Improving the Efficiency of Markov Chain Analysis of Complex Distributed Systems

Christopher Dabrowski
Fern Hunt
Improving Efficiency of Markov Chain Analysis of Complex Distributed Systems

Christopher Dabrowski
Advanced Networking Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8530

Fern Hunt
Mathematical and Computational Sciences Division
Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8530

(Additional author(s) to be named when draft is competed)

June 2010
Abstract: In large-scale distributed computing systems, the interactions of many independent components may lead to emergent global system behaviors with unforeseen, often detrimental, outcomes. The increasing economic importance of distributed systems such as cloud computing systems, grid computing systems, and the Internet, argues for developing analytical tools to understand, and predict, complex system behavior in order to ensure availability and reliability of computing services. In previous work, we described one such tool in which a piece-wise homogeneous Discrete Markov chain representation of a grid computing system can be systematically perturbed to predict situations that lead to marked performance degradations and system-wide failure. While the run times of the Markov chain model compared favorably with testbeds or detailed large-scale simulations, it was still often necessary to execute a sizable number of alternative perturbations of the model to identify scenarios in which system performance is likely to degrade or in which anomalous behaviors may occur. Here, we evolve our original approach and describe two novel methods for more quickly identifying portions of the Markov chain that are likely to be sensitive to perturbation. The first method involves finding cut sets, consisting of state transitions that effectively disconnect all paths in a Markov chain from the initial state to a desired end state. We show that by perturbing the state transitions in the cut set, it is possible to more quickly identify scenarios in which system performance is adversely affected. We also show this new method can be applied to larger Markov models than in our earlier work and therefore provides better scalability. We then present a second method, in which the Spectral Expansion Theorem is used to analyze the eigensystem of a set of Markov transition probability matrices (TPMs) in order to identify eigenvectors and eigenvalues that can be used to predict system performance. We describe how this second approach can also be used to indicate which state transitions, if perturbed, are likely to adversely affect system performance. Results are presented for both methods to show that they can be used to identify the same failure scenarios presented in our earlier paper (as well as additional scenarios, using the first method), while reducing the number of perturbations of the Markov model (or eliminating Markov simulation altogether, using the second method). We believe that these methods provide a basis for creating practical tools for analysis of complex systems and discuss future work towards this end.

Keywords: complex systems; perturbation analysis; discrete time piece-wise homogenous Markov chain; graph theory; minimal s-t cut set; Spectral Expansion Theorem; eigenvector; eigenvalue; grid computing.
1. Introduction

In recent years, the advent of large-scale distributed systems, such as computing grids and commercial cloud systems has enabled mass computing services to be made available to large numbers of users on demand. In large-scale heterogeneous, dynamic, systems such as these, the interactions of many independent components will likely lead to emergent system-wide behaviors with unforeseen, often detrimental, outcomes [Mi2006]. The rapid growth and increasing economic importance of these systems [Ca2008a, Ra2006] argues for developing analytical tools to understand, and predict, complex system behavior in order to ensure availability and reliability of computing services.

In particular, tools that can predict how system performance is impacted by changes to workload, system design, and key operational parameters will be of great importance. Studies of alternative economic strategies ([Ch2002], Ye2005, Mi2008] and failure scenarios [Mi2006] have shown that small variations in key system variables can lead to large differences in overall system performance. While large-scale simulations are more practical than operational testbeds, computational expense can increase dramatically with model size, a critical factor for studying large-scale systems such as the Internet.

To remedy this situation, we presented an approach in earlier work [Da2009a] in which discrete time Markov chain analysis was used to model dynamics of large-scale grid systems. In this approach, we developed a succinct Markov chain representation of a grid computing system that included a set of transition probability matrices (TPMs) that summarized system dynamics over different time periods. The TPMs could be perturbed to represent different system execution paths by changing values of individual transition probabilities. A perturbation algorithm was developed to systematically identify execution paths that led to degradations of grid system performance and system-wide failures. This allowed Markov chain analysis to be used as a predictive tool for how an operational system might react over time under different conditions. The approach could be used in cases where transition probabilities changed with (e.g., were non-homogenous with respect to) time and workload. In comparison with detailed large-scale simulation or testbeds, computational cost of this approach was reduced. One reason for this was that the stochastic characteristics of Markov chains allow model size to be unaffected by the scale of the system being modeled, as expressed in terms of number of components or workload. Another reason was that the perturbation algorithm was designed to enumerate alternative paths only within defined sub-areas of the Markov chain. Despite these gains in efficiency, computational effort could increase significantly as the number of model states increased, making it expensive to apply the perturbation algorithm to larger Markov chains. This in turn made it difficult to quickly discover those parts of a large Markov chain where changes could lead to declines in system performance.

To address this problem, we expand on our previous work by adding capabilities that allow fast identification of portions of a Markov chain where perturbation is most likely to affect system performance. We describe two methods that represent different approaches to do this. First, we employ efficient algorithms based on graph theory concepts to identify cut sets that disconnect all paths between two vertices in a graph. These algorithms can be used to identify states and state transitions, which if removed or blocked, would disconnect all paths from an initial state to desired end states and thus prevent processes from entering them. This allows specific state transitions to be directly perturbed to determine impact on performance. We show that this approach can be used to find the same parts of the grid system Markov chain where the perturbation algorithm also predicted marked performance degradation as reported in [Da2009a], but with a much lower computational cost. Using the large-scale simulation as a real-world proxy, we also apply the method to the grid computing system under near steady state conditions and then extend the procedure to a new domain—the modeling of the impact of different congestion control regimes on data flows in a network. Finally, we show that the method can be used on much larger Markov chains to identify areas of performance
degradation. To our knowledge, graph theory concepts have not previously been used in this manner to identify perturbations of Markov chains that predict drastic changes in system performance.

We then present a second method, called the theoretical method, in which the Spectral Expansion Theorem is used to analyze the eigensystem of a set of Markov transition probability matrices (TPMs) in order to identify eigenvectors that are critical for predicting system performance. We show that changes in the leading eigenvector of the transient part of the TPM correlate reasonably well with performance changes discovered through Markov simulation. We describe how this second approach can also be used to indicate which state transitions, if perturbed, are likely to adversely affect system performance. Examples are provided of the use of the theoretical method to identify the same parts of the grid system Markov chain that were identified in [Da2009a], and by the first method, as being sensitive to perturbation. In this way, we show that the theoretical method can also provide a viable alternative to the perturbation algorithm presented in [Da2009a] at reduced computation cost. Results from application of both methods are detailed for all cases and corroborated by earlier results obtained by applying more exhaustive perturbation algorithm. We show that the two general approaches presented in this paper can be equally effective, but more efficient, than our previous Markov chain analysis approach of [Da2009a]. We also present results indicating scalability of the graph theoretic method.

The plan of this paper is as follows. Section 2 discusses previous work by other authors on using Markov chain analysis including previous uses of Graph theory for Markov chains. Section 3 overviews the most important results in our previous publication [Da2009a], focusing on use of Markov chain concepts to model dynamic systems. Most importantly, Section 3 describes the perturbation algorithm for critical transitions in Markov chains where changes are likely to affect performance, which while effective, still requires a large expenditure of computational effort. This sets the stage for the contributions of this paper. Section 4 describes how minimal cut sets on paths through the graph of a Markov chain can be used to directly identify critical state transitions where perturbation causes performance degradations. In section 4, a simple method for identifying minimal cut sets is discussed and examples are provided. Section 5 presents the results of using the methods described in section 4 to predict areas of the Markov chain that are sensitive to perturbation. Section 5 compares these results with those produced by the perturbation algorithm described in section 3 as well as with more detailed large-scale simulation. The results of this comparison show that minimal cut set identification are equally effective as the perturbation algorithm in finding areas of the Markov chain that are sensitive to perturbation. Section 6 addresses the issue of using this approach on large Markov chains and presents an algorithm for finding minimal cut sets that is intended to work on larger problems. Preliminary results are then presented on the application of the new algorithm to larger problems. Section 7 presents the theoretical method describe above together with examples of its application. Section 8 concludes.

<table>
<thead>
<tr>
<th>Section</th>
<th>Topics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 2</td>
<td>Review of previous work on use of Markov chain analysis to study dynamic systems.</td>
</tr>
<tr>
<td>Section 3</td>
<td>Use of Markov chain models to represent dynamic systems. Review of perturbation algorithm in [Dabr2009a] for finding areas of Markov chains that are sensitive to perturbation.</td>
</tr>
<tr>
<td>Section 4</td>
<td>Use of minimal cut set analysis to find state transitions in Markov chains that are sensitive to perturbation.</td>
</tr>
<tr>
<td>Section 5</td>
<td>Comparison of results of minimal cut set analysis with results from perturbation algorithm described in Section 3. Verification that minimal cut set analysis finds areas of Markov chains that are sensitive to perturbation.</td>
</tr>
<tr>
<td>Section 6</td>
<td>Description of minimal cut set analysis method for large, complex Markov chains and presentation of preliminary results</td>
</tr>
<tr>
<td>Section 7</td>
<td>Presentation theoretical method, in which the Spectral Expansion Theorem is used to analyze the eigensystem of a Markov chain.</td>
</tr>
</tbody>
</table>
2. Previous Work

Discrete Time Markov chains (DTMCs) have long been used to study dynamic system behavior in many real-world domains. In [Da2009a], we provide examples of these efforts which have focused on providing quantitative measures of performance or reliability. Most closely related to our work is that of [Wu2006, Fe2009], in which a feedback control loop process is used to moderate delay in a network, where delay characteristics are modeled using a Markov chain. In contrast, our work [Da2009a] uses Markov chain models to understand system-wide behaviors that occur as a consequence of significant events or decisions that affect the system as a whole. Therefore, this section summarizes previous work on using Markov chains to study dynamic system behavior, focusing on the main topic of this paper—methods to reduce problem size and, more specifically, perturbation analysis techniques that reduce the size of the perturbation space.

The combinatorial increase of the number of states in large DTMC problems has long been recognized as a significant barrier. On solution approach is to combine, or lump, states with similar characteristics into larger aggregated units, first introduced in [Ke1976]. Since then, various lumping approaches have been proposed, including [Si1992] [Bu1995] [Ni2004] who use model structure and symmetry to reduce size, [Au1991] who rely on group-theoretic concepts for size reduction. Other approaches for reducing model size have been based on Stochastic Activity Nets [Sa1991], stochastic colored nets [Ch1993], use of reward variable structures to identify symmetries [Ob2001], and use of eigenvector equivalence classes to partition a Markov state space into lumps [Ja2007]. While these approaches have merit, their reliance on existence of specific structural characteristics limits use in many cases. Moreover, the process of lumping could eliminate critical states and related state transitions that crucially impact overall system performance and need be explicitly identified (as we show in this paper).

In the last three decades, perturbation analysis of discrete time Markov chains has been the topic of significant theoretical [Sc1968][De1983][Ha1992] and computational study [Me1989][St1994]. However, much of this work, as for example [Ha1992][Me1989][St1994], has focused on ergodic Markov chains and computation of the stationary vectors. In contrast, our work focuses on identifying perturbations that have a significant effect on the behavior of an absorbing Markov chain.

Like the problem of model size, the size of a typical perturbation space may quickly become computationally intractable, if there are many combinations of alternative system variable values to consider. To attack this problem, some researchers [Ho1985], [Su1989] have advanced the idea of perturbation analysis of discrete event systems by calculating system performance gradients that are based on key decision parameters. This approach estimated the sensitivity of changes to decision parameters in order to optimize system performance. In some cases, gradient-based approaches needed to observe as few as one execution path of a system to reduce the size of the perturbation space. This approach was adapted for Markov chains by estimating gradients for alternative execution paths [Ho1988], [Su1989] and extended by [Ca2005], [Ca2008b] who reduced problem size by grouping state transitions on the basis of related events. This approach was believed to scale with the number of events and size of the system. However, not all problems were found to be reducible to a form which allowed tractable calculation of gradients. While gradient-based perturbation algorithms have demonstrated potential as efficient tools for analysis of some complex systems, they also introduced significant computational issues and were found not to be applicable to all Markov problems. Perhaps more importantly, the gradient-based approaches appeared more geared to optimization problems, rather than the more general problem of assessing alternative execution paths in dynamic systems and identifying areas of potential drastic performance reduction. While gradient-based methods merit, they did not appear of direct use for large DTMC problems where it is desirable to identify specific states and state transitions that affect performance; hence, we turned to graph theory.

Graph-theoretic methods have previously been applied to Markov chains. Graph decomposition has also been used to calculate stationary probability distributions of Markov chains [Be2002, Fo1988, Ga2008]
including large-scale models [Ha1996]. In [Ga2008], the authors developed methods for computing approximations for first passage times and number of visits in a fixed state before absorption in cases where the size of perturbation was small. In [Ki2004], graph decomposition was used to estimate bounds on volatility of stationary distribution of Markov chains with irreducible TPMs. Similarly, distances between stationary distributions of perturbed Markov chains have been calculated through analysis of directed subgraphs [So2003]. In the preceding works, graph theoretic methods were used to measure distance between individual perturbations, a measurement that could be used to aid in finding parts of a Markov chain that were sensitive to perturbation. In contrast, we seek to provide a more direct means to determine where perturbation of the probability of transitions between states leads to large system performance degradations. To do this, we leverage previous work on minimal cut set analysis [Ts1980, Ka1996, Li2003, Ye2003] and others described below. Cut set analysis methods have long been used for analysis of VLSI design, network systems, and design of various other distributed systems. For example, in [Du1999, Ta2004], minimal cut set generation has been applied to graphs of avionics system components in order to identify the shortest sequence of individual component failure. However, to our knowledge, minimal cut set analysis has not yet been used to analyze Markov chain representations of the evolution of a system through a sequence of states. The approach we describe here appears to be novel. While here, as in our earlier effort [Da2009a], the approach we explore does not completely solve the issue of reducing the size of the Markov chain perturbation space, it does further reduce computational cost and is equally effective in comparison to [Da2009a]. Thus it serves as a method of practical significance that can be potentially of use in pinpointing where in the Markov chain, perturbation can have the greatest effect on system performance.
3. Review of Discrete Time Markov Chain Approach

In this section, we review previous work on our approach to modeling a dynamic system as a piecewise homogenous discrete time Markov chain. We show the application of this approach to the grid computing system and Abilene network models [Mi2009], both of which were developed by observing the operation of large-scale simulations. We then present an overview of the perturbation algorithm described in [Da2009a] that does a limited brute-force search of selected parts of a Markov chain to identify areas where changes to state transitions probabilities lead to significant performance degradations. Finally, we provide an analysis of the efficiency and computational cost of the algorithm. The success of this perturbation algorithm and its relatively high computational cost for large problems provide the motivation for the development of more efficient methods. These are described in sections 4-6.

