraction

The dynamics of complex networks ordinarily evolve on *hybrid* state spaces composed of both continuous and discrete sets. For example, the states of the simple electric power grid in Example 2.1 include five continuous variables, which model the generator and load dynamics, and one discrete variable, which specifies the number of tripped lines. This same structure is present in larger, more realistic power grid models [Ilic and Zaborszky 2000] and many other advanced technologies (e.g., [Glass et al. 2007]). Biological networks describing the behaviors of systems ranging from cells to ecosystems are also usefully represented in this way. Cell metabolism, for instance, can be modeled using stochastic differential equations for the (continuous) concentrations of the various metabolites and discrete switching to capture transitions between different modes of metabolism. Social processes from epidemics to opinion formation to organizational dynamics also have hybrid state spaces. As one example, recent work has revealed that global epidemics can be well-modeled using simple stochastic differential equations for the fractions of susceptible, infected, and recovered populations, provided that the travel patterns of these populations (e.g., via commercial airlines) are captured [Colizza et al. 2006]. We show in [Colbaugh and Glass 2007] that these latter dynamics can be represented as a finite state Markov process, making the complete model a hybrid system.

The large-scale, hybrid nature of complex network dynamics represents a major challenge in their analysis. Clearly, developing methods for managing this considerable complexity would be an important advance. In this section we present a framework for modeling infinite state complex networks with finite state models in a way that preserves the relevant properties of the original network. The advantages of obtaining *equivalent* finite state models cannot be overstated. For instance, finite state systems can be analyzed using the powerful methods of theoretical computer science (see, e.g., [Milner 1989], [Clarke et al. 1999], [Alur et al. 2000] for an introduction to some of these methods), and an equivalence between the original and finite state models enables analytic conclusions derived for the latter to be applied directly to the former. Additionally, finite state abstractions for infinite state systems usually contain only a small subset of the parameters of the original model, so that abstraction provides an effective means of managing the parametric uncertainty which is so prevalent in complex network models.

We begin our discussion of finite state abstractions for complex networks by introducing the basic abstraction concept and reviewing some technical background which will be useful in our development. We then present the main abstraction results, including methods for approximate abstraction, and provide efficient computational techniques for realizing these abstractions.

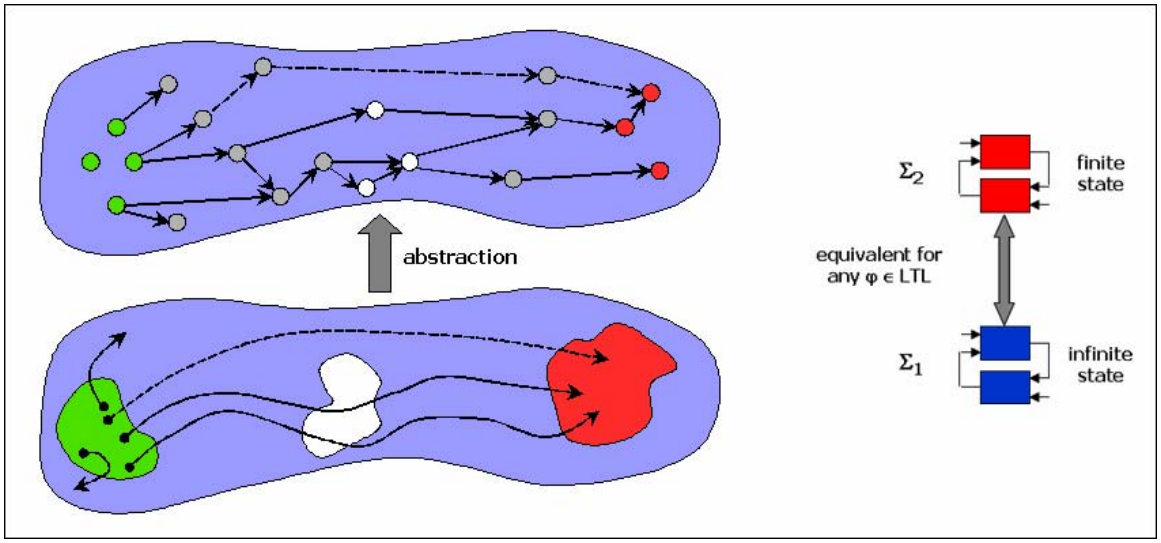


Figure 5: Basic concepts associated with finite state abstraction. Cartoon at left illustrates that abstraction preserves dynamical properties: infinite state trajectories of original system (curves in blue region at bottom) are mapped to equivalent finite state trajectories (sequences of state transitions at top). Diagram at right highlights the focus on property preserving transformations of infinite state HDS to feedback interconnections of finite state systems.

3.1 Preliminaries

The basic notion of a finite state abstraction for an infinite state system is illustrated in the cartoon on the left side of Figure 5. Consider a complex network with states that evolve on a continuous space and an analysis question of interest. Such a situation is depicted in Figure 5, where the continuous dynamics are shown as curves on a continuous state space (blue region at bottom left) and the analysis question involves deciding whether states in the green region can evolve to the red region. Reachability questions of this sort are quite difficult to answer for generic complex networks. However, if it is possible to construct a finite state abstraction of the network which possesses equivalent dynamics, then the analysis task becomes much easier. To see this, observe from Figure 5 that a finite state abstraction of the original dynamics takes the form of a graph, where the states are graph vertices (nodes within the blue region at top left) and admissible state transitions define the graph's directed edges. Reachability analysis is straightforward with a graph – check whether there exists a directed path from a green vertex to a red vertex – and if the complex network and its abstraction have equivalent reachability properties then this graph analysis also characterizes reachability for the original system.

Reachability assessment, while valuable, is rarely sufficient to answer real world analysis questions. For instance, suppose that the red region in Figure 5 is the set of failure states. It may be of interest to determine if all system trajectories which reach the red region first pass through the white, observable, region (so there is "warning" of impending failure), or whether all trajectories which reach the red region subsequently return to the blue ("normal") region (and thereby "recover" from failure). Addressing these more sophisticated questions requires that the analysis be conducted using a language which allows refined, nuanced description of, and reasoning about, network dynamics. We propose that *linear temporal logic* (LTL) provides such a language [Clarke et al. 1999]. LTL offers a precise, expressive framework for analyzing dynamical phenomena which is similar to natural language and is thus convenient to use. It is an extension of propositional logic which enables temporal issues to be considered.

As we wish to use LTL to analyze the dynamics of complex networks and we model these networks as HDS, we tailor our definition of LTL to be compatible with this setting:

Definition 3.1: The *syntax* of LTL consists of

- *atomic propositions* (q,k), where $q \in Q$ is an HDS discrete state and $k \in K$ is a label for a subset in the continuous system state space partition;
- *formulas* composed from atomic propositions using a grammar of Boolean ($\varphi \vee \theta$, $\neg\varphi$) and temporal ($\varphi \boldsymbol{U} \theta$, $\bigcirc\varphi$) operators.

The *semantics* of LTL follows from the interpretation of formulas on trajectories of HDS, that is, on sequences of (q, k) pairs: $(\mathbf{q}, \mathbf{k}) = (q_0, k_0), (q_1, k_1), \ldots, (q_T, k_T)$.

The Boolean operators $\vee$ and $\neg$ are disjunction and negation, as usual. The temporal operators $\boldsymbol{U}$ and $\bigcirc$ are read "until" and "next", respectively, with $\varphi \boldsymbol{U} \theta$ specifying that φ must hold until θ holds and $\bigcirc\varphi$ signifying that φ will be true at the next time instant (see [Clarke et al. 1999] for a more careful description of these operators).

We are now in a position to make precise the notion of property preserving abstraction: we seek abstractions which preserve LTL, that is, which are such that for any LTL formula ϕ either both the system and its abstraction satisfy ϕ or neither do. More quantitatively, for system Σ_1 and abstraction Σ_2, the abstraction is property preserving if and only if $\{\Sigma_1 \models \varphi\} \Leftrightarrow \{\Sigma_2 \models \varphi\}$ for all LTL formulas φ, where $\models$ denotes formula satisfaction (see Figure 5).

Bisimulation is a powerful method for abstracting finite state systems to yield simpler finite state systems which are equivalent from the perspective of LTL [Milner 1989]. However, the problem of constructing finite state bisimulations for continuous state systems is largely

unexplored. Indeed, one of the main contributions of this paper is to develop a collection of mathematically rigorous, computationally tractable methods for obtaining finite bisimulations of HDS models.

Bisimulation is typically defined for transition systems, so we first introduce this notion (see [Milner 1989] for additional details):

Definition 3.2: A *transition system* is a four-tuple $T = (S, \rightarrow, Y, h)$ with state set S, transition relation $\rightarrow \subseteq S \times S$, output set Y, and output map $h: S \rightarrow Y$. T is *finite* if $|S|$ is finite.

Transition systems are a very general form of dynamical system, with the transition relation $\rightarrow$ defining the admissible state transitions (so that $(q, q') \in \rightarrow$, usually denoted $q \rightarrow q'$, if T can undergo a state transition from q to q').

Bisimilar transition systems share a common output set and have dynamics which are equivalent from the perspective of these outputs:

Definition 3.3: Transition systems $T_S = (S, \rightarrow_S, Y, h_S)$ and $T_P = (P, \rightarrow_P, Y, h_P)$ are *bisimilar* via relation $R \subseteq S \times P$ iff:

- $s \sim p \Rightarrow h_S(s) = h_P(p)$ (R respects observations);
- $s \sim p$, $s \rightarrow_S s' \Rightarrow \exists\ p' \sim s'$ such that $p \rightarrow_P p'$ (T_P simulates T_S, denoted $T_S \angle T_P$);
- $p \sim s$, $p \rightarrow_P p' \Rightarrow \exists\ s' \sim p'$ such that $s \rightarrow_S s'$ ($T_P \angle T_S$).

A standard result from theoretical computer science shows that bisimulation preserves LTL, a fact which will be of considerable value in the subsequent development:

Proposition 3.1 [Milner 1989]**:** If T_1 and T_2 are bisimilar transition systems and φ is an LTL formula then $\{T_1 \models \varphi\} \Leftrightarrow \{T_2 \models \varphi\}$.

The following alternative definition for bisimulation is easily shown to be equivalent to the one presented in Definition 3.3 and provides a convenient basis upon which to develop finite bisimulations for continuous state transition systems:

Definition 3.4: A finite partition $\Phi: S \rightarrow P$ of the state space S of transition system $T = (S, \rightarrow, Y, h)$ naturally induces a *quotient transition system* $T/\sim = (P, \rightarrow_\sim, Y, h_\sim)$ of T, provided that

- $\Phi(s) = \Phi(s')$ (denoted $s \sim s'$) $\Rightarrow h(s) = h(s')$;
- $h_\sim(p) = h(s)$ if $p = \Phi(s)$;
- $\rightarrow_\sim$ is defined so that $\Phi(s) \rightarrow_\sim \Phi(s')$ iff $s \rightarrow s'$.

Equivalently, the quotient transition system T/~ can be defined by specifying the equivalence relation $R \subseteq S \times S$ directly.

Transition system T and its quotient T/~ are bisimilar if an additional condition holds:

Proposition 3.2: Suppose T/~ is defined as in Definition 3.4 and, in addition, $\Phi(s) \rightarrow_{\sim} \Phi(s') \Rightarrow \forall$ s'' ~ s $\exists$ s''' ~ s' such that s'' $\rightarrow$ s'''. Then T and T/~ are bisimilar.

The proof of Proposition 3.2 follows easily from standard theoretical computer science results and is omitted.

Proposition 3.2, while useful, provides no guidance regarding the actual construction of a bisimulation-inducing finite partition Φ. Moreover, the result does not help in the identification of systems for which such partitions exists. This latter point is crucial because most continuous state transition systems do not admit finite bisimulations [Alur et al. 2000].

We now introduce a class of continuous state (control) systems which is both important, in that many real world systems belong to this class, and finite state abstractable.

Definition 3.5: The continuous-time system dx/dt = f(x, u), with f: $\Re^n \times \Re^m \rightarrow \Re^n$, is *differentially flat* if there exists (flat) outputs $z \in \Re^m$ such that z = H(x), x = F_1(z, dz/dt, …, $d^r z/dt^r$), and u = F_2(z, dz/dt, …, $d^r z/dt^r$) for some maps H, F_1, and F_2.

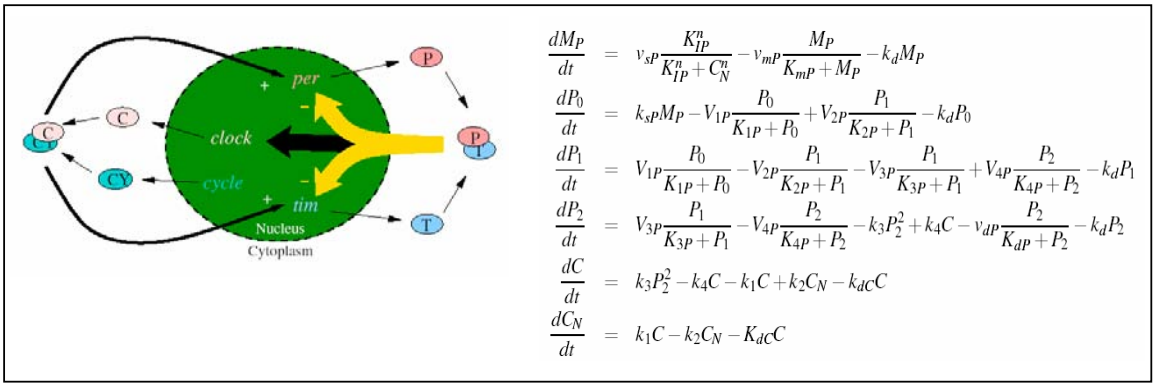
Definition 3.6: The discrete-time system x+ = f(x, u) is *difference flat* (with memory k) if there exists (flat) outputs $z \in \Re^m$ such that z = H(x), x(t) = F_1(z(t), z(t+1), …, z(t+k−1)), and u(t) = F_2(z(t), z(t+1), …, z(t+k−1)).

Differentially flat systems are discussed at length in the report [Martin et al. 2003]. Our definition of difference flat systems is a natural extension of this notion to discrete-time systems. It is easy to show that for flat systems, any flat output trajectory z*: [0, T] $\rightarrow \Re^m$ is realizable (provided z* is compatible with x(0)), and that specifying a trajectory for the flat outputs completely defines the system evolution.

As indicated above, many real world systems are flat. For example, all controllable linear systems are flat, as are all (static or dynamic) feedback linearizable systems. Moreover, complex, evolving networks often possess continuous dynamics which are flat. [Martin et al. 2003] presents a extensive collection of advanced technologies whose dynamics are differentially flat. The next example illustrates that naturally evolving systems can also be flat.

Example 3.1: *Drosophila* circadian rhythm

Many aspects of the physiology of living organisms oscillate with a period of approximately 24 hours, corresponding to the duration of a day, and the molecular basis for this circadian rhythm has been quantified in several organisms. For instance, a model for the gene regulatory network responsible for circadian rhythm in *Drosophila* (fruit fly) is shown in Figure 6 [Goncalves and Yi 2004]. The diagram at the left of Figure 6 depicts the main elements of the network, including the regulatory feedback loops. The model itself consists of the six coupled ordinary differential equations shown at the right of the figure, where M_P, P_0, P_1, P_2, C, and C_N are state variables corresponding to the concentrations of the constituents of the circadian rhythm gene network, v_{sP} can be viewed as an exogenous (control) input associated with the light-dark cycle of the environment, and all other terms are constant model parameters.



$$\frac{dM_P}{dt} = v_{sP}\frac{K_{IP}^n}{K_{IP}^n + C_N^n} - v_{mP}\frac{M_P}{K_{mP} + M_P} - k_d M_P$$

$$\frac{dP_0}{dt} = k_{sP}M_P - V_{1P}\frac{P_0}{K_{1P} + P_0} + V_{2P}\frac{P_1}{K_{2P} + P_1} - k_d P_0$$

$$\frac{dP_1}{dt} = V_{1P}\frac{P_0}{K_{1P} + P_0} - V_{2P}\frac{P_1}{K_{2P} + P_1} - V_{3P}\frac{P_1}{K_{3P} + P_1} + V_{4P}\frac{P_2}{K_{4P} + P_2} - k_d P_1$$

$$\frac{dP_2}{dt} = V_{3P}\frac{P_1}{K_{3P} + P_1} - V_{4P}\frac{P_2}{K_{4P} + P_2} - k_3 P_2^2 + k_4 C - v_{dP}\frac{P_2}{K_{dP} + P_2} - k_d P_2$$

$$\frac{dC}{dt} = k_3 P_2^2 - k_4 C - k_1 C + k_2 C_N - k_{dC} C$$

$$\frac{dC_N}{dt} = k_1 C - k_2 C_N - K_{dC} C$$

Figure 6: Model for circadian rhythm in *Drosophila*. Diagram at left shows the main elements of the gene regulatory network, including the negative (yellow) and positive (black) feedback loops. Differential equations at right quantify the network dynamics.

As is evident from Definition 3.5, a differentially flat system possesses (flat) outputs, equal in number to the number of inputs, which permit the system states and inputs to be recovered through algebraic manipulation of these outputs and their time derivatives. In the case of *Drosophila* circadian rhythm, C_N is a flat output. To see this, note that C and its time derivatives can be obtained from the sixth equation through manipulation of C_N and its derivatives. These terms, in turn, permit P_2 (and its derivatives) to be obtained from the fifth equation, and continuing in this way up the "chain" of equations gives all of the states and the input v_{sP} [Colbaugh et al. 2007].

3.2 Basic results

We now demonstrate that hybrid systems with (difference or differentially) flat continuous systems admit finite state bisimulations. As HDS provide a powerful framework within which to model complex networks and many real world systems possess flat continuous dynamics, this result is of considerable practical as well as theoretical interest.

Consider, then, an HDS of the form given in Definition 2.1 or 2.2. We begin by providing a transition system representation for the continuous system dynamics of HDS:

Definition 3.7: The *transition system model* T_{HDSc} for the continuous system portion of Σ_{HDS} is the collection $T_{HDSc} = \{T_q^k\}$, with one transition system $T_q^k = (X_q^k, \rightarrow_q^k, Y_q^k, h_q^k)$ specified for each pair (q, k). In this specification, X_q^k and Y_q^k are (bounded) state and output sets, respectively, and the maps h_q^k: $X_q^k \rightarrow Y_q^k$ will be used to specify the relevant quotient partitions (see Definition 3.4); the transition relations $\rightarrow_q^k$ are defined as

- discrete-time continuous system: $x \rightarrow_q^k x'$ iff $\exists$ u such that $x' = f_q(x, u)$ on subset k;
- continuous-time continuous system: $x \rightarrow_q^k x'$ iff there is a trajectory x: $[0, T] \rightarrow X_q^k$ of $dx/dt = f_q(x, u)$, a time $t' \in (0, T)$, and adjacent quotient partitions (labeled) y, $y' \in Y_q^k$ such that $x(0) = x$, $x(T) = x'$, $x([0, t')) \subseteq y$, and $x((t', T]) \subseteq y'$.

We make the standard assumption that k: $X \rightarrow K$ partitions the state space X into polytopes and that all HDS discrete system transitions are triggered by k transitions.

Definition 3.7 allows an HDS to be modeled as a feedback interconnection of two transition systems, one with continuous state space and one with finite state set:

Definition 3.8: The *transition system* T_{HDS} associated with the HDS given in Definition 2.1 or 2.2 is a feedback interconnection of 1.) the continuous system transition system $T_{HDSc} = \{T_q^k\}$ and 2.) the transition system associated with the HDS discrete system, defined as $T_{HDSd} = (Q, \rightarrow_d, Q, id)$, where $q \rightarrow_d q'$ iff $\exists$ k such that $q' = h(q, k)$ and id is the identity map. Thus $T_{HDS} = (Q \times X, \rightarrow_{HDS}, Q \times Y, h_{HDS})$, where $Q \times X = \cup_q (\cup_k \{q\} \times X_q^k)$, $Q \times Y = \cup_q (\cup_k \{q\} \times Y_q^k)$, and the definitions for $\rightarrow_{HDS}$ and h_{HDS} follow immediately from the transition relation and output map definitions specified for T_{HDSc} and T_{HDSd}.

Because the transition system T_{HDSd} corresponding to the HDS discrete system is already a finite state system, the main challenge in abstracting HDS to finite state systems is associated with finding finite state bisimulations for the continuous systems $T_{HDSc} = \{T_q^k\}$. This is made explicit in the following

Theorem 2: If each transition system T_q^k associated with T_{HDS} is bisimilar to its finite quotient transition system $T_q^k/\sim = (Y_q^k, \rightarrow_\sim, Y_q^k, id)$ and the state space quotient partitions defined by the h_q^k satisfy a mild compatibility condition then T_{HDS} admits a finite bisimulation.