3.1 The Markov Chain Model of a Grid Computing System

The Markov chain model is derived from a previous large-scale grid computing system model [Mi2006] [Mi2008] that simulates of the progress of a large number of computing tasks from the time they are submitted to the grid for execution by an end user to the time they either complete or fail. Figure 3.1 shows this Markov model as a state diagram for a single task. The state diagram has 7 states: an Initial state, where the task remains prior to submission; a Discovering state, during which service discovery middleware locates candidate grid service providers to execute the task; a Negotiating state during which a Service Level Agreement (SLA) to execute the task is negotiated with one of the discovered providers; a Waiting state for tasks that are temporarily unsuccessful in discovery or negotiation; a Monitoring phase in which a task is executed by a contracted provider; and the Tasks Completed or Tasks Failed states. Transitions between states, illustrated by the arrows in Figure 3.1, represent actions taken by the grid system to process a task as described in [Da2009a]. The model is considered an absorbing chain because all tasks ultimately must enter one of two absorbing states, Tasks Completed or Tasks Failed, from which they cannot leave.

![Figure 3.1. State model of grid computing system.](image-url)

To convert the state model in Figure 3.1 into a Markov chain, we observed the large-scale grid simulation and counted the frequency of transitions between states over a simulated duration. Each probability of transition from state $i$ to state $j$, written as $s_i \rightarrow s_j$, was considered separately. The probability of transitioning between
any two states $s_i, s_j$, written as $p_{ij}$, was estimated by calculating the frequency of $s_i \rightarrow s_j$, or $f_{ij}$, divided by the sum of the frequencies of $s_i$ to all other states $s_k$ that $s_i$ could transition to, where $k$ ranges from 1..$n$ and $n$ is the number of states (7).

$$p_{ij} = \frac{f_{ij}}{\sum_{k=1}^{n} f_{ik}} \quad (1)$$

Repeating computation (1) for all $i, j = 1..n$ resulted in an $n \times n$ transition probability matrix (TPM) that succinctly summarized the dynamics of the grid system. Extensive simulation of the large-scale grid system revealed that system dynamics change over time, and for this reason we subdivided the simulated time duration into equal time periods and computed (1) for each period. This subdivision enabled the Markov model to capture changes in system behavior over time, or to be considered as piece-wise homogenous [Ro2004]. A time period duration of 2 hours, or 7200 s was chosen as the duration of a subdivided time period. Thus, a simulated 8-hour duration had 4 time periods, plus a fifth for clean-up operations. For each time period, (1) was used to compute a separate TPM. The weighted average of these five TPMs, or the summary matrix, is shown in Figure 3.2(a). Following this, we repeated these observations for the large-scale grid simulation over a 640 hour period, which resulted in 321 time period TPMs, summarized in figure 3.2(b).

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Wait</th>
<th>Disc</th>
<th>Ngt</th>
<th>Mon</th>
<th>Comp</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>0.9697</td>
<td>0</td>
<td>0.0303</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Waiting</td>
<td>0</td>
<td>0.8363</td>
<td>0.0673</td>
<td>0.0918</td>
<td>0</td>
<td>0</td>
<td>0.0046</td>
</tr>
<tr>
<td>Disc</td>
<td>0</td>
<td>0.0355</td>
<td>0.6714</td>
<td>0.2931</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ngt</td>
<td>0</td>
<td>0.4974</td>
<td>0.0182</td>
<td>0.2882</td>
<td>0.1961</td>
<td>0</td>
<td>0.0001</td>
</tr>
<tr>
<td>Mon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0003</td>
<td>0.9917</td>
<td>0.0080</td>
<td>0</td>
</tr>
<tr>
<td>Comp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fail</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Initial</th>
<th>Wait</th>
<th>Disc</th>
<th>Ngt</th>
<th>Mon</th>
<th>Comp</th>
<th>Fail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>0.9997</td>
<td>0</td>
<td>0.0003</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wait</td>
<td>0</td>
<td>0.7612</td>
<td>0.0460</td>
<td>0.1911</td>
<td>0</td>
<td>0</td>
<td>0.0017</td>
</tr>
<tr>
<td>Disc</td>
<td>0</td>
<td>0.0686</td>
<td>0.6084</td>
<td>0.3230</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ngt</td>
<td>0</td>
<td>0.2401</td>
<td>0.0062</td>
<td>0.2378</td>
<td>0.4801</td>
<td>0</td>
<td>0.0358</td>
</tr>
<tr>
<td>Mon</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.0007</td>
<td>0.9902</td>
<td>0.009</td>
<td>0</td>
</tr>
<tr>
<td>Compl</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
<td>0</td>
</tr>
<tr>
<td>Fail</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Figure 3.2 (a, b). Summary TPM for the grid computing system: (a) over 8-hour duration (plus a 2-hour clean-up period), and (b) over 640-hour duration in near steady-state conditions. Both summary TPMs were computed as weighted averages of 5 (a) and 321 (b) TPMs for equal time period divisions of 2 hours each (7200 s). To compute the summary TPMs, the individual $p_{ij}$ from the time period matrices are weighted by the relative number of transitions in their respective period.

A well-known use of Discrete Time Markov chain is to simulate change in a dynamic system over time in discrete time steps. To do this, the system state is represented as a vector $v$, where each element represents the proportion of tasks in one of the seven states. A discrete time step represents a fixed time duration, which in our experiments was chosen to be 85 s, or $h = 85$ (thus, a time period of duration $d_{\text{period}} = 7200$ s has $s = d_{\text{period}} / h$ or 85 time steps). To advance the system state one time step, a vector $v_m$, which represents the system state at time step $m$, is multiplied by the TPM $Q^p$ for the applicable time period $tp$ to produce a new system state $v_{m+1}$. This operation is represented by equation (2) below,

$$(Q^p)^T \cdot v_m = v_{m+1}, \text{ where } tp = \text{integral value } (m/S) + 1 \quad (2)$$

where $T$ indicates a matrix transpose. To perform this operation over a simulated duration consisting of many time steps, we start with state $v_0$, which represents an initial system state with a value of 1.0 for the Initial state and 0 for all others (e.g., all tasks are in the Initial state). Assuming $k$ time periods, equation (2) is
repeated for \( s \times k \) time steps until the end of the total simulated duration to produce the end state vector \( v(s \times k) \). The result of this process, which we refer to as a Markov simulation is shown in figure 3.3(a) for the 8-hour simulated duration and in figure 3.3(b) for 640 hour duration which approached steady-state conditions. In figure 3.3 (a, b), the progress of Tasks Completed and Tasks Failed is compared to the results in the large-scale simulation. As these figures show, the Markov simulations provide reasonable approximations of the large-scale simulations that they are models of.

![Figure 3.3 (a, b). Comparison between large-scale and Markov chain simulations of the change in Tasks Completed and Tasks Failed in grid computing system for: (a) 8-hour time duration (plus a 2-hour clean-up period) in which Markov simulation covered 421, 85 s time steps; and (b) 640-hour duration in which Markov simulation covered > 27000 time steps.](image)

### 3.2 The Markov Chain Model the Abilene System

The Abilene system Markov model is derived from a large-scale model of the Abilene network [Mi2009] that simulates the performance of a network using alternative congestion control algorithms. The large-scale model shows in detail how different congestion control algorithms are used in the transmission of data flows from the time the flows are submitted to the network to the time they either complete or fail. The procedures used in deriving the Abilene system Markov model were the same as those described in section 3.1 for the grid system Markov model. Figure 3.4 shows the Abilene system Markov model as a state diagram for a single task. This state model describes how a single flow may progress through different congestion control regimes. This state model consists of 8 states. As in the grid system, prior to submission, flows reside in an Initial state. Flow submission results in entering a Connecting state, during which a source to sink connection is established. Once connected, flows enter an Initial Slow Start (ISS) state from which they may either complete or enter states representing three additional congestion control regimes: Normal Congestion Control (NCC), Alternate Congestion Control (ACA), and Slow Start (SS). Flows may re-enter any of these three states according to criteria described in [Mi2009] until they complete (or fail). Like the grid system model, the Abilene system Markov model is an absorbing chain.
To obtain probabilities of transition, the procedure described in section 3.1 was repeated using a large-scale simulation of the Abilene system which, in this case, executed for a simulated 1500 s. Again, the Markov model was made to be piece-wise homogenous by subdividing the duration into 5 equal time periods of 300 s each. Here, a much smaller time step of 0.05 s was chosen. The resulting summary TPM is shown in figure 3.5. The Markov simulation of the Abilene system and its comparison with the related large-scale simulation is shown in figure 3.6. Results show that the Markov chain simulation provides a very close approximation of the large-scale simulation.

Figure 3.5. Summary stochastic TPM for Abilene network Markov chain. This TPM is a weighted average of 5 TPMs for equal time period divisions of 300 s duration. Individual $p_{ij}$ from the five periods are weighted as described for figure 3.2 (a).

Figure 3.6 shows that, as before, the Markov simulation is able to closely approximate the large-scale simulation.
3.3 The Perturbation Algorithm for Predicting Performance Degradations

Our work in [Da2009a] demonstrates that a suitably perturbed Markov chain model can replicate (with good agreement) specific scenarios in the large-scale grid computing system simulation in which performance degrades significantly. However, we also found that it is difficult to identify a set of state transitions and their respective perturbations that capture such a scenario. We found that some form of search must be undertaken of a large space of possible perturbations in order to find the state transitions and perturbations that represent scenarios in which system performance degrades.

Therefore we developed a perturbation algorithm for this purpose. The algorithm constitutes a limited, brute-force search that is restricted in order to conserve resources while exploring a reasonable range of alternatives. The algorithm predicts approximate changes in system performance that occur as specific state transition probabilities are gradually altered. The output of the algorithm is a set of Markov chain simulation results that identify, or predict, situations where system performance degrades in response to changes to a specific set of transition probabilities. These predictions can be tested by comparing them with large-scale grid simulations.

The algorithm permits simultaneous perturbation of combinations of two rows in a TPM for a Markov chain in order to capture situations where inter-row, i.e., inter-state, dependencies exist. The algorithm proceeds by incrementally raising and lowering all feasible combinations of non-zero state transitions in these rows. To begin, a user must first select a state to perturb, which is represented by the primary row, \( r \). Each row element, or column, in this row with a probability of transition greater than zero is selected in turn for incremental increase and designated as a primary increase column \( c^1 \). The primary increase column corresponds to a state being transitioned into from \( r \), whose transition probability will be raised. To offset this increase, a row element corresponding to a different state is selected as a primary sink column, \( c^1 \), which is decreased by a portion of the increase to \( c^1 \), where the portion is determined by the weight \( w \leq 1 \). The remainder of the increase to \( c^1 \) is offset by decreasing remaining elements of \( r \) with transition probabilities greater than 0 by amounts that are proportional to their non-zero values. In this way the changes \( c^1 \), \( c^1 \), and the non-zero elements of row \( r \) are made to ensure that the individual transition probabilities of all elements in the \( r \) still sum to 1.

The second row to be perturbed, or secondary row \( s \), is determined on the basis of which \( c^1 \) has been selected. The procedure for perturbing the secondary row \( s \) is simpler than the procedure for the primary row, and the secondary row elements are perturbed by larger amounts. In \( s \), each row element with a value greater than 0 is in turn designated as a secondary increase column, \( d^1 \). The row element \( d^1 \) is raised by the amount \( m^s \) to a predetermined maximum, as described further just below. In the perturbation of the secondary row, the decrease is distributed proportionally to all other row elements with non-zero transition probabilities. A set of values \( \{r, c^1, c^1, w, s, d^1, m^s\} \) is considered a perturbation combination, which represents a set of state transition probabilities to be altered in order to explore alternative execution paths in the Markov simulation.

To investigate the various perturbation combinations, the user also selects the perturbation limit \( L \) to limit how far transition probabilities can be perturbed (a separate \( L_r \) and \( L_s \) may be chosen for \( r \) and \( s \), if desired). The user must also select the incremental amounts, \( v_r \) and \( v_s \), by which \( c^1 \) and \( d^1 \) are raised respectively. These decisions define the extent and granularity of the perturbation (note, \( v_r \) is used to determine \( m^s \) mentioned above). The algorithm proceeds by enumerating all feasible perturbation combinations. For each combination, an iteration is performed to raise \( c^1 \) and \( d^1 \) by the designated increments (and correspondingly lower the other elements) across all time-period matrices until \( L \) is reached in each time period matrix. The Markov simulation is executed for each incremental perturbation of each perturbation combination, and the result is recorded. This set of results can then be examined to identify those perturbation combinations where systematic changes to transition probabilities lead to performance degradations.

Figures 3.7 (a) and (b) illustrate an example of one such drastic performance degradation predicted by the perturbation algorithm in both the 8-hour and 640-hour cases. These figures show the impact of perturbing a
single combination (blue curves) in which lowering the probability of transition to 0 for Negotiating $\rightarrow$ Monitoring causes the proportion of Tasks Completed to also fall 0. In both the 8- and 640-hour cases, the Markov simulation predictions are borne out when the large-scale simulation, which we use as a real-world proxy, is also altered to behave aberrantly (red curve) so that negotiations that would otherwise succeed instead fail and task execution is prevented.

![Figure 3.7](image1.png)

Figure 3.7 (a,b). Perturbation of Negotiating State ($r=4$) in grid system Markov chain model to predict effect of reducing probability of transition from Negotiating $\rightarrow$ Monitoring in (a) the 8-hour case and (b) the 640-hour case for the grid computing system. Proportion of tasks complete in the large-scale (red curve) and Markov chain simulations (blue curves) is shown for (a) and (b). The probability of transition from Negotiating to Waiting is raised ($c^f = Waiting$) and probability of transition from Negotiating to Monitoring lowered ($c^f = Monitoring$, $w = 0.8$). The secondary perturbation row is $s = Waiting$. For (a) $L_c=0.5$ and $v_i = 0.01$; for (b) $L_c=1$ and $v_i = 0.01$. In both cases $L_s=0.25$ and $v_s = 0.0625$.

Although Figures 3.7 (a) and (b) show that the decrease in Tasks Completed is a straightforward consequence of this perturbation, there is a less obvious persistence in the high rate of Tasks Completed as the probability of transition from Negotiating $\rightarrow$ Monitoring is steadily decreased. In the large-scale simulation, both figures show that the rate of Tasks Completed remains relatively high until the decrease in probability of transition nears 0; then the rate of Tasks Completed declines sharply. This pattern is in fact predicted by the Markov chain simulation.

![Figure 3.8](image2.png)

Figure 3.8. Perturbation of Connecting State ($r=2$) in the Abilene Markov chain model to predict effect of the reducing probability of transition to 0 for Connecting $\rightarrow$ Initial Slow Start ($c^f = 3$, $w = 1$), while raising probability of Connecting self-transition ($c^f = 2$). The affect of this perturbation on proportion of flows complete is shown. No Secondary perturbation row, $s=2$. Note: the probability of Connecting self-transition is increased to 1, making this a trap state situation as well.
Draft Document

For the Abilene network system, figure 3.8 shows a similar trend in the proportion of Flows Completed when the probability of transition from Connecting \(\rightarrow\) Initial Slow Start is reduced to 0. Here, the perturbation also causes Flows Completed to decline to 0. This scenario describes a somewhat obvious real-world situation where flows are unable to connect. As in the case of the grid system, the proportion of flows complete remains relatively high until the probability of transition from Connecting \(\rightarrow\) Initial Slow Start approaches 0, then it drops sharply.

3.4 Efficiency of the Perturbation Algorithm

Using the perturbation algorithm, a complete perturbation of one matrix row \(r\) would require examining \(O(\alpha b^2 (b-1))\) perturbation combinations. In this formula, \(b\) is a constant that indicates the expected number of \(c^i\) (or \(d^i\)) in the row being perturbed, or the branching factor, and \(\alpha\) is the number of weight values that the primary sink element may take on. If \(L/v_s\) is the number of incremental increases for the primary row \(r\) and \(L/v_s\) is the number increases for row \(s\), then the number of executions, \(E\), of the Markov simulation needed to explore all perturbation combinations for one row is

\[
E = O \left( L/v_r \ast (L/v_s + 1) \ast \alpha(a^3 - b^2) \right),
\]

if one assumes that that there is no possibility of self-transition, so that \(c^i \neq s\) is always true. However, if \(c^i = s\), then \(E\) is slightly smaller:

\[
E = O \left( L/v_r \ast (L/v_s + 1) \ast \alpha(b^3 - 2b^2 - b) \right).
\]

The complete exploration of a TPM having a dimension \(d\) is thus \(O(d \ast E)\), which grows polynomially with respect to the branching factor but linearly with respect to matrix size, or the number of states. In practice, neither figure will be completely accurate since the branching factor is not constant for all states.

The complete exploration of the grid system Markov chain in the 8-hour scenario required approximately 56 minutes, while the 640-hour scenario required 15.4 hours. This, however, is favorable in comparison with the running time of the large simulation which took 1 week and 5.2 weeks respectively. For the 8-hour scenario, large-scale simulation needed about two orders of magnitude more time; in the 640-hour case, large-scale simulation required 1.5 orders of magnitude more time, though it also included a number of extra runs to test extreme conditions (discussed further in section 5). For the Abilene system problem, execution of the perturbation algorithm to completely perturb rows 2-6 of the related Markov chain TPMs required 27.3 hours, an effort that involved 330 perturbation combinations and almost 100,000 executions of the Markov simulation. Still, this is an impressive improvement over the Abilene system large-scale simulation where a single execution required 7.3 hours! In section 5, we provide a complete summary of the results of these simulations and their predictions of performance degradation in the systems they model.

Despite these significant gains in comparative efficiency, it is clear that the running time of the Markov simulation would be very substantial for significantly larger matrices than those discussed above. Further, it would impose on the analyst (whether automated or human) quite a burden in analyzing large amounts of output to identify situations that predict performance degradations and other behavioral anomalies. Thus we seek more efficient methods to identify areas of the Markov chain TPM where perturbation leads to performance degradation. This is the subject of the subsequent sections in this paper, in which we describe our work on two methods for this purpose. In sections 4 and 5, we show how generation of minimal s-t cut sets that disconnect all paths between an Initial state and a desired absorbing state can be used to identify critical state transitions, which if perturbed, can cause severe performance degradations. We show that generation of such minimal s-t cut sets finds all areas of the Markov chain problems described above where perturbation causes the proportion of tasks to fall to 0, but at a much smaller cost than the perturbation...
algorithm. We then present preliminary results indicating that this approach can be effective for larger, more complex Markov chains and their TPMs. In section 6, we show how the Spectral Expansion Theorem can be used as an alternative to Markov simulations described earlier by the evaluation of analytical formulae. Furthermore, we show that changes in the leading eigenvector of the transient part of the TPM correlates reasonably well with performance changes due to state transition perturbations. While not perfect, we claim these changes can be used as to distinguish critical perturbations (i.e. those that lead to significant performance degradation) from non-critical.
4. Identifying Critical System Execution Paths and Minimal Cut Set Analysis

In this section, we describe a method based on graph theory concepts for identifying minimal cut sets between an Initial state and desired absorbing state that consist of critical state transitions, which if perturbed, are likely to lead to system performance degradations. In section 5, we see that this method is capable of finding the same areas of performance degradation as the perturbation algorithm described above, but at a small fraction of the computational cost.

4.1 Finding State Transition that Disconnect Paths to Absorbing States

In basic graph theory, a graph $G(V, E)$ consists of a set of vertices $V$ connected by edges from the set $E$. A path is a sequence of edges that connects two vertices in a graph. It is easy to see that a Markov chain can be represented as a directed graph, in which the set of vertices $V$ corresponds to the set of states, while the set of directed edges $E$ correspond to transitions between states that can occur in only one direction. A path is then a sequence of transitions that lead from one state to another. For our purposes, the most important paths are those which lead from the Initial state to an absorbing state.

![Diagram of state transitions](image)

Figure 4.1. Two unique non-cyclic paths (numbered and denoted by heavy arrows) from the Initial State to Tasks Completed state for grid computing system. Three single-transition s-t cuts appear as heavy bars over transitions. Trap states are denoted by $T$.

In the Markov chain models for the grid system and the Abilene system, the paths of interest are those that lead from the Initial state to one of the two absorbing states: Tasks Completed or Tasks Failed. For the remainder of this analysis we will consider only the Tasks Completed absorbing state. To render the analysis tractable, we consider only paths that are non-cyclic. For example, figure 4.1 shows two paths through grid system Markov chain from the Initial to the Tasks Completed State. For a small Markov chain, a well known algorithm for finding shortest non-cyclic paths between two vertices in a graph, such as given in [Ev1979], can be modified to do a breadth-first search and enumerate all paths between the Initial and Tasks Completed state (we will return to the question of the tractability of this computation in a moment). By finding one or more state transitions that are common to all paths from the Initial state to the Tasks Completed state, it is possible to disconnect, or block, all paths to the Tasks Completed state by removing these common transitions—a condition which could obviously adversely affect system performance. These common transitions identify areas of the Markov chain that are sensitive to perturbation. By reducing the transition probability values of these common transitions to 0, the flow of tasks to the Tasks Completed state is also reduced to 0.
Draft Document

In discrete mathematics, a set of one or more edges, which if removed, disconnects all paths between two vertices $s$ and $t$ is often referred to as an $s$-$t$ cut set, as for example [Ts1980]. An $s$-$t$ cut set is defined to be a minimal $s$-$t$ cut set if removal of any edge from the cut set causes $s$ and $t$ to be reconnected. In Figure 4.1, the state transition Initial $\rightarrow$ Discovering, by itself, constitutes a minimal $s$-$t$ cut set consisting of one edge. This transition is common to all paths from the Initial state ($s$) to the Tasks Completed state ($t$). If the transition is removed, all paths to the Tasks Completed state would be disconnected. In this paper, minimal $s$-$t$ cut sets with a single member will be referred to as single-transition $s$-$t$ cuts and are an important special case of interest, as illustrated further below. We will return to the topic of minimal $s$-$t$ cut sets consisting of multiple transitions shortly.

4.2 Using Single-Transition $s$-$t$ Cuts to Analyze Markov Chain Models

In figure 4.1, there are three single-transition $s$-$t$ cuts: Initial $\rightarrow$ Discovering, Negotiating $\rightarrow$ Monitoring, and Monitoring $\rightarrow$ Tasks Completed. Figure 3.7 (a, b) shows graphically the result of reducing the probability of transition for Negotiating $\rightarrow$ Monitoring to 0 in both the 8-hour and 640-hour cases. When this occurs, the proportion of tasks that reaches the Tasks Completed state drops to 0 in both cases. The same result occurs when the other two transitions identified as single-transition $s$-$t$ cuts, Initial $\rightarrow$ Discovering and Monitoring $\rightarrow$ Tasks Completed, are similarly perturbed, as discussed further in [Da2009b]. As will be described in section 5, an exhaustive application of the perturbation algorithm to the Waiting, Discovering, Negotiating, and Monitoring states corroborated that these three single-transition $s$-$t$ cuts are the only state transitions, which if individually reduced to 0, also cause the proportion of tasks reaching the Tasks Completed state to fall to 0.

In the Abilene system, there are two single-transition $s$-$t$ cuts: Initial $\rightarrow$ Connecting and Connecting $\rightarrow$ Initial Slow Start, shown in Figure 4.2. Figure 3.8 shows the result of reducing the probability of transition for Connecting $\rightarrow$ Initial Slow Start to 0, which results in the proportion of flows reaching the Flows Completed state to fall to 0. Again, Section 5 presents results to corroborate that these single-transition $s$-$t$ cuts are the only state transitions, which if individually reduced to 0, also cause the proportion of flows completed to fall to 0.

Figure 4.2. Unique non-cyclic paths (denoted by heavy arrows) from the Initial to Tasks Completed states for the Abilene network system. There are 16 possible paths, of which 4 are shown. Two single-transition $s$-$t$ cuts appear as heavy bars over transitions. Trap states are denoted by $T$. 

21
Thus for both the grid system and the Abilene System, it was possible to use a simple algorithm for enumerating all paths derived from [Ev1979] to identify state transitions that were highly sensitive to perturbation and that would have a dramatic effect on system performance. Excluding transitions out of the Initial state, the single-transition s-t cuts correspond to obvious critical real-world processes in both the grid and Abilene systems. In the grid system, the single-transition s-t cuts for Negotiating \( \rightarrow \) Monitoring and Monitoring \( \rightarrow \) Tasks Completed are clearly related to critical steps in the process of allocating resources to, and executing, tasks. More trivially in the Abilene network, the Connecting \( \rightarrow \) Initial Slow Start single-transition s-t cut represents the obvious consequences of failing to establish a connection. We next proceed to the related topic of trap states, which like single-transition s-t cuts, lie on all paths between the Initial state and absorbing state, and can have similar impacts of system performance in Markov chains.

4.3 Identifying Trap States as Potential Sources of Drastic Performance Degradation

In the discussion of the perturbation algorithm, reducing probabilities of transition out of one state requires raising the sum of one or more probabilities of transition from that state to different states by an equal amount. The \( \text{“to states”} \) state(s) which will receive increased probabilities of transition may be different from the \( \text{“from state”} \) or may be the same. In the latter case, when the \( \text{“to”} \) and \( \text{“from”} \) states are the same, the process merely remains in the same state, or transitions back to itself--which we refer to as a self-transition\(^1\). If a self-transition probability is raised so that it approaches 1, or even equal 1, the process remains in the \( \text{“from state”} \) for a prolonged time. In this case, the \( \text{“from state”} \) effectively becomes a trap state for processes. A trap state is distinguished from a permanent absorbing state, such as Tasks Completed, because self-transition probability of the trap states may vary.

\[ \text{Figure 4.3 (a, b). Perturbation of Discovering State (r= Discovering) to predict effect of increasing probability of self-transition Discovering } \rightarrow \text{ Discovering (c}^1 = \text{ Discovering) in (a) the 8-hour case and (b) the 640-hour case for the grid system model. Proportion of tasks complete in the large-scale and Markov chain simulations shown for (a) and (b). Probability of transition from Discovering to all other states lowered to 0. Secondary perturbation row, s=Monitoring (result is the same for all secondary perturbations). For (a) } L_r=0.5, \, v_i=0.01, \, L_s=0.25 , \text{ and } v_s = 0.0625; \text{ for (b) } L_r=0.75, \, v_i=0.01, \, L_s=0.4 \text{ and } v_s = 0.2. \]

An example of such a trap state and its impact on system performance for the grid system is shown in figure 4.3 when the Discovering state is made a trap state in the 8-hour and 640-hour cases. Tasks never leave the Discovering state, so that they cannot proceed to other states and finish. The perturbation described for the Abilene system in figure 3.8 can also be accomplished by raising the Connecting self-transition to 1.