Proof: Consider the transition system T_{HDS} given in Definition 3.8. Let T be the transition system obtained from T_{HDS} by replacing each T_q^k in T_{HDS} with $T_q^k/\sim$, so that T = $(Q \times Y, \rightarrow, Q \times Y, id)$ with $\rightarrow$ defined in the obvious way. Note first that T is finite. Referring to Definition 2.1, consider two adjacent state space regions labeled k_i, k_j and their shared boundary (as defined by p(x)). Suppose that the partitions on k_i, k_j induced by the maps h_q^{ki}, h_q^{kj} "line up" at their common boundary, so that the segmentations of the boundary implied by h_q^{ki} and by h_q^{kj} are identical. In this case, it is easy to check that the quotient map Φ: $Q \times X \rightarrow Q \times Y$ induced by the h_q^k satisfies 1.) $\Phi(s) = \Phi(s') \Rightarrow h_{HDS}(s) = h_{HDS}(s')$, 2.) $\Phi(s) \rightarrow \Phi(s') \Leftrightarrow s \rightarrow_{HDS} s'$, and 3.) $\Phi(s) \rightarrow \Phi(s') \Rightarrow \forall\ s'' \sim s\ \exists\ s''' \sim s'$ such that $s'' \rightarrow_{HDS} s'''$. Therefore, from Proposition 3.2, T_{HDS} and T are bisimilar. ■

While it is possible to relax the state space partition compatibility condition given above, we do not pursue this as the condition is mild and straightforward to satisfy in applications.

Theorem 2 shows that the key step in obtaining a finite state bisimulation for hybrid system T_{HDS} (and so Σ_{HDS}) is constructing bisimulations for the continuous state transition systems T_q^k. We therefore focus on this latter problem for the remainder of this section. Our first basic result along these lines concerns difference flat continuous systems and is summarized in

Theorem 3: Given any finite partition π: $Z \rightarrow Y$ of the flat output space Z of a difference flat system, the associated transition system $T_F = (X, \rightarrow, Y, \pi \circ H)$ admits a bisimilar quotient $T_F/\sim$.

Proof: Consider the equivalence relation R that identifies state pairs (x, x') which generate identical sets of k-length output symbol sequences $y = y_0\ y_1 \ldots y_{k-1}$, and the quotient system $T_F/\sim$ induced by R. Clearly R defines a finite partition of X (both |Y| and k are finite), and $x \sim x' \Rightarrow \pi \circ H(x) = \pi \circ H(x')$ so that R respects observations. $T_F \angle T_F/\sim$ follows immediately from the definition of quotient systems. To see that $T_F/\sim \angle T_F$, note that flatness ensures that *any* symbol string $y = y_k\ y_{k+1} \ldots$ is realizable by transition system T_F; thus $x \sim x'$ at time t implies that x and x' can transition to equivalent states at time $t + 1$, and from Definition 3.3 T_F and $T_F/\sim$ are bisimilar. ■

Remark 3.1: Efficient algorithms exist for checking if a given system is difference flat, so that Theorem 3 provides a practically implementable means of identifying discrete-time continuous state systems which admit finite bisimulation. For instance, discrete-time feedback linearizable

systems are flat [Colbaugh et al. 2007] and there are fast algorithms for checking feedback linearizability for both continuous-time and discrete-time systems [e.g., Sontag 1998].

Remark 3.2: The trajectory of the flat outputs completely defines the evolution of a difference flat system. Thus, because *any* finite partition of flat output space induces a finite bisimilar quotient for a flat system, the flat output space partition can be refined to yield any desired abstraction resolution.

An analogous result holds for differentially flat continuous systems. Our development of this result requires the following lemmas.

Lemma 3.1: A control system is differentially flat iff it is dynamic feedback linearizable.
Proof: See [Aranda-Bricaire et al. 1995]. ■

Lemma 3.2: Control system Σ admits a finite bisimulation iff any representation of Σ obtained through coordinate transformation and/or invertible feedback also admits a finite bisimulation.
Proof: The proof is straightforward. ■

Taken together, Lemmas 3.1 and 3.2 suggest the following procedure for constructing finite bisimulations for differentially flat systems: 1.) transform the flat system of interest into a linear control system via feedback linearization (possibly enlarging the state space in the process), 2.) compute a finite bisimulation for the linear system, and 3.) map the bisimilar model back to the original system representation (if desired). In view of these results, we focus in what follows on building finite bisimulations for linear control systems.

In particular, consider as the control system of interest one "chain" of a Brunovsky normal form (BNF) system Σ_{BNF} [Sontag 1998]:

$$dx_1/dt = x_2,$$
$$dx_2/dt = x_3,$$
$$\dots$$
$$dx_n/dt = u.$$

Concentrating on this system entails no loss of generality, as any controllable linear system can be modeled as a collection of these single chain systems, one for each input, and the decoupled nature of the chains ensures that we can abstract each one independently and then simply "patch" the abstractions together to obtain an abstraction for the original (multi-input) system.

Consider the following partition of the (assumed bounded) state space $X \subseteq \Re^n$ of Σ_{BNF}:

Definition 3.9: The *partition* π_ε is the map π_ε: X $\rightarrow$ Y which partitions X into subsets y_{is} = {x $\in$ X | $x_1 \in [i\varepsilon, (i+1)\varepsilon)$, sign($x_2$) = s_1, …, sign(x_n) = s_{n-1}}, where i is an integer and s is an (n−1)-vector of "signs" corresponding to a particular orthant of X.

Thus π_ε partitions X into "slices" of width ε of the orthants of the n-dimensional space X, with each slice orthogonal to the x_1-axis.

We are now in a position to state

Theorem 4: The transition system T_{BNF} = (X, $\rightarrow$, Y, π_ε) associated with system Σ_{BNF} and partition π_ε admits a finite bisimilar quotient $T_{BNF}/\sim$ = (Y, $\rightarrow_\sim$, Y, id).

Proof: $T_{BNF}/\sim$ is finite because |Y| is finite. Assume $\rightarrow_\sim$ is constructed so that $\pi_\varepsilon(x) \rightarrow_\sim \pi_\varepsilon(x') \Leftrightarrow x \rightarrow x'$ (as specified in Definition 3.4). Then all conditions given in Definition 3.4 are satisfied, and from Proposition 3.2 we need only show $\pi_\varepsilon(x) \rightarrow_\sim \pi_\varepsilon(x') \Rightarrow \forall\, x'' \sim x\ \exists\, x''' \sim x'$ such that $x'' \rightarrow x'''$. This amounts to demonstrating that if x can be driven through face F of slice $\pi_\varepsilon(x)$ then any x'' in that slice can be driven through F as well; the latter result follows from straightforward (though tedious) checking that this behavior holds for the system $x_1^{(n)} = u$ on each orthant of X. ■

Remark 3.3: The result given in Theorem 4 is most useful in situations where the control input u can be chosen large relative to the "drift" of the system. Applications in which control authority is limited will be addressed in the next subsection.

We close this discussion with a simple example.

Example 3.2: Discrete-time BNF control systems

It is easy to show that the discrete-time linear system in BNF

$$x_1+ = x_2,$$
$$x_2+ = x_3,$$
$$\dots$$
$$x_n+ = u$$

is difference flat with flat output x_1, so from Theorem 3 it follows that this system admits a finite bisimilar quotient system on any bounded state space $X \subseteq \Re^n$. In particular, let π: X $\rightarrow$ Y be any finite, "hypercubic" partition of X (i.e., π partitions X into regular n-dimensional hypercubes whose edges align with the coordinate axes). Then the quotient system T/~ = (Y, $\rightarrow_\sim$, Y, id), with $\rightarrow_\sim$ constructed according to Definition 3.4, is one such finite bisimilar quotient.

Remark 3.3: An alternative finite bisimulation result for discrete-time linear systems is derived in [Tabuada and Pappas 2006].

3.3 Approximate bisimulation

The previous subsection shows that HDS with continuous systems that are difference flat or differentially flat admit finite state bisimulations. Intuitively, a bisimulation relation between T_{HDS}, the transition system associated with Σ_{HDS}, and a finite state transition system T is a relation between the state sets of T_{HDS} and T describing how any trajectory of T_{HDS} can be mapped to a trajectory of T which has an identical output trajectory, and vice versa. Because HDS with flat continuous systems are a broad and important class of dynamical system, this result is of considerable practical value. However, the requirement that the trajectories of T_{HDS} and T possess *identical* output traces is too strong for some applications. For example, it may be sufficient to require the output trajectories of the two transition systems to be close, or to consider only a subspace of the state space of T_{HDS} when relating the state sets of T_{HDS} and T. This subsection focuses on these notions of *approximate* bisimulation.

Consider first the concept of approximate bisimulation introduced in [Girard and Pappas 2007]. This concept is most naturally defined for labeled, metric transition systems:

Definition 3.10: A *labeled, metric transition system* is a tuple $T_m = (S, L, \rightarrow, Y, h)$, where S and h are the state set and output map, as before, the output set Y is equipped with a metric d, L is a set of labels, and the dynamics defined by the transition relation $\rightarrow \subseteq S \times L \times S$ depends on the labels as well as the states.

The notion of approximate bisimulation introduced in [Girard and Pappas 2007] relaxes the requirements of a bisimulation relation as follows:

Definition 3.11: Let $T_{m1} = (S_1, L, \rightarrow_1, Y, h_1)$ and $T_{m2} = (S_2, L, \rightarrow_2, Y, h_2)$ be two labeled, metric transition systems and $\varepsilon \geq 0$ be a given precision. $R \subseteq S_1 \times S_2$ is an *ε-approximate bisimulation relation* between S_1 and S_2 if for all $(s_1, s_2) \in R$:

- $d(h_1(s_1), h_2(s_2)) \leq \varepsilon$;
- $\forall\ s_1 \rightarrow_1^l s_1'\ \exists\ s_2 \rightarrow_2^l s_2'$ such that $(s_1', s_2') \in R$;
- $\forall\ s_2 \rightarrow_2^l s_2'\ \exists\ s_1 \rightarrow_1^l s_1'$ such that $(s_1', s_2') \in R$.

Transition systems T_{m1} and T_{m2} which possess an ε-approximate bisimulation relation R are said to be *approximately bisimilar with precision ε*, denoted $T_{m1} \sim_\varepsilon T_{m2}$.

The precision ε provides a bound on the proximity of output trajectories of T_{m1} and T_{m2}, in that if $T_{m1} \sim_\varepsilon T_{m2}$ then for any output trajectory $y_1(0)\ y_1(1)\ \ldots\ y_1(k)$ of T_{m1} there is an output trajectory $y_2(0)\ y_2(1)\ \ldots\ y_2(k)$ of T_{m2} such that $d(y_1(i), y_2(i)) \leq \varepsilon\ \forall i$, and conversely.

We now show that differentially flat systems admit finite state approximate bisimulations of arbitrarily accurate precision ε and, moreover, that this approximate bisimulation is realizable with moderate control authority; thus this result is of significant practical interest. We begin by specifying the "time-sampled", labeled, metric transition system associated with differentially flat system Σ_F. Let U denote the set of measurable, bounded control input trajectories for Σ_F, $\mathbf{u} \in U$ be one such input trajectory, and x(T, x, **u**) be the state reached by Σ_F at time T under input **u** starting from initial condition x.

Definition 3.12 The *labeled, metric transition system* $T_{F,T}$ associated with flat system Σ_F is the tuple $T_{F,T} = (X, U, \rightarrow_T^{\mathbf{u}}, X, id)$, where X and U are the sets of flat system states and input trajectories, respectively, $(x, \mathbf{u}, x') \in \rightarrow_T^{\mathbf{u}}$ iff $x' = x(T, x, \mathbf{u})$, and the metric d on X is given by $d(y, z) = \max[|y_1 - z_1|, \ldots, |y_n - z_n|]$.

Note that $T_{F,T}$ is time-sampled in that only state transitions achievable in time T are considered.

Assume that the state space X of Σ_F is a bounded subset of $\Re^n$ and let $[X]_\eta = \{x \in X \mid x_i = k_i\eta\}$, with the k_i integers, denote a "lattice" discretization of X of Σ_F. We are now in a position to state our approximate bisimulation result for differentially flat systems:

Theorem 5: Consider the transition system $T_{\eta,T} = ([X]_\eta, U, \rightarrow_T^{\mathbf{u}}{}_{\mathbf{app}}, X, id)$, with $(x, \mathbf{u}, x') \in \rightarrow_T^{\mathbf{u}}{}_{\mathbf{app}}$ iff $||x' - x(T, x, \mathbf{u})|| \leq \eta$. Given any desired ε there exist parameters T, η such that $T_{\eta,T}$ is finite and approximately bisimilar to $T_{F,T}$ with approximation precision ε.

Proof: Note first that because X is bounded the set $[X]_\eta$ is finite for any η, so $T_{\eta,T}$ is finite. We claim that the relation $R \subseteq X \times [X]_\eta$, defined by $(x, s) \in R$ iff $||x - s|| \leq \varepsilon$, is an ε-approximate bisimulation relation between $T_{F,T}$ and $T_{\eta,T}$, and demonstrate this by showing that R satisfies the three conditions given in Definition 3.11. Consider any $(x, s) \in R$. We have $d(x, s) \leq \varepsilon$ by definition. We next show that $\forall\ x \rightarrow_T^{\mathbf{u}} x'\ \exists\ s \rightarrow_T^{\mathbf{u}}{}_{\mathbf{app}} s'$ such that $(x', s') \in R$. Given any $x \rightarrow_T^{\mathbf{u}} x'\ \exists$ $\mathbf{u}_{stab}$ which "stabilizes" this trajectory, so that states near x are driven to states near x' by $\mathbf{u}_{stab}$; this follows from flatness of Σ_F [Martin et al. 2003]. Additionally, $x \rightarrow_T^{\mathbf{ustab}} x'$ and $||x(T, x, \mathbf{u}_{stab}) - x(T, y, \mathbf{u}_{stab})|| \leq \beta(||x - y||, T)$ for some class KL function β (see [Sontag 1998] for background on such stabilization concepts). Given x, $s\ \exists\ x''$, s' such that $x \rightarrow_T^{\mathbf{ustab}} x' \Rightarrow s \rightarrow_T^{\mathbf{ustab}} x''$ and $||x'' - s'|| \leq \eta$, so that $s \rightarrow_T^{\mathbf{ustab}}{}_{\mathbf{app}} s'$. Therefore $||x' - s'|| \leq ||x' - x''|| + ||x'' - s'|| \leq \beta(||x - s||, T) + \eta \leq \beta(\varepsilon, T) + \eta \leq \varepsilon$ (for proper choice of T, η), from which it follows that $(x', s') \in R$. The proof that $\forall\ s \rightarrow_T^{\mathbf{u}}{}_{\mathbf{app}} s'\ \exists\ x \rightarrow_T^{\mathbf{u}} x'$ such that $(x', s') \in R$ is analogous. ■

Theorem 5 shows that relaxing the requirement that the trajectories of a control system and its finite abstraction possess identical output traces enables useful approximate bisimulations to be derived. Indeed, approximation bisimulations for flat systems can be constructed that ensure the output trajectories of the system and its abstraction are as close as desired. An alternative approach to developing approximate bisimulations is to restrict attention to a subset of system behaviors and to abstract only these dynamics. For example, it may be sufficient to focus attention on system behaviors which begin and end at *equilibria*, as this is usually what is important in applications. To make this notion precise, we introduce

Definition 3.13: The *equilibrium manifold* E of a control system dx/dt = f(x, u) is the set E = {$x^* \in X$ | $\exists u^*$ for which $f(x^*, u^*) = 0$}.

In fact, in many situations of real world interest the initial and final states belong to E and, moreover, system trajectories remain close to E, and the problem naturally becomes one of abstracting the behavior of the system close to its equilibrium manifold.

We now pursue this idea for differentially flat control systems. Consider the continuous-time BNF system Σ_{BNF}, repeated here for convenience of reference:

$$dx_1/dt = x_2,$$
$$dx_2/dt = x_3,$$
$$\dots$$
$$dx_n/dt = u.$$

Recall that focusing on Σ_{BNF} is without loss of generality, as any flat system can be transformed to a set of (decoupled) chains of the form Σ_{BNF} through dynamic feedback linearization. The equilibrium manifold for Σ_{BNF} is given by $E_{BNF} = \{x \in X \mid x_2 = x_3 = \dots = x_n = 0\}$, so that E_{BNF} is identical to the flat output space Z for Σ_{BNF}. (The fact that $E_{BNF} = Z$ is interesting and currently under investigation.)

Consider the transition system $T_{BNF,eq} = (Z, \rightarrow_{BNF,eq}, Y, \pi)$ associated with the system Σ_{BNF} near E_{BNF}, where $\pi: Z \rightarrow Y$ is any finite partition of Z and $Y = \{y_1, \dots, y_p\}$ is the set of partition labels. Admissible state transitions are those which can be made while remaining "near" E_{BNF}. More specifically, for any $z \in y$, $z' \in y'$, $(z, z') \in \rightarrow_{BNF,eq}$ iff the trajectory connecting z, z' remains arbitrarily close to E_{BNF} and either y = y' or y, y' are adjacent partition elements. We are now in a position to state

Theorem 6: Let $T_{eq,app} = (Y, \rightarrow_{eq,app}, Y, id)$, with $(y, y') \in \rightarrow_{eq,app}$ iff $\exists\ z \in y, z' \in y'$ such that $z \rightarrow_{BNF,eq} z'$. $T_{eq,app}$ is a finite bisimulation for $T_{BNF,eq}$.

Proof: The proof is trivial and therefore omitted. ■

Because $T_{eq,app}$ and $T_{BNF,eq}$ are bisimilar, $T_{eq,app}$ is finite, and $T_{BNF,eq}$ is an arbitrarily accurate approximation of Σ_{BNF} on E_{BNF}, Theorem 6 gives an approximate bisimulation result for flat systems which is useful in many applications.

Another way to obtain approximate bisimulations is to focus on a particular subspace of the system state space, $X_1 \subseteq X$, which is relevant to the problem at hand. For instance, consider the system

$$dx_1/dt = f_1(x_1, x_2, u),$$
$$dx_2/dt = f_2(x_2),$$

where $x_1 \in X_1$, $x_2 \in X_2$, $X = X_1 \times X_2$, and the system (i.e., $f_1(x_1, x_2, u)$) is controllable on X_1. If the dynamics of x_2 are reasonably well-behaved (e.g., stable), it may be sufficient to concentrate on the x_1 dynamics. In this case, the preceding results imply

Corollary 3.1: Let π: $X \rightarrow X_1$ be the state space projection onto X_1 and h: $X_1 \rightarrow Y$ define a finite bisimulation partition for $dx_1/dt = f_1(x_1, x_2, u)$. Then $h \circ \pi$: $X \rightarrow Y$ is a finite bisimulation partition for the complete system.

Finally, note that given a transition system T_Σ associated with control system Σ, it is often sufficient to construct a finite abstraction T which *simulates* T_Σ, so that $T_\Sigma \angle T$ (see Definition 3.3). For instance, we will show in the case studies that simulating abstractions are very useful when Σ is differentially flat. The following corollary is relevant in this situation and is an immediate consequence of Definition 3.4:

Corollary 3.2: Let Φ: $X \rightarrow Y$ be any finite, hypercubic partition of the state space X of transition system $T = (X, \rightarrow, Y, \Phi)$. The quotient transition system $T/\sim = (Y, \rightarrow_\sim, Y, id)$ simulates T if $\rightarrow_\sim$ is defined so that $\Phi(x) \rightarrow_\sim \Phi(x')$ iff $x \rightarrow x'$.

We close this discussion of approximate bisimulations with a simple example.

Example 3.3: Continuous-time BNF control systems

It is easy to show that the planar continuous-time BNF system

$$dx_1/dt = x_2,$$
$$dx_2/dt = u$$

is differentially flat with flat output x_1, so from Theorem 5 it follows that the system admits a finite approximate bisimulation on any bounded state space $X \subseteq \Re^n$. In particular, let $X = [-1, 1] \times [-1, 1]$. It is straightforward to verify that the transition system $T_{\eta,T} = ([X]_\eta, U, \rightarrow_T^{\mathbf{u}}{}_{,\mathbf{app}}, X, id)$, with $\eta = 0.4$, $T = 2$, $[X]_\eta = \{-2\eta, -\eta, 0, \eta, 2\eta\} \times \{-2\eta, -\eta, 0, \eta, 2\eta\}$, and $\rightarrow_T^{\mathbf{u}}{}_{,\mathbf{app}}$ specified as in Figure 7, provides such an approximate bisimulation with precision $\varepsilon = 1/4$.

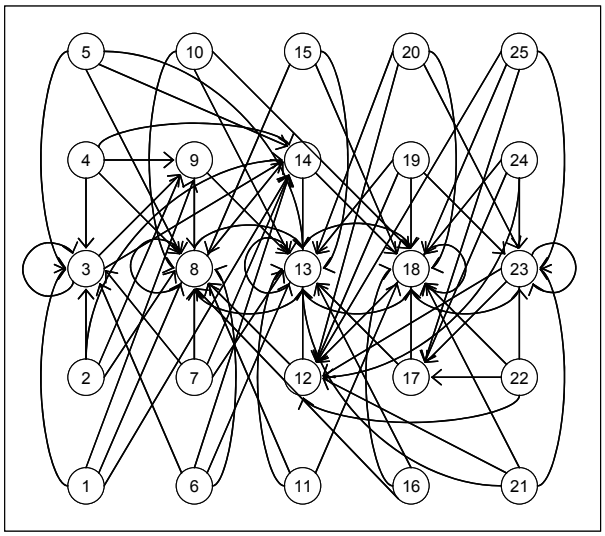


Figure 7: State transitions associated with $T_{\eta,T}$ in Example 3.3.

3.4 Computation

We now consider the problem of efficiently computing finite bisimulations for HDS. As in the preceding sections we focus on constructing bisimulations for HDS continuous systems, and in particular differentially or difference flat continuous systems, as HDS discrete systems already possess finite state representations. In fact, Lemmas 3.1 and 3.2 demonstrate that without loss of generality we can concentrate on computing finite bisimilar transition systems for controllable linear continuous- and discrete-time systems.

Consider the problem of computing a finite state simulation of continuous-time linear control system Σ_{lc}: $dx/dt = Ax + Bu$. The transition system associated with Σ_{lc} is $T_{lc} = (X, \rightarrow_{lc}, Y, h)$, with $h: X \rightarrow Y$ any finite, hypercubic partition of X and $\rightarrow_{lc}$ specified as in Definition 3.7. Corollary 3.2 shows that the quotient transition system $T_{lc}/\sim = (Y, \rightarrow_{lc\sim}, Y, id)$ simulates T_{lc} if $\rightarrow_{lc\sim}$ is defined so that $h(x) \rightarrow_{lc\sim} h(x')$ iff $x \rightarrow_{lc} x'$. Additionally, Theorems 4 and 5 indicate that this $T_{lc}/\sim$ is "close" to bisimilar to T_{lc}. Finally, observe that a primary motivation for deriving finite state abstractions for continuous state systems is complexity management, implying that many applications of interest

will involve large-scale systems. Thus there is great interest in developing an efficient algorithm for computing $\rightarrow_{lc\sim}$. We now introduce such an algorithm.

For simplicity of exposition we assume the lattice of hypercubes into which X is partitioned have sides of unit length. The algorithm decides whether a transition $y \rightarrow_{lc\sim} y'$ between two adjacent cells of the lattice y, y' is allowed, and the algorithm is repeated for all transitions of interest. We begin by summarizing a simple algorithm, based on a computational linear systems result given in [Habets and van Schuppen 2004], for deciding whether $y \rightarrow_{lc\sim} y'$ is admissible. Let k be the number of the coordinate axis orthogonal to the common face between y and y', V be the set of vertices shared by y and y', and a_k^T represent row k of A. Define $\Pi^k(x)$ to be the projection of x onto axis k and suppose y < y'. Then $y \rightarrow_{lc\sim} y'$ iff $\Pi^k(Av_i + Bu) > 0$ for some $v_i \in V$ and $u \in U$. An algorithm which "operationalizes" this observation is

Algorithm 3.1:

If y < y':

If any element of row k of B is nonzero, $y \rightarrow_{lc\sim} y'$ is true. STOP.

Repeat until $y \rightarrow_{lc\sim} y'$ is determined to be true or all vertices have been checked:

Select a vertex $v_i \in V$.

Compute the inner product $p = a_k^T v_i$.

If p > 0 then $y \rightarrow_{lc\sim} y'$ is true. STOP.

If $y \rightarrow_{lc\sim} y'$ has not been found to be true it is false.

If y > y': Algorithm is the same except that the comparison p > 0 is replaced by p < 0.

A difficulty with Algorithm 3.1 is that the number of vertices shared by two adjacent cells is 2^{n-1}, so that checking them becomes unmanageable even for moderately-sized models. For example, if n = 100 then approximately 10^{30} operations would be required to compute a single transition.

Interestingly, this algorithm can be modified so that feasibility of a transition can be tested by considering only a single well-chosen vertex, independent of the size of the model [Gardiner and Colbaugh 2006]. The new algorithm is therefore very efficient and can be applied to models with n = 10 000 or more without resorting to high performance computers. Let v_0 be the lowest vertex (in a component-wise sense) shared by y, y' and let a_k^+ (a_k^-) be the sum of positive (negative) elements of row k of A, excluding the diagonal. We are now in a position to state

Algorithm 3.2 [Gardiner and Colbaugh 2006]**:**

If y < y':

If any element of row k of B is nonzero, $y \rightarrow_{lc\sim} y'$ is true. STOP.

Compute the inner product $p = a_k^T v_0$.

If $p + a_k^+ > 0$ then $y \rightarrow_{lc\sim} y'$ is true. STOP.

Otherwise $y \rightarrow_{lc\sim} y'$ is false.

If $y > y'$: Algorithm is the same except that the comparison $p + a_k^+ > 0$ is replaced by $p + a_k^- < 0$.

Algorithm 3.2 provides an extremely efficient means of constructing the transition relation $\rightarrow_{lc\sim}$ and, therefore, the finite abstraction $T_{lc}/\sim$. A Matlab program that implements this algorithm has been developed and tested on systems with dimension n = 10 000; this code is provided in Appendix A. As an indication of the efficiency of the proposed procedure, the performance of Algorithm 3.2 is compared with Algorithm 3.1 (a quite respectable algorithm for this sort of computation) in Figure 8.

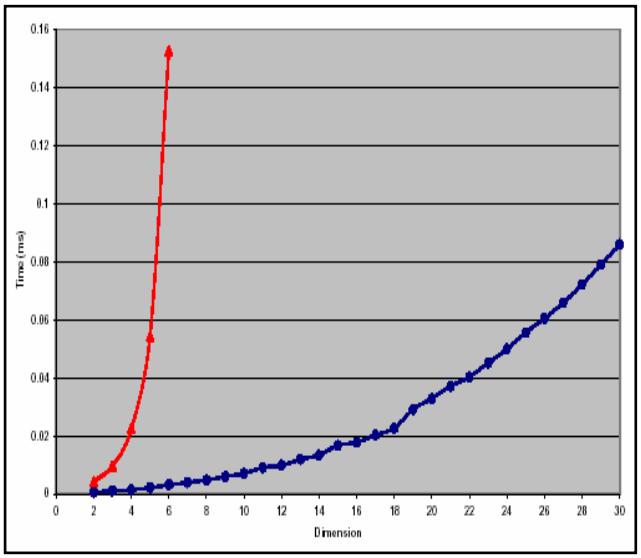


Figure 8: Results of timing study of Algorithm 3.1 (red curve) and Algorithm 3.2 (blue curve) applied to the problem of computing finite state abstractions for linear controllable systems (horizontal axis is state space dimension and vertical axis is run time in ms).

Next consider the problem of computing a finite state bisimulation for discrete-time linear control system Σ_{ld}: x+ = Ax + Bu. It is assumed without loss of generality that (A, B) is in BNF. The transition system associated with Σ_{ld} is $T_{ld} = (X, \rightarrow_{ld}, Y, h)$, where $h: X \rightarrow Y$ is any finite, hypercubic partition of X and $\rightarrow_{ld}$ is specified as in Definition 3.7. Example 3.2 shows that the quotient transition system $T_{ld}/\sim = (Y, \rightarrow_{ld\sim}, Y, id)$ is a finite bisimulation for T_{ld} if $\rightarrow_{ld\sim}$ is defined so

that $h(x) \rightarrow_{ld\sim} h(x')$ iff $x \rightarrow_{ld} x'$. As mentioned above, a primary motivation for deriving finite state abstractions for Σ_{ld} is complexity management, so that applications of interest often will involve large-scale systems and computational efficiency is a central concern.

We now introduce an efficient algorithm for constructing $\rightarrow_{ld\sim}$.

Algorithm 3.3: Let $y_c \in X$ denote the coordinate vector for the centroid of cell y.
Compute y_c for all cells of interest.
For each cell y:

Let $y_c = [c_1, c_2, \ldots, c_n]^T$.

$y \rightarrow_{ld\sim} y'$ is true for any y' which has its centroid on the line passing through $[c_2, c_3, \ldots, 0]^T$ and $[c_2, c_3, \ldots, 1]^T$.

Algorithm 3.3 provides an extremely efficient means of constructing the transition relation $\rightarrow_{ld\sim}$ and, therefore, the finite abstraction $T_{ld}/\sim$. Prototype code implementing the algorithm has been developed and tested on systems with dimension n = 10 000 [Ackley 2008].

For completeness we provide a procedure for computing finite (bi)simulations for HDS:

Algorithm 3.4 Given a complex network Σ and a class of behaviors of interest:

1. Develop an HDS model Σ_{HDS} for Σ and check that the continuous systems are (differentially or difference) flat. If not: STOP.
2. Construct a transition system representation T_{HDS} for Σ_{HDS} (according to Definition 3.8) in such a way that the continuous state space partitions h_q^k: $X_q^k \rightarrow Y_q^k$ 1.) are compatible (according to Theorem 2) and 2.) allow expression of the system behaviors of interest.
3. Transform each continuous system into the appropriate form (e.g., through feedback linearization).
4. Compute a finite state abstraction for each continuous system transition system using Algorithm 3.2 or 3.3.
5. Compose the feedback interconnection of the finite state abstractions obtained above with the HDS discrete system.

Observe that if the continuous systems are difference flat then the output of Algorithm 3.4 is a finite transition system which is *bisimilar* to T_{HDS}, and if the continuous systems are differentially flat then the output transition system *simulates* T_{HDS}. If bisimulation (approximate bisimulation) is desired in the case of differentially flat systems then the results given in Theorem 4 (Theorems 5 or 6) can be used in Step 4 of the algorithm.

4. One-Dimensional Abstraction

The previous section presented a collection of techniques for transforming the large-scale, hybrid dynamics of complex networks into much simpler, but dynamically equivalent, finite state models. We now present an alternative approach to managing the complexity of these systems, in which high-dimensional network dynamics are captured in a meaningful way using a single scalar variable. The advantages of obtaining scalar representations for high dimensional systems are, of course, well understood. For instance, one-dimensional systems are amenable to simple topological arguments, frequently yielding elegant solutions to otherwise very difficult problems. Additionally, by considering *sets* of parameter values and initial conditions, often entire families of complex networks can be analyzed using a single one-dimensional abstraction; this strategy provides an effective means of handling the parametric uncertainty present in many complex network models.

We begin our discussion of one-dimensional abstractions for high-dimensional networks by introducing the basic abstraction concept and reviewing some technical background which will be useful in our development. We then present abstraction results for both nondeterministic and stochastic systems, and provide efficient computational methods for realizing the abstractions.

4.1 Preliminaries

The basic concept of one-dimensional abstraction for high-dimensional complex networks is illustrated in Figure 9. Consider a complex network with dynamics that evolve on a continuous state space, depicted in light blue in Figure 9, according to the vector field (arrows) shown in the figure. Assume it is of interest to decide whether states in the green region can evolve to the red region. Such reachability questions are usually approached by computing the reach set associated with the green region and determining if it intersects the red region, and this analysis is quite difficult for generic complex networks. Suppose, however, that it is possible to find an "altitude" function A(x) which has a level curve, say the A(x) = 0 surface, that separates the green and red regions and on which the system vector field points in the direction of the partition containing the green region (see Figure 9). In this case we can conclude that the red region is not reachable from the green region and, moreover, this conclusion is reached *without* computing system trajectories. The trick, of course, is to find such a function A(x) or prove that no such function exists. Recent work in semidefinite programming and semialgebraic geometry [Parrilo 2000] provides a computationally tractable framework within which to search for such functions, and we will leverage this work in our proposed abstraction methodology.

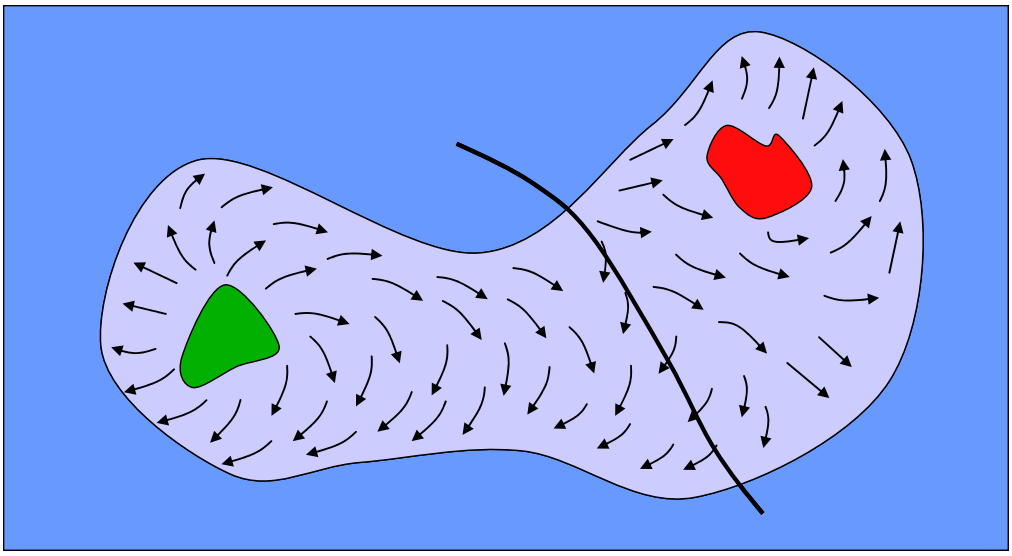


Figure 9: Cartoon of one-dimensional abstraction. The light blue region is the system state space, with green and red regions corresponding to sets of initial and "undesirable" states and the arrows indicating dynamical system flow. The cartoon illustrates that abstraction can enable reachability questions to be efficiently decided. The black curve is a level curve of a scalar altitude function which divides the state space into two portions, and because the flow points from the right portion to the left it can be concluded that the red set is not reachable from the green set *without* computing system trajectories.

Consider the problem of assessing the reachability of complex processes in the presence of uncertainty regarding the process parameters. Particularly desirable are methods that can be implemented both in stochastic settings, where probabilistic characterizations of the uncertainty are available, and nonstochastic situations, for instance where bounds on the uncertainty are known; we refer to the latter situation as nondeterministic, because the uncertainties are permitted to vary arbitrarily within their bounds. Additionally, the practical utility of any analytic approach is greatly enhanced if it is possible for reachability assessments to be carried out for *sets* of initial conditions and system parameter values. The following two subsections present a collection of abstraction-based methods for reachability assessment which possess these characteristics.

4.2 Nondeterministic systems

As indicated in Section 2, complex networks are naturally modeled as HDS, and indeed the focus of this section is abstraction-based reachability analysis of HDS. Before developing this methodology in detail, however, we introduce the basic concepts and approach by providing two

simple but very useful results for continuous dynamical systems. Consider, then, a set of differential equations

$$dx/dt = f(x, d),$$

where $x \in X$ and $d \in D$ are the system state and "disturbance" input, respectively, and $X \subseteq \Re^n$, $D \subseteq \Re^p$ are assumed bounded. Let $X_0 \subseteq X$, $X_u \subseteq X$ denote the sets of initial and "undesirable" states, respectively. Suppose we wish to show that no system trajectory starting from X_0 can evolve to the set X_u for any admissible disturbance. We adopt an analysis methodology which is analogous to the one underlying Lyapunov function-based stability analysis [Sontag 1998]: we seek a scalar function A(x) of the system state which permits this reachability question to be resolved *without* computing system trajectories.

More specifically, we have

Proposition 4.1 [Prajna et al. 2007]**:** Suppose there exists a differentiable function $A: X \rightarrow \Re$ such that

- $A(x) \leq 0 \ \forall x \in X_0$;
- $A(x) > 0 \ \forall x \in X_u$;
- $(\partial A/\partial x) f(x,d) \leq 0 \ \forall x \in X, \forall d \in D$.

Then X_u is not reachable from X_0 for any disturbance input.

Proof: The proof is straightforward and given in [Prajna et al. 2007]. ■

Remark 4.1: Observe that in establishing that X_u cannot be reached from X_0, no computation of system trajectories is required.

Remark 4.2: The set of altitude functions which satisfy the conditions given in Proposition 4.1 is convex: if $A_1(x)$ and $A_2(x)$ satisfy the conditions then $\forall \beta \in [0, 1]$, $A(x) = \beta A_1(x) + (1 - \beta) A_2(x)$ also satisfies the conditions. This convexity will be exploited to develop efficient computational methods for finding altitude functions.

Although our primary focus is showing that undesirable sets X_u cannot be reached by a system of interest, it is also possible to use the altitude function approach to demonstrate that a given system is guaranteed to reach a "desirable" set X_d:

Proposition 4.2: Suppose there exists a differentiable function $A: X \rightarrow \Re$ such that

- $A(x) < 0 \ \forall x \in X_0$,
- $A(x) \geq 0 \ \forall x \in X_e \setminus X$,

- $(\partial A/\partial x)\, f(x,d) < 0 \;\forall x \in \mathrm{cl}(X \setminus X_d),\; \forall d \in D,$

where $X_e \supseteq X$ is a global "environment" set. Then the system must reach X_d from X_0 in finite time $\forall d \in D$.

Proof: Any system trajectory starting from $x \in X_0$ must exit $\mathrm{cl}(X \setminus X_d)$ in finite time, because $A(x)$ is bounded below, and this exit cannot involve exiting X, because $A(x)$ is nonnegative outside X. Thus the system must enter X_d in finite time. ■

We now turn to one-dimensional abstractions for (nondeterministic) HDS. Consider the HDS given in Definition 2.1, repeated here for convenience:

$$\Sigma_{HDSct} \qquad \begin{aligned} q+ &= h(q,k), \\ dx/dt &= f_q(x,d), \\ k &= p(x), \end{aligned}$$

where the input is denoted by d rather than u to emphasize its role as a disturbance (rather than a control signal which can be specified by the designer).

For hybrid systems we seek a *family* of scalar abstractions $\{A_q(x)\}_{q\in Q}$, one for each discrete state q, so that the collection is a function of the HDS state (q, x). The conditions to be satisfied by this collection of altitude functions to ensure that the undesirable set X_u is not reachable from X_0 are given in

Theorem 7: Suppose there exists a family of differentiable functions $\{A_q(x)\}_{q\in Q}$ such that

- $A_q(x) \le 0 \;\forall x \in X_0,\; \forall q \in Q;$
- $A_q(x) > 0 \;\forall x \in X_u,\; \forall q \in Q;$
- $(\partial A_q/\partial x)\, f_q(x,d) \le 0 \;\forall x \in \mathrm{Inv}(q),\; \forall d \in D,\; \forall q \in Q;$
- $A_{q'}(x) \le 0 \;\forall (x, q, q')$ such that $q' = h(q,k(x))$;

where Inv(q) is the set of continuous states at which q is a feasible discrete state for the HDS. Then no trajectory of Σ_{HDSct} which starts in X_0 can reach X_u.

Proof: Let (**q**, **x**) be an admissible trajectory of Σ_{HDSct} and consider the evolution of $A_{q(t)}(x(t))$ along this trajectory. $A_q(x)$ is nonpositive initially, it is nonincreasing along any continuous flow, and it cannot increase during a discrete transition. Thus $A_{q(t)}(x(t))$ is nonpositive $\forall t$, and since $A_q(x) > 0$ on X_u we can conclude that X_u cannot be reached from X_0. ■

Remark 4.3: Results analogous to that given in Proposition 4.2 also can be derived for HDS [Colbaugh and Glass 2007].

4.3 Stochastic systems

While characterizing the uncertainty associated with complex networks in terms of bounds on the possible values of parameters and disturbances is useful and natural, in many cases a probabilistic characterization of this uncertainty is available. In such situations, application of the results presented above may lead to very conservative assessments. Indeed, if a probabilistic description of system uncertainty is available then it may be more useful and appropriate to seek a probabilistic assessment of system reachability. For example, in many applications it would be valuable to prove that the probability of a system trajectory reaching an undesirable set is lower than some acceptable "safety" threshold. We now develop such a capability.

We begin our treatment with a study of continuous state stochastic processes. Consider the set of stochastic differential equations which comprise the continuous system portion of the S-HDS given in Definition 2.3, repeated here for convenience of reference:

Σ_{sc} $$dx = f(x, p)\, dt + G(x, p)\, dw,$$

where $p \in P \subseteq \Re^p$ is the system parameter vector and all other terms are as in Definition 2.3. Denote by $X \subseteq \Re^n$, $X_0 \subseteq X$, and $X_u \subseteq X$ the sets of feasible, initial, and undesirable states, respectively, and assume that X and P are bounded.

We are interested in a probabilistic version of the reachability problem:

Definition 4.1: Given system Σ_{sc} and the sets X, X_0, X_u, the *stochastic safety problem* involves computing an upper bound $\gamma \in [0,1]$ on the probability that any trajectory of Σ_{sc} will reach X_u; thus γ satisfies

$$P\{\underline{x}(t) \in X_u \text{ for some } t\} \leq \gamma,$$

for all $x_0 \in X_0$, where $\underline{x}$ is the "stopped" process associated with x_0 and Σ_{sc} (roughly, $\underline{x}(t)$ is the trajectory of Σ_{sc} which starts at x_0 and is stopped if it encounters the boundary of X [Kushner 1967]).

We adopt an analysis methodology which is analogous to the one underlying Lyapunov function-based stability analysis [Sontag 1998]: we seek a scalar function of the system state that permits the probability upper bound γ to be deduced *without* computing system trajectories. In order to derive such results we require a few additional concepts.