\(^1\) Self transition probabilities are determined using (1) by observing processes that remain in a state longer than one time step (85 s in the grid system case and 0.05 s in the Abilene case. That is, \( s_i \rightarrow s_i \), or \( f_{ij} \) is tabulated for processes whose duration in a state exceeds a time step.
Making the Connecting state a trap state prevents flows from reaching the data transmission phase (in which they enter congestion control states) and then finishing. Trap states may be easily identified as being states that are common to all paths between an Initial state and an absorbing state. Hence, their removal also effectively disconnects all paths between the vertex $s$, Initial state, and the vertex $t$, the absorbing state (Tasks Complete in the grid model or Flows Complete in the Abilene model). In the grid system model, the trap states are Initial, Discovering, Negotiating, and Monitoring, shown in Figure 4.1; for the Abilene model, these are Initial, Connecting, and Initial Slow Start, shown in Figure 4.2.

As Section 5 will show in more detail, the use of path enumeration to find single-transition s-t cuts and trap states achieves the same goal as our original perturbation algorithm [Da2009a] but at less cost. However, the approach is insufficient for analysis of substantially larger Markov chains, as we describe in Section 6.

### 4.4 Minimal s-t Cut Sets With Multiple Transitions

A minimal s-t cut set between the Initial and absorbing states that consists of more than one state transition will be referred to in this paper as a multiple-transition s-t cut. In such a minimal cut set on a Markov chain graph, it is necessary to lower probabilities of transitions to 0 for all state transitions that are members of the cut set in order to radically affect system performance. Figure 4.4 shows two multiple-transition s-t cuts for the Grid computing system Markov chain.

![Figure 4.4](image.png)

Figure 4.4. Two multiple-transition s-t cuts for the grid computing system: (a) Discovering → Negotiating and Discovering → Waiting; and (b) Discovering → Negotiating and Waiting → Negotiating.

In the Abilene system Markov chain, there are 8 possible multiple-transition s-t cuts listed in Table 4.1, one on which is shown as an example in Figure 4.5. As in Figures 4.1 and 4.2 for single-transition s-t cuts, it is easy to see that the two multiple-transition, minimal s-t cut sets in Figure 4.4 disconnect the Initial from the Tasks Completed state, while in Figure 4.5, the sample multiple-transition minimal s-t cut set also disconnects the Initial from the Flows Completed state.
Table 4.1. Complete list of 10 minimal s-t cut sets for Abilene system Markov chain produced by node contraction algorithm. This list was computed in less than 1 second.

<table>
<thead>
<tr>
<th>Number from states/ transitions</th>
<th>List of transitions (Ab)</th>
<th>Sum of transition probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Init → Conn</td>
<td>0.000167</td>
</tr>
<tr>
<td>2</td>
<td>Conn → ISS</td>
<td>0.182</td>
</tr>
<tr>
<td>3</td>
<td>ISS → NCA, ISS → ACA, ISS → SS, ISS → Cmp</td>
<td>0.112</td>
</tr>
<tr>
<td>4</td>
<td>ISS → Cmp, NCA → Cmp, ACA → Cmp, SS → Cmp</td>
<td>0.311</td>
</tr>
<tr>
<td>5</td>
<td>ISS → ACA, ISS → SS, ISS → Cmp, NCA → ACA, NCA → SS, NCA → Cmp</td>
<td>0.119</td>
</tr>
<tr>
<td>6</td>
<td>ISS → NCA, ISS → SS, ISS → Cmp, ACA → NCA, ACA → SS, ACA → Cmp</td>
<td>0.263</td>
</tr>
<tr>
<td>7</td>
<td>ISS → NCA, ISS → ACA, ISS → Cmp, SS → NCA, SS → ACA, SS → Cmp</td>
<td>0.316</td>
</tr>
<tr>
<td>8</td>
<td>ISS → ACA, ISS → Cmp, NCA → ACA, NCA → Cmp, SS → ACA, SS → Cmp</td>
<td>0.207</td>
</tr>
<tr>
<td>9</td>
<td>ISS → SS, ISS → Cmp, NCA → SS, NCA → Cmp, ACA → SS, ACA → Cmp</td>
<td>0.263</td>
</tr>
<tr>
<td>10</td>
<td>ISS → NCA, ISS → Cmp, ACA → NCA, ACA → Cmp, SS → NCA, SS → Cmp</td>
<td>0.430</td>
</tr>
</tbody>
</table>

Figure 4.5. Example of multiple-transition, minimal s-t cut set for the Abilene network system

The concept of an s-t cut set can be extended to include vertices whose removal disconnects all paths from s to t. Such a set of elements (edges and vertices) is sometimes known as an s-t separating set; if this set is minimal, then it is a minimal s-t separating set [Ha1999]. In the context of a discrete time Markov chain, vertices, whose removal results in disconnection of all paths to a desired absorbing state, correspond to trap
states whose self-transition probability approaches 1. In the grid system Markov model, the trap states Initial, Discovering, Negotiating, and Monitoring are actually minimal s-t separating sets with a single vertex. In the Abilene network model, Initial, Connecting, and Initial Slow Start also fall into this category. In the grid system Markov chain graph, there is one combination of edges and vertices that disconnect all paths from the Initial to the absorbing states, e.g. state transitions and states whose probabilities of self-transition go to 1: the state transition from Discovering $\rightarrow$ Negotiating and the state, Waiting. In the Abilene system model, 47 such combinations were found.

It is likely that multiple-transition s-t cuts and multiple-transition separating sets are more common in larger Markov chain problems. They cannot be easily found using simple approaches such as path enumeration and require more powerful algorithms. In section 6, we limit ourselves to discussion of the methods for finding multiple-transition s-t cuts to an algorithm that is appropriate for large Markov chains. But first, in Section 5, we verify that minimal s-t cut sets, both single- and multiple-transition, can be used to identify areas in a Markov chain where perturbations lead to performance degradations.
5. Results of Minimal Cut Set Analysis

This section shows that identification of minimal s-t cut sets, including single-transition and multiple-transition s-t cuts, can be used to predict which state transitions, if perturbed, are most likely to adversely impact system performance. We first verify this conclusion by comparing minimal s-t cut sets for paths between the Initial and Tasks Completed states that were found in the Grid system Markov chain model against both the results produced by the perturbation algorithm and the results of the large-scale Grid simulations. We then verify this conclusion by comparing minimal s-t cut sets for paths between the Initial and Flows Completed states discovered in the Abilene system Markov chain model against the results produced by the perturbation algorithm. The conclusions are verified showing the correspondence between specific minimal s-t cut sets and state transitions that, if perturbed, cause dramatic declines in system performance. We assess the potential savings in computation time provided by this new method in analyzing both the Grid and Abilene systems. For both the grid system and Abilene problems, minimal s-t cut sets could be found either through the adapted path enumeration algorithm [Ev1979] described earlier or through the node contraction algorithm, which is described in the next sections.

In both cases, the perturbation algorithm is applied only to the primary row; no secondary row perturbation is used. This permits better focus on the perturbation algorithm results that involve states and transitions of interest. Where appropriate, accentuating or mitigating effects of secondary row perturbation are discussed (see [Da2009a] for the full results).

5.1 Grid System

In this section, we verify that all minimal s-t cut sets for paths between the Initial and Tasks Completed states in the Grid system Markov chain correspond to state transitions, which if suitably perturbed using the perturbation algorithm described in section 3, can adversely impact system performance. Table 5.1 shows the results of the application of the perturbation algorithm described in section 3.1 to the Grid system Markov model for the 8-hour Grid system case, while table 5.2 shows the same for the 640-hour case. Specifically, the tables show the results of perturbing rows, \( r \), corresponding to the states Waiting, Discovering, Negotiating, and Monitoring, by raising the probability of transition from the states designated as \( r \) to states that correspond to primary increase columns \( c^I \), while lowering the probability of transition from, \( r \), to other states that are designated as sink columns, \( c^I \). Both tables show that in all cases where application of the perturbation algorithm causes declines in the percentage of Tasks Completed that approach 100% (i.e., the proportion of tasks completed approaches 0\(^2\)), a correspondence can be drawn to the existence of a single-transition s-t cut. Perturbation algorithm results that were verified by the large-scale grid simulation are in shaded cells.

5.1.1 Correspondence of Single-transition s-t Cuts to Perturbation Algorithm Results

We first consider how well single-transition s-t cuts for paths between the Initial and Tasks Completed states in Figure 4.1 correspond to results produced by the perturbation algorithm for the grid system. The first case occurs when \( r = Negotiating \). Here, Tables 5.1 (c) and 5.2 (c) show that designating Monitoring as the sink column \( c^I \), i.e., lowering the probability of Negotiating \( \rightarrow \) Monitoring to 0, resulted in tasks completed approaching 0, which was verified by the large-scale simulation. This result occurs regardless of whether the state transition probability for Waiting, Discovering, or Negotiating is raised, i.e., made the primary increase

---

2 Note that the perturbation algorithm is designed with built-in tolerances by which perturbed values approach, but do not reach, limits of 0 or 1 within a specific number of significant digits. Hence, the actual proportion of Tasks Complete also approaches, but does not equal, 0
column, \( c^\dagger \). In this case, figure 4.1 shows that the state transition Negotiating → Monitoring is a single-transition \( s-t \) cut.

Table 5.1. Correspondence between cases where perturbation algorithm results in the proportion of Tasks Completed approaching 0 and the existence of single-transition \( s-t \) cuts for 8-hour grid system simulation. The table shows the proportion of tasks completed and percent change when perturbation algorithm is applied to rows, \( r \), of the TPM in figure 3.2 (a) to decrease the probability of transition to 0 for the sink column, \( c^\dagger \), or state transition \( r \rightarrow c^\dagger \), while increasing the probability of transition in the primary increase column, \( c^\checkmark \), or \( r \rightarrow c^\checkmark \) at sink weight = 1. Secondary perturbation excluded. Right-most column indicates if single-transition \( s-t \) cut exists for \( r \rightarrow c^\dagger \) in Figure 4.1. In all cases where perturbation causes proportion of Tasks Complete to approach 0, a positive correspondence exists with a single-transition \( s-t \) cut. Shaded cells represent perturbations where decrease in tasks completed was verified by large-scale simulation. Note: trap states are excluded and discussed separately below.

\[
\begin{array}{|c|c|c|c|c|c|c|}
\hline
(a) & \text{Sink column \( c^\dagger \)} & \text{Primary increase column } c^\checkmark \text{)} & \text{Prop. Tasks Complete and } \% \text{ change when prob. } (r \rightarrow c^\dagger) \rightarrow 0 & \text{Single-transition } s-t \text{ cut exists} & (b) & \text{Sink column \( c^\dagger \)} & \text{Primary increase column } c^\checkmark \text{) } & \text{Prop. Tasks Complete and } \% \text{ change when prob. } (r \rightarrow c^\dagger) \rightarrow 0 & \text{Single-transition } s-t \text{ cut exists} \\
\hline
\text{Waiting} & \text{Discovering} & 0.956 & +0.10 & \text{No} & \text{Waiting} & \text{Discovering} & 0.973 & +1.18 & \text{No} \\
\text{Waiting} & \text{Negotiating} & 0.962 & +0.73 & \text{No} & \text{Waiting} & \text{Waiting} & 0.982 & +2.83 & \text{No} \\
\text{Discovering} & \text{Discovering} & 0.910 & -0.71 & \text{No} & \text{Discovering} & \text{Waiting} & 0.925 & -3.14 & \text{No} \\
\text{Discovering} & \text{Negotiating} & 0.962 & +0.73 & \text{No} & \text{Discovering} & \text{Negotiating} & 0.981 & +2.72 & \text{No} \\
\text{Negotiating} & \text{Waiting} & 0.906 & -5.13 & \text{No} & \text{Negotiating} & \text{Discovering} & 0.742 & -22.30 & \text{No} \\
\text{Negotiating} & \text{Discovering} & 0.633 & -33.72 & \text{No} & \text{Negotiating} & \text{Negotiating} & 0.973 & +1.88 & \text{No} \\
\hline
(c) & \text{Sink column \( c^\dagger \)} & \text{Primary increase column } c^\checkmark \text{)} & \text{Prop. Tasks Complete and } \% \text{ change when prob. } (r \rightarrow c^\checkmark) \rightarrow 0 & \text{Single-transition } s-t \text{ cut exists} & (d) & \text{Sink column \( c^\dagger \)} & \text{Primary increase column } c^\checkmark \text{) } & \text{Prop. Tasks Complete and } \% \text{ change when prob. } (r \rightarrow c^\checkmark) \rightarrow 0 & \text{Single-transition } s-t \text{ cut exists} \\
\hline
\text{Waiting} & \text{Discovering} & 0.966 & +1.15 & \text{No} & \text{Waiting} & \text{Discovering} & 0.996 & +2.20 & \text{No} \\
\text{Waiting} & \text{Negotiating} & 0.983 & +2.93 & \text{No} & \text{Waiting} & \text{Negotiating} & 0.984 & +3.04 & \text{No} \\
\text{Waiting} & \text{Monitoring} & 0.998 & +4.50 & \text{No} & \text{Waiting} & \text{Monitoring} & 0.997 & +2.51 & \text{No} \\
\text{Discovering} & \text{Discovering} & 0.950 & -0.52 & \text{No} & \text{Discovering} & \text{Discovering} & 0.976 & +2.20 & \text{No} \\
\text{Discovering} & \text{Negotiating} & 0.956 & +0.10 & \text{No} & \text{Discovering} & \text{Negotiating} & 0.976 & +2.20 & \text{No} \\
\text{Discovering} & \text{Monitoring} & 0.976 & +2.20 & \text{No} & \text{Discovering} & \text{Monitoring} & 0.976 & +2.20 & \text{No} \\
\text{Negotiating} & \text{Waiting} & 0.870 & -8.90 & \text{No} & \text{Negotiating} & \text{Tasks Comp} & 0.010 & -98.95 & \text{Yes}^b \\
\text{Negotiating} & \text{Discovering} & 0.931 & -2.51 & \text{No} & \text{Negotiating} & \text{Tasks Comp} & 0.018 & -98.12 & \text{Yes}^b \\
\text{Negotiating} & \text{Monitoring} & 0.997 & +4.40 & \text{No} & \text{Negotiating} & \text{Monitoring} & 0.997 & +2.51 & \text{No} \\
\text{Monitoring} & \text{Waiting} & 0.014 & -98.53 & \text{Yes}^a & \text{Monitoring} & \text{Initial} & 0.0 & -100.00 & \text{Yes}^c \\
\text{Monitoring} & \text{Discovering} & 0.022 & -97.70 & \text{Yes}^a & \text{Monitoring} & \text{Monitoring} & 0.097 & +1.57 & \text{No} \\
\text{Monitoring} & \text{Negotiating} & 0.030 & -96.86 & \text{Yes}^a & \hline
\end{array}
\]