Definition 4.2: The *infinitesimal generator B* for the process x(t) is given by

$$BA(x_0) = \lim_{t \to 0} (E[A(x(t)) \mid x(0) = x_0] - A(x_0)) / t,$$

where A(x) is any differentiable scalar function [Kushner 1967].

The infinitesimal generator B is the stochastic analog of the Lie derivative and characterizes the evolution of the expectation of A(x(t)):

$$E[A(\underline{x}(t_2)) \mid \underline{x}(t_1)] = A(\underline{x}(t_1)) + E[\int BA(\underline{x}(t))\, dt \mid \underline{x}(t_1)].$$

Definition 4.3: The process $A(\underline{x}(t))$ associated with a twice continuously differentiable function A(x) and stopped process $\underline{x}(t)$ is a *supermartingale* if $E[A(\underline{x}(t_2)) \mid \underline{x}(t_1)] \leq A(\underline{x}(t_1)) \; \forall \; t_2 \geq t_1$.

The following property of supermartingales is proved in [Kushner 1967] and will be instrumental in our development:

Lemma 4.1: If $A(\underline{x}(t))$ is a nonnegative supermartingale then for any x_0 and $\lambda > 0$

$$P\{ \sup A(\underline{x}(t)) \geq \lambda \mid \underline{x}(0) = x_0 \} \leq A(x_0) / \lambda.$$

We are now in a position to state our first result for the stochastic safety problem:

Theorem 8: γ is an upper bound on the probability of trajectories of Σ_{sc} reaching X_u from X_0 while remaining in X if there exists A(x) such that

- $A(x) \leq \gamma \; \forall x \in X_0$;
- $A(x) \geq 1 \; \forall x \in X_u$;
- $A(x) \geq 0 \; \forall x \in X$;
- $(\partial A/\partial x)\, f + (1/2)\, tr\, [G^T (\partial^2 A/\partial x^2)\, G] \leq 0 \; \forall x \in X, \; \forall p \in P$.

Proof: Because $(\partial A/\partial x)\, f + (1/2)\, tr\, [G^T (\partial^2 A/\partial x^2)\, G]$ is the infinitesimal generator for Σ_{sc}, the third and fourth conditions of the theorem imply that $A(\underline{x}(t))$ is a nonnegative supermartingale $\forall p \in P$. Thus from Lemma 4.1 we can conclude that $P\{\underline{x}(t) \in X_u \text{ for some } t\} \leq P\{ \sup A(\underline{x}(t)) \geq 1 \mid \underline{x}(0)=x_0 \}$ $\leq A(x_0) \leq \gamma \; \forall x \in X_0, \; \forall p \in P$. ■

Remark 4.4: Theorem 8 extends the analysis given in [Prajna et al. 2007] to allow probability upper bounds to be established in the presence of parametric uncertainty.

Results analogous to Theorem 8 can be derived for stochastic *hybrid* systems. Let $\Sigma_{S\text{-}HDSg}$ denote a general S-HDS with bounded state space $Q \times X$ and infinitesimal generator BA(q, x). We have:

Theorem 9: γ is an upper bound on the probability of trajectories of $\Sigma_{S\text{-}HDSg}$ reaching X_u from X_0 while remaining in $Q \times X$ if there exists a family of differentiable functions $\{A_q(x)\}_{q \in Q}$ such that

- $A_q(x) \leq \gamma \ \forall x \in X_0, \ \forall q \in Q$;
- $A_q(x) \geq 1 \ \forall x \in X_u, \ \forall q \in Q$;
- $A_q(x) \geq 0 \ \forall x \in X, \ \forall q \in Q$;
- $BA_q(x) \leq 0 \ \forall x \in X, \ \forall q \in Q, \ \forall p \in P$.

Proof: The proof is exactly analogous to the proof of Theorem 8. ■

The following corollary specializes Theorem 9 to the S-HDS specified in Definition 2.3 and Example 2.2 is very useful for analyzing social processes modeled as stochastic hybrid systems. More precisely, consider the S-HDS

$$\Sigma_{S\text{-}HDS} \quad \begin{array}{c} \{Q, P(x)\}, \\ dx = f_q(x)dt + G_q(x)dw, \\ k = p(x), \end{array}$$

where the transition probabilities are defined in terms of transition rates $\lambda_{qq'}(x)$, with $\lambda_{qq'}(x) \geq 0$ if $q \neq q'$ and $\Sigma_{q'} \lambda_{qq'}(x) = 0 \ \forall q$ (see Example 2.2). We have

Corollary 4.1: γ is an upper bound on the probability that trajectories of the S-HDS specified in Definition 2.3 will reach X_u from X_0 while remaining in $Q \times X$ if there exists a family $\{A_q(x)\}_{q \in Q}$ of differentiable functions such that

- $A_q(x) \leq \gamma \ \forall x \in X_0, \ \forall q \in Q$;
- $A_q(x) \geq 1 \ \forall x \in X_u, \ \forall q \in Q$;
- $A_q(x) \geq 0 \ \forall x \in X, \ \forall q \in Q$;
- $(\partial A_q/\partial x) f_q + (1/2) \text{ tr } [G_q^T (\partial^2 A_q/\partial x^2) G_q] + \Sigma_{q' \in Q} \lambda_{qq'} B_{q'} \leq 0 \ \forall x \in X, \ \forall q \in Q, \ \forall p \in P$.

Proof: The proof follows from that of Theorem 9 once it is noted that the infinitesimal generator for the above S-HDS is given by $(\partial A_q/\partial x) f_q + (1/2) \text{ tr } [G_q^T (\partial^2 A_q/\partial x^2) G_q] + \Sigma_{q' \in Q} \lambda_{qq'} B_{q'}$ [Bujorianu and Lygeros 2004]. ■

Rather than present examples of one-dimensional abstraction for nondeterministic and/or stochastic systems in this section, we refer the reader to Section 6 for a detailed discussion of abstraction-based analysis of several real world case studies.

4.4 Computation

The preceding theoretical results on one-dimensional abstraction are of significant practical interest only if it is possible to efficiently compute altitude functions A(x). Toward that end, observe that the results presented in Propositions 4.1 and 4.2, Theorems 7-9, and Corollary 4.1

specify *convex* conditions which must be satisfied by the associated altitude functions; thus the search for altitude functions can be formulated as a convex programming problem [Parrilo 2000]. Moreover, if the system of interest admits a polynomial description (i.e., the system vector fields are polynomials and system sets are semialgebraic) and if we restrict our search to polynomial altitude functions then the search can be very efficiently carried out using sum of squares (SOS) optimization [SOSTOOLS 2007].

SOS optimization is a convex relaxation framework based on SOS decomposition of the relevant polynomials and semidefinite programming. SOS relaxation involves replacing the (non-)negative and (non-)positive conditions to be satisfied by the altitude functions with SOS conditions. As a simple example of the basic idea, consider the following relaxation of the conditions given in Proposition 4.1:

$$A(x) \le 0 \ \forall x \in X_0 \quad \rightarrow \quad -A(x) - \lambda_0^T(x)\, g_0(x) \text{ is SOS}$$

$$A(x) > 0 \ \forall x \in X_u \quad \rightarrow \quad A(x) - \varepsilon - \lambda_u^T(x)\, g_u(x) \text{ is SOS}$$

$$(\partial A/\partial x)\, f(x,d) \le 0 \ \forall x \in X, \ \forall d \in D \quad \rightarrow \quad -(\partial A/\partial x)\, f - \lambda_X^T(x)\, g_X(x) - \lambda_D^T(d)\, g_D(d) \text{ is SOS}$$

where ε is a small positive constant, the entries of the vector functions λ_* are SOS, and $g_* \ge 0$ (entry-wise) when $x \in X_*$ (or $d \in D$). More complex conditions on A can be relaxed analogously. The relaxed SOS conditions are obviously sufficient and typically are not overly conservative.

The existence of polynomial functions A(x) satisfying SOS conditions can be verified – efficiently and constructively – by solving an SOS program, that is, a convex optimization problem of the form

$$\min_{c_j} \Sigma_j w_j c_j$$

$$\text{subject to } a_i(x) + \Sigma_j a_{ij}(x)\, c_j \text{ is SOS for } i = 1, 2, \ldots, p,$$

where the c_j are scalar decision variables and the w_j, $a_i(x)$, and $a_{ij}(x)$ are given. These facts lead to the following algorithm for computing A:

Algorithm 4.1 (sketch)**:**

1. Parameterize A as $A(x) = \Sigma\, c_k\, a_k(x)$, where the a_k are monomials up to a desired degree bound and the c_k are unknown coefficients.
2. Relax all criteria in the relevant theorem to SOS conditions.
3. Formulate the search for A as an SOS program, map the SOS program to an semidefinite program and solve using a standard algorithm, and return A or the information that no such A exists.

Computing families of altitude functions, $\{A_q(x)\}_{q \in Q}$, is identical. Software for solving SOS programs is available as a third-party Matlab toolbox [SOSTOOLS 2007], and example SOS programs are given in the discussion of the one-dimensional abstraction case studies (see Section 6). Importantly, the approach is tractable: for fixed polynomial degrees, the computational complexity of the associated SOS program grows polynomially in the dimensions of the continuous state and parameter spaces and the cardinality of the discrete state set.

We close this discussion of computing one-dimensional abstractions with a simple example application.

Example 4.1: Controller synthesis for nonlinear systems

Given the close conceptual relationship between altitude functions and Lyapunov functions and the utility of Lyapunov functions in control system synthesis [Sontag 1998], it is natural to consider applying the computational methodology described above to the problem of designing controllers for nonlinear systems. Interestingly, this problem is more difficult than may be evident at first glance. To see this, note first that given a Lyapunov function V(x), the search for a feedback control law d(x) which makes $-(\partial V/\partial x)f(x, d)$ SOS (and therefore is guaranteed to stabilize the system [Sontag 1998]) can be formulated as an SOS program. Similarly, given a feedback control law u(x), the search for an SOS Lyapunov function V(x) which makes $-(\partial V/\partial x)f(x, u)$ SOS also can be formulated as an SOS program. Unfortunately, the set of *pairs* (V(x), u(x)) is not convex, and thus the simultaneous search for V and u cannot be conducted using SOS programming.

Fortunately, there exist controller synthesis methods which permit V(x) to be derived first, and which then exploit this V(x) to derive the feedback law u(x); for instance, adaptive control is provides a wide range of such methods [Narendra and Annaswamy 1988]. This suggests the following approach to computationally tractable controller synthesis: 1.) use SOS programming to obtain an SOS Lyapunov function V(x) which has a Lie derivative that satisfied the appropriate SOS conditions, and 2.) derive the feedback control law u(x) using this V(x) via (say) adaptive control techniques.

5. Case Study One: Vulnerability Analysis

The first case study considers the problem of identifying and characterizing vulnerabilities of complex networks. We begin by introducing the basic problem, then propose a formal approach to vulnerability analysis for large-scale networks, and finally apply the proposed methodology to an important and challenging infrastructure network: the electric power grid.

5.1 Introduction

There is increasing recognition that complex, evolving networks, while impressively robust in most circumstances, can fail catastrophically in response to focused attacks. Indeed, this fragility appears to be a defining characteristic of these systems [Carlson and Doyle 2002], and has been observed in a range of technological networks (e.g., power grids, Internet), biological systems (e.g., gene networks, immune systems), and social processes (e.g., financial markets, social movements). Moreover, the failures associated with these vulnerabilities can have great economic and societal impact. Thus there is considerable motivation to develop a framework within which to identify these vulnerabilities and to design strategies for exploiting and mitigating them.

The challenges of vulnerability analysis are particularly daunting in the case of complex networks. Most such networks are large-scale “systems of systems”, so that the analysis methods must be extremely efficient. Additionally, as discussed earlier in the paper, these networks perform reliably almost all of the time and almost all of their features are robust. As a consequence, standard techniques for finding vulnerabilities (e.g., computer simulations, “red teaming”) can be ineffective and, in any case, are not guaranteed to identify all vulnerabilities.

In this case study we propose a vulnerability analysis methodology which is 1.) *scalable*, to enable analysis of networks of real world size and complexity; 2.) *rigorous*, so that for instance the vulnerability identification process is guaranteed to find all vulnerabilities of a given class; 3.) *robust*, for implementation in the presence of uncertainty regarding the networks and their environments; and 4.) *comprehensive*, in that a broad range of vulnerability questions can be addressed. Given a complex network and a class of failures, we are interested in four main vulnerability analysis tasks:

- Assessment: Can the system be made to experience such failure?
- Exploitation: What strategies can be used to cause such failure?
- Mitigation: How can the adverse consequences of attacks be minimized?
- Warning: What are useful precursors of impending failure?

In this section, we focus on the vulnerability assessment and exploitation tasks. It will be shown that the proposed approach to vulnerability exploitation can be employed, in slightly modified form, for designing mitigation strategies. The early warning task is the subject of the second case study, and a detailed discussion of this problem is presented there (see Section 6).

The proposed approach to vulnerability analysis for complex networks leverages existence of finite bisimulations for these networks. The basic idea is straightforward: given an HDS model for a network of interest and a class of failures of concern: 1.) construct a finite (bi)simulation for the HDS network model, 2.) conduct the vulnerability analysis on the system abstraction (e.g., using the powerful analysis methods available for finite state systems), and 3.) map the analysis results back to the original system model. Observe that the proposed approach possesses the desired characteristics. First, the analytic process is scalable because it applies sophisticated, efficient computer science analysis tools to finite state models. Second, the analysis is rigorous. Consider the situation in which the network model and its abstraction are bisimilar. The developments in Section 3 show that HDS vulnerabilities and failure/recovery dynamics are expressible as LTL formulas and that bisimulation preserves LTL, which implies the original system and its abstraction have identical vulnerabilities. Because formal computer science analysis tools (e.g., model checking) can be structured to identify all vulnerabilities of the finite state abstraction, the proposed approach is guaranteed to find all vulnerabilities of the original network as well. A similar, though slightly weaker, argument can be made if the abstraction *simulates*, rather than bisimulates, the original network. The vulnerability analysis is also robust, as aggressive abstraction provides a powerful means of managing system uncertainty (e.g., because abstraction "projects out" much of the uncertainty [Colbaugh et al. 2007]). Finally, as we will show, abstraction-based vulnerability analysis is comprehensive in that it naturally accommodates assessment, exploitation, mitigation, and early warning within a common framework.

5.2 Vulnerability analysis methodology

We now present the proposed approach to vulnerability analysis, with an emphasis on the assessment and exploitation tasks. It is supposed that the complex network of interest can be modeled as an HDS, Σ_{HDS}, and that the network's desired or "normal" behavior can be characterized with an LTL formula φ; the discussion in Sections 2 and 3 indicates that these assumptions are reasonable. We begin with the vulnerability assessment problem. Consider the following:

Definition 5.1: Given an HDS Σ_{HDS} and an LTL encoding φ of the desired network behavior, the *vulnerability assessment problem* involves determining whether Σ_{HDS} can be made to violate φ.

The proposed approach to vulnerability assessment uses *bounded model checking* (BMC), a very powerful procedure for checking whether a given finite state transition system satisfies a particular LTL specification over a finite, user-specified time horizon [Clarke et al. 1999]. Briefly, BMC checks whether a given labeled finite transition system $T = (S, L, \rightarrow, Y, h)$ satisfies an LTL specification φ on a time interval [0, k], denoted $T \models_k \varphi$, in two steps:

1. Translate $T \models_k \varphi$ to a proposition $[T, \varphi]_k$ which is satisfied by (and only by) transition system trajectories that *violate* the specification φ; such a reformulation of $T \models_k \varphi$ is always possible. For instance, $T \models_k \varphi = \Box\theta(s)$ (where $\Box$ means `always') can be translated to

$$[T, \varphi]_k = (\text{init}(s_0)) \wedge (\wedge_0^{k-1} \text{trans}(s_i, l_i, s_{i+1})) \wedge (\vee_0^{k} \neg\theta(s_i)),$$

   where the propositions $\text{init}(s_0)$ and $\text{trans}(s_i, l_i, s_{i+1})$ are true if s_0 is an admissible initial state and $(s_i, l_i, s_{i+1}) \in \rightarrow$.
2. Check if $[T, \varphi]_k$ is satisfiable using a SAT solver [Clarke et al. 1999]:
   - if $[T, \varphi]_k$ is satisfiable then T can violate φ and the "witness" generated by the SAT solver is an example violation;
   - if $[T, \varphi]_k$ is not satisfiable then T satisfies φ on all trajectories of length k or smaller.

It is stressed that modern SAT solvers are extremely powerful, so that the BMC approach to model checking is implementable with problems of real world scale.

We are now in a position to state our vulnerability assessment algorithm. Let T_{HDS} denote the transition system associated with Σ_{HDS} (as in Definition 3.8), and consider the vulnerability assessment problem given in Definition 5.1. We have:

Algorithm 5.1: Vulnerability assessment using bisimulation abstraction

1. Construct a finite bisimilar abstraction T for T_{HDS} (using, e.g., the results in Section 3).
2. Check satisfiability of $[T, \varphi]_k$ using BMC:
   - if $[T, \varphi]_k$ is not satisfiable then T is not vulnerable and thus Σ_{HDS} is not vulnerable (on time horizon k);
   - if $[T, \varphi]_k$ is satisfiable then T, and therefore Σ_{HDS}, is vulnerable, and the SAT solver witness is a vulnerability.

It is sometimes more convenient to construct a finite transition system T which simulates (rather than bisimulates) T_{HDS} (i.e., ensures $T_{HDS} \angle T$). In such cases the following vulnerability assessment algorithm is useful:

Algorithm 5.2: Vulnerability assessment using simulation abstraction

1. Construct a finite abstraction T which simulates T_{HDS} (using, e.g., the results in Section 3).
2. Check satisfiability of $[T, \varphi]_k$ using BMC:
   - if $[T, \varphi]_k$ is not satisfiable then T is not vulnerable and thus Σ_{HDS} is not vulnerable (on time horizon k);
   - if $[T, \varphi]_k$ is satisfiable then T is vulnerable and the witness generated by the SAT solver can be checked against T_{HDS} to determine vulnerability of Σ_{HDS}.

Note that checking whether T_{HDS} can experience a particular failure trajectory is much easier than *discovering* a reasonable candidate failure trajectory.

Consider next the vulnerability exploitation problem. We suppose an LTL formula φ that encodes a failure of interest is available, for example from analysis using Algorithm 5.1 or 5.2.

Definition 5.2: Given an HDS Σ_{HDS} and an LTL encoding φ of a failure behavior of interest, the *vulnerability exploitation problem* involves synthesizing an admissible input trajectory for Σ_{HDS} which ensures φ is satisfied.

The proposed approach to vulnerability exploitation uses *supervisory control for finite state systems* (SCFS), a mature body of theory and practice that includes a collection of algorithms for designing supervisors which ensure a given finite state transition system satisfies a particular LTL specification (see, e.g., [Ramadge and Wonham 1987] and also [Ackley 2008] for a review of recent advances). Note that modern SCFS algorithms are very efficient, so that this approach to controller design is implementable with finite state systems of real world scale.

We are now in a position to state our vulnerability exploitation algorithm. Let T_{HDS} denote the transition system associated with Σ_{HDS}, as before, and consider the vulnerability exploitation problem given in Definition 5.2. Assume Σ_{HDS} is vulnerable to failure behavior φ, for instance as verified via Algorithm 5.1 or 5.2. We have:

Algorithm 5.3: Vulnerability exploitation using (bi)simulation abstraction

1. Construct a finite abstraction T which simulates or bisimulates T_{HDS} (using, e.g., the results in Section 3).
2. Synthesize a vulnerability exploitation supervisor T_c for T via SCFS:

- design a finite supervisor T_c‘ which ensures (T_c‘ || T) |= $\Diamond\varphi$ (where $\Diamond$ means `eventually’ and || is the parallel product operator), plus any other required behavior;
- $T_c = T_c$‘ || T is then a finite supervisor which ensures (T_c || T_{HDS}) |= $\Diamond\varphi$.

3. Refine T_c to a (hybrid system) vulnerability exploitation strategy which can be implemented with Σ_{HDS} (this is a well studied problem in control engineering and its solution is summarized in, for instance, [Ackley 2008]).

Observe that Algorithm 5.3 produces a vulnerability exploitation supervisor which is guaranteed to cause Σ_{HDS} to experience failure φ provided T *simulates* T_{HDS}; thus T need not be a bisimulation.

Finally, we briefly consider the vulnerability mitigation problem. Both the problem definition and the solution algorithm are analogous to those for vulnerability exploitation and are stated here for completeness.

Definition 5.3: Given an HDS Σ_{HDS} and an LTL encoding φ of a failure behavior of interest, the *vulnerability mitigation problem* involves synthesizing an admissible input trajectory for Σ_{HDS} which prevents and/or recovers from φ.