Similarly, when \( r = \text{Monitoring} \), Tables 5.1 (d) and 5.2 (d) show that designating Tasks Completed as the sink column \( c^\dagger \), i.e., lowering the probability of Monitoring → Tasks Completed to 0, creates the obvious result where the proportion of Tasks Complete also approaches 0 (a percentage decline that approaches 100%). Large-scale simulation verified that this result occurs regardless of what state transition probability is raised, i.e., made the primary increase column, \( c^\checkmark \). Again, figure 4.1 shows that the state transition Monitoring → Tasks Completed is a single-transition \( s-t \) cut. Note also that lowering the probability of Monitoring self-transition (i.e., designating it the sink column \( c^\dagger \)), while raising the transition Monitoring → Negotiating, also resulted in tasks completed approaching 0. This happens because the probability of transition for Monitoring → Tasks Completed is initially very low (0.008 for the 8-hour case and 0.009 for the 640-hour case. See figure 3.2(a) and (b)). Thus, the probability of Monitoring self-transition must be very high (over 0.99 in figure 3.2 (a) and (b)) to ensure all tasks remain in the Monitoring state long enough to have an opportunity to transition to Tasks Completed. Thus, a perturbation to reduce the probability of Monitoring self-transition to 0 has the effect of preventing tasks from transitioning to Tasks Completed—and is equivalent to a reduction of the probability of transition for Monitoring → Tasks Completed.
Table 5.2. Correspondence between cases where perturbation algorithm results in tasks completed falling to 0 and existence of single-transition s-t cuts in 640-hour Grid system simulation. The table shows the proportion of tasks completed and percent change when perturbation algorithm is applied to rows, r, of the TPM in figure 3.2 (b) to decrease the probability of transition to 0 for the sink column, c^↓, or state transition r → c^↓, while increasing the probability of transition in the primary increase column, c^↑, or r → c^↑ at sink weight = 1. Secondary perturbation excluded. Right-most column indicates if single-transition s-t cut exists for r → c^↓ in Figure 4.1. In all cases where perturbation causes proportion of Tasks Complete to approach 0, a positive correspondence exists with a single-transition s-t cut. Shaded cells represent perturbations where decrease in tasks completed was verified by large-scale simulation. Note: trap states are excluded and discussed separately below.

Finally, there were situations where the proportion of Tasks Complete dropped significantly, but not near to 0. For example, this occurred in Tables 5.1 (b) and 5.2 (b) when Waiting was made the primary increase column, c^↑, while Negotiating was made the sink column, c^↓. Here the combined probabilities of transition of matrix elements c^↓ and c^↑ approached, but did, not equal 1. Hence raising c^↑ to the limit while lowering c^↓ to 0 created conditions in which the flow of tasks to the Tasks Completed state was constrained but not stopped. These correspond to the state transitions of the 2 multiple transition s-t cuts in Fig. 4.4, which were identified by node contraction, but not by the perturbation algorithm.

5.1.2 Correspondence of Trap States to Perturbation Algorithm Results

With regard to trap states in the grid system, Table 5.3 shows the results of the Markov and large-scale simulations for perturbations of self-transitions of individual states common to all paths between the Initial and Tasks Completed states in the Grid 8-hour and 640-hour cases. This table shows the effect on tasks completed of raising the self-transition probability to 1 for four states; with the Initial state again omitted. These results confirm that as the self-transition probability of the three trap states, Discovering (mentioned above), Negotiating, and Monitoring, reaches 1, Tasks Completed approaches 0. The fourth state, Waiting, is not a trap state; but it is one of only two paths to the Completed state. Hence if the self-transition probability of Waiting is raised to 1, there is a partial downward effect on tasks completed, which also impacts the results in Tables 5.1 and 5.2, mentioned above. In summary, the data in Table 5.1, 5.2, and 5.3 shows that all single-transition s-t cuts and trap states identified in figure 4.1 correspond to cases where the perturbation algorithm
found perturbation combinations that caused Tasks Completed to approach 0. For all other perturbation combinations shown in Tables 5.1 and 5.2, tasks completed remained relatively stable (i.e., did not decline significantly), and none of these combinations corresponded to a trap state. These findings hold for the grid system in both the 8-hour and 640-hour cases.

Table 5.3. Identification of trap states in 8-hour and 640-hour grid system. Table shows proportion of Tasks Completed by the Markov chain simulation for states in Figure 3.1 when their probability of self-transition is raised to 1. The states where the proportion of Tasks Completed approaches 0 correspond to trap states. Shaded cells indicate verification by the large-scale simulation.

<table>
<thead>
<tr>
<th>State for which probability of self-transition is raised to 1</th>
<th>Proportion Flows Complete</th>
<th>Corresponds to Trap State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Waiting</td>
<td>0.400</td>
<td>No</td>
</tr>
<tr>
<td>Discovering</td>
<td>0.030</td>
<td>Yes</td>
</tr>
<tr>
<td>Negotiating</td>
<td>0.018</td>
<td>Yes</td>
</tr>
<tr>
<td>Monitoring</td>
<td>0.000</td>
<td>Yes</td>
</tr>
<tr>
<td>8-hour Flows Complete</td>
<td>0.756</td>
<td>0.019</td>
</tr>
<tr>
<td>640-hour Flows Complete</td>
<td>0.045</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Executing the path enumeration to find single-transition s-t cuts required less than 0.01 s second for both the Grid system and Abilene problems. Having used these methods to identify single-transition s-t cuts, trap states and minimal cut s-t sets with more than one state transition, it was then desirable to apply the perturbation algorithm to the corresponding TPM rows to verify these conditions and document the drop in tasks completed. To do this for the three single-transition s-t cuts shown in figure 4.1 requires systematically lowering one transition probability in each row with a weight of 1—the transition probability for the single-transition cut itself—while raising at most 5 others: one in the row for Initial, 3 for Negotiating, and one for Monitoring. In the 8-hour case, this entailed about 71 s of execution time. Reducing the two two-transition cut sets in figure 4.4 to approach 0 requires an additional 40 s each, while perturbing the three traps states to raise probability of self-transition to 1 as discussed above would add about 93 s execution time, or about 7% of the 56 minutes required for the original complete perturbation. For the 640-hour case, the time to perform the same perturbations was about 230 s, but is about 1.4% of the total 4.5 hours. These results are summarized and compared to the times for the large-scale simulation in table 5.4. The results show that path enumeration and minimal s-t cut set identification can lead to up to two orders of magnitude improvement in computational time over the use of the perturbation algorithm reported in [Da2009], which itself was found to constitute a two order magnitude improvement over large-scale simulation. Finally, it is important to point out that incremental perturbation of transitions identified in cut sets and trap states can be avoided to save time; however if this is done, one loses information about the rate of degradation in tasks completed which may be vital in understanding system sensitivity and the existence of thresholds. For example, see figures 3.7(a, b) or figure 3.8.

5.2 Abilene System

Table 5.5 shows the results of applying the perturbation algorithm to the Abilene Markov chain model. As before, the probabilities of transition from the Connecting, Initial Slow Start (ISS), Normal Congestion Control (NCA), Alternate Congestion Control (ACA), and Slow Start (SS) to other states is lowered to 0 (i.e., designated as sink columns $c^>$). At the same time probabilities of transitions to other states are raised (i.e., designated as primary increase columns $c^<$). However, in contrast to the Grid system case, we were unable to verify these results through the large-scale simulation due to the extreme cost in execution time (over 7 hours for a single execution). As before, the transition from the Initial state is omitted from the analysis.
With respect to single-transition s-t cuts for paths between the Initial and Flows Completed states, Table 5.5 shows that only the probability of transition for Connecting → Initial Slow Start, which if lowered to 0 (i.e., r = Connecting and Initial Slow Start is made c↑), causes Flows Completed to approach 0. Figure 4.2 shows that the state transition Connecting → Initial Slow Start is indeed a single-transition s-t cut. Table 5.5 shows no other cases where lowering a transition probability, by itself, causes Flows Completed to approach 0, except the transition out of the Initial state. Figure 4.2 shows no other single-transition s-t cuts, other than the cut from the Initial state. Hence, in the case of the Abilene system as in the Grid system, all single-transition s-t cuts found (there was 1) correspond to state transitions adversely impact system performance, if they are perturbed by the perturbation algorithm so that the probability of transition falls to 0.

Table 5.5. Correspondence between cases where perturbation algorithm results in tasks completed falling to 0 and existence of single-transition s-t cuts in 30-minute Abilene system simulation. The table shows the proportion of tasks completed and percent change when perturbation algorithm is applied to rows, r, of the TPM in figure 3.5 to decrease the probability of transition to 0 for the sink column, c↑, or state transition r → c↑, while increasing the probability of transition in the primary increase column, c↑, or r → c↑ at sink weight = 1. Secondary perturbation excluded. Right-most column indicates if single-transition s-t cut exists for r → c↑ in Figure 4.2. In all cases where perturbation causes proportion of Tasks Complete to approach 0, a positive correspondence exists with a single-transition s-t cut. Note: trap states are excluded and discussed separately below.

<table>
<thead>
<tr>
<th>(a) r = Initial Slow Start (ISS)</th>
<th>(b) r = Normal Congestion Avoidance (NCA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sink column (c↑)</td>
<td>Primary increase column (c↑)</td>
</tr>
<tr>
<td>ISS</td>
<td>NCA</td>
</tr>
<tr>
<td>ISS</td>
<td>ACA</td>
</tr>
<tr>
<td>ISS</td>
<td>SS</td>
</tr>
<tr>
<td>NCA</td>
<td>ISS</td>
</tr>
<tr>
<td>NCA</td>
<td>ACA</td>
</tr>
<tr>
<td>NCA</td>
<td>SS</td>
</tr>
<tr>
<td>ACA</td>
<td>ISS</td>
</tr>
<tr>
<td>ACA</td>
<td>ACA</td>
</tr>
<tr>
<td>ACA</td>
<td>SS</td>
</tr>
<tr>
<td>SS</td>
<td>ISS</td>
</tr>
<tr>
<td>SS</td>
<td>ACA</td>
</tr>
<tr>
<td>SS</td>
<td>NCA</td>
</tr>
</tbody>
</table>

In Figure 4.2, corresponds to Connected → Initial Slow Start single-transition s-t cut and Connected trap state

For the Abilene model, Table 5.6 shows that the perturbation algorithm found that two trap states, Connecting and Initial Slow Start, are the only states that result in Flows Completed approaching 0 when their self-transition probabilities are raised to 1. Figure 4.2 shows that Connecting and Initial Slow Start are the only trap states in the Abilene Markov chain model, and their existence is predicted by the path enumeration algorithm presented in Section 4. As in the case of the Grid system, the perturbation algorithm revealed no additional states for which raising the probability of self-transition to 1 caused Flows Completed to approach 0. As Table 5.6 shows, when the probability of self-transition is raised to 1 for the other three states—Normal Congestion Control, Alternate Congestion Control, and Slow Start—a smaller reduction in flows complete occurs that is relatively proportional to the chances that a flow enters a particular state. This
suggests that if a system fault occurs that causes flows to remain in one of these three states, use of an alternative congestion control regime is a possible remedy that would allow the system to function without a disastrous decline in performance.

Table 5.6. Identification of trap states in Abilene system Markov simulation. Table shows proportion of Flows Completed by the Markov chain simulation for states in Figure 3.4 when their probability of self-transition is raised to 1. The states where the proportion of Flows Completed approaches 0 correspond to trap states.

<table>
<thead>
<tr>
<th>State for which probability of self-transition is raised to 1</th>
<th>Proportion Flows Complete</th>
<th>Correlates to Trap State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>0.008</td>
<td>Yes</td>
</tr>
<tr>
<td>Initial Slow Start</td>
<td>0.007</td>
<td>Yes</td>
</tr>
<tr>
<td>Normal Congestion Avoidance</td>
<td>0.824</td>
<td>No</td>
</tr>
<tr>
<td>Alternate Congestion Avoidance</td>
<td>0.967</td>
<td>No</td>
</tr>
<tr>
<td>Slow Start</td>
<td>0.918</td>
<td>No</td>
</tr>
</tbody>
</table>

The path enumeration algorithm identified 10 minimal s-t cut sets that were multiple transition s-t cuts. These are listed in table 4.2, one of which is shown in figure 4.5. Simultaneously lowering the transition probabilities to 0 of all state transitions identified by these cut sets does indeed cause Flows Completed to approach 0. However, each of these cases represents situations that are less likely than the case of a single-transition s-t cut. For all multiple-transition s-t cuts in Table 4.2, the perturbation algorithm results showed that if the probability of transition of any single member of the cut set was not lowered, the proportion of Flows Completed remained high. This result suggests that the Abilene system possesses a relative degree of robustness as long as at least one path to a congestion control regime exists that can lead to completion of flows. Further experimentation would be necessary to determine if this conjecture actually holds in all cases for the large-scale Abilene simulation.

Finally, it is necessary to consider the relative efficiency on the minimal s-t cut set generation approach when compared to the perturbation algorithm overviewed in section 3.1. As in the Grid system case, it is desirable to apply the perturbation algorithm to the limited areas of the Abilene system TPM that were identified by the minimal s-t cut sets and trap states. For the single-transition s-t cut on Connecting→Initial Slow Start, this involves just one perturbation combination, which ran about 13 s. For the two trap states, Connecting involved one perturbation combination that ran 90 s, while Initial Slow Start required an additional three, for about 260 s. Simultaneously lowering the transition probabilities to 0 in the any of the minimal s-t cut sets shown in table 4.2 involved roughly 10-20 s each. These results are summarized in Table 5.7. As in the case of the Grid system, these computations required a very small fraction of the 27.3 hours that was needed to apply the perturbation algorithm to all rows of the Abilene Markov chain TPMs.
Table 5.7. Comparative execution times. Comparison of execution times for grid computing system and Abilene network using (a) Identification of minimal s-t cut sets followed by use of perturbation algorithm on state transitions identified as single transition s-t cuts, with (b) Exhaustive search of the rows of a TPM using the perturbation algorithm as described in [Da2009] and (c) large-scale simulation [Mi2006].

<table>
<thead>
<tr>
<th></th>
<th>Minimal s-t cut set Identification</th>
<th>Exhaustive search of TPM rows with perturbation algorithm</th>
<th>Large-scale simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Path Enumeration Algorithm&lt;sup&gt;a&lt;/sup&gt;</td>
<td>Perturbation of individual state transitions only</td>
<td></td>
</tr>
<tr>
<td>Grid Computing System</td>
<td>8-hour &lt;0.01 s &lt;0.01 s</td>
<td>244 s</td>
<td>56 minutes</td>
</tr>
<tr>
<td></td>
<td>640-hour &lt;0.01 s &lt;0.01 s</td>
<td>230 s</td>
<td>4.5 hours</td>
</tr>
<tr>
<td>Abilene Network</td>
<td>8-hour &lt;0.01 s &lt;0.01 s</td>
<td>450 s</td>
<td>27.3 hours</td>
</tr>
</tbody>
</table>

<sup>a</sup>Note: results achieved by path enumeration were also achieved using the node contraction algorithm, as described in the Section 6.
6. Application of Minimal s-t Cut Set Identification to a Larger Markov Chain Problems

Section 5 showed the feasibility of using minimal s-t cut sets on paths from the Initial to the absorbing state to predict which state transitions, if perturbed, are most likely to adversely impact system performance. However, the path enumeration algorithm for finding minimal s-t cut sets discussed in Section 5 is not scalable for two reasons. First, in the worst-case situation of a complete graph (a graph where each pair of vertices is connected by an edge), enumeration of all non-cyclic paths between two vertices has a complexity that is factorial with respect to the number of vertices \([Fr1975, Ba2006]\). This makes a path enumeration algorithm impractical for larger Markov chains. Second, in larger Markov chains, it may be that all paths leading to an absorbing state cannot be disconnected by cutting a single state transition. Therefore, it is necessary to find an algorithm that can (1) work efficiently on larger Markov chain problems; and (2) identify minimal s-t cut sets consisting of multiple state transition that disconnect all paths leading from the Initial to the absorbing state.

In this section we address the question of how an approach based on minimal s-t cut set analysis might work for larger Markov chains models. We first introduce an algorithm, known as the node contraction algorithm, which fulfills the two requirements identified in the preceding paragraph. The algorithm finds minimal s-t cut sets probabilistically and can be bounded to run for a limited amount of time, though it is not guaranteed to find all cut sets. We then provide examples of algorithm’s use for Markov chain problems and assess its potential for Markov problems of significant size and complexity. We find that while these investigations into the feasibility of use of non-exhaustive techniques to generate minimal s-t cut sets are not complete and much more work remains to be done, they do provide some encouragement that this approach will work for larger problems. Finally, we compare the different approaches presented in this report to finding minimal s-t cut sets on paths between the Initial and absorbing state, and describe the circumstances under which each should be used.

6.1 A Probabilistic Algorithm for Finding Minimal s-t Cut Sets in Larger Markov Chains

This section describes an algorithm for finding minimal s-t cut sets that also has the potential to be effective for larger Markov chain models. As before, this algorithm should also identify single-transition s-t cuts and trap states. However, the algorithm should also efficiently identify minimal s-t cut sets with multiple edges, which correspond to sets of state transitions in a Markov chain, which if perturbed together, cause degradations in performance (as measured by the proportion of tasks that enter the Tasks Completed state or proportion of flows that enter the Flows Completed). Possibly, the set of state transitions in such a minimal s-t cut set will represent concrete related circumstances in the domain being modeled, or may represent unrelated events which could randomly occur together. We assume that in large domains, minimal s-t cut sets of greatest interest, though plural, will be small, that the most critical cut sets will have a small number of transitions, since small combinations are more likely, as shown in [Ka2001]. Therefore, a reasonable goal would be to generate minimal s-t cut sets consisting of a limited number of state transitions. It may be desirable that these cut sets should contain the minimum number of state transitions needed to disconnect the initial state from the absorbing state, or be slightly larger.

A number of algorithms have been developed for enumerating all minimal s-t cut sets between two vertices in undirected graphs \([Ha1999, Pr1996, Ts1980, Ya1994, Lin2003]\) and directed graphs \([Pr1996, Ya1994, Lin2003]\). All require considerable computational effort for large graphs, which is not fully determined in all cases. An algorithm that is often noted for its efficiency is that \([Ts1980]\), which has a known complexity of \(O(n + m)(u + 1)\) for undirected graphs, where \(n\) is the number of vertices, \(m\) is the number edges, and \(u\) is the actual number of cut sets that can be found. However, this algorithm can be computationally expensive as well, as Markov chains with as few as fifty states, can contain over \(10^8\) minimal s-t cut sets between the
initial and absorbing states (see below). An interesting alternative to enumeration is the node contraction algorithm, which while not guaranteed to find all minimal s-t cut sets, can be controlled to bound computational cost. Efficient implementations of this algorithm for undirected graphs run in $O(n^2)$ time [Ka1996]. However, computational characteristics for directed graphs have not been determined and remain a topic for future work. Below we show that this algorithm can find a large proportion of minimum and near minimum cut sets in a sample of larger Markov chain problems. Section 5 returns to the subject of other minimal s-t cut set algorithms which could be explored in future work.

The node contraction algorithm operates by randomly choosing two vertices connected by an edge and replacing these vertices with a single, new vertex. The new vertex assumes the edges by which the two replaced vertices were connected to the remainder of the graph (i.e., the edges of replaced vertices become the edges of the new vertex) and takes up the edges that connected the two replaced vertices. The process of randomly selecting pairs of vertices repeats until only two large, mega-vertices remain. The directed edges between the two remaining mega-vertices $c_1$ and $c_2$, and the directed edges between vertices $<v_1, v_2>$, $v_1 \neq v_2$, in which $v_1$ was replaced by $c_1$ and $v_2$ was replaced by $c_2$, constitute a minimal s-t cut set of the graph. We apply this algorithm to the Markov chain, modifying it to prevent the two vertices representing the Initial state and desired absorbing state (Tasks or Flows Completed) from being replaced by the same vertex. This ensures that the Initial state, $s$, and Tasks Completed state, $t$, will not both end up in either $c_1$ or $c_2$. In this way, the edges between the two remaining mega-vertices, $c_1$ and $c_2$, together with the vertices each has absorbed, yield an s-t cut set of state transitions, which if removed, disconnect the Initial and absorbing state (Tasks or Flows Completed).

Since the algorithm randomly selects two connected vertices to combine, repeated applications produce different cut sets. The more the algorithm is repeated, the greater the chances that a large proportion, if not all, of the minimal s-t cut sets of interest will be obtained. Hence, the operation of the algorithm can be said to be probabilistic. Because the number of repetitions can be controlled, computation cost can be bounded. Further, cut sets can identify potential trap states, which exist when all transitions in the cut set emanate from the same state. Lastly, Markov simulation need be applied only to the transitions in the s-t cut sets, in order to generate curves of tasks completed, such as are shown in Figures 3.7 (a, b) and 3.8, and to identify performance thresholds. However, to be scalable, the algorithm must be effective in producing the most critical minimal s-t cut sets in a relatively limited number of repetitions. In the next section, we describe an example of the operation of this algorithm. Pseudo-code for one repetition of the algorithm is given in Appendix A.

6.2 Examples of the Application of the Node Contraction Algorithm

In this section we provide two examples of the application of the node contraction algorithm so that the reader can see more clearly how the algorithm operates. First, we apply the node contraction algorithm to the Grid computing system and Abilene system Markov models and compare the results of these small problems to the results provided in Section 4. Then we apply the node contraction algorithm to a much larger example problem to see what these results look like. In Section 6.3, we provide a more extended analysis of the application of the algorithm to a set of larger matrices and provide some quantitative results.

6.2.1 Example Application to the Grid Computing and Abilene System Markov Chain Problems

The node contraction algorithm produced 5 minimal s-t cut sets for the grid computing system Markov chain. These are the three single-transition s-t cuts that appear in Figure 4.4 and two multiple transition s-t cuts that appear in Figure 4.5. This algorithm also found 10 minimal cut sets for the Abilene system, shown in Table 4.1. Both are the complete sets of minimal s-t cut sets from the Initial to absorbing states for both Markov chain problems. Both sets required 100 repetitions of this algorithm, which consumed less than a
0.01 s of CPU time. In both cases, the algorithm identified all single-transition s-t cuts—in other words, all minimal s-t cut sets that with a single state transition—that were also obtained through path enumeration described in Section 4.1. In section 5, the single- and multiple-transition cuts were shown to correspond exactly to critical transitions in Markov chain graphs that could be perturbed to produce large performance degradations.

### 6.2.2 Example Application to a Large Markov Chain TPM

This section illustrates an example of the use of the node contraction algorithm on a larger Markov Chain matrix with 136 states. Unlike the Grid or Abilene system problems, this Markov chain was generated by a matrix generation program as a test problem. Hence, it does not model a real-world system in which states can be given concrete interpretations; instead, states are numbered from 1 to 136. This Markov chain is homogeneous with respect to time, and provides no time step information. Though originally an ergodic chain, it has been modified to substitute a single absorbing state (state 134) to allow it to behave as an absorbing chain for our purposes. A compressed visualization of TPM appears in figure 6.1. As is easy to see, it is a sparse matrix in which processes proceed by following state transitions roughly along the matrix diagonal to the introduced absorbing state 134. However, most states also provide transitions that lead backwards toward states with lower numbers, which greatly increases the number of potential paths.

![Figure 6.1: Compressed view of TPM rw136 for a Markov chain of 136 states from [rw136]. The horizontal axis represents the numbered states in ascending order from left to right; the vertical axis represents the states in descending order from bottom to top. Shaded cells represent non-zero values. The pattern of shaded cells clearly shows a sparse diagonal matrix.](image)

This problem is of sufficient size and complexity to easily prevent enumeration of all paths from the initial state to the single absorbing state, making it a good test problem for our purposes as well. An application of the breadth-first path exhaustive search algorithm described in Section 4.1 does quickly confirm that there are at least two paths having no common states (other than the Initial state or the single absorbing state); hence,
no single-single-transition s-t cuts or individual trap states will be found. However, an application of the node contraction algorithm does find a number of minimal s-t cut sets consisting of 2-5 state transitions—as well as larger cut sets. Table 6.1 shows a sample from a total produced by 200 repetitions of the cut-set algorithm, which generated in 491 seconds. The sample is chosen on the basis of a low total transition probability for cut set members. A much longer execution of 2500 repetitions produced only two cut sets having lower total transition probabilities. The time needed to generate the sample in Table 6.1 is no doubt a small fraction of time needed to completely perturb this large matrix using the perturbation algorithm described in [Da2009a] and overviewed in section 3.1 (which we did not even attempt). Each of the minimal s-t cut sets in table 6.1 was also verified by removing its transitions (edges) from the 136-state graph and executing the path search algorithm, which is then unable to find any paths.

Table 6.1. Selected minimal s-t cut sets between the Initial state and the absorbing state in a Markov chain of 136 states produced by the node contraction algorithm. These cut sets represent low combinations of number of from states, state transitions, and total of state transition probabilities. Each was verified as an s-t cut set using Markov simulation program.