Algorithm 5.4: Vulnerability mitigation using (bi)simulation abstraction

1. Construct a finite abstraction T which simulates or bisimulates T_{HDS} (using, e.g., the results in Section 3).
2. Synthesize a vulnerability mitigation supervisor T_c for T via SCFS:
   - design a finite supervisor T_c‘ that ensures (T_c‘ || T) |= ($\Diamond$goal) $\wedge$ ($\Box\neg\varphi$), thereby providing failure prevention, or (T_c‘ || T) |= ($\Diamond$goal) $\wedge$ (($\Box\neg\varphi$) $\vee$ ($\Box(\neg\varphi$ ***U*** ($\varphi \wedge \bigcirc$recover)))), which enables recovery if failure is unavoidable;
   - $T_c = T_c$‘ || T is then a finite supervisor which ensures (T_c || T_{HDS}) exhibits the above behavior.
3. Refine T_c to a (hybrid system) vulnerability mitigation strategy which can be implemented with Σ_{HDS}.

5.3 Power grid example

We now apply the proposed approach to vulnerability analysis to an example problem taken from advanced technology: the electric power grid. The focus of the analysis is vulnerability assessment, although a few brief remarks are made regarding exploitation of the identified vulnerabilities. Electric power grids play an essential role in virtually all aspects of modern life. Additionally, power grids are complex, evolving networks that possess the RYF property. For

instance, it has been established that the distribution for power grid outage size is well-approximated by a power law [Carlson and Doyle 2002], which in this class of networks suggests RYF behavior. Figure 10 shows that, while the average number of outages on the North American grid is decreasing (over the period for which data is available), the average size of *large* outages is increasing; this is a classic signature of "myopic" evolution leading to RYF behavior (see Section 2). Therefore vulnerability analysis for these large-scale technological networks is simultaneously crucially important and extremely challenging. In this example, we summarize some results obtained when applying the vulnerability analysis process described earlier in this section to an HDS model of a real (national scale) electric power system. An alternative view of grid vulnerability analysis is given in [Stamp et al. 2008].

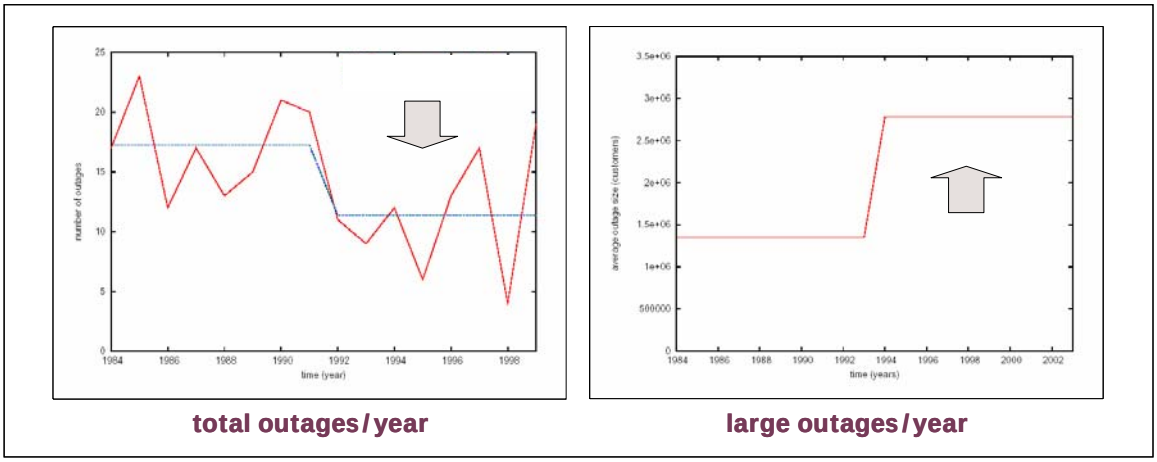


Figure 10: Electric power grids as RYF/DILO systems. The plot at left shows the evolution of number of outages per year (red) during 1984-1999; it is seen that the average number of outages/year for the first eight years is larger than for the second eight years of this period (blue). The right plot depicts average size of large outages during 1984-2003 and indicates that large outages observed during the second ten years of this period are bigger than those that occurred during the first ten years. Both plots are assembled from NERC data.

As indicated in Example 2.1, electric power grids are naturally represented as HDS, with the continuous system modeling the generator and load dynamics as well as the power flow constraints and the discrete system capturing protection logic switching (e.g., line tripping, load shedding) and other "supervisory control" behavior (e.g., scheduling phenomena). In fact, we can make

Definition 5.4: The (fairly general) *HDS power grid model* Σ_{EP} takes the form

$$\Sigma_{EP} \quad \begin{aligned} q+ &= h(q, k, v), \\ dx/dt &= f_q(x, y, u), \\ 0 &= g_q(x, y), \\ k &= p(x, y), \end{aligned}$$

where $q \in Q$ and $x \in \Re^n$ are the discrete system states (e.g., specifying which lines are in service and loads are met) and continuous system states (e.g., angles and frequencies for generators), respectively, $v \in V$ and $u \in \Re^m$ are exogenous inputs (e.g., from the SCADA system), $y \in \Re^a$ is the vector of "algebraic variables" (e.g., bus voltages and angles), h defines the discrete dynamics, the $\{f_q\}$ and $\{g_q\}$ are families of vector fields characterizing the continuous system dynamics and power flow constraints, respectively, and p is a partition of the spaces of continuous state and algebraic variables that defines the switching thresholds (e.g., for load shedding).

Example 2.1 provides a simple, concrete instantiation of this model. Additional information on power grid modeling can be found in [Ilic and Zaborszky 2000].

Power grid models possess considerable structure. For example, it has been shown that the continuous system portion of the model in Definition 5.4 is feedback linearizable [Ilic and Zaborszky 2000], which implies that the continuous system is differentially flat and consequently that Σ_{EP} admits a finite abstraction (see Section 3). It also follows from Section 3 that grid vulnerabilities are expressible as LTL formulas composed of atomic propositions which depend only on q and k. Therefore the abstraction-based vulnerability analysis methodology developed earlier in this case study is directly applicable to power grids.

To illustrate this, we conduct a vulnerability assessment for the 20bus grid shown in Figure 11. This grid provides an extremely simple representation of an actual national-scale electric power system for which (proprietary) data was obtained. The grid model, denoted Σ_{EP}, is an HDS of the form described in Definition 5.4 [Wedeward 2006]; a Matlab encoding of the model is presented in Appendix B. Because the model Σ_{EP} corresponds to a real world grid, the behaviors of model and grid can be compared. For example, the real grid experienced a large cascading voltage collapse for which data was collected. We simulated this cascading outage (see Figure 11) and found close agreement between the behavior of the actual grid and the model Σ_{EP}. Observe that this result is encouraging given the simplicity of the model and the well-known difficulties associated with reproducing such cascading dynamics with computer models.

We conducted a vulnerability assessment for the power grid Σ_{EP} shown in Figure 11 using Algorithm 5.2. It was assumed that the grid's attacker wishes to drive the voltage at bus 11 to

unacceptably low levels, so that the loads at this bus would not be served, and that the attacker has only limited grid access. For instance, we considered a scenario in which the attacker can gain assess to the generator at bus 2 via cyber means (see [Stamp et al. 2008] for a discussion of cyber attacks to power systems). Note that this sort of vulnerability is interesting because the access point – the generator at bus 2 – is geographically remote from the target of the attack – the loads at bus 11.

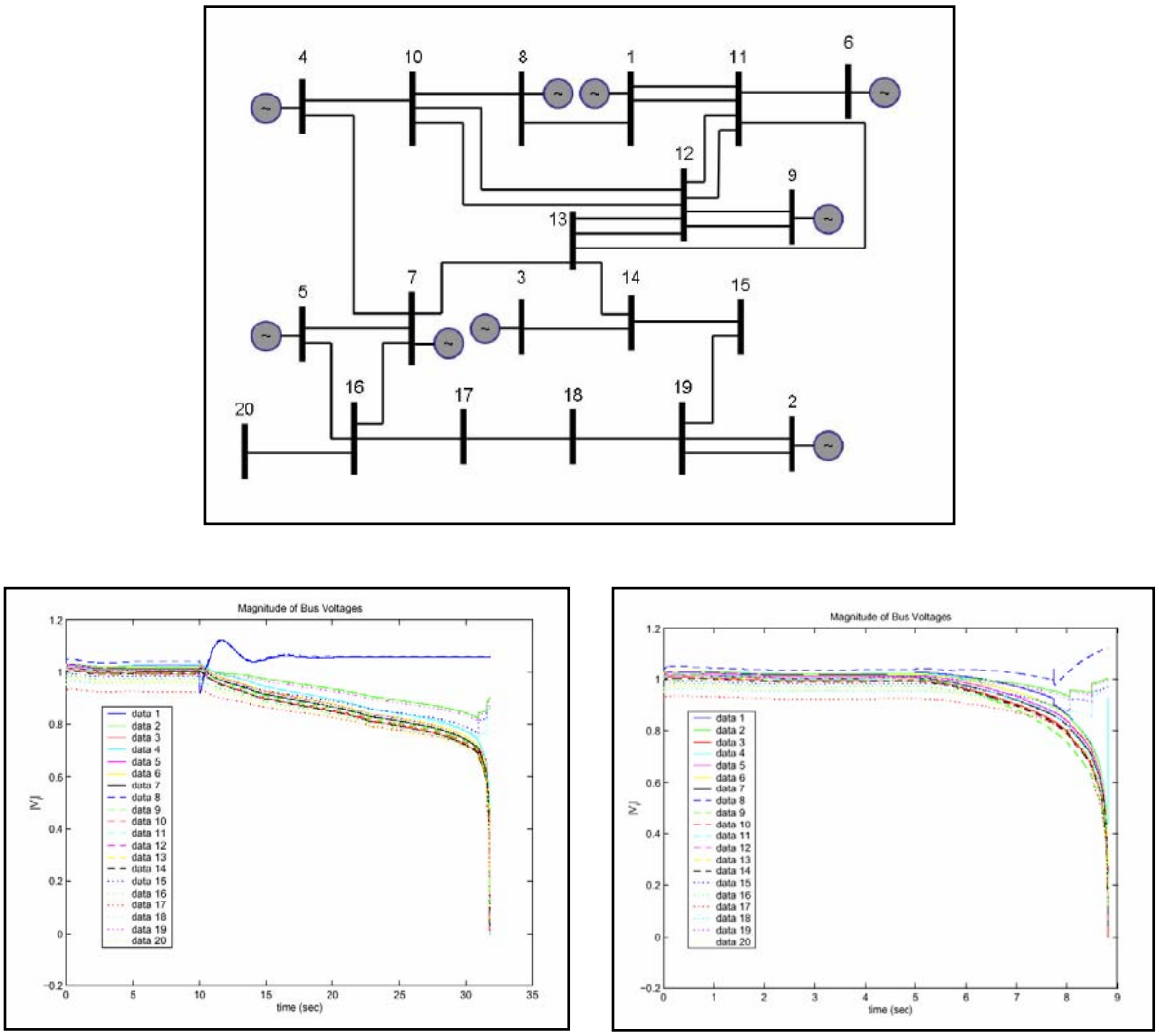


Figure 11: 20bus electric power grid used in the vulnerability analysis case study. The diagram at top is a one-line representation of the grid. The plot at bottom left shows voltage time series for all 20 buses during the simulation of a cascading voltage collapse observed with the actual grid. The plot at bottom right depicts voltage time series which result from a vulnerability exploitation synthesized using Algorithm 5.3.

The first step in the vulnerability assessment procedure specified in Algorithm 5.2 involves constructing a finite state simulating abstraction T for Σ_{EP}. This abstraction was computed using Algorithm 3.2, and a Matlab program for generating the abstraction is given in Appendix B. The second step in Algorithm 5.2 is to apply BMC to T to determine if it is possible to realize the attack objective, i.e., low voltage at bus 11, through admissible manipulation of the generator at bus 2. We employed NuSMV, an open source software tool for formal verification of finite state systems, for this analysis [NuSMV 2007]. The vulnerability assessment demonstrates that it is possible for the attacker to realize the given objective via the assumed grid access. Specific details regarding this vulnerability are considered sensitive and hence are not given in this document. Such information can be requested from the authors.

Finally, we remark that these vulnerability assessment results can serve as a starting point for synthesizing a vulnerability exploitation strategy using the analytic procedure given in Algorithm 5.3. Briefly, one such exploitation was designed and tested through simulation of Σ_{EP}. Sample simulation results are shown in Figure 11 and demonstrate that the attacker's goals can indeed be realized, in this case by initiating a cascading voltage collapse which takes down most of the power grid.

6. Case Study Two: Predictive Analysis

The second case study involves predictive analysis for complex networks, and in particular for social processes which have proven to be both practically important and challenging to predict. We begin by formulating the predictive analysis problem in mathematically precise terms and then consider predictive analysis for two classes of social processes: online markets and social movements.

6.1 Problem formulation

We formulate prediction problems as questions about the expected dynamics of a system of interest, with the system dynamics specified within an LTL framework. Recall that in LTL, propositional formulas are obtained by combining "atomic" propositions using a grammar of Boolean and temporal operators (see Section 3.1). Defining atomic propositions to correspond to problem-relevant subsets of the system's state space enables expressive characterization of the dynamics. As a consequence, predictions about the evolution of the system can be naturally posed in terms of (the satisfaction of) LTL formulas. As an illustrative example, consider the problem of predicting ultimate market share in a cultural market (e.g., music or films) in which "buzz" about a product propagates through various social networks. If, in a market containing two products with indistinguishable "intrinsic appeal", it is possible for one of the products to achieve a dominant market share, we might view the market to be unpredictable. Conversely, a predictable market would be one in which market shares of indistinguishable products evolve similarly and market shares of superior products are typically larger than those of inferior ones. Prediction, of course, then involves estimating the ultimate market share of a product of interest, perhaps based on measures of appeal. It is easy to see that these intuitive ideas can be naturally and quantitatively expressed using LTL. For instance, market share dominance by product A is associated with a region of market share state space, and the condition that A eventually achieves such dominance and simultaneously possesses an appeal that is indistinguishable from product B is easily written as an LTL formula.

Perhaps the simplest way to formulate prediction questions within an LTL framework is in terms of *reachability*. In this setting, the behavior about which predictions are to be made is used to define the system state space subsets of interest (SSI). Available measurables allow identification of indistinguishable starting sets (ISS), that is, those sets of initial conditions and system parameters which cannot be resolved with the available data. Predictability assessment then involves determining which SSI can be reached from ISS. If the system's reachability properties are incompatible with the prediction goals – if, for instance, "hit" and "flop" are both

reachable from a single ISS – then the given prediction question should be refined in some way. Possible refinements include relaxing the level of detail to be predicted (by redefining the SSI) or using additional measurables to resolve the ISS. If and when a predictable situation is obtained, the problem of forming robust, useful predictions can be addressed. This problem is also naturally studied within the reachability framework, as it involves determining the most likely evolution of the system and quantifying the uncertainty associated with this estimate.

The preceding discussion motivates the need to develop a rigorous, tractable methodology for assessing reachability of complex processes in the presence of uncertainty regarding the process parameters. Particularly desirable are methods that can be applied in both stochastic and nonstochastic settings and for *sets* of initial conditions and system parameter values. An approach to reachability assessment which possesses these characteristics is developed in Section 4 of this paper, and it is this framework that we employ in the following case study.

6.2 Social processes

The proposed approach to predictive analysis is now applied to a broad and important class of social phenomena. We begin with a brief, qualitative description of the type of social process of interest, then introduce a novel framework within which to model and analyze these systems, and finally consider two real world case studies which illustrate the considerable potential of the proposed analytic methodology.

In many social situations, individuals are influenced by observations of (or expectations about) the behavior of others, for instance seeking to obtain the benefits of coordinated actions, infer otherwise inaccessible information, or manage complexity in decision-making. Processes in which observing a certain behavior increases an agent's probability of adopting that behavior are often referred to as *positive externality processes* (PEP), and we use that term here. PEP have been widely studied in the social and behavioral sciences (e.g., economics, finance, sociology, social psychology) and, more recently, by the computer science and informatics communities. As an example of work in the latter domain, Figure 12 shows measurements of online purchasing and social networking behavior, each of which clearly exhibits the "micro", or individual-level, response associated with PEP. One hallmark of PEP is their unpredictability: phenomena from fads and fashions to financial market bubbles and crashes appear resistant to predictive analysis (although there is no shortage of *ex post* explanations for their occurrence!). These considerations are important for national security applications as well. For example, there is increasing recognition that collective dynamics are central to social movements, including revolution, political and religious radicalization, and cultural/ethnic conflict.

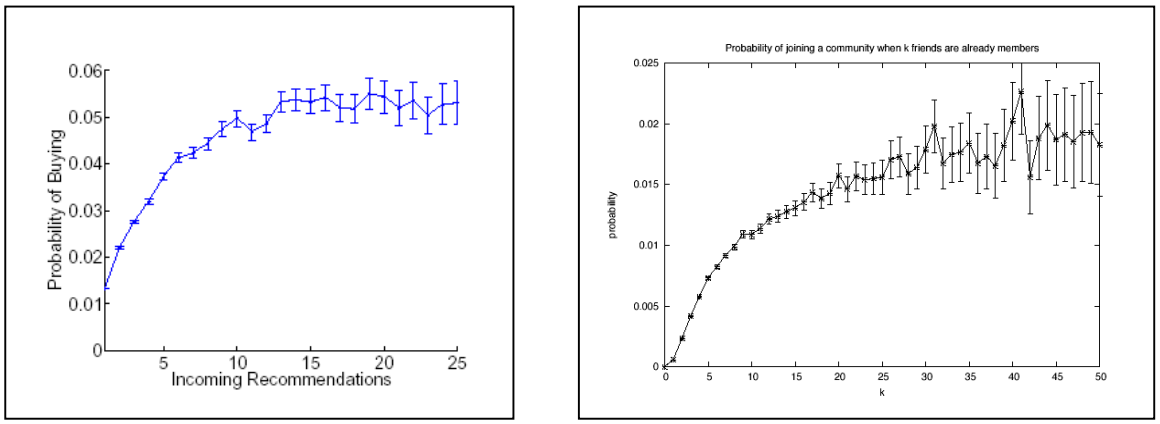


Figure 12: Positive externality processes. The plot at left demonstrates that the probability of purchasing a DVD increases as the number of purchasing recommendations increases [Leskovec et al. 2006] and the plot at right shows that the probability of joining a LiveJournal community increases as the number of friends belonging to the community increases [Kleinberg 2007].

A key step in understanding PEP dynamics, and the impact of these dynamics on predictive analysis, is the formulation of appropriate social dynamics models. Social system modeling typically proceeds along one of two lines: 1.) high level modeling which captures the behavior of aggregates of individuals, for instance using ordinary differential equations or statistical regressions to characterize the fractions of individuals exhibiting certain behaviors over time (see, e.g., [Hedstrom et al. 2000] and the references therein), and 2.) detailed modeling of the behavior of individuals and the ways individuals interact, often using agent-based methods [e.g., Epstein 2006]. Interestingly, there is often little justification for, or evaluation of, the choices made regarding the system features which are included and omitted. As shown in the preceding sections of this paper, however, complex systems often evolve so that some system features require only simple, abstract models while others demand highly accurate representations, and it is often challenging to allocate modeling detail appropriately. For example, recent work has clearly demonstrated the importance of capturing social network effects when modeling social processes [Newman 2003]. Unfortunately, detailed information concerning the relevant social networks is not typically available. Additionally, even when these data can be estimated, it is often the case that naïve approaches to network modeling lead to unnecessarily complicated models and subsequent analytic difficulties.

These facts motivate the development of a class of *multi-scale* models for social processes which respect the function and evolution of the underlying systems. The proposed multi-scale

representation reflects the essential structures present in positive externality social systems through the use of three modeling scales:

- a *micro-scale*, for modeling the behavior of individuals;
- a *meso-scale*, which represents the collective dynamics within *social contexts* via "fully mixed" models for the interaction dynamics (see, e.g., [Hedstrom et al. 2000], [Watts et al. 2002] for social network-oriented introductions to the concept of social contexts);
- a *macro-scale*, which characterizes the interaction between the social contexts.

A schematic of the basic framework is given in Figure 13.

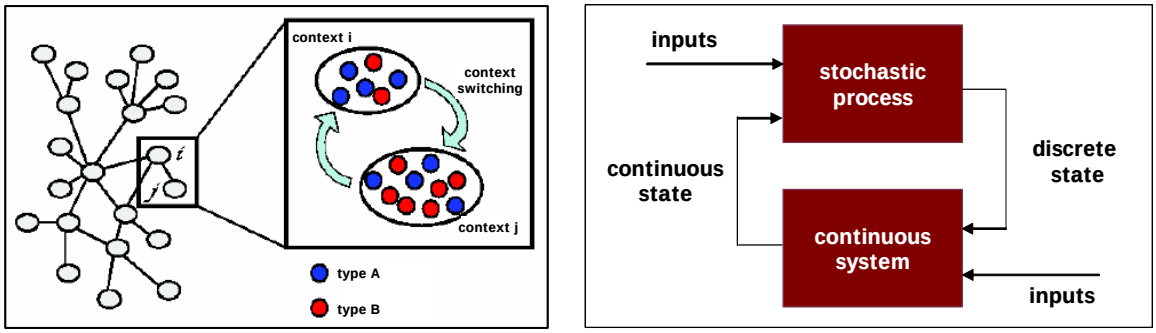


Figure 13: Multi-scale model for social processes. The cartoon at left illustrates the basic model structure, in which individuals (blue and red nodes) interact *within* social contexts (ellipses encircling nodes) via fully mixed dynamics and *between* contexts according to the network topology characterizing context interdependencies. The block diagram at right depicts an S-HDS encoding of the model, in which the S-HDS continuous system captures intra-context dynamics and the discrete system models the inter-context behaviors.