<table>
<thead>
<tr>
<th>Number from states/ transitions</th>
<th>List of transitions (200)</th>
<th>Sum of transition probabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 3 / 4</td>
<td>131→134, 128→132, 129→133, 131→132</td>
<td>0.233</td>
</tr>
<tr>
<td>2 3 / 4</td>
<td>128→132, 129→131, 129→133, 127→131</td>
<td>0.233</td>
</tr>
<tr>
<td>3 4 / 5</td>
<td>123→128, 124→129, 125→130, 125→127, 122→127</td>
<td>0.367</td>
</tr>
<tr>
<td>4 4 / 5</td>
<td>125→130, 124→129, 123→128, 127→131, 127→128</td>
<td>0.400</td>
</tr>
<tr>
<td>5 5 / 6</td>
<td>123→128, 124→125, 124→129, 119→125, 120→126, 122→127</td>
<td>0.467</td>
</tr>
<tr>
<td>6 6 / 7</td>
<td>114→116, 109→116, 110→117, 111→118, 112→119, 113→120, 114→121</td>
<td>0.733</td>
</tr>
<tr>
<td>7 3 / 4</td>
<td>131→134, 131→132, 128→132, 133→132</td>
<td>0.700</td>
</tr>
<tr>
<td>8 1 / 2</td>
<td>1→2, 1→17</td>
<td>1.0</td>
</tr>
<tr>
<td>9 2 / 3</td>
<td>2→18, 2→3, 1→17</td>
<td>1.43</td>
</tr>
</tbody>
</table>

In this example, the identification of a moderate number of small minimal s-t cut sets in a reasonable amount of time provides a tractable means for identifying where performance degradations can occur within a larger Markov chain. To verify that implementing the cut sets in Table 6.1 would actually cause Flows Completed to approach 0 (i.e. disconnect the graph), we first executed the 136 × 136 TPM in our Markov simulation program and modeled the evolution of the 136-element state vector, using the procedure described in section 3 and [Da2009a]. Again, by modifying the Markov perturbation algorithm to simultaneously lower a set of transition probabilities for state transitions in a cut set to 0, it was possible to model the effect of these reductions on the flow of processes into state 134, the designated absorbing state. The result for each of the minimal s-t cut sets in Table 6.1 was that Flows Completed fell toward 0, thus repeating the outcome of the Grid and Abilene system models.
6.3 Performance of the Node Contraction Algorithm on Four Larger Markov Chain Problems

In this section we follow up on the example of the application of minimal s-t cut set analysis using the node contraction algorithm to a large Markov chain problem. In doing this, we hope to provide a preliminary evaluation of the effectiveness of minimal s-t cut set analysis using this algorithm on a set of large, complex problems. As pointed out above, this algorithm generates minimal s-t cut sets probabilistically, but does not enumerate all possible cut sets. Therefore, the question naturally arises as to whether it either misses critical minimal s-t cut sets in larger Markov chains, or if it requires too many repetitions to generate the most critical ones.

To investigate these questions required that the node contraction algorithm be applied to a large absorbing Markov chain for which, ideally, all critical minimal s-t cut sets between the initial and absorbing states could be obtained by other means to provide a baseline for purposes of comparison. To accomplish this, we adapted the exhaustive minimal s-t cut set enumeration algorithm of [Ts1980], which was designed to generate all minimal s-t cut sets between two vertices in an undirected graph. We modified the algorithm to work for directed graphs. We chose this algorithm because of its known efficiency in enumerating s-t cuts in undirected graphs (see discussion above). We conjecture that the modified algorithm enumerates all minimal s-t cut sets in directed graphs, and we are developing a proof of this. Extensive testing on sample problems found no cases in which the modified algorithm failed to produce all minimal s-t cut sets and has also verified that all cut sets produced were not duplicative. We are also investigating the use of works such as [Pr1996], which describe efficient algorithms for enumerating minimal s-t cuts in directed graphs and might also be used as a baseline in future experiments.

We obtained four large Markov chain matrices for which most, if not all, minimal s-t cut sets between initial and absorbing state could be generated using the algorithm described in [Ts1980], though it might take many hours or even days. These matrices, which appear in Appendix B, are sparse matrices. Matrices 1 and 2 were generated using [Hu1994] and are based on [Je1986] and [Bo1988] respectively. Matrix 3 was a 50 × 50 subset of a very large 136 ×136 Markov chain matrix described in [rw136]. Matrix 4 was generated using [Hu1994] and is based on [Gr1997]. Each of these matrices was originally an ergodic Markov chain that was modified to become absorbing chains by designating a single absorbing state.

To identify which minimal s-t cut sets between the Initial and absorbing states were critical, we developed selection criteria that might conceivably be used in a real-world situation. These criteria are based on the idea that in an actual real-world system, it is likely that expert guidance will be required to select which cut sets that represent circumstances of interest. A domain expert might consider various criteria for selecting minimal s-t cut sets to examine further. For example, one possible selection criteria would be minimal s-t cut sets that have few edges (state transitions). The justification for this criterion might be that the most critical minimal s-t cut sets will consist of a small number of transitions, since small cut sets represent combinations of circumstances that are more likely to occur together and thus more likely to severely impact a system. (Note: in [Ka2001], this intuition is partially corroborated for the case of undirected graphs by the finding that small cut sets are more likely to lead to disconnection of undirected graphs, if the transitions in such cut sets can be equated to edges of a graph that fail independently with a known probability.) Another criterion might be cut sets for which the total probability of transition of all member state transitions is low. Minimal s-t cut sets with total transition probability near 0 are likely to be sensitive to small perturbations, which quickly drive down system performance. On the other hand, cut sets with high total transition probabilities consist of state transitions that are more likely to be taken. Hence, they may be good candidates as well. Hence, we

---

3 For instance, in one test, the TPM for a Markov chain described below (Matrix 1) was subjected to over $5 \times 10^6$ repetitions of the node contraction algorithm. These repetitions were unable to generate any new minimal s-t cut sets that were not also obtained through the modified algorithm of [Ts1980]. Such results provide confidence that the algorithm we are using actually enumerates all minimal s-t cut sets for directed graphs.
defined three such criteria for ordering minimal s-t cut sets that are based on such considerations and that we believe are sufficient for the purposes of this experiment. Therefore, we chose the first criterion, Sort A, to rank minimal s-t cut sets by the fewest number of edges as a primary sorting criterion and lowest total transition probability of edges as the secondary criterion. The second, Sort B, uses only the lowest total transition probability of edges in the cut set as a sorting criterion (which also tends to rank cut sets with fewer transitions higher). Hence, Sorts A and B are likely to identify minimal s-t cut sets in which the smaller perturbations to the fewest number of state transitions are likely to produce the largest changes. The third ranking criteria, Sort C, uses least number of edges as a primary sorting criterion and highest total transition probability of edges as a secondary criterion. Sort C identifies cut sets consisting of state transitions more likely to be taken and therefore, if perturbed, could impact on system behavior. No doubt further research is necessary to investigate criteria for organizing minimal s-t cut sets for larger Markov chains in order to decide which are likely to be important. Nevertheless, having found a means to determine the critical minimal cut sets between the initial and absorbing states in a sufficiently large Markov chain, it was then possible to run the node contraction algorithm to see if it could also find the critical cut sets in a reasonable amount of time.

Table 6.2 shows the results of the application of the modified algorithm of [Ts1980] and the node contraction algorithm to all four matrices. Both algorithms were parameterized to rank minimal s-t cut sets generated by three sorting criteria described above. Table 6.2 lists the total number of minimal s-t cut sets generated by the modified algorithm of [Ts1980] and the time required to do this. The table then compares the performance of the node contraction algorithm. The table shows what proportion of the top-ranked 100 cut sets, as ordered by the three sorting criteria described above, that the node contraction algorithm was able to produce in a specified number of repetitions and the time required. We examined the performance of the node contraction algorithm at three levels of effort: 1000, 10,000, and 100,000 repetitions. Note that the number of minimal s-t cut sets generated and time required did not always grow linearly with the matrix order (number of rows and columns). This was due to differences in topology and interconnectedness in the Markov chains; in some cases, a few states with large numbers of transitions can drastically increase the number of possible cut sets and thus increase the level of effort needed to enumerate all transitions.

Table 6.2. Comparison of minimal s-t cut sets for paths between the Initial and absorbing states generated by the modified algorithm of [Ts1980] and the node contraction algorithm. Both algorithms were applied to the four matrices reproduced in Appendix B. The node contraction algorithm was executed a three levels of effort: 1000, 10,000, and 100,000 repetitions. Minimal s-t cut sets generated were sorted by: (Sort A) fewest number of edges as a primary sorting criteria with lowest total transition probability of edges as a secondary sorting criteria; (Sort B) lowest total transition probability of all edges in the cut set; and (Sort C) fewest number of edges as a primary sorting criteria with highest total transition probability of edges as a secondary criterion.

<table>
<thead>
<tr>
<th>Matrix Number of [Ts1980]</th>
<th>Modified algorithm of [Ts1980]</th>
<th>Proportion (in %) of 100 top-ranked minimal s-t cut sets ranked by criteria A, B and C, which were found by node contraction algorithm</th>
<th>Time</th>
<th>Sort A</th>
<th>Sort B</th>
<th>Sort C</th>
<th>Time</th>
<th>Sort A</th>
<th>Sort B</th>
<th>Sort C</th>
<th>Time</th>
<th>Sort A</th>
<th>Sort B</th>
<th>Sort C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Number of minimal s-t cut sets generated</td>
<td></td>
<td>After 1000 repetitions</td>
<td>After 10,000 repetitions</td>
<td>After 100,000 repetitions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Time</td>
<td>Sort</td>
<td>Sort</td>
<td>Sort</td>
<td>Time</td>
<td>Sort</td>
<td>Sort</td>
<td>Time</td>
<td>Sort</td>
<td>Sort</td>
<td>Sort</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>50</td>
<td>530,432</td>
<td>345 s</td>
<td>63 s</td>
<td>67</td>
<td>56</td>
<td>22</td>
<td>640 s</td>
<td>100</td>
<td>80</td>
<td>96</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>2</td>
<td>50</td>
<td>28,230,288</td>
<td>14.6 hours</td>
<td>17 s</td>
<td>58</td>
<td>49</td>
<td>36</td>
<td>171 s</td>
<td>98</td>
<td>93</td>
<td>65</td>
<td>1710 s</td>
<td>100</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>50</td>
<td>27,242,634</td>
<td>50 hours</td>
<td>22 s</td>
<td>86</td>
<td>48</td>
<td>87</td>
<td>218 s</td>
<td>100</td>
<td>67</td>
<td>100</td>
<td>2288 s</td>
<td>100</td>
<td>88</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>422,060,801</td>
<td>193.6 hours</td>
<td>11 s</td>
<td>30</td>
<td>15</td>
<td>22</td>
<td>106 s</td>
<td>80</td>
<td>30</td>
<td>62</td>
<td>1051 s</td>
<td>100</td>
<td>37</td>
</tr>
</tbody>
</table>

Table 6.2 shows that the node contraction algorithm, if executed for a sufficient number of repetitions, could generate all the top ranked minimal s-t cut sets between the Initial and absorbing states that were also generated using the modified algorithm of [Ts1980]—in most cases. For instance, for Matrices 2 and 3, the

Draft Document
algorithm was able to find almost all top 100 minimal s-t cut sets in a relatively small fraction of the number of hours required by the modified algorithm of [Ts1980]. For matrix 4, the node contraction algorithm could find all the top 100 minimal s-t cut sets under sort criteria A and C in about 15 minutes (as opposed to 193.6 hours by the modified algorithm of [Ts1980]). However, the algorithm found only 37 of 100 high ranked minimal s-t cut sets under Sort B. Moreover, for Matrix 1, Table 6.2 shows that the node contraction algorithm had to run longer than the modified algorithm of [Ts1980], before it began to produce a large percentage of highly-ranked cut sets. The differences in performance are attributable in part to large problem size, as for instance, Matrix 4 which has 422,060,801 minimal s-t cut sets. The differences in performance may also be attributable to topological characteristics such as vertices (states) with large numbers of edges (state transitions), which increase the amount of interconnectivity, as in Matrices 1 and 4 (see Appendix B). Matrix characteristics such as high interconnectivity may possibly serve as impediments to the operation of the node contraction algorithm, causing it too run longer to find all highly-ranked cut sets. In the case of Matrix 1 which has only 530,432 minimal s-t cut sets, it may actually be more efficient to enumerate cut sets rather than to generate them probabilistically.

Despite these significant exceptions, the data shows that it is possible to use the node contraction algorithm to find a high proportion of critical minimal s-t cut sets between the Initial and absorbing states in larger Markov chains. However, further work is necessary on a wide variety of problems to fully demonstrate scalability and the ability to handle large, complex Markov chain problems. In addition, further research is needed to find more effective minimal s-t cut set generation methods that employ probabilistic or heuristic approaches.

6.4. Discussion and Future Work

The results suggest that the four approaches that employ graph theory concepts could provide an effective means for discovering areas of sensitivity within an absorbing Markov chain representation of a dynamic system. These four approaches are based on use of:

1. A perturbation algorithm of the type described in [Da2009a] and overviewed section 3.1 (note: a second perturbation algorithm is discussed in the next section and presented in Appendix C),
2. A algorithm, such as described in Section 4.1, to enumerate all paths between an Initial state and an absorbing state of interest,
3. An exhaustive algorithm for directed graphs [Pr1996, Ya1994, Lin2003] (or an adaption of an algorithm such as [Ts1980] for undirected graphs) that could generate all minimal s-t cut sets between an Initial state and an absorbing state of interest,
4. A robust, probabilistic approach, such as the node contraction algorithm, that generates a high proportion of the most critical minimal s-t cut sets in significantly less time than exhaustive enumeration.

The choice of which of the four approaches to use may depend on problem size and circumstances. If the problem is very small, a perturbation algorithm such as described in section 3 can be used directly as in [Da2009a]. For larger problems, one might apply a path enumeration algorithm to identify all paths to the absorbing state and then determine if cutting one transition can disconnect all paths. Then one may apply the perturbation algorithm to a limited number of perturbation combinations determined by the “from” and “to” states of the single-transition s-t cut(s). For a problem of moderate size for which sufficient time and computational resources exist, it may be desirable to enumerate all minimal s-t cut sets that disconnect the Initial state and the desired absorbing state. If the problem is larger, or one state transition is insufficient to disconnect all paths to the absorbing state, a more robust approach, such as the node contraction algorithm, can be applied to find minimal s-t cut sets with multiple transitions. The perturbation algorithm can then be
applied in parallel to the related TPM rows as was described above to learn the rate of degradation in system performance.

With regards to future work, methods will be needed to better determine which of possibly many minimal s-t cut sets are most likely to be important in affecting system performance. For instance, one may explore different criteria for ranking alternative cut sets, in addition to the criteria provided in Section 6.3. It will also be important to understand how domain expertise might be leveraged in combination with various criteria for ranking cut sets.