This multi-scale approach alleviates the need for detailed social network data, because interactions within social contexts are modeled as fully mixed. Moreover, by distinguishing between the way individuals interact within and across social contexts, we simultaneously capture the important social network structure and obtain analytically tractable mathematical formulations. Note that the characterization of intra-context and inter-context dynamics implicit in the proposed multi-scale framework is based on established social science understanding (e.g., [Hedstrom et al. 2000], [Watts, et al. 2002]).

We develop and analyze multi-scale social dynamics models using the S-HDS formalism described in Section 2 [Colbaugh and Glass 2007]. Briefly, these S-HDS are feedback interconnections of continuous dynamics, such as the dynamics of individuals exchanging ideas

within a social context, and discrete dynamics, capturing for instance the switching behavior encountered when an individual from one context moves to another and introduces an idea which is novel in the latter context (see Figure 13). An advantage of representing multi-scale social dynamics using an S-HDS framework is that the resulting models are amenable to one-dimensional abstraction-based analysis. In fact, we show in the following two case studies that one-dimensional abstraction enables the development of a mathematically rigorous, computationally tractable methodology for predictive analysis of PEP.

Online markets

Consider an online market in which individuals visit a web site, browse an assortment of available items, and choose one or more items to download. A surprising and interesting characteristic of these markets – and many other markets as well – is that they are often both unequal and unpredictable: a few items capture a large share of the market, but which items achieve popularity appears to be hard to anticipate. For instance, the study reported in [Salganik et al. 2006] created an artificial music market and demonstrated this phenomenon experimentally. Moreover, this work showed that increasing the opportunity for social influence increased both the inequality of the ultimate market shares and the unpredictability of which songs attain market dominance. Our study of CNET, the online software library, yielded similar results: 1.) daily download market share of a software item is positively correlated with cumulative item downloads (the market exhibits positive externalities) and not correlated with measures of item quality (e.g., expert reviews, user reviews, technical data), and 2.) the average quality of the most popular software is indistinguishable from the average quality of all software available on the site [Colbaugh and Glass 2007]. The positive externalities present in these markets makes predictive analysis using standard methods a challenging undertaking.

We now apply the predictive analysis framework summarized above to the problem of forecasting ultimate market share in online markets. Consider a market visited by a sequence of consumers, with each visitor choosing between two items {A, B}; generalizing this simple binary choice setting to any finite number of choices is straightforward. We model this situation by supposing that each agent i chooses item A with probability

Σ_{online} $$P_i(A) = \beta\pi + (1-\beta)\, f$$

where $f \in [0,1]$ is item A's current market share, $(1-\beta)$ quantifies the intensity of social influence (with $\beta \in [0,1]$), and π is the probability of an agent choosing A in the "no social influence" case (i.e., when $\beta=1$); agent i chooses item B with probability $1 - P_i(A)$. In this model, π can be

interpreted as a measure of the "appeal" of item A (relative to B), f is the social signal, and β quantifies the relative importance of appeal and social influence in the decision-making process.

The model Σ_{online} is extremely simple, perhaps the simplest possible representation which captures the effects of both social influence and appeal in an online market. Nevertheless, this model is able to reproduce all of the key behavior observed in the music market study described in [Salganik et al. 2006], in our investigation of CNET market dynamics, and in other online markets (e.g., for books and DVDs) [Colbaugh and Glass 2007]. In particular, as β decreases (social influence increases) both the inequality and unpredictability of market shares increase. Thus, despite its simplicity, Σ_{online} provides a useful starting point for studying predictability and prediction of online markets. In particular, Σ_{online} can be written in the form of the continuous system portion of the S-HDS model given in Definition 2.3, so that Theorem 8 is directly applicable; see Appendix C for the precise form of the resulting model.

As a simple illustration of the sorts of predictive analysis questions which can be addressed using the proposed framework, consider the problem of identifying measurables which can be used to predict ultimate market share. The standard approach is to assume that item appeal is such a measurable, estimate appeal in some way, and use this estimate to predict ultimate market share. To examine the utility of this approach, we assess the predictability of market share for identical items (π=1/2) with identical initial market share (f(0)=1/2). If it is reasonably likely that the market will evolve so one or the other item dominates (f becomes large or small), then the market dynamics is not (very) dependant on item appeal and therefore is unpredictable using standard methods. In this case we should seek a different prediction method, one based on other measurables. Alternatively, if market dominance by either item is unlikely then the market dynamics depends on item appeal in a more predictable way and standard methods may lead to useful prediction.

Predictability assessment can be conducted using Theorem 8 by setting X_u to correspond to large/small f and computing an upper bound on the probability of reaching X_u (see Figure 14). More specifically, we compute an upper bound on the probability that Σ_{online} with π=1/2 will evolve from X_0^1 to X_u in Figure 14, as this corresponds to the probability that indistinguishable items with equal initial market shares will evolve so that one item dominates. We consider two situations: the low social influence (SI) case, obtained by setting β large in Σ_{online}, and the high SI case, corresponding to small β. Theorem 8 is applied to Σ_{online} for sets of initial conditions and model parameter values using SOSTOOLS (see Appendix C for the Matlab program used in this analysis). In the high SI case, the analysis generates fairly high probability bounds for a

broad range of noise models, with $\gamma \in [0.5, 0.7]$ being typical, and also shows that large f (A dominance) and small f (B dominance) are equally likely. This result is consistent with empirical findings (e.g., [Salganik et al. 2006]) and suggests that standard approaches to market share predictions are not likely to produce accurate forecasts. When SI is low, however, very small upper bounds are found for the probability of reaching X_u (on the order $\gamma \sim 10^{-3}$); this result also matches empirical findings and indicates that in this case product appeal matters and that predicting market share based on appeal can be sensible.

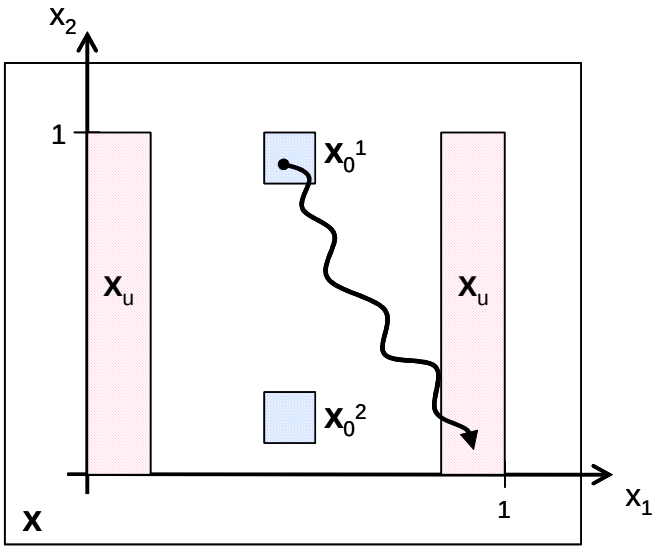


Figure 14: Predictability analysis setup for online market example. Note that in this diagram the x_1, x_2 coordinates correspond to f (market share for item A) and 1/(t+1), respectively.

An advantage of the proposed predictive analysis framework is the convenience with which alternative measurables can be examined. For example, it might be supposed that very early market share time series data would be useful for prediction in the high SI case. The intuition behind this idea is that the "herding" behavior that can arise from SI, and which makes market prediction hard using standard methods, may lead to a lock-in effect, in which very early market share leaders become difficult to displace. To test this hypothesis, we assess the predictability of market share for identical items which have identical early market share time series. Thus we compute an upper bound on the probability that Σ_{online} with π=1/2 will evolve from X_0^2 to X_u in Figure 14. We again apply Theorem 8 to Σ_{online} for sets of initial conditions and model parameter values using SOSTOOLS (see Appendix C for the Matlab program used in this analysis). In this case, the analysis generates very small upper bounds ($\gamma \sim 10^{-3}$ is typical). Thus items with similar

early market share time series are very unlikely to evolve so that one dominates. This analysis suggests that using early time series data to refine the ISS leads to a more tractable prediction problem, in which indistinguishable market configurations evolve to indistinguishable outcomes even in the presence of high SI.

We mention in closing that we have developed an algorithm for predicting online market share which leverages these insights regarding the role of SI in market evolution and the importance of early market dynamics in high SI situations. The algorithm first estimates the intensity of SI in the market using a combination of adaptive methods [Narendra and Annaswamy 1988] and HDS theory [Colbaugh and Glass 2007]. This estimate then forms the basis for choosing either an appeal-based or herding-based prediction strategy to predict ultimate market share. Application of this approach to online markets as well as other markets (e.g., movie box office revenue) yields predictions of quite respectable accuracy [Colbaugh and Glass 2007].

Social movements

Social movements are large, informal groupings of individuals and/or organizations focused on a particular political, social, economic, or religious issue. Example issues include racial and gender equality, temperance, labor rights, political ideology, economic philosophy, religious fundamentalism, and environmental concerns. Consider the problem of distinguishing successful social movements, that is, movements which attract significant following, from unsuccessful ones early in their lifecycle. Here we apply our proposed approach to predictive analysis for social processes to this analysis task. As shown above, a crucial step in the approach is assessment of the predictability of the process of interest. Among other things, predictability assessment enables identification of those measurables which are most useful for prediction. We expect this function to be important here, as there are myriad measurables associated with radicalization which *may* have predictive power, and identifying which (if any) actually do is both challenging and critical for successful warning analysis.

The study consists of three parts: 1.) a *theoretical investigation* of social movement warning analysis using models taken from the social movement theory (SMT) literature, 2.) an *empirical study* of social movement warning analysis involving a “data-rich” social movement (i.e., the emergence and diffusion through Sweden of their Social Democratic Party), and 3.) a combined *empirical/theoretical investigation* of Islamic mobilization warning analysis involving both successful and unsuccessful mobilization events and using online social activity as the main data source. We begin with an investigation of *general* social movements. This broader setting is reasonably well-characterized both theoretically and empirically and therefore provides the

opportunity to identify candidate early indicators in a principled way; candidate indicators (if any) can then be tested for relevance to particular applications, such as Islamic mobilization and radicalization. Consider the problem of identifying measurables which permit successful social movements to be distinguished from unsuccessful ones early in their lifecycle. This problem is naturally formulated within the proposed predictability assessment framework. Movement success is quantified by defining a subset X_u of the social system state space that corresponds to a level of movement membership consistent with movement goals, and we seek measurables which allow (early) identification of those movements that are likely to evolve to X_u.

In the theoretical study of social movements, we first collect a family of models from the SMT literature and formulate these within our multi-scale framework. This approach yields models that appropriately represent social network effects while remaining broadly consistent with current SMT thinking. More specifically, we derive an S-HDS representation for social movements which is of the form given in Definition 2.3 , and find that this formulation enables *simultaneous* analysis of an entire collection of relevant SMT models. This result is achieved by employing the model abstraction methodology presented in [Colbaugh and Glass 2007] to derive a "base model" whose parameterization enables any model in the family of interest to be recovered through suitable specification of parameter values. We then work with the complete family of models all at once by conducting the analysis for the entire set of feasible parameter values.

The continuous system portion of the S-HDS model is a collection of stochastic differential equations (SDE), each of which captures a particular instantiation of the intra-context social dynamics. This collection is indexed by a discrete "mode" $q \in Q$ that specifies which (vector) SDE is currently active. The mode q concisely quantifies the history of inter-context interactions, and specification of the appropriate active SDE is based on this history. Thus, for example, a particular sequence of inter-context communications concerning new information leads to a certain distribution of informed individuals across social contexts, and this distribution in turn impacts the subsequent intra-context dynamics. Mode q evolves according to a Markov chain with state set Q and continuous state dependent transition probabilities; this is the discrete system component of the S-HDS (see Definition 2.3 and Example 2.2).

Predictability assessment is performed for the collection of S-HDS social movement models using the result given in Corollary 4.1. That is, we compute provably-correct upper bounds for the probability that any model in the collection will reach X_u from X_0. Because this computation does not require forward simulation and can be conducted for *sets* of initial states and parameter values, we can efficiently explore the way various measurables affect these

probability bounds. Those measurables for which the probability of reaching X_u exhibits sensitive dependence are designated to be potentially useful indicators of movement success.

Briefly, this study produced two main results. First, the degree to which movement-related activity shows *early diffusion* across multiple social contexts is a powerful distinguisher of successful and unsuccessful social movements. Indeed, this measurable has considerably more predictive power than the *volume* of such activity and also more power than various system intrinsics. Second, significant social movements can occur only if both 1.) the intra-context "infectivity" of the movement exceeds a certain threshold and 2.) the inter-context interactions associated with the movement occur with a frequency that is larger than another threshold. Note that this is reminiscent of, and significantly extends, well-known results for epidemic thresholds in disease propagation models. Appendices D and E of this paper present sample Matlab programs which implement predictability analysis for S-HDS social system models for epidemics (Appendix D) and social movements (Appendix E).

The empirical investigation of early warning analysis for social movements focuses on the emergence and growth of the Swedish Social Democratic Party (SDP). The case of the SDP is particularly relevant for our purposes, as the early activities of political "agitators" associated with the SDP led to the establishment of a well-defined, and well-documented, network linking previously disparate geographically- and demographically-based social contexts in Sweden [Hedstrom et al. 2000]. We explore the role played by this inter-context network by analyzing archived data and published accounts describing the dynamics of the SDP. Our investigation uses standard time series analysis techniques similar to those employed in [Hedstrom et al. 2000] and reveals that an important predictor of SDP spatio-temporal dynamics is *early diffusion* of SDP-related activity across social contexts. In fact, this measurable has more predictive power than demographic and political features of the population.

We now briefly summarize a few details of the study. The visualization at the top of Figure 15 depicts the temporal evolution of the concentration of SDP members in approximately 360 Swedish jurisdictional districts over the period 1885-1947. In this rendering of the data, the horizontal coordinate axis is district index, the vertical coordinate is time, and the colors indicate variation from minimum member concentration (dark blue) to maximum concentration (red). Examination of the figure reveals the expected "contagion" effect in membership evolution, in which districts that are close geographically experience similar membership trajectories (geographically proximate districts have index values which are close). This visualization also shows that in the early years of the party, some geographically disparate districts initiated local party chapters almost simultaneously and then experienced similar growth patterns.

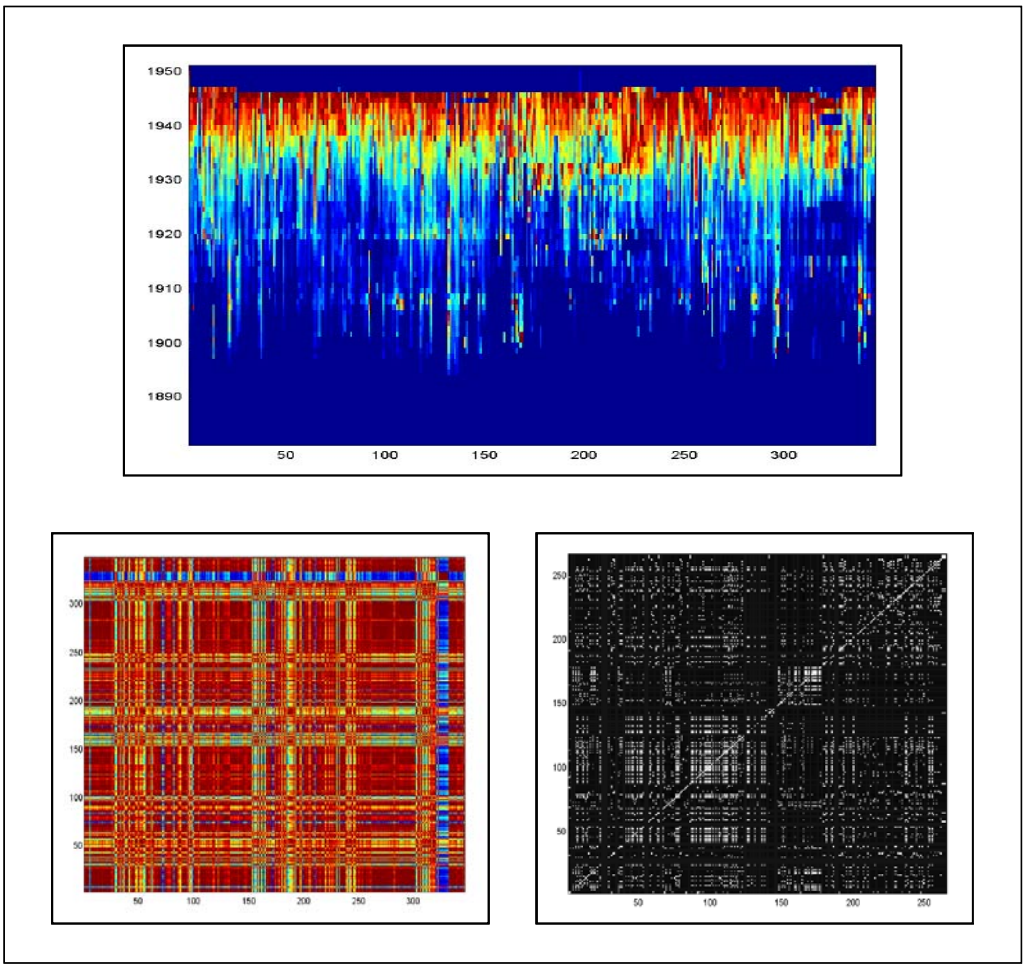


Figure 15: Sample results for the Swedish SDP case study. The figure at top is a visualization of the temporal evolution of the concentration of SDP members in approximately 360 Swedish jurisdictional districts over the period 1885-1945. The bottom figures show time series cross correlations for membership concentration (left) and party office founding events (right), both across districts.

The cross correlation results shown at the bottom of Figure 15 confirm this observation. The time series of district membership concentration (left) and party office founding events in districts (right) show both local geographic correlations, corresponding to contagion effects, and also non-local effects. Further analysis (not shown) indicates that these correlations are significantly larger than those observed in data with randomized district indices. More interestingly, we find that the non-local correlations can be explained by the inter-context network established by early party activists: those districts which exhibit similar early party initiation and growth are also those with direct (activist-induced) inter-context links. Thus inter-context dynamics played an important role in the emergence and growth of the SDP.

The theoretical and empirical results summarized above suggest social network dynamics are critical to social movement success. Moreover, the results show that the features of these dynamics which may be useful early indicators of movement success are practically measurable in many applications. For instance, diffusion across social contexts often can be inferred from analysis of public opinion and demographic data, as this measure requires only incomplete information regarding the relevant social networks. Alternatively, we show below that *online* (i.e., Web-based) social activity can sometimes serve as a proxy for these social dynamics.

We now investigate whether diffusion across social contexts is a useful early indicator for successful Islamic mobilization and protest events. The study focuses on Muslim reaction to six recent incidents, each of which appeared at their outset to have the potential to trigger significant protest activities:

- publication of photographs and accounts of prisoner abuse at Abu Ghraib in Spring 2004;
- publication of cartoons depicting the prophet Mohammad in the Danish newspaper *Jyllands-Posten* in September 2005;
- distribution of the DVD *I was blind but now I can see* in Egypt in October 2005;
- the lecture given by Pope Benedict XVI in September 2006 in which he quoted controversial material concerning Islam;
- Salman Rushdie being knighted in June 2007;
- republication of the "Danish cartoons" in various newspapers in February 2008.

Recall that the first Danish cartoons event ultimately led to substantial Muslim mobilization, including massive protests and considerable violence, and that the Egypt DVD event also resulted in significant Muslim mobilization and violence. In contrast, Muslim outrage triggered by Abu Ghraib, the pope lecture, the Rushdie knighting, and the second Danish cartoons event all subsided quickly with essentially no violence. Therefore, taken together, these six events provide a useful setting for testing whether the extent of early diffusion across social contexts can be used to distinguish nascent Islamic mobilization events which become large and self-sustaining (and potentially violent) from those that quickly dissipate.

A central element in the proposed approach to early warning analysis is the measurement, and appropriate processing, of social dynamics associated with the process of interest. Indeed, the preceding results suggest that in many cases reliable warning analysis *requires* such data. In the present case study we use online social activity as a proxy for "real world" diffusion of mobilization-relevant information. More specifically, we use blog-based communications and discussions as our primary data set. The "blogosphere" is modeled as a graph composed of two

types of vertices, the blogs themselves and the concepts which appear in them. Two blogs are linked if a post in one hyperlinks to a post in the other, and a blog is linked to a concept if the blog contains (a significant occurrence of) that concept. See Figure 16 for a schematic representation of this sort of blog graph. Among other things, this blog graph model enables the identification of blog communities – that is, sets of blogs with intra-group edge densities that are significantly higher than expected [Newman 2003]. In what follows, these blog communities serve as one proxy for social contexts (see Figure 16).

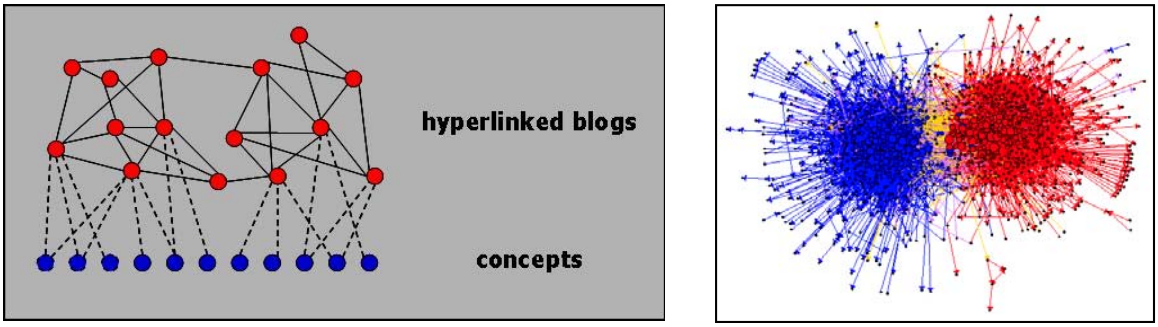


Figure 16: Blog graph model. Schematic on left depicts the basic graph structure, in which blogs (red vertices) can be connected to each other via hyperlinks (solid edges) and also connected to concepts (blue vertices) they contain. Blog graph on right corresponds to political blogs; note that in this graph liberal (blue) and conservative (red) blogs form two distinct communities [Adamic and Glance 2005].

We propose the following procedure for warning analysis using blog data: given a potential "triggering" event of interest

1. Use key words and concepts associated with the triggering event to collect relevant blog posts and build the associated blog graph.
2. Identify the blog social contexts (e.g., graph community-based, language-based).
3. Construct the post volume time series for each social context. Compute the post/context entropy (PCE) time series associated with the post volume time series.
4. Construct a synthetic ensemble of PCE time series from (actual) post volume dynamics using a general S-HDS social diffusion model.
5. Perform motif detection: compare the actual PCE time series to the synthetic ensemble series to determine if the early diffusion of activity across contexts is "excessive". Flag events with excessive early diffusion for further (e.g., manual) analysis.

We now provide additional details concerning this procedure and its implementation. Step 1 is by now a standard operation in web mining applications, and various "off the shelf" tools exist which can perform this task. For instance, in this study we employ Google Blogs together with tools developed by the Artificial Intelligence Laboratory at the University of Arizona [AI Lab 2007]. In Step 2 we use two definitions for blog social context: graph community-based, in which the contexts are graph communities found through standard community extraction applied to the blog graph, and language-based, in which contexts are defined based on the language of the posts (Google Blogs archives blog posts in 43 languages).

In Step 3, post volume for a given context i and sampling interval t is obtained by counting the number of (relevant) posts made in the blogs comprising context i during interval t, and the post volume time series are simply the concatenation of these counts. PCE for a given sampling interval t is defined as follows:

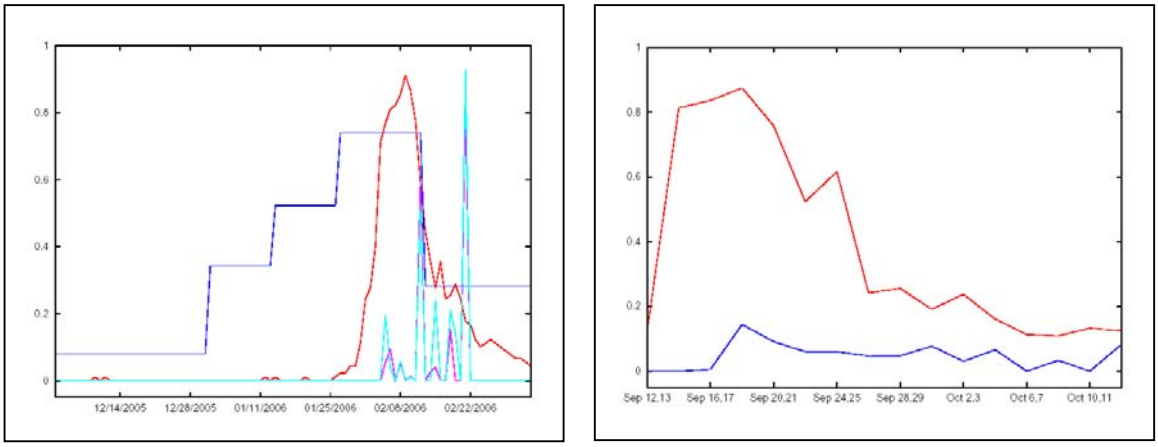
$$PCE(t) = -\Sigma_i \, f_i(t) \log(f_i(t)),$$

where $f_i(t)$ is the fraction of total relevant posts during interval t which occur in context i; the associated time series is again simply the sequence of these values.

Given the post volume time series obtained in Step 3, Step 4 involves the construction of an ensemble of PCE time series which would be expected under "normal circumstances", that is, if Muslim reaction to the triggering event diffused from a small "seed set" of initiators according to SMT social dynamics. For this study, we use the multi-scale S-HDS modeling framework to generate the PCE time series ensembles. Finally, motif detection in Step 5 is carried out by searching for periods, if any, during which the actual PCE time series is excessive relative to the synthetic PCE ensemble (e.g., exceeds the mean of the ensemble by two standard deviations).

We now apply the proposed approach to early warning analysis to the Islamic mobilization case study. If early diffusion of discussions across blog communities is an indicator that the associated Islamic mobilization event will be large, we would expect to observe such diffusion with the mobilization associated with the first Danish cartoons and Egypt DVD events and not with the other four events. Additionally, we would expect this early diffusion to be "excessive", relative to the synthetic ensemble, for the first two events and not for the latter four. As can be seen in Figure 17, this is what we find. In the case of the first Danish cartoons event, the entropy of diffusion of relevant discussions across blog communities (blue curve) experiences a dramatic increase a few weeks before the corresponding increase in the volume of blog discussions (red curve); this latter increase, in turn, occurs before any violence (see Figure 17). In contrast, in the case of the pope event, the entropy of diffusion of discussions across blog

communities is small relative to the cartoons event, and any increase in this measure lags discussion volume. Similar curves are obtained for the other four events. More importantly, the proposed motif detection process also yields the expected result: motifs are found only for the Danish cartoons and Egypt DVD events, and these motifs precede significant blog volume and real world violence. Note that qualitatively similar results are obtained for both the graph community-based and language-based definitions of social context. This case study suggests that early diffusion of mobilization-related activity (here blog discussions) across disparate social contexts (blog communities) may be a useful early indicator of successful mobilization events.



Time series motif analysis

Event	Motif
Danish cartoons 1:	**1/1—1/26/2006.**
Egypt DVD release:	**10/2–10/9/2005.**
Abu Ghraib story:	**none.**
Pope lecture:	**none.**
Rushdie knighting:	**none.**
Danish cartoons 2:	**none**

Figure 17: Sample results for Islamic mobilization case study. The time series plots at the top correspond to the first Danish cartoons event (left) and the pope event (right). In each plot, the red curve is blog volume and the blue curve is blog entropy; the Danish cartoon plot also shows two measures of violence. Note that while the data are scaled to allow multiple data sets to be graphed on each plot, the scale for entropy is consistent across plots to enable cross-event comparison. The table at the bottom summarizes the results of the motif analysis study. Note that only the first Danish cartoons event and Egypt DVD event exhibit time series motifs.

7. Concluding Remarks

This paper presents a new framework for analyzing complex networks based on aggressive abstraction, that is, a dramatically simplifying and property preserving abstraction of the network of interest. In the proposed analytic procedure, the given network is first abstracted to a much simpler (but equivalent) representation, the required analysis is performed using the abstraction, and analytic conclusions are then mapped back to the original network and interpreted there. The paper makes three main contributions:

1. We identify broad and important classes of complex networks which are amenable to aggressive abstraction. These networks are typically the result of an evolving process which favors systems that are robust to (familiar) environmental perturbations and internal flaws. We also provide efficient computational algorithms for testing whether a given network can be abstracted using any of our methods.
2. We introduce and develop two forms of aggressive abstraction: i.) finite state abstraction, in which dynamical networks with uncountable state spaces are modeled using finite state dynamical systems, and ii.) one-dimensional abstraction, whereby high dimensional network dynamics are meaningfully captured using scalar functions. In each case, the property preserving nature of the abstraction process is rigorously established, efficient algorithms for computing the abstraction are presented, and the main concepts are illustrated through simple examples.
3. We demonstrate the considerable potential of the proposed approach to complex networks analysis through real world case studies. In our first case study, we develop a powerful new approach for vulnerability analysis of large, complex networks by leveraging the scalability and property preserving character of the finite state abstraction process; the efficacy of this vulnerability assessment framework is illustrated through analysis of fairly large-scale electric power grids. The second case study focuses on predictive analysis for complex network dynamics, with a focus on social processes on networks. This study develops a formal approach to predictability and prediction analysis that is enabled by one-dimensional abstraction techniques, and the utility of the methodology is illustrated through analysis of social processes for which standard approaches to prediction have been ineffective.

Future work will involve extending the theoretical and computational foundations as well as applying the proposed analytic approach in a broader range of domains. Theoretical extensions of interest include developing methods for finite state abstraction of stochastic systems and for

graph topology-based abstraction of complex networks; as an example of the latter, we expect to study the problem of “coarse-graining” network topologies by representing certain subgraphs of the original network with vertices in the abstraction. Work on computational methods will include exploring the utility of massively multi-threaded computer architectures for abstraction procedures. Among the application areas anticipated to be of interest are:

- vulnerability analysis of the large-scale, socio-technical “systems of systems” associated with national infrastructures, for instance for problems in the critical infrastructure protection and information operation domains;
- analysis of those biological networks (e.g., gene regulation, metabolic) which are central to human disease;
- predictive analysis for social processes of relevance to national security (e.g., terrorism, proliferation) and other (e.g., financial and other markets) domains.

8. References

[Ackley 2008] Ackley, D., *Finite State Abstraction for Continuous Control Systems: Analysis and Synthesis*, MS Thesis, Department of Computer Science, New Mexico Institute of Mining and Technology, 2008.

[Adamic and Glance 2005] Adamic, L. and N. Glance, "The political blogosphere and the 2004 U.S. election: Divided they blog", *Proc. Link-KDD-2005*, Chicago, IL, August 2005.

[AI Lab 2007] http://ai.bpa.arizona.edu/start.html, 2007.

[Alur et al. 2000] Alur, R., T. Henzinger, G. Lafferriere, and G. Pappas, "Discrete abstractions of hybrid systems", *Proc. IEEE*, Vol. 88, pp. 971-984, 2000.

[Aranda-Bricaire et al. 1995] Aranda-Bricaire, E., C. Moog, and J. Pomet, "A linear algebraic framework for dynamic feedback linearization", *IEEE Trans. Automatic Control*, Vol. 40, 1995, pp. 127-132.

[Barabasi and Oltvai 2004] Barabasi, A. and Z. Oltvai, "Network biology", *Nature Reviews Genetics*, Vol. 5, pp. 101-113, 2004.

[Bemporad et al. 2007] Bemporad, A., A. Bicchi, and G. Buttazzo, eds., *Proc. 10th International Conference on Hybrid Systems: Computation and Control,* Pisa, Italy, April 2007.

[Bujorianu and Lygeros 2004] Bujorianu, M. and J. Lygeros, "General stochastic hybrid systems: Modeling and optimal control", *Proc. 43rd IEEE Conference on Decision and Control*, Bahamas, December 2004.

[Carlson and Doyle 2002] Carlson, J. and J. Doyle, "Complexity and robustness", *Proc. National Academy of Sciences USA*, Vol. 99, pp. 2538-2545, 2002.

[Clarke et al. 1999] Clarke, E., O. Grumberg, and D. Peled, *Model Checking*, MIT Press, MA, 1999.

[Colbaugh and Glass 2003] Colbaugh, R. and K. Glass, "Information Extraction in Complex Systems", *Proc. NAA Conference on Computational Social and Organizational Science*, Pittsburgh, PA, June 2003 (invited plenary talk).

[Colbaugh and Glass 2007] Colbaugh, R. and K. Glass, "Predictability and prediction of social processes", *Proc. 4th Lake Arrowhead Conference on Human Complex Systems*, Lake Arrowhead, CA, April 2007 (invited talk).

[Colbaugh et al. 2007] Colbaugh, R., K. Glass, and G. Willard, "Scalable method for vulnerability analysis of complex networks", Patent application submitted by the National Security Agency, May 2007.

[Colizza et al. 2006] Colizza, V., A. Barrat, M. Barthelemy, and A. Vespignani, "The modeling of global epidemics: Stochastic dynamics and predictability", *Bull. Mathematical Biology*, Vol. 68, pp. 1893-1921, 2006.

[Epstein 2006] Epstein, J., *Generative Social Science: Studies in Agent-Based Computational Modeling*, Princeton University Press, NJ, 2006.

[Gardiner and Colbaugh 2006] Gardiner, J. and R. Colbaugh, "Development of a scalable bisimulation algorithm", Technical Report, New Mexico Institute of Mining and Technology, January 2006.

[Girard and Pappas 2007] Girard, A. and G. Pappas, "Approximation metrics for discrete and continuous systems", *IEEE Trans. Automatic Control*, Vol. 52, 2007, pp. 782-798.

[Glass et al. 2007] Glass, K., R. Colbaugh, and G. Willard, "The complex, evolving Internet: Implications for security", *Proc. MORS Workshop on Information Assurance*, Johns Hopkins University, March 2007 (invited plenary talk).

[Goncalves and Yi 2004] Goncalves, J. and T. Yi, "*Drosophila* circadian rhythms: Stability, robustness analysis, and model reduction", *Symp. Mathematical Theory of Networks and Systems*, Leuven, Belgium, July 2004.

[Gorban et al. 2006] Gorban, A., N. Kazantzis, I. Kevrekidis, H. Ottinger, and C. Theodoropoulos, Eds., *Model Reduction and Coarse-Graining Approaches for Multi-Scale Phenomena*, Springer-Verlag, Berlin, 2006.

[Habets and van Schuppen 2004] Habets, L. and J. van Schuppen, "A control problem for affine dynamical systems on a full-dimensional polytope", *Automatica*, Vol. 40, 2004, pp. 21–35.

[Hedstrom et al. 2000] Hedstrom, P., R. Sandell, and C. Stern, "Mesolevel networks and the diffusion of social movements: The case of the Swedish Social Democratic Party", *American Journal of Sociology*, Vol. 106, 2000, pp. 145-172.

[Ilic and Zaborszky 2000] Ilic, M. and J. Zaborszky, *Dynamics and Control of Large Electric Power Systems*, Wiley, NY, 2000.

[Kleinberg 2007] Kleinberg, J., "Cascading behavior in networks: Algorithmic and economic issues", *Algorithmic Game Theory* (Nisan et al., eds.), Cambridge University Press, 2007.

[Kushner 1967] Kushner, H.*, Stochastic Stability and Control*. Academic Press, NY, 1967.

Laviolette et al. 2008] Laviolette, R., K. Glass, and R. Colbaugh, "Deep information from limited observations of RYF systems: The forest fire model", Presentation at 28th Annual CNLS/LANL Conference, Santa Fe, NM, May 2008.

[Leskovec et al. 2006] Leskovec, J., L. Adamic, and B. Huberman, "The dynamics of viral marketing", *Proc. 7th ACM Conference on Electronic Commerce*, Ann Arbor, MI, June 2006.

[Martin et al. 2003] Martin, P., R. Murray, and P. Rouchon, "Flat systems, equivalence, and trajectory generation", CDS Technical Report 2003-008, California Institute of Technology, 2003.

[Milner 1989] Milner, R., *Communication and Concurrency*, Prentice-Hall, NJ, 1989.

[Narendra and Annaswamy 1988] Narendra, K. and A. Annaswamy, *Stable Adaptive Systems*, Prentice Hall, NJ, 1988.

[Newman 2003] Newman, M., "The structure and function of complex networks", *SIAM Review*, Vol. 45, pp. 167-256, 2003.

[NuSMV 2007] http://nusmv.irst.itc.it, 2007.

[Parrilo 2000] Parrilo, P., *Structured Semidefinite Programs and Semialgebraic Geometry Methods in Robustness and Optimization*, PhD dissertation, California Institute of Technology, 2000.

[Prajna et al. 2007] Prajna, S., A. Jadbabaie, and G. Pappas, "A framework for worst case and stochastic safety verification using barrier certificates", *IEEE Trans. Automatic Control*, Vol. 52, 2007, pp. 1415-1428.

[Ramadge and Wonham 1987] Ramadge, P. and W. Wonham, "Supervisory control of a class of discrete event systems", *SIAM J. Control and Optimization*, Vol. 25, pp. 206–230, 1987.

[Salganik et al. 2006] Salganik, M., P. Dodds, and D. Watts, “Experimental study of inequality and unpredictability in an artificial cultural market”, *Science*, Vol. 311, pp. 854-856, 2006.

[Sontag 1998] Sontag, E., *Mathematical Control Theory*, Second Edition, Springer, NY, 1998.

[SOSTOOLS 2007] http://www.cds.caltech.edu/sostools/, 2007.

[Stamp et al. 2009] Stamp, J., R. Colbaugh, R. Laviolette, A. McIntyre, and B. Richardson, “Impacts analysis for cyber attack on electric power systems”, SAND Report, Sandia National Laboratories, September 2008.

[Tabuada and Pappas 2006] Tabuada, P. and G. Pappas, ”Linear time logic control of discrete-time linear systems”, *IEEE Trans. Automatic Control*, Vol. 51, pp. 1862-1877, 2006.

[Watts et al. 2002] Watts, D., P. Dodds, and M. Newman, “Identity and search in social networks”, *Science*, Vol. 296, pp. 1302-1305, 2002.

[Wedeward 2006] Wedeward, K. Personal communication, Summer 2006.