Another area of future work is the investigation of other scalable methods for finding minimal s-t cut sets in directed graphs, such as, for instance, adapting alternative approaches to probabilistic node contraction [Ka1996]. Along these lines, it may be possible to combine node contraction with lumping techniques, mentioned earlier in the section on previous work [Ke1976, Si1992, Bu1995, Ni2004], to reduce problem size. By selectively lumping vertices (states) into clusters, in such as way as to eliminate non-critical state transitions, it may be possible to reduce the complexity of finding minimal s-t cut sets. Related strategies for simplifying large Markov chains involve partitioning the graphs into clusters of closely related vertices and exploring cluster connections as possible minimal s-t cut sets. For this, we may consider adapting the contraction algorithm or the work of [Ta1972] and others on graph division.

One may also explore other approaches that can be used to generate minimal cut sets between two vertices with less than exponential time complexity. These include, for instance, adaptation of methods for analysis of network flows [Fo1962, Ev1998] for use on Markov chain problems. Finally, it is important to extend the analysis to minimal s-t separating sets, consisting of combinations of multiple trap states and state transitions (even though doing so will have the effect of increasing problem size rather than reducing it). Separating sets may correspond to important real-world circumstances that impact system performance.
7. Theoretical Model of a Markov Chain

(To be provided)
8. Conclusions

This work presents an advance in our efforts to develop analytical and computational tools for analyzing dynamic behavior in large scale distributed computing systems. Building on earlier work in [Da2009a, Da2009b] on the piece-wise homogeneous approximation of the large-scale grid computing simulation, we showed how the application of existing methods in graph theory and the spectral theory of Markov chains can be used to predict system qualitative behavior by comparing it to a discrete event simulation of the full system. Furthermore, we demonstrated that the methods discussed in this paper greatly reduce the time and effort required to identify failure scenarios discussed in these earlier papers.

Our first technique was based on the graph topology induced by the Markov chains that constitute the pieces of the approximation. In the cited previous work, critical transitions that lead to system degradation or failure were identified by a perturbation algorithm, which conducted a restricted exhaustive search based on the systematic perturbation of TPM elements (see Section 3.1). The effect of any single perturbation was then determined by finding the proportion of tasks completed at the end of the observation time interval. This quantity is well approximated by iterating the TPMs over the observation time. In Sections 4 and 5 of this report, key concepts from graph theory were used to introduce more efficient methods for identifying minimal s-t cut sets that disconnected all paths through the Markov chain graph from the Initial state to a desired end state. The edges of the cut sets identified critical state transitions, i.e., transitions that led to large performance degradation when they were perturbed. When applied to the Markov chain for the grid system and that of a second problem, the Abilene network, these methods found the same critical transitions as those found by the perturbation algorithm, but at a substantial savings in computational cost (See Section 5).

To investigate the scalability of this new approach, one of the methods—the node contraction algorithm—was applied to graph topologies induced by larger stochastic matrices than the ones derived from the grid computing system or Abilene network. The results show that, with some exceptions, the contraction algorithm was able to find a large proportion of most highly ranked critical cut sets in two orders of magnitude less time (20 minutes or less) than using the modified algorithm of [Ts1980]. The major exception involved a topology that was highly connected but had a sufficiently small number of cut sets that could be more easily enumerated.
The two methods presented here for efficient analysis of Markov chain representations of dynamic systems are intended to provide a basis for development of more practical tools. As such, this report has also provided considerable detailed data and test results that can be used by researchers in the further development of such tools. Where appropriate, the report has also outlined avenues of future research that will be needed and that could lead to improved analysis methods. As with our earlier work, this report shows that Markov chain analysis is a valuable tool for understanding complex system behavior in large-scale systems and can be used to predict the behavior of such systems.
9. References


Appendix A. Node Contraction Algorithm

/* Procedure for generating minimal s-t cut set of a directed graph. Procedure accepts TPM and ID of start state and absorbing state. It randomly selects two vertices, combines them into one vertex whose out arcs are the combined out arcs of the two vertices. Ensures start and absorbing vertices are always in separate vertices. Procedure repeats until two vertices are left, and cut sets are then generated. */

Procedure PerformContraction (float TPM[ ][*], integer startState, absorbingState, matrixOrder)

returning set (cut set members)

boolean notContractible [matrixOrder]; /* if TRUE, vertex has already been contracted or has been denoted as being not contractible */

integer contractionRecord [matrixOrder]; /* Array in which vertex denoted by index value has been contracted into vertex denoted by value in array element */

integer toBeContracted; /* Vertex that is to be contracted into intoVertex */

integer intoVertex; /* Vertex that toBeContracted is contracted into */

integer verticesLeft; /* Number vertices left to be contracted */

/* Begin by initializing */
verticesLeft = matrixOrder;
notContractible [startState] = TRUE;
notContractible [absorbingState] = TRUE;

forever /* Loop repeats contractions until only two vertices are left */
{
    forever /* Loop to select eligible pair of vertices to contract */
    {
        /* Randomly select next proposed vertex from 1..matrixOrder to contract */
        toBeContracted = selectRandom (1, matrixOrder);

        /* If this vertex has already been contracted, retry (continue to next iteration of forever loop) */
        if ( notContractible [toBeContracted] ) continue;

        /* Otherwise, mark vertex to be contracted and select vertex into which to contract BeContracted. */
        notContractible [toBeContracted] = TRUE;
        intoVertex = getContractedInto (toBeContracted, notContractible, TPM);

        /* Check if proposed contraction causes start and absorbing states to be contracted into same vertex. */
        if ( any of contracted vertices from toBeContracted and intoVertex contain startState or absorbingState )
            continue;

        /* if contractionConflict */
        if (contractionConflict)
            { notContractible [toBeContracted] = FALSE;
             continue; } /* end of forever loop to select pair of vertices to contract */

/* procedure continues on next page */
procedure PerformContraction continued

/* Perform contraction operation */
contractVertices (toBeContracted, intoVertex, contractedMatrix, matrixOrder);
contractionRecord [toBeContracted] = intoVertex;

/* Decrement number of vertices left that can be contracted. If number left = 2, break from loop */
verticesLeft = verticesLeft - 1;
if (verticesLeft == 2) break;
} /* End of forever loop that repeats contractions until only two vertices are left */

/* Retrieve cut set that exists between two remaining vertices. */
NewCutSet = getCutSet (notContractible, contractionRecord);

/* ensure cut set disconnects that graph and that it is minimal */
If (verifyCutSet (NewCutSet, TPM)
{ minimalizeCutSet (newCutSet)
  return NewCutSet;
}
else
  return NULL;
END
} /* end procedure PerformContraction () */
/* Procedure that randomly select of adjoining vertex that toBeContracted will be contracted into and with which toBeContracted has a state transition. */

procedure getContractedInto (integer toBeContracted, boolean notContractible[*], float TPM [[*]*])
returning integer
{
    integer proposedVertex;        /* Candidate vertex selected for contraction into. */

    integer linked [matrixOrder];   /* Identifies candidate vertices to contract into: i.e., vertices that are yet uncontracted with which vertex toBeContracted has arcs. */

    float numLinks;                /* Number of vertices in linked */

    integer state, k, nthState;     /* Counters and index variables*/

    /* Populate linked array with candidate vertices with arcs that indicate transitions with state toBeContracted and that have not previously been contracted. */
    for (state = 1 to matrixOrder)
        if (((TPM [state, toBeContracted] > 0) or (TPM [toBeContracted, state] > 0)) and 
           (not notContractible [state]) and 
           (not (state == toBeContracted)))
            { k = k + 1; 
              linked [k] = state; 
              numLinks = numLinks + 1; 
            }

    /* Randomly pick nth state from 1..numLinks to be proposed state that toBeContracted will be contracted into. Proposed state to be returned is nth value of linked array. */
    nthState = random ( 1, numLinks);
    proposéState = linked [nthState];

    return proposéState;
}

} // procedure getContractedInto
procedure checkContractionConflict ( integer toBeContracted, intoVertex, StartState, TargetState, integer contractionRecord [*])
returning boolean
{
    integer followVertex; // next vertex in chain of contractions
    integer vStartState, // previous (last) vertex that was contracted into for start state
    integer vTargetState; // previous (last) vertex that was contracted into for target state
    integer pContractionRecord [matrixOrder]; // proposed contraction record if toBeContracted is contracted

    /* First, create proposed contraction record in which toBeContracted is contracted into intoVertex. */
    for (i = 1; i <= matrixOrder)  pContractionRecord [i] =  contractionRecord [i];
    draftContractionRecord [toBeContracted] =  intoVertex;

    /* Follows chain of contraction for start state to find vertex into which start state in currently contracted */
    followVertex=StartState;
    forever
    { vStartState=followVertex;
        if (pContractionRecord [followVertex] == 0) break;
        followVertex = draftContractionRecord [followVertex];
    }

    /* Follows chain of contraction for start state to find vertex into which start state in currently contracted */
    followVertex= TargetState;
    forever
    { vTargetState=followVertex;
        if (pContractionRecord [followVertex] == 0) break;
        followVertex = draftContractionRecord [followVertex];
    }

    /* return value of boolean proposition that vertex into which Start State and Target State 
       were contracted into are equal */
    return (vTargetState == vStartState);
}
} // procedure checkContractionConflict
/** Procedure to modify TPM so that arcs going into and out of vertex toBeContracted now go into and out of vertex intoVertex. */

procedure contractVertices (integer toBeContracted, intoVertex, 
inout float TPM [ *][*], integer matrixOrder)
{
    float probabilityOfTransition;

    /* Modify TPM to connect arcs going into toBeContracted so that they go into intoVertex */
    for ( i =1; i <= matrixOrder) 
        if ( TPM [toBeContracted, i] > 0) 
            { probabilityOfTransition = TPM [toBeContracted , i];
              TPM [intoVertex, i] = TPM [intoVertex, i] + probabilityOfTransition;
              TPM [toBeContracted, i]=0;
            }

    /* Modify TPM to connect arcs going from toBeContracted so that they go from intoVertex */
    for ( i =1; i <= matrixOrder)
        if ( TPM [i, toBeContracted] > 0) 
            { TPM [i, intoVertex] = TPM [i, intoVertex] + TPM [i, toBeContracted];
              TPM [i, toBeContracted]=0;
            }

    /* Zero out self-transition probability */
    TPM [intoVertex, intoVertex] = 0;
}
} // procedure contractVertices
procedure getCutSet (boolean notContractible [*], integer contractionRecord)
	returning cutSet
{
	en integer vertex1, vertex2; /* two remaining vertices in contracted graph */

	/* Identify two remaining vertices in contracted graph */
	for (i = 1 to matrixOrder)
		if (not notContractible[i])
			{ vertex1 = i;
				break;
			}

	for (j = i to matrixOrder)
		if (not notContractible[j])
			{ vertex2 = j;
				break;
			}

arcSet1 = retrieveLinks (vertex1, vertex2, TPM, contractionRecord);
arcSet2 = retrieveLinks (vertex2, vertex1, TPM, contractionRecord);
cutSet = mergeArcSets (arcSet1, arcSet2);

return cutSet;
}
} // procedure getCutSet
/* Recursive procedure to retrieve vertices that were contracted into vertex2 via vertex1 and store their cumulative arcs in the cut set that is returned */

procedure retrieveLinks (integer vertex1, vertex2, float TPM [*,*], integer contractionRecord [*]) returning arcSet

{  integer state;
    set arcSet, arcSetA, arcSetB;

    /* Add all non-duplicate arcs from vertex2 to vertex1 into cut set */
    if (TPM [vertex1, vertex2] > 0)
    {  if (not duplicateArc (vertex2 → vertex1))
        place (vertex2 → vertex1) into arcSet;
    }

    /* Add all non-duplicate arcs from vertex1 to vertex2 into cut set */
    if (TPM [vertex2, vertex1] > 0)
    {  if (not duplicateArc (vertex1 → vertex2))
        place (vertex1 → vertex2) into arcSet;
    }

    /* Recurse on vertex2 and vertex1 to follow chain of vertex contractions */
    for (state = 1; state <= matrixOrder)
    {  if (contractionRecord [state] = vertex1)
        arcSetA = retrieveLinks (state, vertex2, TPM, contractionRecord);
        if (contractionRecord [state] == vertex2)
        arcSetB = retrieveLinks (vertex1, state, TPM, contractionRecord);
    }

    arcSet = mergeArcSets (arcSetA, arcSetB);
    return arcSet;
} // procedure retrieveLinks
Appendix B. Four Transition Test Probability Matrices

For all matrices, row and column numbers are provided in margins. Non-zero elements represent state transitions and are shaded. In a state transition, the from state is read along the row, and the to state is read along the column.

B.1 Matrix 1

A matrix of order 50 generated using [Hu1994] from [Je1986]. State 50 has been made the absorbing state. Note that four states, 2, 3, 25, and 26 each have at least 10 transitions to other states, state 24 has 7 transitions to other states, while all other states have a single transition to either state 1 or the absorbing state, state 50. Thus, this TPM exhibits a high degree of interconnectivity. Only 530, 432 minimal s-t cut sets were found.
B.2 Matrix 2

A matrix of order 50 generated using [Hu1994] from [Bo1988]. State 50 has been made the absorbing state. In this case, no state has more than 3 transitions to any other state. There were 28,230,288 minimal s-t cut sets found.
### B.3 Matrix 3

This matrix of order 50 is a subset of the 136 x 136 [rw136] described in Section 5.3.1. It was obtained by taking the first 50 states of that matrix. State 50 has been made the absorbing state. In this case, no state has more than 4 transitions to any other state. There were 27,242,634 minimal s-t cut sets found.
B.4 Matrix 4

A matrix of order 50 generated using [Hu1994] from [Gr1997]. State 40 has been made the absorbing state. Note that three states, 4, 5, 36, and 37 each have at least 7 transitions to other states. Like Matrix 1, this matrix is highly connected, but 422,060,801 minimal s-t cut sets were found.