Appendix A: Matlab program for fast finite (bi-)simulation of linear control systems

```
% Reformulated bisimulation algorithm -- prototype implementation
% Judy Gardiner and Rich Colbaugh
% 11/1/05

function dummy = bisim()

%----------
% Inputs
%----------

% Define continuous state space system
%   xdot = A*x + B

n = 2           % dimension of state space - max of 64 in this prototype
m = 1           % number of inputs
A = [0 1; -1 0]  % system matrix (nxn)
B = [0; 1]      % input matrix (nxm)

% Define lattice for finite state transition system
% A state is a box in the lattice defined as an n-vector.
%   q = [q(1); q(2)2; ... q(n)], with each q(i) ranging from 1 to d(i)
% The lattice is uniform in each dimension. Vertices defined as:
%   x(i) = alpha(i)*q(i) + beta(i) for q(i) ranging from 1 to d(i)+1

dimLattice = [4; 4] % dimensions of lattice, i.e., no. of boxes (states?)
                    % in lattice for each of n dimensions
alpha = [1; 1]      % scale factor for each of n dimensions
beta = [-3; -3]     % offset for each of n dimensions

%---------------------
% Calculate some sizes
%---------------------

numStates = prod(dimLattice)            % size of the discrete state set

%-----------------------------
% Precompute some common values
%-----------------------------

ProjB = any(B,2)    % logical projection of B onto each axis (nx1)
APos = zeros(n)
ANeg = zeros(n);
APos(A>0 & ~eye(n)) = A(A>0 & ~eye(n))  % Positive nondiagonal elements of A,
0 elsewhere
ANeg(A<0 & ~eye(n)) = A(A<0 & ~eye(n))  % Negative nondiagonal elements of A,
0 elsewhere
ProjAPos = sum(APos,2)   % Vector; sum of positive nondiagonal elements of A
in each row
ProjANeg = sum(ANeg,2)   % Vector; sum of negative nondiagonal elements of A
in each row
clear APos ANeg     % release space
```

```
%----------
% Outputs
%----------

% Outputs are two logical arrays defining the possible transitions between
% adjacent states.  Left is defined to be lower index, right is higher.
%   TransRight(n, d1, d2, ... dn)       left-to-right arrows
%   TransLeft(n, d1, d2, ... dn)        right-to-left arrows

% TransRight(i, k1, k2, ... kn) is true if there is a transition into
%   state q = [k1; k2; ... kn] from the left along dimension i

% TransLeft(i, k1, k2, ... kn) is true if there is a transition out of
%   state q = [k1; k2; ... kn] to the left along dimension i

TransRight = logical(zeros(n,numStates));
TransLeft = logical(zeros(n,numStates));

%-------------------------------------------------------
% Determine transitions for each pair of adjacent states
%-------------------------------------------------------

% For each pair of adjacent states q and q' in the lattice, with q>q',
% determine q'-->q (transition to right/up) and q-->q' (transition to
% left/down) based on vertex information. (q' is q prime)
% Note: Each q (except on edge of lattice) has 2n adjacent states, one
% left/lower and one right/upper in each dimension.
% Indices of q are the indices of its lowest numbered vertex.

% loop through all states
for iState = 1:numStates
    iState
    q = vind2sub(dimLattice,iState) % get coordinates for q
    % Look at left (lower) adjacent state in each dimension
    for k = 1:n         % loop through dimensions
        % q' is the same as q except 1 less in dimension i
        if q(k) == 1     % check for left/lower edge of lattice
            continue
        end
        if ProjB(k)
            TransRight(k,iState) = true;
            TransLeft(k,iState) = true;
        end
        x = alpha .* q + beta   % Calculate x from q (scalar multiply)
        p = A(k,:)*x            % Projection of Ax onto axis i
        if p > 0  ||  p+ProjAPos(k) > 0
            TransRight(k,iState) = true;
        end
        if p < 0  ||  p+ProjANeg(k) < 0
            TransLeft(k,iState) = true;
        end
    end
end
```

```
TransRight
TransLeft

%--------------------------------------------------------------------
% Vector form of ind2sub. Converts a 1-dimensional index into an array
% of subscripts for an n-dimensional array with dimensions given by d.
%--------------------------------------------------------------------
function y = vind2sub(d,ind)
n = size(d,1);
y = zeros(n,1);
i = ind - 1;         % change from 1-based to 0-based indexing
if n > 1
    for k=1:n-1
        y(k) = mod(i,d(k));
        i = (i-y(k)) / d(k);
    end
end
y(n) = i;
y = y + ones(n,1);   % go back to 1-based indexing

%--------------------------------------------------------------------
% Vector form of sub2ind. Finds the 1-dimensional index from an array
% of subscripts for an n-dimensional array with dimensions given by d.
%--------------------------------------------------------------------
function ind = vsub2ind(d,s)
n = size(d,1);
ind = s(n) - 1;      % change from 1-based to 0-based indexing
if n > 1
    for k=n-1:-1:1
        ind = ind*d(k) + s(k) - 1;
    end
end
ind = ind + 1;       % go back to 1-based indexing

% Computes the transitions into and out of a state q (defined by vertex v)
% with respect to adjacent states q'<q in the dimensions (k) specified.
% Function bisim_init must be called first.
%
function [TransIn, TransOut] = bisim_trans(v,kDim)

% Inputs:
%   v    -   n-vector; indices of lowest vertex of state q; shared vertex
%            between q and all q' being considered
%   kDim -   row vector defining the dimensions of interest; transitions to
%            be computed are those parallel to these axes; length of kDim is
%            between 1 and n

% Outputs:
%   TransIn    - logical row vector of same size as kDim; true if there is a
%                transition into q from the lower adjacent state in the
%                dimension specified by the corresponding element of kDim.
%                These transitions represent arrows to the right or up.
%   TransOut   - similar to TransIn; indicates transitions out of q into
%                lower adjacent states; represents arrows to the left or
%                down.
```

```
% Global variables -- These must have been set by bisim_init
global n A ProjB ProjAPos ProjANeg
global nParts Xmin Xscale

% Initialize transitions to false
TransIn = logical(zeros(size(kDim)));
TransOut = logical(zeros(size(kDim)));

% Compute x corresponding to v, the lowest vertex of q
x = Xscale .* (v-1) + Xmin;   % Calculate x from q (scalar multiply)

% Determine all transitions into and out of q from adjacent states q' in
% the dimensions listed in kDim, with q'<q.  Each q' is the same as q
% except 1 less in dimension k.
for i = 1:size(kDim,2)    % loop through dimensions (axes)
    k = kDim(i);
    if v(k) == 1     % check for left/lower edge of lattice
        continue
    end
    p = A(k,:)*x;            % Projection of Ax onto axis k
    TransIn(i) = ProjB(k)  |  p+ProjAPos(k) > 0;
    TransOut(i) = ProjB(k)  |  p+ProjANeg(k) < 0;
end
```

Appendix B: Matlab program for finite (bi-)simulation of HDS model for 20-bus power grid

```
% bisim_20bus

% Script to check a trajectory from the 20-bus continuous model against the
% finite-state-machine bisimulation model
% Note:  This code is written specifically to work with Kevin's model.
% Inputs:
%   System matrices: A0, A1, A2, A3, B
%   Incidence matrix for line trips: IMat
%   Thresholds for line trips: dthetamax
%   Output trajectory: xout, tout
%   Finite-state partition size: sizeXParts (state variables),
%     sizeYParts (dtheta variables),
%     both are uniform along each axis and symmetric about 0
%   Number of partitions for each trip variable (dtheta) axis: nYParts
%     Note: Trip variables are treated as system outputs and named y.
%     They are dtheta values scaled by dthetamax so trip points are +/-1.

% Outputs:
%   Flag indicating validity of trajectory under bisimulation (true=valid)
%   Finite-state trajectory with time each state was entered
%   Time and identity of all trip events

% Clear memory and close figures
clear all; close all;

% Define global variables, set in bisim_init
global AA ProjB ProjAPos ProjANeg    % projections used in bisimulation
global Xscale Xoff                   % grid definition used in bisimulation
% Load system matrices
load ABs                % A0, A1, A2, A3, B0, IMat
dthetamax = 0.01*ones(30,1); dthetamax([6,9,10,14,27]) = 0.025;

% Transform system to use theta differences instead of thetas as state
% variables.  This function contains a lot of model-specific information.
% With transformed system, xdot=Ax+Bu; y=Cx; trip if any(abs(y)>1).
% Transformed state grid is orthogonal.
% Note:  Outputs are transformed matrices, except T and C.
[T,A0,A1,A2,A3,B0,C]=transabc(A0,A1,A2,A3,B0,IMat,dthetamax);

% Load trajectory
load traj               % xout, tout
npoints=size(xout,1);
y=C*xout';          % Compute output variables, aka line trip variables
x=T\xout';          % Transpose and transform state vectors
tripped=false(30,1);    % which lines have tripped
% Set finite-state grid information
[Xscale,Xoff,Yscale,Yoff,qymax] = set_fsgrid;

% Initialize variables, assuming trajectory starts at x=0
A=A0;
B=B0;
```

```
bisim_init(A,B);
q=zeros(57,1);      % finite state
qout=q';            % list of finite states
tqout=0;            % list of times associated with finite states
teout=[];           % times of trip events
qeout=[];           % finite state at time of trip events
ieout=[];           % index of trip events
validtraj=true;
xlast=zeros(57,1);  % continuous state
for k=1:npoints
    qnext=floor((x(:,k)-Xoff)./Xscale);
    % Check for valid transition under bisimulation
    validtraj=check_trans(q,qnext,xlast,x(:,k));
    xlast=x(:,k);
    if ~validtraj
        qout=[qout; qnext'];    % save invalid state for output
        tqout=[tqout; tout(k)];
        disp(['Invalid transition found at time ',num2str(tout(k))]);
       break
    end
    % Output new finite state
    q=qnext;
    qout=[qout; q'];
    tqout=[tqout; tout(k)];
    % Check for line trip
    qy=floor((y(:,k)-Yoff)./Yscale);
    itrip=check_trip(qy,qymax,tripped); % ix of newly tripped line or 0
    if itrip>0
        tripped(itrip)=true;
        % Output line trip info
        teout=[teout; tout(k)];
        ieout=[ieout; itrip];
        qeout=[qeout; q'];
        % Update system matrix, projections for bisimulation, fs grid
        disp(['Event ',num2str(itrip),' found at time ',num2str(tout(k))]);
        switch itrip
            case 8
                A=A2;
            case 12
                A=A1;
            case 30
                A=A3;
            otherwise
                disp(['Response not found for ie = ', num2str(itrip)]);
        end
        bisim_init(A,B);
    end
end

if validtraj
    disp('Trajectory is valid under bisimulation.')
end
% Bisimulation finished.  Save output variables to file.
save bisimout.mat validtraj qout tqout teout ieout qeout
```

Appendix C: Matlab programs for predictability assessment of online market model

```
% Predictability Assessment for Online Market Model

% SOS-based predictability assessment for online market
% model: IC uncertainty quantification analysis case.

% Uses SOSTOOLS version 2.01 and SeDuMi 1.05R5

clear; echo on;

syms x1 x2;

sigma = 0.1;        % 0.1 1.0
thresh = 0.8;       % 0.9

% Vector fields
% f1 = [-x1+0.5*x2;
%       -2.0*x2];
% f2 = [0.5*x2-x1*x2;
%       -x2*x2];
% f1 = [-x1*x2+0.5*x2;
%       -2.0*x2];
% g =  10.0*sigma*x2;

  f1 = [-x1*x2+0.5*x2;
        -2.0*x2];

  g =  4.0*sigma*x2;

% Degree of the barrier certificates
deg = 10;

prog = sosprogram([x1; x2]);

% Constructing B1 -- it must be >=0 on \mathcal{X}
[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2],0:deg/2));
B1 = sos1+mu1*(x1+1.0)*(2.0-x1)+mu2*(x2+1.0)*(2.5-x2);

[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
expr1 = B1-mu1*(x1-thresh)*(2.0-x1)-mu2*(x2+1.0)*(2.5-x2)-1;
%expr1 = B1-mu1*(x1-thresh)*(2.0-x1)-mu2*(x2+1.0)*(0.1-x2)-1;
prog = sosineq(prog,expr1);

[prog,gamma] = sospolyvar(prog,1);
[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
expr2 = B1+mu1*(x1-0.3)*(0.6-x1)+mu2*(x2-0.9)*(1.0-x2);
```

```
%expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*(x2-0.9)*(1.0-x2);
%expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*(x2-0.0)*(0.1-x2);
%expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*x2*(0.02-x2);
expr2 = -expr2+gamma;
prog = sosineq(prog,expr2);

[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+0.5*g^2*diff(B1,x1,2))...
        -mu1*(x1+1.0)*(2.0-x1)-mu2*(x2+1.0)*(2.5-x2);
prog = sosineq(prog,expr3);

prog = sossetobj(prog,gamma);

% Impose a lower bound on gamma, for better termination

prog = sosineq(prog,gamma-0.5);

prog = sossolve(prog);

% ================================================

% Get solution
GMA = sosgetsol(prog,gamma)



% Predictability Assessment for Online Market Model

% SOS-based predictability assessment for online market model:
% parametric uncertainty quantification analysis case.

% Uses SOSTOOLS version 2.01 and SeDuMi 1.05R5

clear; echo on;

syms x1 x2 v;   % v1 \in [0.9, 1.1], v1 \in [0.8, 1.2]
                % v2 \in [3.0, 5.0], v2 \in [2.0, 6.0]

sigma = 0.1;        % 0.1 1.0
thresh = 0.8;       % 0.9

% Vector fields
% f1 = [-x1+0.5*x2;
%       -2.0*x2];
% f2 = [0.5*x2-x1*x2;
%       -x2*x2];
% f1 = [-x1*x2+0.5*x2;
%       -2.0*x2];
% g =  10.0*sigma*x2;

  f1 = [-x1*x2+0.5*x2;
        -2.0*x2];
```

```
  g =  v*sigma*x2;

% Degree of the barrier certificates
deg = 10;

prog = sosprogram([x1; x2; v]);

% Constructing B1 -- it must be >=0 on \mathcal{X}
[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2],0:deg/2));
B1 = sos1+mu1*(x1+1.0)*(2.0-x1)+mu2*(x2+1.0)*(2.5-x2);

[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
expr1 = B1-mu1*(x1-thresh)*(2.0-x1)-mu2*(x2+1.0)*(2.5-x2)-1;
%expr1 = B1-mu1*(x1-thresh)*(2.0-x1)-mu2*(x2+1.0)*(0.1-x2)-1;
prog = sosineq(prog,expr1);

[prog,gamma] = sospolyvar(prog,1);
[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*(x2-0.9)*(1.0-x2);
%expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*(x2-0.0)*(0.1-x2);
%expr2 = B1+mu1*(x1-0.4)*(0.5-x1)+mu2*x2*(0.02-x2);
expr2 = -expr2+gamma;
prog = sosineq(prog,expr2);

[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,v],0:deg/2-1));
expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+0.5*g^2*diff(B1,x1,2))...
    -mu1*(x1+1.0)*(2.0-x1)-mu2*(x2+1.0)*(2.5-x2)-mu3*(v-3.0)*(5.0-v);
%expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+0.5*g^2*diff(B1,x1,2))...
%    -mu1*(x1+1.0)*(2.0-x1)-mu2*(x2+1.0)*(2.5-x2);
prog = sosineq(prog,expr3);

prog = sossetobj(prog,gamma);

% Impose a lower bound on gamma, for better termination

prog = sosineq(prog,gamma-0.5);

prog = sossolve(prog);

% ============================================

% Get solution
GMA = sosgetsol(prog,gamma)
```

Appendix D: Matlab program for SOS analysis of S-HDS epidemic model

```
% Social Cascading SIR Via Context Switching

% Multi-scale model implemented as an S-HDS with
% rigorous stochastic SIR continuous dynamics.

% Uses SOSTOOLS version 2.01 and SeDuMi 1.05R5

clear; echo on;

syms x1 x2 x3 v;    % v \in [0.9, 1.1], v \in [0.9, 1.2], v \in [0.8, 1.2]

number_contexts = 2.0;   % 1.0, 2.0, 4.0
little_pop = number_contexts;
big_pop = 10.0;  % 2.0

Imax = 5.0;

beta = (1.0/(number_contexts*number_contexts));  % 0.5 1.0
delta = 0.05;
sigma = 0.02;    % 0.01, 0.02, 0.0225, 0.025

lambda = v*0.001;      % 0.001 0.004
thresh = 0.01*lambda;  % 0.05*lambda


% Initial probability distribution for the discrete state
%p = 0.5;
p = 1.0;

% Vector fields
f1 = [-beta*x1*x2*x2;
      beta*x1*x1*x2-delta*x2;
      0.0];

f2 = [0.0;
      0.0;
      x3*(big_pop-x3)];

%f1 = [-2.0*beta*x1-beta*x1*x2+(beta*pop1-delta)*x1;
%      delta*x1];

%f2 = [-2.0*beta*x1-beta*x1*x2+(beta*pop2-delta)*x1;
%      delta*x1];

%g = [sigma;
%     -sigma;
%     0.0];

g = [-sigma*x2;
```

```
      sigma*x1;
     0.0];

 % Degree of the barrier certificates
deg = 6;  % 6  10

prog = sosprogram([x1; x2; x3; v]);

% Constructing B1, B2 -- they must be >=0 on \mathcal{X}
[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2));
B1 = sos1+mu1*(x1+10.0)*(10.0-x1)+mu2*(x2+10.0)*(10.0-x2)...
     +mu3*(x3+10.0)*(20.0-x3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2));
B2 = sos1+mu1*(x1+10.0)*(10.0-x1)+mu2*(x2+10.0)*(10.0-x2)...
     +mu3*(x3+10.0)*(20.0-x3);


[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr1 = B1-mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
          -mu3*(x3-Imax)*(20.0-x3)-1;
prog = sosineq(prog,expr1);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr1 = B2-mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
          -mu3*(x3-Imax)*(20.0-x3)-1;
prog = sosineq(prog,expr1);


[prog,gamma] = sospolyvar(prog,1);
expr2 = subs(p*B1+(1-p)*B2,{x1,x2,x3},{(number_contexts-0.5),0.5,0.1});
expr2 = -expr2+gamma;
prog = sosineq(prog,expr2);



%[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
%[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
%[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
%expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+diff(B1,x3)*f1(3)...
%        +0.5*sigma^2*(diff(B1,x1,2)+diff(B1,x2,2)-
2.0*diff(diff(B1,x1),x2))...
%        +(lambda*x2-thresh)*B2-(lambda*x2-thresh)*B1)...
%        -mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
```

```
 %        -mu3*(x3+10.0)*(20.0-x3);
%prog = sosineq(prog,expr3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu4] = sossosvar(prog,monomials([x1,x2,x3,v],0:deg/2-1));
expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+diff(B1,x3)*f1(3)...

+0.5*sigma^2*(diff(B1,x1,2)+diff(B1,x2,2)+2.0*diff(diff(B1,x1),x2))...
        +(lambda*x2*x2-thresh)*B2-(lambda*x2*x2-thresh)*B1)...
        -mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
        -mu3*(x3+10.0)*(20.0-x3)-mu4*(v-0.9)*(1.2-v);
prog = sosineq(prog,expr3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr3 = -(diff(B2,x1)*f2(1)+diff(B2,x2)*f2(2)+diff(B2,x3)*f2(3))...

+0.5*sigma^2*(diff(B2,x1,2)+diff(B2,x2,2)+2.0*diff(diff(B2,x1),x2))...
        -mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
        -mu3*(x3+10.0)*(20.0-x3);
prog = sosineq(prog,expr3);

prog = sossetobj(prog,gamma);

% Impose a lower bound on gamma, for better termination
prog = sosineq(prog,gamma-0.1);
%prog = sosineq(prog,gamma-0.2);
%prog = sosineq(prog,gamma-0.346);
%prog = sosineq(prog,gamma-0.145);
%prog = sosineq(prog,gamma-0.069);

prog = sossolve(prog);

% ============================================
% Get solution
GMA = sosgetsol(prog,gamma)
```

Appendix E: Matlab program for SOS analysis of S-HDS social movement model

```
% Social Movement Cascades Via Context Switching

% Multi-scale model implemented as an S-HDS
% with Hedstrom continuous dynamics.

% Uses SOSTOOLS version 2.01 and SeDuMi 1.05R5


clear; echo on;

syms x1 x2 x3;

number_contexts = 4.0;   % 1.0, 2.0, 4.0
little_pop = number_contexts;
big_pop = 10.0;  % 2.0

Imax = 5.0;

beta = (1.0/number_contexts);  % 0.5 1.0
delta1 = 0.1;  % 0.1 0.025 0.55  empirical evidence suggests delta1=10*delta2
delta2 = (0.01/number_contexts);  % 0.002 0.005 0.01

lambda = 0.001; % 0.001 0.004

thresh = 0.05*lambda;


% Initial probability distribution for the discrete state
%p = 0.5;
p = 1.0;

% Vector fields
f1 = [-beta*x1*x2;
      beta*x1*x2-delta1*x2-delta2*x2*(little_pop-x1-x2);
      0.0];

f2 = [0.0;
      0.0;
      x3*(big_pop-x3)];

%f1 = [-2.0*beta*x1-beta*x1*x2+(beta*pop1-delta)*x1;
%      delta*x1];

%f2 = [-2.0*beta*x1-beta*x1*x2+(beta*pop2-delta)*x1;
%      delta*x1];

g = 0.0;
```

```
 % Degree of the barrier certificates
deg = 6;  % 6  10

prog = sosprogram([x1; x2; x3]);

% Constructing B1, B2, B3 -- they must be >=0 on \mathcal{X}
[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2));
B1 = sos1+mu1*(x1+10.0)*(10.0-x1)+mu2*(x2+10.0)*(10.0-x2)...
     +mu3*(x3+10.0)*(20.0-x3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,sos1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2));
B2 = sos1+mu1*(x1+10.0)*(10.0-x1)+mu2*(x2+10.0)*(10.0-x2)...
     +mu3*(x3+10.0)*(20.0-x3);


[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr1 = B1-mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
          -mu3*(x3-Imax)*(20.0-x3)-1;
prog = sosineq(prog,expr1);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr1 = B2-mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
          -mu3*(x3-Imax)*(20.0-x3)-1;
prog = sosineq(prog,expr1);


[prog,gamma] = sospolyvar(prog,1);
expr2 = subs(p*B1+(1-p)*B2,{x1,x2,x3},{(number_contexts-0.5),0.5,0.1});
expr2 = -expr2+gamma;
prog = sosineq(prog,expr2);


%[prog,mu1] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
%[prog,mu2] = sossosvar(prog,monomials([x1,x2],0:deg/2-1));
%expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2) + 0.5*g^2*diff(B1,x2,2)...
%    + 0.5*B2-0.5*B1)-mu1*(4^2-x1^2)-mu2*(4-x2)*(x2+1.5);
%prog = sosineq(prog,expr3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr3 = -(diff(B1,x1)*f1(1)+diff(B1,x2)*f1(2)+diff(B1,x3)*f1(3)...
        +(lambda*x2-thresh)*B2...
        -(lambda*x2-thresh)*B1)...
        -mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
```

```
         -mu3*(x3+10.0)*(20.0-x3);
prog = sosineq(prog,expr3);

[prog,mu1] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu2] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
[prog,mu3] = sossosvar(prog,monomials([x1,x2,x3],0:deg/2-1));
expr3 = -(diff(B2,x1)*f2(1)+diff(B2,x2)*f2(2)+diff(B2,x3)*f2(3))...
        -mu1*(x1+10.0)*(10.0-x1)-mu2*(x2+10.0)*(10.0-x2)...
        -mu3*(x3+10.0)*(20.0-x3);
prog = sosineq(prog,expr3);

prog = sossetobj(prog,gamma);

% Impose a lower bound on gamma, for better termination
prog = sosineq(prog,gamma-0.1);
%prog = sosineq(prog,gamma-0.2);
%prog = sosineq(prog,gamma-0.346);
%prog = sosineq(prog,gamma-0.145);
%prog = sosineq(prog,gamma-0.069);

prog = sossolve(prog);

% ============================================
% Get solution
GMA = sosgetsol(prog,gamma)
```